第一讲_软件框架的搭建

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

目前主流的 Android 软件架构，采用 RadioGroup+Fragment 方式，采用该架构可以实现，网易新闻客户端，微信客户端，ＱＱ客户端等等市面上百分之九十的软件；

本文档已经录制成视频，可以到尚硅谷官网下载：
下载地址：http://atguigu.com/download.shtml#kj
或者百度网盘下载：
http://pan.baidu.com/s/1jHN6avO

1_启动页面(LauncherActivity)

1.1_启动页面的布局

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ffffff">

    <ImageView
        android:id="@+id/iv_icon"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:src="@drawable/atguigu_logo" />

    <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_below="@id/iv_icon"
        android:layout_centerHorizontal="true"
        android:text="尚硅谷 Android 世界..."
        android:textSize="18sp" />

</RelativeLayout>
```

1.2_启动页面的代码

延迟两秒进入主页面

```
package com.atguigu.atguigu.activity;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import com.atguigu.atguigu.R;
/**
 * 作用：启动页面
 */
public class LauncherActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_launcher);

        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                startMainActivity();
            }
        }, 2000);
    }
    /**
     * 启动主页面
     */
    private void startMainActivity() {
```

```
        Intent intent = new Intent(this,MainActivity.class);
        startActivity(intent);
        finish();
    }
}
```

2_主页面

2.1_主页面实现的分析

2.2_主页面的布局

主页面采用的是线性布局，中间是 FrameLayout 并且设置权重 1，底部是 RadioGroup

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <!--标题栏-->
    <include layout="@layout/titlebar"/>
    <!--FrameLayout-->
    <FrameLayout
        android:id="@+id/fl_content"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" />
    <!--RadioGroup-->
    <RadioGroup
        android:id="@+id/rg_bottom_tag"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#11000000"
        android:gravity="center_vertical"
        android:orientation="horizontal"
        android:padding="3dp"
        >
```

```
        <RadioButton
            android:id="@+id/rb_common_frame"
            android:text="常用框架"

android:drawableTop="@drawable/rb_common_frame_drawable_selector"
            style="@style/botttom_tag_style"
            />

        <RadioButton
            android:id="@+id/rb_thirdparty"
            android:text="第三方"

android:drawableTop="@drawable/rb_thirdparty_drawable_selector"
            style="@style/botttom_tag_style"
            />

        <RadioButton
            android:id="@+id/rb_custom"
            android:text="自定义控件"
            android:drawableTop="@drawable/rb_custom_drawable_selector"
            style="@style/botttom_tag_style"
            />

        <RadioButton
            android:id="@+id/rb_other"
            android:text="其他"
            android:drawableTop="@drawable/rb_other_drawable_selector"
            style="@style/botttom_tag_style"
            />

    </RadioGroup>

</LinearLayout>
```

Titlebar 标题栏

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:background="@android:color/holo_blue_light"
    android:gravity="center"
```

```
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:clickable="true"
        android:text="尚硅谷知识库"
        android:textColor="#ffffff"
        android:textSize="20sp" />

</LinearLayout>
```

在 values/styles/botttom_tag_style 的代码

```
<style name="botttom_tag_style">
    <!-- Customize your theme here. -->
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:layout_gravity">center_vertical</item>
    <item name="android:button">@android:color/transparent</item>
    <item name="android:drawablePadding">3dp</item>
    <item name="android:gravity">center</item>
    <item name="android:textColor">@drawable/rb_style_textcolor_selector</item>
    <item name="android:textSize">10sp</item>
    <item name="android:layout_weight">1</item>
</style>
```

rb_style_textcolor_selector 代码

颜色值一般是由设计师提供，不需要记住

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="false" android:color="#363636"/>
```

```
    <item android:state_checked="true" android:color="#3097FD"/>
</selector>
```

2.3_实例化布局控件

```
/**
 * 初始化控件
 */
private void initView() {
    setContentView(R.layout.activity_main);
    rg_bottom_tag = (RadioGroup) findViewById(R.id.rg_bottom_tag);
}
```

3_创建各个子页面

3.1_创建 BaseFragment

```
package com.atguigu.atguigu.base;

import android.content.Context;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

/**
 * 作用：基类Fragment
 * CommonFrameFragment,ThirdParty,CustomFragment,OtherFragment 等类继承它
 */
public abstract class BaseFragment extends Fragment {
    /**
     * 上下文
     */
    protected Context mContext;
    /**
     * 该Fragment 是否被初始化过
     */
```

```
    private boolean isInit = false;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mContext = getActivity();
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        return initView();
    }

    /**
     * 由子类实现该方法，创建自己的视图
     * @return
     */
    protected abstract View initView() ;

    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        initData();
    }
    /**
     * 子类，需要初始化数据，联网请求数据并且绑定数据，等重写该方法
     */
    protected void initData() {
    }
}
```

3.2_定义各个子页面

CommonFrameFragment 常用框架 Fragment

```
/**
 * 作用：常用框架
 */
public class CommonFrameFragment extends BaseFragment {
```

```
    private static final String TAG = CustomFragment.class.getSimpleName();
    private TextView textView;
    @Override
    protected View initView() {
        Log.e(TAG,"常用框架页面初始化了...");
        textView = new TextView(mContext);
        textView.setGravity(Gravity.CENTER);
        textView.setTextSize(20);
        textView.setTextColor(Color.BLACK);
        return textView;
    }

    @Override
    protected void initData() {
        super.initData();
        Log.e(TAG, "常用框架数据初始化了...");
        textView.setText("我是常用框架页面");
    }
}
```

CustomFragment 自定义 Fragment 类

```
/**
 * 作用：自定义
 */
public class CustomFragment extends BaseFragment {

    private static final String TAG = CustomFragment.class.getSimpleName();
    private TextView textView;
    @Override
    protected View initView() {
        Log.e(TAG,"自定义页面初始化了...");
        textView = new TextView(mContext);
        textView.setGravity(Gravity.CENTER);
        textView.setTextSize(20);
        textView.setTextColor(Color.BLACK);
        return textView;
    }

    @Override
```

```
    protected void initData() {
        super.initData();
        Log.e(TAG, "自定义数据初始化了...");
        textView.setText("我是自定义页面");
    }
}
```

ThirdPartyFragment 第三方 Fragment

```
/**
 * 作用：第三方
 */
public class ThirdPartyFragment extends BaseFragment {

    private static final String TAG =
ThirdPartyFragment.class.getSimpleName();
    private TextView textView;
    @Override
    protected View initView() {
        Log.e(TAG,"第三方页面初始化了...");
        textView = new TextView(mContext);
        textView.setGravity(Gravity.CENTER);
        textView.setTextSize(20);
        textView.setTextColor(Color.BLACK);
        return textView;
    }

    @Override
    protected void initData() {
        super.initData();
        Log.e(TAG, "第三方数据初始化了...");
        textView.setText("我是第三方页面");
    }
}
```

OtherFragment 其他 Fragment

```
/**
 * 作用：其他
 */
public class OtherFragment extends BaseFragment {

    private static final String TAG = OtherFragment.class.getSimpleName();
    private TextView textView;
    @Override
    protected View initView() {
        Log.e(TAG,"其他页面初始化了...");
        textView = new TextView(mContext);
        textView.setGravity(Gravity.CENTER);
        textView.setTextSize(20);
        textView.setTextColor(Color.BLACK);
        return textView;
    }

    @Override
    protected void initData() {
        super.initData();
        Log.e(TAG, "其他数据初始化了...");
        textView.setText("我是其他页面");
    }
}
```

3.3_初始化 Fragment

```
/**
 * 初始化 Fragment
 */
private void initFragment() {
    mBaseFragments = new ArrayList<>();
    mBaseFragments.add(new CommonFrameFragment());//常用框架
    mBaseFragments.add(new ThirdPartyFragment());//第三方
    mBaseFragments.add(new CustomFragment());//自定义
    mBaseFragments.add(new OtherFragment());//其他

}
```

3.4_设置 RadioGroup 的监听

```
private void setListener() {
    rg_bottom_tag.setOnCheckedChangeListener(new
MyOnCheckedChangeListener());
    //设置默认主页面
    rg_bottom_tag.check(R.id.rb_common_frame);
}

class MyOnCheckedChangeListener implements RadioGroup.OnCheckedChangeListener
{

    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId) {
            switch (checkedId){
            case R.id.rb_common_frame:
                position = 0;
                break;
            case R.id.rb_thirdparty:
                position = 1;
                break;
            case R.id.rb_custom:
                position = 2;
                break;
            case R.id.rb_other:
                position = 3;
                break;
            default:
                position = 0;
                break;
        }

        Fragment  fragment = getFragment();
        switchFragment(mContent, fragment);

    }
}
```

3.5_得到 Fragment

```
/**
 * 得到Fragment
 * @return
 */
private BaseFragment getFragment() {
    if(mBaseFragments != null){
        BaseFragment baseFragment = mBaseFragments.get(position);
        return baseFragment;
    }
    return null;
}
```

3.6_切换 Fragment

```
private void switchFrament(BaseFragment fragment) {
    //1.得到FragmentManger
    FragmentManager fm = getSupportFragmentManager();
    //2.开启事务
    FragmentTransaction transaction = fm.beginTransaction();
    //3.替换
    transaction.replace(R.id.fl_content,fragment);
    //4.提交事务
    transaction.commit();
}
```

第一讲_软件框架的搭建

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

目前主流的 Android 软件架构，采用 RadioGroup+Fragment 方式，采用该架构可以实现，网易新闻客户端，微信客户端，ＱＱ客户端等等市面上百分之九十的软件；

本文档已经录制成视频，可以到尚硅谷官网下载：
下载地址：http://atguigu.com/download.shtml#kj
或者百度网盘下载：
http://pan.baidu.com/s/1jHN6avO

1_软件框架性能优化

1_解决切换 Fragment 切换导致重新创建 Fragment 问题

在项目中切换 Fragment，一直都是用 replace()方法来替换 Fragment。但是这样做有一个问题，每次切换的时候 Fragment 都会重新实列化，重新加载一次数据，这样做会非常消耗性能用用户的流量。

官方文档解释说：replace()这个方法只是在上一个 Fragment 不再需要时采用的简便方法。

正确的切换方式是 add()，切换时 hide()，add()另一个 Fragment；再次切换时，只需 hide()当前，show()另一个。
这样就能做到多个 Fragment 切换不重新实例化：

切换方法：

```
/**
 * 切换不同的Fragment
 * @param from
 * @param to
 */
public void switchFragment(Fragment from, Fragment to) {
```

```
    if (mContent != to) {
        mContent = to;
        FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();
        if (!to.isAdded()) {
          // 先判断是否被 add 过
            if(from != null){
                transaction.hide(from);
            }
            if(to != null){
                transaction.add(R.id.fl_content, to).commit();
            }

        } else {
            if(from != null){
                transaction.hide(from);
            }
            if(to != null){
                transaction.show(to).commit();
            }
        }
    }
}
```

2_解决横竖屏切换导致的 Fragment 内容重叠问题

在功能清单文件配置

```
<activity android:name=".activity.MainActivity"
    android:configChanges="orientation|keyboardHidden|screenSize"
    >
</activity>
```

2_常用框架页面功能实现

1_常用框架页面布局 fragment_common_frame.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/listview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</RelativeLayout>
```

2_常用框架页面代码

```
/**
 * 作用：常用框架
 */
public class CommonFrameFragment extends BaseFragment {

    private static final String TAG =
CommonFrameFragment.class.getSimpleName();

    private ListView mListview;
    private String [] datas;
    private CommonFrameFragmentAdapter adapter;
    @Override
    protected View initView() {
        Log.e(TAG,"常用框架页面初始化了...");
        View view = View.inflate(mContext,
R.layout.fragment_common_frame,null);
        mListview = (ListView) view.findViewById(R.id.listview);
        mListview.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
```

```
            public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
                String data = datas[position];
                Toast.makeText(mContext, "data=="+data,
Toast.LENGTH_SHORT).show();
            }
        });
        return view;
    }

    @Override
    protected void initData() {
        super.initData();
        Log.e(TAG, "常用框架数据初始化了...");

        //准备数据
        datas = new String[]{ "OKHttp",
"xUtils3","Retrofit2","Fresco","Glide","greenDao","RxJava","volley","Gson",
"FastJson","picasso","evenBus","jcvideoplayer","pulltorefresh","Expandablel
istview","UniversalVideoView","....."};

        //设置适配器
        adapter = new CommonFrameFragmentAdapter(mContext,datas);
        mListview.setAdapter(adapter);
    }
}
```

3_常用框架页面适配器

重新创建一个新包，com.atguigu.android.adapter

```
/**
 * 作用：常用框架的适配器
 */
public class CommonFrameFragmentAdapter extends BaseAdapter {
    private final Context mContext;
    private final String[] mDatas;

    public CommonFrameFragmentAdapter(Context context,String[] datas){
        this.mContext = context;
```

```
        this.mDatas = datas;
    }
    @Override
    public int getCount() {
        return mDatas.length;
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        TextView textView = new TextView(mContext);
        textView.setPadding(10, 10, 0, 10);
        textView.setText(mDatas[position]);
        textView.setTextColor(Color.BLACK);
        textView.setTextSize(20);
        return textView;
    }
}
```

第 2 讲_OKHttp 的使用

谷粉第 46 群：252915839

本文档已经录制成视频，请到尚硅谷官网上下载视频，下载地址：
http://atguigu.com/download.shtml#kj

百度网盘下载地址：
http://pan.baidu.com/s/1hsQ8koW

1_OKHttp 简介

1.1_简介

OKHttp 是一款高效的 HTTP 客户端，支持连接同一地址的链接共享同一个 socket，通过连接池来减小响应延迟，还有透明的 GZIP 压缩，请求缓存等优势，其核心主要有路由、连接协议、拦截器、代理、安全性认证、连接池以及网络适配，拦截器主要是指添加，移除或者转换请求或者回应的头部信息

这个库也是 square 开源的一个网络请求库(okhttp 内部依赖 okio)。现在已被 Google 使用在 Android 源码上了，可见其强大。

关于网络请求库，现在应该还有很多人在使用 android-async-http。他内部使用的是 HttpClient，但是 Google 在 6.0 版本里面删除了 HttpClient 相关 API，可见这个库现在有点过时了。

1.2_下载地址

http://square.github.io/okhttp/

1.3_OKHttp 主要功能

1、联网请求文本数据

2、大文件下载
3、大文件上传
4、请求图片

2_原生 OKHttp 的 Get 和 Post 请求小案例

参照网址：
http://square.github.io/okhttp/

1.1_小案例的布局和代码

请求网络测试地址：
http://api.m.mtime.cn/PageSubArea/TrailerList.api

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:orientation="vertical"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.atguigu.okhttpsample.MainActivity">

    <Button
        android:id="@+id/btn_get"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="get 请求" />
    <Button
        android:id="@+id/btn_post"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="post 请求" />

    <TextView
        android:id="@+id/tv_result"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="显示请求数据" />

</LinearLayout>
```

1.2_点击事件

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    btn_get = (Button) findViewById(R.id.btn_get);
    btn_post = (Button) findViewById(R.id.btn_post);
    tv_result = (TextView) findViewById(R.id.tv_result);
    //设置点击事件
    btn_get.setOnClickListener(this);
    btn_post.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()){
        case R.id.btn_get:
            getDataFromByGet();
            break;
        case R.id.btn_post:
            getDataFromByPost();
            break;
    }
}
```

1.3_OKHttp 的 get 请求

```
private void getDataFromByGet() {

    new Thread(){
        @Override
```

```
        public void run() {
            super.run();
            try {
                String resutl =
get("http://api.m.mtime.cn/PageSubArea/TrailerList.api");
                Message msg = Message.obtain();
                msg.what = GET;
                msg.obj = resutl;
                handler.sendMessage(msg);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }.start();
}


/**
 * get 请求
 * @param url
 * @return
 * @throws IOException
 */
private  String get(String url) throws IOException {
    Request request = new Request.Builder()
            .url(url)
            .build();
    Response response = client.newCall(request).execute();
    return response.body().string();
}
```

1.4_OKHttp 的 post 请求

```
private void getDataFromByPost() {
    new Thread(){
        @Override
        public void run() {
            super.run();
```

```
            String resutl = null;
            try {
                resutl =
post("http://api.m.mtime.cn/PageSubArea/TrailerList.api","");
                Message msg = Message.obtain();
                msg.what = POST;
                msg.obj = resutl;
                handler.sendMessage(msg);
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }.start();
}
/**
 * post 请求
 * @param url
 * @param json
 * @return
 * @throws IOException
 */
private  String post(String url, String json) throws IOException {
    RequestBody body = RequestBody.create(JSON, json);
    Request request = new Request.Builder()
            .url(url)
            .post(body)
            .build();
    Response response = client.newCall(request).execute();
    return response.body().string();
}
```

1.5_在 Handler 中显示数据

```
/**
 * get 请求
 */
private static final int GET = 1;
/**
 * post 请求
 */
```

```
private static final int POST = 2;
private Button btn_get,btn_post;
private TextView tv_result;
private OkHttpClient client = new OkHttpClient();
private Handler handler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        switch (msg.what){
            case GET://get 请求
                tv_result.setText(msg.obj.toString());
                break;
            case POST://post 请求
                tv_result.setText(msg.obj.toString());
                break;
        }
    }
};
```

第 2 讲_OKHttp 的使用

谷粉第 46 群：252915839

本文档已经录制成视频，请到尚硅谷官网上下载视频，下载地址：
http://atguigu.com/download.shtml#kj

百度网盘下载地址：
http://pan.baidu.com/s/1hsQ8koW

3_第三方封装好的 OKHttp 库-okhttp-utils

1_okhttp-utils 简介

1_下载并且运行案例

https://github.com/hongyangAndroid/okhttp-utils

2_解决报错

在 sample-okhttp 中的 build.gradle 文件中
如下配置

```
allprojects {

    repositories {

        maven { url "https://jitpack.io" }
```

```
    }
}
```

2_把 okhttp-utils 集成到案例中

1_关联库

```
compile project(':okhttputils')
```

2_解决报错

因为 okhttputils 库里面本身就有 okhttp 库和 okio 库，需要注释掉

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.3.0'
//    compile files('libs/okhttp-3.4.1.jar')
//    compile files('libs/okio-1.9.0.jar')
    compile project(':okhttputils')
}
```

3_使用 okhttp-utils 请求文本

get 请求

```
public void getDataByOkhttputils()
{
    String url = "http://www.zhiyun-tech.com/App/Rider-M/changelog-zh.txt";
    url="http://api.m.mtime.cn/PageSubArea/TrailerList.api";
    OkHttpUtils
            .get()
            .url(url)
            .id(100)
            .build()
            .execute(new MyStringCallback());
}
```

Post 请求

```
public void getDataByOkhttputils()
{
    String url =
"http://www.zhiyun-tech.com/App/Rider-M/changelog-zh.txt";
    url="http://api.m.mtime.cn/PageSubArea/TrailerList.api";
    OkHttpUtils
            .post()
            .url(url)
            .id(100)
            .build()
            .execute(new MyStringCallback());
}
```

4_使用 okhttp-utils 文件下载

1_布局文件

```
<Button
```

```
    android:id="@+id/btn_downloadFile"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="下载文件" />



<ProgressBar
    style="?android:progressBarStyleHorizontal"
    android:layout_gravity="bottom"
    android:id="@+id/mProgressBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

2_下载文件代码

```
/**
 * 下载文件
 */
public void downloadFile()
{
    String url =
"http://vfx.mtime.cn/Video/2016/07/24/mp4/160724154733643806.mp4";
    OkHttpUtils//
            .get()//
            .url(url)//
            .build()//
            .execute(new
FileCallBack(Environment.getExternalStorageDirectory().getAbsolutePath(),
"160724154733643806.mp4")//
            {

                @Override
                public void onBefore(Request request, int id) {
                }

                @Override
                public void inProgress(float progress, long total, int id) {
                    mProgressBar.setProgress((int) (100 * progress));
```

```
                Log.e(TAG, "inProgress :" + (int) (100 * progress));
            }

            @Override
            public void onError(Call call, Exception e, int id) {
                Log.e(TAG, "onError :" + e.getMessage());
            }

            @Override
            public void onResponse(File file, int id) {
                Log.e(TAG, "onResponse :" + file.getAbsolutePath());
            }
        });
}
```

3_记得加权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

5_文件上传

1_支持文件上传服务器的搭建

2_文件上传

```
/**
 * 文件上传
 */
public void multiFileUpload()
{
    String mBaseUrl =
```

```
"http://192.168.10.165:8080/FileUpload/FileUploadServlet";
    File file = new File(Environment.getExternalStorageDirectory(), "1.jpg");
    File file2 = new File(Environment.getExternalStorageDirectory(), "1.txt");
    if (!file.exists())
    {
        Toast.makeText(MainActivity.this, "文件不存在，请修改文件路径",
Toast.LENGTH_SHORT).show();
        return;
    }
    Map<String, String> params = new HashMap<>();
    params.put("username", "杨光福");
    params.put("password", "123");

    String url = mBaseUrl;
    OkHttpUtils.post()//
            .addFile("mFile", "messenger_01.png", file)//
            .addFile("mFile", "test1.txt", file2)//
            .url(url)
            .params(params)//
            .build()//
            .execute(new MyStringCallback());
}
```

6.使用 okhttp-utils 请求图片

1_请求单张图片

```
public void getImage(View view) {
    mTv.setText("");
    String url = "http://images.csdn.net/20150817/1.jpg";
    OkHttpUtils
            .get()//
            .url(url)//
            .tag(this)//
            .build()//
            .connTimeOut(20000)
            .readTimeOut(20000)
            .writeTimeOut(20000)
            .execute(new BitmapCallback() {
                @Override
                public void onError(Call call, Exception e, int id) {
```

```
                mTv.setText("onError:" + e.getMessage());
            }

            @Override
            public void onResponse(Bitmap bitmap, int id) {
                Log.e("TAG", "onResponse：complete");
                mImageView.setImageBitmap(bitmap);
            }
        });
}
```

2_在列表中 OKHttpActivity 请求图片-布局文件

布局文件

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/listview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

</RelativeLayout>
```

3_请求数据

```
/**
 * 请求数据，如果有缓存先加载缓存数据
 */
private void getData() {
    //获取缓存数据
    String saveJson = CacheUtils.getString(OKHttpActivity.this, url);
```

```
    if (!TextUtils.isEmpty(saveJson)) {
        processData(saveJson);
    }
    //请求网络
    OkHttpUtils
            .get()
            .url(url)
            .id(108)
            .build()
            .execute(new MyStringCallback());
}
```

4.缓存数据

```
public class MyStringCallback extends StringCallback {
    @Override
    public void onBefore(Request request, int id) {
        setTitle("loading...");
    }

    @Override
    public void onAfter(int id) {
        setTitle("Sample-okHttp");
    }

    @Override
    public void onError(Call call, Exception e, int id) {
        e.printStackTrace();
        Log.e("TAG", "onError:" + e.getMessage());
    }

    @Override
    public void onResponse(String response, int id) {
        Log.e("TAG", "onResponse：complete");

        switch (id) {
            case 100:
                Toast.makeText(OKHttpActivity.this, "http",
Toast.LENGTH_SHORT).show();
                break;
```

```
            case 101:
                Toast.makeText(OKHttpActivity.this, "https",
Toast.LENGTH_SHORT).show();
                break;
            case 108://列表中显示数据
                if (response != null) {
                    //缓存数据
                    CacheUtils.putString(OKHttpActivity.this, url, response);
                    //解析和显示数据
                    processData(response);
                }

                break;
        }
    }

    @Override
    public void inProgress(float progress, long total, int id) {
        Log.e("OkHttpActivity", "inProgress:" + progress);
    }
}
```

5_使用 Android 自带 API 解析 json 数据

实例 Bean 类

```
package com.atguigu.okhttpsample;

import java.util.List;

/**
 */
public class DataBean {

    /**
     * id : 61651
     * movieName : 《奇异博士》中文版预告片
     * coverImg : http://img31.mtime.cn/mg/2016/07/25/143053.69987441.jpg
     * movieId : 108737
     * url :
http://vfx.mtime.cn/Video/2016/07/24/mp4/160724154733643806_480.mp4
```

```
     * hightUrl :
http://vfx.mtime.cn/Video/2016/07/24/mp4/160724154733643806.mp4
     * videoTitle : 奇异博士 中文版预告片
     * videoLength : 151
     * rating : -1
     * type : ["动作","冒险","奇幻","科幻"]
     * summary : 卷福"施展魔法 展开宏大幻境
     */

    private List<TrailersEntity> trailers;

    public void setTrailers(List<TrailersEntity> trailers) {
        this.trailers = trailers;
    }

    public List<TrailersEntity> getTrailers() {
        return trailers;
    }

    public static class TrailersEntity {
        private int id;
        private String movieName;
        private String coverImg;
        private int movieId;
        private String url;
        private String hightUrl;
        private String videoTitle;
        private int videoLength;
        private int rating;
        private String summary;
        private List<String> type;

        public void setId(int id) {
            this.id = id;
        }

        public void setMovieName(String movieName) {
            this.movieName = movieName;
        }

        public void setCoverImg(String coverImg) {
            this.coverImg = coverImg;
        }
```

```
    public void setMovieId(int movieId) {
        this.movieId = movieId;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public void setHightUrl(String hightUrl) {
        this.hightUrl = hightUrl;
    }

    public void setVideoTitle(String videoTitle) {
        this.videoTitle = videoTitle;
    }

    public void setVideoLength(int videoLength) {
        this.videoLength = videoLength;
    }

    public void setRating(int rating) {
        this.rating = rating;
    }

    public void setSummary(String summary) {
        this.summary = summary;
    }

    public void setType(List<String> type) {
        this.type = type;
    }

    public int getId() {
        return id;
    }

    public String getMovieName() {
        return movieName;
    }

    public String getCoverImg() {
        return coverImg;
    }
```

```
        public int getMovieId() {
            return movieId;
        }

        public String getUrl() {
            return url;
        }

        public String getHightUrl() {
            return hightUrl;
        }

        public String getVideoTitle() {
            return videoTitle;
        }

        public int getVideoLength() {
            return videoLength;
        }

        public int getRating() {
            return rating;
        }

        public String getSummary() {
            return summary;
        }

        public List<String> getType() {
            return type;
        }
    }
}
```

解析 json 数据

```
/**
 * 解析json数据
 *
 * @param response
 * @return
```

```
 */
private DataBean parsedJson(String response) {
    DataBean dataBean = new DataBean();
    try {
        JSONObject jsonObject = new JSONObject(response);
        JSONArray jsonArray = jsonObject.optJSONArray("trailers");
        if (jsonArray != null && jsonArray.length() > 0) {
            List<DataBean.TrailersEntity> trailers = new ArrayList<>();
            dataBean.setTrailers(trailers);
            for (int i = 0; i < jsonArray.length(); i++) {

                JSONObject jsonObjectItem = (JSONObject) jsonArray.get(i);

                if (jsonObjectItem != null) {

                    DataBean.TrailersEntity mediaItem = new
DataBean.TrailersEntity();

                    String movieName =
jsonObjectItem.optString("movieName");//name
                    mediaItem.setMovieName(movieName);

                    String videoTitle =
jsonObjectItem.optString("videoTitle");//desc
                    mediaItem.setVideoTitle(videoTitle);

                    String imageUrl =
jsonObjectItem.optString("coverImg");//imageUrl
                    mediaItem.setCoverImg(imageUrl);

                    String hightUrl =
jsonObjectItem.optString("hightUrl");//data
                    mediaItem.setHightUrl(hightUrl);

                    //把数据添加到集合
                    trailers.add(mediaItem);
                }
            }
        }

    } catch (JSONException e) {
        e.printStackTrace();
    }
    return dataBean;
```

```
}
```

6_设置适配器

设置适配器

```
/**
 * json 数据解析和设置适配器
 *
 * @param saveJson
 */
private void processData(String saveJson) {
    DataBean dataBean = parsedJson(saveJson);
    List<DataBean.TrailersEntity> datas = dataBean.getTrailers();
    MyAdapter myAdapter = new MyAdapter(OKHttpActivity.this, datas);
    listview.setAdapter(myAdapter);
}
```

适配器代码

```
/**
 */
public class MyAdapter extends BaseAdapter {

    private final Context mContext;
    private final List<DataBean.TrailersEntity> mDatas;

    public MyAdapter(Context context, List<DataBean.TrailersEntity> datas){
        this.mContext = context;
        this.mDatas = datas;
    }
    @Override
    public int getCount() {
        return mDatas.size();
    }

    @Override
    public Object getItem(int position) {
        return null;
    }
```

```
    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        final ViewHoder viewHoder;
        if(convertView ==null){
            convertView = View.inflate(mContext, R.layout.item,null);
            viewHoder = new ViewHoder();
            viewHoder.iv_icon = (ImageView)
convertView.findViewById(R.id.iv_icon);
            viewHoder.tv_name = (TextView)
convertView.findViewById(R.id.tv_name);
            viewHoder.tv_desc = (TextView)
convertView.findViewById(R.id.tv_desc);

            convertView.setTag(viewHoder);
        }else{
            viewHoder = (ViewHoder) convertView.getTag();
        }

        //根据position 得到列表中对应位置的数据
        DataBean.TrailersEntity trailersEntity = mDatas.get(position);
        viewHoder.tv_name.setText(trailersEntity.getMovieName());
        viewHoder.tv_desc.setText(trailersEntity.getVideoTitle());
        //1.使用OKHttp 请求图片
        OkHttpUtils
                .get()//
                .url(trailersEntity.getCoverImg())//
                .tag(this)//
                .build()//
                .connTimeOut(20000)
                .readTimeOut(20000)
                .writeTimeOut(20000)
                .execute(new BitmapCallback()
                {
                    @Override
                    public void onError(Call call, Exception e, int id)
                    {
                        Log.e("TAG", "onError:" + e.getMessage());
                    }
```

```
                @Override
                public void onResponse(Bitmap bitmap, int id)
                {
                    Log.e("TAG", "onResponse：complete");
                    viewHoder.iv_icon.setImageBitmap(bitmap);
                }
            });

        return convertView;
    }


    static class ViewHoder{
        ImageView iv_icon;
        TextView tv_name;
        TextView tv_desc;
    }
}
```

适配器布局

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="100dp"
    android:gravity="center_vertical">

    <RelativeLayout
        android:id="@+id/rl_image"
        android:layout_width="120dp"
        android:layout_height="80dp"
        android:layout_centerVertical="true">

        <ImageView
            android:id="@+id/iv_icon"
            android:layout_width="120dp"
            android:layout_height="80dp"
            android:layout_marginLeft="8dp"
            android:scaleType="fitXY"
            android:src="@drawable/video_default" />

        <ImageView
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_alignParentRight="true"
            android:layout_marginBottom="8dp"
            android:layout_marginRight="8dp"
            android:src="@drawable/center_collect_play" />
    </RelativeLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_marginLeft="8dp"
        android:layout_toRightOf="@id/rl_image"
        android:orientation="vertical">

        <TextView
            android:id="@+id/tv_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:singleLine="true"
            android:text="视频的名称"
            android:textColor="#000000"
            android:textSize="18sp" />

        <TextView
            android:id="@+id/tv_desc"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:singleLine="true"
            android:text="视频的描述"
            android:textColor="#55000000"
            android:textSize="18sp" />

    </LinearLayout>

</RelativeLayout>
```

4_其他库 OkHttpUtils

封装了 okhttp 的网络框架，支持大文件上传下载，上传进度回调，下载进度回调，表单上传（多文件和多参数一起上传），链式调用，可以自定义返 回对象，支持 Https 和自签名证书，支持 cookie 自动管理，支持四种缓存模式缓存网络数据，支持 301、302 重定向，扩展了统一的上传管理和下载 管理功能

1_下载地址

https://github.com/jeasonlzy0216/OkHttpUtils

第 3 讲_JSON 解析之手动解析

谷粉第 47 群：285047793

1_JSON 简介

1.1_简介

JSON 的全称是 JavaScript Object Notation，是一种轻量级的数据交换格式。

1.2_特点

（1）JSON 比 XML 数据传输的有效性要高出很多
（2）JSON 完全独立于编程语言。
（3）本质就是具有特定格式的字符串

2_JSON 数据格式

2.1_整体结构

String json1 = "{"id":12,"name":"Tom"}"
String json2 = "[{"id":12,"name":"Tom"},{"id":12,"name":"Tom"}]"

2.2_Json 数组 ： [ ]

（1) Json 数组的结构: [value1, value2, value3]

（2）例子：
[1, “ab”,[], {“n”:123, “b”:”abc”}] 正确
[1, “a”:3] 错误

2.2_Json 对象: {}

（1）Json 对象的结构: {key1:value1, key2:value2, key3:value3}
（2）key 的数据类型: 字符串
（3）value 的数据类型: 数值、字符串、null、json 数组 []、json 对象 {}
（4）例子:
{"name":"TOM", "age":12} 正确
{"aa":"a", 3} 错误

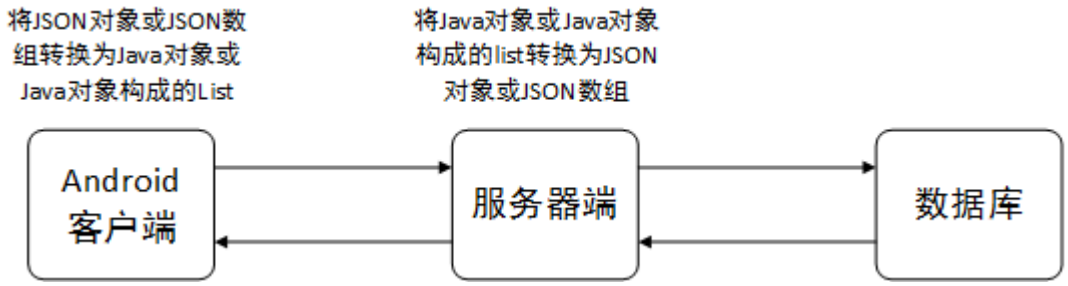
3_JSON 解析方向

3.1_将 java 对象(包含集合)转换为 json 格式字符串

在服务器端应用。

3.2_将 json 格式字符串转换为 java 对象（包含集合）

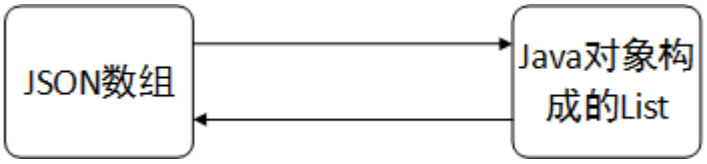
在客户端应用。



3.3_Json 和 Java 之间转换关系

（1）JSON 中的对象对应着 Java 中的对象

（2）Json 中的数组，对应着 Java 中的集合



4_JSON 解析技术

4.1_Android 原生技术

1）特点：编程相对麻烦

4.1.1_将 json 格式的字符串{}转换为 Java 对象

1）API：JsonObject

JSONObject(String json)：将 json 字符串解析为 json 对象

Xxx getXxx(String name)：根据 name，在 json 对象中得到对应的 Value

2）测试数据

```
{
    "id":2, "name":"大虾",
    "price":12.3,
    "imagePath":"http://192.168.10.165:8080/L05_Server/images/f1.jpg"
}
```

3）例子

```
// 将 json 格式的字符串{}转换为 Java 对象
    private void jsonToJavaObjectByNative() {

        // 获取或创建 JSON 数据
        String json = "{\n" +
                "\t\"id\":2, \"name\":\"大虾\", \n" +
                "\t\"price\":12.3, \n" +

"\t\"imagePath\":\"http://192.168.10.165:8080/L05_Server/images/f1.jpg\
"\n" +"}\n";

        ShopInfo shopInfo = null;
        // 解析 json
        try {
            JSONObject jsonObject = new JSONObject(json);
//            int id = jsonObject.getInt("id");
            int id1 = jsonObject.optInt("id");

            String name = jsonObject.optString("name");

            double price = jsonObject.optDouble("price");
```

```
            String imagePath = jsonObject.optString("imagePath");

            // 封装 Java 对象
            shopInfo = new ShopInfo(id1, name, price, imagePath);

        } catch (JSONException e) {
            e.printStackTrace();
        }

        // 显示 JSON 数据
        tv_native_orignal.setText(json);
        tv_native_last.setText(shopInfo.toString());
    }
```

4.1.2_将 json 格式的字符串[]转换为 Java 对象的 List

1）API:JSONArray

JSONArray(String json)：将 json 字符串解析为 json 数组

int length()：得到 json 数组中元素的个数

Xxx getXxx(int index)：根据下标得到 json 数组中对应的元素数据

2）测试数据

```
[
    {
     "id":1, "name":"大虾1",
     "price":12.3,
     "imagePath":"http://192.168.10.165:8080/f1.jpg"
    },
    {
     "id":2, "name":"大虾2",
     "price":12.5,
     "imagePath":"http://192.168.10.165:8080/f2.jpg"
    }
]
```

3）例子

```
// 将 json 格式的字符串[]转换为 Java 对象的 List
private void jsonToJavaListByNative() {

    // 获取或创建 JSON 数据
```

```
    String json = "[\n" +
            "    {\n" +
            "        \"id\": 1,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f1.jpg\",\n" +
            "        \"name\": \"大虾 1\",\n" +
            "        \"price\": 12.3\n" +
            "    },\n" +
            "    {\n" +
            "        \"id\": 2,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f2.jpg\",\n" +
            "        \"name\": \"大虾 2\",\n" +
            "        \"price\": 12.5\n" +
            "    }\n" +
            "]";

    List<ShopInfo> shops = new ArrayList<>();

    // 解析 json
    try {
        JSONArray jsonArray = new JSONArray(json);

        for (int i = 0; i < jsonArray.length(); i++) {
            JSONObject jsonObject = jsonArray.getJSONObject(i);

            if (jsonObject != null) {
                int id = jsonObject.optInt("id");

                String name = jsonObject.optString("name");

                double price = jsonObject.optDouble("price");

                String imagePath = jsonObject.optString("imagePath");

                // 封装 Java 对象
                ShopInfo shopInfo = new ShopInfo(id, name, price, imagePath);

                shops.add(shopInfo);
            }
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
```

```
    // 显示 JSON 数据
    tv_native_orignal.setText(json);
    tv_native_last.setText(shops.toString());
}
```

4.1.3_复杂 json 数据解析

1）测试数据

```
{
    "data": {
        "count": 5,
        "items": [
            {
                "id": 45,
                "title": "坚果"
            },
            {
                "id": 132,
                "title": "炒货"
            },
            {
                "id": 166,
                "title": "蜜饯"
            },
            {
                "id": 195,
                "title": "果脯"
            },
            {
                "id": 196,
                "title": "礼盒"
            }
        ]
    },
    "rs_code": "1000",
    "rs_msg": "success"
}
```

2）例子

```
// 复杂 json 数据解析
private void jsonToJavaOfComplex() {
```

```
// 获取或创建 JSON 数据
String json = "{\n" +
        "    \"data\": {\n" +
        "        \"count\": 5,\n" +
        "        \"items\": [\n" +
        "            {\n" +
        "                \"id\": 45,\n" +
        "                \"title\": \"坚果\"\n" +
        "            },\n" +
        "            {\n" +
        "                \"id\": 132,\n" +
        "                \"title\": \"炒货\"\n" +
        "            },\n" +
        "            {\n" +
        "                \"id\": 166,\n" +
        "                \"title\": \"蜜饯\"\n" +
        "            },\n" +
        "            {\n" +
        "                \"id\": 195,\n" +
        "                \"title\": \"果脯\"\n" +
        "            },\n" +
        "            {\n" +
        "                \"id\": 196,\n" +
        "                \"title\": \"礼盒\"\n" +
        "            }\n" +
        "        ]\n" +
        "    },\n" +
        "    \"rs_code\": \"1000\",\n" +
        "    \"rs_msg\": \"success\"\n" +
        "}";

// 封装 Java 对象
DataInfo dataInfo = new DataInfo();

// 解析 json
try {
    JSONObject jsonObject = new JSONObject(json);

    // 第一层解析
    JSONObject data = jsonObject.optJSONObject("data");
    String rs_code = jsonObject.optString("rs_code");
    String rs_msg = jsonObject.optString("rs_msg");
```

```
        // 第一层封装
        dataInfo.setRs_code(rs_code);
        dataInfo.setRs_msg(rs_msg);
        DataInfo.DataBean dataBean = new DataInfo.DataBean();
        dataInfo.setData(dataBean);

        // 第二层解析
        int count = data.optInt("count");
        JSONArray items = data.optJSONArray("items");

        // 第二层数据的封装
        dataBean.setCount(count);

        List<DataInfo.DataBean.ItemsBean> itemsBean = new ArrayList<>();
        dataBean.setItems(itemsBean);

        // 第三层解析
        for (int i = 0; i < items.length(); i++) {
            JSONObject jsonObject1 = items.optJSONObject(i);

            if (jsonObject1 != null) {
                int id = jsonObject1.optInt("id");

                String title = jsonObject1.optString("title");

                // 第三层数据的封装
                DataInfo.DataBean.ItemsBean bean = new DataInfo.DataBean.ItemsBean();
                bean.setId(id);
                bean.setTitle(title);

                itemsBean.add(bean);
            }
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }

    // 显示 JSON 数据
    tv_native_orignal.setText(json);
    tv_native_last.setText(dataInfo.toString());
}
```

4.1.4_特殊 json 数据解析

1）测试数据

```
{
    "code": 0,
    "list": {
        "0": {
            "aid": "6008965",
            "author": "哔哩哔哩番剧",
            "coins": 170,
            "copyright": "Copy",
            "create": "2016-08-25 21:34"
        },
        "1": {
            "aid": "6008938",
            "author": "哔哩哔哩番剧",
            "coins": 404,
            "copyright": "Copy",
            "create": "2016-08-25 21:33"
        }
    }
}
```

2）例子

```
// (4)特殊 json 数据解析
private void jsonToJavaOfSpecial() {

    // 1 获取或创建 JSON 数据
    String json = "{\n" +
            "    \"code\": 0,\n" +
            "    \"list\": {\n" +
            "        \"0\": {\n" +
            "            \"aid\": \"6008965\",\n" +
            "            \"author\": \"哔哩哔哩番剧\",\n" +
            "            \"coins\": 170,\n" +
            "            \"copyright\": \"Copy\",\n" +
            "            \"create\": \"2016-08-25 21:34\"\n" +
            "        },\n" +
            "        \"1\": {\n" +
            "            \"aid\": \"6008938\",\n" +
            "            \"author\": \"哔哩哔哩番剧\",\n" +
            "            \"coins\": 404,\n" +
            "            \"copyright\": \"Copy\",\n" +
```

```
            "            \"create\": \"2016-08-25 21:33\"\n" +
            "        }\n" +
            "    }\n" +
            "}";

    // 创建封装的Java 对象
    FilmInfo filmInfo = new FilmInfo();

    // 2 解析json
    try {
        JSONObject jsonObject = new JSONObject(json);

        // 第一层解析
        int code = jsonObject.optInt("code");
        JSONObject list = jsonObject.optJSONObject("list");

        // 第一层封装
        filmInfo.setCode(code);
        List<FilmInfo.FilmBean> lists = new ArrayList<>();
        filmInfo.setList(lists);

        // 第二层解析
        for (int i = 0; i < list.length(); i++) {
            JSONObject jsonObject1 = list.optJSONObject(i + "");

            if(jsonObject1 != null) {
                String aid = jsonObject1.optString("aid");

                String author = jsonObject1.optString("author");

                int coins = jsonObject1.optInt("coins");

                String copyright = jsonObject1.optString("copyright");

                String create = jsonObject1.optString("create");

                // 第二层数据封装
                FilmInfo.FilmBean filmBean = new FilmInfo.FilmBean();
                filmBean.setAid(aid);
                filmBean.setAuthor(author);
                filmBean.setCoins(coins);
                filmBean.setCopyright(copyright);
                filmBean.setCreate(create);
```

```
                lists.add(filmBean);
            }
        }

    } catch (JSONException e) {
        e.printStackTrace();
    }

    // 3 显示 JSON 数据
    tv_native_orignal.setText(json);
    tv_native_last.setText(filmInfo.toString());
}
```

4.2_GSON 框架技术

1）特点：编码简洁，谷歌官方推荐
2）下载地址：https://mvnrepository.com/artifact/com.google.code.gson/gson

4.2.1_将 json 格式的字符串{}转换为 Java 对象

1）用到的 API
<T> T fromJson(String json, Class<T> classOfT);//将 json 对象转换为 Java 对象的方法
注意：要求 json 对象中的 key 的名称与 java 对象对应的类中的属性名要相同

2）使用步骤
（1）将 Gson 的 jar 包导入到项目中
（2）创建Gson对象 ： Gson gson = new Gson();
（3）通过创建的Gson对象调用fromJson()方法，返回该JSON数据对应的Java对象
ShopInfo shopInfo = gson.fromJson(json, ShopInfo.class);

3）测试数据

```
{
    "id":2, "name":"大虾",
    "price":12.3,
    "imagePath":"http://192.168.10.165:8080/L05_Server/images/f1.jpg"
}
```

4）例子

```
// (1)将 json 格式的字符串{}转换为 Java 对象
private void jsonToJavaObject() {
```

```
    // 1 获取或创建 json
    String json = "{\n" +
            "\t\"id\":2, \"name\":\"大虾\", \n" +
            "\t\"price\":12.3, \n" +

"\t\"imagePath\":\"http://192.168.10.165:8080/L05_Server/images/f1.jpg\
"\n" +
            "}\n";

    // 2 解析 json
    Gson gson = new Gson();

    ShopInfo shopInfo = gson.fromJson(json, ShopInfo.class);

    // 3 显示 JSON 数据
    tv_native_orignal.setText(json);
    tv_native_last.setText(shopInfo.toString());
}
```

4.2.2_将 json 格式的字符串[]转换为 Java 对象的 List

1）用到的 API

T fromJson(String json, Type typeOfT);//将 json 数组转换为 Java 对象的 list

注意：要求 json 对象中的 key 的名称与 java 对象对应的类中的属性名要相同

2）使用步骤

（1）将 Gson 的 jar 包导入到项目中

（2）创建Gson对象 ： Gson gson = new Gson();

（3）通过创建的Gson对象调用fromJson()方法，返回该JSON数据对应的Java集合：

List<ShopInfo> shops = gson.fromJson(json, new TypeToken<List<ShopInfo>>() {}.getType());

3）测试数据

```
[
    {
        "id": 1,
        "imagePath": "http://192.168.10.165:8080/f1.jpg",
        "name": "大虾1",
        "price": 12.3
    },
```

```
    {
        "id": 2,
        "imagePath": "http://192.168.10.165:8080/f2.jpg",
        "name": "大虾2",
        "price": 12.5
    }
]
```

4）例子

```
//(2)将 json 格式的字符串[]转换为 Java 对象的 List
private void jsonToJavaList() {

    // 1 获取或创建 json
    String json = "[\n" +
            "    {\n" +
            "        \"id\": 1,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f1.jpg\",\n" +
            "        \"name\": \"大虾 1\",\n" +
            "        \"price\": 12.3\n" +
            "    },\n" +
            "    {\n" +
            "        \"id\": 2,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f2.jpg\",\n" +
            "        \"name\": \"大虾 2\",\n" +
            "        \"price\": 12.5\n" +
            "    }\n" +
            "]";
    // 2 解析 json
    Gson gson = new Gson();

    List<ShopInfo> shops = gson.fromJson(json, new
TypeToken<List<ShopInfo>>() {
    }.getType());

    // 3 显示 JSON 数据
    tv_native_orignal.setText(json);
    tv_native_last.setText(shops.toString());
}
```

4.2.3_将 Java 对象转换为 json 字符串{}

1）用到的 API

String toJson(Object src);

2）使用步骤

（1）将 Gson 的 jar 包导入到项目中

（2）创建Gson对象 ： Gson gson = new Gson();

（3）通过创建的Gson对象调用toJson()方法，返回json数据：

ShopInfo shop = new ShopInfo(1, "鲍鱼", 250.0, "");

String json = gson.toJson(shop);

3）例子

```
// (3)将 Java 对象转换为 json 字符串{}
private void javaToJsonObject() {
    // 1 获取或创建 Java 对象
    ShopInfo shop = new ShopInfo(1, "鲍鱼", 250.0, "");

    // 2 生成 JSON 数据
    Gson gson = new Gson();
    String json = gson.toJson(shop);

    // 3 展示 json 数据
    tv_native_orignal.setText(shop.toString());
    tv_native_last.setText(json);
}
```

4.2.3_将 Java 对象的 List 转换为 json 字符串[]

1）用到的 API

String toJson(Object src);

2）使用步骤

（1）将 Gson 的 jar 包导入到项目中

（2）创建Gson对象 ： Gson gson = new Gson();

（3）通过创建的Gson对象调用toJson()方法，返回json数据：

List<ShopInfo> shops = new ArrayList<>();

String json = gson.toJson(shops);

3）例子

```
// (4) 将 Java 对象的 List 转换为 json 字符串[]
```

```
private void javaToJsonList() {
    // 1 获取或创建 Java 集合
    List<ShopInfo> shops = new ArrayList<>();
    ShopInfo baoyu = new ShopInfo(1, "鲍鱼", 250, "baoyu");
    ShopInfo haisen = new ShopInfo(2, "海参", 251, "haisen");
    shops.add(baoyu);
    shops.add(haisen);

    // 2 生成 JSON 数据
    Gson gson = new Gson();

    String json = gson.toJson(shops);

    // 3 展示 json 数据
    tv_native_orignal.setText(shops.toString());
    tv_native_last.setText(json);
}
```

4.3_FastJson 框架技术

1）特点：Fastjson 是一个 Java 语言编写的高性能功能完善的 JSON 库。它采用一种"假定有序快速匹配"的算法，把 JSON Parse 的性能提升到极致，是目前 Java 语言中最快的 JSON 库。

2）下载地址：https://github.com/alibaba/fastjson/wiki

4.3.1_将 json 格式的字符串{}转换为 Java 对象

1）用到的 API

< T > T parseObject(String json, Class<T> classOfT); //将 json 对象转换为 Java 对象的方法

注意：要求 json 对象中的 key 的名称与 java 对象对应的类中的属性名要相同

2）使用步骤

（1）导入 fastjson 的 jar 包

（2）JSON调用parseObject()方法，获取转换后的Java对象

例如：ShopInfo shopInfo = JSON.parseObject(json, ShopInfo.class);

3）测试数据

```
{
    "id":2, "name":"大虾",
```

```
    "price":12.3,
    "imagePath":"http://192.168.10.165:8080/L05_Server/images/f1.jpg"
}
```

4）例子

```
// （1）将 json 格式的字符串{}转换为 Java 对象
private void jsonToJavaObjectByFastJson() {

    // 1 获取或创建 JSON 数据
    String json = "{\n" +
            "\t\"id\":2, \"name\":\"大虾\", \n" +
            "\t\"price\":12.3, \n" +

"\t\"imagePath\":\"http://192.168.10.165:8080/L05_Server/images/f1.jpg\
"\n" +
            "}\n";

    // 2 解析 JSON 数据
    ShopInfo shopInfo = JSON.parseObject(json, ShopInfo.class);

    // 3 显示数据
    tv_fastjson_orignal.setText(json);
    tv_fastjson_last.setText(shopInfo.toString());
}
```

4.3.2_将 json 格式的字符串[]转换为 Java 对象的 List

1）用到的 API

List<T> parseArray(String json,Class<T> classOfT);//将 json 数组转换为 Java 对象的 list

注意：要求 json 对象中的 key 的名称与 java 对象对应的类中的属性名要相同

2）使用步骤

（1）导入 fastjson 的 jar 包

（2）JSON调用parseArray()方法，获取转换后的Java集合

例如：List<ShopInfo> shopInfos = JSON.parseArray(json, ShopInfo.class);

3）测试数据

```
[
    {
        "id": 1,
        "imagePath": "http://192.168.10.165:8080/f1.jpg",
```

```
        "name": "大虾1",
        "price": 12.3
    },
    {
        "id": 2,
        "imagePath": "http://192.168.10.165:8080/f2.jpg",
        "name": "大虾2",
        "price": 12.5
    }
]
```

4）例子

```
// （2）将 json 格式的字符串[]转换为 Java 对象的 List
private void jsonToJavaListByFastJson() {

    // 1 获取或创建 JSON 数据
    String json = "[\n" +
            "    {\n" +
            "        \"id\": 1,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f1.jpg\",\n" +
            "        \"name\": \"大虾 1\",\n" +
            "        \"price\": 12.3\n" +
            "    },\n" +
            "    {\n" +
            "        \"id\": 2,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f2.jpg\",\n" +
            "        \"name\": \"大虾 2\",\n" +
            "        \"price\": 12.5\n" +
            "    }\n" +
            "]";

    // 2 解析 JSON 数据
    List<ShopInfo> shopInfos = JSON.parseArray(json, ShopInfo.class);

    // 3 显示数据
    tv_fastjson_orignal.setText(json);
    tv_fastjson_last.setText(shopInfos.toString());
}
```

4.3.3_将 Java 对象转换为 json 字符串{}

1）用到的 API

String toJSONString(Object object);

2）使用步骤

（1）导入 fastjson 的 jar 包

（2）JSON调用toJSONString()方法，获取转换后的json数据

例如：

```
ShopInfo shopInfo = new ShopInfo(1, "鲍鱼", 250.0, "baoyu");
String json = JSON.toJSONString(shopInfo);
```

3）例子

```
// （3）将 Java 对象转换为 json 字符串{}
private void javaToJsonObjectByFastJson() {

    // 1 获取 Java 对象
    ShopInfo shopInfo = new ShopInfo(1, "鲍鱼", 250.0, "baoyu");

    // 2 生成 JSON 数据
    String json = JSON.toJSONString(shopInfo);

    // 3 数据显示
    tv_fastjson_orignal.setText(shopInfo.toString());
    tv_fastjson_last.setText(json);
}
```

4.3.4_将 Java 对象的 List 转换为 json 字符串[]

1）用到的 API

String toJSONString(Object object);

2）使用步骤

（1）导入 fastjson 的 jar 包

（2）JSON调用toJSONString()方法，获取转换后的json数据

例如：

```
List<ShopInfo> shops = new ArrayList<>();
ShopInfo baoyu = new ShopInfo(1, "鲍鱼", 250.0, "baoyu");
ShopInfo longxia = new ShopInfo(2, "龙虾", 251.0, "longxia");

shops.add(baoyu);
```

```
shops.add(longxia);

String json = JSON.toJSONString(shops);
```

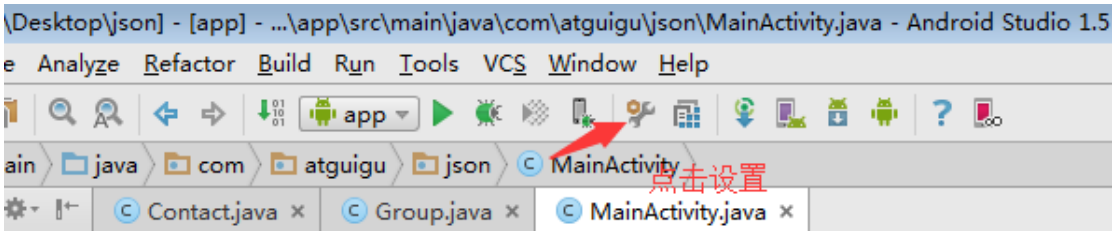
3）例子

```
// （4）将 Java 对象的 List 转换为 json 字符串[]
private void javaToJsonArrayByFastJson() {

    // 1 获取 Java 集合
    List<ShopInfo> shops = new ArrayList<>();
    ShopInfo baoyu = new ShopInfo(1, "鲍鱼", 250.0, "baoyu");
    ShopInfo longxia = new ShopInfo(2, "龙虾", 251.0, "longxia");

    shops.add(baoyu);
    shops.add(longxia);

    // 2 生成 JSON 数据
    String json = JSON.toJSONString(shops);

    // 3 数据显示
    tv_fastjson_orignal.setText(shops.toString());
    tv_fastjson_last.setText(json);
}
```

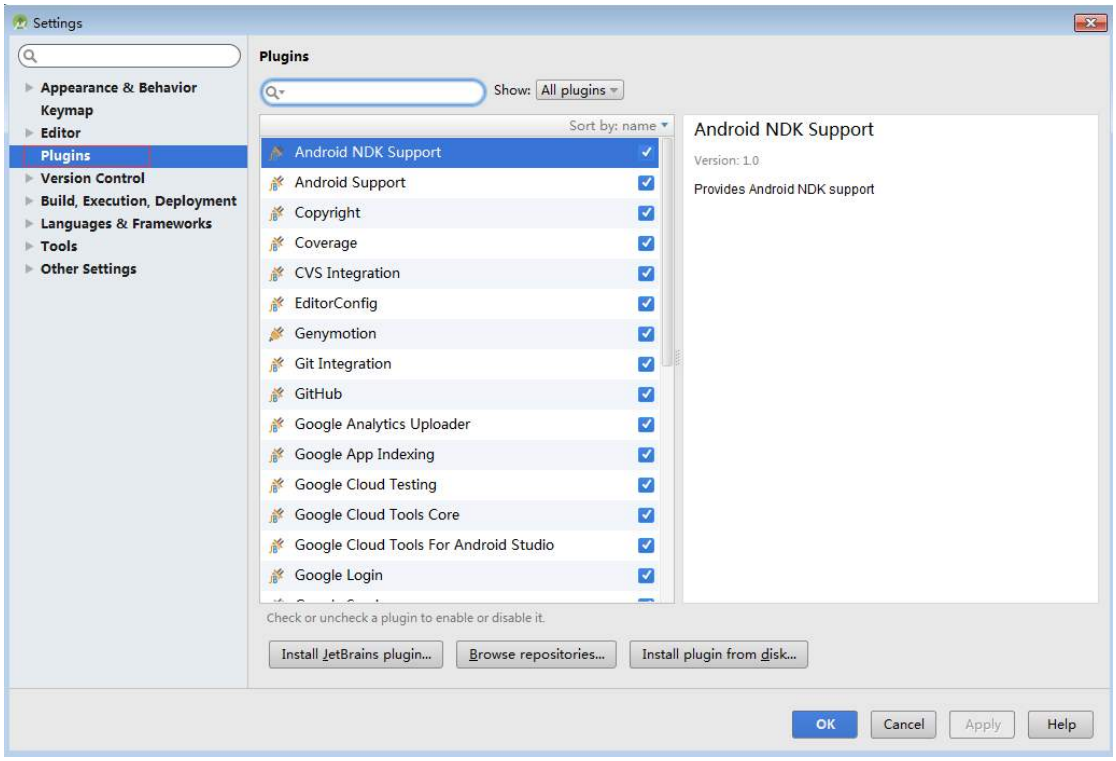
5_工具使用

5.1_GsonFormat

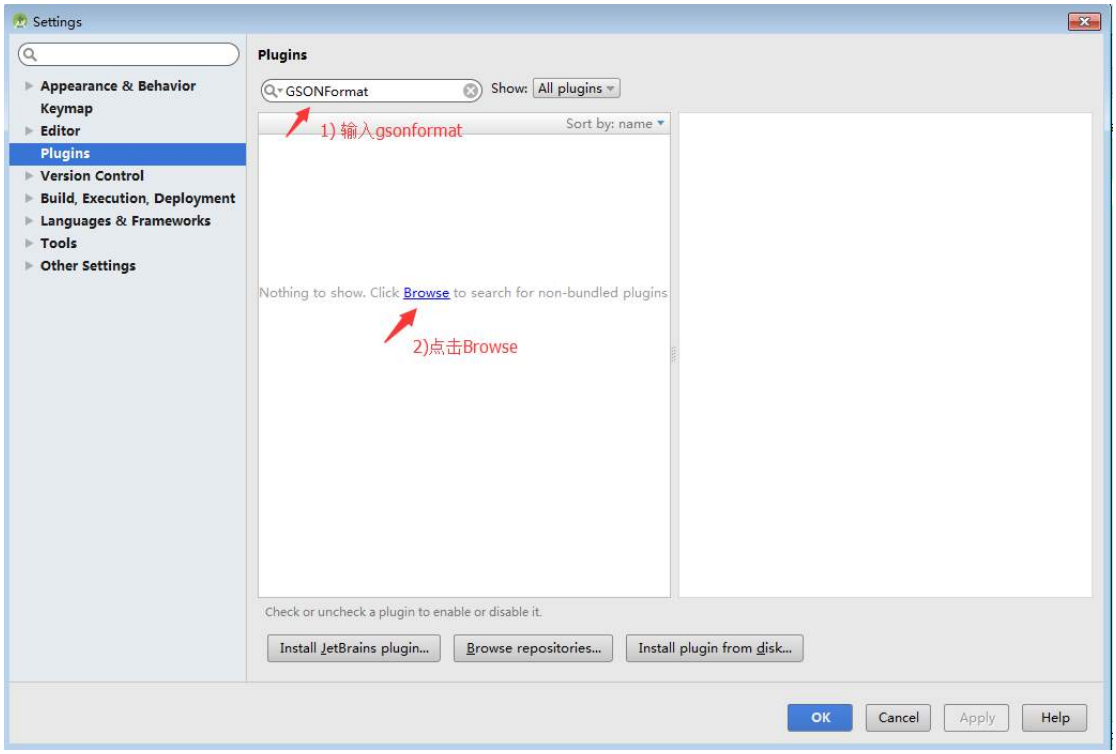
1）打开 Android studio 页面，点击设置按钮。



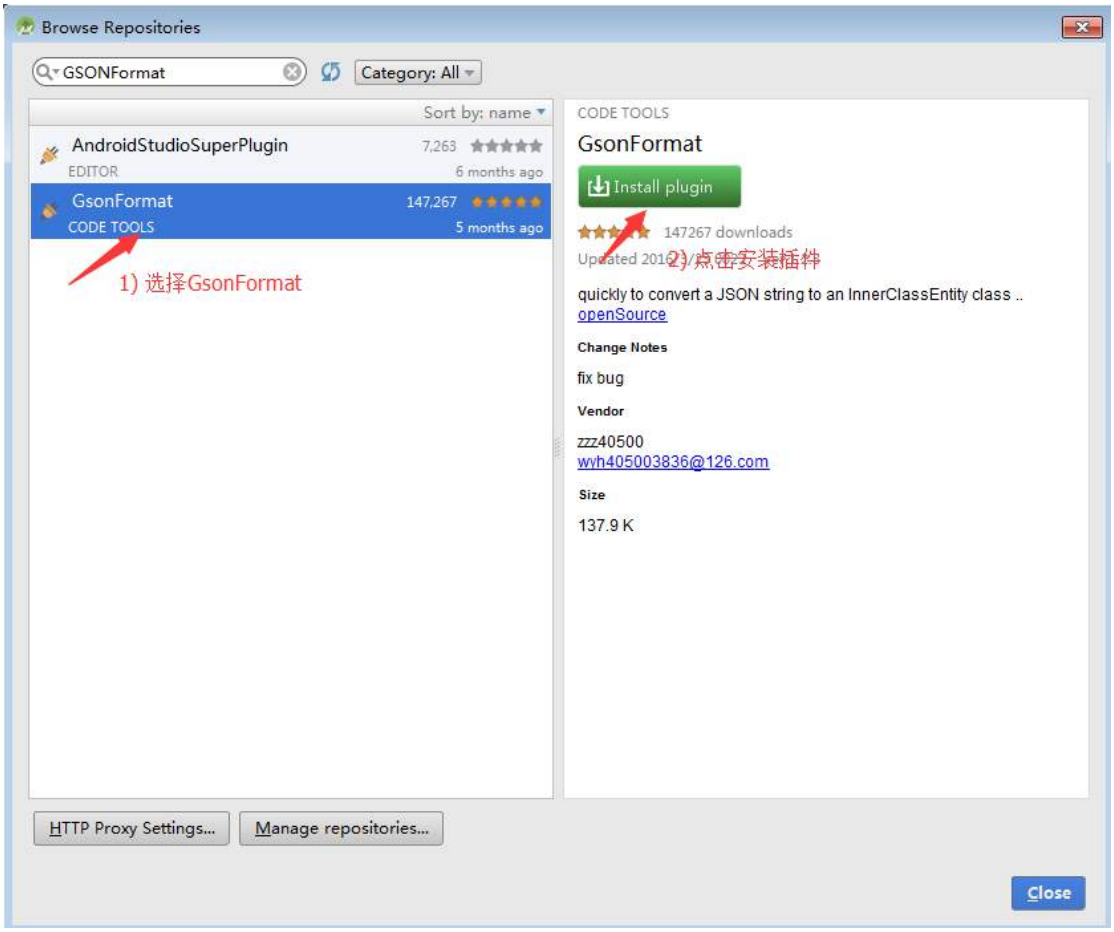
2）点击 Plugins 按钮



3）在右侧输入框中输入 gsonformat,然后点击中间部位的 Browse(必须在联网情况下点击)

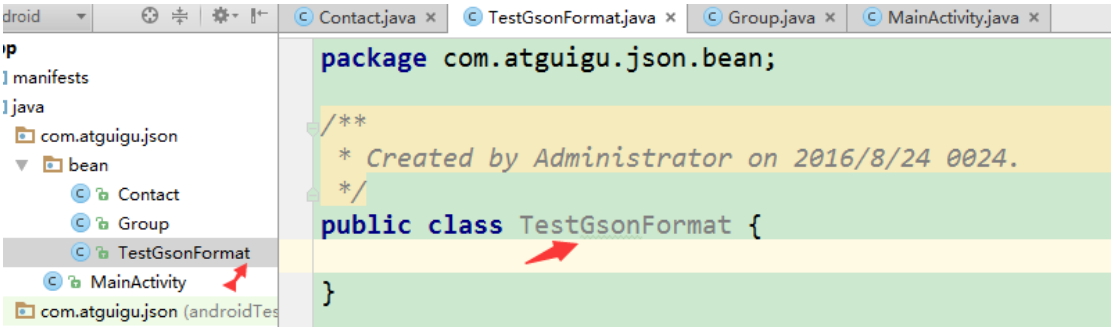


4）选择 GsonFormat，点击右侧的安装插件

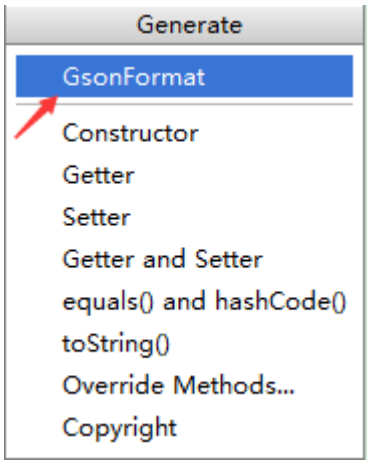


5）重启一下 Android studio

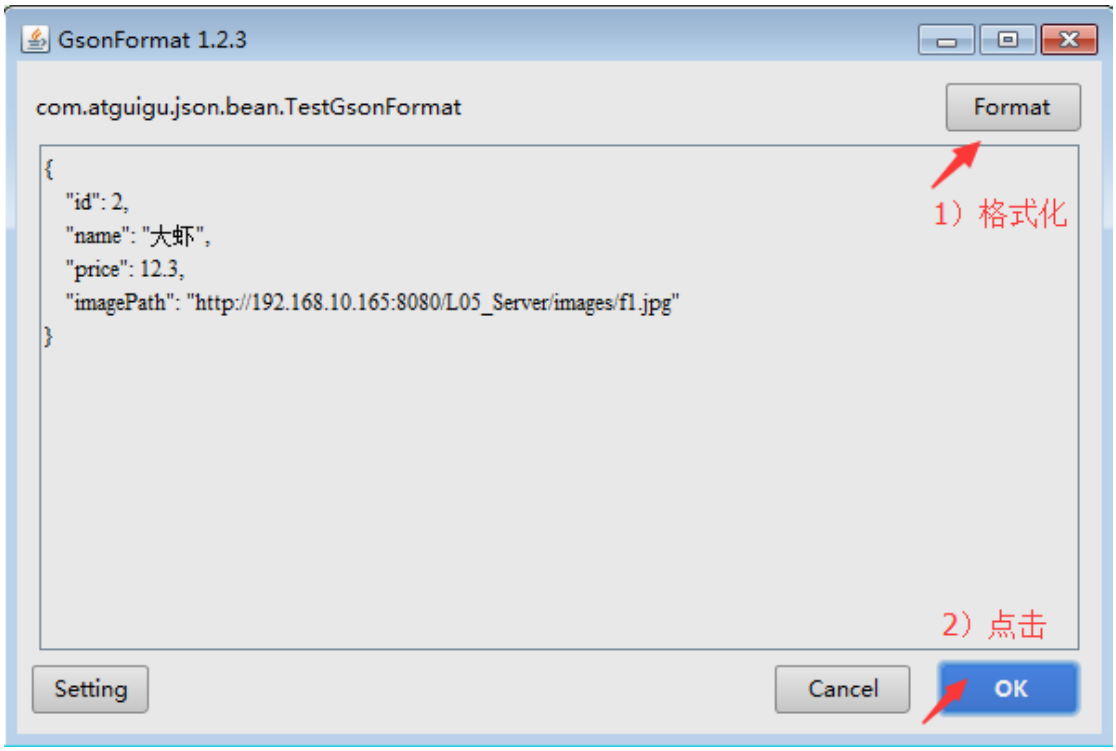
6）在 Android studio 中创建一个类



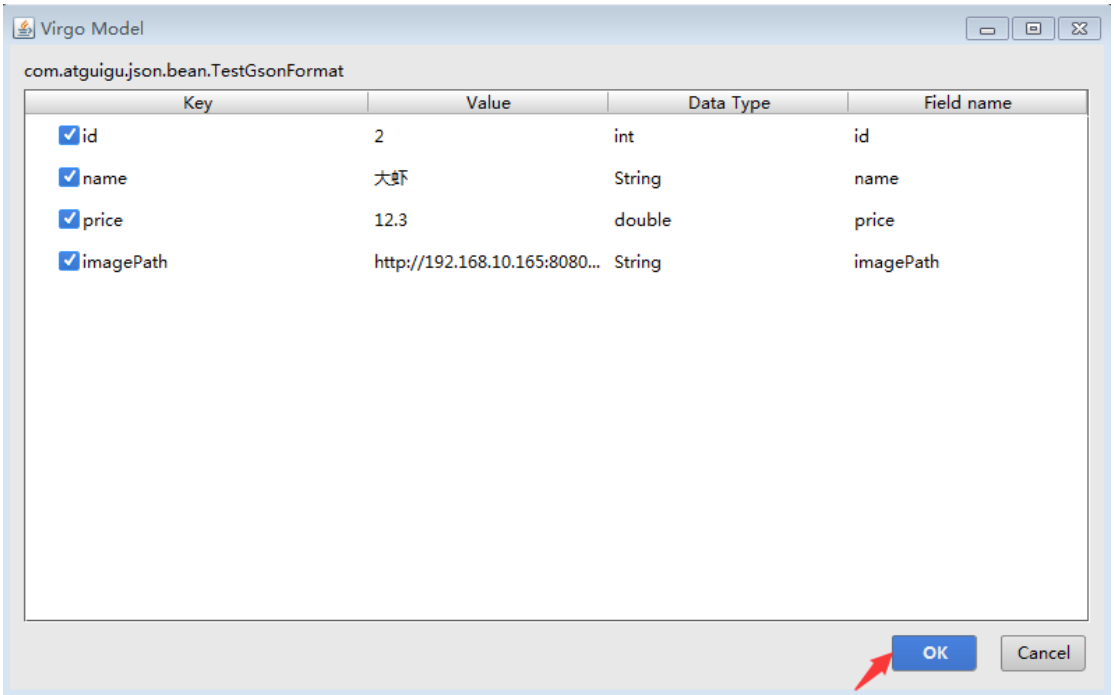
7）在该类中同时按下 alt+shift+s，并点击 GsonFormat



8）将要解析的 JSON 字符串粘贴到 GsonFormat 中



9）点击 OK



10）这样就将输入的 JSON 数据转换为了 bean 对象

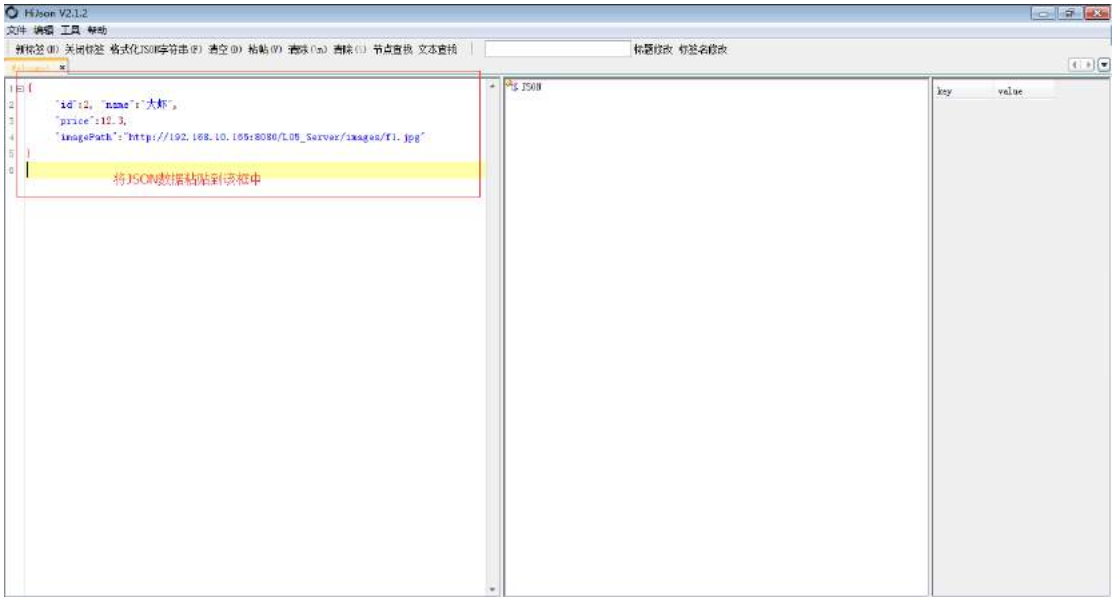
```
public class TestGsonFormat {
    /**
     * id : 2
     * name : 大虾
     * price : 12.3
     * imagePath : http://192.168.10.165:8080/L05_Server/images/f1.jpg
     */
    private int id;
    private String name;
    private double price;
    private String imagePath;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
```

```
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getImagePath() {
        return imagePath;
    }

    public void setImagePath(String imagePath) {
        this.imagePath = imagePath;
    }
}
```

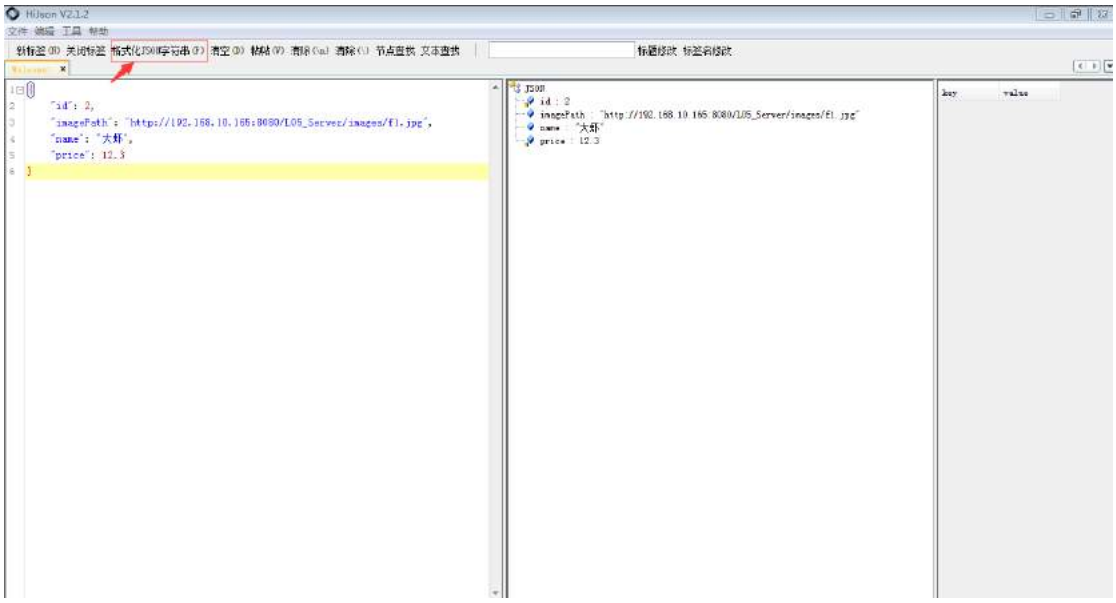
5.2_HiJson

1）双击图标

2）将要解析的 JSON 数据粘贴到左侧页面



3）点击格式化 JSON 字符串，在右侧就会方便查看 JSON 数据



第 4 讲_JSON 解析之 GSON

谷粉第 47 群：285047793

1_GSON 框架技术

1）特点：编码简洁，谷歌官方推荐
2）下载地址：https://mvnrepository.com/artifact/com.google.code.gson/gson

1.1_将 json 格式的字符串{}转换为 Java 对象

1）用到的 API
<T> T fromJson(String json, Class<T> classOfT);//将 json 对象转换为 Java 对象的方法
注意：要求 json 对象中的 key 的名称与 java 对象对应的类中的属性名要相同

2）使用步骤
（1）将 Gson 的 jar 包导入到项目中
（2）创建Gson对象 ： Gson gson = new Gson();
（3）通过创建的Gson对象调用fromJson()方法，返回该JSON数据对应的Java对象
ShopInfo shopInfo = gson.fromJson(json, ShopInfo.class);

3）测试数据

```
{
    "id":2, "name":"大虾",
    "price":12.3,
    "imagePath":"http://192.168.10.165:8080/L05_Server/images/f1.jpg"
}
```

4）例子

```
// (1)将 json 格式的字符串{}转换为 Java 对象
private void jsonToJavaObject() {

    // 1 获取或创建 json
    String json = "{\n" +
            "\t\"id\":2, \"name\":\"大虾\", \n" +
            "\t\"price\":12.3, \n" +
```

```
"\t\"imagePath\":\"http://192.168.10.165:8080/L05_Server/images/f1.jpg\
"\n" +"}\n";

    // 2 解析 json
    Gson gson = new Gson();

    ShopInfo shopInfo = gson.fromJson(json, ShopInfo.class);

    // 3 显示 JSON 数据
    tv_native_orignal.setText(json);
    tv_native_last.setText(shopInfo.toString());
}
```

1.2_将 json 格式的字符串[]转换为 Java 对象的 List

1）用到的 API

T fromJson(String json, Type typeOfT);//将 json 数组转换为 Java 对象的 list

注意：要求 json 对象中的 key 的名称与 java 对象对应的类中的属性名要相同

2）使用步骤

（1）将 Gson 的 jar 包导入到项目中

（2）创建Gson对象 ： Gson gson = new Gson();

（3）通过创建的Gson对象调用fromJson()方法，返回该JSON数据对应的Java集合：

List<ShopInfo> shops = gson.fromJson(json, new TypeToken<List<ShopInfo>>() {}.getType());

3）测试数据

```
[
    {
        "id": 1,
        "imagePath": "http://192.168.10.165:8080/f1.jpg",
        "name": "大虾1",
        "price": 12.3
    },
    {
        "id": 2,
        "imagePath": "http://192.168.10.165:8080/f2.jpg",
        "name": "大虾2",
        "price": 12.5
    }
]
```

4）例子

```
//(2)将 json 格式的字符串[]转换为 Java 对象的 List
private void jsonToJavaList() {

    // 1 获取或创建 json
    String json = "[\n" +
            "    {\n" +
            "        \"id\": 1,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f1.jpg\",\n" +
            "        \"name\": \"大虾 1\",\n" +
            "        \"price\": 12.3\n" +
            "    },\n" +
            "    {\n" +
            "        \"id\": 2,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f2.jpg\",\n" +
            "        \"name\": \"大虾 2\",\n" +
            "        \"price\": 12.5\n" +
            "    }\n" +
            "]";
    // 2 解析 json
    Gson gson = new Gson();

    List<ShopInfo> shops = gson.fromJson(json, new
TypeToken<List<ShopInfo>>() {
    }.getType());

    // 3 显示 JSON 数据
    tv_native_orignal.setText(json);
    tv_native_last.setText(shops.toString());
}
```

1.3_将 Java 对象转换为 json 字符串{}

1）用到的 API

String toJson(Object src);

2）使用步骤

（1）将 Gson 的 jar 包导入到项目中

（2）创建Gson对象 ： Gson gson = new Gson();

（3）通过创建的Gson对象调用toJson()方法，返回json数据：

```
ShopInfo shop = new ShopInfo(1, "鲍鱼", 250.0, "");
String json = gson.toJson(shop);
```

3）例子

```
// (3)将 Java 对象转换为 json 字符串{}
private void javaToJsonObject() {
    // 1 获取或创建 Java 对象
    ShopInfo shop = new ShopInfo(1, "鲍鱼", 250.0, "");

    // 2 生成 JSON 数据
    Gson gson = new Gson();
    String json = gson.toJson(shop);

    // 3 展示 json 数据
    tv_native_orignal.setText(shop.toString());
    tv_native_last.setText(json);
}
```

1.4_将 Java 对象的 List 转换为 json 字符串[]

1）用到的 API

String toJson(Object src);

2）使用步骤

（1）将 Gson 的 jar 包导入到项目中

（2）创建Gson对象 ： Gson gson = new Gson();

（3）通过创建的Gson对象调用toJson()方法，返回json数据：

```
List<ShopInfo> shops = new ArrayList<>();
String json = gson.toJson(shops);
```

3）例子

```
// (4) 将 Java 对象的 List 转换为 json 字符串[]
private void javaToJsonList() {
    // 1 获取或创建 Java 集合
    List<ShopInfo> shops = new ArrayList<>();
    ShopInfo baoyu = new ShopInfo(1, "鲍鱼", 250, "baoyu");
    ShopInfo haisen = new ShopInfo(2, "海参", 251, "haisen");
    shops.add(baoyu);
    shops.add(haisen);

    // 2 生成 JSON 数据
    Gson gson = new Gson();
```

```
    String json = gson.toJson(shops);

    // 3 展示json 数据
    tv_native_orignal.setText(shops.toString());
    tv_native_last.setText(json);
}
```

第 5 讲_JSON 解析之 FastJson

谷粉第 47 群：285047793

1_FastJson 框架技术

1）特点：Fastjson 是一个 Java 语言编写的高性能功能完善的 JSON 库。它采用一种"假定有序快速匹配"的算法，把 JSON Parse 的性能提升到极致，是目前 Java 语言中最快的 JSON 库。

2）下载地址：https://github.com/alibaba/fastjson/wiki

1.1_将 json 格式的字符串{}转换为 Java 对象

1）用到的 API

< T > T parseObject(String json, Class<T> classOfT); //将 json 对象转换为 Java 对象的方法

注意：要求 json 对象中的 key 的名称与 java 对象对应的类中的属性名要相同

2）使用步骤

（1）导入 fastjson 的 jar 包

（2）JSON调用parseObject()方法，获取转换后的Java对象

例如：ShopInfo shopInfo = JSON.parseObject(json, ShopInfo.class);

3）测试数据

```
{
    "id":2, "name":"大虾",
    "price":12.3,
    "imagePath":"http://192.168.10.165:8080/L05_Server/images/f1.jpg"
}
```

4）例子

```
// （1）将 json 格式的字符串{}转换为 Java 对象
private void jsonToJavaObjectByFastJson() {

    // 1 获取或创建 JSON 数据
    String json = "{\n" +
            "\t\"id\":2, \"name\":\"大虾\", \n" +
```

```
            "\t\"price\":12.3, \n" +

"\t\"imagePath\":\"http://192.168.10.165:8080/L05_Server/images/f1.jpg\
"\n" +
            "}\n";

    // 2 解析 JSON 数据
    ShopInfo shopInfo = JSON.parseObject(json, ShopInfo.class);

    // 3 显示数据
    tv_fastjson_orignal.setText(json);
    tv_fastjson_last.setText(shopInfo.toString());
}
```

1.2_将 json 格式的字符串[]转换为 Java 对象的 List

1）用到的 API

List<T> parseArray(String json,Class<T> classOfT);//将 json 数组转换为 Java 对象的 list

注意：要求 json 对象中的 key 的名称与 java 对象对应的类中的属性名要相同

2）使用步骤

（1）导入 fastjson 的 jar 包

（2）JSON调用parseArray()方法，获取转换后的Java集合

例如：List<ShopInfo> shopInfos = JSON.parseArray(json, ShopInfo.class);

3）测试数据

```
[
    {
        "id": 1,
        "imagePath": "http://192.168.10.165:8080/f1.jpg",
        "name": "大虾1",
        "price": 12.3
    },
    {
        "id": 2,
        "imagePath": "http://192.168.10.165:8080/f2.jpg",
        "name": "大虾2",
        "price": 12.5
    }
]
```

4）例子

```
// （2）将 json 格式的字符串[]转换为 Java 对象的 List
private void jsonToJavaListByFastJson() {

    // 1 获取或创建 JSON 数据
    String json = "[\n" +
            "    {\n" +
            "        \"id\": 1,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f1.jpg\",\n" +
            "        \"name\": \"大虾 1\",\n" +
            "        \"price\": 12.3\n" +
            "    },\n" +
            "    {\n" +
            "        \"id\": 2,\n" +
            "        \"imagePath\":
\"http://192.168.10.165:8080/f2.jpg\",\n" +
            "        \"name\": \"大虾 2\",\n" +
            "        \"price\": 12.5\n" +
            "    }\n" +
            "]";

    // 2 解析 JSON 数据
    List<ShopInfo> shopInfos = JSON.parseArray(json, ShopInfo.class);

    // 3 显示数据
    tv_fastjson_orignal.setText(json);
    tv_fastjson_last.setText(shopInfos.toString());
}
```

1.3_将 Java 对象转换为 json 字符串{}

1）用到的 API

String toJSONString(Object object);

2）使用步骤

（1）导入 fastjson 的 jar 包

（2）JSON调用toJSONString()方法，获取转换后的json数据

例如：

ShopInfo shopInfo = new ShopInfo(1, "鲍鱼", 250.0, "baoyu");

String json = JSON.toJSONString(shopInfo);

3）例子

```
// （3）将 Java 对象转换为 json 字符串{}
private void javaToJsonObjectByFastJson() {

    // 1 获取 Java 对象
    ShopInfo shopInfo = new ShopInfo(1, "鲍鱼", 250.0, "baoyu");

    // 2 生成 JSON 数据
    String json = JSON.toJSONString(shopInfo);

    // 3 数据显示
    tv_fastjson_orignal.setText(shopInfo.toString());
    tv_fastjson_last.setText(json);
}
```

1.4_将 Java 对象的 List 转换为 json 字符串[]

1）用到的 API

String toJSONString(Object object);

2）使用步骤

（1）导入 fastjson 的 jar 包

（2）JSON调用toJSONString()方法，获取转换后的json数据

例如：

```
List<ShopInfo> shops = new ArrayList<>();
ShopInfo baoyu = new ShopInfo(1, "鲍鱼", 250.0, "baoyu");
ShopInfo longxia = new ShopInfo(2, "龙虾", 251.0, "longxia");

shops.add(baoyu);
shops.add(longxia);

String json = JSON.toJSONString(shops);
```

3）例子

```
// （4）将 Java 对象的 List 转换为 json 字符串[]
private void javaToJsonArrayByFastJson() {

    // 1 获取 Java 集合
    List<ShopInfo> shops = new ArrayList<>();
    ShopInfo baoyu = new ShopInfo(1, "鲍鱼", 250.0, "baoyu");
    ShopInfo longxia = new ShopInfo(2, "龙虾", 251.0, "longxia");
```

```
    shops.add(baoyu);
    shops.add(longxia);

    // 2 生成 JSON 数据
    String json = JSON.toJSONString(shops);

    // 3 数据显示
    tv_fastjson_orignal.setText(shops.toString());
    tv_fastjson_last.setText(json);
}
```

第 6 讲_xUtils3 注解模块

谷粉第 47 群：285047793

本文档已经录制成视频，可以到尚硅谷官网下载：
下载地址：http://atguigu.com/download.shtml#kj

1_xUtils3 简介

1.1_简介

xUtils3 是 xUtils 的升级版，功能和性能都在提高，xUtils3 主要有四大模块：注解模块，联网模块，图片加载模块，数据库模块；

注解模块：
用于在 Activity 或者 Fragment 中初始化布局文件，便于代码更加简洁；

联网模块：
xUtils3 支持超大文件(超过 2G)上传，更全面的 http 请求协议支持(11 种谓词)，拥有更加灵活的 ORM，更多的事件注解支持且不受混淆影响；

图片加载模块：
加载图片很方便并且不用担心内存溢出，还可以图片绑定支持 gif(受系统兼容性影响，部分 gif 文件只能静态显示)，webp；支持圆角，圆形，方形等裁剪，支持自动旋转.

数据库模块：
数据库 api 简化提高性能，达到和 greenDao 一致的性能

1.2_xUtils3 的主要功能

1、注解
2、联网请求文本数据

3、大文件下载
4、大文件上传
5、请求图片
6、数据库模块达到和 greenDao 一致的性能

1.3_下载地址&运行 xUtils3 案例

https://github.com/wyouflf/xUtils3

2_xUtils3 注解模块

主要讲在 Activity 中使用 xUtils3 的注解和在 Fragment 中使用 xUtils3 的注解

1.1_XUtils3MainActivity 布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <!--标题栏-->
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:background="@android:color/holo_blue_light"
        android:gravity="center"
        android:orientation="horizontal">


        <TextView
            android:id="@+id/tv_title"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="20sp" />

    </LinearLayout>

    <Button
        android:id="@+id/btn_annotation"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="xUtils3 注解模块" />

    <Button
        android:id="@+id/btn_net"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="xUtils3 联网模块" />

    <Button
        android:id="@+id/btn_image"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="xUtils3 请求图片" />


</LinearLayout>
```

1.2_在 Activity 中使用注解初始化布局

Activity 的注解
1. 在 Application 的 oncreate 方法中加入下面代码:
x.Ext.init(this);
2. 在 Activity 的 oncreate 方法中加入下面代码:
x.view().inject(this);
3. 加载当前的 Activity 布局需要如下注解:
@ContentView 加入到 Activity 的上方
4. 给 View 进行初始化需要如下注解:
@InjectView
5. 处理控件的各种响应事件需要如下注解:
@Envent
使用注解后你会发现写代码更加简洁了

```
package com.atguigu.android.xutils3;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import com.atguigu.android.R;

import org.xutils.view.annotation.ContentView;
import org.xutils.view.annotation.Event;
import org.xutils.view.annotation.ViewInject;
import org.xutils.x;

/**
 * 作者：尚硅谷-杨光福 on 2016/8/23 20:21
 * 微信：yangguangfu520
 * QQ 号：541433511
 * 作用：xUtils3
 */
@ContentView(R.layout.activity_xutils3)
public class XUtils3Activity extends Activity {

    /**
     * 初始化 TextView
     */
    @ViewInject(R.id.tv_title)
    private TextView tv_title;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //添加注解
        x.view().inject(this);
        //设置文本
        tv_title.setText("xUtils3 详解");    }


    /**
     * 多个控件的点击事件共用一个方法
     * @param view
```

```
     */
    @Event(value =
{R.id.btn_annotation,R.id.btn_net,R.id.btn_image,R.id.btn_database})
    private void getEnvent(View view){
        switch (view.getId()){
            case R.id.btn_annotation:
                Intent intent = new Intent(this,FragmentActivity.class);
                startActivity(intent);
                break;
            case R.id.btn_net:
                Toast.makeText(XUtils3Activity.this, "进入网络模块",
Toast.LENGTH_SHORT).show();
                break;
            case R.id.btn_image:
                Toast.makeText(XUtils3Activity.this, "进入网络模块",
Toast.LENGTH_SHORT).show();
                break;
            case R.id.btn_database:
                Toast.makeText(XUtils3Activity.this, "进入数据库模块",
Toast.LENGTH_SHORT).show();
                break;
        }
    }

    /**
     * 单个点击事件
     * @param view
     */
    @Event(value = R.id.btn_database)
    private void database(View view){
        Toast.makeText(XUtils3Activity.this, "进入数据库模块单独的",
Toast.LENGTH_SHORT).show();
    }
}
```

1.3_在 Fragment 中使用注解初始化布局

1_在 FragmentActivity 中添加 Fragment

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <!--标题栏-->
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:background="@android:color/holo_blue_light"
        android:gravity="center"
        android:orientation="horizontal">


        <TextView
            android:id="@+id/tv_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="20sp" />

    </LinearLayout>


    <FrameLayout
        android:id="@+id/fl_content"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

Fragment 的注解
代码如下

```
import android.os.Bundle;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
```

```
import android.widget.TextView;

import com.atguigu.android.R;

import org.xutils.view.annotation.ContentView;
import org.xutils.view.annotation.ViewInject;
import org.xutils.x;

/**
 * 作者：尚硅谷-杨光福 on 2016/9/5 10:23
 * 微信：yangguangfu520
 * QQ 号：541433511
 * 作用：FragemtnActivity
 */
//设置布局文件
@ContentView(R.layout.activity_frament)
public class FragmentActivity extends
android.support.v4.app.FragmentActivity {

    //初始化布局里面的文本控件
    @ViewInject(R.id.tv_title)
    private TextView tv_title;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //注入 Activity
        x.view().inject(this);

        //直接使用 TextView 控件
        tv_title.setText("在 Fragment 中使用注解初始化布局");

        FragmentManager fm = getSupportFragmentManager();
        FragmentTransaction transaction = fm.beginTransaction();
        transaction.replace(R.id.fl_content,new DemoFragment());
        transaction.commit();
    }
}
```

2_DemoFragment 布局和代码

布局 fragment_demo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:id="@+id/btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/tv_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

代码

```
/**
 * 作者：尚硅谷-杨光福 on 2016/9/5 10:48
 * 微信：yangguangfu520
 * QQ 号：541433511
 */
@ContentView(R.layout.fragment_demo)
public class DemoFragment extends Fragment {



    @ViewInject(R.id.tv_text)
    private TextView textView;

    @ViewInject(R.id.btn)
    private Button button;

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
//        return super.onCreateView(inflater, container,
```

```
savedInstanceState);
        return x.view().inject(this, inflater, container);
    }


    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        textView.setText("该控件使用注解初始化的");
        button.setText("我是按钮");
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getActivity(), "点击了",
Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

第 6 讲_图片加载框架之 ImageLoader

谷粉第 47 群：285047793

1_特点

1）多线程下载图片，图片可以来源于网络，文件系统，项目文件夹 assets 中以及 drawable 中等

2）支持随意的配置 ImageLoader，例如线程池，图片下载器，内存缓存策略，硬盘缓存策略，图片显示选项以及其他的一些配置

3）支持图片的内存缓存，文件系统缓存或者 SD 卡缓存

4）支持图片下载过程的监听

5）根据控件(ImageView)的大小对 Bitmap 进行裁剪，减少 Bitmap 占用过多的内存

6）较好的控制图片的加载过程，例如暂停图片加载，重新开始加载图片，一般使用在 ListView,GridView 中，滑动过程中暂停加载图片，停止滑动的时候去加载图片

7）提供在较慢的网络下对图片进行加载

2_下载地址

https://github.com/nostra13/Android-Universal-Image-Loader

3_使用步骤

1）导入 universal-image-loader-1.9.5.jar 到项目中

2）创建 MyApplication 继承 Application，在 oncreate()中初始化 ImageLoader

（1）初始化 ImageLoaderConfiguration

```
// 初始化参数
ImageLoaderConfiguration config = new
ImageLoaderConfiguration.Builder(context)
        .threadPriority(Thread.NORM_PRIORITY - 2)              // 线程优先
```

```
级
        .denyCacheImageMultipleSizesInMemory()                    // 当同一个
Uri 获取不同大小的图片，缓存到内存时，只缓存一个。默认会缓存多个不同的大小的相同
图片
        .discCacheFileNameGenerator(new Md5FileNameGenerator()) // 将保存
的时候的URI 名称用MD5
        .tasksProcessingOrder(QueueProcessingType.LIFO)          // 设置图
片下载和显示的工作队列排序
        .writeDebugLogs()                                        // 打印debug
log
        .build();
```

（2）ImageLoader 全局配置

```
// 全局初始化此配置
ImageLoader.getInstance().init(config);
```

（3）完整代码

```
public class MyApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();

        initImageLoader(this);
    }

    // 初始化 imageloader
    private void initImageLoader(Context context) {

        // 初始化参数
        ImageLoaderConfiguration config = new
ImageLoaderConfiguration.Builder(context)
                .threadPriority(Thread.NORM_PRIORITY - 2)                //
线程优先级
                .denyCacheImageMultipleSizesInMemory()                   //
当同一个Uri 获取不同大小的图片，缓存到内存时，只缓存一个。默认会缓存多个不同的
大小的相同图片
                .discCacheFileNameGenerator(new Md5FileNameGenerator()) //
将保存的时候的URI 名称用MD5
                .tasksProcessingOrder(QueueProcessingType.LIFO)          //
设置图片下载和显示的工作队列排序
                .writeDebugLogs()                                        // 打
印debug log
                .build();
```

```
        // 全局初始化此配置
        ImageLoader.getInstance().init(config);
    }
}
```

3）将创建的 MyApplication 在 AndroidManifest.xml 中注册

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.atguigu.imageloader">

    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>

    <application
        android:name=".MyApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="imageloader"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".activity.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

4）在 AndroidManifest.xml 中添加联网权限和写 sdk 权限

```
<uses-permission
android:name="android.permission.INTERNET"></uses-permission>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permiss
ion>
```

5）初始化 DisplayImageOptions

```
DisplayImageOptions options = new DisplayImageOptions.Builder()
        .showStubImage(R.drawable.ic_stub)          // 设置图片下载期间显示
的图片
        .showImageForEmptyUri(R.drawable.ic_empty)  // 设置图片 Uri 为空或
是错误的时候显示的图片
        .showImageOnFail(R.drawable.ic_error)       // 设置图片加载或解码过
程中发生错误显示的图片
        .cacheInMemory(true)                        // 设置下载的图片是否缓
存在内存中
        .cacheOnDisk(true)                          // 设置下载的图片是否缓存
在 SD 卡中
        .displayer(new RoundedBitmapDisplayer(20))  // 设置成圆角图片
        .build();                                   // 创建配置过得
DisplayImageOption 对象
```

6）获取 ImageLoader 实例

```
private ImageLoader imageLoader = ImageLoader.getInstance();
```

7）显示加载的图片

```
// 参数 1:图片 url; 参数 2:显示图片的控件; 参数 3:显示图片的设置; 参数 4:监听器
imageLoader.displayImage(Constants.IMAGES[position], holder.image,
options, mFirstLoadImageListener);
```

4_例子

4.1_准备工作 （详见 3_使用步骤）

1）导入 universal-image-loader-1.9.5.jar 到项目中

2）创建 MyApplication 继承 Application，在 oncreate()中初始化 ImageLoader

（1）初始化 ImageLoaderConfiguration

（2）ImageLoader 全局配置

3）将创建的 MyApplication 在 AndroidManifest.xml 中注册

4）在 AndroidManifest.xml 中添加联网权限和写 sdk 权限

4.2_在 ListView 中加载图片

1）布局文件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="@android:color/holo_blue_light"
        android:gravity="center"
        android:text="ImagleLoager_Listview"
        android:textColor="@android:color/white"
        android:textSize="25sp" />

    <ListView
        android:id="@+id/lv_imageloader"
        android:layout_width="match_parent"
```

```
        android:layout_height="match_parent">
    </ListView>
</LinearLayout>
```

2）初始化 ListView

```
private void initView() {
    lv_imageloader = (ListView)findViewById(R.id.lv_imageloader);
}

private void initData() {
 // 创建适配器
     listViewAdapter = new ListViewAdapter(this);
 // 添加适配器到 listview 中
     lv_imageloader.setAdapter(listViewAdapter);
 }
```

3）初始化适配器

（1）构造器

```
public ListViewAdapter(Context context) {
    // 获取上下文
    mContext = context;

    // 获取 ImageLoader 实例
    imageLoader = ImageLoader.getInstance();

    options = new DisplayImageOptions.Builder()
            .showStubImage(R.drawable.ic_stub)          // 设置图片下载期间
显示的图片
            .showImageForEmptyUri(R.drawable.ic_empty)  // 设置图片 Uri 为
空或是错误的时候显示的图片
            .showImageOnFail(R.drawable.ic_error)       // 设置图片加载或解
码过程中发生错误显示的图片
            .cacheInMemory(true)                        // 设置下载的图片是
否缓存在内存中
            .cacheOnDisk(true)                          // 设置下载的图片是否
缓存在 SD 卡中
            .displayer(new RoundedBitmapDisplayer(20))  // 设置成圆角图片
            .build();                                   // 创建配置过得
DisplayImageOption 对象

    // 存放已经显示的图片的集合
    displayedImages = Collections.synchronizedList(new
LinkedList<String>());
}
```

（2）四个核心方法

```
@Override
public int getCount() {
    return Constants.IMAGES.length;
}

@Override
public Object getItem(int position) {
    return Constants.IMAGES[position];
}

@Override
public long getItemId(int position) {
    return position;
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;

    if (convertView == null) {
        convertView = View.inflate(mContext, R.layout.item_listview_image,
null);

        holder = new ViewHolder();
        holder.text = (TextView) convertView.findViewById(R.id.text);
        holder.image = (ImageView) convertView.findViewById(R.id.image);

        convertView.setTag(holder);        // 给 View 添加一个格外的数据
    } else {
        holder = (ViewHolder) convertView.getTag(); // 把数据取出来
    }

    holder.text.setText("Item " + (position + 1));    // TextView 设置文本

    /**
     * 显示图片
     * 参数1：图片 url
     * 参数2：显示图片的控件
     * 参数3：显示图片的设置
     * 参数4：监听器
     */
    imageLoader.displayImage(Constants.IMAGES[position], holder.image,
```

```
options, mFirstLoadImageListener);

    return convertView;
}
```

（3）监听图片下载过程

```
// 监听图片下载过程
private
com.nostra13.universalimageloader.core.listener.ImageLoadingListener
mFirstLoadImageListener = new SimpleImageLoadingListener(){

    // 图片加载完毕
    @Override
    public void onLoadingComplete(String imageUri, View view, Bitmap
loadedImage) {
        super.onLoadingComplete(imageUri, view, loadedImage);

        if(loadedImage != null) {
            ImageView imageView = (ImageView) view;

            // 是否第一次显示
            boolean firstDisplay = !displayedImages.contains(imageUri);

            if (firstDisplay) {

                // 图片淡入效果
                FadeInBitmapDisplayer.animate(imageView, 500);

                displayedImages.add(imageUri);
            }
        }
    }
}
```

（4）清除已经显示图片的集合

```
// 清除已经显示图片的集合
public void clearImage(){
    displayedImages.clear();
}
```

4.3_在 GridView 中加载图片

1）初始化布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="@android:color/holo_blue_light"
        android:gravity="center"
        android:text="ImagleLoager_Gridview"
        android:textColor="@android:color/white"
        android:textSize="25sp" />

    <GridView
        android:id="@+id/gv_imageloader"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center"
        android:horizontalSpacing="4dip"
        android:numColumns="3"
        android:padding="4dip"
        android:stretchMode="columnWidth"
        android:verticalSpacing="4dip" />

</LinearLayout>
```

2）初始化 view

```
private void initView() {
    gv_imageloader = (GridView)findViewById(R.id.gv_imageloader);
}

private void initData() {
    GridViewAdaper gridViewAdaper = new GridViewAdaper(this);
    gv_imageloader.setAdapter(gridViewAdaper);
}
```

3）初始化适配器

（1）构造器

```
public GridViewAdaper(Context context) {
    // 获取上下文
    mContext = context;
```

```
    // 获取 ImageLoader 实例
    imageLoader = ImageLoader.getInstance();

    // 配置加载图片参数
    options = new DisplayImageOptions.Builder()
            .showStubImage(R.drawable.ic_stub)          // 设置图片下载期间显示的图片
            .showImageForEmptyUri(R.drawable.ic_empty)  // 设置图片 Uri 为空或是错误的时候显示的图片
            .showImageOnFail(R.drawable.ic_error)       // 设置图片加载或解码过程中发生错误显示的图片
            .cacheInMemory(true)                        // 设置下载的图片是否缓存在内存中
            .cacheOnDisk(true)                          // 设置下载的图片是否缓存在 SD 卡中
            .bitmapConfig(Bitmap.Config.RGB_565)        // 设置图片的解码类型
            .build();                                   // 创建配置过得 DisplayImageOption 对象
}
```

（2）四个核心方法

```
@Override
public int getCount() {
    return Constants.IMAGES.length;
}

@Override
public Object getItem(int position) {
    return Constants.IMAGES[position];
}

@Override
public long getItemId(int position) {
    return position;
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {

    ImageView imageView;

    if (convertView == null) {
```

```
        imageView = (ImageView) View.inflate(mContext,
R.layout.item_grid_image, null);
    } else {
        imageView = (ImageView) convertView;
    }

    // 将图片显示任务增加到执行池，图片将被显示到 ImageView 当轮到此 ImageView
    imageLoader.displayImage(Constants.IMAGES[position], imageView,
options);

    return imageView;
}
```

4.4_在 ViewPager 中加载图片

1）初始化布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="@android:color/holo_blue_light"
        android:gravity="center"
        android:text="ImagleLoager_ViewPager"
        android:textColor="@android:color/white"
        android:textSize="25sp" />

    <android.support.v4.view.ViewPager
        android:id="@+id/vp_imageloader"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

2）初始化 view

```
private void initView() {
    vp_imageloader = (ViewPager)findViewById(R.id.vp_imageloader);
```

```
}

private void initData() {
    // 创建适配器
    ViewPagerAdapter viewPagerAdapter = new ViewPagerAdapter(this);
    // 将适配器添加到 viewPager 中
    vp_imageloader.setAdapter(viewPagerAdapter);

    // 默认显示第一个页面
    vp_imageloader.setCurrentItem(0);
}
```

3）初始化适配器

（1）构造器

```
public ViewPagerAdapter(Context context) {
    // 获取上下文
    mContext = context;

    // 获取 ImageLoader 实例
    imageLoader = ImageLoader.getInstance();

    // 配置图片加载参数
    options = new DisplayImageOptions.Builder()
            .showImageForEmptyUri(R.drawable.ic_empty)  // 设置图片 Uri 为
空或是错误的时候显示的图片
            .showImageOnFail(R.drawable.ic_error)       // 设置图片加载或解
码过程中发生错误显示的图片
            .resetViewBeforeLoading(true)               // 设置图片在下载前
是否重置，复位
            .cacheOnDisc(true)                          // 设置下载的图片是否
缓存在 SD 卡中
            .imageScaleType(ImageScaleType.EXACTLY)     // 设置图片以如何的
编码方式显示
            .bitmapConfig(Bitmap.Config.RGB_565)        // 设置图片的解码类
型
            .displayer(new FadeInBitmapDisplayer(300))  // 设置图片渐变显示
            .build();
}
```

（2）四个核心方法

```
@Override
    public void destroyItem(ViewGroup container, int position, Object
object) {
//        super.destroyItem(container, position, object);
```

```
        ((ViewPager) container).removeView((View) object);
    }

    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        View imageLayout = View.inflate(mContext,
R.layout.item_pager_image, null);

        ImageView imageView = (ImageView)
imageLayout.findViewById(R.id.image);
        final ProgressBar spinner = (ProgressBar)
imageLayout.findViewById(R.id.loading);

        // 加载图片
        imageLoader.displayImage(Constants.IMAGES[position], imageView,
options, new SimpleImageLoadingListener() {
            @Override
            public void onLoadingStarted(String imageUri, View view) {
                // 显示加载进度条
                spinner.setVisibility(View.VISIBLE);
            }

            @Override
            public void onLoadingFailed(String imageUri, View view,
FailReason failReason) {
                String message = null;

                // 获取图片失败类型
                switch (failReason.getType()) {
                    case IO_ERROR:          // 文件 I/O 错误
                        message = "Input/Output error";
                        break;

                    case DECODING_ERROR:      // 解码错误
                        message = "Image can't be decoded";
                        break;

                    case NETWORK_DENIED:      // 网络延迟
                        message = "Downloads are denied";
                        break;

                    case OUT_OF_MEMORY:           // 内存不足
                        message = "Out Of Memory error";
                        break;
```

```
                case UNKNOWN:           // 原因不明
                    message = "Unknown error";
                    break;
                }

                Toast.makeText(mContext, message,
Toast.LENGTH_SHORT).show();

                // 隐藏加载进度条
                spinner.setVisibility(View.GONE);
            }

            @Override
            public void onLoadingComplete(String imageUri, View view, Bitmap
loadedImage) {
                // 隐藏加载进度条
                spinner.setVisibility(View.GONE);            // 不显示圆形
进度条
            }
        });

        ((ViewPager) container).addView(imageLayout, 0);      // 将图片增加
到ViewPager

        return imageLayout;
    }

    @Override
    public int getCount() {
        return Constants.IMAGES.length;
    }

    @Override
    public boolean isViewFromObject(View view, Object object) {
        return view.equals(object);
    }
```

5_ImageLoader 内存溢出解决办法

1）减少线程池中线程的个数，在 ImageLoaderConfiguration 中的（.threadPoolSize）中配置，推荐配置 1-5。

2）在 DisplayImageOptions 选项中配置 bitmapConfig 为 Bitmap.Config.RGB_565，因为默认是 ARGB_8888， 使用 RGB_565 会比使用 ARGB_8888 少消耗 2 倍的内存。

3）在 ImageLoaderConfiguration 中配置图片的内存缓存为 memoryCache(new WeakMemoryCache()) 或者不使用内存缓存。

4）在 DisplayImageOptions 选项中设置.imageScaleType(ImageScaleType.IN_SAMPLE_INT)或者 imageScaleType(ImageScaleType.EXACTLY)。

第 7 讲_xUtils3 联网模块

谷粉第 47 群：285047793

本文档已经录制成视频，可以到尚硅谷官网下载：
下载地址：http://atguigu.com/download.shtml#kj

xUtils3 联网模块

在做 Android 软件中，联网请求网络是必不可少的功能，xUtils3 封装了多种网络相关功能，网络请求文本，并且可以使用 Get 或者 Post 请求文本；还支持超过 2G 大文件的下载，最满意功能是还支持断点续传，什么是断点续传呢，就是下载一半文件后，接着原来下载的部分接着下载文件，这样不至于重新下载，节约用户流量。
xUtils3 还支持文件上传，上传也支持大于 2G 的文件，另外录制的视频可以使用 xUtils3 上传视频，用户头像也可以 xUtils3 上传。

1_xUtils3NetActivity 布局文件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <!--标题栏-->
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:background="@android:color/holo_blue_light"
        android:gravity="center"
        android:orientation="horizontal">
        <TextView
            android:text="xUtils3 网络模块"
            android:id="@+id/tv_title"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="20sp" />
    </LinearLayout>
    <Button
        android:id="@+id/btn_get_post"
        android:text="get 和 pos 请求"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/btn_downloadfile"
        android:text="大文件下载"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/btn_uploadfile"
        android:text="文件上传"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/tv_result"
        android:text="显示结果"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

1.1 实例化布局

```
/**
 * 作者：尚硅谷-杨光福 on 2016/9/5 11:36
 * 微信：yangguangfu520
 * QQ 号：541433511
```

```
 * 作用：网络模块
 */
@ContentView(R.layout.activity_net)
public class xUtils3NetActivity extends Activity{



    @ViewInject(R.id.tv_result)
    private TextView textView;

    @ViewInject(R.id.progressBar)
    private ProgressBar progressBar;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        x.view().inject(this);
    }

    @Event(value =
{R.id.btn_get_post,R.id.btn_downloadfile,R.id.btn_uploadfile})
    private void getEvent(View view){
        switch (view.getId()){
            case R.id.btn_get_post:
                getDataByGet_Post();
                break;
            case R.id.btn_downloadfile:
                downloadfile();
                break;
            case R.id.btn_uploadfile:
                uploadfile();
                break;
        }

    }

   }
```

2_使用 xUtils3 的 Get 请求文本

Get 请求，成功后会有回调，本方法是最常用的请求文本方式

```
private void getDataByGet_Post() {
    RequestParams request = new
RequestParams("http://api.m.mtime.cn/PageSubArea/TrailerList.api");
    x.http().get(request, new Callback.CommonCallback<String>() {

        @Override
        public void onSuccess(String result) {
            LogUtil.e("onSuccess-result==" + result);
            Toast.makeText(xUtils3NetActivity.this, "result==" + result,
Toast.LENGTH_SHORT).show();
            textView.setText(result+"");
        }

        @Override
        public void onError(Throwable ex, boolean isOnCallback) {
            LogUtil.e("onError==" + ex.getMessage());
        }

        @Override
        public void onCancelled(CancelledException cex) {
            LogUtil.e("onCancelled==" + cex.getMessage());
        }

        @Override
        public void onFinished() {
            LogUtil.e("onFinished==");
        }
    });
}
```

3_使用 xUtils3 的 Post 请求文本

```
private void getDataByGet_Post() {
    RequestParams request = new
RequestParams("http://api.m.mtime.cn/PageSubArea/TrailerList.api");
    x.http().post(request, new Callback.CommonCallback<String>() {

        @Override
        public void onSuccess(String result) {
```

```
            LogUtil.e("onSuccess-result==" + result);
            Toast.makeText(xUtils3NetActivity.this, "result==" + result,
Toast.LENGTH_SHORT).show();
            textView.setText(result+"");
        }

        @Override
        public void onError(Throwable ex, boolean isOnCallback) {
            LogUtil.e("onError==" + ex.getMessage());
        }

        @Override
        public void onCancelled(CancelledException cex) {
            LogUtil.e("onCancelled==" + cex.getMessage());
        }

        @Override
        public void onFinished() {
            LogUtil.e("onFinished==");
        }
    });
}
```

4_使用 xUtils3 文件下载&断点续传

xUtils3 支持大于 2G 的文件的下载，最惊喜的是还支持断点续传下载。

1_下载文件代码

```
/**
 * 下载文件
 */
private void downloadfile() {
    final RequestParams reqest = new
RequestParams("http://vfx.mtime.cn/Video/2016/09/02/mp4/160902093947207
```

```
009_480.mp4");
    reqest.setAutoRename(false);//设置是否根据头信息自动命名文件
    reqest.setSaveFilePath(Environment.getExternalStorageDirectory() +
"/atguigu/oppo.mp4");
    reqest.setExecutor(new PriorityExecutor(3, true));//自定义线程池,有效
的值范围[1, 3], 设置为3时, 可能阻塞图片加载.
    reqest.setCancelFast(true);//是否可以被立即停止.

    x.http().get(reqest, new Callback.ProgressCallback<File>() {
        @Override
        public void onSuccess(File result) {

            LogUtil.e("onSuccess-下载文件成功"+result.toString());
        }

        @Override
        public void onError(Throwable ex, boolean isOnCallback) {
            LogUtil.e("onError-下载文件失败"+ex.getMessage());
        }

        @Override
        public void onCancelled(CancelledException cex) {
            LogUtil.e("onCancelled-");
        }

        @Override
        public void onFinished() {
            LogUtil.e("onFinished-");
        }

        @Override
        public void onWaiting() {
            LogUtil.e("onWaiting-");
        }

        @Override
        public void onStarted() {
            LogUtil.e("onStarted-");
        }

        @Override
        public void onLoading(long total, long current, boolean
isDownloading) {
            progressBar.setMax((int) total);
```

```
            progressBar.setProgress((int) current);
            LogUtil.e("onLoading-"+current+"/"+total);

        }
    });

}
```

2_记得加权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

3_设置断点续传

下面红色的地方就是设置支持断点续传的地方

```
/**
 * 断点下载文件
 */
private void downloadfile() {
    final RequestParams reqest = new
RequestParams("http://vfx.mtime.cn/Video/2016/09/02/mp4/160902093947207
009_480.mp4");
    reqest.setAutoRename(false);//设置是否根据头信息自动命名文件
    reqest.setSaveFilePath(Environment.getExternalStorageDirectory() +
"/atguigu/oppo.mp4");
    reqest.setAutoResume(true);//设置是否在下载是自动断点续传
    reqest.setExecutor(new PriorityExecutor(3, true));//自定义线程池,有效
的值范围[1, 3], 设置为3时, 可能阻塞图片加载.
    reqest.setCancelFast(true);//是否可以被立即停止.

    x.http().get(reqest, new Callback.ProgressCallback<File>() {
        @Override
        public void onSuccess(File result) {

            LogUtil.e("onSuccess-下载文件成功"+result.toString());
        }

        @Override
```

```
        public void onError(Throwable ex, boolean isOnCallback) {
            LogUtil.e("onError-下载文件失败"+ex.getMessage());
        }

        @Override
        public void onCancelled(CancelledException cex) {
            LogUtil.e("onCancelled-");
        }

        @Override
        public void onFinished() {
            LogUtil.e("onFinished-");
        }

        @Override
        public void onWaiting() {
            LogUtil.e("onWaiting-");
        }

        @Override
        public void onStarted() {
            LogUtil.e("onStarted-");
        }

        @Override
        public void onLoading(long total, long current, boolean
isDownloading) {
            progressBar.setMax((int) total);
            progressBar.setProgress((int) current);
            LogUtil.e("onLoading-"+current+"/"+total);

        }
    });

}
```

5_大文件上传

1_支持文件上传服务器的搭建

使用 tomcat 搭建服务，支持文件上传

2_文件上传

文件上传的时候也设置，上传的进度，也需要重写 `ProgressCallback` 这个接口

```
/**
 * 文件上传
 */
private void uploadfile() {
    RequestParams params = new
RequestParams("http://192.168.11.215:8080/FileUpload/FileUploadServlet"
);
    //// 使用multipart 表单上传文件
    params.setMultipart(true);
    params.addBodyParameter("file", new
File(Environment.getExternalStorageDirectory() + "/atguigu/oppo.mp4"),
null, "oppo.mp4");
    x.http().post(params, new Callback.ProgressCallback<File>() {
        @Override
        public void onSuccess(File result) {

            LogUtil.e("onSuccess-上传文件成功" + result.toString());
        }

        @Override
        public void onError(Throwable ex, boolean isOnCallback) {
            LogUtil.e("onError-下载文件失败" + ex.getMessage());
        }

        @Override
        public void onCancelled(CancelledException cex) {
            LogUtil.e("onCancelled-");
        }
```

```
        @Override
        public void onFinished() {
            LogUtil.e("onFinished-");
        }

        @Override
        public void onWaiting() {
            LogUtil.e("onWaiting-");
        }

        @Override
        public void onStarted() {
            LogUtil.e("onStarted-");
        }

        @Override
        public void onLoading(long total, long current, boolean
isDownloading) {
            progressBar.setMax((int) total);
            progressBar.setProgress((int) current);
            LogUtil.e("onLoading-" + current + "/" + total);

        }
    });
}
```

第 8 讲_xUtils3 图片加载模块

谷粉第 47 群：285047793

本文档已经录制成视频，可以到尚硅谷官网下载：
下载地址：http://atguigu.com/download.shtml#kj

xUtils3 图片加载模块

xUtils3 提供的主要方法如下：

```
 x.image().bind(imageView, url, imageOptions);
// assets file
x.image().bind(imageView, "assets://test.gif", imageOptions);
// local file
x.image().bind(imageView, new
File("/sdcard/test.gif").toURI().toString(), imageOptions);
x.image().bind(imageView, "/sdcard/test.gif", imageOptions);
x.image().bind(imageView, "file:///sdcard/test.gif", imageOptions);
x.image().bind(imageView, "file:/sdcard/test.gif", imageOptions);

x.image().bind(imageView, url, imageOptions, new
Callback.CommonCallback<Drawable>() {...});
x.image().loadDrawable(url, imageOptions, new
Callback.CommonCallback<Drawable>() {...});
x.image().loadFile(url, imageOptions, new
Callback.CommonCallback<File>() {...});
```

可以加载网络图片，加载 sdcard 里面的图片，可以加载当前工程 assets 目录的图片。

1_使用 xUtils3 加载单张图片

使用 xUtils3 加载图片的时候，要设置一些配置，如果内存不足，可以设置图片的宽和高小一些

```
/**
     * 加载单张图片
     */
    private void getImage() {

          imageOptions = new ImageOptions.Builder()
          .setSize(DensityUtil.dip2px(80), DensityUtil.dip2px(80))
             //设置圆角
           .setRadius(DensityUtil.dip2px(5))
//          .setImageScaleType(ImageView.ScaleType.CENTER_CROP)
           .setLoadingDrawableId(R.mipmap.ic_launcher)
            .setFailureDrawableId(R.mipmap.ic_launcher)
            .build();

x.image().bind(iv_icon,"http://img31.mtime.cn/mg/2016/09/02/113643.5194
1003.jpg",imageOptions);
    }
```

2_使用 xUtils3 加载 gif 图片

设置加载 gif 图片一定要设置 setIgnoreGif(false)为 false,还可以加载本地的 gif 图片。

```
/**
     * 加载单张图片
     */
    private void getImage() {

         imageOptions = new ImageOptions.Builder()
         .setSize(DensityUtil.dip2px(80), DensityUtil.dip2px(80))
          //设置圆角
         .setRadius(DensityUtil.dip2px(5))
         .setIgnoreGif(false)//是否忽略gif图。false表示不忽略。不写这句,
默认是true
          .setImageScaleType(ImageView.ScaleType.CENTER_CROP)
           .setLoadingDrawableId(R.mipmap.ic_launcher)
           .setFailureDrawableId(R.mipmap.ic_launcher)
           .build();
//
x.image().bind(iv_icon,"http://img31.mtime.cn/mg/2016/09/02/113643.5194
1003.jpg",imageOptions);
```

```
x.image().bind(iv_icon,"http://image82.360doc.com/DownloadImg/2015/02/1621/50253472_10.gif",imageOptions);
//        x.image().bind(iv_icon, Environment.getExternalStorageDirectory()+"/test.gif", imageOptions);
//        x.image().bind(iv_icon, "assets://test.gif", imageOptions);
//        x.image().bind(iv_icon, "file:///sdcard/test.gif", imageOptions);
    }
```

3_使用 xUtils3 在列表中加载图片

在列表中加载图片，和加载单张图片类似，也要配置一下即可。
主要是在适配器的 getView 中加载图片

第 9 讲_xUtils3 数据库模块

谷粉第 47 群：285047793

本文档已经录制成视频，可以到尚硅谷官网下载：
下载地址：http://atguigu.com/download.shtml#kj

xUtils3 数据库模块

1.开源地址

https://github.com/wyouflf/xUtils3/tree/master

2.导入工程

使用 Gradle 构建时添加一下依赖即可:

```
compile 'org.xutils:xutils:3.3.36'
```

3.使用步骤

1. 使用 XUtils3 的一系列初始化。
2. 创建一个 javaBean 类，该类即是需要存储的数据，在数据库中的表。
3. 调用 x.getDb 方法获取 DbManager 对象
4. 使用 DbManager 对象进行 CRUD.

初始化 XUtils3

在 Application 的 onCreate 方法中加入如下代码

```
    //xUtils3 初始化
    x.Ext.init(this);
    //是否打开 log
    x.Ext.setDebug(true);
```

添加权限

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
```

建立数据库的表（JavaBean）

1.在类名上面加入@Table 标签，标签里面的属性 name 的值就是以后生成的数据库的表的名字

2.实体 bean 里面的属性需要加上@Column 标签，这样这个标签的 name 属性的值会对应数据库里面的表的字段。

3.实体 bean 里面的普通属性，如果没有加上@Column 标签就不会在生成表的时候在表里面加入字段。

4.实体 bean 中必须有一个主键，如果没有主键，表以后不会创建成功，@Column(name=”id”,isId=true,autoGen=true)这个属性 name 的值代表的是表的主键的标识，isId 这个属性代表的是该属性是不是表的主键，autoGen 代表的是主键是否是自增长，如果不写 autoGen 这个属性，默认是自增长的属性。

```
/**
 * Created by Administrator on 2016/7/8.
 * 数据库存储 表 的实体类
 * 在类名上加上@Table 标签，标签里面的 name 就是以后生成数据库的表的表名。
 * 实体 bean 里面的属性，如果没有加上@Column，在生成表的时候，就不会在表里面加上该字段，反之亦然。
 */
@Table(name = "Parent")
public class Parent {
    //id
    @Column(name = "id", isId = true, autoGen = true)
    private int id;

    //姓名
    @Column(name = "name")
```

```
private String name;


//年龄
@Column(name = "age")
private int age;


//性别
@Column(name = "sex")
private String sex;


//工资
@Column(name = "salary")
private String salary;


public int getId() {
    return id;
}


public void setId(int id) {
    this.id = id;
}
```

```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public String getSex() {
    return sex;
}

public void setSex(String sex) {
    this.sex = sex;
```

```
    }

    public String getSalary() {
        return salary;
    }

    public void setSalary(String salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "[id=" + getId() +",name=" + getName() + ",age=" + getAge()
                + ",sex=" + getSex() + ",salary=" + getSalary() +"}\n";
    }
}
```

获取 DBManager 对象

```
DbManager db = x.getDb(DaoConfig daoConfig);
```

获取 DbManager 需要一个 daoConfig 对象。

注意：数据库里面表的创建的时间，只有在你对数据库里面的操作涉及到这张表的操作时，会先判断当前的表是否存在，如果不存在，才会创建一张表，如果存在，才会进行相应的 CRUD 操作，但是只要我们想进行一张表的 CRUD 操作，我们必须先执行上面的步骤，通俗点说就是必须拿到一个 Dbmanger 这个对象。

daoCinfig 对象主要用于对于数据库的一些初始化设置

1.setDbName 设置数据库的名称

2.setDbDir 设置数据库存放的路径

3.setDbVersion 设置数据库的版本

4.setAllowTransaction(true) 设置允许开启事务

5.setDbUpgradeListener 设置一个版本升级的监听方法

```
public class DbConfig {
    private static DbManager.DaoConfig daoConfig;

    public static DbManager.DaoConfig getDaoConfig(){
        File file = new File(Environment.getExternalStorageDirectory().getPath());

        if(daoConfig == null){
            daoConfig = new DbManager.DaoConfig()
                    .setDbName("xiaoxiao.db")    //设置数据库的名字
//                      .setDbDir(file)            //设置数据库存储的位置
                    .setDbVersion(1)             //设置数据的版本号
                    .setAllowTransaction(true)   //设置是否允许开启事务
                    .setDbUpgradeListener(new DbManager.DbUpgradeListener() {//
设置一个数据库版本升级的监听
                        @Override
                        public void onUpgrade(DbManager db, int oldVersion, int
newVersion) {

                        }
                    });
```

```
        }
        return daoConfig;
    }
}
```

通过 DbManager 这个类我们知道主要它做了以下几件事情:

1.getDaoConfig 获取数据库的配置信息

2.getDatabase 获取数据库实例

3.saveBindingId saveOrUpdate save 插入数据的 3 个方法(保存数据)

4.replace 只有存在唯一索引时才有用 慎重

5.delete 操作的 4 种方法(删除数据)

6.update 操作的 2 种方法(修改数据)

7.find 操作 6 种方法(查询数据)

8.dropTable 删除表

9.addColumn 添加一列

10.dropDb 删除数据库

添加数据

插入数据有三个方法

Save 插入数据

saveOrUpdate 插入数据，与 save 的区别:当一个实体里面的主键一样时如果使用 saveOrUpdate 会将当前主键对应的这条数据进行替换，而如果你使用了 save 就会报错。

saveBindingId 使用 saveBindingId 保存实体时会为实体的 id 赋值

```
/**
 * 存储 Parent 到数据库
 * @param parent 需要存储的对象
 * @return
 */
public boolean insertParent(Parent parent) {
    try {
        return dbManager.saveBindingId(parent);
    } catch (DbException e) {
        e.printStackTrace();
    }
    return false;
}
```

查找数据

1.findById

通过主键的值来查找表里的数据

```
/**
 * 根据数据的唯一标识 ID 查找数据
 * @param entityType    需要查询的表
 * @param id
 * @param <T>
 * @return
 */
public <T> T findByIdParent(Class<T> entityType,String id){
    try {
        return dbManager.findById(entityType, id);
    } catch (DbException e) {
        e.printStackTrace();
    }
    return null;
}
```

2.findFirst

返回表里的第一条数据

```
/**
 * 返回给定表的第一条数据
 * @param entityType
 * @param <T>
 * @return
 */
public <T> T findFirstParent(Class<T> entityType){
    try {
        return dbManager.findFirst(entityType);
    } catch (DbException e) {
        e.printStackTrace();
    }
    return null;
}
```

3.findAll

返回给定表的所有数据

```
/**
 * 查询给定表的所有数据
 * @param entityType
 * @param <T>
```

```
 * @return
 */
public <T> List<T> findAllParent(Class<T> entityType){
    try {
        return dbManager.findAll(entityType);
    } catch (DbException e) {
        e.printStackTrace();
    }
    return null;
}
```

4.Selector

返回指定条件查询出的数据

```
/**
 * 根据 age 和 sex 查询所有符合条件的 Patent
 * @param entity
 * @param Age
 * @param sex
 * @param <T>
 * @return
 */
```

```
public <T> List<T> selectByAgeAndSex(Class<T> entity, int Age, String sex){
    try {
        return
dbManager.selector(entity).where("age",">",Age).and("sex","=",sex).findAll();
    } catch (DbException e) {
        e.printStackTrace();
    }
    return null;
}
```

下面两个方法主要使用 sql 语句查询数据：

该方法返回一个 DbModel 对象，该对象以 hashMap 的方式存储查询结构，即：key 为表的字段，value 为记录的值　　例：　　key:name　value:小丽

下面是 DbModel 的部分源码：

5.findDbModelFirst

返回 sql 语句查询到的第一条结果

```
/**
 * 根据给定 sql 语句返回查询到的第一条数据
 * @param sql sql 语句
 * @return
 */
public DbModel findDbModelFirst(String sql){
    try {
        return dbManager.findDbModelFirst(new SqlInfo(sql));
    } catch (DbException e) {
        e.printStackTrace();
    }
    return null;
}
```

```
DbModel dbModel = dbHelper.findDbModelFirst("select * from Parent");
if(dbModel != null){
    mTextView.setText(dbModel.getString("name"));
}
```

6.findDbModelAll

返回 sql 语句查询到的所有结果

```
/**
 * 根据给定 sql 语句返回查询到的所有数据
 * @param sql
 * @return
 */
public List<DbModel> findDbModelAll(String sql){
    try {
        return dbManager.findDbModelAll(new SqlInfo(sql));
    } catch (DbException e) {
        e.printStackTrace();
    }
    return null;
}
```

```
List<DbModel> dbModelList = dbHelper.findDbModelAll("select * from Parent
where age > 24");
StringBuilder builder = new StringBuilder();
if(dbModelList != null){
    for (DbModel dbModel : dbModelList){
```

```
        builder.append(dbModel.getString("name") + "\n");
    }
    mTextView.setText("" + builder.toString());
}
```

输出结果：

修改操作

1.修改单条数据的某个字段或多个字段。

Update 第一个参数是需要修改的实体

第二个参数是一个可变参数，是你需要修改的字段的 name。

```
/**
 * 修改第一条数据的值
 */
public void update(){
    //将 id 为 1 的这条记录的数据的 age 修改为 30,name 修改为 “张三丰”
    try {
        Parent parent = dbManager.findById(Parent.class, 1);
        parent.setAge(30);
        parent.setName("张三丰");
        dbManager.update(parent,"name","age");
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

2.修改符合条件的数据的对应字段。

keyValue 是一个键值对对象，第一个参数为字段名，第二个参数为需要修改成的值

whereBuilder.b 为需要修改数据的条件，符合的数据都会被修改。

```
public void update2(){
    //将 parent 表中的性别为 man 的工资 salasy 都变为 6000
```

```
    KeyValue keyValue = new KeyValue("salary","6000");
    try {
        dbManager.update(Parent.class,
WhereBuilder.b("sex","=","men"),keyValue);
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

结果

删除操作

1.deleteById

根据指定的主键来删除对应的一条数据

```
/**
 * 根据主键进行单条数据的删除
```

```
 * @param entity
 * @param id
 */
public void deleteById(Class entity, String id){
    try {
        dbManager.deleteById(entity, id);
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

2.delete(Object entity)

根据具体的实体对象来删除数据

```
/**
 *根据实体 bean 来删除数据
 */
public void deleteObject(){
    try {
        Parent parent = dbManager.findFirst(Parent.class);
        dbManager.delete(parent);
```

```
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

3.delete(Class<?> entityType)

删除指定表的所有数据，注意：删除的是数据，表还是存在的。

```
/**
 * 删除表里的所有数据
 * 注意：表还是会存在，只是数据没了
 */
public void deleteTableAllTate(){
    try {
        dbManager.delete(Parent.class);
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

4.delete(Class<?> entityType, WhereBuilder whereBuilder)

根据 where 语句删除数据

```
/**
 * 根据 where 语句进行删除操作
 */
public void delteWhere(){
    //删除 sex=men,    salary=6000 的数据
    try {
        dbManager.delete(Parent.class,
                WhereBuilder.b("sex","=","men").and("salary","=","6000"));
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

5.dropTable(Class<?> entityType)

删除表

```
/**
 * 删除指定的表
```

```
 */
public void dropTable(){
    try {
        dbManager.dropTable(Parent.class);
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

6.dropDb()

删除数据库

```
/**
 * 删除数据库
 */
public void dropDb(){
    try {
        dbManager.dropDb();
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

给表增加字段

需求：我们需要在 parent 表中添加一个 country 字段。

步骤：

1.在 parent 类中添加 country 属性

2.调用 dbManager.addColumn(Parent.class,"country");方法添加字段

3.在 DbManager.DaoConfig()中的版本号+1，是否必须未知。

```
//国家
@Column(name = "country",property = "中国")
private String country = "";
```

```
/**
 * 添加 country 字段
 */
public void addColumn(){
    try {
```

```
        dbManager.addColumn(Parent.class,"country");
    } catch (DbException e) {
        e.printStackTrace();
    }
}
```

```
public synchronized static DbHelper getInstance(){
    if(dbManager == null){
        dbHelper = new DbHelper();
    }
    if(dbManager == null){
        DbManager.DaoConfig daoConfig = DbConfig.getDaoConfig();
        dbManager = x.getDb(daoConfig);
    }
    dbHelper.addColumn();
    return dbHelper;
}
```

第 14 讲_RecyclerView

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

1_创建 RecyclerViewDemo

在 build.gradle 配置 RecyclerView 的库

```
compile 'com.android.support:recyclerview-v7:23.3.0'
```

2_在布局文件写定义 RecyclerView

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".HelloRecyclerView">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
      />
</RelativeLayout>
```

3_在代码中实例化 RecyclerView 并且初始化数据

```
/**
 * 数据集合
```

```
 */
private ArrayList<String> datas;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    recyclerview = (RecyclerView) findViewById(R.id.recyclerview);
    initData();


}

/**
 * 初始化数据
 */
private void initData() {
    datas = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        datas.add("Content" + i);
    }
}
```

4_设置适配器（难点重点）

1.定义适配器

```
public class MyAdapter extends
RecyclerView.Adapter<MyAdapter.MyViewHolder> {

    private final Context context;
    private final ArrayList<String> datas;


    public MyAdapter(Context context,ArrayList<String> datas){
        this.context = context;
        this.datas = datas;
    }
    /**
     * 相当于ListView适配器中的getView的创建holder布局
     *
```

```
     * @param parent
     * @param viewType
     * @return
     */
    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType)
{
        View view = View.inflate(context, R.layout.item_hello, null);
        return new MyViewHolder(view);
    }

    @Override
    public void onBindViewHolder(MyViewHolder holder, int position) {
        holder.tv_text.setText(datas.get(position));
        holder.iv_icon.setBackgroundResource(R.mipmap.ic_launcher);

    }

    @Override
    public int getItemCount() {
        return datas.size();
    }

    class MyViewHolder extends RecyclerView.ViewHolder {

        private TextView tv_text;
        private ImageView iv_icon;

        public MyViewHolder(View itemView) {
            super(itemView);
            tv_text = (TextView) itemView.findViewById(R.id.tv_text);
            iv_icon = (ImageView) itemView.findViewById(R.id.iv_icon);

        }
    }
}
```

2.设置布局文件

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#33000000"
    android:padding="2dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffffff"
        android:gravity="center"
        android:orientation="horizontal">

        <ImageView
            android:id="@+id/iv_icon"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="3dp"
            android:src="@drawable/video_default" />

        <TextView
            android:id="@+id/tv_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:padding="3dp"
            android:text="内容"
            android:textColor="#000000"
            android:textSize="15sp" />

    </LinearLayout>

</RelativeLayout>
```

5_设置适配器-水平方向-竖直方向-网格-瀑布流

```
myAdapter = new MyAdapter(this,datas);
recyclerview.setAdapter(myAdapter);
```

```
/**
 *  设置布局：
 * 第一个参数：上下文
 * 第二参数：方向
 * 第三个参数：排序低到高还是高到低显示，false 是低到高显示
 *
 */
LinearLayoutManager linearLayoutManager =  new LinearLayoutManager(this,
LinearLayoutManager.VERTICAL, false);
GridLayoutManager gridLayoutManager =  new GridLayoutManager(this,
3,GridLayoutManager.VERTICAL, false);
StaggeredGridLayoutManager staggeredGridLayoutManager =  new
StaggeredGridLayoutManager(2,StaggeredGridLayoutManager.HORIZONTAL);
recyclerview.setLayoutManager(linearLayoutManager);
```

6_设置分割线

参照网址：http://blog.csdn.net/lmj623565791/article/details/45059587

```
//设置分割线-分割线需要自定义&还可以自定义分割线的样式
//没有提供默认的分割线
recyclerview.addItemDecoration(new DividerListItemDecoration(this,
DividerListItemDecoration.VERTICAL_LIST));
```

```
public class DividerListItemDecoration extends RecyclerView.ItemDecoration
{
    private static final int[] ATTRS = new int[]{
            android.R.attr.listDivider
    };

    public static final int HORIZONTAL_LIST =
LinearLayoutManager.HORIZONTAL;

    public static final int VERTICAL_LIST = LinearLayoutManager.VERTICAL;

    private Drawable mDivider;
```

```
    private int mOrientation;

    public DividerListItemDecoration(Context context, int orientation) {
        final TypedArray a = context.obtainStyledAttributes(ATTRS);
        mDivider = a.getDrawable(0);
        a.recycle();
        setOrientation(orientation);
    }

    public void setOrientation(int orientation) {
        if (orientation != HORIZONTAL_LIST && orientation != VERTICAL_LIST)
{
            throw new IllegalArgumentException("invalid orientation");
        }
        mOrientation = orientation;
    }

    @Override
    public void onDraw(Canvas c, RecyclerView parent) {
//        Log.e("recyclerview - itemdecoration", "onDraw()");

         if (mOrientation == VERTICAL_LIST) {
            drawVertical(c, parent);
        } else {
            drawHorizontal(c, parent);
        }

    }


    public void drawVertical(Canvas c, RecyclerView parent) {
        final int left = parent.getPaddingLeft();
        final int right = parent.getWidth() - parent.getPaddingRight();

        final int childCount = parent.getChildCount();
        for (int i = 0; i < childCount; i++) {
            final View child = parent.getChildAt(i);
            android.support.v7.widget.RecyclerView v = new
android.support.v7.widget.RecyclerView(parent.getContext());
            final RecyclerView.LayoutParams params =
(RecyclerView.LayoutParams) child
                    .getLayoutParams();
```

```
            final int top = child.getBottom() + params.bottomMargin;
            final int bottom = top + mDivider.getIntrinsicHeight();
            mDivider.setBounds(left, top, right, bottom);
            mDivider.draw(c);
        }
    }

    public void drawHorizontal(Canvas c, RecyclerView parent) {
        final int top = parent.getPaddingTop();
        final int bottom = parent.getHeight() - parent.getPaddingBottom();

        final int childCount = parent.getChildCount();
        for (int i = 0; i < childCount; i++) {
            final View child = parent.getChildAt(i);
            final RecyclerView.LayoutParams params =
(RecyclerView.LayoutParams) child
                    .getLayoutParams();
            final int left = child.getRight() + params.rightMargin;
            final int right = left + mDivider.getIntrinsicHeight();
            mDivider.setBounds(left, top, right, bottom);
            mDivider.draw(c);
        }
    }

    @Override
    public void getItemOffsets(Rect outRect, int itemPosition, RecyclerView
parent) {
        if (mOrientation == VERTICAL_LIST) {
            outRect.set(0, 0, 0, mDivider.getIntrinsicHeight());
        } else {
            outRect.set(0, 0, mDivider.getIntrinsicWidth(), 0);
        }
    }
}
```

设置分割线样式

在 styles.xml 样式文件中

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
```

```
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
    <item name="android:listDivider">@drawable/divider_bg</item>
</style>


<!-- Application theme. -->
<style name="listDividerTheme" >
    <item name="android:listDivider">@drawable/divider_bg</item>
</style>
```

在 drawable 目录下的 divider_bg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <gradient
        android:centerColor="#ff00ff00"
        android:endColor="#ff0000ff"
        android:startColor="#ffff0000"
        android:type="linear" />
    <size android:height="2dp"/>

</shape>
```

7_自定义设置 item 的点击事件或者 item 中某个控件的点击事件

RecyclerView 默认是没有点击事件的，需要自定义点击事件
用到知识点：接口，getLayoutPosition()

```
public class MyAdapter extends
RecyclerView.Adapter<MyAdapter.MyViewHolder> {
```

```
    private final Context context;
    private final ArrayList<String> datas;


    //设置点击某个item 的监听
    public interface OnItemClickListener{

         void onItemClick(View view,int position,String content);
    }

    private OnItemClickListener onItemClickListener;
    /**
     * 设置某条的监听
     * @param onItemClickListener
     */
    public void setOnItemClickListener(OnItemClickListener
onItemClickListener) {
        this.onItemClickListener = onItemClickListener;
    }

    //设置点击图片

    //设置点击某个item 的监听
    public interface OnImageViewClickListener{

        void onImageViewClick(View view,int position);
    }

    private OnImageViewClickListener onImageViewClickListener;
    /**
     * 设置监听图片
     * @param onImageViewClickListener
     */
    public void setOnImageViewClickListener(OnImageViewClickListener
onImageViewClickListener) {
        this.onImageViewClickListener = onImageViewClickListener;
    }

  ..................................

    class MyViewHolder extends RecyclerView.ViewHolder {
```

```
        private TextView tv_text;
        private ImageView iv_icon;

        public MyViewHolder(View itemView) {
            super(itemView);
            tv_text = (TextView) itemView.findViewById(R.id.tv_text);
            iv_icon = (ImageView) itemView.findViewById(R.id.iv_icon);

            //设置点击事件
            itemView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    if(onItemClickListener != null){

onItemClickListener.onItemClick(v,getLayoutPosition(),datas.get(getLayo
utPosition()));
                    }
                }
            });

            //设置监听
            iv_icon.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    if(onImageViewClickListener != null){

onImageViewClickListener.onImageViewClick(v,getLayoutPosition());
                    }
                }
            });


        }
    }

}
```

在 Activity 中使用自定义的点击事件

```
//设置点击 item 的点击事件
myAdapter.setOnItemClickListener(new MyAdapter.OnItemClickListener() {
    @Override
```

```
    public void onItemClick(View view, int position, String content) {
        Toast.makeText(RecyclerViewActivity.this,
"content=="+content+",--position=="+position, Toast.LENGTH_SHORT).show();
    }
});


//设置点击某张图片的点击事件
myAdapter.setOnImageViewClickListener(new
MyAdapter.OnImageViewClickListener() {
    @Override
    public void onImageViewClick(View view, int position) {
        Toast.makeText(RecyclerViewActivity.this,
"position=="+position+",view=="+view.toString(),
Toast.LENGTH_SHORT).show();
    }
});
```

8_删除和增加数据

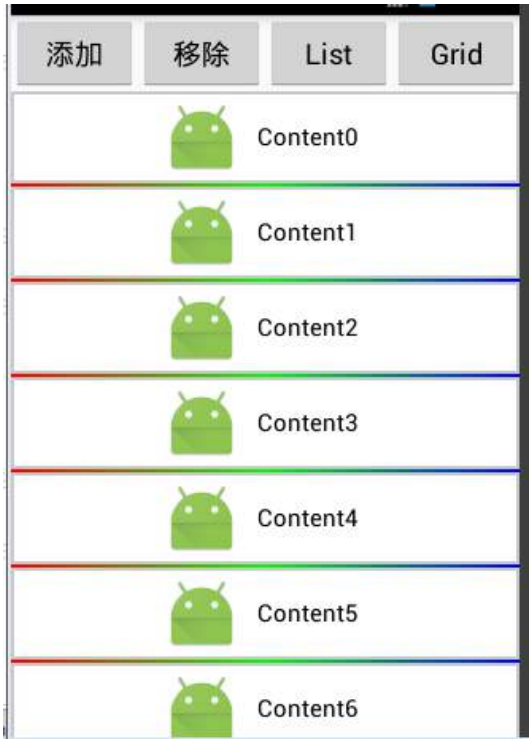
1_在适配器中新增加添加和删除两个方法

```
public class MyAdapter extends
RecyclerView.Adapter<MyAdapter.MyViewHolder> {

     ......................

    /**
     * 添加数据
     * @param position
     * @param content
     */
    public void addData(int position,String content){
        datas.add(position,content);
        notifyItemInserted(position);

    }
```

```
    /**
     * 移除数据
     * @param position
     */
    public void removeData(int position){
        datas.remove(position);
        notifyItemRemoved(position);
    }


}
```

2_在布局文件中添加点击按钮

如图：



代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".RecyclerViewActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/btn_add"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="添加" />

        <Button
            android:id="@+id/btn_remove"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="移除" />

        <Button
            android:id="@+id/btn_list"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="List" />

        <Button
            android:id="@+id/btn_grid"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Grid" />


    </LinearLayout>
```

```
    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

3_点击事件

```
private void setLitener() {
    btn_add.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            myAdapter.addData(0,"Content NetData");
            //定位到第 0 个位置
            recyclerview.scrollToPosition(0);
        }
    });

    btn_remove.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            myAdapter.removeData(0);
        }
    });

    btn_grid.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            GridLayoutManager gridLayoutManager =  new
GridLayoutManager(RecyclerViewActivity.this,
2,GridLayoutManager.VERTICAL, false);
            recyclerview.setLayoutManager(gridLayoutManager);
        }
    });

    btn_list.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
```

```
            LinearLayoutManager linearLayoutManager =  new
LinearLayoutManager(RecyclerViewActivity.this,
LinearLayoutManager.VERTICAL, false);
            recyclerview.setLayoutManager(linearLayoutManager);
        }
    });
}
```

9_设置删除某条和增加某条的动画

没有设置的情况是默认有动画的，也可以自己设置

```
//设置动画
recyclerview.setItemAnimator(new DefaultItemAnimator());
```

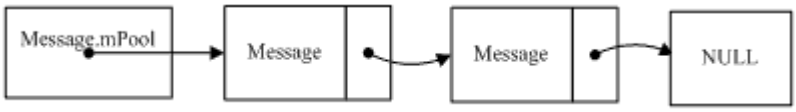
题目：Android 中消息机制分析

1.活用 Android 线程间通信的 Message 机制

1.1.Message ： 可理解为线程间通讯的数据单元, 可通过 message 携带需要的数据
代码在 frameworks\base\core\java\android\Os\Message.java 中。
创建对象: Message.obtain(what) --消息池 (字符串常量池，连接池)
封装数据
public int what //id 标识
public int arg1
public int arg2
public Object obj

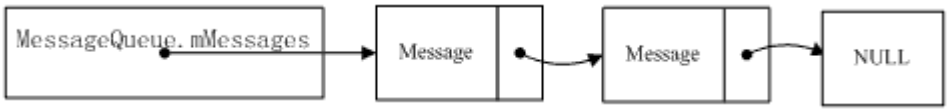
Message.obtain 函数：有多个 obtain 函数，主要功能一样，只是参数不一样。作用是从 Message Pool 中取出一个 Message，如果 Message Pool 中已经没有 Message 可取则新建一个 Message 返回，同时用对应的参数给得到的 Message 对象赋值。

Message Pool：大小为 10 个；通过 Message.mPool->（Message 并且 Message.next）->（Message 并且 Message.next）->（Message 并且 Message.next）...构造一个 Message Pool。Message Pool 的第一个元素直接 new 出来，然后把 Message.mPool（static 类的 static 变量）指向它。其他的元素都是使用完的 Message 通过 Message 的 recycle 函数清理后放到 Message Pool（通过 Message Pool 最后一个 Message 的 next 指向需要回收的 Message 的方式实现）。下图为 Message Pool 的结构：



1.2.MessageQueue
MessageQueue 里面有一个收到的 Message 的队列：

MessageQueue.mMessages(static 变量)->(Message 并且 Message.next)-> (Message 并且 Message.next)->...，下图为接收消息的消息队列：



上层代码通过 Handler 的 sendMessage 等函数放入一个 message 到 MessageQueue 里面时最终会调用 MessageQueue 的 enqueueMessage 函数。enqueueMessage 根据上面的接收的 Message 的队列的构造把接收到的 Message 放入队列中。

MessageQueue 的 removeMessages 函数根据上面的接收的 Message 的队列的构造把接收到的 Message 从队列中删除，并且调用对应 Message 对象的 recycle 函数把不用的 Message 放入 Message Pool 中。

1.3.Looper

Looper 对象的创建是通过 prepare 函数，而且每一个 Looper 对象会和一个线程关联。

Java 代码

```
1.  public static final void prepare() {
2.      if (sThreadLocal.get() != null) {
3.          throw new RuntimeException("Only one Looper may be created per threa
    d");
4.      }
5.      sThreadLocal.set(new Looper());
6.  }
```

Looper 对象创建时会创建一个 MessageQueue，主线程默认会创建一个 Looper 从而有 MessageQueue，其他线程默认是没有 MessageQueue 的不能接收 Message，如果需要接收 Message 则需要通过 prepare 函数创建一个 MessageQueue。具体操作请见示例代码。

Java 代码

```
1.  private Looper() {
2.      mQueue = new MessageQueue();
3.      mRun = true;
4.      mThread = Thread.currentThread();
5.  }
```

prepareMainLooper 函数只给主线程调用（系统处理，程序员不用处理），它会调用 prepare 建立 Looper 对象和 MessageQueue。

Java 代码

```
1.  public static final void prepareMainLooper() {
2.      prepare();
3.      setMainLooper(myLooper());
4.      if (Process.supportsProcesses()) {
5.          myLooper().mQueue.mQuitAllowed = false;
6.      }
7.  }
1.
```

Loop 函数从 MessageQueue 中从前往后取出 Message，然后通过 Handler 的 dispatchMessage 函数进行消息的处理（可见消息的处理是 Handler 负责的），消息处理完了以后通过 Message 对象的 recycle 函数放到 Message Pool 中，以便下次使用，通过 Pool 的处理提供了一定的内存管理从而加速消息对象的获取。至于需要定时处理的消息如何做到定时处理，请见 MessageQueue 的 next 函数，它在取 Message 来进行处理时通过判断 MessageQueue 里面的

Message 是否符合时间要求来决定是否需要把 Message 取出来做处理，通过这种方式做到消息的定时处理。

Java 代码

```
1.  public static final void loop() {
2.      Looper me = myLooper();
3.      MessageQueue queue = me.mQueue;
4.      while (true) {
5.          Message msg = queue.next(); // might block
6.          //if (!me.mRun) {
7.          //    break;
8.          //}
9.          if (msg != null) {
10.             if (msg.target == null) {
11.                 // No target is a magic identifier for the quit message
12.                 return;
13.             }
14.
15.             if (me.mLogging!= null)
16.                 me.mLogging.println(">>>>> Dispatching to " + msg.targe
    t + " "+ msg.callback + ": " + msg.what);
17.             msg.target.dispatchMessage(msg);
18.             if (me.mLogging!= null)
19.                 me.mLogging.println("<<<<< Finished to" + msg.target + " "
    + msg.callback);
20.             msg.recycle();
21.         }
22.     }
23. }
```

1.4.Handler

Handler 的构造函数表示 Handler 会有成员变量指向 Looper 和 MessageQueue，后面我们会看到没什么需要这些引用；至于 callback 是实现了 Callback 接口的对象，后面会看到这个对象的作用。

Java 代码

```
1.  public Handler(Looper looper, Callback callback) {
2.      mLooper = looper;
3.      mQueue = looper.mQueue;
4.      mCallback = callback;
5.  }
6.
7.  public interface Callback {
```

```
8.      public boolean handleMessage(Message msg);
9.  }
```

获取消息：直接通过 Message 的 obtain 方法获取一个 Message 对象。

Java 代码

```
1.  public final Message obtainMessage(int what, int arg1, int arg2, Object obj)
    {
2.      return Message.obtain(this, what, arg1, arg2, obj);
3.  }
```

发送消息：通过 MessageQueue 的 enqueueMessage 把 Message 对象放到 MessageQueue 的接收消息队列中

Java 代码

```
1.  public boolean sendMessageAtTime(Message msg, long uptimeMillis){
2.      boolean sent = false;
3.      MessageQueue queue = mQueue;
4.      if (queue != null) {
5.          msg.target = this;
6.      sent = queue.enqueueMessage(msg, uptimeMillis);
7.      } else {
8.          RuntimeException e = new RuntimeException(this + " sendMessageAtTime
    () called with no mQueue");
9.          Log.w("Looper", e.getMessage(), e);
10.     }
11.     return sent;
12. }
```

线程如何处理 MessageQueue 中接收的消息：在 Looper 的 loop 函数中循环取出 MessageQueue 的接收消息队列中的消息，然后调用 Hander 的 dispatchMessage 函数对消息进行处理，至于如何处理（相应消息）则由用户指定（三个方法，优先级从高到低：Message 里面的 Callback，一个实现了 Runnable 接口的对象，其中 run 函数做处理工作；Handler 里面的 mCallback 指向的一个实现了 Callback 接口的对象，里面的 handleMessage 进行处理；处理消息 Handler 对象对应的类继承并实现了其中 handleMessage 函数，通过这个实现的 handleMessage 函数处理消息）。

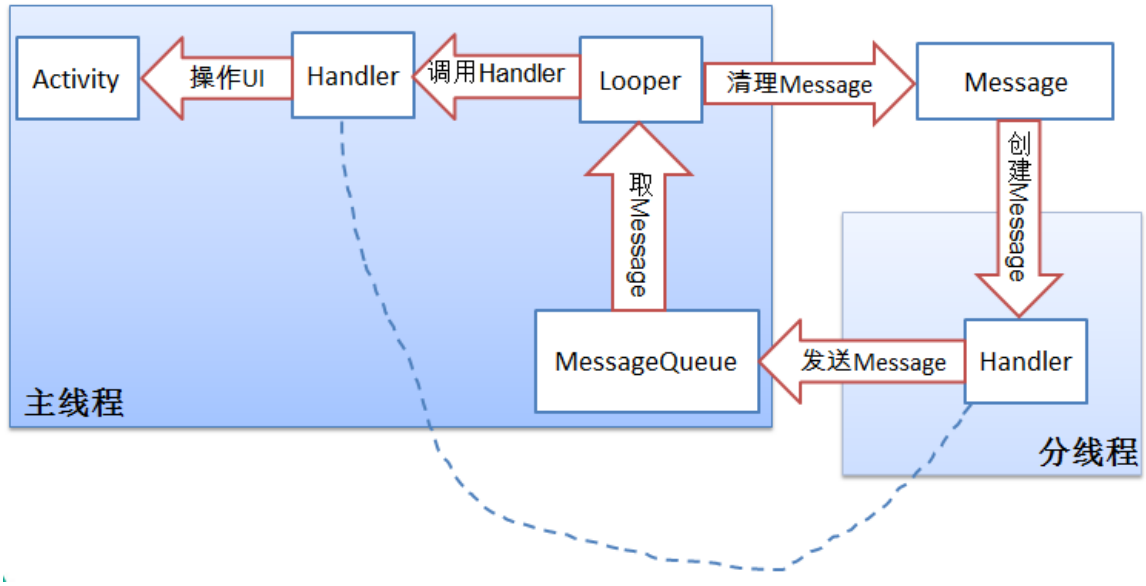
Java 代码

```
1.  public void dispatchMessage(Message msg) {
2.      if (msg.callback != null) {
3.          handleCallback(msg);
4.      } else {
5.          if (mCallback != null) {
```

```
6.              if (mCallback.handleMessage(msg)) {
7.                  return;
8.              }
9.          }
10.         handleMessage(msg);
11.     }
12. }
```

Runnable 说明：Runnable 只是一个接口，实现了这个接口的类对应的对象也只是个普通的对象，并不是一个 Java 中的 Thread。Thread 类经常使用 Runnable，很多人有误解，所以这里澄清一下。

从上可知以下关系图：



其中清理 Message 是 Looper 里面的 loop 函数指把处理过的 Message 放到 Message 的 Pool 里面去，如果里面已经超过最大值 10 个，则丢弃这个 Message 对象。

调用 Handler 是指 Looper 里面的 loop 函数从 MessageQueue 的接收消息队列里面取出消息，然后根据消息指向的 Handler 对象调用其对应的处理方法。

1.5.代码示例

Java 代码

```
1.  package com.android.messageexample;
2.  import android.app.Activity;
3.  import android.content.Context;
4.  import android.graphics.Color;
5.  import android.os.Bundle;
6.  import android.os.Handler;
7.  import android.os.Looper;
```

```
8.  import android.os.Message;
9.  import android.util.Log;
10. import android.view.View;
11. import android.view.View.OnClickListener;
12. import android.widget.Button;
13. import android.widget.LinearLayout;
14. import android.widget.TextView;
15. public class MessageExample extends Activity implements OnClickListener {
16.  private final int WC = LinearLayout.LayoutParams.WRAP_CONTENT;
17.     private final int FP = LinearLayout.LayoutParams.FILL_PARENT;
18.     public TextView tv;
19.     private EventHandler mHandler;
20.     private Handler mOtherThreadHandler=null;
21.     private Button btn, btn2, btn3, btn4, btn5, btn6;
22.     private NoLooperThread noLooerThread = null;
23.     private OwnLooperThread ownLooperThread = null;
24.     private ReceiveMessageThread receiveMessageThread =null;
25.     private Context context = null;
26.     private final String sTag = "MessageExample";
27.     private boolean postRunnable = false;
28.
29.  /** Called when the activity is first created. */
30.  @Override
31.     public void onCreate(Bundle savedInstanceState) {
32.         super.onCreate(savedInstanceState);
33.         context = this.getApplicationContext();
34.         LinearLayout layout = new LinearLayout(this);
35.         layout.setOrientation(LinearLayout.VERTICAL);
36.         btn = new Button(this);
37.         btn.setId(101);
38.         btn.setText("message from main thread self");
39.         btn.setOnClickListener(this);
40.         LinearLayout.LayoutParams param =
41.             new LinearLayout.LayoutParams(250,50);
42.         param.topMargin = 10;
43.         layout.addView(btn, param);
44.         btn2 = new Button(this);
45.         btn2.setId(102);
46.         btn2.setText("message from other thread to main thread");
47.         btn2.setOnClickListener(this);
48.         layout.addView(btn2, param);
49.         btn3 = new Button(this);
50.         btn3.setId(103);
51.         btn3.setText("message to other thread from itself");
```

```
52.         btn3.setOnClickListener(this);
53.         layout.addView(btn3, param);
54.         btn4 = new Button(this);
55.         btn4.setId(104);
56.         btn4.setText("message with Runnable as callback from other thread t
    o main thread");
57.         btn4.setOnClickListener(this);
58.         layout.addView(btn4, param);
59.         btn5 = new Button(this);
60.         btn5.setId(105);
61.         btn5.setText("main thread's message to other thread");
62.         btn5.setOnClickListener(this);
63.         layout.addView(btn5, param);
64.         btn6 = new Button(this);
65.         btn6.setId(106);
66.         btn6.setText("exit");
67.         btn6.setOnClickListener(this);
68.         layout.addView(btn6, param);
69.         tv = new TextView(this);
70.         tv.setTextColor(Color.WHITE);
71.         tv.setText("");
72.         LinearLayout.LayoutParams param2 =
73.            new LinearLayout.LayoutParams(FP, WC);
74.         param2.topMargin = 10;
75.         layout.addView(tv, param2);
76.         setContentView(layout);
77.
78.         //主线程要发送消息给 other thread， 这里创建那个 other thread
79.   receiveMessageThread = new ReceiveMessageThread();
80.   receiveMessageThread.start();
81.     }
82.
83.  //implement the OnClickListener interface
84.  @Override
85.  public void onClick(View v) {
86.   switch(v.getId()){
87.   case 101:
88.    //主线程发送消息给自己
89.    Looper looper;
90.    looper = Looper.myLooper();  //get the Main looper related with the mai
    n thread
91.    //如果不给任何参数的话会用当前线程对应的 Looper(这里就是 Main Looper)为 Handler
    里面的成员 mLooper 赋值
92.    mHandler = new EventHandler(looper);
```

```
93.    //mHandler = new EventHandler();
94.    // 清除整个 MessageQueue 里的消息
95.    mHandler.removeMessages(0);
96.    String obj = "This main thread's message and received by itself!";
97.    //得到 Message 对象
98.    Message m = mHandler.obtainMessage(1, 1, 1, obj);
99.    // 将 Message 对象送入到 main thread 的 MessageQueue 里面
100.    mHandler.sendMessage(m);
101.    break;
102.   case 102:
103.    //other 线程发送消息给主线程
104.    postRunnable = false;
105.    noLooerThread = new NoLooperThread();
106.    noLooerThread.start();
107.    break;
108.   case 103:
109.    //other thread 获取它自己发送的消息
110.    tv.setText("please look at the error level log for other thread receive
    d message");
111.    ownLooperThread = new OwnLooperThread();
112.    ownLooperThread.start();
113.    break;
114.   case 104:
115.    //other thread 通过 Post Runnable 方式发送消息给主线程
116.    postRunnable = true;
117.    noLooerThread = new NoLooperThread();
118.    noLooerThread.start();
119.    break;
120.   case 105:
121.    //主线程发送消息给 other thread
122.    if(null!=mOtherThreadHandler){
123.     tv.setText("please look at the error level log for other thread receive
    d message from main thread");
124.     String msgObj = "message from mainThread";
125.     Message mainThreadMsg = mOtherThreadHandler.obtainMessage(1, 1, 1, msgO
    bj);
126.     mOtherThreadHandler.sendMessage(mainThreadMsg);
127.    }
128.    break;
129.   case 106:
130.    finish();
131.    break;
132.   }
133.  }
```

```
134.  class EventHandler extends Handler
135.  {
136.   public EventHandler(Looper looper) {
137.    super(looper);
138.   }
139.   public EventHandler() {
140.    super();
141.   }
142.   public void handleMessage(Message msg) {
143.    //可以根据 msg.what 执行不同的处理，这里没有这么做
144.    switch(msg.what){
145.    case 1:
146.     tv.setText((String)msg.obj);
147.     break;
148.    case 2:
149.     tv.setText((String)msg.obj);
150.     noLooerThread.stop();
151.     break;
152.    case 3:
153.     //不能在非主线程的线程里面更新 UI，所以这里通过 Log 打印收到的消息
154.     Log.e(sTag, (String)msg.obj);
155.     ownLooperThread.stop();
156.     break;
157.    default:
158.     //不能在非主线程的线程里面更新 UI，所以这里通过 Log 打印收到的消息
159.     Log.e(sTag, (String)msg.obj);
160.     break;
161.    }
162.   }
163.  }
164.  //NoLooperThread
165.  class NoLooperThread extends Thread{
166.   private EventHandler mNoLooperThreadHandler;
167.   public void run() {
168.    Looper myLooper, mainLooper;
169.    myLooper = Looper.myLooper();
170.    mainLooper = Looper.getMainLooper();     //这是一个 static 函数
171.    String obj;
172.    if(myLooper == null){
173.     mNoLooperThreadHandler = new EventHandler(mainLooper);
174.     obj = "NoLooperThread has no looper and handleMessage function execute
    d in main thread!";
175.    }
176.    else {
```

```
177.     mNoLooperThreadHandler = new EventHandler(myLooper);
178.     obj = "This is from NoLooperThread self and handleMessage function exec
    uted in NoLooperThread!";
179.    }
180.    mNoLooperThreadHandler.removeMessages(0);
181.    if(false == postRunnable){
182.     //send message to main thread
183.     Message m = mNoLooperThreadHandler.obtainMessage(2, 1, 1, obj);
184.     mNoLooperThreadHandler.sendMessage(m);
185.     Log.e(sTag, "NoLooperThread id:" + this.getId());
186.    }else{
187.     //下面 new 出来的实现了 Runnable 接口的对象中 run 函数是在 Main Thread 中执行，
    不是在 NoLooperThread 中执行
188.     //注意 Runnable 是一个接口，它里面的 run 函数被执行时不会再新建一个线程
189.     //您可以在 run 上加断点然后在 eclipse 调试中看它在哪个线程中执行
190.     mNoLooperThreadHandler.post(new Runnable(){
191.      @Override
192.      public void run() {
193.       tv.setText("update UI through handler post runnalbe mechanism!");
194.       noLooerThread.stop();
195.      }
196.     });
197.    }
198.   }
199.  }
200.
201.  //OwnLooperThread has his own message queue by execute Looper.prepare();
202.  class OwnLooperThread extends Thread{
203.   private EventHandler mOwnLooperThreadHandler;
204.   public void run() {
205.    Looper.prepare();
206.    Looper myLooper, mainLooper;
207.    myLooper = Looper.myLooper();
208.    mainLooper = Looper.getMainLooper();    //这是一个 static 函数
209.    String obj;
210.    if(myLooper == null){
211.     mOwnLooperThreadHandler = new EventHandler(mainLooper);
212.     obj = "OwnLooperThread has no looper and handleMessage function execute
    d in main thread!";
213.    }
214.    else {
215.     mOwnLooperThreadHandler = new EventHandler(myLooper);
216.     obj = "This is from OwnLooperThread self and handleMessage function exe
    cuted in NoLooperThread!";
```

```
217.     }
218.     mOwnLooperThreadHandler.removeMessages(0);
219.     //给自己发送消息
220.     Message m = mOwnLooperThreadHandler.obtainMessage(3, 1, 1, obj);
221.     mOwnLooperThreadHandler.sendMessage(m);
222.     Looper.loop();
223.    }
224.   }
225.
226.   //ReceiveMessageThread has his own message queue by execute Looper.prepare
    ();
227.   class ReceiveMessageThread extends Thread{
228.    public void run() {
229.     Looper.prepare();
230.     mOtherThreadHandler = new Handler(){
231.      public void handleMessage(Message msg) {
232.       Log.e(sTag, (String)msg.obj);
233.      }
234.     };
235.     Looper.loop();
236.    }
237.   }
238.
239. }
```

使用 Looper.myLooper 静态方法可以取得当前线程的 Looper 对象。

使用 mHandler = new EevntHandler(Looper.myLooper()); 可建立用来处理当前线程的 Handler 对象；其中，EevntHandler 是 Handler 的子类。

1.5.1.主线程给自己发送消息示例

主线程发送消息：

在 onClick 的 case 101 中创建一个继承自 Handler 的 EventHandler 对象，然后获取一个消息，然后通过 EventHandler 对象调用 sendMessage 把消息发送到主线程的 MessageQueue 中。主线程由系统创建，系统会给它建立一个 Looper 对象和 MessageQueue，所以可以接收消息。这里只要根据主线程的 Looper 对象初始化 EventHandler 对象，就可以通过 EventHandler 对象发送消息到主线程的消息队列中。

主线程处理消息：

这里是通过 EventHandler 的 handleMessage 函数处理的，其中收到的 Message 对象中 what

值为一的消息就是发送给它的，然后把消息里面附带的字符串在 TextView 上显示出来。

1.5.2.其他线程给主线程发送消息示例

其他线程发送消息（这里是说不使用 Runnable 作为 callback 的消息）：

首先 postRunnable 设为 false，表示不通过 Runnable 方式进行消息相关的操作。然后启动线程 noLooerThread，然后以主线程的 Looper 对象为参数建立 EventHandler 的对象 mNoLooperThreadHandler，然后获取一个 Message 并把一个字符串赋值给它的一个成员 obj，然后通过 mNoLooperThreadHandler 把消息发送到主线程的 MessageQueue 中。

主线程处理消息：

这里是通过 EventHandler 的 handleMessage 函数处理的，其中收到的 Message 对象中 what 值为二的消息就是上面发送给它的，然后把消息里面附带的字符串在 TextView 上显示出来。

1.5.3.其他线程给自己发送消息示例

其他线程发送消息：

其他非主线程建立后没有自己的 Looper 对象，所以也没有 MessageQueue，需要给非主线程发送消息时需要建立 MessageQueue 以便接收消息。下面说明如何给自己建立 MessageQueue 和 Looper 对象。从 OwnLooperThread 的 run 函数中可以看见有一个 Looper.prepare()调用，这个就是用来建立非主线程的 MessageQueue 和 Looper 对象的。

所以这里的发送消息过程是建立线程 mOwnLooperThread，然后线程建立自己的 Looper 和 MessageQueue 对象，然后根据上面建立的 Looper 对象建立对应的 EventHandler 对象 mOwnLooperThreadHandler，然后由 mOwnLooperThreadHandler 建立消息并且发送到自己的 MessageQueue 里面。

其他线程处理接收的消息：

线程要接收消息需要在 run 函数中调用 Looper.loop()，然后 loop 函数会从 MessageQueue 中取出消息交给对应的 Handler 对象 mOwnLooperThreadHandler 处理，在 mOwnLooperThreadHandler 的 handleMessage 函数中会把 Message 对象中 what 值为三的消息（上面发送的消息）在 Log 中打印出来，可以通过 Logcat 工具查看 log。

1.5.4.其他线程以 Runnable 为消息参数给主线程发送消息示例

其他线程发送消息（这里是说使用 Runnable 作为 callback 的消息）：

首先 postRunnable 设为 true，表示通过 Runnable 方式进行消息相关的操作。然后启动线程 noLooerThread，然后以主线程的 Looper 对象为参数建立 EventHandler 的对象

mNoLooperThreadHandler，然后获取一个 Message 并把一个字符串赋值给它的一个成员 obj，然后通过 mNoLooperThreadHandler 把消息发送到主线程的 MessageQueue 中。

主线程处理消息：

主线程收到上面发送的 Message 后直接运行上面 Runnable 对象中的 run 函数进行相应的操作。run 函数通过 Log 打印一个字符串，可以通过 Logcat 工具查看 log。

1.5.5.主线程给其他线程发送消息示例

主线程发送消息：

这里首先要求线程 receiveMessageThread 运行（在 onCreate 函数中完成），并且准备好自己的 Looper 和 MessageQueue（这个通过 ReceiveMessageThread 中的 run 函数中的 Looper.prepare()调用完成），然后根据建立的 Looper 对象初始化 Handler 对象 mOtherThreadHandler。然后在 onClick 的 case 105 中由 mOtherThreadHandler 建立一个消息（消息中有一个字符串对象）并且发送到线程 receiveMessageThread 中的 MessageQueue 中。

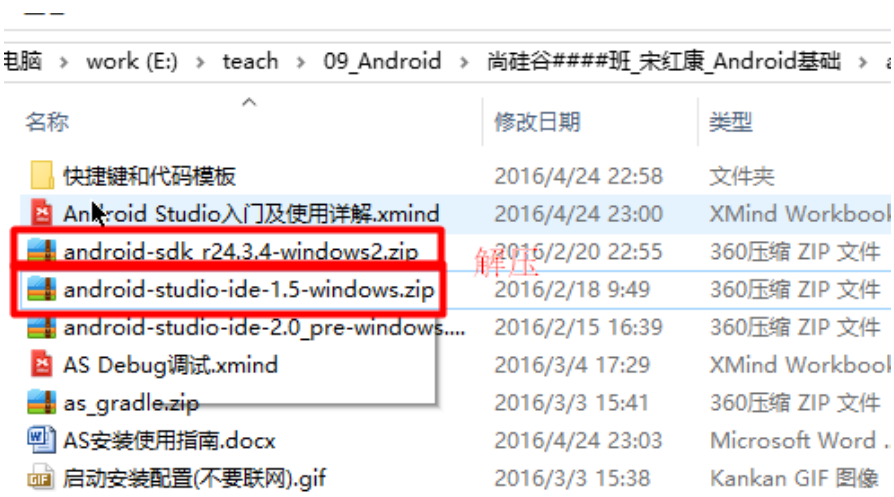
其他线程处理接收的消息：

线程要接收消息需要在 run 函数中调用 Looper.loop()，然后 loop 函数会从 MessageQueue 中取出消息交给对应的 Handler 对象 mOtherThreadHandler 处理，在 mOtherThreadHandler 的 handleMessage 函数中会把 Message 对象中的字符串对象在 Log 中打印出来，可以通过 Logcat 工具查看 log。

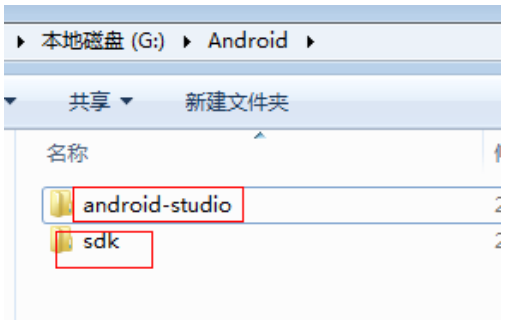
AndroidStudio 使用指南

一.安装准备

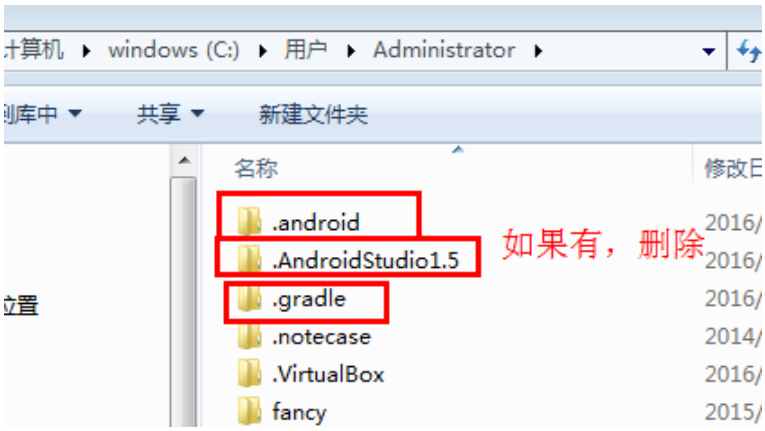
1_解压 zip 包_无中文_无空格



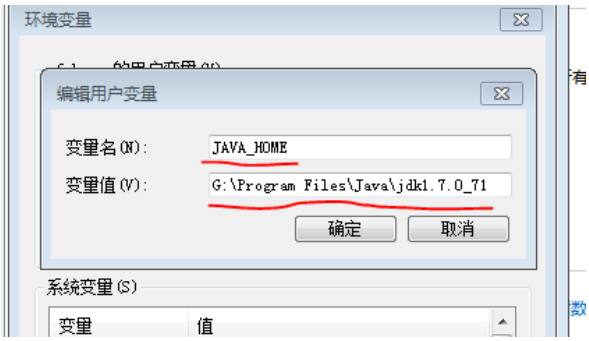
解压到不含中文且没有空格的文件目录下。（比如如下目录）



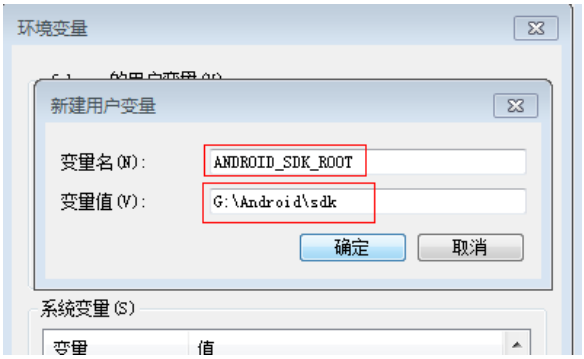
2_删除以前残留的配置文件



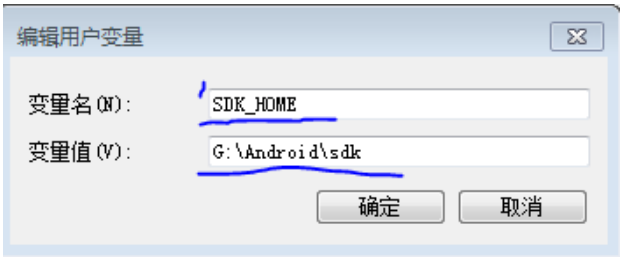
3_配置 JAVA_HOME 环境变量



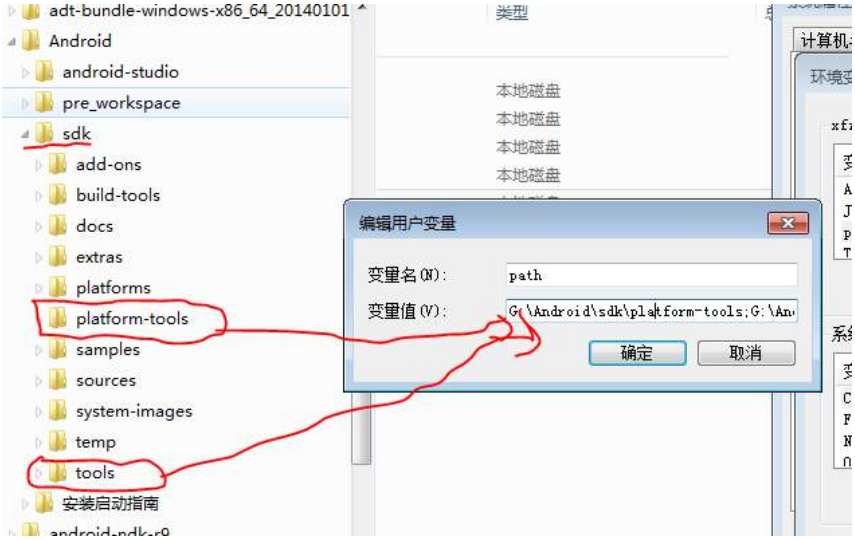
4_配置 ANDROID_SDK_ROOT 环境变量



5_配置 SDK_HOME 环境变量

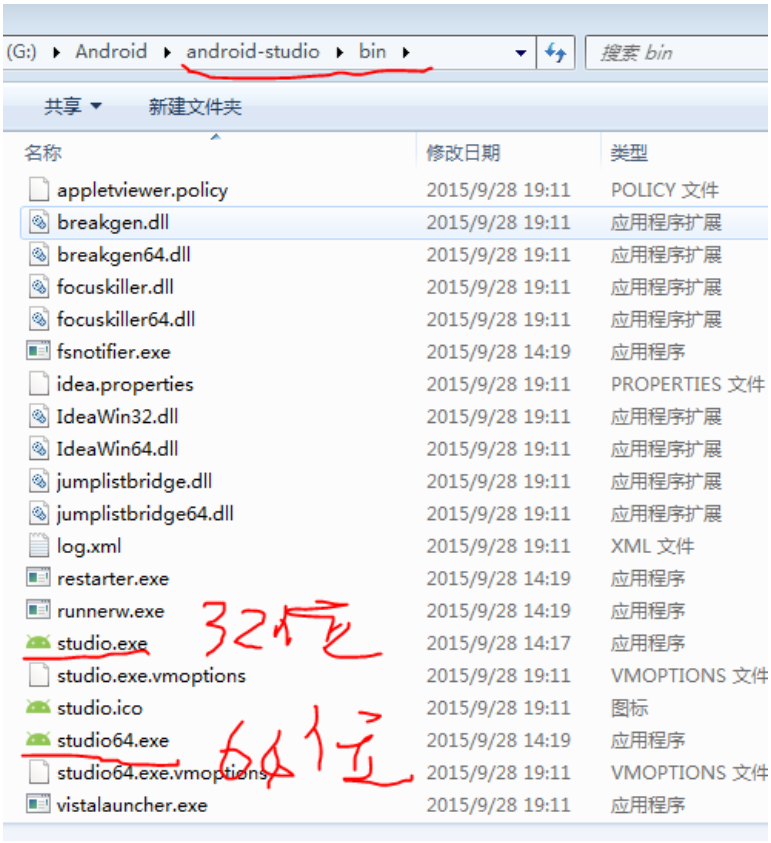


6_将 sdk 的两个工具(platform-tools 和 tools)配置到 path

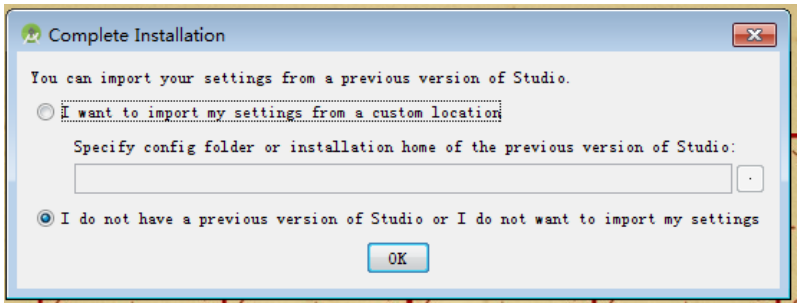


二. 启动安装配置(不要联网)

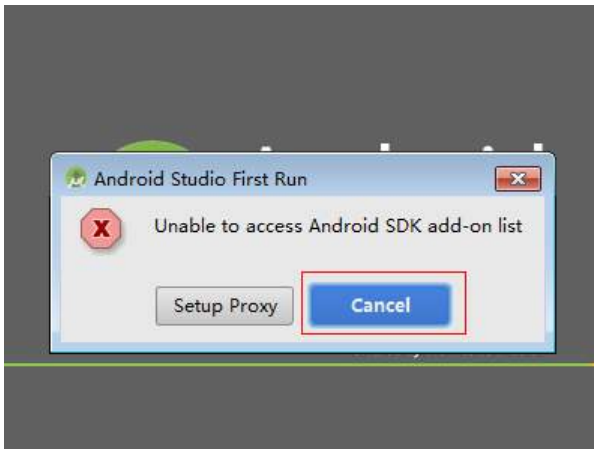
1_启动 AS(第一次)



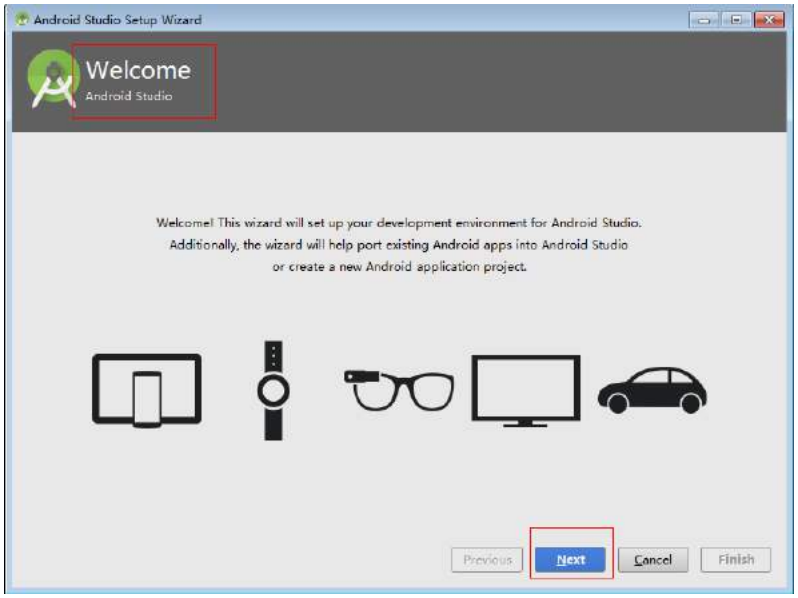
2_是否导入已有的配置(选择 I do not ….)



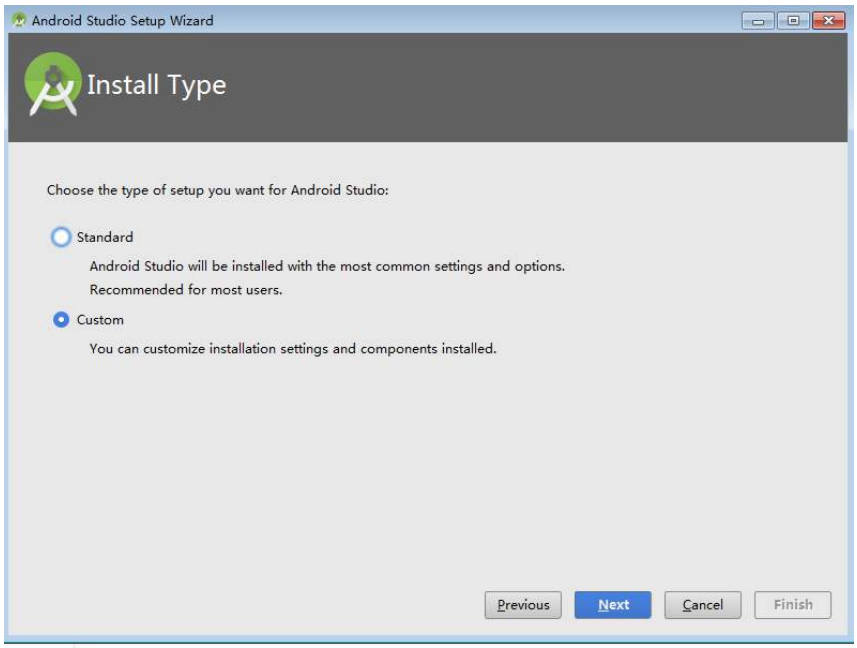
3_提示不能访问远程 SDK(选择取消)



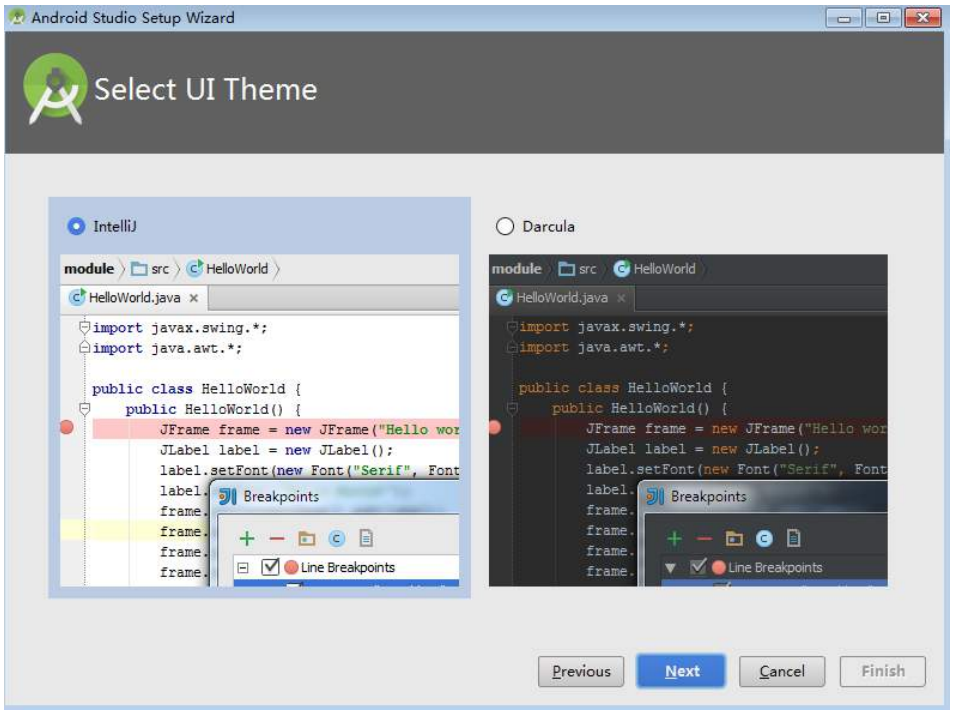
4_安装向导欢迎界面



5_使用哪种安装模式(选择 custom)

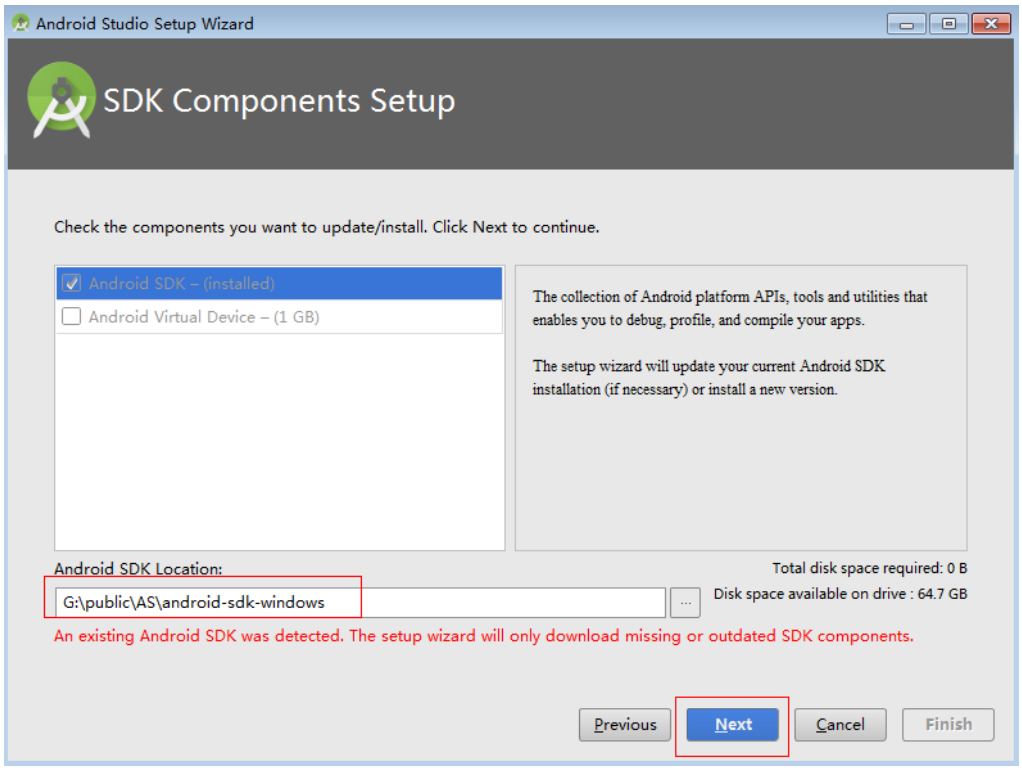


6_使用哪种 UI 主题(选择白色)

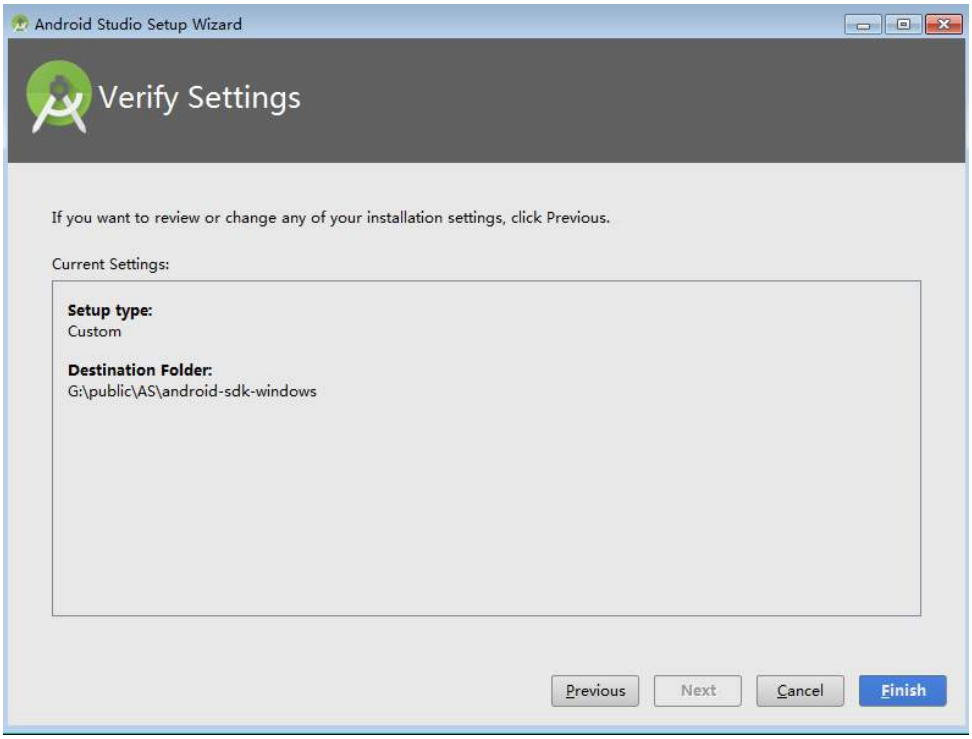


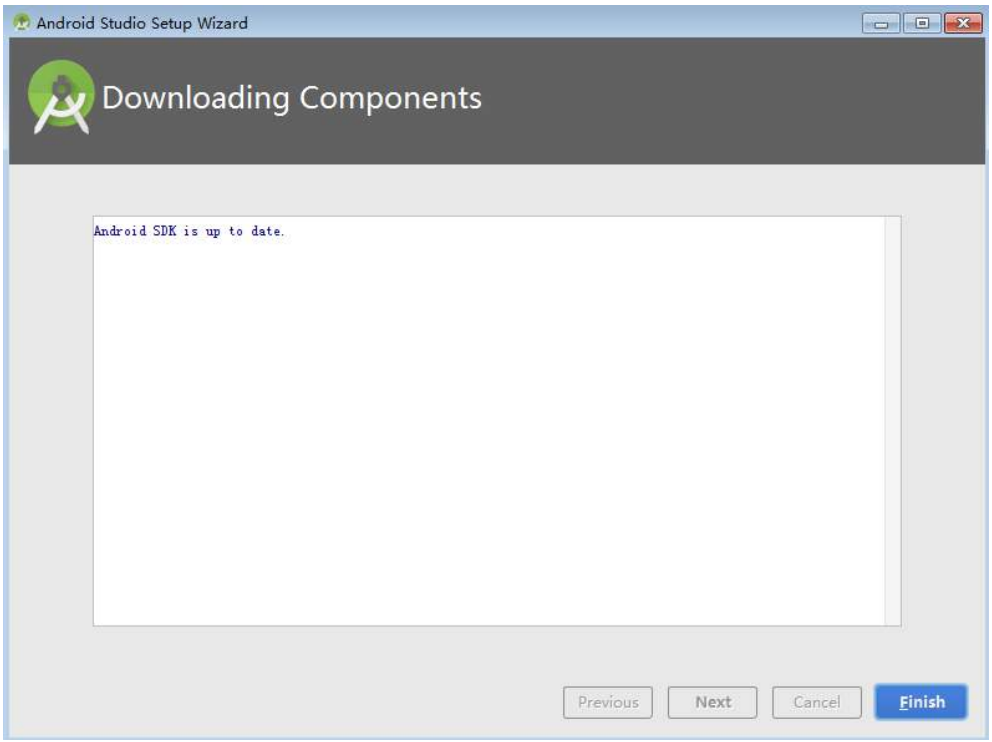
7_设置 sdk 的位置(指定 SDK 根目录)

(这里提示 sdk 目录里面已经有内容了，因为我们已经下载了相应版本的 sdk 了，不用管)



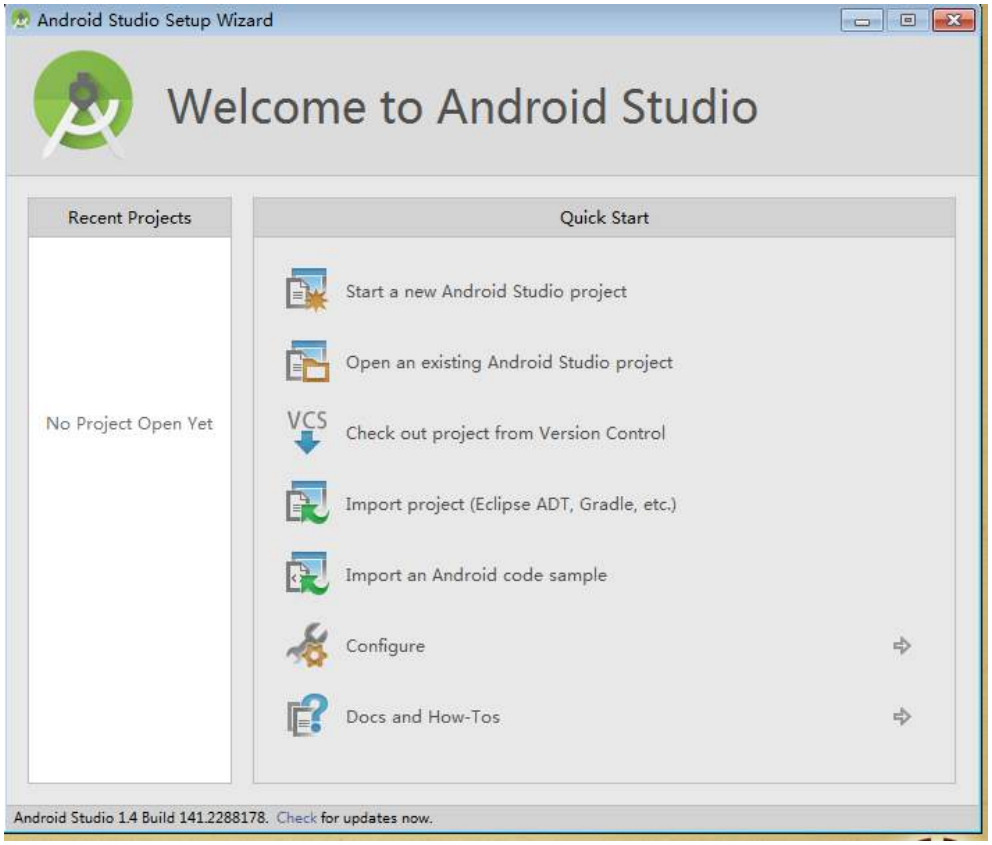
8_确定前面的配置(finish)



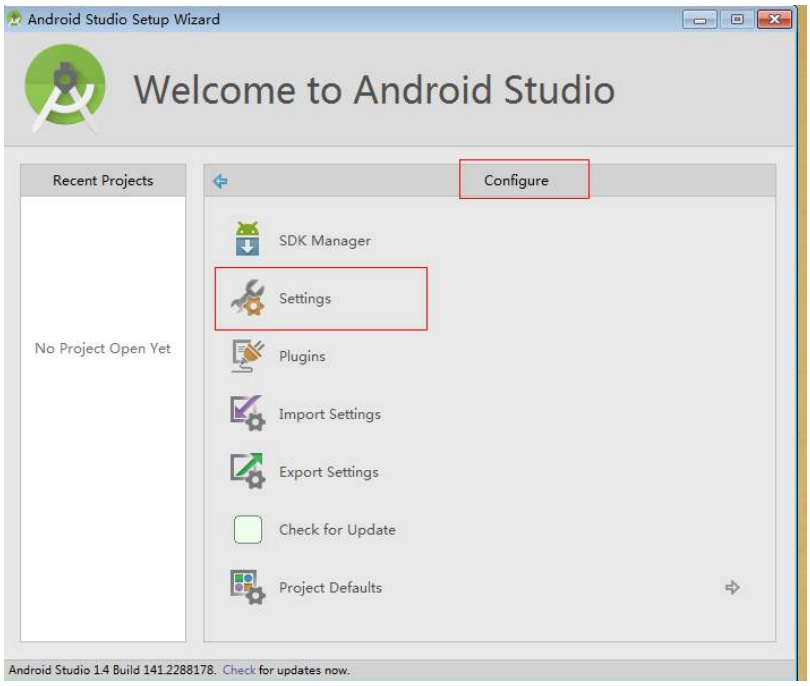


三. 相关常用设置

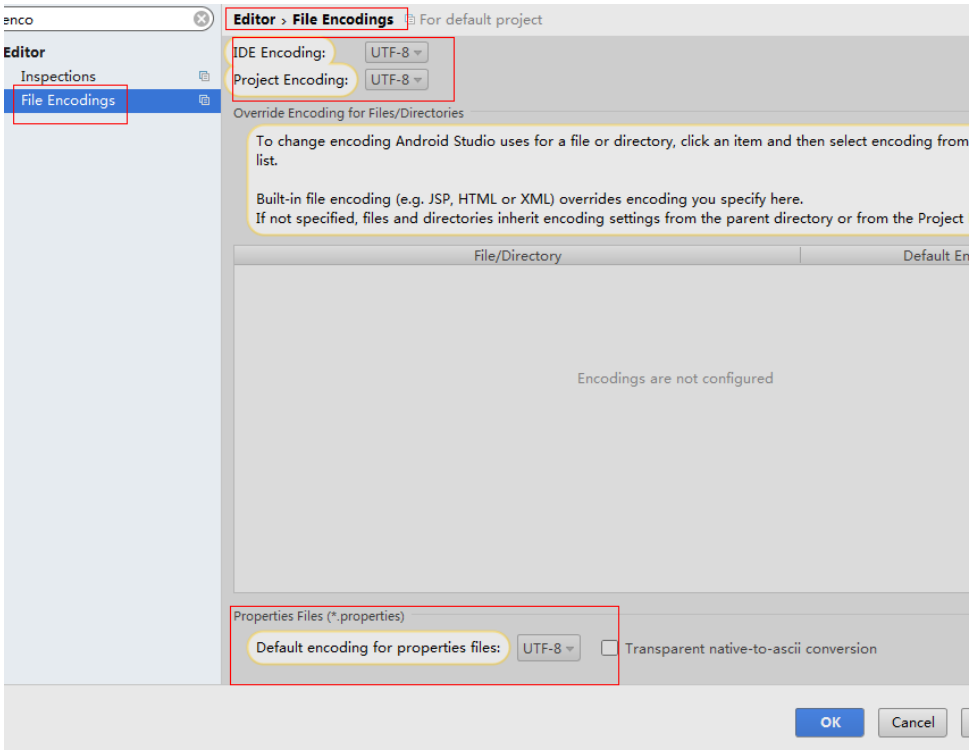
1_AS 启动向导(首页)



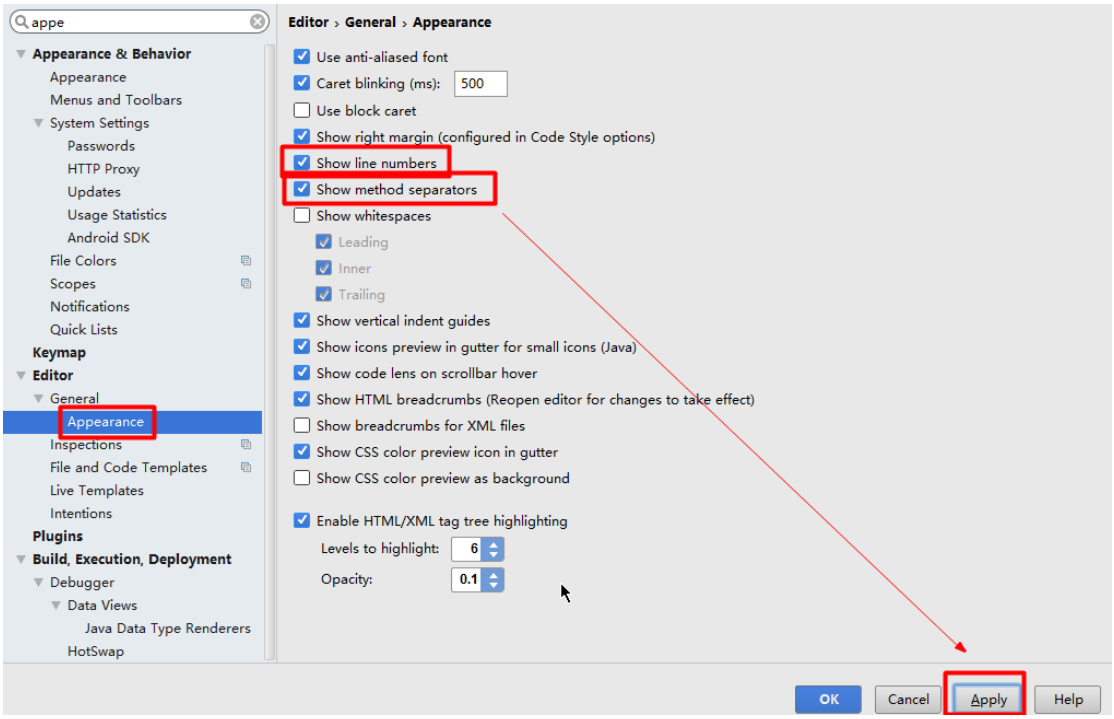
2_相关设置向导(登录以后，首先进行必要的设置)



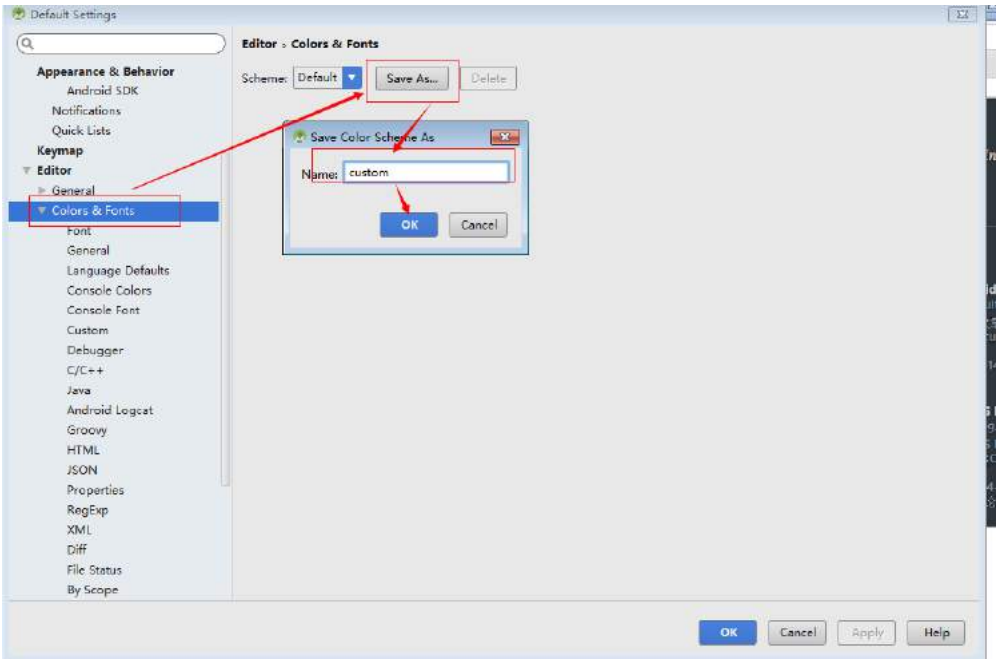
3_修改文件编码



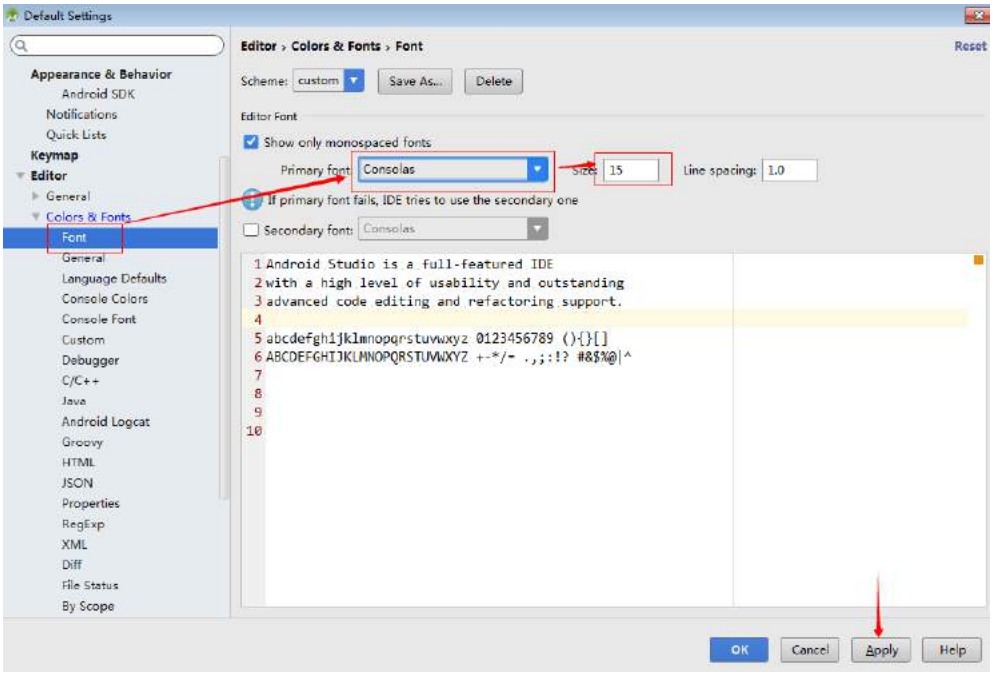
4_设置显示行号和方法间的分隔符



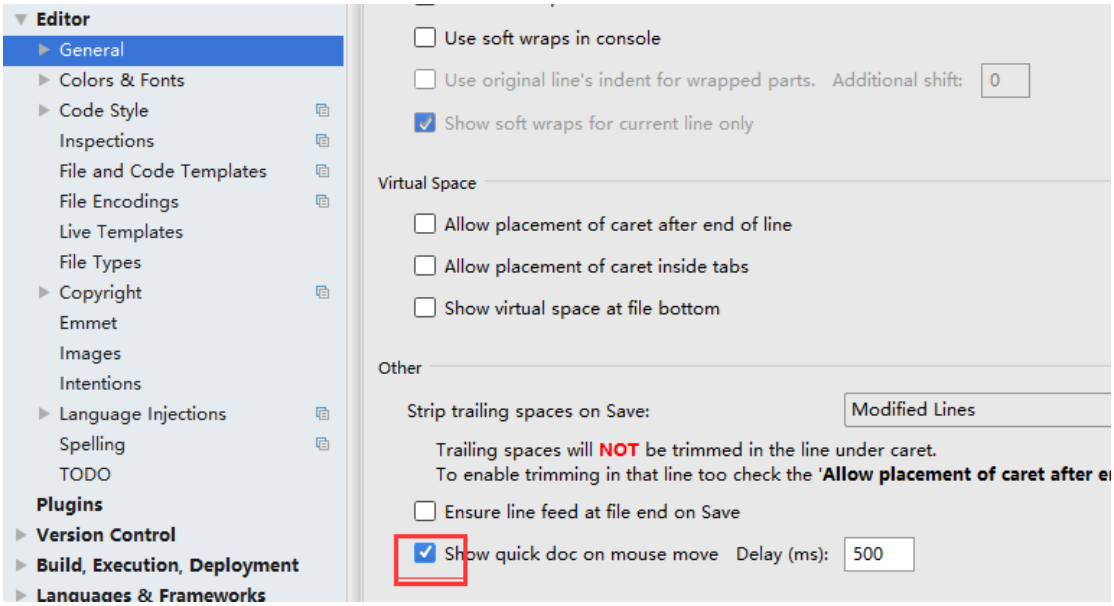
5_设置添加自定义字体和颜色



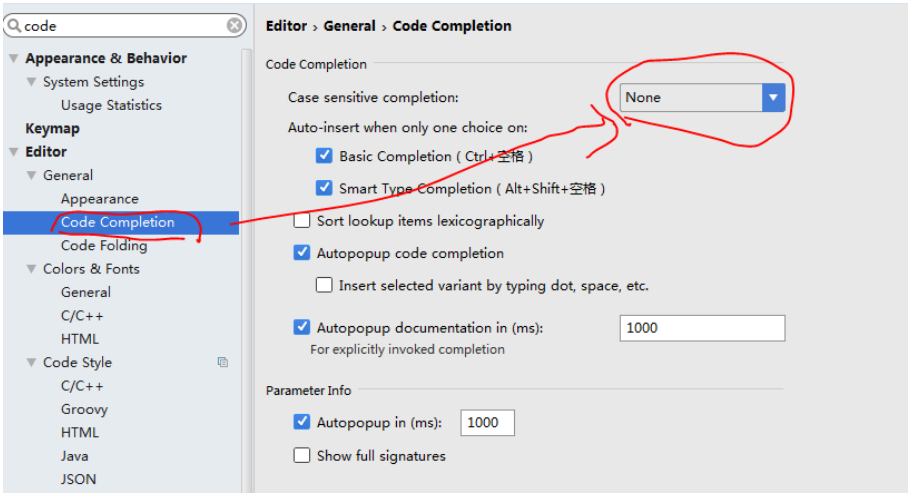
6_修改字体大小和样式



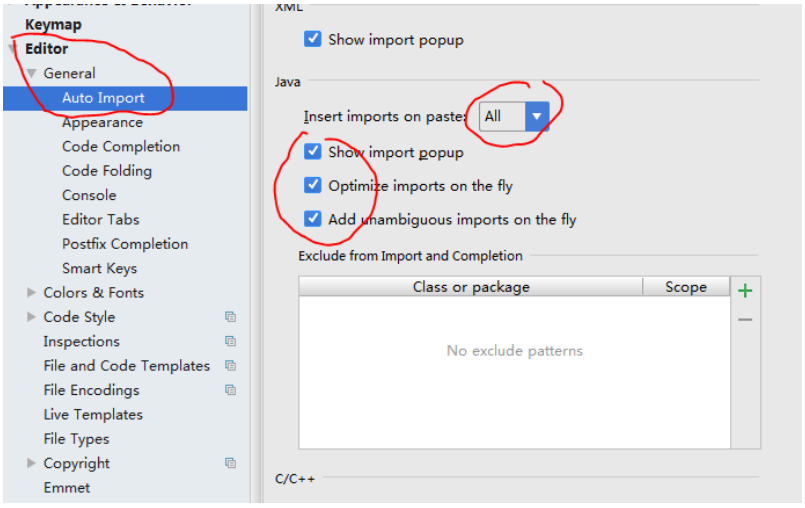
7.设置鼠标悬浮提示



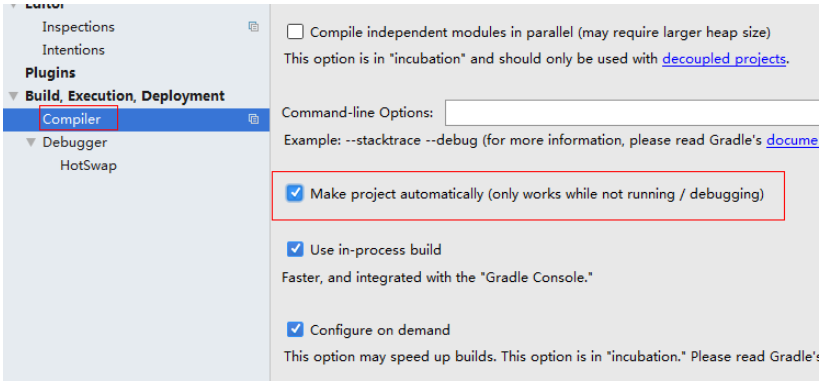
8_忽略大小写_提示



9_设置自动导包

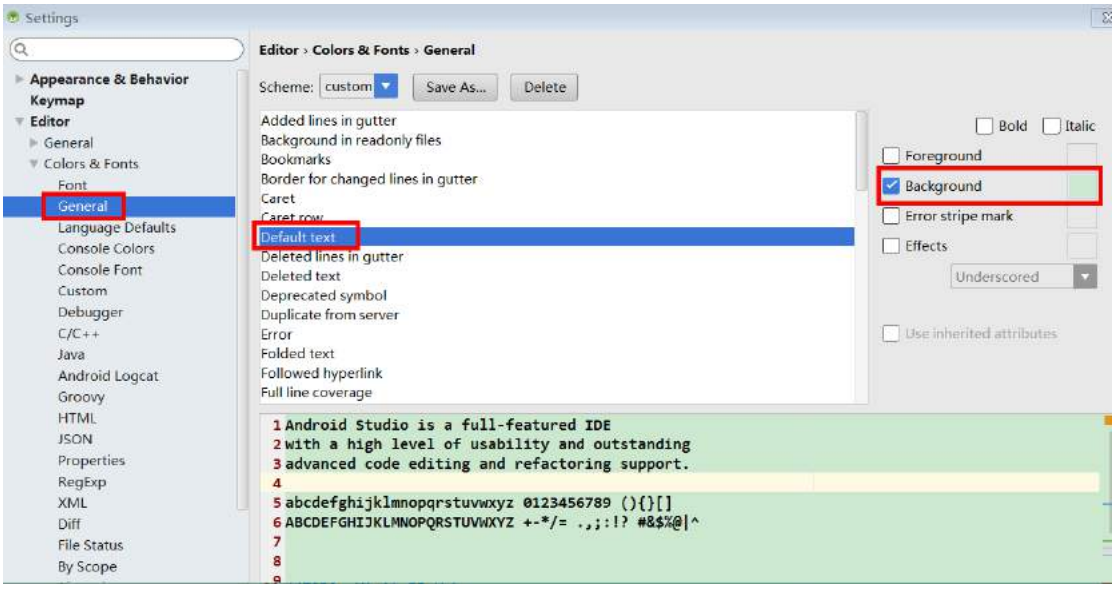


10.设置自动编译

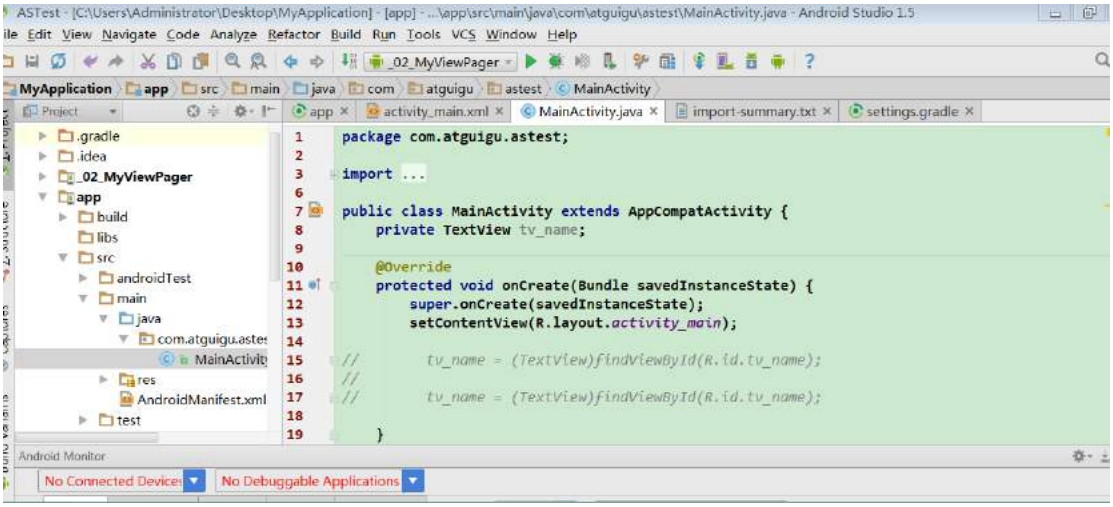


11.设置编译区背景色

1、点击 Color&Fonts-> General-> Default text -> 点击右侧的 Background-> 修改背景色为豆绿色(204,232,207)

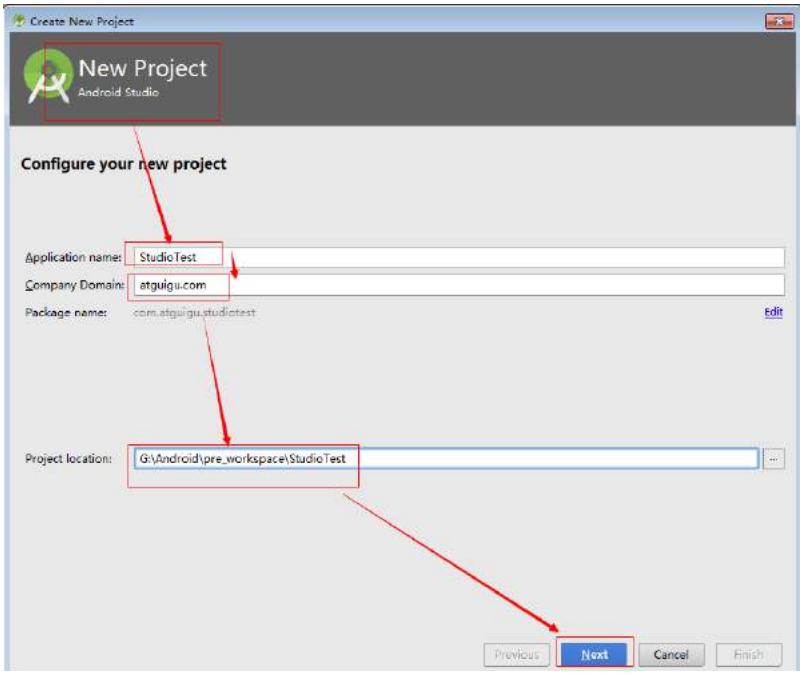


2、修改后的编译区页面显示

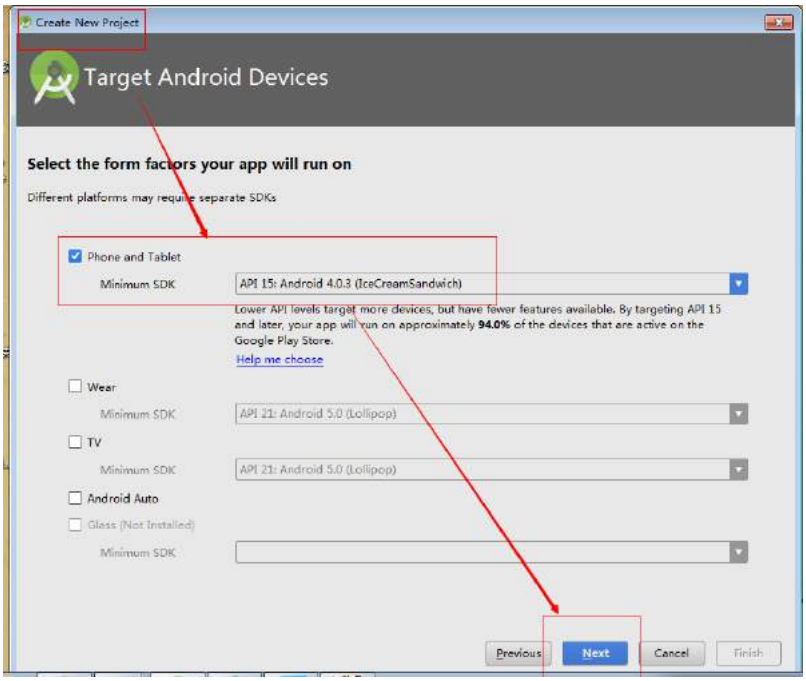


四. 创建 AS project(最好联网)

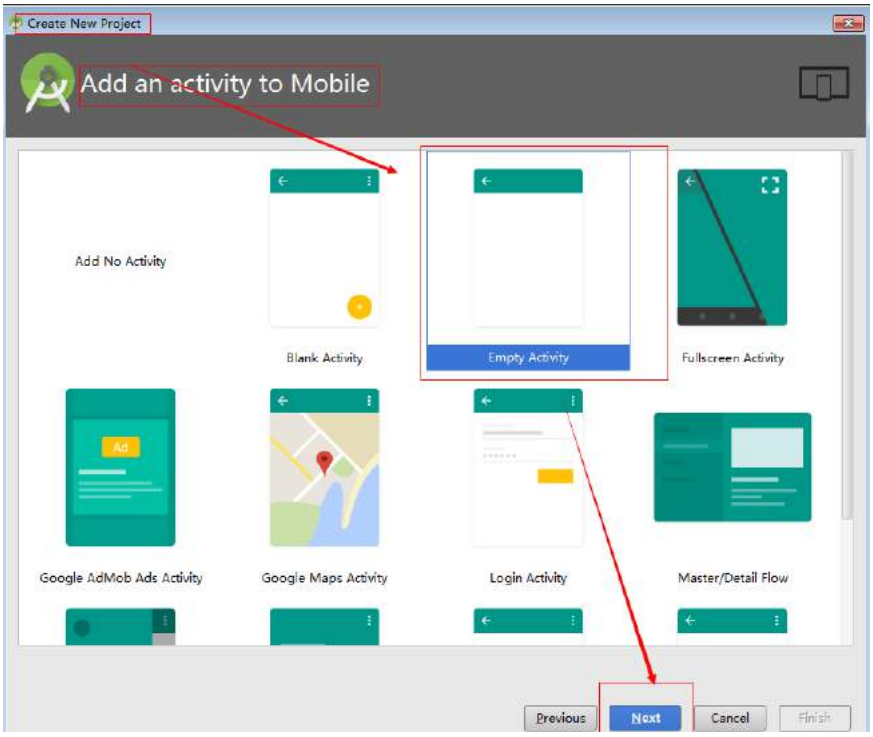
1_新建 AS project(AS 的 project 相当于 Eclipse 的 workspace)



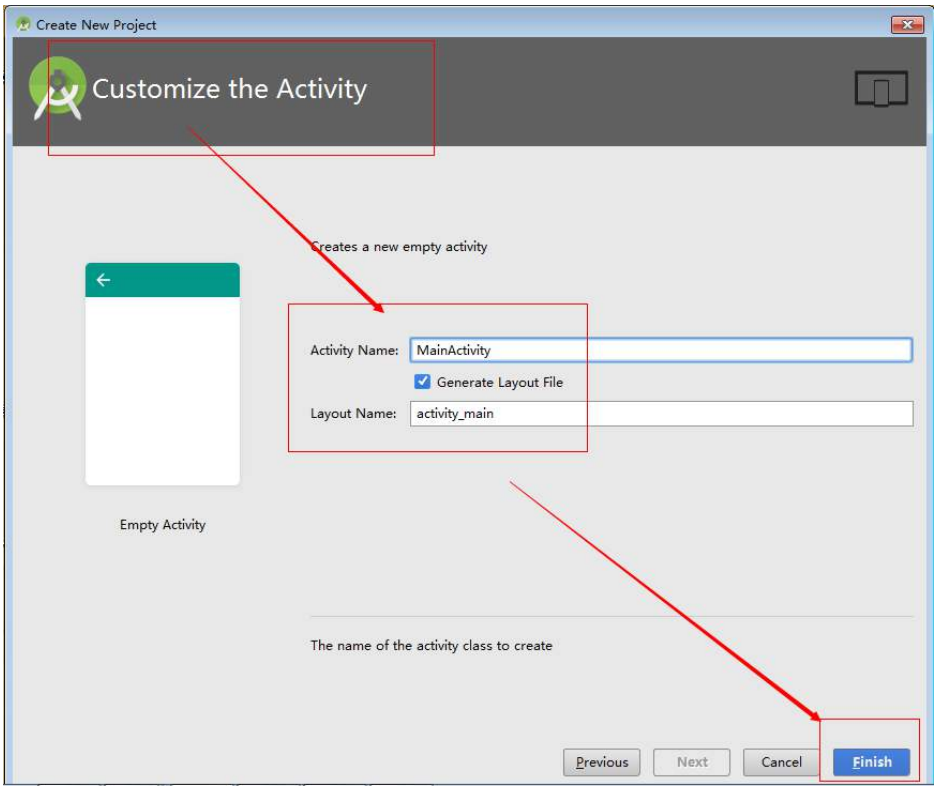
2_指定项目的手机_平板项目



3_指定创建一个空 Activity

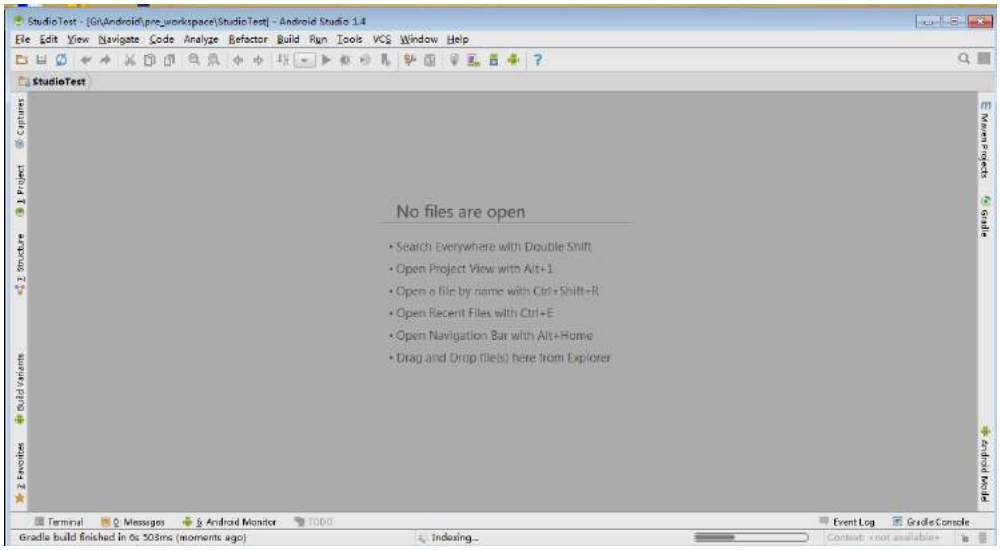


4_完成配置并去创建 Project

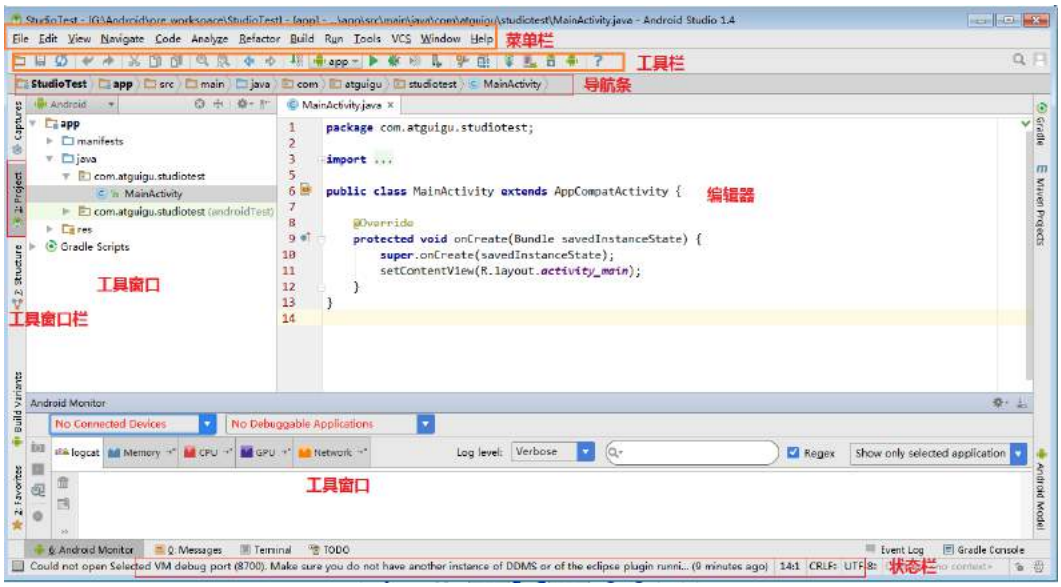


5_创建生成相关数据的过程(要一些时间)

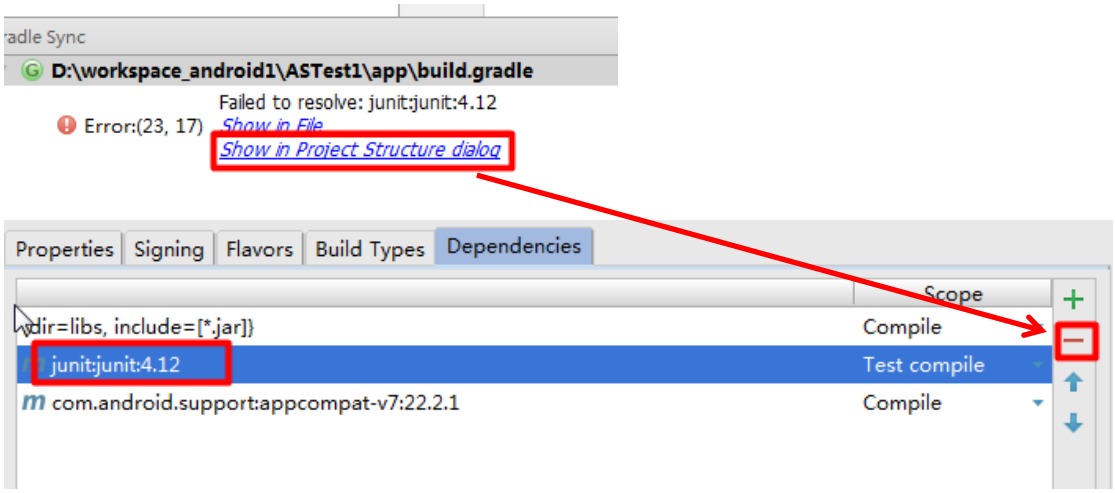
6_进入 AS 工作界面



7_AS 工作界面整体结构

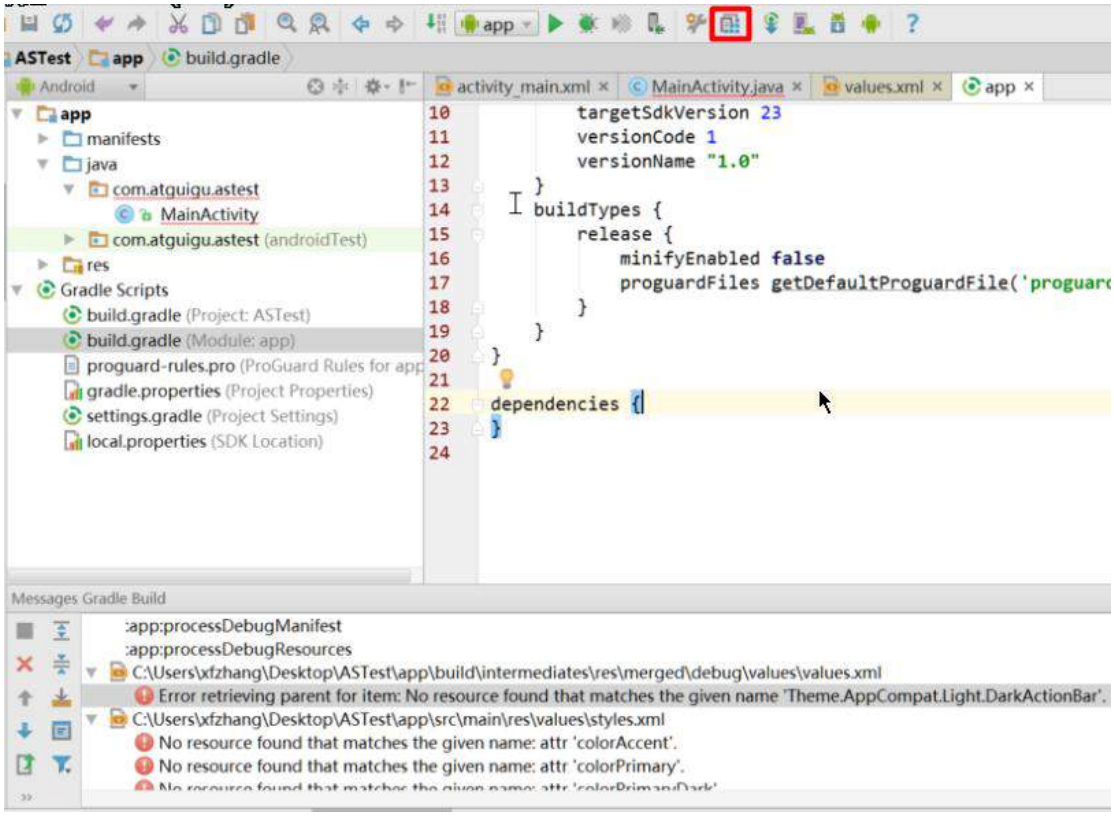


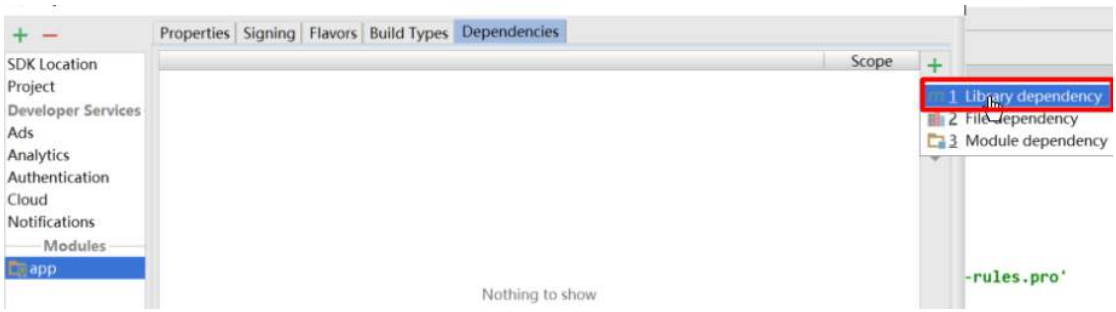
如果创建工程时，没有网络：



然后在右下角有加载的提示：

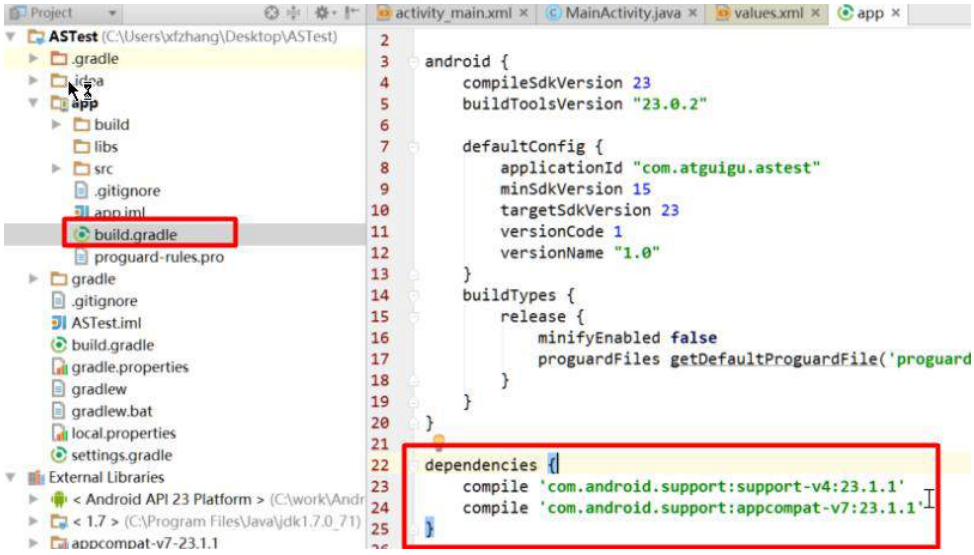
然后加载完以后，发现还报错：按照图示添加 v4 和 v7 包



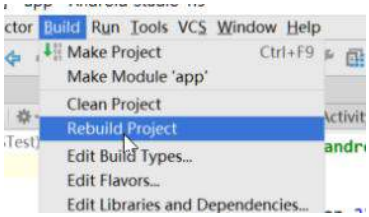


然后确定后加载，就没有问题了。

其实我们添加的配置在如下的位置有显示：

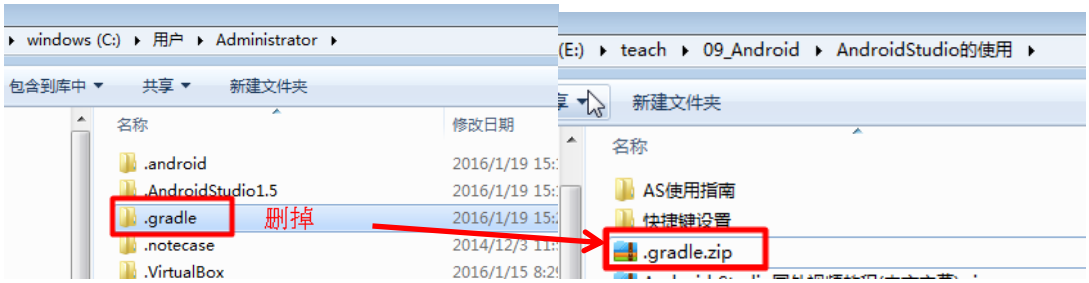


然后还可以 rebuild 一下，确保界面不报错：



或者还有一种方式解决创建工程后的编译错误问题：直接删除 module 下缺少的文件配置，然后 rebuild 一下工程即可。

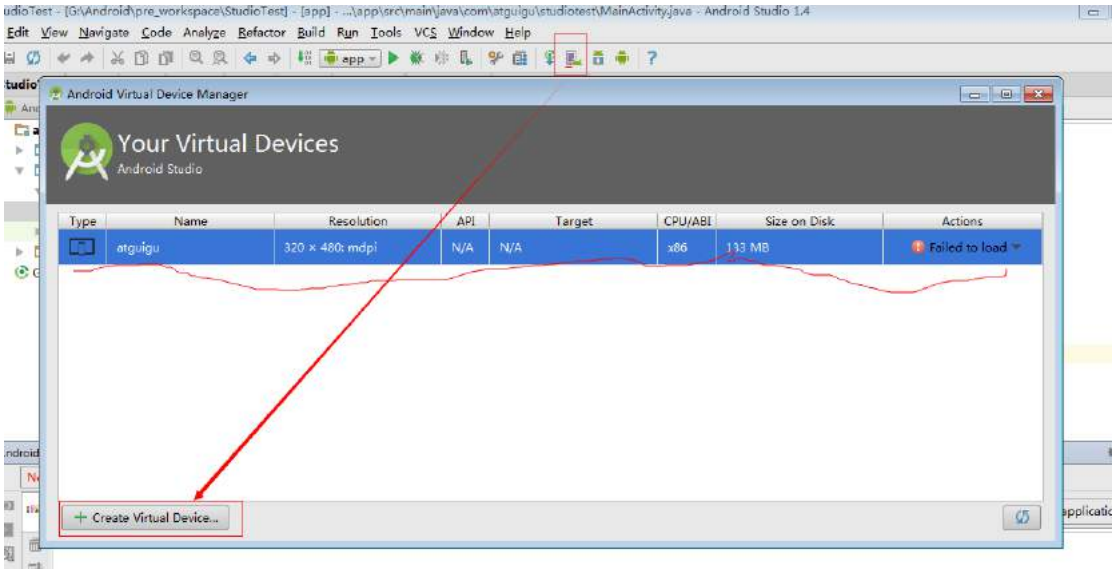
注意：首次打开 Android Studio 时，会联网下载更新 gradle，在没网的情况下，每次创建 module 都可能会报异常。这里根据报的 Junit 找不到的异常，可以把相应的文件删掉。这里用户可以将解压后生成的目录下.gradle 删掉，改为自己解压的.gradle。因为内部包含了 Junit 等 jar 包。



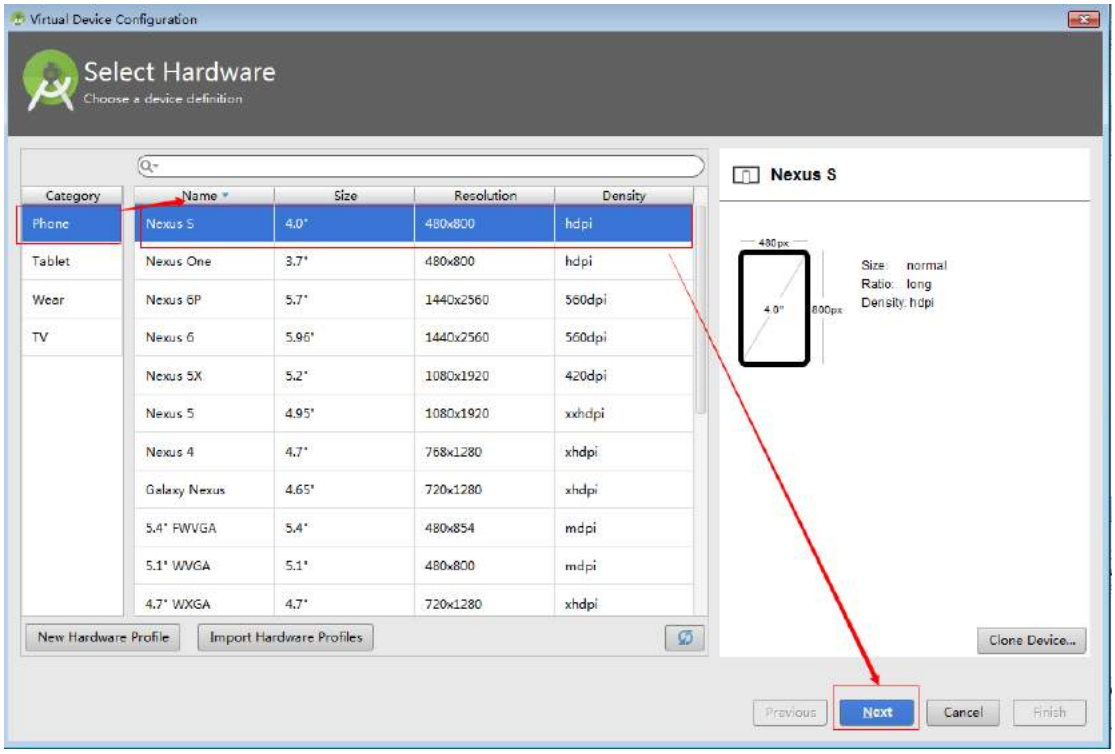
使用下面压缩文件解压以后的.gradle 替换上面自动生成的.gradle 文件

五. 创建模拟器并运行项目

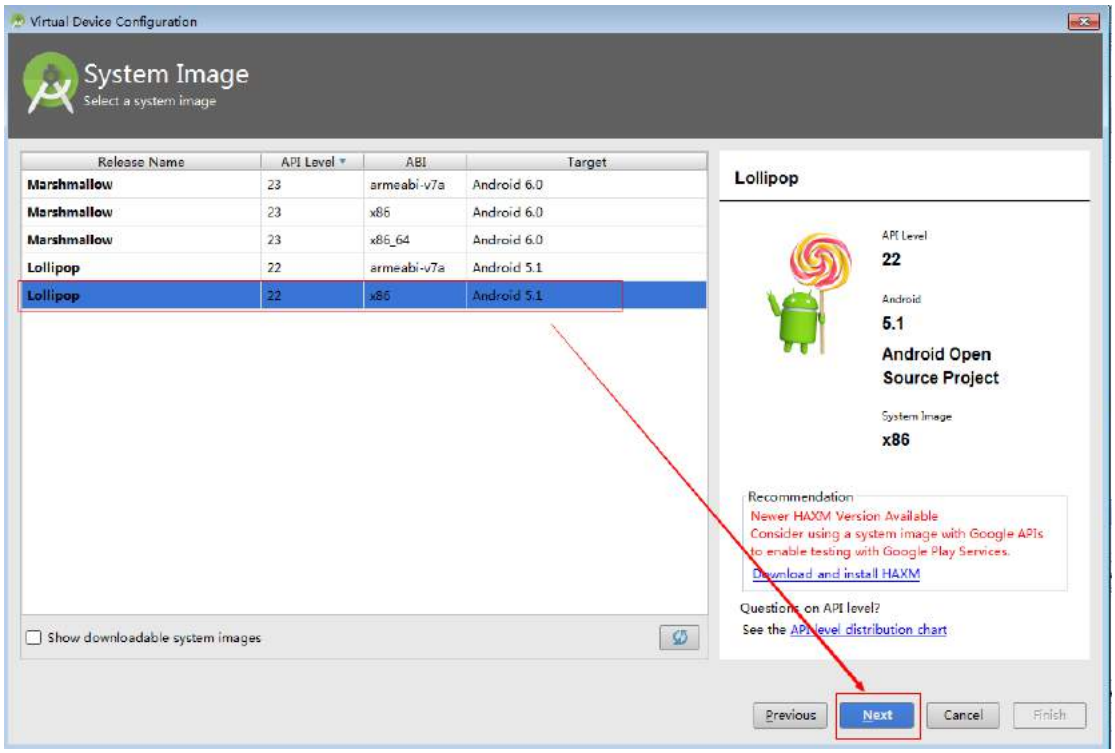
1_创建模拟器(开始)



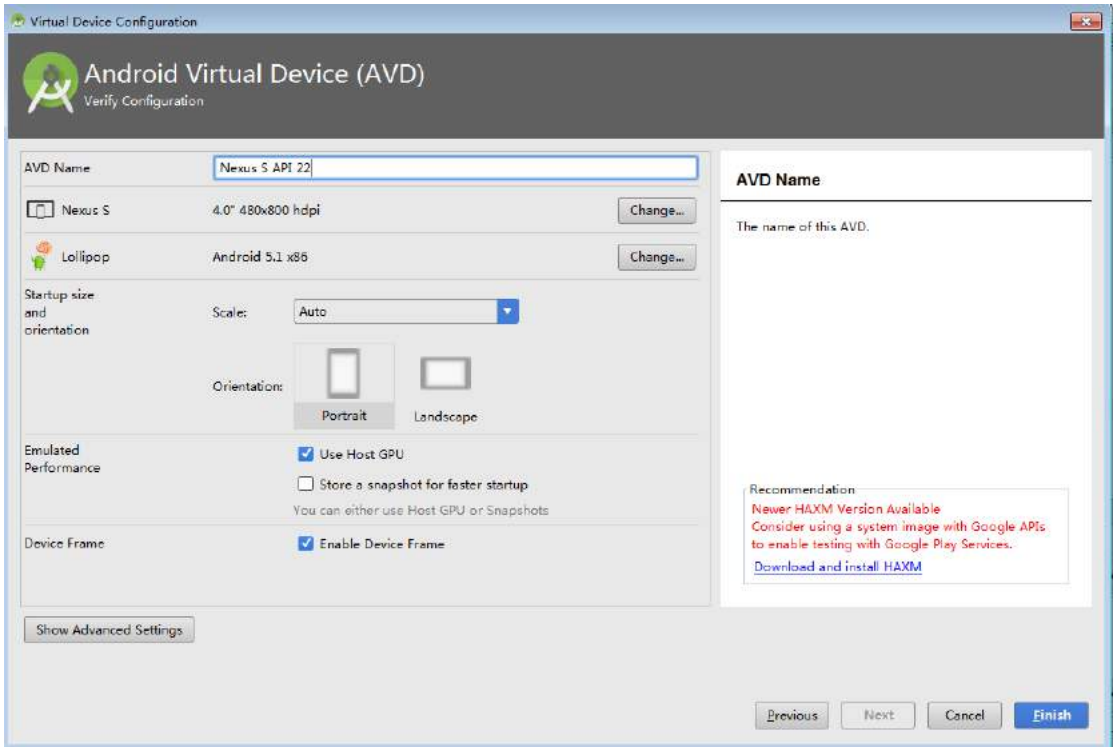
2_指定模拟器型号



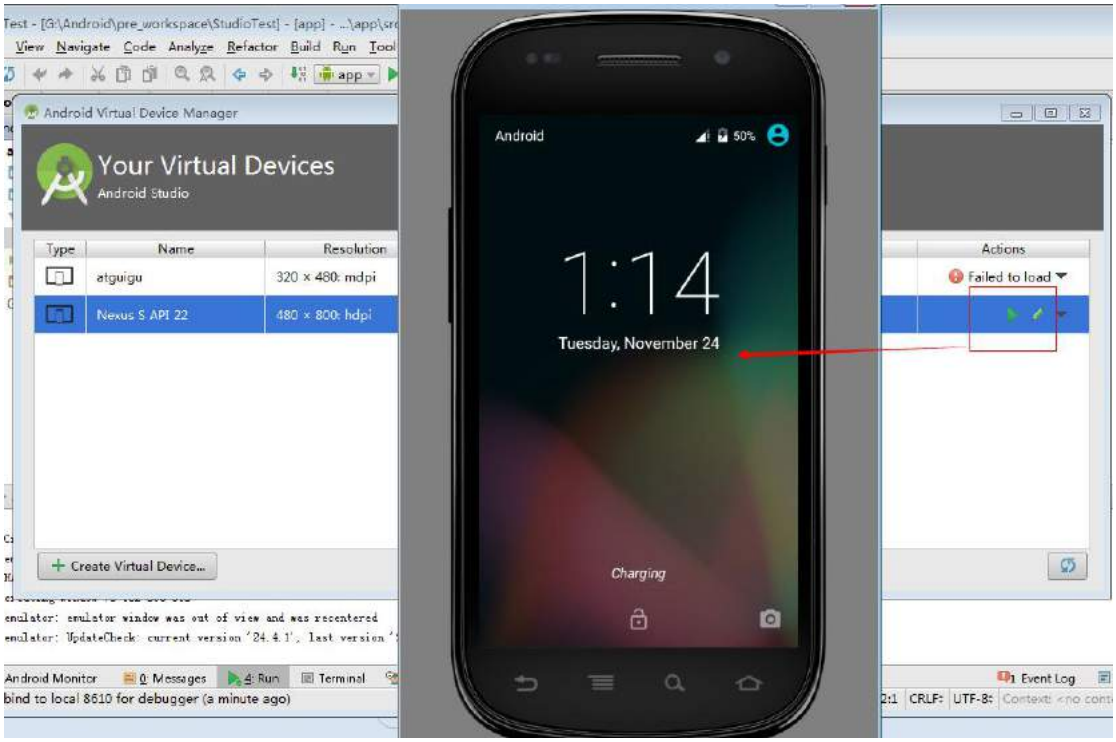
3_指定模拟器的 sdk 版本



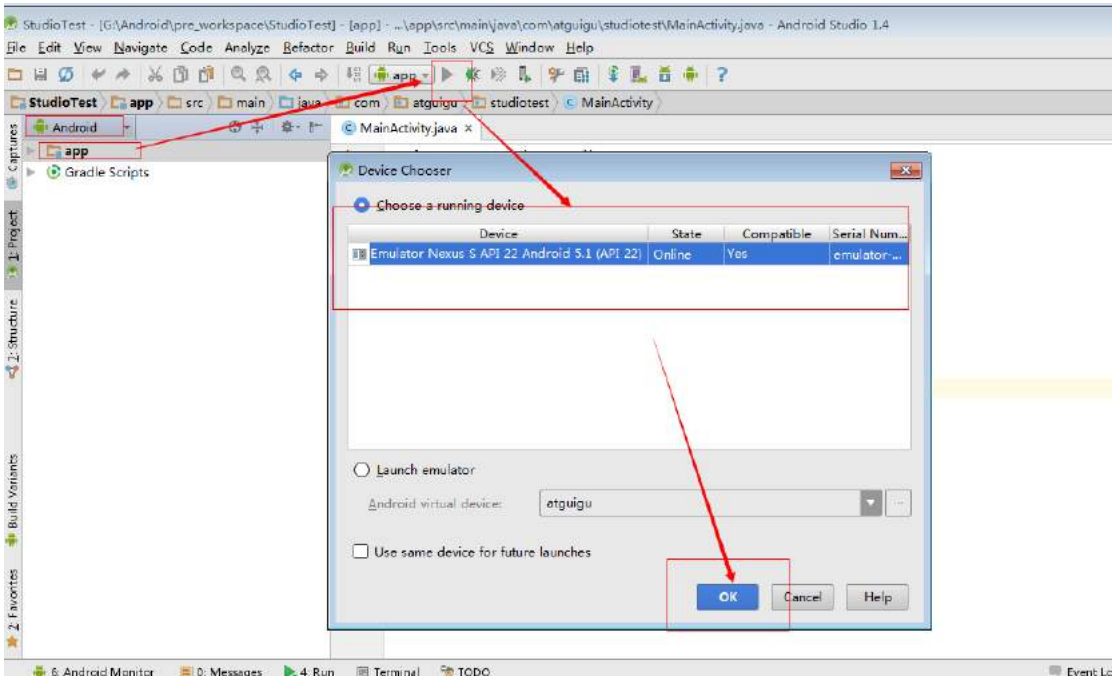
4_确定创建模拟器



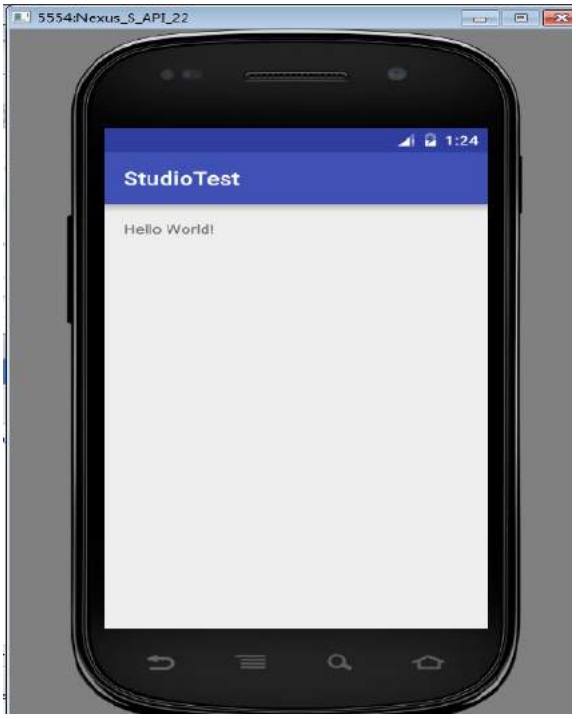
5_启动创建的模拟器



6_将项目运行安装到模拟器

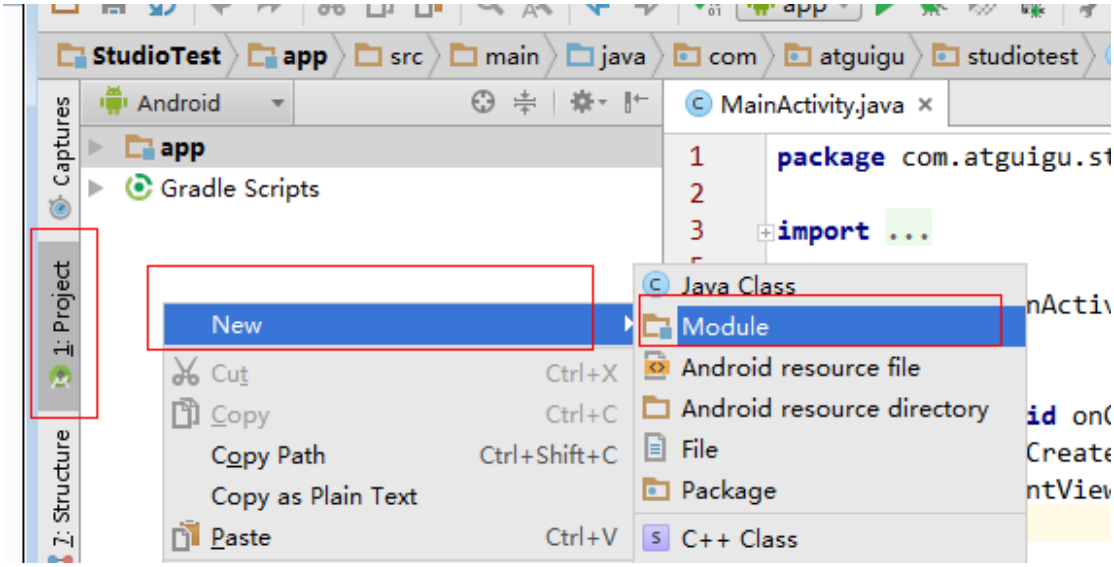


7_模拟器运行应用

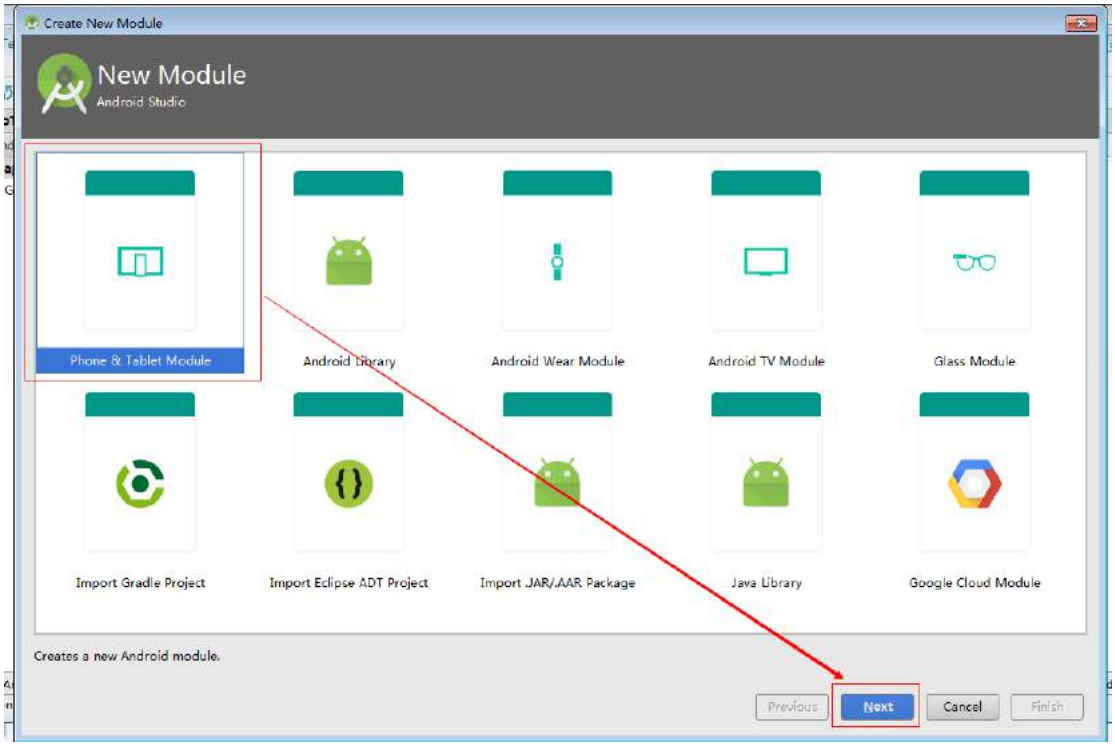


六. 新建和删除 module

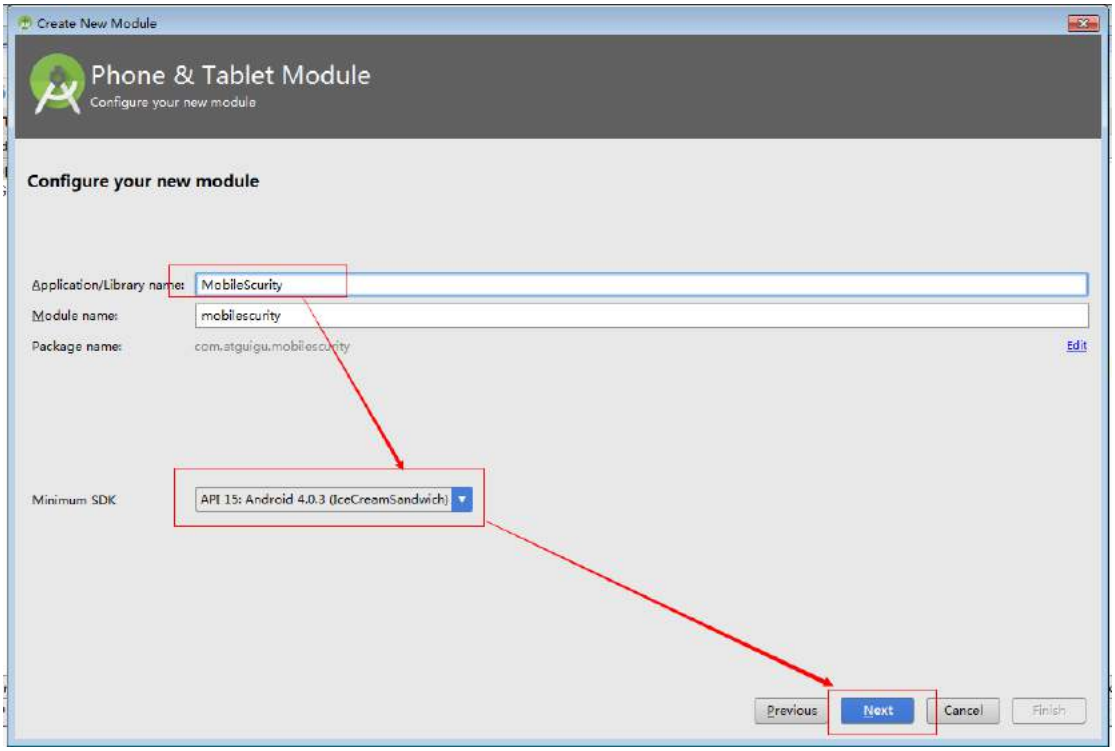
1_新建 Module



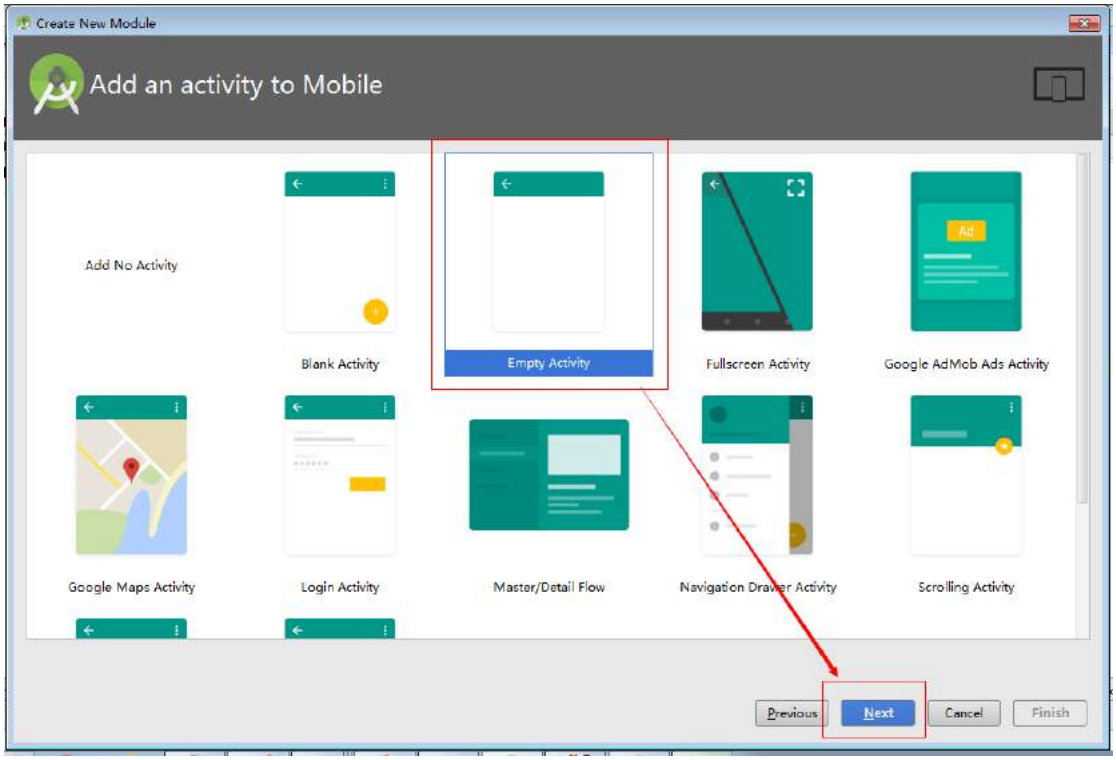
2_指定为手机或平板应用



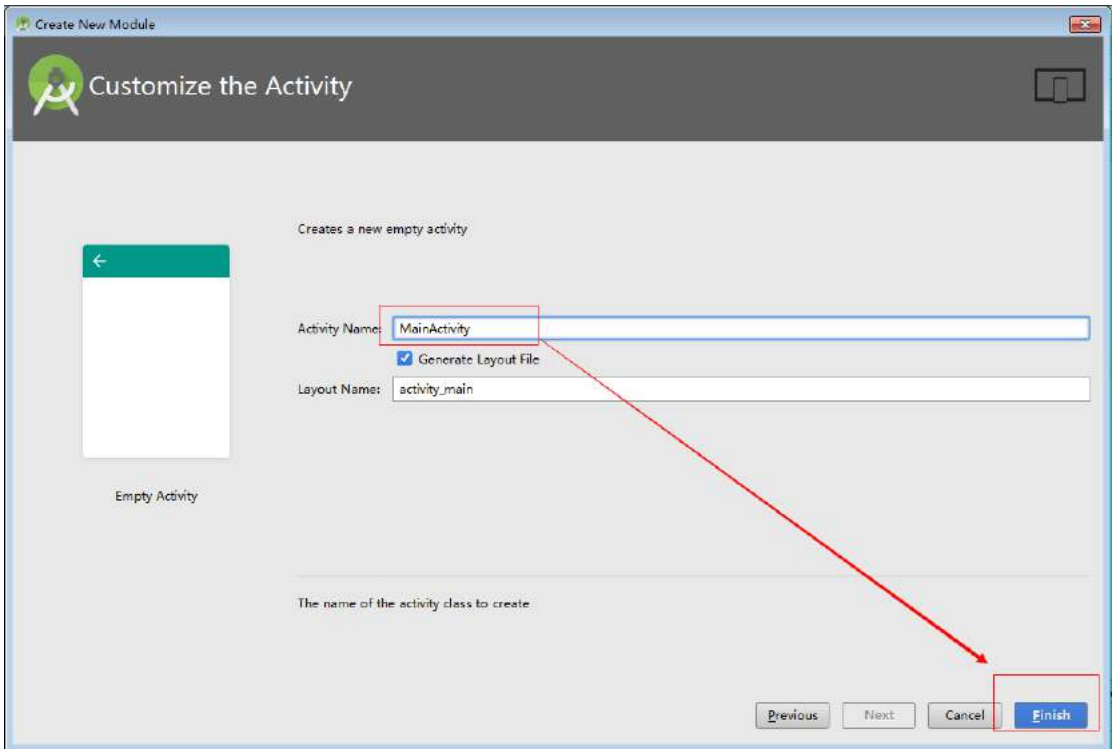
3_指定应用名称



4_指定创建空 Activity

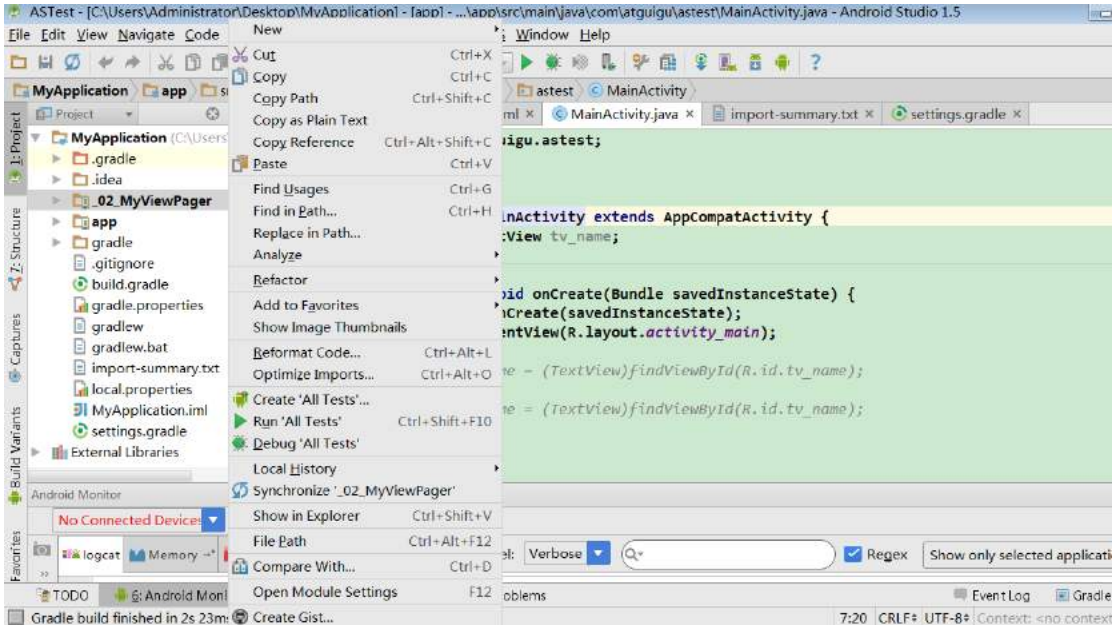


5_确定完成 Module 的创建

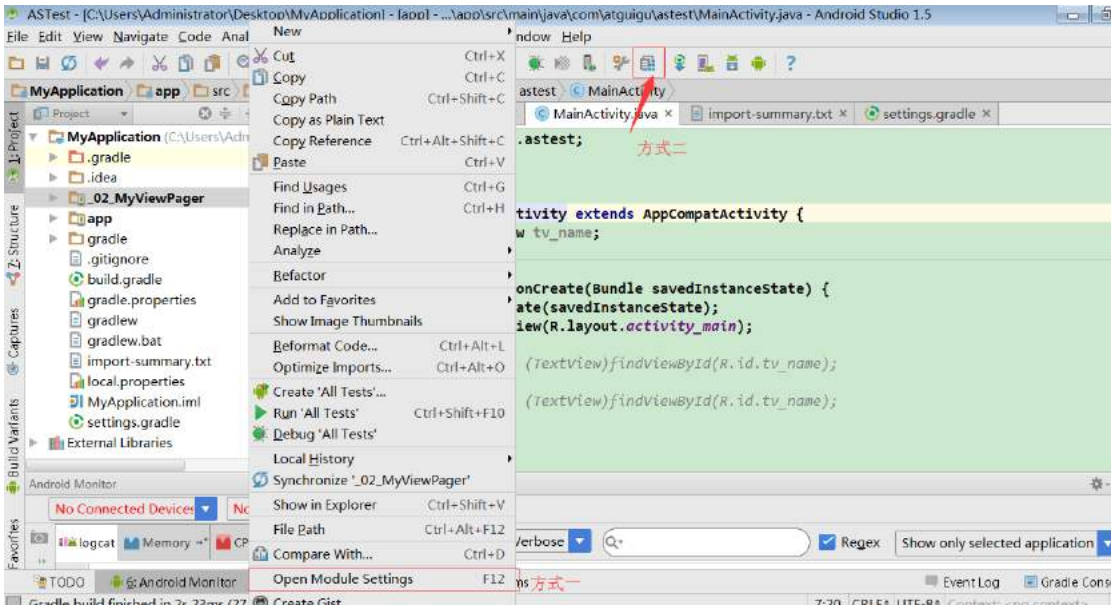


6_删除 Module 的步骤

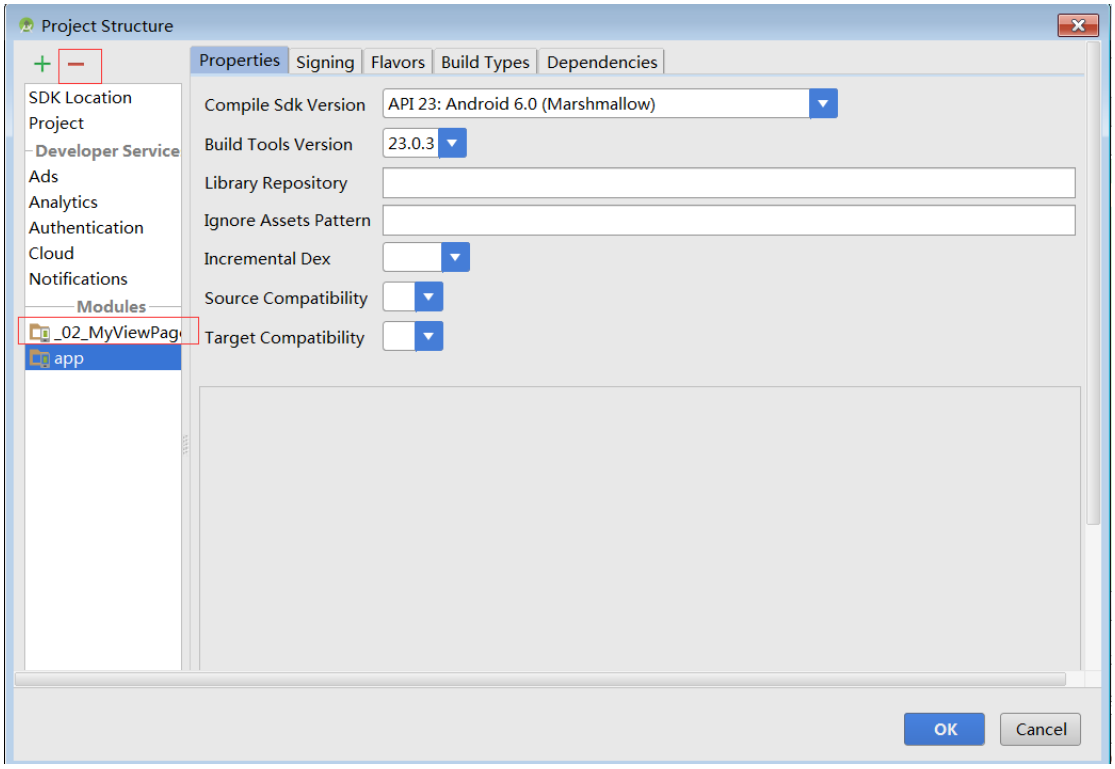
1、 在准备删除的 module 上点击鼠标右键，默认是找不到 delete 按钮的



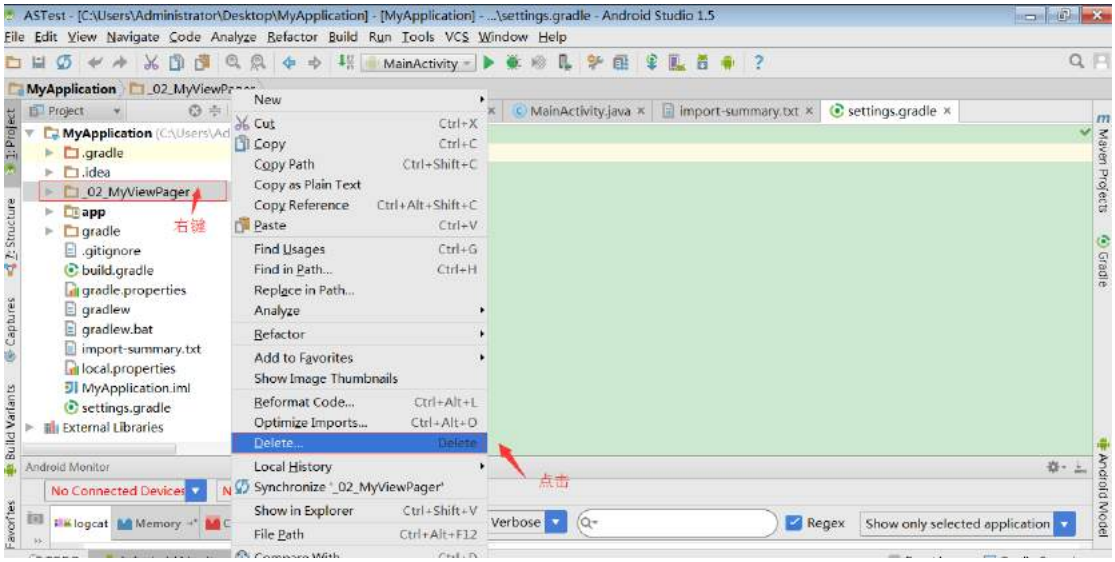
2、 点击方式一和方式二都可以；或者在项目的 setting.gradle 的文件中中直接将要删除的模块删除



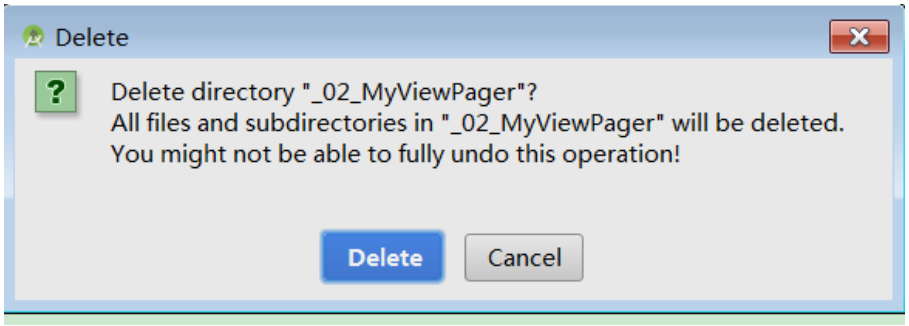
3、点击要删除的模块->点击“-”号删除该模块，然后点击 OK 按钮完成



3、 再次回到主窗体中的 project 视图，在要删除的模块上右键



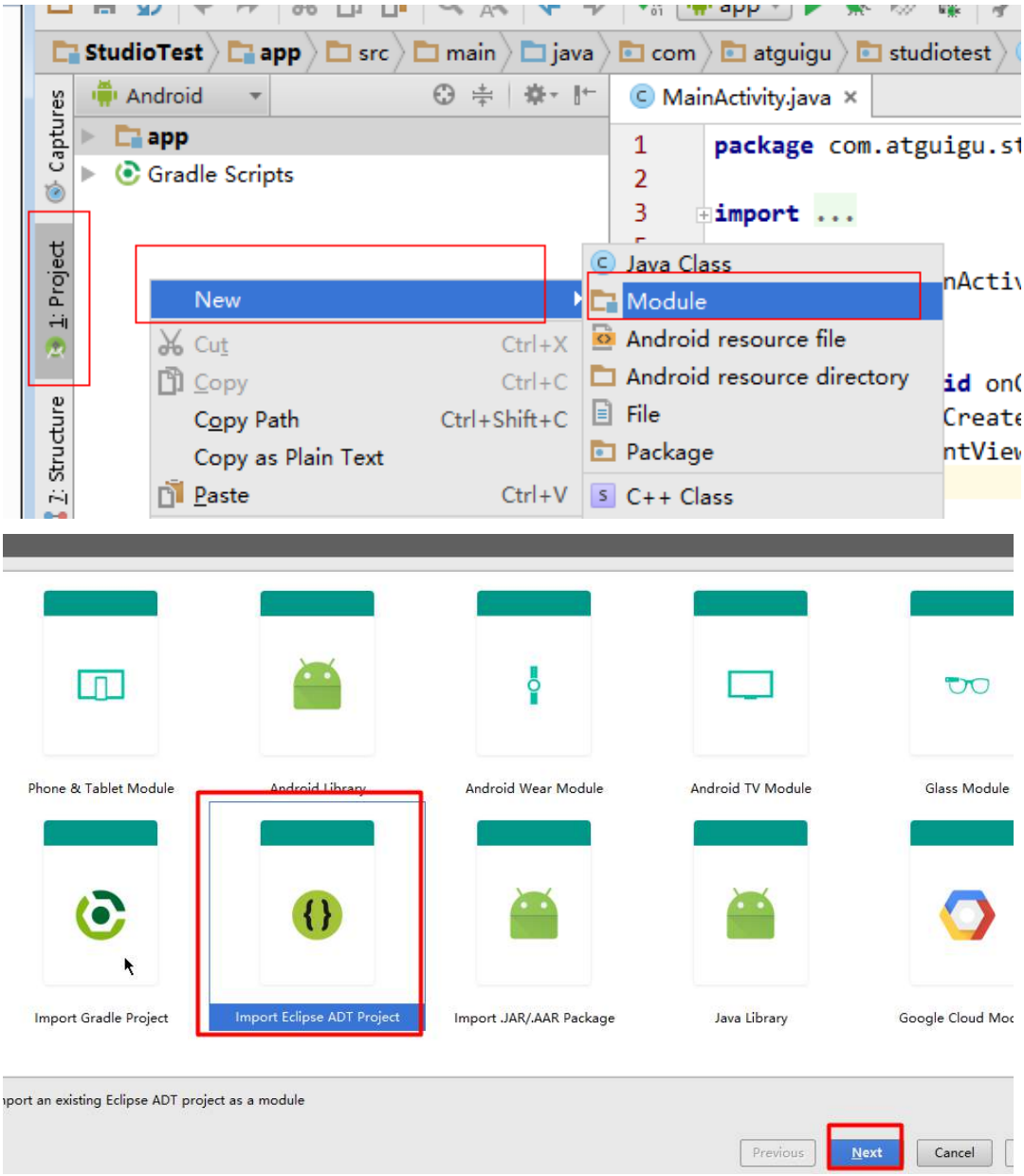
4、在弹出的对话框中点击 Delete



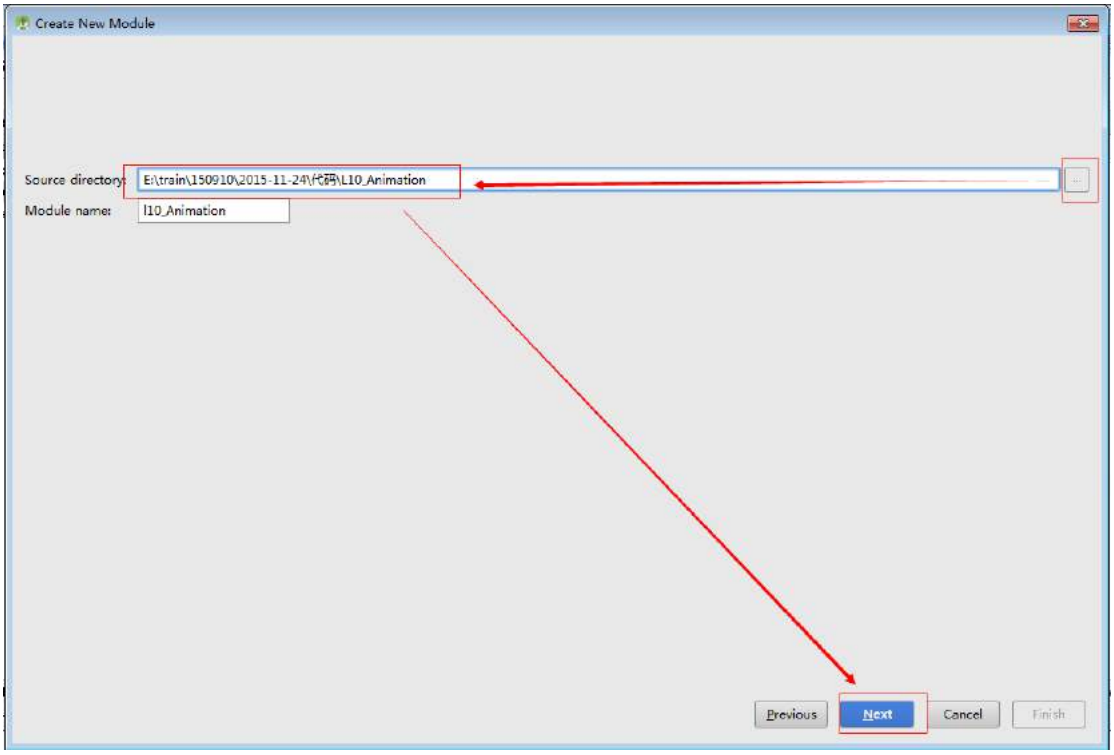
5、这样在项目中就将 module 模块删除了。

七. 导入 Eclipse 工程

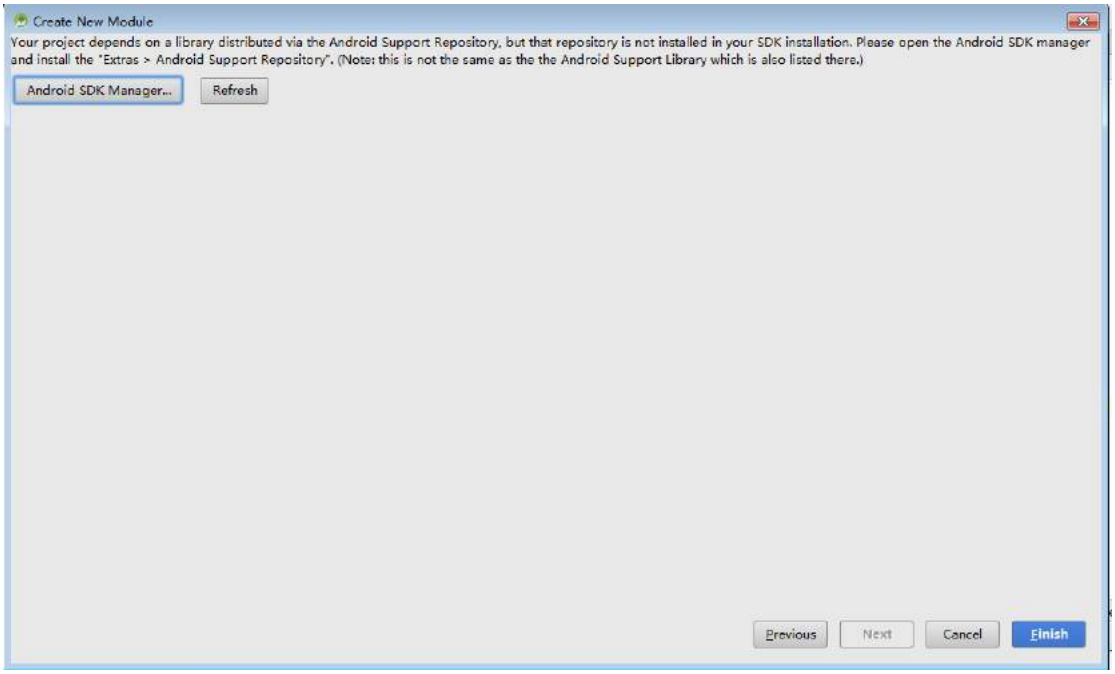
1_新建 Module



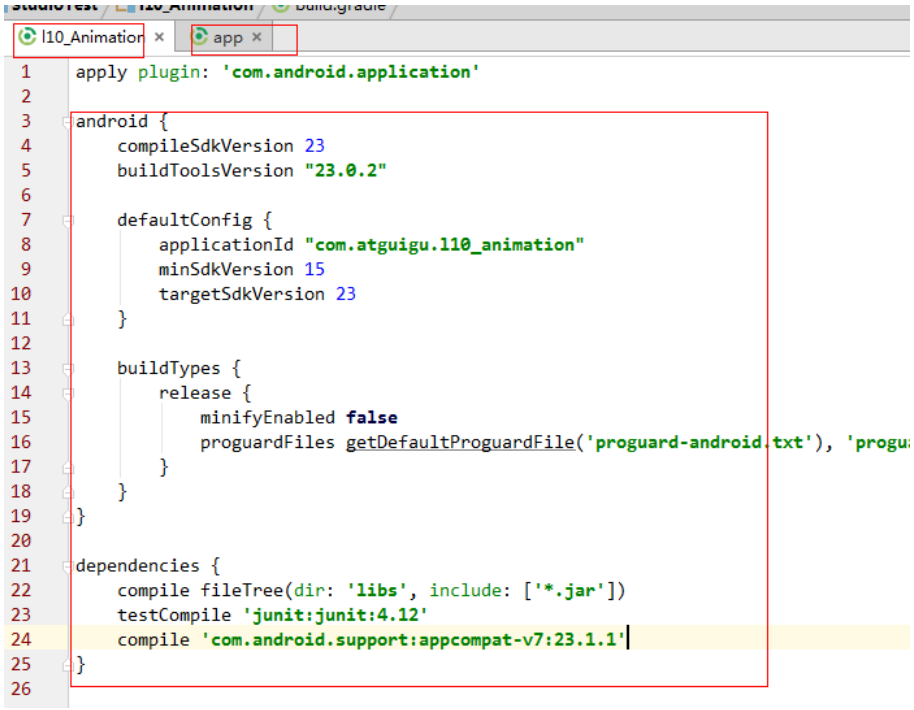
2_指定要导入 Eclipse 工程的目录



3_完成



4_修改因为 sdk 版本问题错误(参照其它 module)



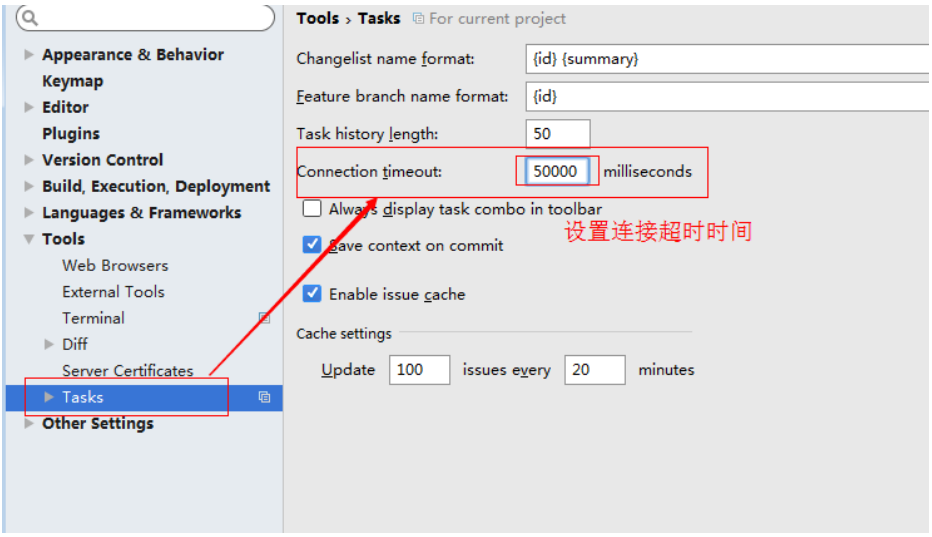
八.其它设置

1_区别一般 Module 与 Android 库

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "com.atguigu.ms_final"
        minSdkVersion 15
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
```

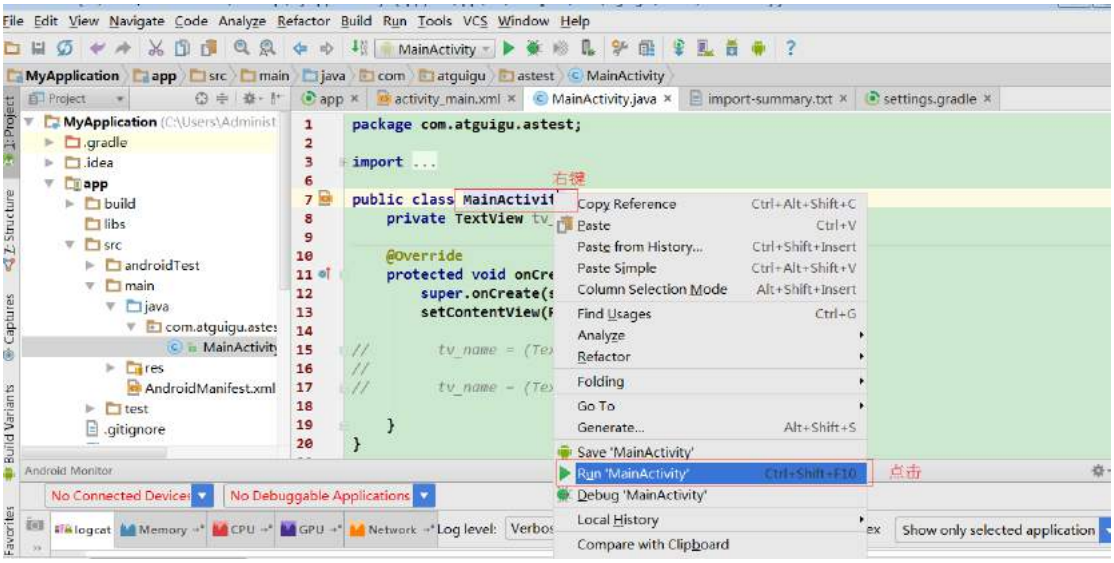
```
apply plugin: 'com.android.library'

android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        minSdkVersion 15
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
```

2.设置连接超时时间



3.单个 Activity 运行

在任意一个 activity 页面，在 activity 类上方点击右键->Run 当前 activity 类（例如：’Run MainActivity’）



4.查看本地 SDK 路径下的 V7 和 V4 包版本

1、v7 包路径：C:\android-studio-ide-1.5-windows\sdk\extras\android\m2repository\com\android\support\appcompat-v7

2、v4 包路径

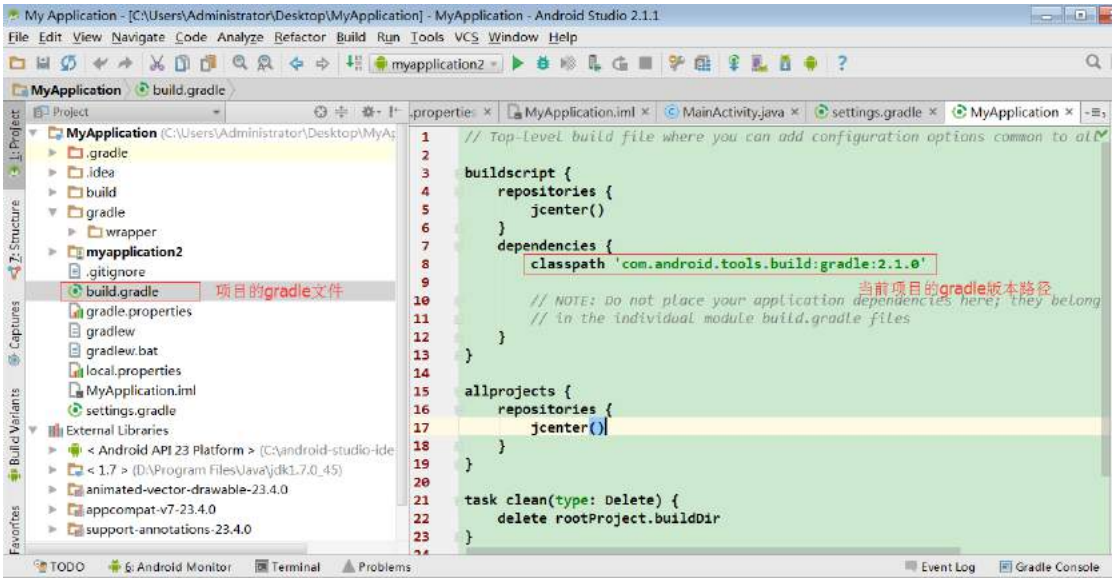
C:\android-studio-ide-1.5-windows\sdk\extras\android\m2repository\com\android\support\support-v4

5.Eclipse 与 android studio 的四个重要概念

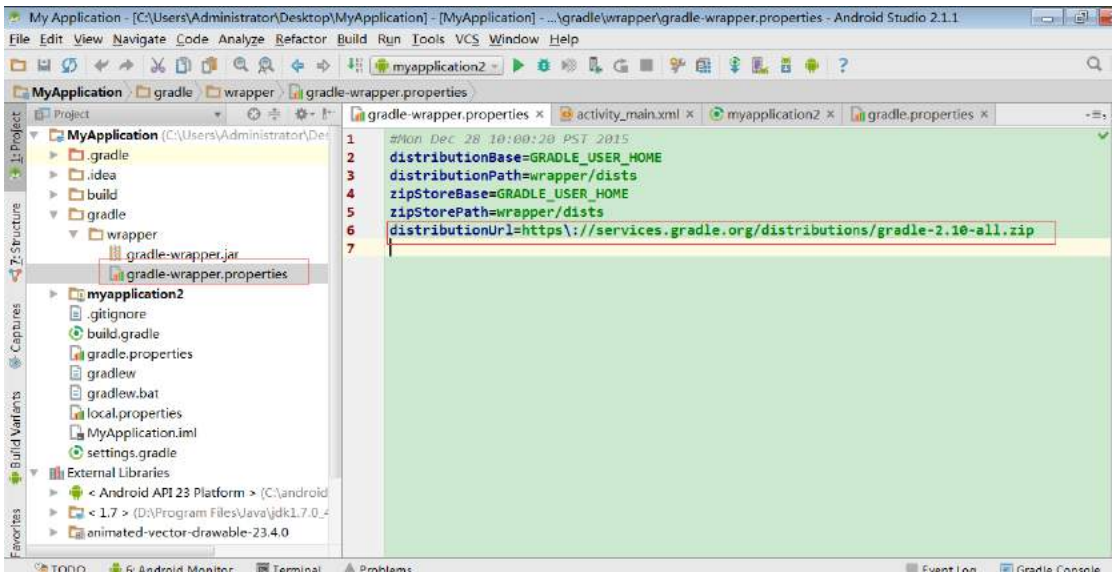
1、Eclipse 的 workspace 相当于 Android studio 中的 Project
2、Eclipse 的 Project 相当于 Android studio 中的 Module

6.android studio1.5 和 android2.1.1 项目的相互转换步骤

1、修改当前项目的 gradle 版本：1.5.0 转换为 2.1.0 版本
classpath 'com.android.tools.build:gradle:1.5.0' 修改为
classpath **'com.android.tools.build:gradle:2.1.0'**

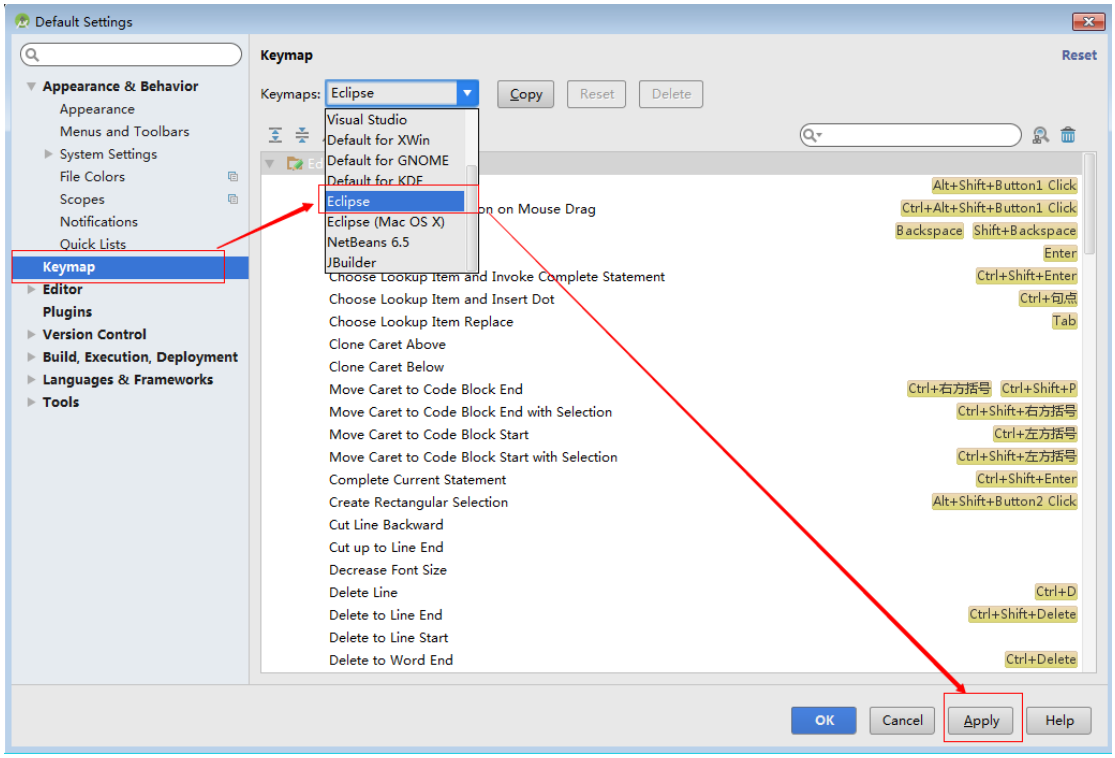


2、修改 gradle-wrapper.properties 中的（1.5.0 转换为 2.1.0 版本）
distributionUrl=https\://services.gradle.org/distributions/gradle-2.8-all.zip 修改为
distributionUrl=https\://services.gradle.org/distributions/gradle-2.10-all.zip

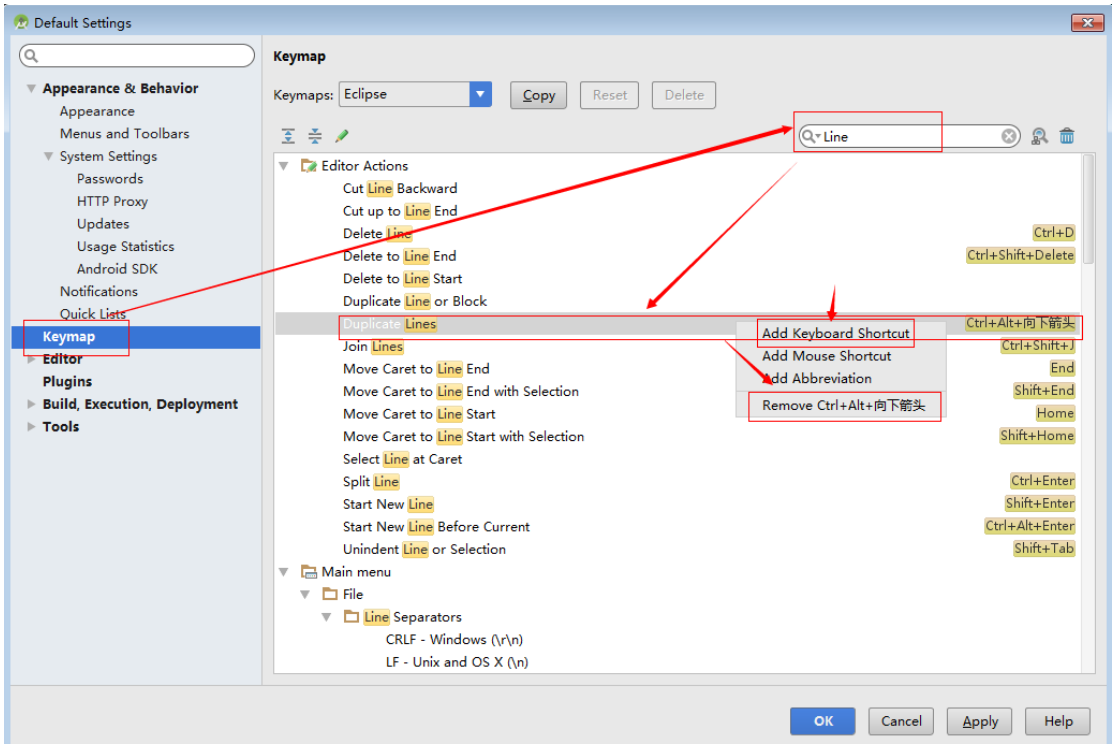


九.AS 快捷键

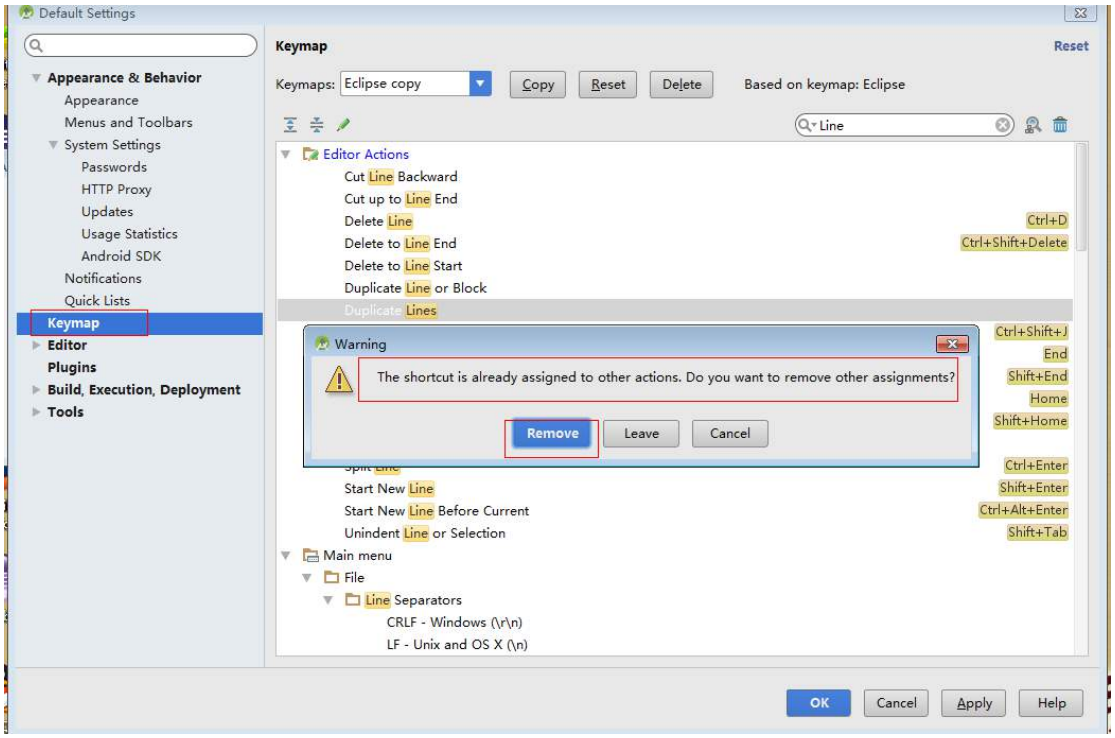
1_设置快捷为 Eclipse 的快捷键(但还是有些会不同)



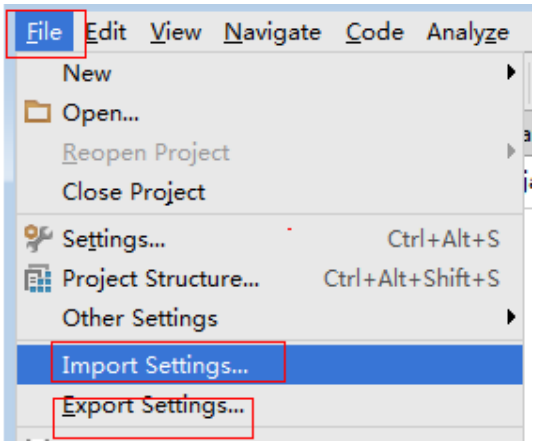
2_修改快捷键

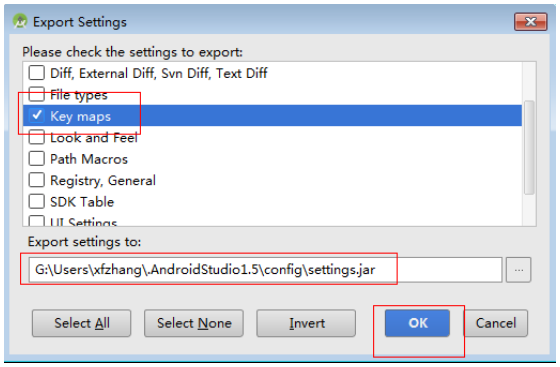


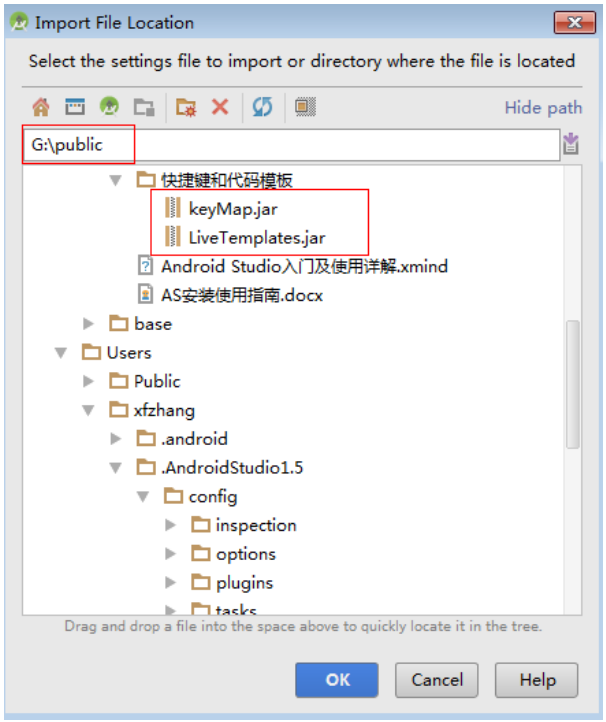
3_修改快捷键_删除重复的

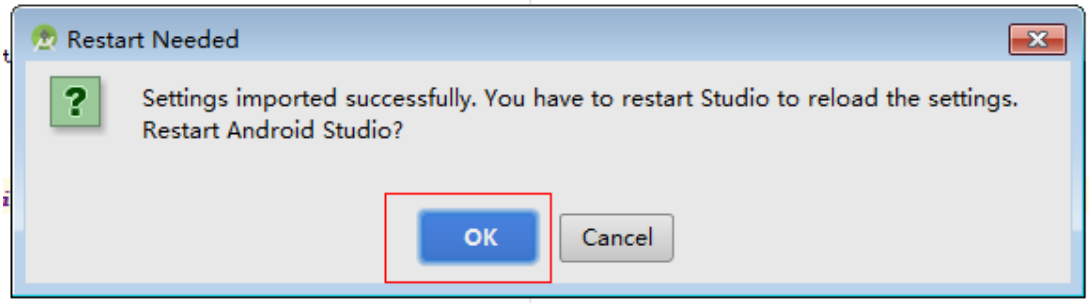


4_导出导入设置


File
Edit
View
Navigate
Code
Analyze
New
Open...
Reopen Project
Close Project
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
Other Settings
Import Settings...
Export Settings...


Export Settings
Please check the settings to export:
Diff, External Diff, Svn Diff, Text Diff
File types
Key maps
Look and Feel
Path Macros
Registry, General
SDK Table
Export settings to:
G:\Users\xfzhang\.AndroidStudio1.5\config\settings.jar
Select All
Select None
Invert
OK
Cancel


Import File Location
Select the settings file to import or directory where the file is located
Hide path
G:\public
快捷键和代码模板
keyMap.jar
LiveTemplates.jar
Android Studio入门及使用详解.xmind
AS安装使用指南.docx
base
Users
Public
xfzhang
.android
.AndroidStudio1.5
config
inspection
options
plugins
Drag and drop a file into the space above to quickly locate it in the tree.
OK
Cancel
Help

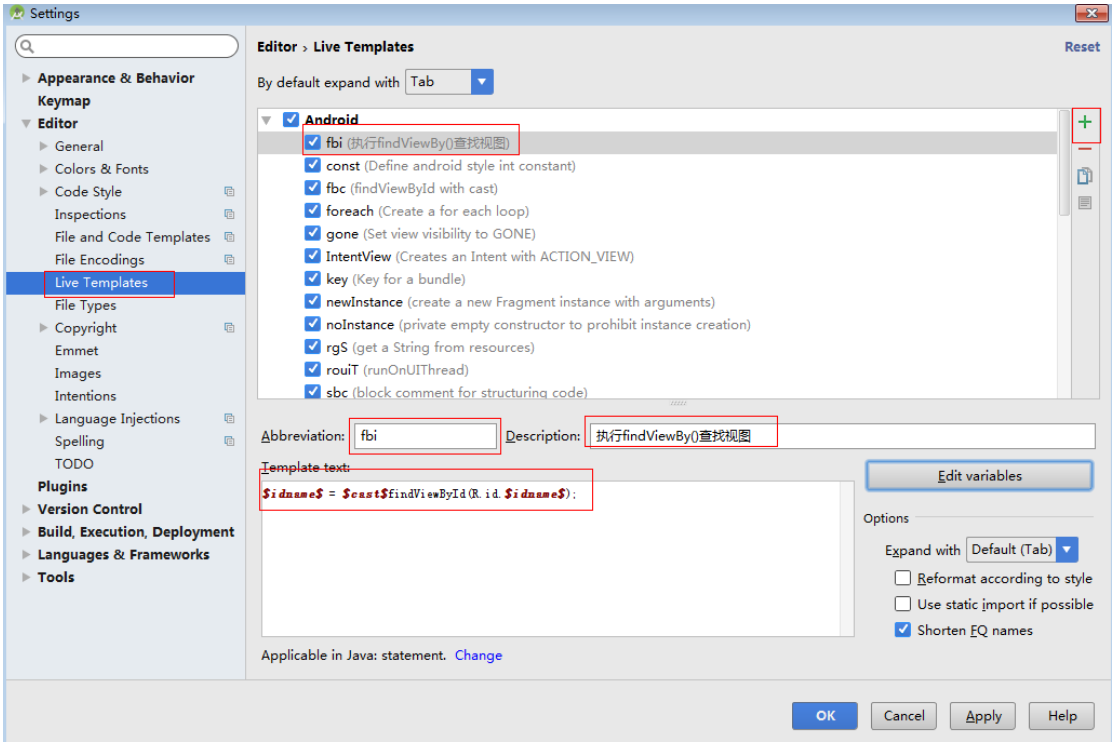


5_常用快捷键

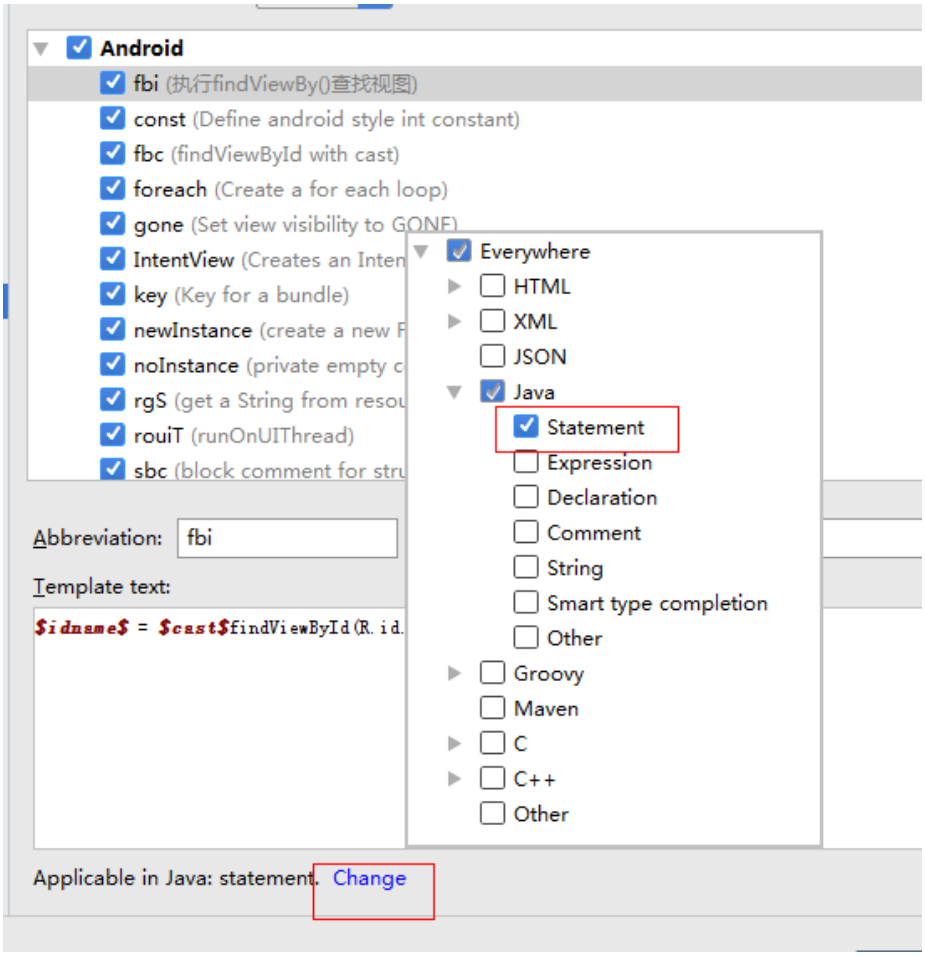
向下复制一行 *(Duplicate Lines)*	**ctrl+down**	修改变量名与方法名	**alt+shift+R**
向下移动行 *(move line down)*	**Alt+down**	打开 Module setting 窗口	**ctrl+shift+E**
向上开始新的一行 *(Start New Line before current)*	**ctrl+shift+enter**	查看类继承关系	**F4**
`向下开始新的一行	**shift+enter**	查看文档说明	**F2**
提示补全 *(Class Name Completion)*	**alt+/**	查看类的结构	**ctrl+O**
万能解错/生成返回值变量	**alt + enter**	单选注释 多行注释	**ctrl + /** **ctrl + shift + /**
关闭展开的窗口栏	**shift+esc**	格式化代码	**Ctrl+shift+F**
大写转小写 小写转大写	**Ctrl+shift+Y** **Ctrl+shift+X**	查找/替换(当前) 查找/替换(全局)	**Ctrl+F** **Ctrl+H**
最近打开过的文件	**ctrl+E**	删除一行或选中行	**Ctrl+D**
查找文件	**double Shift**	查找没有使用的变量和方法*(Inspect Code..)*	**alt+sift+A**
查找某个类	**ctrl+shift+T**	生成 try----catch	**alt+shift+Z**
生成构造/get/set/toString *(generate)*	**alt +shift + S**	提示方法参数类型 *(Parameter Info)*	**ctrl+alt+/**
抽取方法 *(Extract Method)*	**alt+shift+Q**	补全当前语句 *(Complete Current Statement)*	**ctrl+shift +U**
打开指定文件夹 *(Show in Explorer)*	**Ctrl+shift+V**	局部变量抽取为成员变量 *(Introduce Field)*	**Alt+shift+F**
查找方法在哪有被调用 *(Call Hierarchy)*	**Ctrl+shift+H**	快速搜索类中的错误 *(Go to Next Error)*	**ctrl + shift + Q**
创建 Actvity *(Empty Activity)*	**Ctrl+Alt+A**	导入 Module *(Import Module)*	**Ctrl+alt+M**
提示忽略大小 ：Settings -->Code Completion-->case sensitive Completion-->选择 none			
设置自动导包 ：Settings→Auto Import→Insert Import on past→选择 All，勾选所有			

十．代码模板

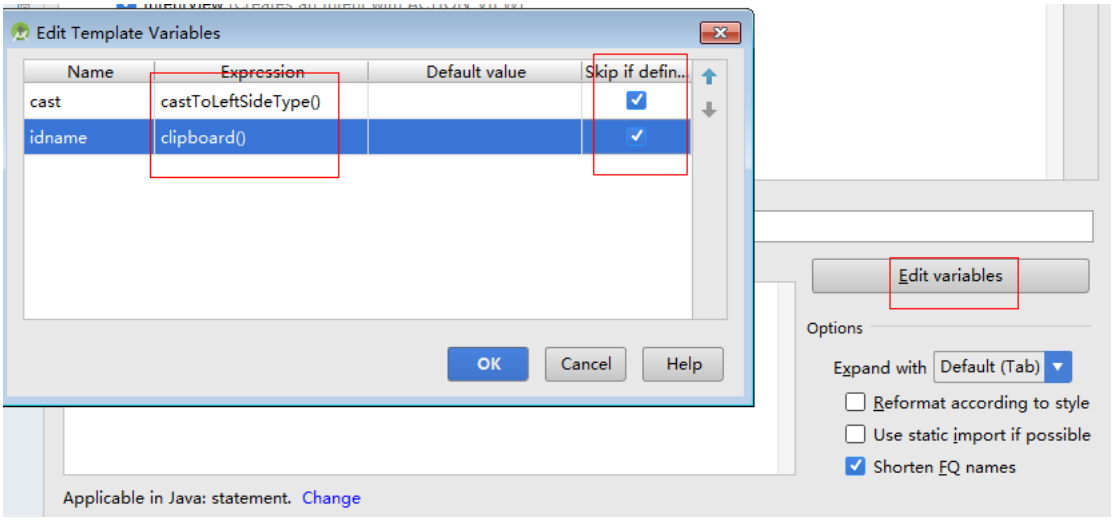
1_添加模板


Settings
Editor › Live Templates
Reset
By default expand with Tab
Appearance & Behavior
Keymap
Editor
General
Colors & Fonts
Code Style
Inspections
File and Code Templates
File Encodings
Live Templates
File Types
Copyright
Emmet
Images
Intentions
Language Injections
Spelling
TODO
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Android
fbi (执行findViewBy()查找视图)
const (Define android style int constant)
fbc (findViewById with cast)
foreach (Create a for each loop)
gone (Set view visibility to GONE)
IntentView (Creates an Intent with ACTION_VIEW)
key (Key for a bundle)
newInstance (create a new Fragment instance with arguments)
noInstance (private empty constructor to prohibit instance creation)
rgS (get a String from resources)
rouiT (runOnUIThread)
sbc (block comment for structuring code)
Abbreviation: fbi
Description: 执行findViewBy()查找视图
Template text:
$idname$ = $cast$findViewById(R.id.$idname$);
Edit variables
Options
Expand with Default (Tab)
Reformat according to style
Use static import if possible
Shorten FQ names
Applicable in Java: statement. Change
OK
Cancel
Apply
Help

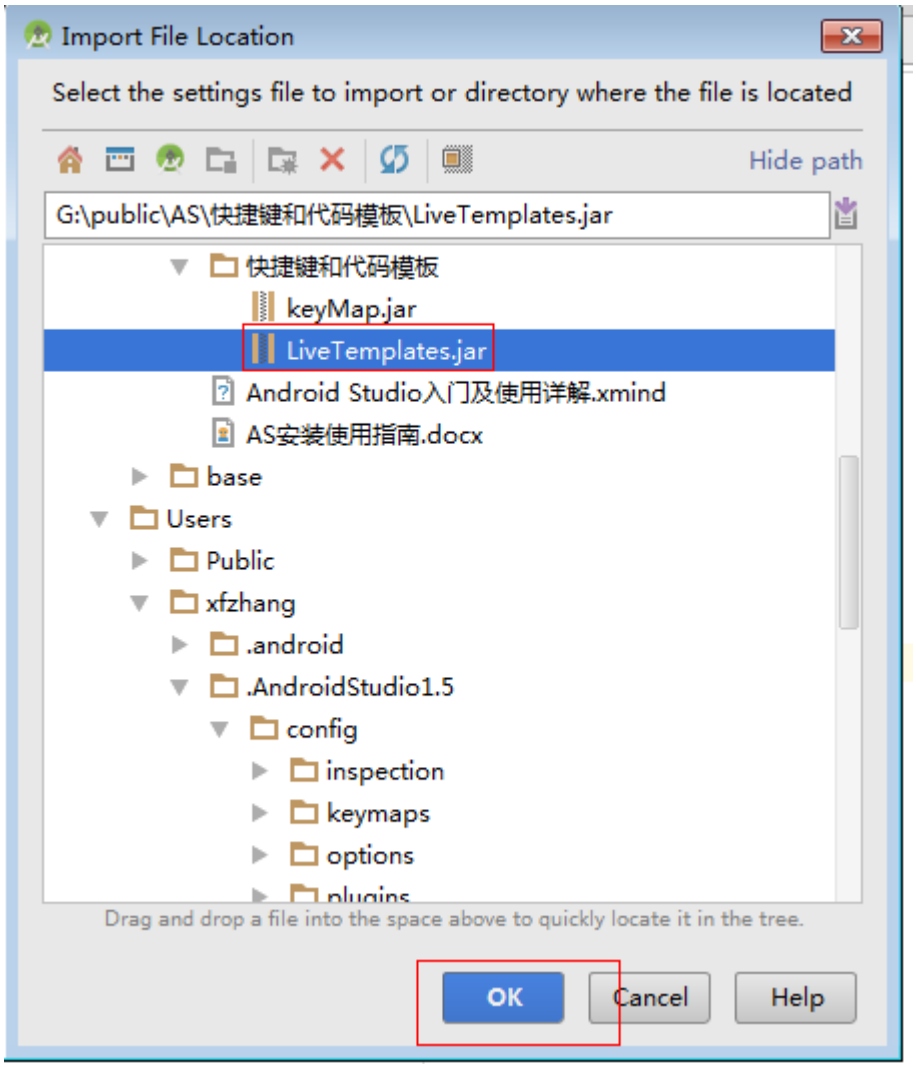
2_指定代码类型



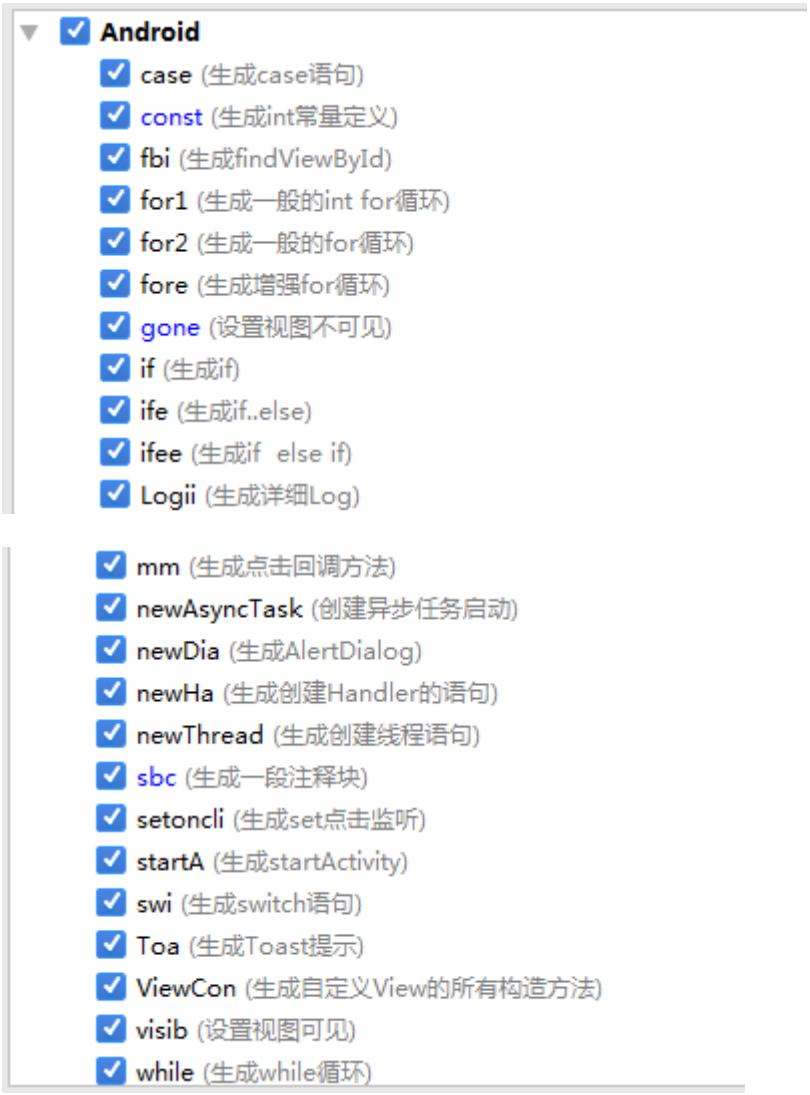
3. 指定模板参数类型



4_导入代码模板


Import File Location
Select the settings file to import or directory where the file is located
Hide path
G:\public\AS\快捷键和代码模板\LiveTemplates.jar
快捷键和代码模板
keyMap.jar
LiveTemplates.jar
Android Studio入门及使用详解.xmind
AS安装使用指南.docx
base
Users
Public
xfzhang
.android
.AndroidStudio1.5
config
inspection
keymaps
options
plugins
Drag and drop a file into the space above to quickly locate it in the tree.
OK
Cancel
Help

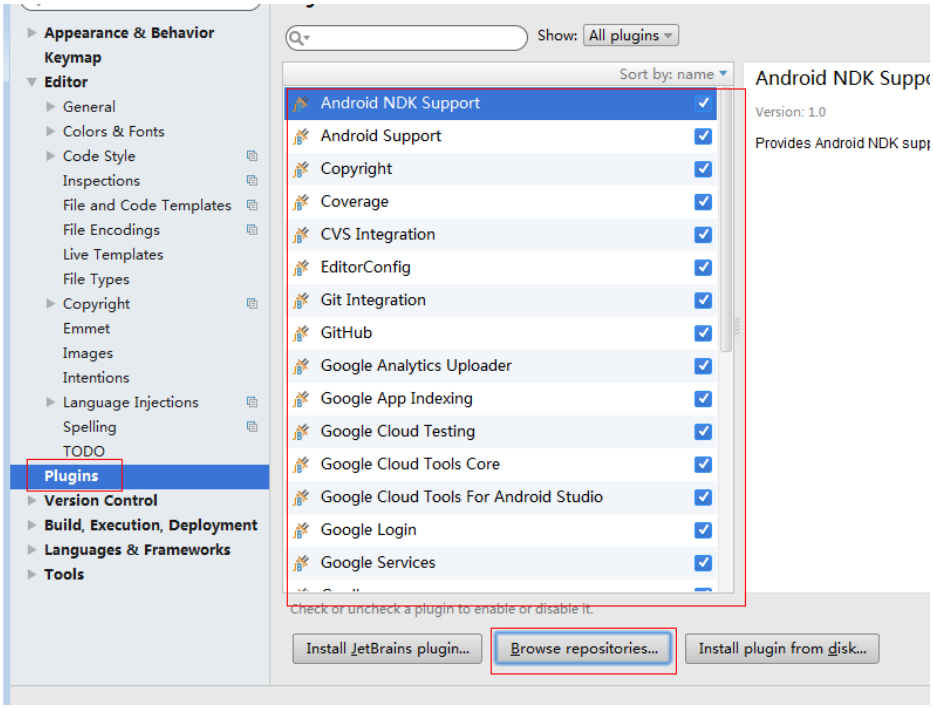
5_常用模板



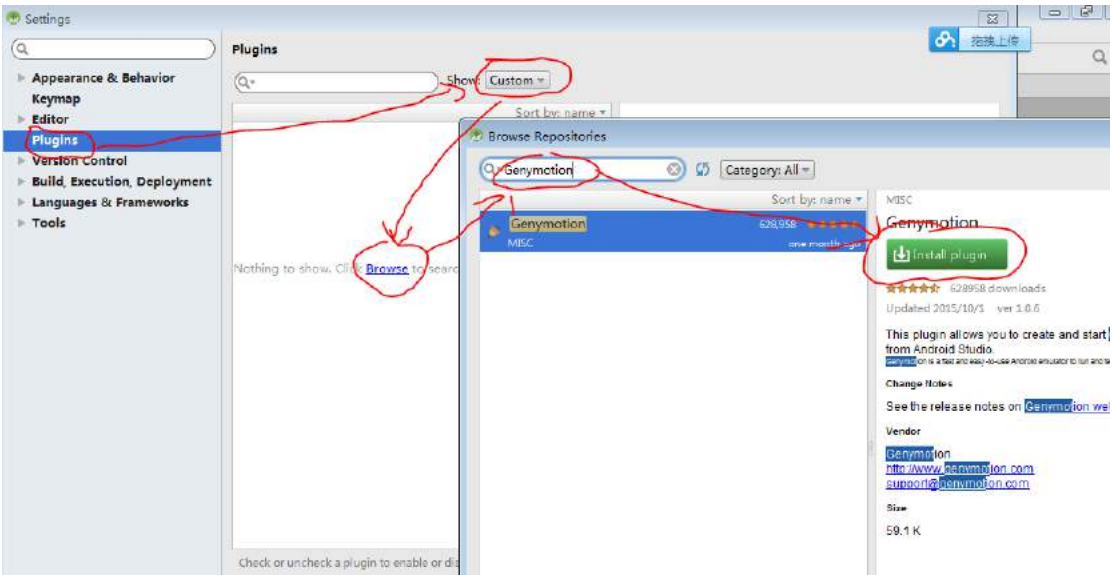
Loge(生成 log.e())

十一． 安装插件

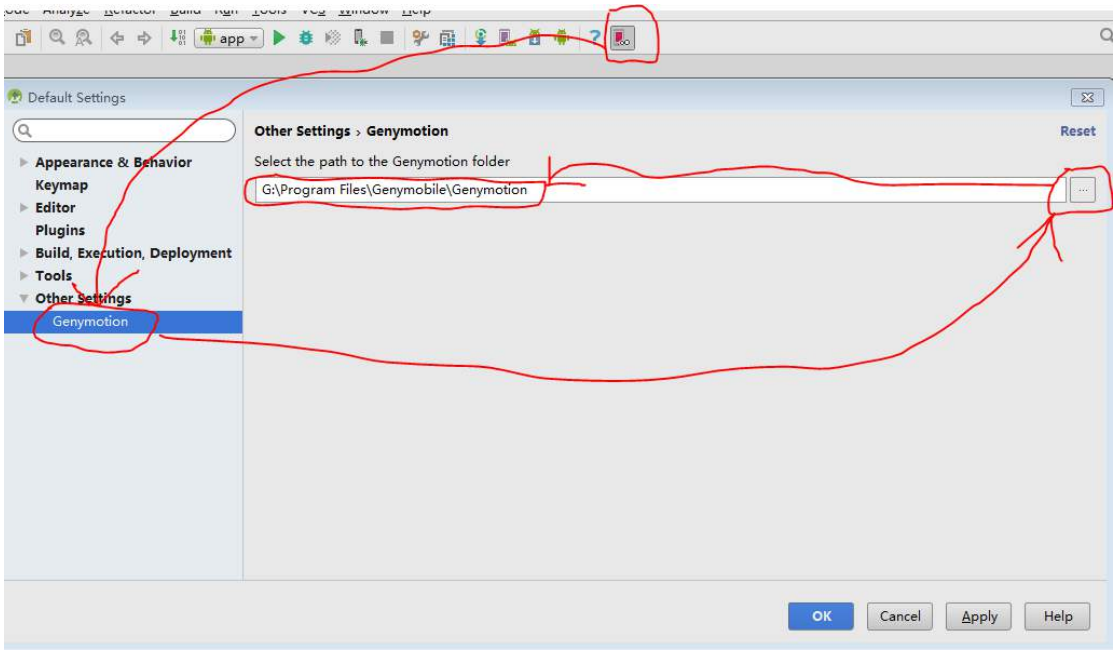
1_查看已有插件



2_查找下载插件



关联 Genymotion:



3.其它第三方插件(需要联网)

常用插件

- GsonFormat
- genymotion
- .ignore
- butterknife

十二. Gradle 技术快速入门

1_介绍

1、 Gradle 是什么？

gradle跟ant/maven一样，是一种依赖管理/自动化构建工具。但是跟ant/maven不一样，它抛弃了基于XML的各种繁琐配置，取而代之的是一种基于Groovy的内部领域特定（DSL）语言，面向Java应用为主。这使得它更加简洁、灵活，更加强大的是，gradle完全兼容maven和ivy。

更多详细介绍可以看它的官网：http://www.gradle.org/

2、为什么要用？

更强大的代码提示与便捷操作;

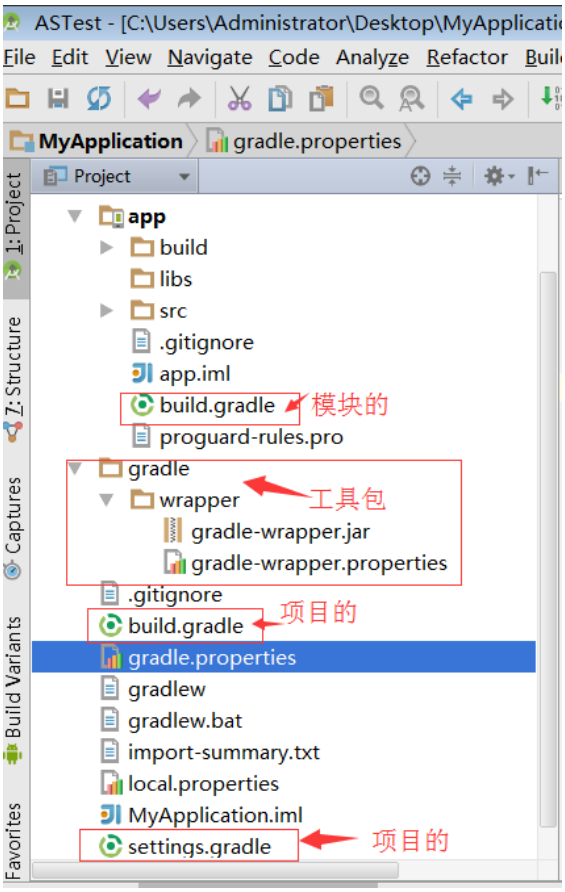
更容易配置，扩展;

更强大的依赖管理，版本控制

更好的 IDE 集成

2_AS 中的 gradle

1、 相关文件



2、 说明

a) project 下的 build.gradle

```
buildscript {
    //指定远程中央仓库
    repositories {
        //jcenter指向的是: https://jcenter.bintray.com/，兼容maven中心仓库，性能更优
        jcenter()
    }

    //指定整个工程的依赖
    dependencies {
        //指定用于编译工具为gradle及其版本号(如果本地没有就会从中央仓库中下载)
        classpath 'com.android.tools.build:gradle:1.5.0'
    }
}

//所有工程及其modle都使用jcenter中央仓库
allprojects {
    repositories {
        jcenter()
    }
}

//执行delete构建时，删除工程下所有构建产生的文件夹
task clean(type: Delete) {
    delete rootProject.buildDir
}
```

b) module 下的 build.gradle

```
// 声明是Android程序
apply plugin: 'com.android.application'

android {
    // 指定编译SDK的版本
    compileSdkVersion 23
    // build tools的版本
    buildToolsVersion "23.0.2"

    defaultConfig {
        // 应用的包名
        applicationId "com.atguigu.astest"
```

```
        //最小版本
        minSdkVersion 15
        //目标版本
        targetSdkVersion 23
        versionCode 1
        //应用版本号
        versionName "1.0"
    }
    buildTypes {
        release {
                // 是否进行代码混淆
            minifyEnabled false
                // 混淆配置文件的位置
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

//包含所有依赖的jar或库
dependencies {
    // 编译libs目录下的所有jar包
    compile fileTree(dir: 'libs', include: ['*.jar'])
    //测试时才编译junit包
    testCompile 'junit:junit:4.12'
    //编译v7包
    compile 'com.android.support:appcompat-v7:23.1.1'
}
```

c) settings.gradle

```
//这个文件是全局的项目配置文件
//指定了当前Project中所有包含的module
include ':app', ':extras:ShimmerAndroid'
```

d) gradle 文件夹及其子文件

```
包含
    gradle-wrapper.jar
    gradle-wrapper.properties
    这两个是 gradle 需要的两个文件，在创建 Project 时自动生成，不用我们修改
```

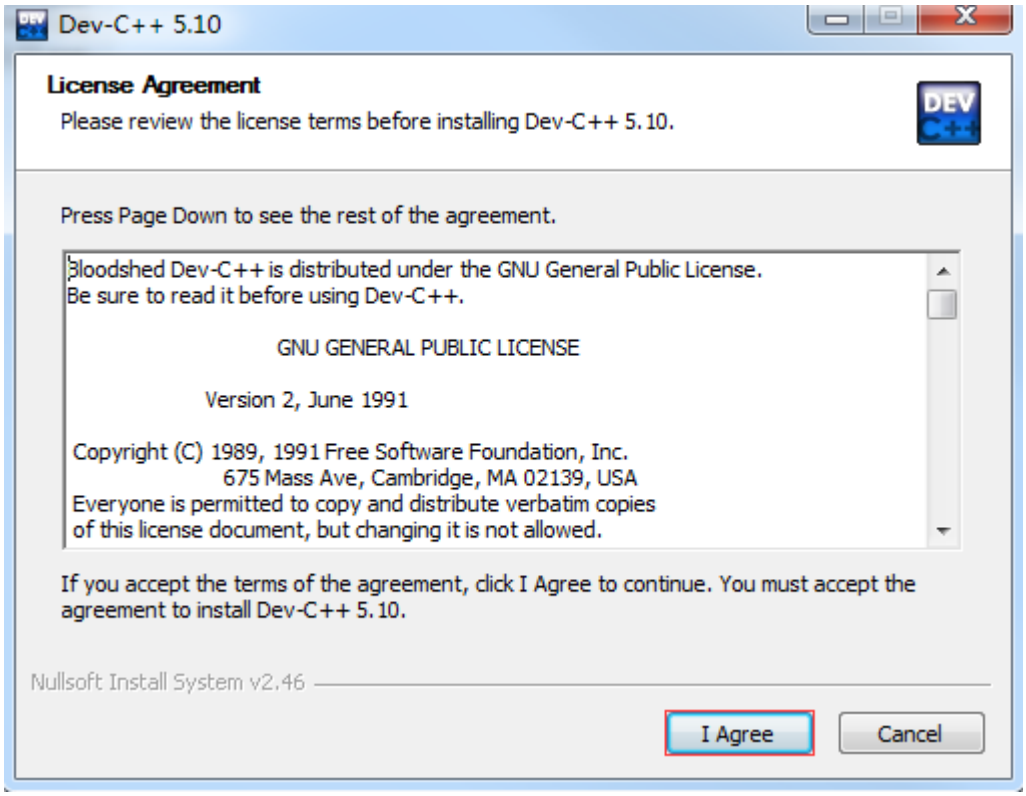
C 语言基础

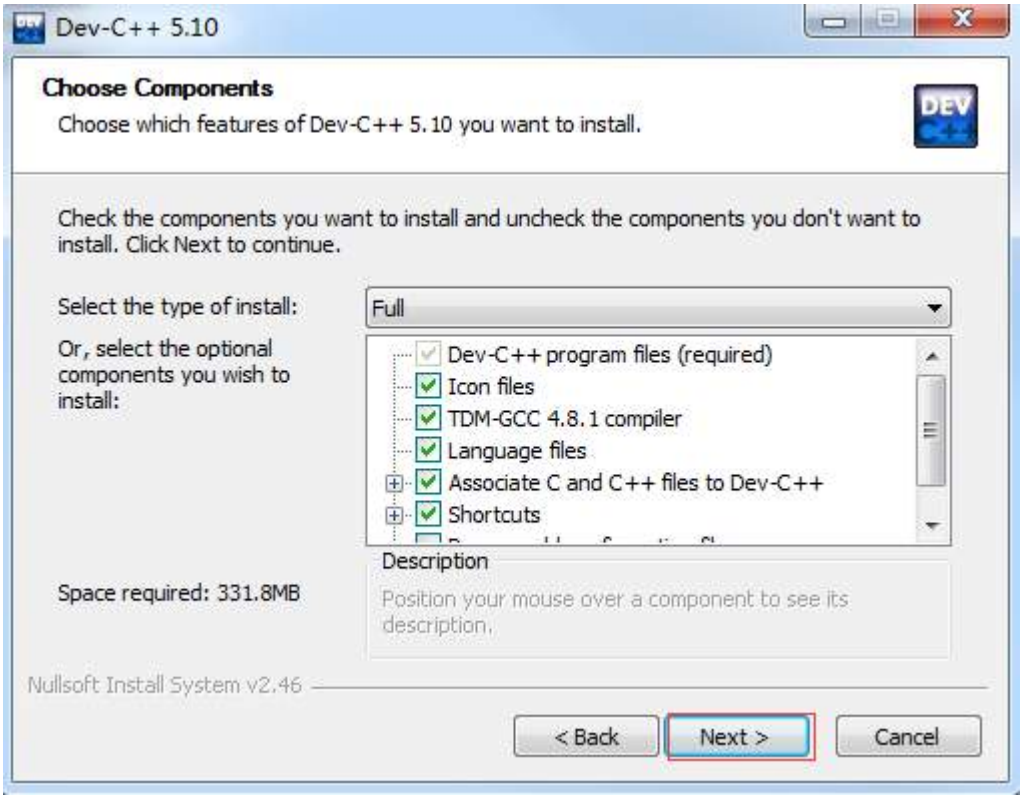
(作者：大海哥)

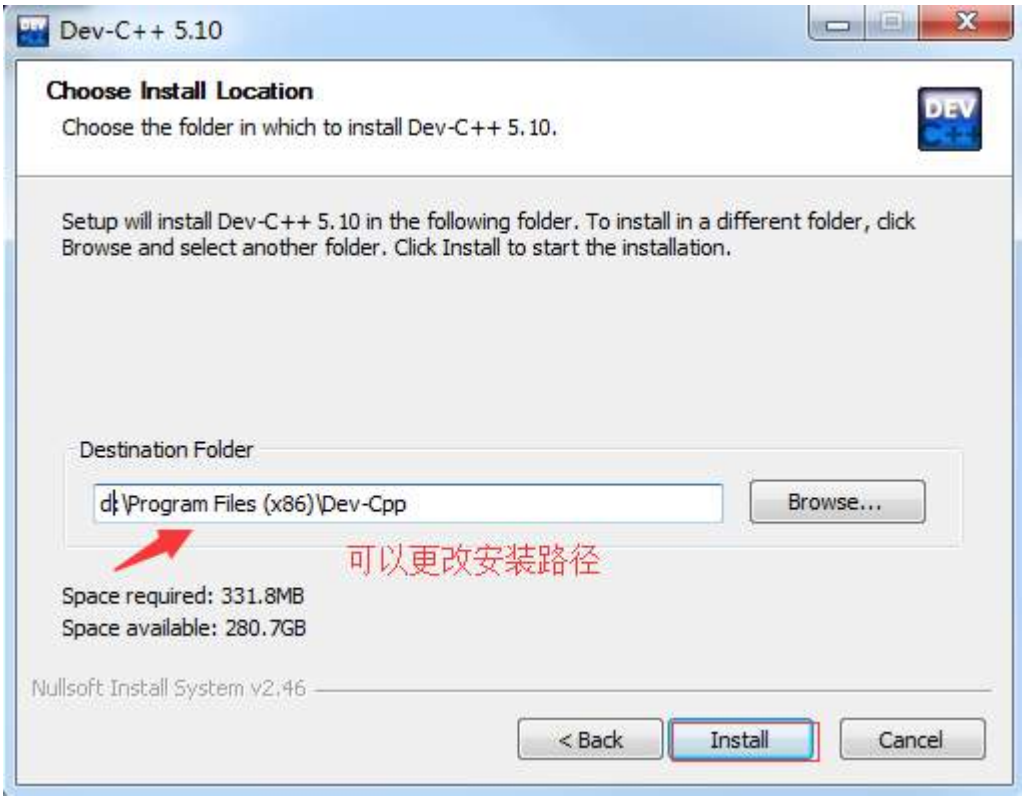
1 快速入门:

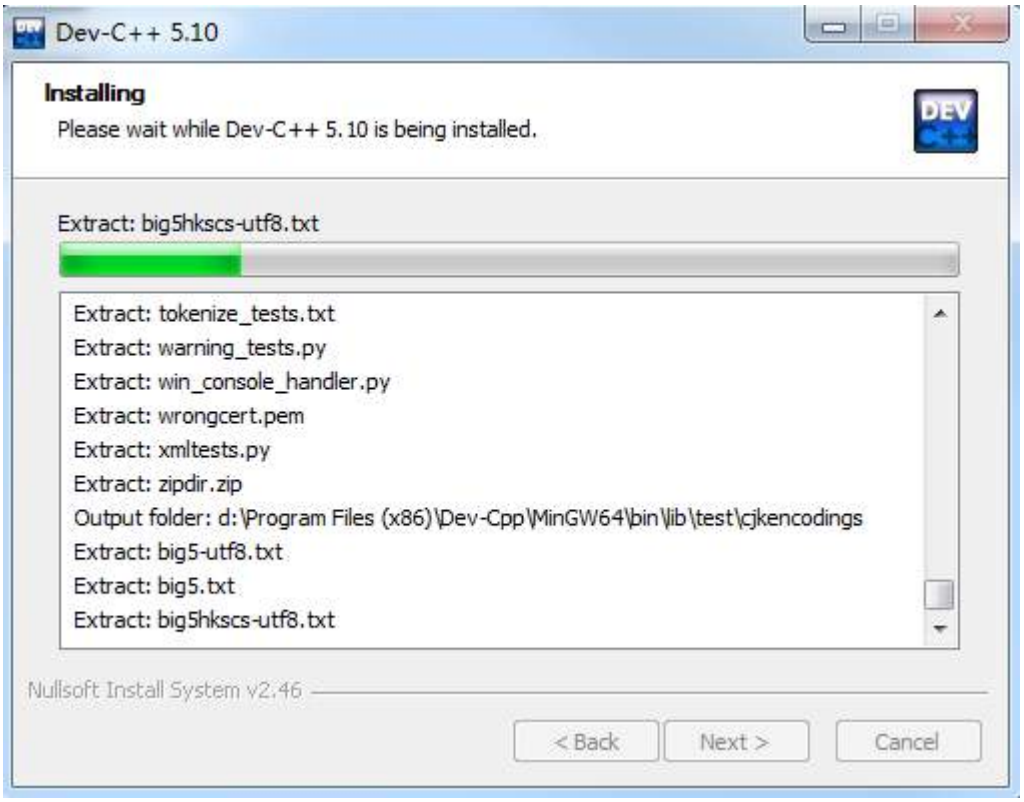
1.1 安装开发工具: Dev-C++

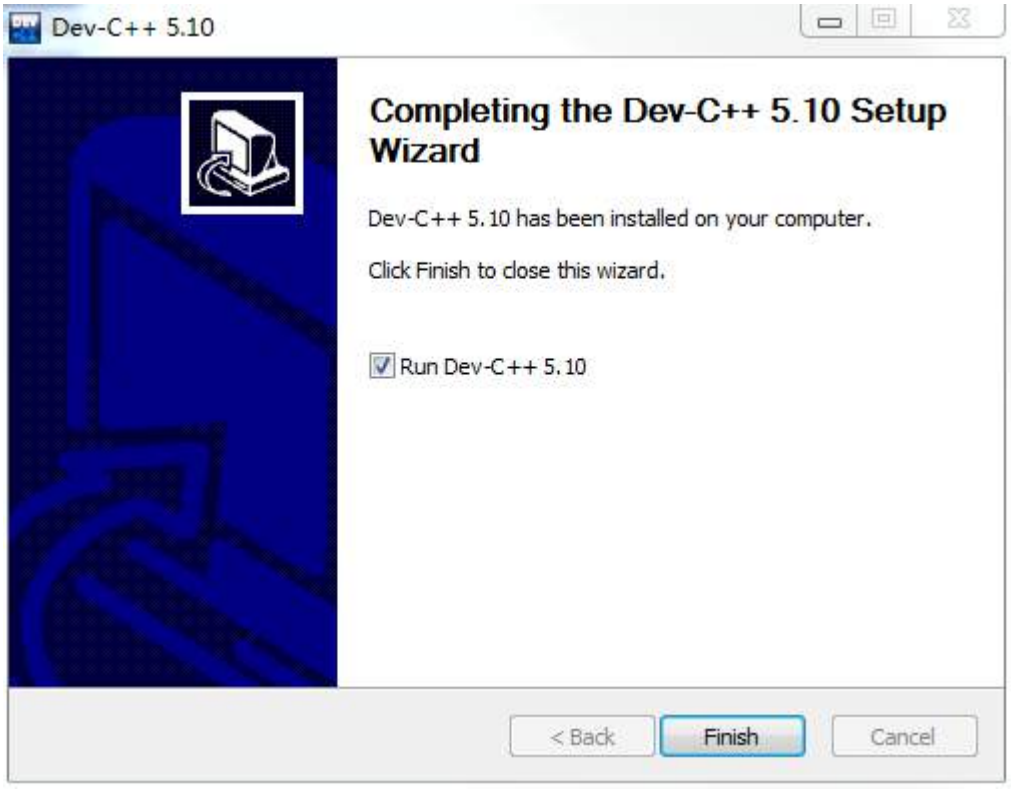
1）点击 Dev-Cpp5.10.exe

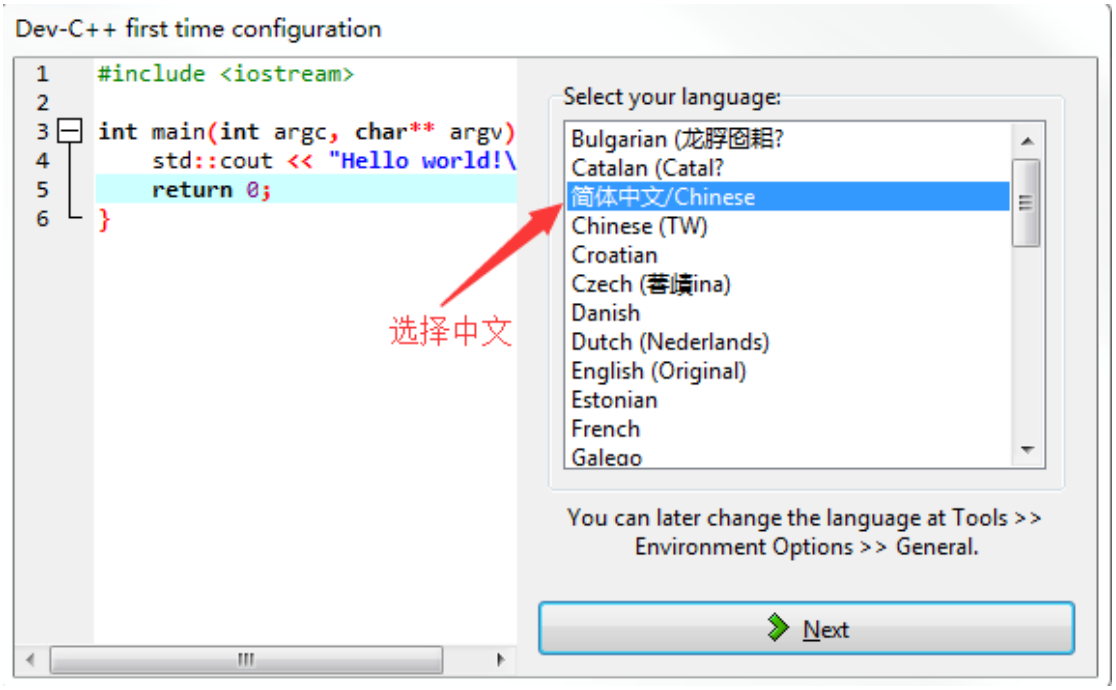


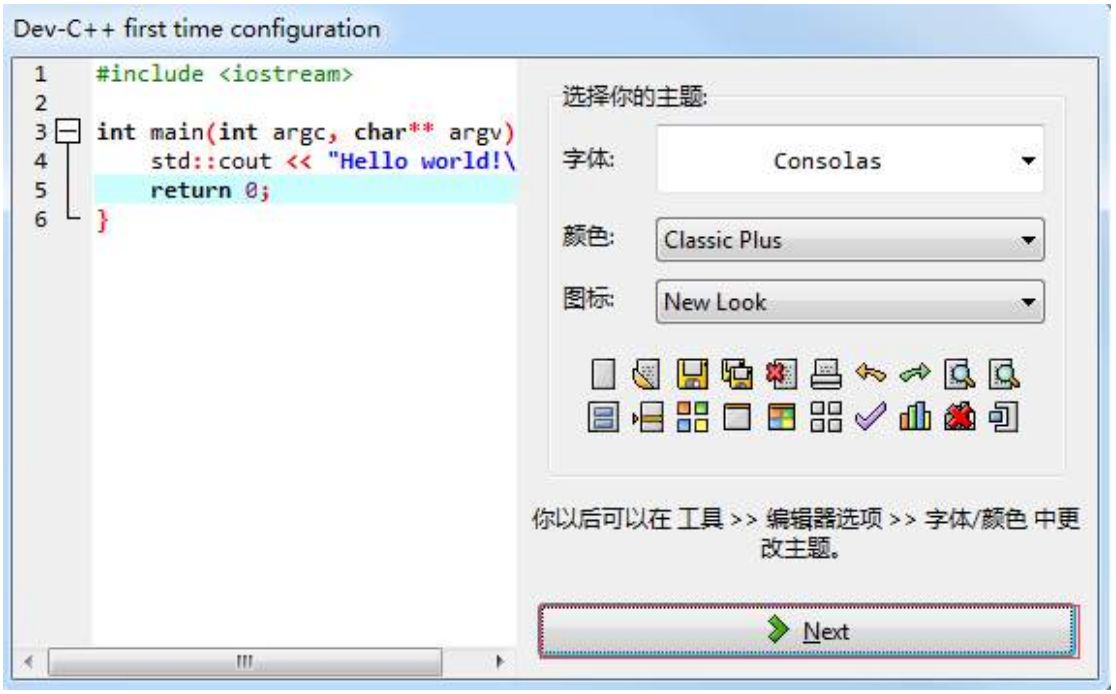

Dev-C++ 5.10
Choose Components
Choose which features of Dev-C++ 5.10 you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select the type of install:
Full
Or, select the optional components you wish to install:
Dev-C++ program files (required)
Icon files
TDM-GCC 4.8.1 compiler
Language files
Associate C and C++ files to Dev-C++
Shortcuts
Description
Position your mouse over a component to see its description.
Space required: 331.8MB
Nullsoft Install System v2.46
< Back
Next >
Cancel

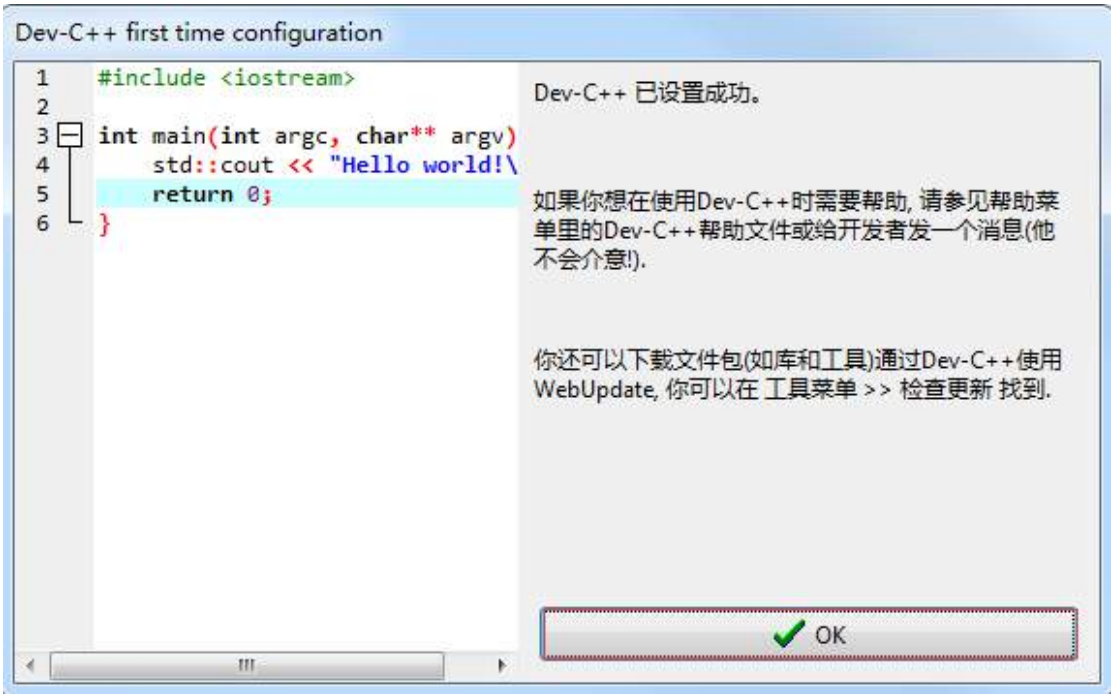

Dev-C++ 5.10
Choose Install Location
Choose the folder in which to install Dev-C++ 5.10.
Setup will install Dev-C++ 5.10 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
d:\Program Files (x86)\Dev-Cpp
Browse...
可以更改安装路径
Space required: 331.8MB
Space available: 280.7GB
Nullsoft Install System v2.46
< Back
Install
Cancel

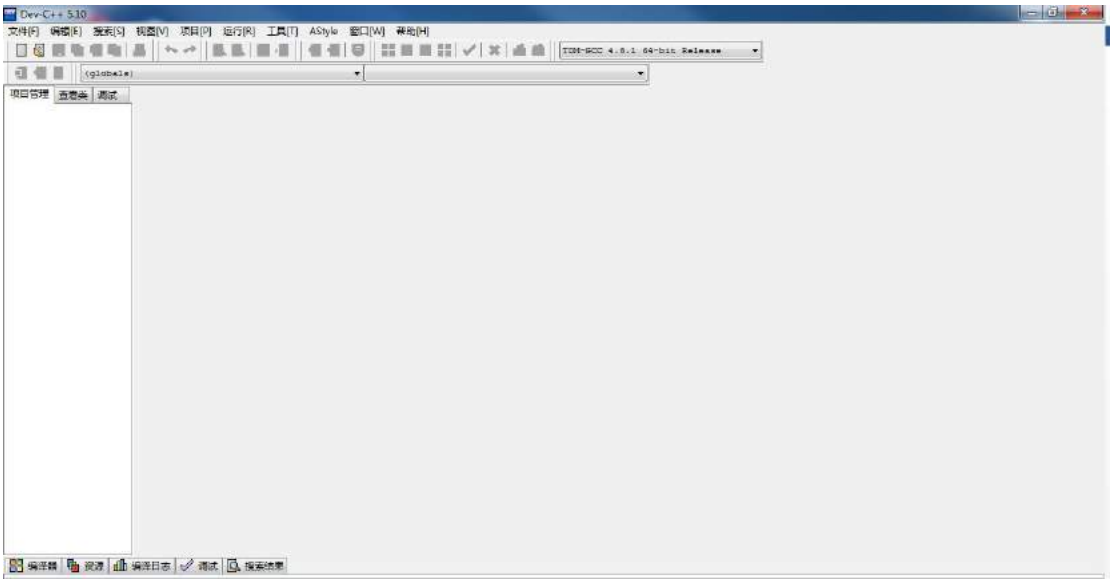

Dev-C++ 5.10
Installing
Please wait while Dev-C++ 5.10 is being installed.
Extract: big5hkscs-utf8.txt
Extract: tokenize_tests.txt
Extract: warning_tests.py
Extract: win_console_handler.py
Extract: wrongcert.pem
Extract: xmltests.py
Extract: zipdir.zip
Output folder: d:\Program Files (x86)\Dev-Cpp\MinGW64\bin\lib\test\cjkencodings
Extract: big5-utf8.txt
Extract: big5.txt
Extract: big5hkscs-utf8.txt
Nullsoft Install System v2.46
< Back
Next >
Cancel


Dev-C++ 5.10
Completing the Dev-C++ 5.10 Setup Wizard
Dev-C++ 5.10 has been installed on your computer.
Click Finish to close this wizard.
Run Dev-C++ 5.10
< Back
Finish
Cancel


Dev-C++ first time configuration
#include <iostream>
int main(int argc, char** argv)
std::cout << "Hello world!\
return 0;
}
选择中文
Select your language:
Bulgarian (龙脬囵耜?
Catalan (Catal?
简体中文/Chinese
Chinese (TW)
Croatian
Czech (菩嘖ina)
Danish
Dutch (Nederlands)
English (Original)
Estonian
French
Galego
You can later change the language at Tools >> Environment Options >> General.
Next


Dev-C++ first time configuration
#include <iostream>
int main(int argc, char** argv)
std::cout << "Hello world!\
return 0;
}
选择你的主题:
字体:
Consolas
颜色:
Classic Plus
图标:
New Look
你以后可以在 工具 >> 编辑器选项 >> 字体/颜色 中更改主题。
Next

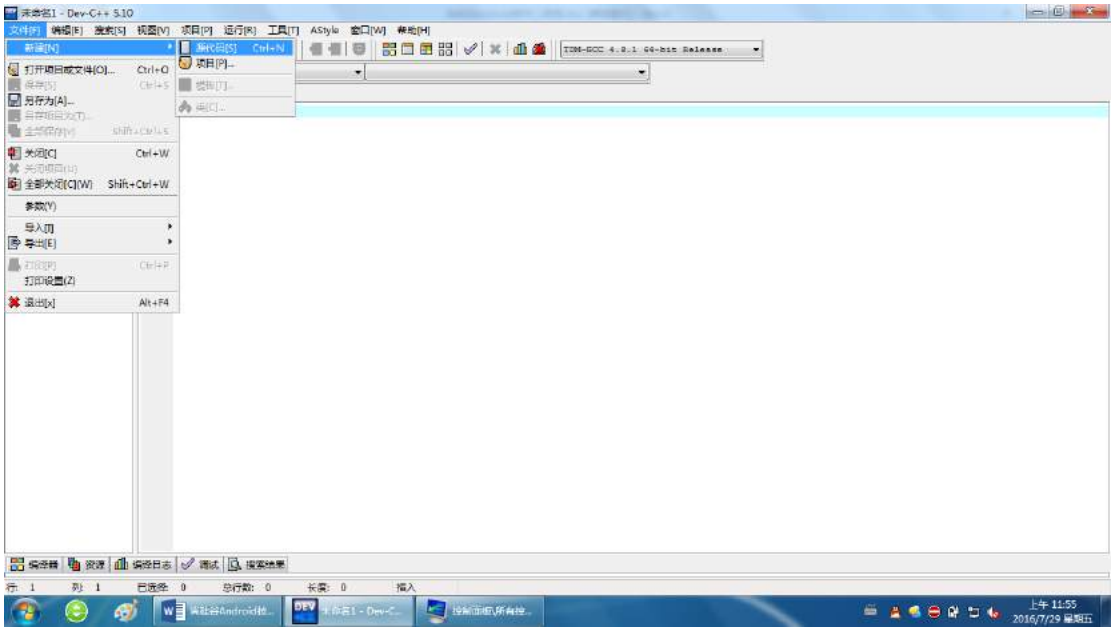




1.2 编写第一个 C 程序

1）新建一个文件

文件->新建->源代码

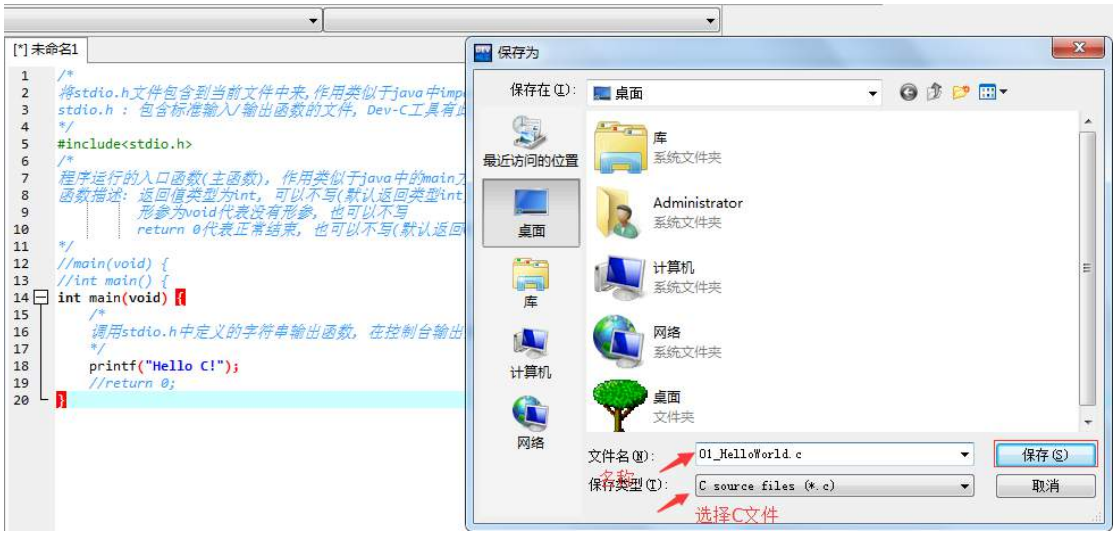


2）编写 hello world 代码

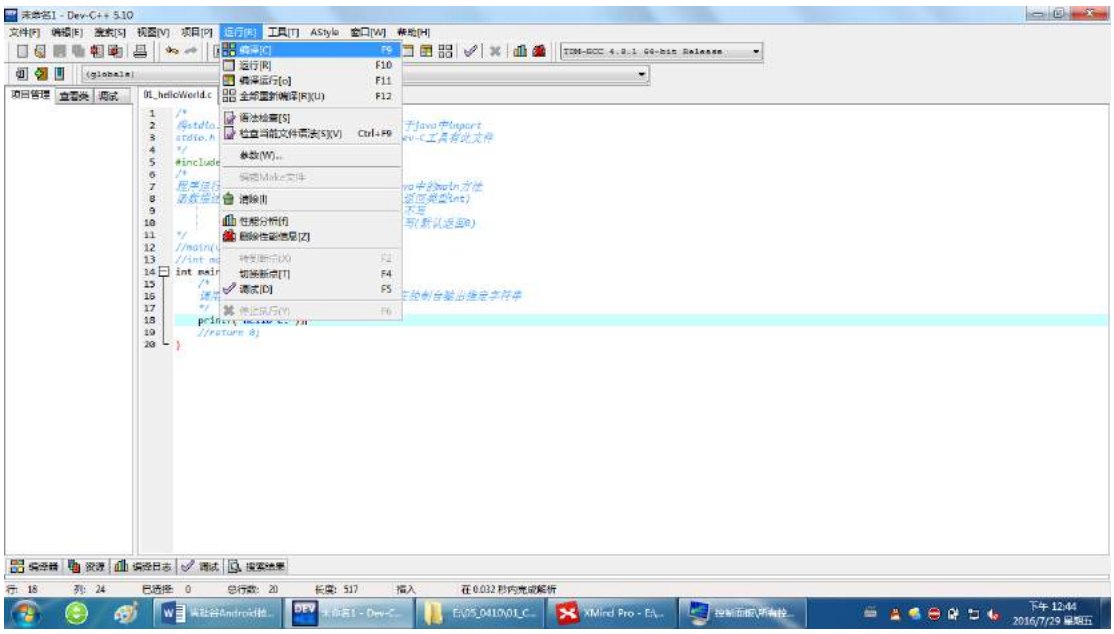
```
/*
将stdio.h文件包含到当前文件中来,作用类似于java中import
stdio.h : 包含标准输入/输出函数的文件, Dev-C工具有此文件
*/
#include<stdio.h>
/*
程序运行的入口函数(主函数), 作用类似于java中的main方法
函数描述: 返回值类型为int, 可以不写(默认返回类型int)
        形参为void代表没有形参, 也可以不写
        return 0代表正常结束, 也可以不写(默认返回0)
*/
//main(void) {
```

```
//int main() {

int main(void) {

	/*

	调用stdio.h中定义的字符串输出函数，在控制台输出指定字符串

	*/

	printf("Hello C!");

	//return 0;

}
```

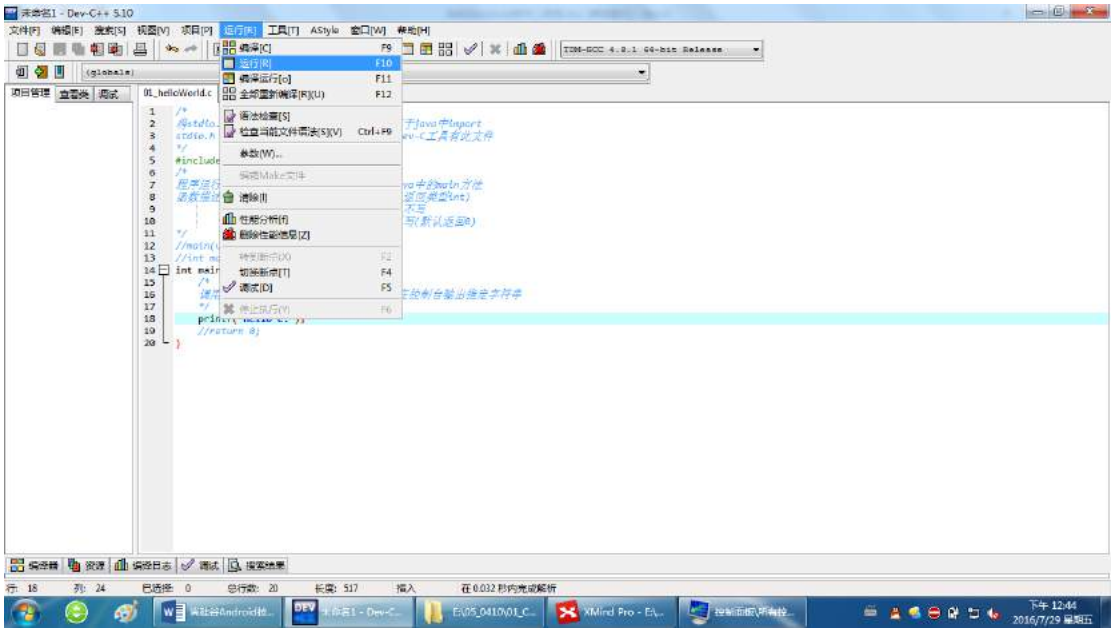
3）保存源代码（ctrl+s）



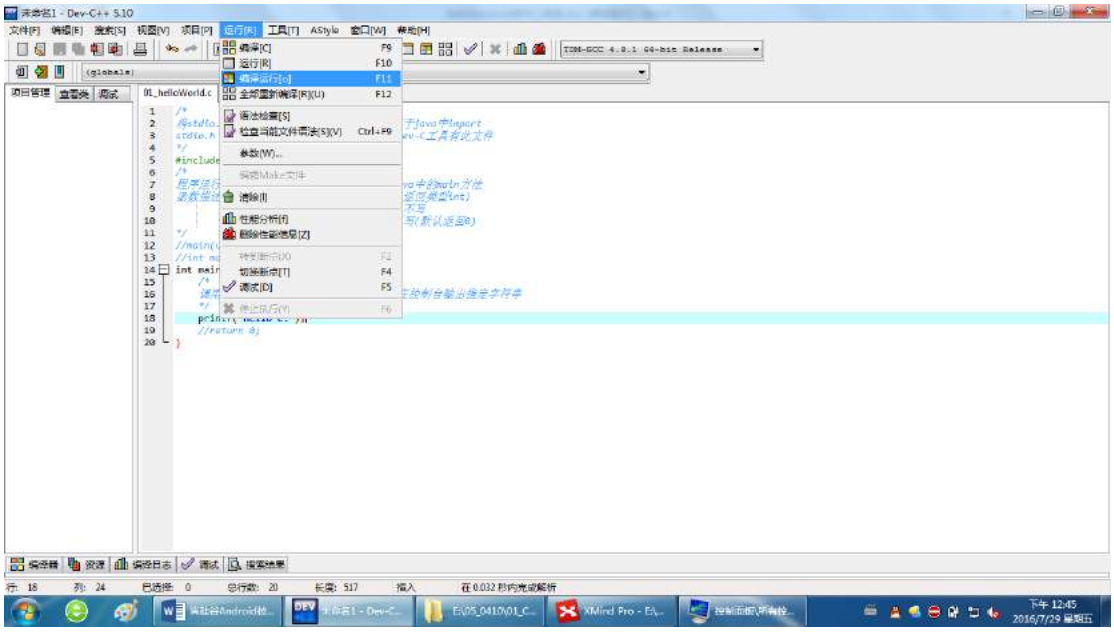
4）编译



5）运行



6）编译并运行（可以代替 编译+运行操作）



2 基本类型

2.1 回顾 Java 的 8 大基本类型

类型	占用字节数	备注
byte	占 1 个字节	整数
short	占 2 个字节	
int	占 4 个字节	
long	占 8 个字节	
float	占 4 个字节	浮点数
double	占 8 个字节	
boolean	占 1 个字节	
char	占 2 个字节	

2.2 C 中的基本数据类型

类型	名称	备注
int	有符号整数	整型数
short	有符号短整数	

long	有符号长整数	
unsigned	无符号 int	
float	单精度小数	浮点数
double	双精度小数	
char	表示字符	

2.3 注意

1）C 语言没有 boolean ,byte,string

2）不同数据类型数据占用的位数

（1）int sizeof(type/var)：得到指定变量/类型占用的字节数

（2）在不同操作系统上可能不一样

a. C 标准中并没有具体给出规定那个基本类型应该是多少字节数，这个与机器、OS、编译器有关

b. 但有如下基本原则:

sizeof(short)<=sizeof(int)

sizeof(int)<=sizeof(long int)

short int至少应为16位（2字节）

long int至少应为32位

char 类型总是一个字节

3）输出不同类型变量数据的占位符

不同数据类型对应不同的占位符

类型	占位符

char	%c
int	%d
short	%hd
long	%ld
unsigned	%u
float	%f
double	%lf
十六进制整数	%x
八进制整数	%o
字符串	%s

4）\n 在字符串代表：换行

2.4 测试

1）测试 1：不同类型变量的定义及赋值

1) 整数类型:

int: 有符号整数

short: 有符号短整数

long: 有符号长整数

unsigned: 无符号 int

2) 浮点数类型

float: 单精度小数

double: 双精度小数

3) 字符型:

char: 表示字符

2）测试 2: 不同类型变量值的输出

%d - int

%hd –short

%ld - long

%u –unsigned

%f - float

%lf – double

%c - char

%x – 十六进制输出

%o - 八进制输出

%s – 字符串

3）测试 3: 不同类型变量的长度

int i = sizeof var //也可以用: sizeof(var)

4）代码实现

```
#include <stdio.h>

int main() {
    //1. 不同类型变量的定义及赋值
    /*
        1). 整数类型:
            int: 有符号整数
            short: 有符号短整数
            long: 有符号长整数
            unsigned: 无符号 int
        2). 浮点数类型:
```

```
                float: 单精度小数
                double: 双精度小数
            3). 字符型:
                char: 表示字符
        */
        int i = 20;
        short s = 2;
        long l = 2020;
        unsigned ui = 30;

        float f = 123.123;
        double d = 123.123123;

        char c = 'a';

        /*
            2. 不同类型变量值的输出
            3. 不同类型变量的长度
        */
        printf("i=%d, 长度为%d\n", i, sizeof(i));
        printf("s=%hd, 长度为%d\n", s, sizeof(s));
        printf("l=%ld, 长度为%d\n", l, sizeof(l));
        printf("ui=%u, 长度为%d\n", ui, sizeof(ui));

        printf("f=%f, 长度为%d\n", f, sizeof(f));
        printf("d=%lf, 长度为%d\n", d, sizeof(d));

        printf("c=%c, 长度为%d \n", c, sizeof(c));
}
```

3 基本语法

3.1 变量

1）全局变量

定义在函数之外的变量

对所有的函数都可见

在程序运行过程一直存在并可用

2）局部变量

定义在函数内部的变量

只在所在的函数内可见，其它函数不能使用

一般的局部变量当函数执行完后自动释放

3.2 运算符

1）定义

用来在内存中进行特定运算的符号

2）分类

基本运算符

=：赋值　　+：相加　　-：相减

*：相乘　　/：相除　　%：求余

其它运算符

sizeof：得到某个变量或某个类型占用的字节数

sizeof i：得到变量i占用的字节数

sizeof(int/i)：得到int类型占用的字节数

++ / --：增量和减量运算符

++i

i++

&：除了是位运算，还可以代表：变量地址

&i： 得到变量i在内存中的地址值

*：除了是相乘运算外，还可以代表：得到指针所对应的值

int * iPoint = &i;

*iPoint ： 返回 iPoint 所指向的 i 的值

3）什么是操作数?

运算符操作的对象，可以是常量或变量

3.3 表达式

1) 表达式：操作数或操作数与运算符的组合

2) 每个表达式都产生一个值

3.4 语句

1）定义

（1）语句以分号结束

（2）一条语句是一条完整的计算机指令

```
普通语句：

i++;

复合语句

if(true) {}
```

2）分类

（1）简单语句

声明语句：int i;

赋值语句：i = 2;

函数调用语句：printf("hello");

空语句：;

（2）复合语句

分支

if 语句

switch 语句

循环

while 语句

for 循环语句

3.5 测试

1）测试 1：全局变量与局部变量

```
int global_i = 2;

int main(void) {

    //测试1: 测试1: 全局变量与局部变量

    printf("全局变量global_i=%d\n",global_i);

    int local_i = 3;

    printf("局部变量local_i=%d\n",local_i);

}
```

2）测试 2: 运算符,表达式,一般语句的基本使用

```
/*
测试2: 运算符,表达式,一般语句的基本使用
*/
int i;
i = 5;
int j = 10;
j = --i;
j = j%i;
printf("i=%d, j=%d", i, j);
```

3）测试 3: while 循环的使用(输出 100 以内所有的奇数)

```
/*
测试3: while循环的使用(输出100以内所有的奇数)
*/
int m = 1;
while(m<100) {
    printf("%d\n", m);
    m += 2;
}
```

4）测试 4: for 循环的使用(输出 100 以内所有的偶数)

```
/*
测试4: for循环的使用(输出100以内所有的偶数)
*/
int k; //必须在for之前声明
for(k=2;k<100;k=k+2) {
	printf("%d\n", k);
}
```

5）测试 5: if 语句使用(判断是否已成年)

```
/*
测试5: if语句使用(判断是否已成年)
*/
int age = 10;
if(age<18) {
	printf("你还未成年\n");
} else {
	printf("你已是成年人啦!\n");
}
//进行真假判断时: 0为假, 其它为真
i =0;
```

```
i = 2;

i = -2;

if(i) {

    printf("条件成立\n");

} else {

    printf("条件不成立\n");

}
```

4 函数

4.1 理解

1）what?

用于完成特定任务(功能)的程序代码的自包含单元

2）why?

复用代码

模块化代码，便于阅读

3）How?

（1）如何定义函数?

```
返回值类型  函数名(形参列表) {

    函数体语句
```

```
}
```

（2）如何调用函数?

```
函数名(实参列表);
```

（3）如何声明函数?

```
在所有函数定义之前对后面定义的函数进行声明

格式: 返回值类型  函数名(形参类型列表);

作用: 可以让函数的定义在main函数之后

        统一声明, 便于阅读
```

4.2 几个重要的函数

1）字符串的输出输入函数

printf(string, value...)

scanf(string, 地址值)

2）主函数

main(){}

int main(void){}

int main(int argc,char *argv[])

4.3 测试

1）测试 1：函数的定义和调用(得到 2 个整数的和)

```
/*
测试1: 函数的定义和调用(得到2个整数的和)
*/
void test1() {
	int t = sum(3,4);
	printf("sum=%d\n", t);
}
//得到2个整数的和
int sum(int i, int j) {
	int sum = i + j;
	return sum;
}
```

2）测试 2：定义函数输出九九乘法表

```
/*
测试2: 定义函数输出九九乘法表
1*1=1
1*2=2 2*2=4
```

```
1*3=3 2*3=6 3*3=9

1*4=4 2*4=8 3*4=12 4*4=16

1*5=5 2*5=10 3*5=15 4*5=20 5*5=25

1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36

1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49

1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64

1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

*/

void test2() {

	int left, right;

	for(right=1;right<=9;right++) {

		for(left=1;left<=right;left++) {

			//输出: left*right=乘积

			int product = left* right;

			printf("%d*%d=%d ", left, right, product);

		}

		printf("\n");

	}

}
```

3）测试 3：使用 printf()和 scanf()

```
/*
测试3: 使用printf()和scanf()
    提示用户输入2个两个数，输出它们的和
*/
void test3() {
    int i;
    int j;
    printf("请输入第一个整数: ");
    scanf("%d", &i);
    printf("请输入第二个整数: ");
    scanf("%d", &j);
    printf("这两个数的和为: %d\n", i+j);
}
```

4）测试 4：测试函数声明

```
/*
测试4: 函数声明
*/
int test4(int, char);
```

```
int test4(int i, char c) {
	printf("测试函数声明: i=%d, c=%c", i, c);
	return 0;
}
```

5 指针

5.1 理解

1）what?

保存地址值的变量,称为指针变量,即为指针

2）why?

可以更方便的对内存中的数据进行操作

3）How?

*：定义指针变量

得到指针对应的值

&：得到变量的地址

5.2 基本使用

1）指针的定义与赋值

2）指针的传递

3）区别指针变量与一般变量

5.3 测试

1）测试 1：指针的定义与赋值，读取数据

```
/*
测试1: 指针的定义与赋值, 读取数据
*/
void test1() {
	//定义指针
	int * iPoint;
	//一般变量
	int i = 3;
	//取得i的地址值并赋值给iPoint
	iPoint = &i;

	//打印指针本身的值
	printf("iPoint=%d\n", iPoint);
	//打印指针对应的值
	printf("*iPoint=%d\n", *iPoint);
	//打印指针的地址值
	printf("&iPoint=%d\n", &iPoint);

	printf("/**********************************************/\n");
```

```
    //修改i的值
    i = 5;
    printf("iPoint=%d\n", iPoint);//不变
    printf("*iPoint=%d\n", *iPoint);//5
    printf("&iPoint=%d\n", &iPoint); //不变

    printf("/*********************************************/\n");
    //修改*iPoint的值
    *iPoint = 10;
    printf("iPoint=%d\n", iPoint);//不变
    printf("*iPoint=%d\n", *iPoint);//10
    printf("&iPoint=%d\n", &iPoint); //不变

    printf("/*********************************************/\n");
    //修改iPoint的值
    iPoint = 100;
    printf("iPoint=%d\n", iPoint);//100
    printf("&iPoint=%d\n", &iPoint); //不变
    //printf("*iPoint=%d\n", *iPoint);//不能执行,找不到对应的地址
}
```

2）测试 2：使用指针实现交换两个数的函数

```
/*
测试2: 使用指针实现交换两个数的函数
*/
void test2() {
	int i = 2;
	int j = 4;
	printf("交换前i=%d, j=%d\n", i, j);
	exchange(&i, &j);
	printf("交换后i=%d, j=%d\n", i, j);
}

void exchange(int *i, int *j) {
	int temp;
	temp = *i;
	*i = *j;
	*j = temp;
}
```

3）测试 3：多级指针

```
/*
```

```
测试3: 多级指针

*/

void test3() {

	//一级指针

	int * iPoint;

	int i = 10;

	iPoint = &i;

	printf("iPoint=%d\n", iPoint);//地址值

	printf("*iPoint=%d\n", *iPoint);//10


	//二级指针

	int ** iPoint2;

	iPoint2 = &iPoint;

	printf("iPoint2=%d\n", iPoint2);//地址值

	printf("*iPoint2=%d\n", *iPoint2);//地址值  iPoint

	printf("**iPoint2=%d\n", **iPoint2);//10


	//N级指针以此类推

}
```

6 数组

6.1 理解

1）what?

由一系列类型相同的元素组成

数组中的所有元素保存在一大块连续的内存中

2）why?

对多个数据进行统一管理

3）how?

（1）数组的声明与初始化

元素类型 变量名[元素个数] = {元素1, 元素2, ..元素n}

int iArr[3]

int iArr[] = {1, 3, 4, 5};

（2）数组元素的读取和赋值

数组变量[index]

int i = iArr[1]

iArr[1] = 8

6.2 基本使用

1）声明

int iArray[3];

2）初始化

iArray = {2, 4, 6}

3）数组元素的读取和赋值

iArray[2]

4）遍历

见测试。

6.3 数组与指针

1）数组变量本质是指针．它保存的是数组第一个元素的地址：*iArr

2）指针+n：指向原地址向右移动 n 个元素字节大小的空间

i+1

*(iArr+1)

6.4 测试

1）测试 1：数组的基本使用

```
/*
测试1: 数组的基本使用
*/
void test1() {
    //定义数组变量, 并初始化
    //int iArr[3];
    int iArr[] = {3, 2, 1};
```

```
    //设置元素的值
    iArr[1] = 10;
    //读取元素的值
    int i;
    int length = sizeof(iArr)/sizeof(iArr[0]);
    printf("iArr的长度为: %d\n", length);
    for(i=0;i<length;i++) {
        printf("iArr[%d]=%d\n", i, iArr[i]);
    }
}
```

2）测试 2：数组与指针

```
/*
测试2: 数组与指针
    数组变量本质是指针. 它保存的是数组第一个元素的地址
    指针+n: 指向原地址向右移动n个元素字节大小的空间
*/
void test2() {
    int iArr[] = {3, 2, 1};
    //输出数组的第1个元素
    printf("iArr[0]=%d\n", iArr[0]);
```

```
    printf("*iArr=%d\n", *iArr);

    //输出数组的第3个元素

    printf("iArr[2]=%d\n", iArr[2]);

    printf("*(iArr+2)=%d\n", *(iArr+2));

}
```

3）测试 3：输出用户输入数组

```
/*

测试3: 输出用户输入数组

    //提示用户输入数组的长度

    //提示用户输入数组中一个一个的元素

    //输出用户输入的所有数据

*/

void test3() {

    //提示用户输入数组的长度

    printf("请输入数组元素的个数:\n");

    int length;

    scanf("%d", &length);

    //提示用户输入数组中一个一个的元素

        /*定义指定长度的数组*/

    int iArr[length];
```

```
    int i;
    for(i=0;i<length;i++) {
        printf("请输入第%d个元素的值:\n", i+1);
        /*读取用户输入的值保存到指定内存中*/
        //scanf("%d", &iArr[i]);
        scanf("%d", iArr+i);
    }
    //输出用户输入的所有数据
    for(i=0;i<length;i++) {
        //printf("iArr[%d]=%d\n", i, iArr[i]);
        printf("iArr[%d]=%d\n", i, *(iArr+i));
    }
}
```

7 其他

7.1 函数指针

1）理解

当一个指针指向的是一个函数时称为函数指针

2）定义

返回值类型 (*指针名)(形参类型列表);

3）使用

函数指针名 = 函数名　　funP = fun

作为实参传递给调用的函数

4）测试：函数指针作为实参传递

```
#include <stdio.h>

/*

1. 理解

    当一个指针指向的是一个函数时称为函数指针

2. 定义

    返回值类型 (*指针名)(形参类型列表);

3. 使用

    1). 函数指针名=函数名

    2). 执行函数指针函数

    3). 作为实参传递给调用的函数

*/

int sum(int i, int j) {

    return i+j;

}


int multiply(int i, int j) {

    return i*j;
```

```
}


//在函数形参中使用函数指针
int compute(int (*fp)(int, int), int m, int n) {
	return fp(m, n);
}


main() {
	int i1 = 3;
	int i2 = 4;

	//定义函数指针
	int (*fPoint)(int, int);

	//将函数赋值给函数指针
	fPoint = sum;

	int result = compute(fPoint, i1, i2);
	printf("result=%d\n", result);

	fPoint = multiply;
	result = compute(fPoint, i1, i2);
```

```
    printf("result=%d\n", result);

}
```

7.2 结构体

1）理解

（1）在程序中有时需要操作一个复合型(包含多个属性)对象

（2）比如：需要存储书的相关信息，商品相关信息

（3）在 C 中可以使用 struct 来实现

（4）它类似于 Java 中的类

2）使用

（1）结构声明

```
struct s_name {
    一般变量定义;
    一般指针定义;
    函数指针定义;
};
```

（2）结构变量

声明：

```
struct s_name v_name;
```

赋值：

```
v_name.property_name = value;
v_name.function_point = function;
```

使用：

v_name.property_name;

v_name.function_point();

（3）高级使用

使用->

3）测试

```
#include <stdio.h>

/*

1. 定义结构:

    struct s_name {

        变量定义;

        函数指针定义;

    };

2. 定义结构变量

    struct s_name v_name;

3. 初始化结构变量

    v_name.property_name = value;

    v_name.function_point = function;

4. 使用结构变量

    v_name.property_name;

    v_name.function_point();
```

```
5. 定义结构体指针
    struct s_name * point_name;
6. 使用结构体指针
    (*point_name).property_name;
        也可以简化为:  point_name->property_name;
    (*point_name).function_point();
        也可以简化为:  point_name->function_point();
*/

//1. 定义结构:
struct book {
    char * title;//一般指针
    char * author;//一般指针
    double price;//一般变量
    void (*piPoint)(int); //函数指针
};

void test(int i) {
    printf("test() i=%d\n", i);
}
```

```
main() {
    //2. 定义结构变量
    struct book b;

    //3. 初始化结构变量
    b.title = "C语言";
    b.author = "张某某";
    b.price = 30;
    b.piPoint = test;
    //4. 使用结构变量
    printf("你指定的书名为%s, 作者为%s, 售价为%lf\n", b.title, b.author, b.price);
    b.piPoint(1111);

    //5. 定义结构体指针
    struct book * bPoint = &b;

    //6. 使用结构体指针
        //得到价格
    double p1 = (*bPoint).price;
    double p2 = bPoint->price;
    printf("p1=%lf, p2=%lf\n", p1, p2);
        //调用函数指针
```

```
    (*bPoint).piPoint(5);

    bPoint->piPoint(5);

}
```

7.3 字符串

1）由多个字符组成的序列

2）它的最后一位是隐藏的空字符

3）C 中并没有定义 string 这个类型

4）表示字符串的 3 种方式

"" ：常量字符串

char 数组

char 类型指针

5）测试：字符串综合测试

```
/*

测试1: 字符串

*/

void test1() {

    printf("abc的长度为%d\n", sizeof("abc"));//4个字节(后面有一个空字符)
```

```
    char * cs = "abc";

    printf("cs=%s\n", cs);



    char cs2[] = "efg";

    printf("cs2=%s\n", cs2);



    char cs3[3];

    printf("请输入名称: ");

    scanf("%s", cs3);

    printf("cs3=%s\n", cs3);

}
```

7.4 预处理指令

1）理解

在程序的顶部总会有以#开头的声明，它们就是预处理指令

在 C 编译器执行之前，预编译器会对预处理指令进行处理

2）常用的

（1）#include

包含头文件

#include <xxx.h>例 #include <stdlib.h>

#inclue "头文件名"

（2）#define

定义全局常量

#define NAME value 例 #define SIZE 5

3）测试

```
#include <stdio.h>



#define NAME "宋老师"

#define COUNT 20



main(){

    printf("技术最牛的老湿：%s\n",NAME);



    printf("尚硅谷美女老湿个数：%d",COUNT);

}
```

7.5 typedef

1）what?

用来给某个类型定义别名

语法: typedef 类型名 别名

2) why?

简化编码

3) how?

（1）给内置类型定义别名

```
typedef char* string;

string s = "abc";
```

（2）给自定义类型定义别名

```
typedef struct book* bp;
```

4）测试

```
#include<stdio.h>

#include<stdlib.h>

/*

1. typedef用来给某个类型定义别名

2. 语法: typedef  类型名  别名

3. 使用:

    给内置类型定义别名

    给自定义类型(struct)定义别名

*/



//给内置类型定义别名
```

```
typedef char* pstr;


//给自定义类型(struct)定义别名
typedef struct book* bookPoint;


struct book {
	pstr name;
	double price;
};


//给自定义类型(struct)定义别名
typedef struct {
	pstr n;
	double p;
} shop;


int main() {


	pstr ps = "abc";
	printf("ps=%s", ps);


	struct book b1 = {
```

```
        "C语言精华",
        100
    };

    bookPoint bp = &b1;
    printf("name=%s\n", bp->name);

    shop s;
    s.n = "好吃的";
    s.p = 1000;
    printf("n=%s,p=%lf\n", s.n, s.p);

    return 0;
}
```

7.6 内存分配

1）理解

一个变量的内存什么时候分配和释放?

我们是否可以自己手动分配(申请)内存?

2）分类

1）静态自动分配

（1）特点

分配：在程序(函数)开始执行之前

释放：整个程序结束

（2）哪些

全局变量

2）动态自动分配

（1）特点

分配：程序运行到所在语句

释放：程序运行出了对应的代码块

（2）哪些

局部变量(包括形参)

3）动态手动分配

（1）特点

分配：执行 malloc(size)函数

释放：执行 free()

（2）哪些

执行 malloc()返回赋值的变量

3）测试 1：静态分配

```
/*

测试1: 静态分配

*/
```

```
int i = 3;

void test1() {

	printf("test1() i=%d\n", i);

	add();

	printf("test1()2 i=%d\n", i);

}

void add() {

	printf("add() i=%d\n", i);

	i++;

}
```

4）测试 2：动态自动分配

```
void test2() {



	int i = 2;

	int * iPoint = &i;

	printf("test2_a before *iPoint=%d\n", *iPoint); //2

	test2_a(&iPoint);

	Sleep(1); //Sleep(毫秒)

	printf("test2_a after *iPoint=%d\n", *iPoint);   //20

	printf("test2_a after *iPoint=%d\n", *iPoint);   //0
```

```
}


void test2_a(int ** iPoint) {
	int i = 20;
	*iPoint = &i;
	printf("test2_a **iPoint=%d\n", **iPoint);//20
}
```

5）测试 3：动态手动分配

```
/*
测试3: 动态手动分配
*/
void test3() {
	int i = 3;
	int * iPoint = &i;
	printf("test3_a before *iPoint=%d\n", *iPoint);
	test3_a(&iPoint);
	Sleep(1);
	printf("test3_a after *iPoint=%d\n", *iPoint);
	printf("test3_a after *iPoint=%d\n", *iPoint);
}
```

```
void test3_a(int ** iPoint) {
    int i = 30;
    int * temP = malloc(sizeof(int)); //手动动态分配int类型大小的内存，并返回其地址值
    *temP = i;
    *iPoint = temP;
    printf("test3_a **iPoint=%d\n", **iPoint);
    //free(temP);//释放指定指针对应地址的内存
}
```

7.7 枚举

1）理解

枚举类型声明代表整数常量的符号名称

使用关键字：enum

枚举变量本质就是 int 类型

目的：提高程序的可读性

2）测试

```
#include<stdio.h>
/*
测试: 使用枚举操作星期数据
    提示用户输入今天星期的数值(1-7)， 输出今天是星期几
```

```
*/
enum DAY{
        MON=1, TUE=2, WED=3, THU=4, FRI=5, SAT=6, SUN=7
        //MON=1, TUE, WED, THU, FRI, SAT, SUN
};


main() {


    printf("请输入今天星期的数值(1-7): ");
    enum DAY today;
    scanf("%d", &today);
    switch(today) {
        case MON :
            printf("你指定的是星期一");
            break;
        case TUE :
            printf("你指定的是星期二");
            break;
        case WED :
            printf("你指定的是星期三");
            break;
        case THU :
```

```
            printf("你指定的是星期四");
            break;
        case FRI :
            printf("你指定的是星期五");
            break;
        case SAT :
            printf("你指定的是星期六");
            break;
        case SUN :
            printf("你指定的是星期日");
            break;
        default :
            printf("指定的数值只能是1--7");
    }
}
```

Android6.0 新特性之 DataBinding

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

简介

去年谷歌 I/O 大会上介绍了一个非常厉害的新框架 DataBinding， 数据绑定框架给我们带来了很大的方便，以前我们可能需要在每个 Activity 里写很多的 findViewById，不仅麻烦，还增加了代码的耦合性，如果我们使用 DataBinding，就可以抛弃那么多的 findViewById，省时省力。说到这里，其实网上也有很多快速的注解框架，但是注解框架与 DataBinding 想比还是不好用，而且官网文档说 DataBinding 还能提高解析 XML 的速度，其实 DataBinding 的好用，不仅仅体现在可以省去使用很多啰嗦 findViewById，还有很多。

环境配置

1、环境需要：

1.Android 2.1 (API level 7+)

2.Gradle 1.5.0-alpha1

3.Android Studio 1.3

2、环境搭建

在 AndroidStudio1.5 以上在 build.gradle 的 android 中加入如下字段,等待系统重新编译

```
android{
    ...
    dataBinding {
      enabled = true
    }
    ...
}
```

注意: 在配置完成的时候一定要先编译一下

HelloWord

布局格式

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>
      <variable
        name="user"
        type="com.atguigu.myapplication.User"></variable>
    </data>

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
      <TextView
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:gravity="center"
        android:background="#ffccff"
        android:text="@{user.username}"/>
    </LinearLayout>
  </layout>
```

注意：

1. 布局写完后都要编译一下生成相应的类
   2. 注意布局格式必须安照这种格式来写 最外层是一个 layout，里面包含两个部分，一个是数据部分，一个是布局

先来解释上面的数据部分

Type：属性的意思是数据 been 的全名
name: 是给这个 been 起个名字，名字可以随便起（最好和 been 的名字一样）
也可以写成这种形式

```
//把需要的 been 类导进来
<import type="com.atguigu.myapplication.User"></import>
<variable
```

```
    name="aaa"
    type="User"> //要写 been 的类名
  </variable>
```

解析下面的布局部分

```
//@{}是固定写法  user 是指我们的 been 的类名  username 是类里的属性
  text = "@{user.username}"
```

代码部分

Been 类

```
public class User {

    public String username;
    public String address;
    public int age;

    public User(String username, String address, int age) {
      this.username = username;
      this.address = address;
      this.age = age;
    }

        ......

  }
```

Activity 类里

在我们的 onCreate（）

```
  ActivityMainBinding binding = DataBindingUtil.setContentView(this,  R.layout.activity_main);
  //  相当于把数据 set 进我们的 been 里
  binding.setUser(new User("HelloWord","atguigu",18));
```

注意: ActivityMainBinding 是我们的布局名字后面加上 binding 如果没有生成这个类的话一定是布局没有编译

是不是已经看到我们亲切的 helloword 了!!!

databinding 布局中的语法

```
//三目运算是可以用的
  <TextView
     android:visibility="@{user.isAdmin ? View.VISIBLE : View.GONE}"
     //如果 been 类里 age 是 int 类型不能直接使用 可以加上一个字符串 注意：外部是单引号
     android:text='@{user.age+" bb "}'/>

  //格式化字符串（还可使用正则表达式等）
  android:text="@{@string/nameFormat(firstName, lastName)}"
  android:text="@{String.valueOf(index + 1)}"
  android:visibility="@{age < 13 ? View.GONE : View.VISIBLE}"
  android:transitionName='@{"image_" + id}'
  //Null Coalescing 运算符
  android:text="@{user.displayName ?? user.lastName}"
  //就等价于
  android:text="@{user.displayName != null ? user.displayName : user.lastName}"
```

表达式

```
数学 + - / * %
字符串连接 +
逻辑 && ||
二进制 & | ^
一元运算 + - ! ~
移位 >> >>> <<
比较 == > < >= <=
instanceof
分组 ()
null
Cast
方法调用
数据访问 []
```

三元运算 ?:

如何实现单击事件

第一步 我们先看布局

```
<data>
  <variable
    name="event"
    type="com.atguigu.myapplication.MainActivity"></variable>
</data>
<TextView
    android:layout_margin="10dp"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:gravity="center"
    android:background="@{@color/colorAccent}"
    android:text="@{Item.name}"
    //这个就是我们类里的第一种点击事件方法
    android:onClick="@{event.onClick}"
    />
<Button
  android:layout_width="match_parent"
  android:layout_height="50dp"
//这个是我们的第二种点击方式 使用了 lambda 表达式
  android:onClick="@{() -> present.onClickDemo()}"/>
```

第二步看代码

```
binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
binding.setEvent(this);
```

绑定

- 双向绑定：就是当我们 UI 展示的值改变的时候我们的 Been 对象里对应的属性值也会改变， 相反当我们数据里的值改变的时候我们的 UI 值也会马上改变
- 单向绑定指的就是当我们的 been 的属性改变的时候 UI 也会马上进行改变

//在我们的 MainActivity 里

```
    public class Prensent {
  //第一种
    public void onClick(View view){
      Toast.makeText(MainActivity.this,user.getName(),Toast.LENGTH_SHORT).show();
    }

  //第二种
    public void onClickDemo(){
      Toast.makeText(MainActivity.this,user.getName(),Toast.LENGTH_SHORT).show();
    }

  }
```

单向绑定

第一种

我们通过继承 Observable 实现单向绑定。继承 BaseObservable+Bindable 注解 +notifyPropertyChanged(BR.xx)

```
    public class ObservableUser extends BaseObservable {
      private String userName;
      @Bindable
      public String getUserName() {
        return userName;
      }
      public void setUserName(String userName) {
        this.userName = userName;
        notifyPropertyChanged(BR.userName);
      }
    }
```

所有的 set 方法必须加上面的注解，所有的 get 方法必须加上 notifyPropertyChanged(BR.userName);

注意 ： 当使用@Bindable 注解后，对应的 BR.xx 就会自动生成

第二种

继承 BaseObservable+notifyChange();

```
    public class TestUser extends BaseObservable{
      private String firstName;
      public String getFirstName() {
```

```
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
        notifyChange();
    }
}
```

这个比上一个简单。不用注解，直接 notifyChange()。

其它

一些小工作会涉及到创建 Observable 类，因此那些想要节省时间或者几乎没有几个属性的开发者可以使用 ObservableFields。ObservableFields 是自包含具有单个字段的 observable 对象。它有所有基本类型和一个是引用类型。要使用它需要在 data 对象中创建 public final 字段：

```
private static class User extends BaseObservable {
  public final ObservableField<String> firstName =
      new ObservableField<>();
  public final ObservableField<String> lastName =
      new ObservableField<>();
  public final ObservableInt age = new ObservableInt();
}
```

就是这样，要访问该值，使用 set 和 get 方法：

```
user.firstName.set("Google");
int age = user.age.get();
```

进行测试

```
new Handler().postDelayed(new Runnable() {
    @Override
    public void run() {

        user.setName("bbbbbbbbb");
    }
},2000);
```

是不是看到了效果呢 就是这么的潇洒!!!

双向绑定

布局

```
<EditText
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:id="@+id/et"
      //注意这里啊写法不一样了呢
        android:text="@={user.name}"/>
```

只要我们在另一个控件了展示这个数据就可以了现在你输入一些数据进去是不是另一个控件的数据也进行了变化呢!!!

使用@BindingAdapter

我们在控件中可以自定义属性，也可以使用本生带有的属性.

举例：

```
//布局
android:src="@{user.name}"
//代码
 @android.databinding.BindingAdapter({"android:src"})
  public static void loadImage(ImageView view, String url) {
     Log.e("aaaaaaaaaaaaa", "url: "+url);
     Glide.with(view.getContext()).load(url).into(view);
  }
```

大家去执行的时候就会发现我们写的方法被调用了，是不是很 COOL

自定义：

布局

```
<ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        //自定义属性
        bind:imageUrl="@{user.name}"/>

@BindingAdapter({"bind:imageUrl"})
public static void loadImage(ImageView view, String url, Drawable error) {
    Picasso.with(view.getContext()).load(url).error(error).into(view);
```

```
}
```

是不是比我们原来自定义控件爽的不行不行呀!!!!!!

注意：

```
// 1. 全部满足 当两个自定义属性都被使用时才会调用此方法
@BindingAdapter({"bind:imageUrl", "bind:error"})
public static void loadImage(ImageView view, String url, Drawable error) {
  Picasso.with(view.getContext()).load(url).error(error).into(view);
}
// 2. 满足其中一个的时候
@BindingAdapter(value = {"bind:imageUrl", "bind:error"} , requireAll = false)
public static void loadImage(ImageView view, String url, Drawable error) {
  Picasso.with(view.getContext()).load(url).error(error).into(view);
}
```

转换器 Converters

1. 对象转换

如果是表达式返回的对象，则会从自动的、重命名或自定义的 setter 方法中选择一个 setter 方法。对象会强转成适合 setter 方法中参数类型。这种方式适合用 ObservableMaps 来保存数据的布局，如：

```
<TextView
android:text='@{userMap["lastName"]}'
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
```

userMap 返回的对象自动强转为 setText(CharSequence)中参数类型。当 setter 方法的参数类型存在二义性时，如存在两个 setter 方法，但是参数不一样时，需要使用手动强转。

2. 自定义转换

有时需要让布局在指定类型之前自动转换，如设置背景时

```
<View
```

```
    android:background="@{isError ? @color/red : @color/white}"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

这里 android:background 的 setter 方法参数类型为 Drawable，但是 Color 是整型。可以将整型的 color 转为 Drawable 类型的 ColorDrawable，通过 BindingConversion 注解的静态方法来实现。

```
    @BindingConversion
public static ColorDrawable convertColorToDrawable(int color) {
  return new ColorDrawable(color);
}
```

注：转换器只在 setter 层执行，禁止使用混合类型。如：

```
<View
  android:background="@{isError ? @drawable/error : @color/white}"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"/>
```

MaterialDesign 之 DrawerLayout

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

DrawerLayout 基本使用

我们先来看一下布局

```
<android.support.v4.widget.DrawerLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/dl">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <android.support.design.widget.TabLayout
            android:layout_height="50dp"
            android:layout_width="match_parent"
            android:id="@+id/tb">
        </android.support.design.widget.TabLayout>

        <android.support.v4.view.ViewPager
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/vp"
            >
        </android.support.v4.view.ViewPager>
    </LinearLayout>


    <LinearLayout
        android:layout_width="100dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:orientation="horizontal"
        android:background="#ffccff"
        >
        <TextView
```

```
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="aaa"/>
    </LinearLayout>
</android.support.v4.widget.DrawerLayout>
```

在我们的 DrawerLayout 里包括两个子布局 第一个是主页布局 第二个是侧滑布局

注意：

第一个布局可以是任意布局 但必须设置成 全屏显示，

第二个布局可以任意宽度 但必须设置

`android:layout_gravity="start"`

Start 表示从左边显示

End 表示从右边显示

然后重启一下是不是可以从左边滑出菜单来了，就是这么 easy

DrawerLayout 和 ToolBar 的绑定

首先我们要在上面的布局里 DrawerLayout 上填加一个布局 ToolBar。

```
<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?android:attr/actionBarSize"
        android:background="?attr/colorPrimary"
        app:layout_scrollFlags="scroll|enterAlways"
        />
</android.support.design.widget.AppBarLayout>
```

第一步初始化 ToolBar 控件

```
toolbar = (Toolbar) findViewById(R.id.toolbar);
dl = (DrawerLayout) findViewById(R.id.dl);
setSupportActionBar(toolbar);
```

第二步显示箭头

（效果图）

在这的时候就可以看到我们的图标了（嘿嘿）

```
ActionBar actionBar = getSupportActionBar();
//设置当前的控件可用
actionBar.setDisplayHomeAsUpEnabled(true);
actionBar.setHomeButtonEnabled(true);
actionBar.setDisplayShowTitleEnabled(true);
```

第三步 关联 DrawerLayout

```
//第一个参数 activity 第二个参数 drawlayout
//第三个参数 Toolbar 第四个和第五个是打开和关闭的文字
toggle = new ActionBarDrawerToggle(this, dl, toolbar,R.string.aa, R.string.bb);
//该方法会自动和 ToolBar 关联
toggle.syncState();
```

第四部设置监听（给我们的 DrawerLayout 设置监听）

```
dl.addDrawerListener(new DrawerLayout.DrawerListener() {
    @Override
    public void onDrawerSlide(View drawerView, float slideOffset) {
        toggle.onDrawerSlide(drawerView, slideOffset);
    }

    @Override
    public void onDrawerOpened(View drawerView) {
        toggle.onDrawerOpened(drawerView);
    }

    @Override
    public void onDrawerClosed(View drawerView) {
        toggle.onDrawerClosed(drawerView);
    }

    @Override
    public void onDrawerStateChanged(int newState) {
        toggle.onDrawerStateChanged(newState);
    }
});
```

第五部 响应事件

```
@Override
```

```
public boolean onOptionsItemSelected(MenuItem item) {
        return toggle.onOptionsItemSelected(item);
}
```

Git 版本控制

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

Git 简介

Git 是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。Git 是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。

Git 的作用

如果你用 word 写过作文可能有这样的经历，想删除一个段落，但觉得这段就算不用在这用在别的地方一样很好？有办法，先把当前文件“另存为……”一个新的 word 文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去，最后你的 word 文档变的十分混乱：

过了一年，突然有一天你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了。

看着一堆乱七八糟的文件，想保留最新的一个，然后把其他的删掉，又怕哪天会用上，还不敢删，真郁闷。

于是你想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便？

如果你有过上述的困惑 那么相信 Git 一定不会让你失望

Git 和 SVN 的区别

SVN 是集中式分布管理系统 Git 是分布式管理系统。OK 这就是他们最大的区别。

看完以后是不是有点想骂街的赶脚。哈哈.....别急别急。看来有些东西用太专业的词的确难以让人理解。

大家用过 svn 的人应该都知道我们合并代码 分支 回滚 一系列的版本控制操作都必须在服务器存在的前提下才能完成，比如现在我刚写了一个支付的模块在没有服务器的前提下我是不能提交的，然后又写了一个分享模块，结果分享模块出现了重大问题，想回到写完支付模块的状态发现根本回滚不了，因为没有服务器。这个时候就觉得太尴尬了。而我们的 Git 就不需要他可以在自己的电脑上去处理，这个时候小伙伴又晕了没有服务器怎么处理，其实是这样的，他会对当前电脑上的代码进行版本控制，连上服务器那根没有问题了。也就是说 Git 主要由本地版本控制和网络版本控制合在一起。这个时候你在本地的任何操作就都没有问题了。

本地版本控制

在 Windows 上安装 Git

从 https://git-for-windows.github.io 下载 然后按默认选项安装即可。安装完成后，在开始菜单里找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明 Git 安装成功！

创建版本库

什么是版本库呢？版本库又名仓库，英文名 repository，你可以简单理解成一个目录或者数据库，这个目录里面的所有文件都可以被 Git 管理起来，每个文件的修改、删除，Git 都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

```
$ cd d:         //进入D盘目录
$ mkdir demo    //创建一个文件夹名叫demo
$ cd demo       //进入这个demo里
$ pwd           //查看这个文件夹的详细路径
$ git init    //初始化我们的仓库
Initialized       empty       Git       repository       in
C:/Users/Administrator/Desktop/demo2/.git/
```

扩展一下 cd .. 是退出当前目录和我们的返回是一样的
我在桌面创建的仓库 大家找到你创建的文件夹进去看一下是不是有了一个.git 的文件恭喜你初始化成功了，如果没有可以在你的 系统的文件夹选项-查看-显示隐藏的文件
或者$ ls -ah //ls 是查看当前目录下有哪些文夹 -ah 是隐藏的也显示

./ ../ .git/
是不是看到了呢。

添加文件

进入我们刚才创建的仓库，在里面创建一个 readme.txt 写上一句话“第一个文件”保存。

```
$ git status   //查看当前的状态
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be
committed)
        readme.txt
nothing added to commit but untracked files present (use
"git add" to track)
//    是不是看到我们的 readme.txt 是一个红色这说明我们的文夹还没有添加到仓库里

$ git add readme.txt   //把我们的文件添加到仓库

$ git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   readme.txt
//   是不是变绿了这说明我们已经添加成功了 大家再看这句话
(use "git rm --cached <file>..." to unstage) //执行这条命令
就可以删除

我们继续用命令git commit告诉Git，把文件提交到仓库
$ git commit -m "第一次提交"  //-m  是写注释这个必须得写主要是用
来对你所提交内容的一个描述
[master (root-commit) d34b337] 第一次提交
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt

$ git status   //当我们再看一下状态是不是已经没有显示文件了这就说
明添加成功了
On branch master
nothing to commit, working directory clean
```

版本回退

我们模拟修了好多次 这次在我们的 readme.txt 里再加上一句话 “第二次修改”
然后我们执行

```
$ git status
  ......
        modified:   readme.txt
```

是不是又变红了 没错因为我们修改过了所以要重新提交 和上面一样要add 和commit操作 可以多进行几次修改。

```
$ git log  //查看记录
commit 8267f0e5b335161c4ffb270343b0565769cade15
Author: wangfeilong <flyloong_job@163.com>
Date:   Wed Aug 31 11:53:32 2016 +0800

    第二次修改

commit d34b3379d0766c2c11a856d1b20a4d988af523e3
Author: wangfeilong <flyloong_job@163.com>
Date:   Wed Aug 31 09:28:35 2016 +0800

    第一次提交
```

如果觉得太乱 可以加上以下参数 --pretty=oneline

```
$ git log --pretty=oneline
8267f0e5b335161c4ffb270343b0565769cade15 第二次修改
d34b3379d0766c2c11a856d1b20a4d988af523e3 第一次提交
```

上面的命令大家可以看到我们的提交记录和修改记录
上面一大串的数字其实是Git的commit id 是一个SHA1计算出来的一个非常大的数字，用十六进制表示

假如我们现在的需求要回到第一次提交的时候，可以如下操作

```
$ git reset --hard HEAD^  //回退到上一个版本
HEAD is now at d34b337 第一次提交

Administrator@PC-201608231916 MINGW64 ~/Desktop/demo2
(master)
$ git log
commit d34b3379d0766c2c11a856d1b20a4d988af523e3
Author: wangfeilong <flyloong_job@163.com>
```

```
Date:   Wed Aug 31 09:28:35 2016 +0800
```

    第一次提交

哈哈是不是到我们的第一次提交了呢 再看我们的文件内容是不是也变了

那比如说回退了三次我现在又想回到最开始的怎么办，只要你的id还能找到那就没有问题我们再试一下

```
$ git reset --hard 8267f0e5b33516  //找到原来版本的ID可以只写前几位
HEAD is now at 8267f0e 第二次修改

Administrator@PC-201608231916 MINGW64 ~/Desktop/demo2 (master)
$ git log
commit 8267f0e5b335161c4ffb270343b0565769cade15
Author: wangfeilong <flyloong_job@163.com>
Date:   Wed Aug 31 11:53:32 2016 +0800

    第二次修改

commit d34b3379d0766c2c11a856d1b20a4d988af523e3
Author: wangfeilong <flyloong_job@163.com>
Date:   Wed Aug 31 09:28:35 2016 +0800

    第一次提交
```

是不是又回到了我们指定要回到的版本，就是这么爽。

那比如我的ID找不到了怎么办，请大家看下面

```
$ git reflog   //用来查询我们前面的操作
8267f0e HEAD@{0}: reset: moving to 8267f0e5b33516
d34b337 HEAD@{1}: reset: moving to HEAD^
8267f0e HEAD@{2}: commit: 第二次修改
d34b337 HEAD@{3}: commit (initial): 第一次提交
```

看左边是不是出来了呢

远程仓库

所谓的远程仓库类似我们的 SVN 服务器用来帮我们管理代码，这次我们就选用 github 当然还有很多类似的服务器我们就不一一说明了

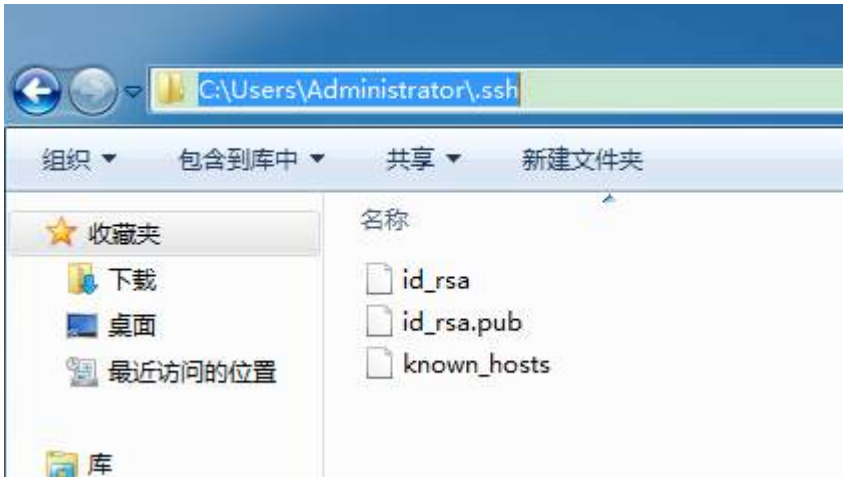
Github 的简单使用

注册 github

Github 网址 ： https://github.com/
首先注册一个账号 首页就是注册的界面 登录的界面在左上角 singin 是登录

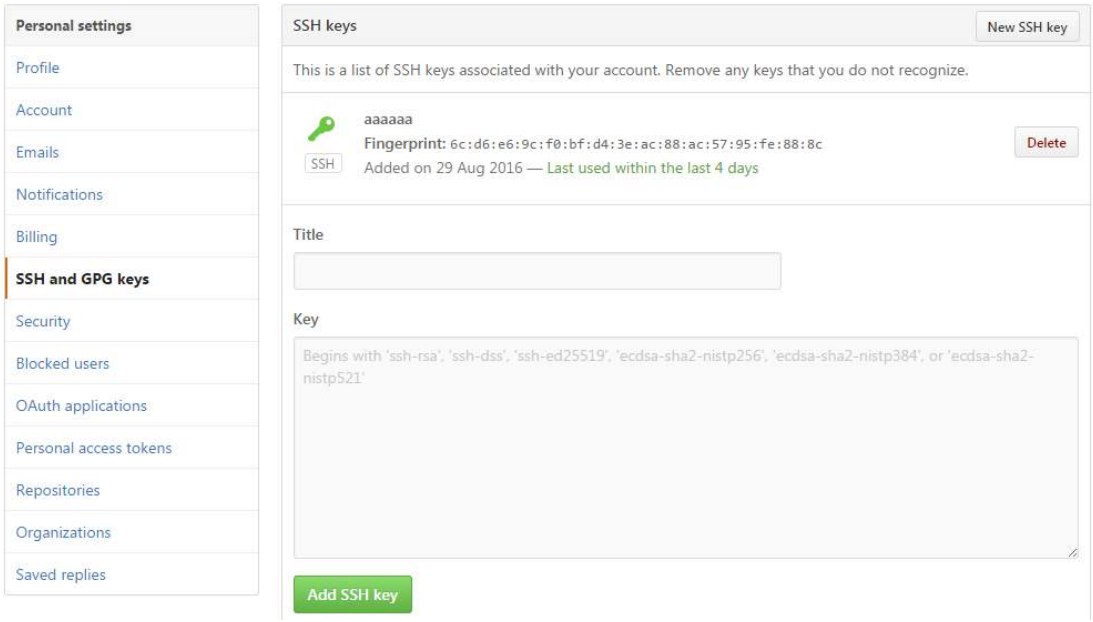
SSH

由于你的本地 Git 仓库和 GitHub 仓库之间的传输是通过 SSH 加密的，所以，需要一点设置：
第一步 ：创建 SSH Key。在用户主目录下，看看有没有.ssh 目录，如果有，再看看这个目录下有没有 id_rsa 和 id_rsa.pub 这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开 Shell（Windows 下打开 Git Bash），创建 SSH Key



重点看路径 当然我这已经创建了 如果没有我们就执行以下操作
$ ssh-keygen -t rsa -C "youremail@example.com"
//后的邮箱是你注册的邮箱然后一路回车，使用默认值即可
这样就可以在我们刚才的路径里找到.ssh 目录，里面有 id_rsa 和 id_rsa.pub 两个文件，这两个就是 SSH Key 的秘钥对，id_rsa 是私钥，不能泄露出去，id_rsa.pub 是公钥，可以放心地告诉任何人。

第 2 步：登陆 GitHub，打开“settings”，“SSH Keys”页面：



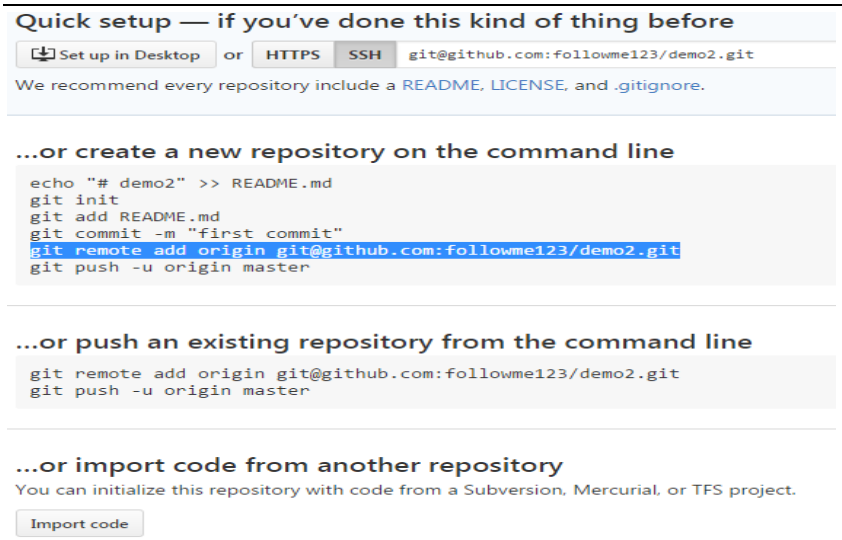
Tittle 随意写
Key　id_rsa.pub 里的内容粘过来就好 然后点击 Add SSH key

解释 ： 为什么 GitHub 需要 SSH Key 呢？因为 GitHub 需要识别出你推送的提交确实是你推送的，而 Git 支持 SSH 协议，所以，GitHub 只要知道了你的公钥，就可以确认只有你自己才能推送。当然，GitHub 允许你添加多个 Key。只要把每台电脑的 Key 都添加到 GitHub,就可以在每台电脑上往 GitHub 推送了。在 GitHub 上免费托管的 Git 仓库，任何人都可以看到（但只有你自己才能改）。如果你不想让别人看到 Git 库，有两个办法，一个是交费，让 GitHub 把公开的仓库变成私有的，这样别人就看不见了（不可读更不可写）。另一个办法是自己动手，搭一个 Git 服务器

本地仓库和远程仓库进行同步

第一步在远程仓库建立一个新的仓库


View 2 new broadcasts
Your repositories 3
New repository
Find a repository...
All
Public
Private
Sources
Forks
demo
aaa
test
Owner
Repository name
followme123
/
填写仓库的名称
Great repository names are short and memorable. Need inspiration? How about furry-parakeet.
Description (optional)
描述可以不写
Public
仓库是公有的免费 但是别人都可以看见和下载
Anyone can see this repository. You choose who can commit.
Private
仓库是私有的但是需要效一定费用
You choose who can see and commit to this repository.
Initialize this repository with a README
生成README文档可以不选
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.
Add .gitignore: None
Add a license: None
Create repository



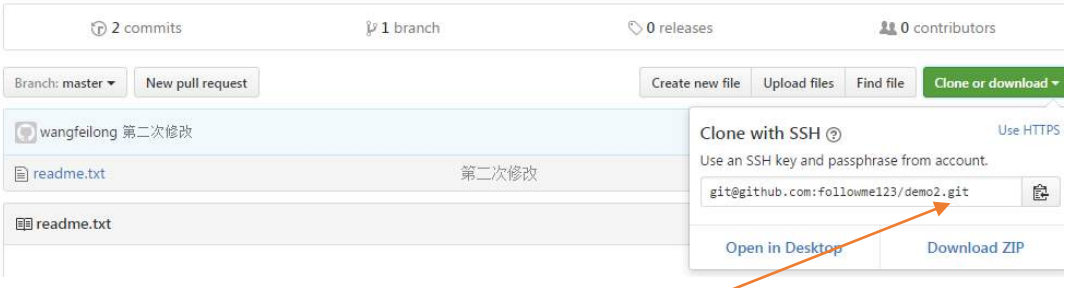
在 GitHub 上的这个仓库还是空的，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到 GitHub 仓库。

```
$ git remote add origin
    git@github.com:followme123/demo2.git //关联仓库
$ git push -u origin master
```

//把本地库的内容推送到远程，用 git push 命令，实际上是把当前分支 maste 推送到远程。由于远程库是空的，我们第一次推送 master 分支时，加上了-u 参数，Git 不但会把本地的 master 分支内容推送的远程新的 master 分支，还会把本地的 master 分支和远程的 master 分支关联起来，在以后的推送或者拉取时就可以简化命令。

我们检测下看有没有成功 可以刷新一下我们建好的仓库看我们本地的仓库数据有没有 Push 到服务器上，如果有的话 OK 那么你成功了

从远程仓库 Clone 代码



```
$ git clone git@github.com:michaelliao/gitskills.git
```

git@github.com:michaelliao/gitskills.git 的地址就是我们图片上的地址

分支管理

什么是分支

举个例子来说 我们小时候经常在写作业的时候有这种想法，如果我能分身就好了即可以写作业，又可以打游戏。现在 GIT 的分支就满足了我们小时候的愿望嘿嘿。当你需要同时做两件事的时候你就分身一个人写作业，一个人看电影，都搞好以后再合并大功告成全部做完。是不是很爽.....

分支在实际中有什么用？假设你准备开发一个新功能需要两周才能完成，第一周你写了 50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。
现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。
无论创建、切换和删除分支，Git 在 1 秒钟之内就能完成！无论你的版本库是 1 个文件还是 1 万个文件。

创建与合并分支

```
$ git branch  //查看当前分支 绿色就代表现在在当前的分支是master
* master
```

下面我们来创建分支 dev

```
$ git checkout -b dev
Switched to a new branch 'dev'
Administrator@PC-201608231916 MINGW64 ~/Desktop/demo2
(dev   //创建完成后直接就进入到我们的分支了
```

再次查看分支 是不是变了呢

```
$ git branch
* dev
```

```
  Master
```
切换分支
```
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

Administrator@PC-201608231916 MINGW64 ~/Desktop/demo2
(master) //是不是已经切换成功了呢 嘿嘿
```
其实分支相当于两个用户 在每个用户操作的数据 只有在当前分支能看到大家可以自行尝试在切换分支 进行数据修改 肯定看到的也不一样
如果想把两个分支的数据合在一起就要使用
```
$ git merge dev  //合并分支的时候要在master
```
删除分支
```
$ git branch -d dev
```
通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。
如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。
下面我们实战一下--no-ff方式的git merge：
首先，仍然创建并切换dev分支：

```
$ git merge --no-ff -m "merge with no-ff" dev
```

哇!!! 讲了这么多 相信很多朋友都在抱怨这么多的命令行，实在记不住啊!!!

没关系其实 GIT 也有视图化工具比如 sourcetree.

不过我仍然建议大家先用用命令行，再用工具这样大家会更容易理解。就这样了 有问题大家可以加我们的官方群

Http 协议详解

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

HTTP 简介

HTTP 协议（HyperText Transfer Protocol，超文本传输协议）是用于从 WWW 服务器传输超文本到本地浏览器的传送协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。

计算机相互之间的通信

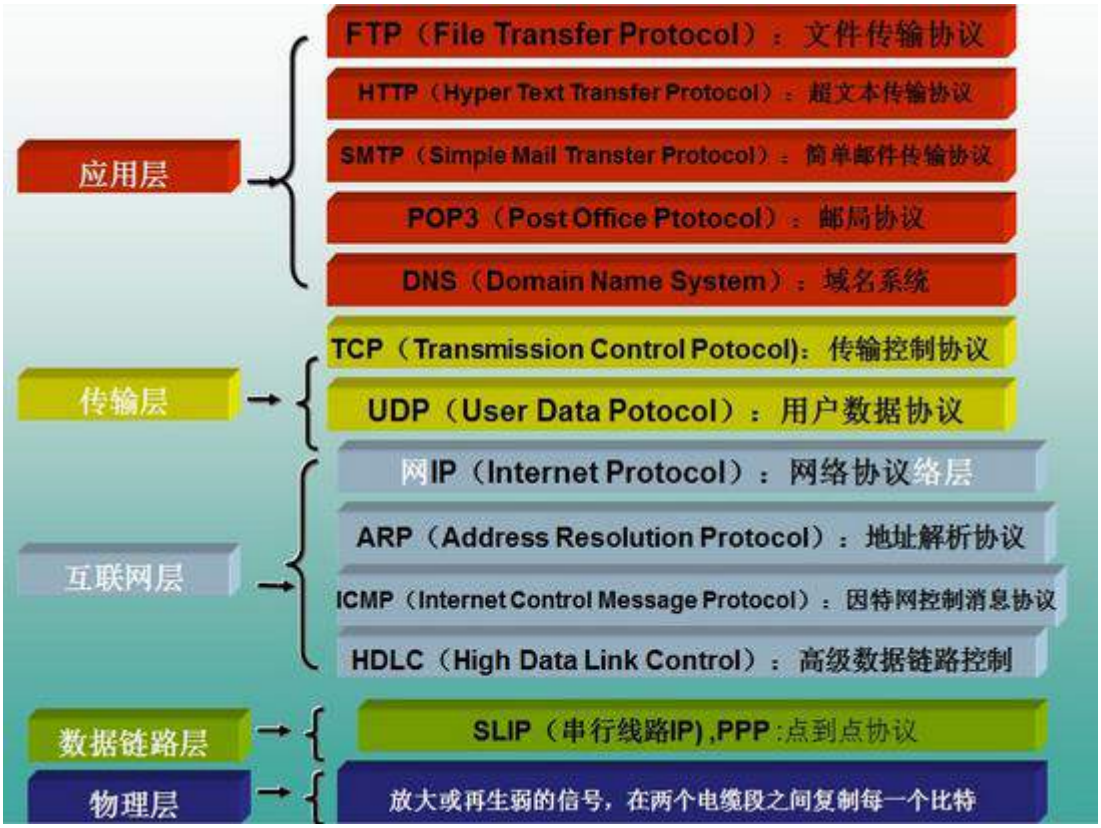
互联网的关键技术就是 TCP/IP 协议。两台计算机之间的通信是通过 TCP/IP 协议在因特网上进行的。实际上这个是两个协议：

TCP : Transmission Control Protocol 传输控制协议

IP： Internet Protocol 网际协议。

TCP 负责应用软件（比如你的浏览器）和网络软件之间的通信。IP 负责计算机之间的通信。TCP 负责将数据分割并装入 IP 包，IP 负责将包发送至接受者，传输过程要经 IP 路由器负责根据通信量、网络中的错误或者其他参数来进行正确地寻址，然后在它们到达的时候重新组合它们。

HTTP 协议所在的协议层



HTTP 请求响应模型

HTTP 由请求和响应构成，是一个标准的客户端服务器模型（B/S）。HTTP 协议永远都是客户端发起请求，服务器回送响应

HTTP 是一个无状态的协议。无状态是指客户机（Web 浏览器）和服务器之间不需要建立持久的连接

HTTP 工作过程

1）地址解析
2）封装 HTTP 请求数据包
3）封装成 TCP 包，建立 TCP 连接（TCP 的三次握手）
4）客户机发送请求命令
5）服务器响应
6）服务器关闭 TCP 连接

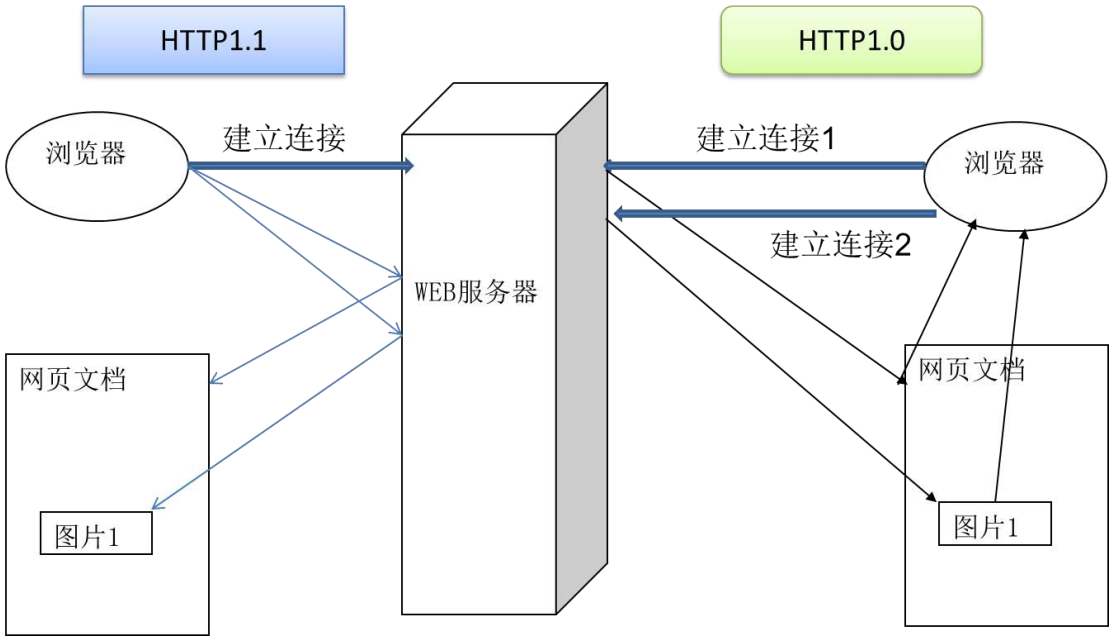
一般情况下，一旦 Web 服务器向浏览器发送了请求数据，它就要关闭 TCP 连接，然后如果浏览器或者服务器在其头信息加入了这行代码

Connection:keep-alive

TCP 连接在发送后将仍然保持打开状态，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

HTTP1.0 和 HTTP1.1 的区别

● 在 HTTP1.0 版本中，浏览器请求一个带有图片的网页，会由于下载图片而与服务器之间开启一个新的连接；但在 HTTP1.1 版本中，允许浏览器在拿到当前请求对应的全部资源后再断开连接，提高了效率。



Http 请求方式

HTTP1.1 支持 7 种请求方法：GET、POST、HEAD、OPTIONS、PUT、DELETE 和 TARCE

方法	作用
GET	请求获取由 Request-URI 所标识的资源
POST	请求服务器接收在请求中封装的实体，并将其作为由 Request-Line 中的 Request-URI 所标识的资源的一部分
HEAD	请求获取由 Request-URI 所标识的资源的响应消息报头
PUT	请求服务器存储一个资源，并用 Request-URI 作为其标识符
DELETE	请求服务器删除由 Request-URI 所标识的资源
TRACE	请求服务器回送到的请求信息，主要用于测试或诊断
CONNECT	保留将来使用
OPTIONS	请求查询服务器的性能，或者查询与资源相关的选项和需求

下面重点介绍 GET POST 请求

get 请求

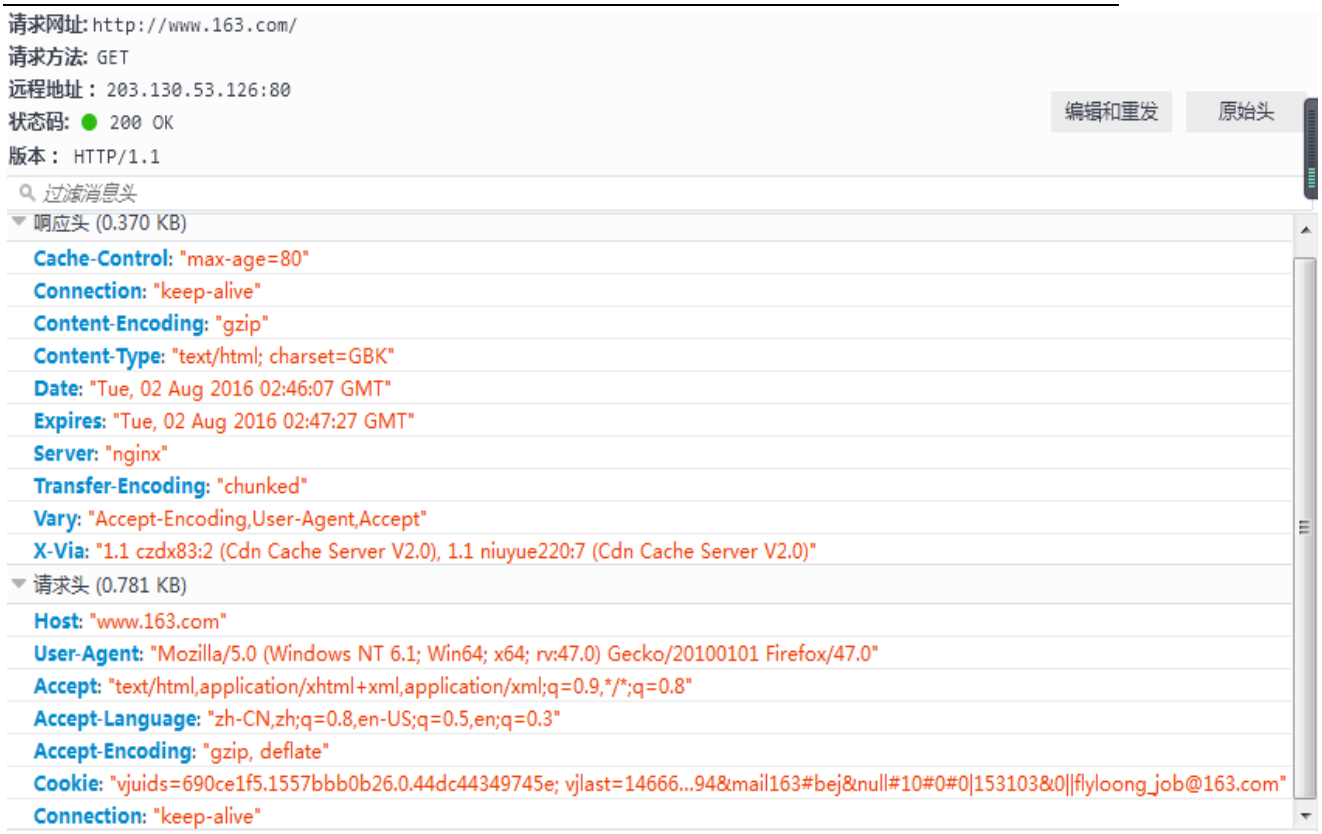
- GET 方法是默认的 HTTP 请求方法，输入网址的方式去访问网页的时候，浏览器采用的就是 GET 方法向服务器获取资源。

- 我们可以使用 GET 方法来提交表单数据，用 GET 方法提交的表单数据只经过了简单的编码，同时它将作为 URL 的一部分向服务器发送，因此，如果使用 GET 方法来提交表单数据就存在着安全隐患上。例如：

Http://localhost/login.php?username=aa&password=1234

从上面的 URL 请求中，很容易就可以辩认出表单提交的内容。（？之后的内容）

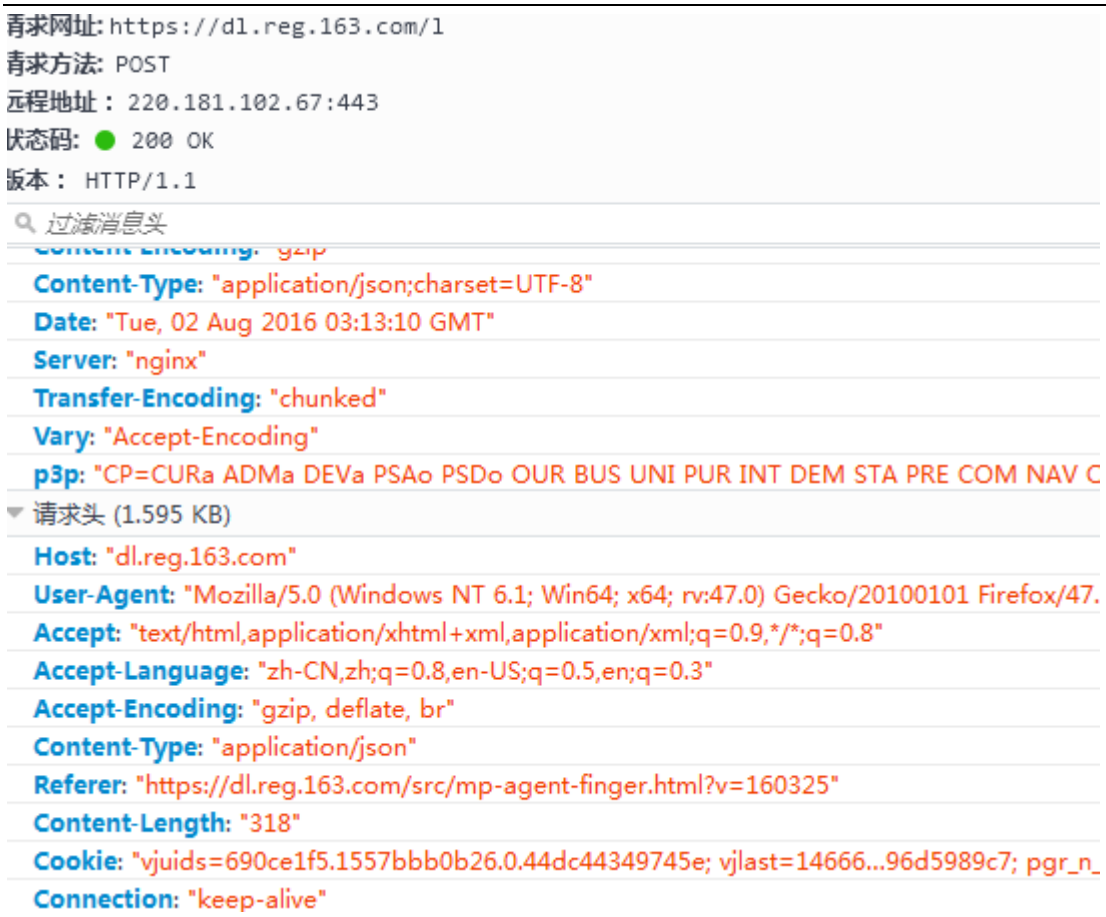
- 由于 GET 方法提交的数据是作为 URL 请求的一部分所以提交的数据量不能太大。这是因为浏览器对 url 的长度有限制各种浏览器也会对 url 的长度有所限制



post 请求

下面这是我要填写的一个表，我要把数据传给服务器用的就是 post

当我们点击保存的时候



请求头

Host： 服务器地址

User-Agent --- 用户代理 它是一个特殊字符串头，包含了浏览器类型以及版 本以及操作系统，浏览器内核，等信息的标识，通过这些信息可以提供更好的体验或者进行信息统计

Accept --- 接收的类型 audio/* 接受所有的声音文件。video/* 接受所有的视频文件。image/* 接受所有的图像文件。IME_type一个有效的 MIME 类型，不带参数。请参阅 IANA MIME 类型，获得标准 MIME 类型的完整列表。

Accept-Language --- 用于告诉服务器浏览器可以支持什么语言。 如果网站支持多语种的话，可以使用这个信息来决定返回什么语言的网页

Accept - Encoding： 是浏览器发给服务器,声明浏览器支持的编码类型[1]
常见的有 compress, gzip //支持 compress 和 gzip 类型(压缩)

Cookie： 主要用来保存账号和密码

Connection： 表示是否需要持久连接

Content-Type：告诉浏览器自己响应的对象的类型和字符集, text/html; charset=utf-8 / Content-Type: image/jpeg

响应头

Cache-Control 告诉所有的缓存机制是否可以缓存及哪种类型 Cache-Control: no-cache

Content-Type --- content-type 是必须的 multipart/form-data 是设置表单的 MIME 编码。默认情况，这个编码格式是 application/x-www-form-urlencoded，不能用于文件上传；只有使用了 multipart/form-data，才能完整的传递文件数据.

Content-Length --- 这里的 3693 是要上传文件的总长度

Date - 原始服务器消息发出的时间 Date: Tue, 15 Nov 2010 08:12:31 GMT

Expires 响应过期的日期和时间 Expires: Thu, 01 Dec 2010 16:00:00 GMT

Server web 服务器软件名称

Transfer-Encoding 文件传输编码 Transfer-Encoding:chunked

Vary 告诉下游代理是使用缓存响应还是从原始服务器请求

Accept-Ranges 表明服务器是否支持指定范围请求及哪种类型的分段请求 bytes(断点下载的时候用得到)

响应状态码

200 成功 服务器已成功处理了请求。通常，这表示服务器提供了请求的网页

302 重定向 代表让浏览器重新请求另一个资源

404 找不到 找不到请求的资源，但有时请求路径正确也返回 404 往往是由于 Web 应用有配置方面的问题，例如按照配置文件中指定的组件的全类名找不到指定的类。

500 错误	服务器内部错误，例如服务器端程序运行时抛出异常。

●响应状态码以 2 开头的通常表示成功。
●响应状态码以 3 开头的通常表示转移。
●响应状态码以 4 开头的通常表示无法访问，其中包括找不到资源或没有权限等。
●响应状态码以 5 开头的通常表示服务器端程序运行出错。

【尚硅谷-IT 精英计划】

JavaSE 学习笔记

视频下载导航（Java 学习路线图）

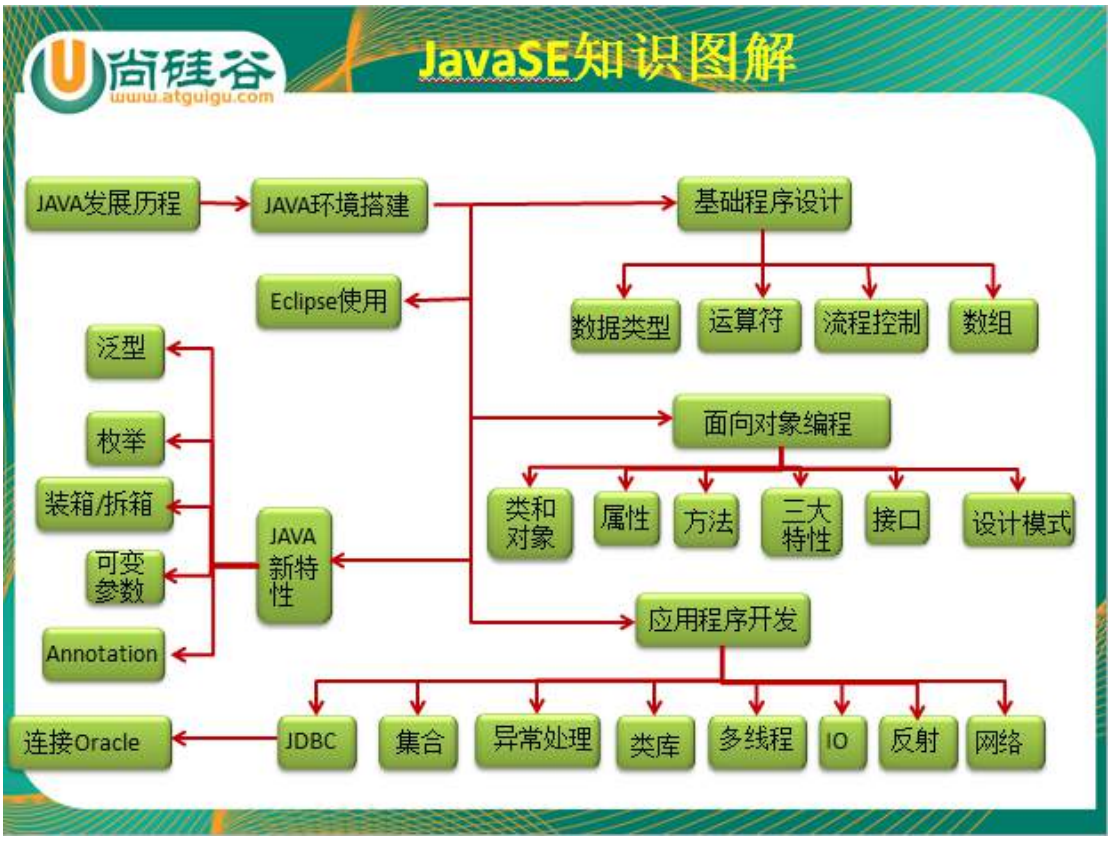
JavaEE 学科体系：

Android 学科体系：

第3阶段 **Android基础** **(基于Android6.0)**	Android开发环境搭建	熟悉使用Android Studio	四大应用组件	界面编程
	数据存储与IO	网络应用	Handler消息机制	事件处理机制
	应用资源	图片三级缓存	Fragement和ViewPager	图形与图像处理
	多媒体应用开发	管理桌面	传感器开发	GPS应用
	Git/SVN			
第4阶段 **Android** **高级开发**	自定义圆形进度条	自定义快速索引	自定义侧滑菜单	自定义滑动删除
	自定义优酷菜单	自定义广告条	自定义下拉刷新	自定义滑动开关
	自定义水波纹	自定义ViewPager	C语言	JNI/NDK开发
	锅炉压力显示案例	应用卸载问卷调查案例	美图处理案例	Android软件的破解技术
	Android系统源码分析	常用第三方框架源码分析	百度地图	
第5阶段 **Android** **真实项目**	图片拾取器	手机卫士	手机影音	百思不得姐
	新闻资讯	硅谷社交	硅谷商城	
第6阶段 **Android** **重量级框架**	Afinal	xUtils3	Volley	OkHttp
	Retrofit	Image-Loader	PhotoView	Picasso
	Fresco	Glide	Gson	FastJson
	EventBus	Otto	GreenDao	SnappyDB
	Butterknife	AndroidAnnotations	TabLayout	ViewPagerIndicator
	SlidingMenu	ActionBar	Toolbar	ActionBarSherlock
	PagerSlidingTabStrip	PinterestLikeAdapterView	NotBoringActionBar	StickyListHeaders
	NineOldAndroids	Android-gif-drawable	Vitamio5.0	Pulltorefresh
	xListView	Expandablelistview	SwipeRefreshLayout	SwipeMenuListView
第7阶段 **Android** **前沿实用技术**	Android主流软件架构搭建	图文混排技术	百分比布局	短信验证技术
	二维码扫描技术	第三方登录（QQ/微信/微博）	第三方分享	第三方支付（支付宝/微信等）
	语音识别	消息推送	H5混合开发	APP增量升级
	软件崩溃收集	多渠道打包及软件上线	Android6.0 / 7.0新特性	Android内存优化
	前沿技术分享			
第8阶段 **Android+H5** **混合开发**	React基本语法	React Native开发环境搭建	React Native开发基础	React Native常用组件
	React Native常用API	豆瓣搜索项目		

----------JavaSE 学习目录----------

【基础体系框架】



----------第 1 章：Java 语言概述 ----------

1.1 基础常识

软件：系统软件和应用软件

人机交互方式：图形化界面和命令行方式

常用的 DOS 命令：

dir : 列出当前目录下的文件以及文件夹

md : 创建目录

rd : 删除目录

cd : 进入指定目录

cd.. : 退回到上一级目录

cd\: 退回到根目录

del : 删除文件

exit : 退出 dos 命令行

---->学会如何在 DOS 命令下编译并运行 java 源程序（重点）：javac.exe java.exe

1.2 Java 语言概述

- 了解语言的分代：第一代：机器语言 第二代：汇编语言 第三代：高级语言（面向过程 & 面向对象）

1.3 Java 程序运行机制及运行过程

Java 语言的特点：①纯面向对象性：类&对象；面向对象的三大特性：封装性、继承性、多态、（抽象）

②健壮性：----->Java 的内存回收机制

③跨平台性：一次编译，到处运行。 ----->JVM

1.4 Java 语言的环境搭建：掌握下载、安装 JDK，并且配置环境变量（重点）

1）JDK 和 JRE 以及 JVM 的关系

2）JDK 的安装

3）配置 path 环境变量 path：window 执行命令时所需要搜寻的路径。

将 D:\Java\jdk1.7.0_07\bin 复制在 path 环境变量下。

1.5 开发体验 — HelloWorld

```
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("HelloWorld!!");
    }
}
```

1.6 小结第一个程序

1) Java 源文件以“java”为扩展名。源文件的基本组成部分是类（class），如本类中的 HelloWorld 类。

2) 一个 Java 源程序中可以存在多个 class 类，但是最多只能有一个类声明为 public. 若存在声明为 public 的类，那么这个源程序文件的名字必须以此类的类名来命名

3) 程序的入口是 public static void main(String[] args){} 称为主方法。它的写法是固定的。

4) Java 方法由一条条语句构成，每个语句以“;”结束。

5) Java 语言严格区分大小写

6) 大括号都是成对出现的，缺一不可，用于表明类中成员的作用范围。

1.7 常见问题及解决方法

1.8 注 释

作用：提高了代码的阅读性；调试程序的重要方法。

三种注释：

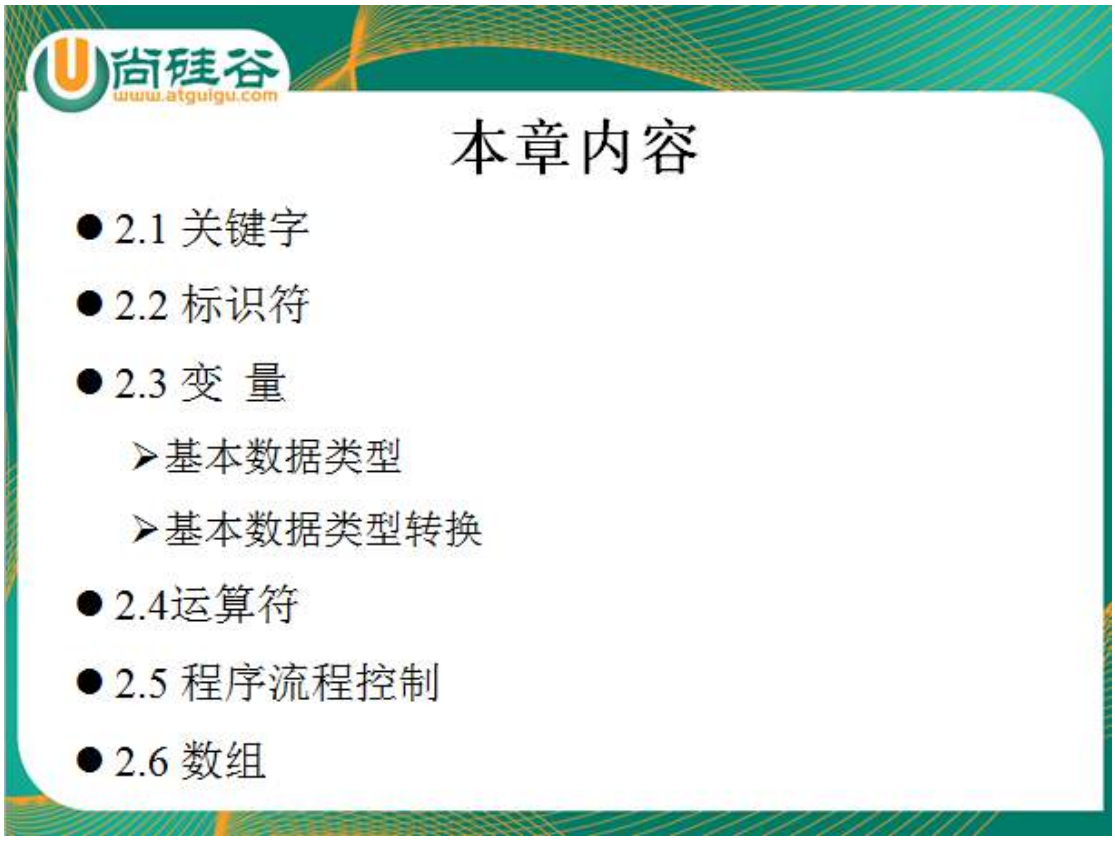
当行注释：//注释的内容

多行注释:/*注释的内容*/

文档注释（Java 所特有的，可以为 javadoc 命令所解析）：/** 注释的内容 */

1.9Java API 文档： API:application programming interface

----------第 2 章：基本语法 ----------



2.1 关键字 & 保留字

被 Java 语言赋予了特殊含义，用做专门用途的字符串（单词）

2.2 标识符

凡是自己可以起名字的地方都叫标识符。

通常有：类名、变量名、方法名。。。。包名、接口名、。。。

规则:（必须按照如下的规则执行，一旦某规则不符合，编译运行时就会出问题）

由 26 个英文字母大小写，0-9 ，_或 $ 组成

数字不可以开头。

不可以使用关键字和保留字，但能包含关键字和保留字。

Java 中严格区分大小写，长度无限制。

标识符不能包含空格。

命名的习惯:（如果不遵守，实际上程序编译运行也不会出问题）

包名：多单词组成时所有字母都小写：xxxyyyzzz

类名、接口名：多单词组成时，所有单词的首字母大写：XxxYyyZzz

变量名、方法名：多单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写：xxxYyyZzz

常量名：所有字母都大写。多单词时每个单词用下划线连接：XXX_YYY_ZZZ

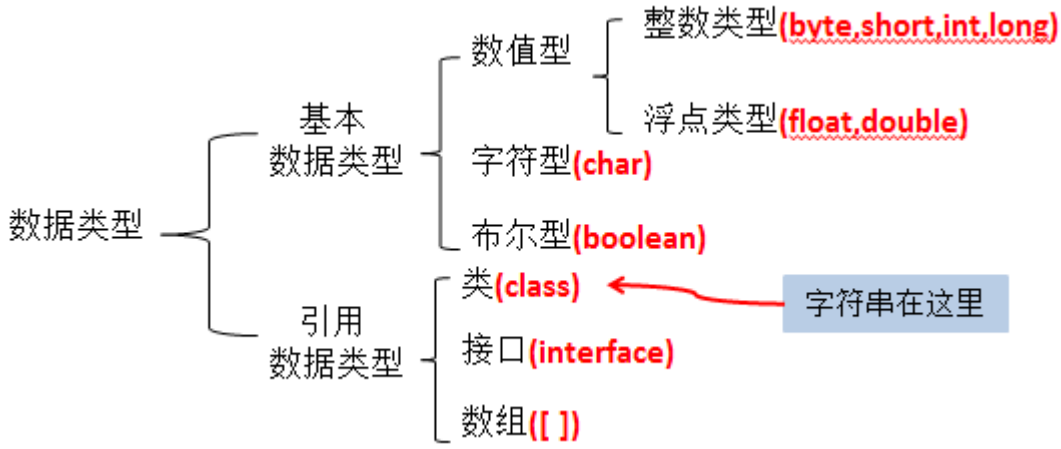
2.3 变量

1.变量的分类

1）存在的位置的不同：成员变量（属性、Field）(存在于类内部，方法体的外部)和局部变量(方法体的内部、

构造器的内部、代码块的内部、方法体或构造器的形参部分)

2）数据类型的不同：



基本数据类型（8 种)

整型：byte(1 个字节)、short、int（默认值类型）、long（后缀是 l 或 L）。

浮点型：float（后缀是 f 或 F）和 double（默认值类型）。

布尔型：boolean （只有两个值：true 和 false，绝对不会取值为 null）

字符型：char（1 个字符）

引用数据类型：类（比如：String 字符串类型）、接口、数组

2.如何声明：

变量的类型 变量的名字 初始化值（显式的初始化、默认初始化）

```
int    i;
i = j + 12;
boolean b = false;
```

String str = "atguigu";

Customer c = new Customer();

3.变量必须先声明后使用

4.使用变量注意：

变量的作用域：一对{ }之间有效

初始化值

5.自动类型转化&强制类型转化（重点）（不包含 boolean 型，及 String 型）

1）强制类型转化时，可能会损失精度。

2.4 进制

十进制

二进制

八进制

十六进制

1.进制之间的转化：

会将-128 到 127 范围内的十进制转化为二进制　& 能将一个二进制数转换为十进制数（延伸：十进制、二进制、八进制、十六进制）

2.对于几类基本类型的数据的某个具体值，能够用二进制来表示。同时，如果给出一个数据的二进制，要能够会

转化为十进制！

正数：原码、反码、补码三码合一。

负数：原码、反码、补码的关系。负数在计算机底层是以补码的形式存储的。

2.5 运算符

算术运算符

赋值运算符

比较运算符（关系运算符）

逻辑运算符

“&”和“&&”的区别：&：当左端为 false 时，右端继续执行

&&：当左端为 false 时，右端不再执行

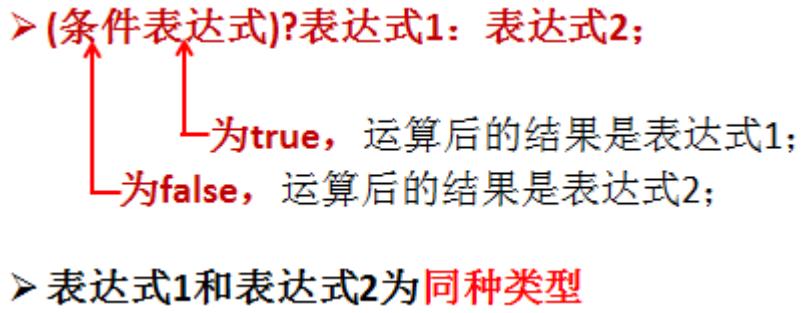
“|”和“||”的区别：|：当左端为 true 时，右端继续执行

||：当左端为 true 时，右端不再执行

位运算符

<<　　>>　　>>>　&　　|　^　~

三元运算符



1.三元运算符与 if-else 语句的联系

1）三元运算符可以简化 if-else 语句

2）三元运算符一定要返回一个结果。结果的类型取决于表达式 1 和 2 的类型。（表达式 1 和 2 是同种类型的）

3）if-else 语句的代码块中可以有多条语句。

2.6 流程控制

条件判断：

```
if(表达式 1){
    //执行语句
}else if(表达式 2){
    //执行语句
}else{
    //执行语句
}
```

1.一旦执行条件判断语句，有且仅有一个选择“路径”里的代码块被执行。

2.如果多条表达式间是“互斥”关系，彼此是可以调换顺序。

如果多条表达式间存在“包含”关系，一定要将范围小的表达式写在范围大的表达式对应的语句的上面。

3.选择判断语句可以“嵌套”

4. 若执行语句只有一句，那么对应的一对{}可以省略。

选择结构：switch-case

1.结构

```
switch(表达式){
case 常量1:
      语句1;
      break;
case 常量2:
      语句2;
      break;
… …
case 常量N:
      语句N;
      break;
default:
      语句;
      break;
 }
```

2.表达式可以存放的数据类型：int byte short char String 枚举

3.表达式存放是数值或者说是离散的点所代表的常量，绝对不是取值范围。

3.default 是可选的。default 的位置也不是固定的，当没有匹配的 case 时，执行 default

4.break 在 switch-case 中的使用。当执行到某条 case 语句后，使用 break 可以跳出当前的 switch 语句。

如果此 case 中没有 break，那么，程序将依次执行下面的 case 语句，直至程序结束或者遇到 break。

【switch-case 与 if-else if-else 的联系】

1.表达式是一些离散的点，并且取值范围不是很大，要求是 int byte short char String 枚举类型之一。

建议使用 switch-case。执行效率更高

2.如果表达式表示的是范围（或区间）、取值范围很大，建议使用 if-else if-else

循环结构：①初始化条件②循环条件部分③循环体部分④迭代部分

1.for 循环

```
for(①;②;④){
    ③
}
```

执行顺序：①-②-③-④-②-③-④-…-②-③-④-②截止

```
//死循环：
for(;;){
    //要循环执行的代码。
}
```

2.while 循环

```
①
while(②){
    ③
    ④
}
//死循环：
while(1>0){
    //要循环执行的代码
}
```

注意：for 循环和 while 循环之间可以相互转化！

3.do-while 循环

①

```
do{
    ③
    ④
}while(②);
//程序至少执行一次!
```

掌握：1.会使用 for 循环和 while 循环

2.能够实现 for 循环和 while 循环的相互转化。

循环可以相互嵌套

例题 1.九九乘法表；

例题 2.输入 1-1000 之间的所有质数

2.7 关键字：break & continue

break

1.使用范围：循环结构或 switch-case 结构中。

2.在循环结构中：一旦执行到 break，代表结束当前循环。

continue

1.使用范围：循环结构

2.在循环结构中：一旦执行到 continue,代表结束当次循环。

相同点：如果 break 或 continue 后面还有代码，那么这些代码将不会被执行。所以当有代码，编译会出错！

如何使用标签，实现结束指定“层次”的循环。（理解）

例题：

```
class Test{
    public static void main(String[] args){
        for(int i = 1 ;i <= 100;i++){
```

```
                if(i % 10 == 0){
                    break;
                    //continue;
                    //下面不能写入任何代码
                }
                System.out.println(i);
            }
        }
}
```

2.8 数组

1.如何创建一维数组

1）数组的声明： int[] i ; String[] str; Animal[] animal;

2)数组的初始化：

①动态初始化：i = new int[4]; //i[0]开始，至 i[3]结束。默认初始化值都为 0;

i[0] = 12;i[1] = 34;....

②静态初始化：str = new String[]{"北京","尚硅谷","java0715 班"};

//也是先有默认初始化赋值，然后显示的初始化，替换原来的默认初始化值

//对于引用数据类型来说，默认初始化值为 null。

//对于基本数据类型来说: byte、short、int ：0 long :0L float : 0.0F double : 0.0 char :'\u0000' boolean : false 引用类型：null

2.数组元素的下角标从 0 开始。

3. float[] f = new float[]{1.2F,3.4F,5.6F};

double[] d = new double[4]; d[1] = 3.4;.....

4.数组的长度: .length;

5.数组的遍历：（习惯使用 for 循环）

```
for(int i = 0 ;i < f.length;i++){
    System.out.print(f[i] + "\t");
}
```

6.常见的异常：

1）ArrayIndexOutOfBoundsException :数组下标越界异常　　下标从 0 开始，到 length-1 结束。如果下角标的取值不在此范围内，将报此异常

2）空指针异常(NullPointerException)

2.二维数组

1)

①动态初始化：

---> int[][] i = new int[3][];　　i[0] = new int[2];　i[1] = new int[3];　i[1] = new int[4];

--->String[][] str = new String[2][4];

②　静态初始化：

int[][] i = new int[][]{{1,2,3},{3,4},{0}};

2)遍历（补充完整）

7. 数组常用的算法：最大值、最小值、和、平均数、排序（涉及数据结构中各种排序算法）、复制、反转。

8. Arrays：操作数组的工具类

Array.sort(数组类型形参);　可以对形参部分的数组进行排序，默认是从小到大的顺序。

----------第 3 章：面向对象编程 ----------

3.1 面向过程与面向对象

1.面向过程关注于功能和行为
　　面向对象关注于功能和行为所属的对象。

2.Java 语言，作为面向对象的语言，更多的关注于类的设计！

3.面向对象两大元素：类和对象
　　三大特性：封装、继承、多态、（抽象）

3.2 类

1.java 的源程序是由一个一个的类构成的。
源文件名【类名 3.java】

```
class   类名 1{
    属性 1
    属性 2
    。。。
    方法 1(){
    }
    方法 2(){
    }
    .....
}
```

class 类名 2{

}

public class 类名 3{

}

2.类的组成部分：1）属性　2）方法　3）构造器　4）代码块　5）内部类

1）属性（Field、成员变量、字段）：定义在类内部，方法外部的变量。
①格式：修饰符(public private 缺省 protected) 数据类型　名字 = 初始化值
②属性的初始化值：如果不显式初始化的话，系统会根据属性的数据类型，隐式初始化。
当你显式的给属性赋值，那么属性经过默认初始化-->显式初始化这样的步骤。
③相对于成员变量，有局部变量
局部变量：在方法体内部（或在代码块内部或方法的形参部分）定义的变量。
格式：　数据类型　名字 = 初始化值
初始化值：必须要显式的初始化。系统不会提供默认初始化值。

成员变量和局部变量的区别与联系：
* 相同点：1）变量，在声明的时候，都需要指定数据类型和变量名。
* 2）都有生命周期。
* 不同点：1）声明的位置不同
* 2）在内存中的加载不同：成员变量随着 new Person()的加载，而加载到堆空间中
* 局部变量是加载在栈空间。
* 3）初始化值：成员变量：有默认初始化值，当然也可以显式的初始化
* 局部变量：除形参部分的局部变量外，必须显式的初始化
* 4）访问修饰符：　成员变量：需要有访问修饰符.访问权限从大到小有：public protected 缺省状态 private
* 局部变量：不需要有访问修饰符。实际上它的访问权限由其所在的方法的权限所反映。

2）方法(method、函数、成员方法)
①格式：（写一个方法，从如下声明的 4 部分去考虑）
修饰符 返回值类型 方法名(形参 1 类型 形参 1 名字,形参 2 类型 形参 2 的名字，。。。。){
//方法体

}

②说明：1.在方法内部，可以调用当前类的属性。（有一个例外：在 static 声明的方法里，不能调用非 static 的属性。）

2.方法内部，可以声明局部变量

3.方法内部可以调用其他方法，但是不能定义方法。

* 方法的返回值类型： 有返回值的 & 无返回值的
* 有返回值的：在方法声明时，指定返回值的类型。在方法体的里面需要 return + 返回值类型的“实体”;
* 无返回值的：在方法声明的返回值类型位置写上 void.此时不需要返回值，即无 return.
*
* 记忆：void 和 return 像是一对冤家，你出现，我就躲起来不出现。

3.3 对象的创建

1. 创建对象的格式：

类名 类名的一个引用 = new 类对应的构造器;

比如：String str = new String("atguigu");

Person p = new Person();

创建类的多个对象，每个对象都在堆空间有独立的一块区域。对 a 对象的区域进行的操作，并不影响 b 对象或其它对象区域的内容。

2.匿名对象的创建

不定义对象的句柄，而直接调用这个对象的方法。这样的对象叫做匿名对象。

如：new Person().shout();

使用情况：

如果对一个对象只需要进行一次方法调用，那么就可以使用匿名对象。

我们经常将匿名对象作为实参传递给一个方法调用。

3.4 方法的重载

方法的重载（overload）：

1.满足的条件：“两同一不同” 同一个类中，同一个方法名，参数列表不同（参数类型，参数个数的不同）

注意： 至于方法的权限修饰符和返回值类型并不影响方法之间是否构成重载

参数名不作为是否构成方法重载的条件。

```
//返回两个整数的和
  int add(int x,int y){return x+y;}
//返回三个整数的和
  int add(int x,int y,int z){return x+y+z;}
//返回两个小数的和
  double add(double x,double y){return x+y;}
```

3.5 形参的参数传递

swap(int i ,int j)与 swap(int[] arr,int i ,int j)的区别

m,n　ar　0x1234
Java 的实参值如何传入方法呢？
　　Java 里方法的参数传递方式只有一种：值传递。　即将实际参数值的副本（复制品）传入方法内，
而参数本身不受影响。
1.如果是基本数据类型：就将基本数据类型的值赋给方法的形参部分。
2.如果是引用数据类型：就将引用变量 a 的地址值赋给方法的形参部分 b，这样，方法的形参部分 b 和引用的变量 a
　就指向同一份堆空间的内存区域。

3.6 可变形参

*　测试可变个数的形参。
 * 1.是指方法的参数部分的指定类型的的参数个数是可变的。
 * 2.个数可以从 0 个开始，到无穷多个
 * 3.格式: method(数据类型　...　数据的引用名){}
 * 4.可变参数方法的使用与方法参数部分使用数组是一致的。并且这个两种参数对应的方法名必须不同。
 *　如：method2(String[] str){}与 method2(String ... str){}不能同时出现在一个类中。
 * 5.定义可变参数方法时，要将可变参数写在参数部分的最后。---->一个方法中，最多有一个可变个数的形参!!
 * 6.method1(String ... str){}方法与 method1(String str){}构成重载

```
public void method1(String str){
    System.out.println(str);
}

public void method2(String[] str){
    for(int i = 0;i < str.length;i++){
        System.out.print(str[i] + "\t");
    }
}
public void method1(String ... str){
    for(int i = 0;i < str.length;i++){
        System.out.print(str[i] + "\t");
    }
}
```

3.7 递归方法

方法递归：一个方法体内调用它自身。
方法递归包含了一种隐式的循环，它会重复执行某段代码，但这种重复执行无须循环控制。

例题：已知有一个数列：f(0) = 1,f(1) = 4,f(n+2)=2*f(n+1) + f(n),其中 n 是大于 0 的整数，求 f(10)的值。

递归一定要向已知方向递归，否则这种递归就变成了无穷递归，类似于死循环。

3.8 构造器

1.构造器(constractor)的作用：1）创建类的对象　2）通过构造器的形参赋值，初始化对象的属性

2.如果声明一个类的构造器

1）构造器与类同名，有修饰符，有形参，没有返回值也无 void。

```
class A{
  public A(){
  //构造器内部代码
  }
}
```

2）类内部可以声明多个重载的构造器。

3.构造器的说明：

1）Java 语言中，每个类都至少有一个构造器

2）默认构造器的修饰符与所属类的修饰符一致

3）一旦显式定义了构造器，则系统不再提供默认构造器

4）一个类可以创建多个重载的构造器

5）父类的构造器不可被子类继承

3.9 封装与隐藏

通过设置类的属性为 private,使用 public 的方法调用和修改类中 private 的属性。它是封装和隐藏的一种表现形式。

```
public class Animal{
    private int legs;//将属性 legs 定义为 private，只能被 Animal 类内部访问
    public void setLegs(int i){    //在这里定义方法 eat() 和 move()
        if (i != 0 && i != 2 && i != 4){
            System.out.println("Wrong number of legs!");
            return;
        }
        legs=i;
    }
    public int getLegs(){
        return legs;
    }
}
```

3.10 四种权限修饰符

public protected 缺省 private

修饰符	类内部	同一个包	子类	任何地方
private	Yes			
(缺省)	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

对于class的权限修饰只可以用public和default(缺省)。
- public类可以在任意地方被访问。
- default类只可以被同一个包内部的类访问。

3.11this

1.this 可以用来修饰属性、方法和构造器
1）this 修饰属性和方法表示调用当前对象（或正在初始化的对象）的属性和方法。

```
public void display(){   //调用 display()方法的对象，即为当前对象
 this.name = "Lily";
 this.setAge(12);
}

public Person(String name,int age){   //此时 this 表示：正在初始化的对象
 this.name = name;
 this.age = age;
}
```

2）在类内部，构造器内，使用 this（参数列表）的形式，调用本类重载的构造器
要求：1）this（参数列表）一定要写在构造器的首行！
2）在类内部的构造器中，至少有一个构造器内部是没有 this(参数列表)的。

3.12 package & import

package :包。
写在源文件的第一行，声明源文件的“位置”。
package com.atguigu.java;

import 引入

如: import java.util.Scanner;　　表示导入 java.util 包下的某一个类，即 Scanner 类
import java.sql.*;表示导入 java.sql 包下的所有的类

然后，可以在程序中，直接使用。即:　　Scanner s　= new Scanner(System.in);

1)如果需要导入 java.lang 的话，可以省略不写。
2）如果不显式的 import 的话，也可以。　　java.util.Scanner s = new java.util.Scanner(System.in);
3）像 Date 这样，既存在于 java.util 包下，又存在于 java.sql 包下，如果在源程序中同时出现了两个 Date 类，那么
只能是使用 2）表示的方式。

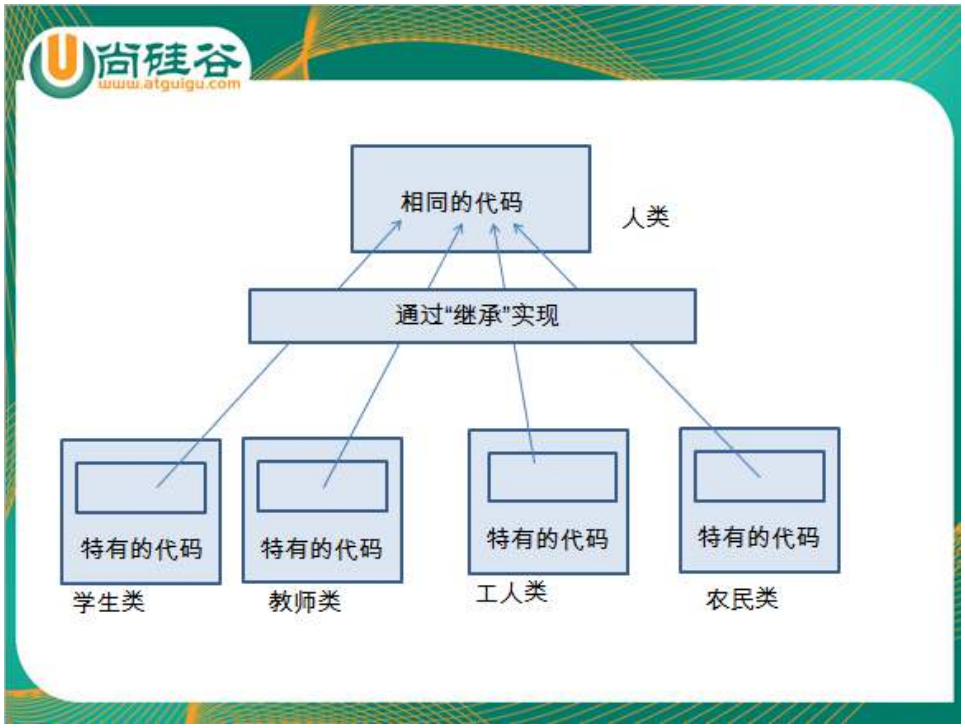
import static 在 JDK1.5 引入的。表示导入某个类下的某个静态方法、属性或全部的静态方法或属性

import static java.lang.Math.PI;
import static java.lang.System.*;

----------第 4 章：高级类特性 1 ----------

4.1 继承

1.继承的想法



2.继承的格式　　class A extends B{

}

其中，A 叫做子类，叫做父类

1）通过继承，子类 A 就获得了父类 B 的属性和方法。在父类权限修饰符允许的情况下，子类可以直接调用父类含有的
属性和方法。

2）子类不能继承父类的构造器。但是子类可以通过 super（参数列表）的形式调用父类的构造器

3）java 中只能单继承

```
class C extends B{}
class Dextends    A{}
```

4）在 java 语言中，所有类的根父类是 java.lang.Object 类。

4.2 方法的重写

1.重写的前提：一定要有继承；

2.重写的要求：1）返回值类型、方法名、形参列表（形参个数，形参类型）跟父类相同

2）访问权限：子类重写方法的访问权限不能小于父类被重写方法的访问权限

3）关于异常：子类跑出的异常不能大于父类被重写的方法抛出的异常

4）关于 static：重写和被重写的方法，必须同时为 static 或者同时为非 static 修饰的。

【重载（overload）和重写（override）的区别】
1.答出重载与重写对方法声明的要求。
2.重载：在同一个类中。
　　重写：是在子类中，“重写”的父类的方法。

4.3super 关键字

super ：对父类的方法、属性或构造器的引用。

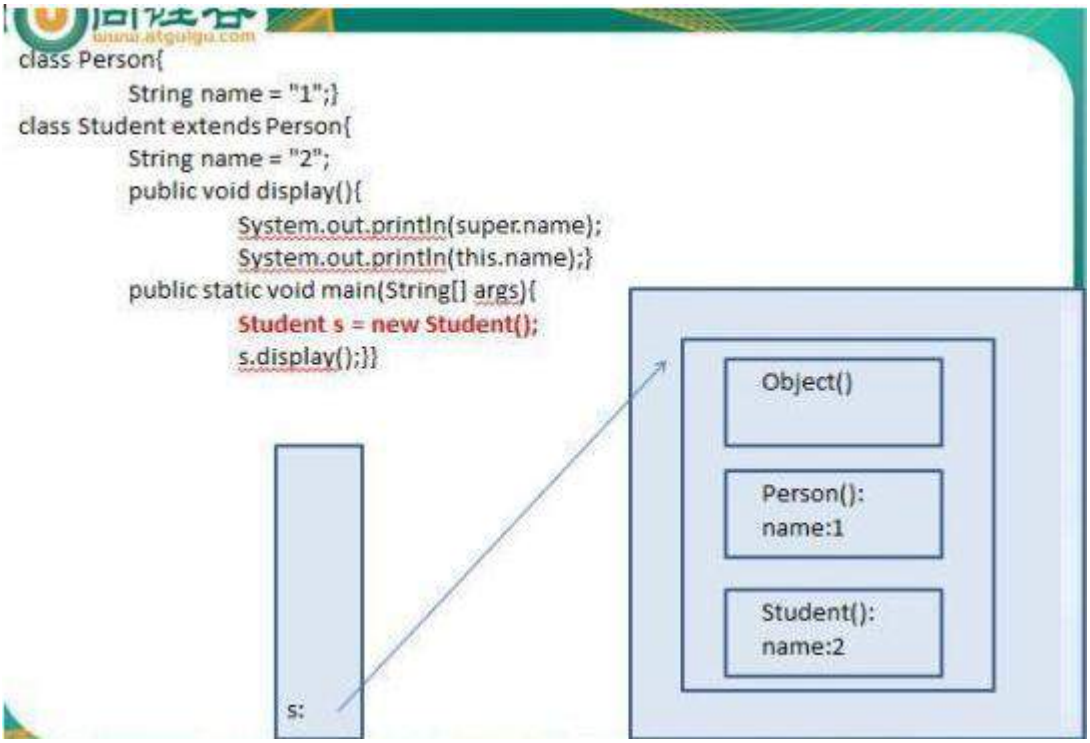
```
class Person{
    String name = "1";
}

class Student extends Person{
    String name = "2";
    public void display(){
        System.out.println(super.name);
        System.out.println(this.name);
    }

    public static void main(String[] args){
        Student s = new Student();
        s.display();
    }
}
```

super 修饰构造器：在子类的构造器中，可以通过显示的调用 super(参数列表）的形式，调用指定父类的构造器。
如果不显式的调用的话,子类的构造器也会默认去调用父类的 super()的构造器!!

4.4 子类对象实例化的全过程

```
class Person{
    String name = "1";
}

class Student extends Person{
    String name = "2";
    public void display(){
        System.out.println(super.name);
        System.out.println(this.name);
    }

    public static void main(String[] args){
        Student s = new Student();
```

```
        s.display();
    }
}
```



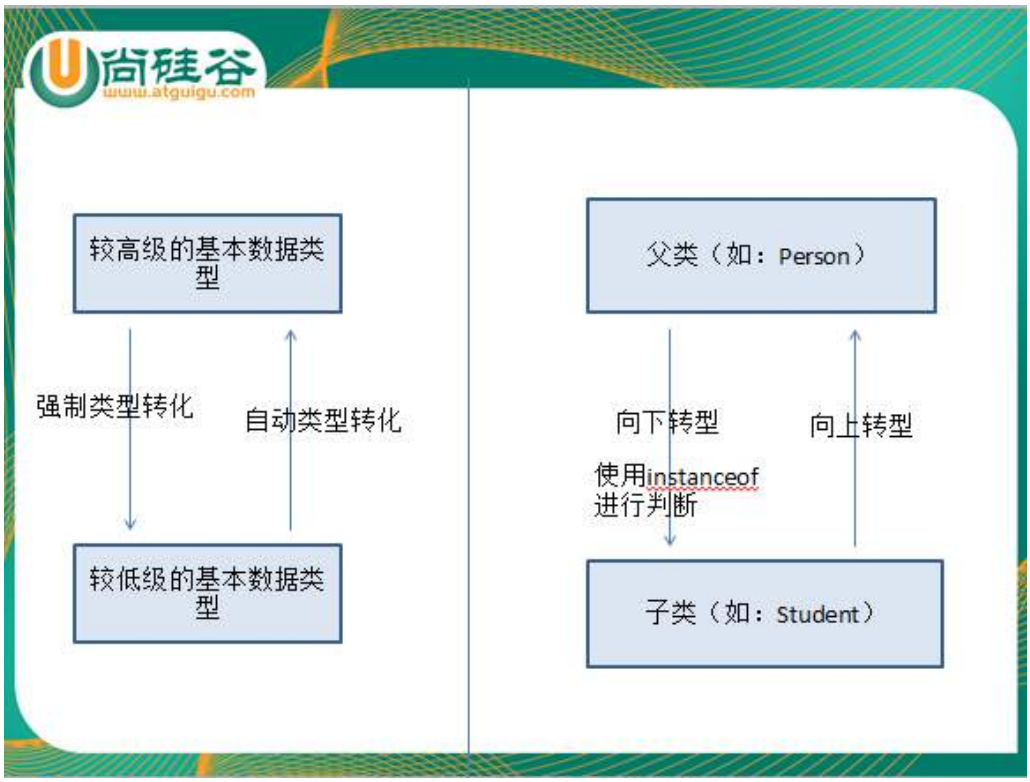
4.5 多态性

1.Java 中多态的表现形式：
1）方法的重载(overload)和重写(override)。
2）对象的多态性　——可以直接应用在抽象类和接口上
2.对象的多态性使用的前提：1）有继承关系　2）子类要重写父类的方法
3.Java 程序分为编译时状态和运行时状态。

```
class Person{
  public void method1(){
      //方法体
  }
}
class Student extends Person{

  public void method1(){
      //方法体，重写父类的 method1()方法
  }

  public static void main(String[] args){
      Student st = new Student();

      Person p = new Student();//对象的多态：即将子类的对象 new Student()
```

赋给父类的引用 p。

```
        //虚拟方法调用
        p.method1();//执行的是 Student 类中的重写的 method1()方法!

        //p....是无法调用 Student 类特有的属性和方法的!编译不通过。
        //但是堆空间的对象中，是含有 Student 类所特有的属性和方法的!
        //如何实现 p 可以调用 Student 类所特有的属性和方法呢？造型(或向下转型、强转)。
        //为了保证不报 ClassCastException ，需要使用 instanceof 关键字进行转型前的判断。
    }
  }

public class Test{
    public void func(Person p){
        p.method1();
    }
}
```



4.6Object 类

NO.	方法名称	类型	描述
1	public Object()	构造	构造方法
2	public boolean equals(Object obj)	普通	对象比较
3	public int hashCode()	普通	取得Hash码
4	public String toString()	普通	对象打印时调用

1.只有一个构造器 public Object();每次我们创建任何一个类的对象的时候，都会调用到 Object 类的无参的构造器！

2. public boolean equals(Object obj){}

Object 类的原码：

```
public boolean equals(Object obj) {
        return (this == obj);
}
```

2.1）对于 JDK 提供的 String、包装类、Date 等，已经重写了 Object 类的 equals()方法。

如：String 类重写的 equals(Object object)方法如下：

```
 public boolean equals(Object anObject) {
if (this == anObject) {
    return true;
}
if (anObject instanceof String) {
    String anotherString = (String) anObject;
    int n = value.length;
    if (n == anotherString.value.length) {
        char v1[] = value;
        char v2[] = anotherString.value;
        int i = 0;
        while (n-- != 0) {
            if (v1[i] != v2[i])
                return false;
            i++;
        }
        return true;
    }
```

```
}
return false;
}
```

2.2）自定义类的话，如果需要使用 equals 方法，那么就必须重写 Object 类的 equals();

```
//自定义类重写了 Object 类的 equals()方法
public boolean equals(Object obj) {
    if (this == obj)//引用地址是一样的，指向同一块堆空间的区域
        return true;

    if (obj instanceof Person) {
        Person p = (Person) obj;
        return (this.name.equals(p.name) && this.age == p.age);
    }

    return false;
}
```

```
3.public String toString(){
    }
```

Object 类的源码：

```
public String toString() {
        return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

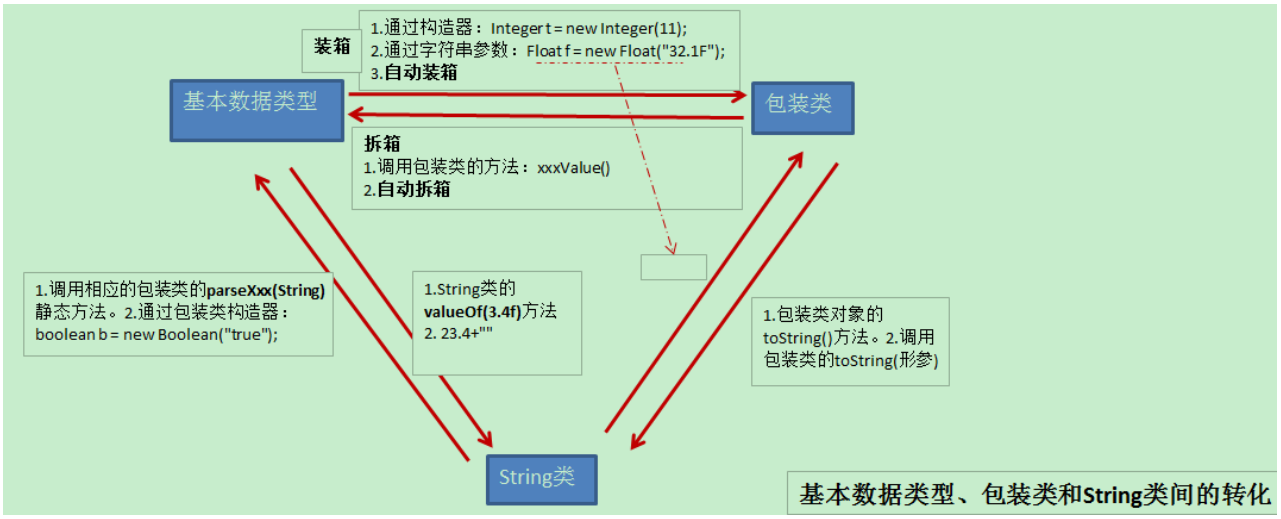
System.out.println(p); 相当于 System.out.println(p.toString());

3.1)对于 JDK 提供的 String、包装类、Date 等，已经重写了 Object 类的 equals()方法。

3.2)自定义类的话，如果需要使用 toString()方法，那么就必须重写 Object 类的 toString();

```
//自定义类重写了 Object 类的 toString()方法
public String toString() {
        return "Person [name=" + name + ", age=" + age + "]";
}
```

4.7 包装类



//对于基本数据类型和对应的包装类来说，有自动装箱和拆箱

```
Integer i = 23;//自动装箱
Integer j1 = new Integer(34);
Integer j2 = new Integer("34");
Boolean b = new Boolean("true");
int j = j1;
int j3 = j2;//自动拆箱
j1 = null;
boolean boo = b;
System.out.println(boo);//false     注意：不会报异常！


String str = "123";
//将 String 类型转化成基本数据类型(或包装类) :使用 Xxx 包装类的parseXxx(String str)方法
int k = Integer.parseInt(str);
System.out.println(k);

//基本数据类型(或包装类)转成 String 类
String str1 = String.valueOf(boo);
System.out.println(str1);

String str2 = boo + "";

String str3 = Boolean.toString(boo);
System.out.println(str3);
String str4 = b.toString();
```

----------第 5 章：高级类特性 2 ----------

5.1static 关键字

1.static :静态的　　实现功能：随着类的加载而加载，要早于对象的出现

2.static 可以用来修饰：属性 、 方法、代码块、内部类

3.static 修饰属性（类属性）

1）随着类的加载而加载，而类的消亡而消亡

2）先于对象创建

3）可以直接用类来调用

4）在内存中存在于数据区，只有一份。不同的对象共同使用同一份 static 的属性

5）当一个对象对 static 的属性进行修改，会导致其他对象调用此属性时，是修改过的属性。

4.static 修饰方法：

1）随着类的加载而加载，而类的消亡而消亡

2）先于对象创建

3）可以直接用类来调用

4）在内存中存在于数据区，只有一份。不同的对象共同使用同一份 static 的属性

5） 在 static 的方法内，只能调用 static 的属性或方法。

5.用 static 修饰属性或方法的利弊

利：可以直接通过类来调用，不用创建对象

弊：生命周期长，占用内存。

5.2 单例模式

单例模式：要求类只能创建一个对象。

懒汉式 & 饿汉式（存在线程安全问题）

5.3main 方法

public static void main(String[] args){}

5.4 代码块（或初始化块）

代码块（初始化块）： 用来初始化对象。

1. 静态代码块 和 非静态代码块

*静态代码块：

* 1.代码块内可以有多条语句：输出语句，赋值语句（只能对 static 的属性赋值）

* 2.随着类的加载而加载

* 3.静态的代码块只被加载一次

* 4.静态代码块的执行要先于非静态的代码块

* 5.只能调用静态的变量或方法

*

* 非静态的代码块：

* 1.代码块内可以有多条语句：输出语句，赋值语句（对 static 的或非 static 的属性赋值）

* 2.随着对象的创建才被加载

* 3.每次创建一个对象，非静态代码块都会被执行

* 4.非静态的代码块执行的先后顺序按程序中其出现的先后顺序来执行。

* 5.可以调用静态或非静态的属性和方法。

重难点：创建一个对象，那个对象实例化的全过程。

1）子类和父类的问题

2）子类来说，属性的默认/显式初始化，代码块（静态/非静态），构造器。。。

经典的代码：

```
class Root{
    static{
        System.out.println("Root 的静态初始化块");
    }
    {
        System.out.println("Root 的普通初始化块");
    }
    public Root(){
        System.out.println("Root 的无参数的构造器");
    }
}
class Mid extends Root{
    static{
        System.out.println("Mid 的静态初始化块");
    }
    {
        System.out.println("Mid 的普通初始化块");
    }
    public Mid(){
        System.out.println("Mid 的无参数的构造器");
    }
    public Mid(String msg){
        //通过 this 调用同一类中重载的构造器
        this();
```

```
        System.out.println("Mid 的带参数构造器，其参数值："
            + msg);
    }
}
class Leaf extends Mid{
    static{
        System.out.println("Leaf 的静态初始化块");
    }
    {
        System.out.println("Leaf 的普通初始化块");
    }
    public Leaf(){
        //通过 super 调用父类中有一个字符串参数的构造器
        super("atguigu");
        System.out.println("执行 Leaf 的构造器");
    }
}
public class TestLeaf{
    public static void main(String[] args){
        new Leaf();
        new Leaf();
    }
}
```

5.5final 关键字

* final:表示：最终的，可以用来修饰类、属性、方法

 * 1.修饰类：修饰的类不能被继承。

 *

 * 2.修饰方法：这个方法不能被重写

 *

 * 3.修饰属性：这个属性值，不能被修改。即，此变量是一个常量。

* “哪里”可以为 final 的属性赋值：1.声明的时候 2.在代码块内部 3.构造器中

*

* static final 修饰变量 ：全局常量。

5.6 抽象类

abstract:抽象的，可以用来修饰类，修饰方法。

1.使用抽象类的前提：在类可以继承的基础上。

2.抽象类：abstract class 类名{} 表明此类不能被实例化。

2.1 抽象类内部可以有构造器！

2.2 抽象类内部不一定有抽象方法。

如：abstract class Person{

abstract void walk();

abstract void eat();

}

class Test{

public static void main(String[] args){

Person p = new Person();//不能被实例化

Person p1 = new Person(){

void walk(){

//方法体

}

void eat(){

//方法体

}

};

}

2.抽象方法：用 abstract 声明，同时，方法没有方法体！

如：public abstract void walk();

2.1 抽象方法所属的类一定是抽象类。

2.2 抽象方法只有被重写才有意义。（需要子类继承抽象类，然后重写抽象方法）

3. abstract 不能跟哪些关键字同时使用：final static private 构造器

4.设计模式：模板设计模式

5.7 接口

接口：彻底的抽象类 ,不可以被实例化

声明：
```
interface MyInterface{
    //属性(全局常量)
    public static final double PI =3.14；
    int E = 2;
    //方法（全部是抽象方法）
    public abstract int getId();
    void display();
}
```

1. 格式：（abstract）class MyClass extends Object implements MyInterface,Comparable{//可以实现多个接口

```
    //重写（所有的/部分的）抽象方法
}
```

2.一个类可以实现多个接口。

3.类需要重写接口中的所有的抽象方法，方可实例化。如果不的话，那么此类仍未抽象类

4.接口与接口间是可以继承的，而且可以多继承

5.设计模式：工厂设计模式、代理模式

5.8 内部类

1.内部的分类：成员内部类（静态的/非静态的） & 局部内部类

```
class Person{
    String name = "Tom";
```

```
    class Bird{
        String name = "小鸟";
        //4 的举例放在此处
    }
    static clas Dog{

    }
}
```

2.成员内部类来说：　1）创建非静态内部类的实例：1.1 创建外部类的实例　1.2 通过外部类的对象调用内部类的构造器

```
//1.
Person p = new Person();
//2.
Person.Bird b = p.new Bird();
```

　2）创建静态内部类的实例:直接通过外部类的类名来调用内部类的构造器。

```
Person.Dog d = new Person.Dog();
```

3.内部类作为类的一类成员，可以调用外部类的属性和方法。

注意：如果内部类是 static 的，那么只能调用外部类的 static 的属性和方法。

4.在内部类中调用的问题（尤其是当外部类的属性、内部类的属性、内部类的局部变量名都相同）

举例：

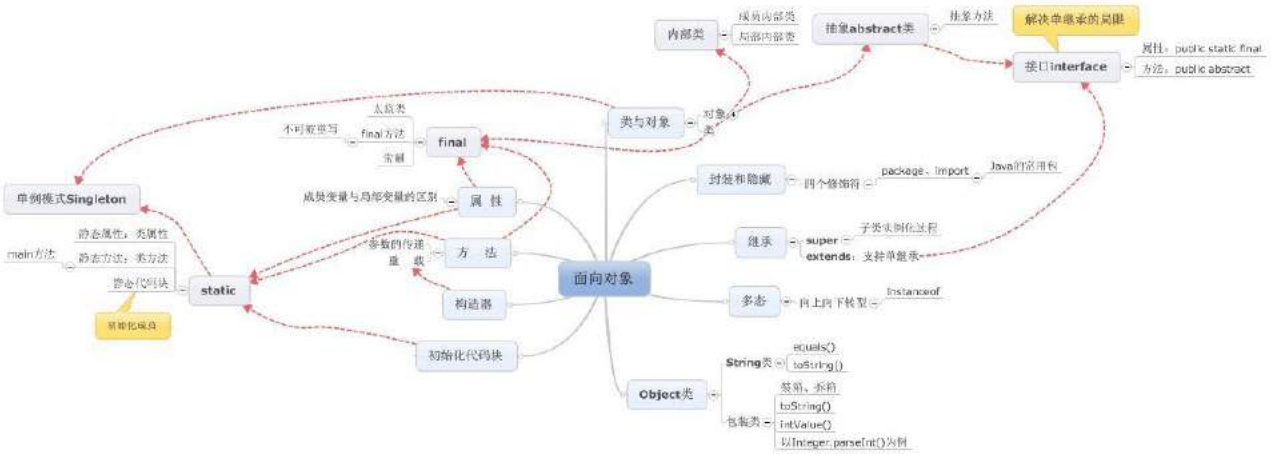
```
public void display(String name){
    System.out.println(name);//黄鹂
    System.out.println(this.name);//小鸟
    System.out.println(Person.this.name);//Tom
}
```

5.局部内部类：在方法里面定义的类。

```
//与上面定义的 get()方法是一样的。
    public Comparable get1(){
        class MyComparable implements Comparable{
            public int compareTo(Object obj){
                retrun 0;
            }
        }

        return new MyComparable();
    }
```

6.匿名内部类

5.9 面向对象总结



1.面向对象“三条主线”

----->关注与类的创建

1.1 考虑类内部都可以有什么？

属性　方法　构造器　代码块　内部类

1.2　面向对象的特征是什么?

封装和隐藏(四个权限修饰符)　　继承 extends　　多态

1.3 其他（几个重要的关键字、杂项）

几个重要的关键字：this　super　static final　abstract interface package import

杂项：Object 类　包装类（包装类、基本数据类型、String 间的转化）　main 函数　可变形参　方法的参数传递（难点）

关键点：子类对象实例化的全过程（类的内部哪些部分可以给属性赋值，以及先

后顺序；子父类间的调用）

----------第 6 章：异常处理 ----------

1.Exception ： RuntimeException(运行时异常)和 CheckedException(编译时异常）

1.1 运行时异常：可以不显式的进行处理。如果出现异常，会将异常的信息显式在控制台上。

几类常见的运行时异常：（4 种）

1.2 编译时异常：编译就不通过。必须要求程序员进行显式的处理！将编译时异常转化为运行时异常。

2.如何处理异常（处理异常的方式）

2.1 try-catch

```
try{
    //有可能出现异常的代码
  } catch(Exception1 e1){
        //如何处理的代码；
 }catch(Exception1 e2){
  }finally{
   //一定会被执行的代码
  }
```

说明：

1）try 中存放的是可能有异常的代码。如果 try 代码中发生异常，那么异常代码之后的代码不会被执行。

try 当中声明的变量，在出了 try 代码块后，将无效

2）catch 语句将异常类型小的放在异常类型大的上面。

在{}中给出异常的处理意见: e.printStackTrace() sysout(e.getMessage);

3)finally:不管是否发生异常，都会被执行：try 中是否有异常；catch 中是否有异常；catch 中是否有 return

4)catch 和 finally 是可选。

5)异常可以嵌套

2.2 throws :如果在方法中抛出了异常，但方法不知道如何处理，那么就通过 throws 的形式，将异常往上抛,交给方法的调用者来处理。

举例： public void method() throws Exception{

//存在异常

}

3.异常，既可以是 JDK 提供给我们的异常类，也可以是我们自己定义的异常类。

3.1.1 如果是 JDK 提供给我们的话，那么当出现异常的时候，程序会自动生成一个对应异常类的对象，然后抛出此对象。

3.1.2 如果是自定义的异常，需要在程序中显式的将这个异常对象抛出。如何抛出？使用 throw 关键字

3.2 throw 关键字：定义在方法内部，通过“throw 自定义异常类的对象”格式，抛给方法。

3.3 如何自定义异常类？①自定义异常类继承现有的异常类（一般情况下，继承 RuntimeException）②写几个重载的构造器

③static final long serialVersionUID = 948L;

【典型例题】

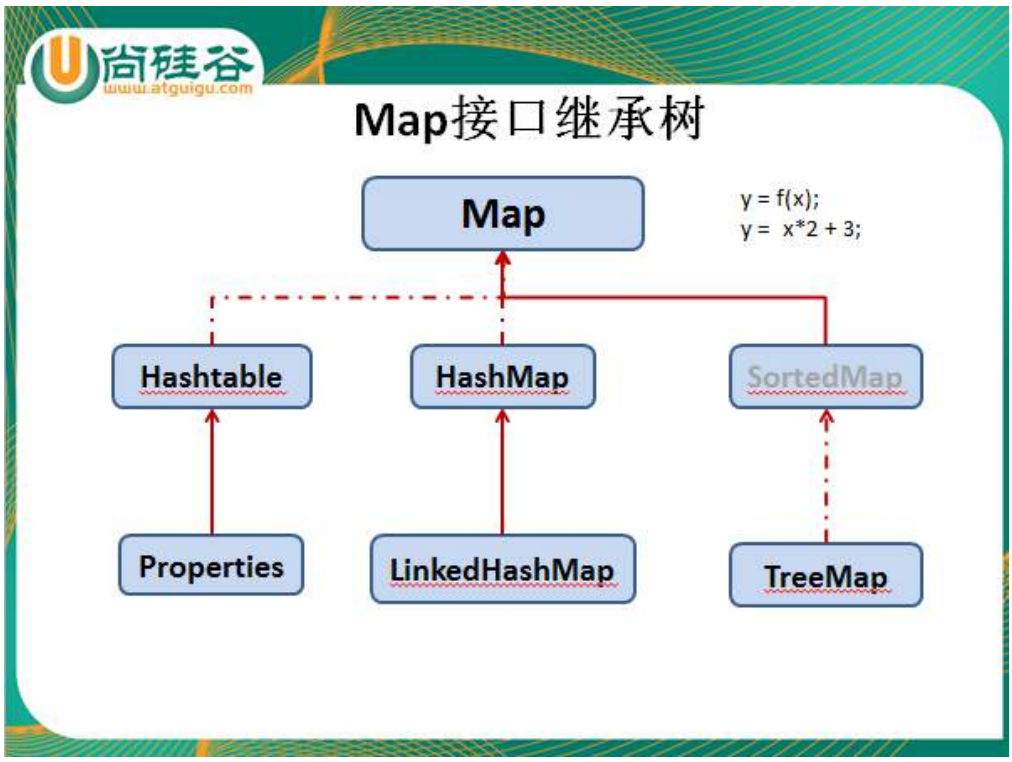
```
class EcDef extends RuntimeException{
    static final long serialVersionUID = 9L;
    public EcDef(String name){
        super(name);
    }
}
public class EcmDef {
    public static void main(String[] args) {
        try{
        int i    = Integer.parseInt(args[0]);
        int j    = Integer.parseInt(args[1]);
        ecm(i,j);
```

```
        }catch(NumberFormatException e){
            System.out.println("数据类型转化的异常");
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("数组下标越界的异常");
        }catch(ArithmeticException e){
            System.out.println("算术异常：/ by zero");
        }catch(EcDef e){
            System.out.println(e.getMessage());
        }finally{
            System.out.println("程序执行完毕！");
        }
    }

    public static void ecm(int m,int n) throws RuntimeException{
        if(m < 0 || n < 0){
            throw new EcDef("不能输入负数！");
        }else
            System.out.println("两数相除的结果为：" + m/n);

    }
}
```

----------第 7 章：Java 集合 ----------

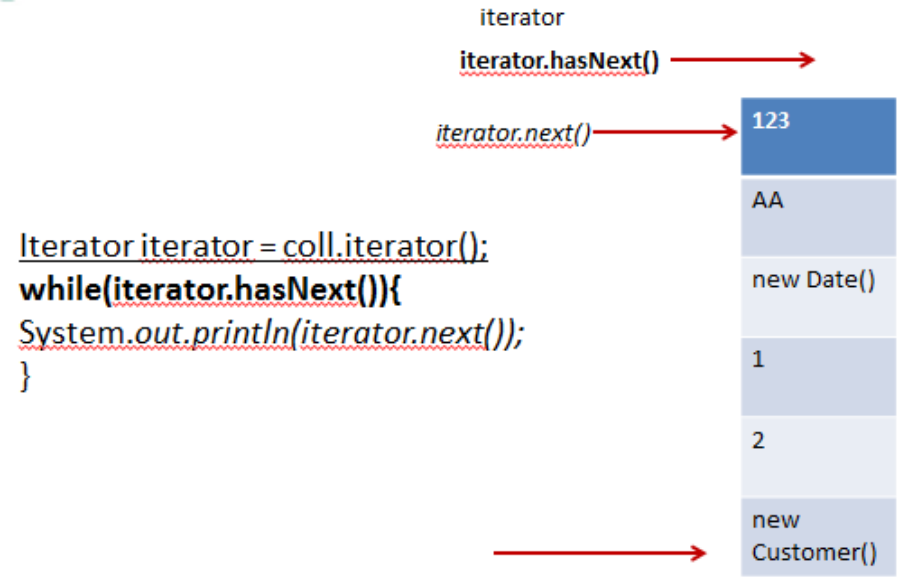
1. 主要有 Collection 和 Map 接口 （体系里的实线表示 extends 关系，虚线表示 implements 关系）

以及



7.1Connection 接口的常用方法

7.2 迭代器 Iterator 的使用：



1.用于实现集合（Set、List、Map）元素的遍历

2.通过集合类的 iterator()方法返回一个 Iterator 实现类的对象，用来遍历调用 iterator()方法的集合的元素

3.步骤：

```
Colleciton coll = new HashSet();
coll.add()……//添加诸多个元素
Iterator iterator = coll.iterator();
while(iterator.hasNext()){
        System.out.println(iterator.next());
}
```

7.3 Connection 的子接口：Set

1.存储的元素是无序的、不可重复的！类似于高中概念中的集合。

2.主要实现类：HashSet　　LinkedHashSet　　TreeSet

3.HashSet 是 Set 的主要实现类　　（可以添加 null 值，线程不安全的）

3.1 在不使用泛型的时候，创建 HashSet 可以用来存储任何数据类型的数据，即 add(Object obj);

3.2 如何判断添加元素的不可重复性？（重点、难点）

标准：1.如果添加的 JDK 提供给我们的类的对象，会自动判断是否重复

2.如果是添加自定义类的对象，自定义类一定要重写

equals()和 hashCode()

注：当两个对象的 equals() 方法比较返回 true 时，这两个对象的 hashCode() 方法的返回值也应相等

4.添加元素的时候如何使用 hashCode 和 equals()方法的？

4.1 先调用 hashCode 方法获取对象的 hash 值。如果 hash 值跟之前的对象值都不同，直接存储

4.2 如果 hash 值与之前的某个对象相同，那么再通过 equals()方法进行比较。如果 equals 返回 true，添加不成功。返回 false，添加成功

5.LinkedHashSet ①遍历时，是按照添加元素的顺序实现的。②是可以添加任何数据类型的。③遍历时，效率比较高

6.TreeSet 存入的元素需要排序，遍历时，按照指定的顺序遍历。

关键点：TreeSet 中只能添加相同类型的数据。

6.1 自然排序

1.对于自定义类，此类要实现 Comparable 接口，重写此接口的 CompareTo（Object obj）方法

2.创建 TreeSet 对象，使用 add()方法添加元素即可

注意点：当需要把一个对象放入 TreeSet 中，重写该对象对应的 equals() 方法时，应保证该方法与 compareTo(Object obj) 方法有
一致的结果：如果两个对象通过 equals() 方法比较返回 true，则通过 compareTo(Object obj) 方法比较应返回 0

6.2 定制排序

1.创建一个实现 Comparator 接口的匿名对象

2.将此对象作为形参传递给 TreeSet 的构造器形参部分

3.创建 TreeSet 对象，使用 add()方法添加元素即可

注意点：此时添加对象如果是自定义的，无需再实现 Comparable 接口

7.4Connection 的子接口：List

1.存储的数据是可重复的，是有序的。相当于可变长度的数组

2.实现类： ArrayList LinkedList（实现添加、删除、修改，效率高） Vector（线程安全，但是是个古老的实现类，现在不常用了）

3.典型实现类 ： ArrayList （线程不安全的）

```
List list = new ArrayList();
```

4.常用的三个方法： ① add(Object obj) ② Object get(int index) ③ int

size()

5.使用 Interator 迭代器实现遍历

7.5Map 接口及常用方法

1.存储的数据是以键值对（key-value）的形式存在的。类似于高中的函数

2.实现类： HashMap LinkedHashMap TreeMap Hashtable(古老类，将其子类：Properties）

3.HashMap 是 Map 接口的主要实现类 Map map = new HashMap();

4.常用方法： map.put(key,value);添加一个元素
map.size();获取键值对的个数
map.get(key):获取指定 key 的 value 值

5.Map 实现类的遍历（重点、难点）
Set keySet()
Collection values()
Set entrySet()
5.1 遍历 key 集

```
Set set = map.keySet();
for(Object obj : set){
    System.out.println(obj);
}
```

5.2 遍历 value 集

```
Collection coll = map.values();
Iterator i = coll.iterator();
while(i.hasNext()){
    System.out.println(i.next());
}
```

5.3 遍历 key-value 对集
（法一）

```
Set set1 = map.keySet();
for(Object obj : set1){
    System.out.println(obj + "----->" + map.get(obj));
}
```

（法二）

```
Set set2 = map.entrySet();
for(Object obj : set2){
    Map.Entry entry = (Map.Entry)obj;
    System.out.println(entry.getKey() + ":" + entry.getValue());
}
```

6.LinkedHashMap：按添加的顺序实现遍历

7.TreeMap ： 遍历时，按照添加元素时指定的顺序实现遍历。
要求，key 是一个 Set 集合。要求 key 必须实现自然排序或者是定制排序

8.Properties: 常用来操作属性文件 。key 和 value 都是 String 型的

7.6Collections 工具类

集合的工具类

可以用来操作：Set List Map

1.对于集合元素的一般方法： reverse(list) sort(list) ..

2.保证线程安全的同步方法： synchronizedXxx()

```
ArrayList list = new ArrayList();
list = Collections.synchronizecList(list);
```

----------第 8 章：泛型 ----------

泛型 ： JDK1.5 新加入的特性

使用：1.在集合中使用泛型

（在集合中使用泛型的声明和初始化）

```
Collection<String>  coll = new ArrayList<String>();
Collection<Object> coll1 = new ArrayList<Object>();
Map<Integer,String> map = new HashMap<>();
Iterator<String> iterator = coll.iterator();
```

注意点：1.一旦声明泛型，只能往里面添加声明类型的元素

2.不同泛型的声明之间不能相互引用

3.定义了泛型，但是初始化时没有指明类型，那么默认存入的是 Object 类型的元素

4.声明泛型的接口或抽象类是不可以实例化的

5.什么时候指明泛型类型？当我们实例化泛型类时，指明泛型类型

(题外话：数组的声明和初始化 String[] str = new String[4]; Integer i =

new Integer[]{21,3,5,67};)

2.自定义泛型类　　泛型方法　泛型接口 (E：　是 Object 类或其子类，不能是基本数据类型）

泛型类：　　class 类名<E>{

```
E e;
public void set(E e){
        this.e = e;
}
public E getE(){
        return e;
}
public void show(){
    System.out.println(e);
}
//泛型方法：
public <T> void fromArrayToCollection(T[] t,Collection<T> coll){
        for(T   tt : t){
            coll.add(tt);
        }
}
}
```

3.泛型跟继承的关系

List<Object> 、List<Number> 、List<Integer>三者之间没有关系！

4.通配符　<?>

List<Object> 、List<Number> 、List<Integer>都是 List<?>的子类

```
//public <？> void fromArrayToCollection(Collection<？> coll){
        //coll.add(tt);
//}
```

List<?>　只能够读取，不能够写入数据。（特殊：.add(null));

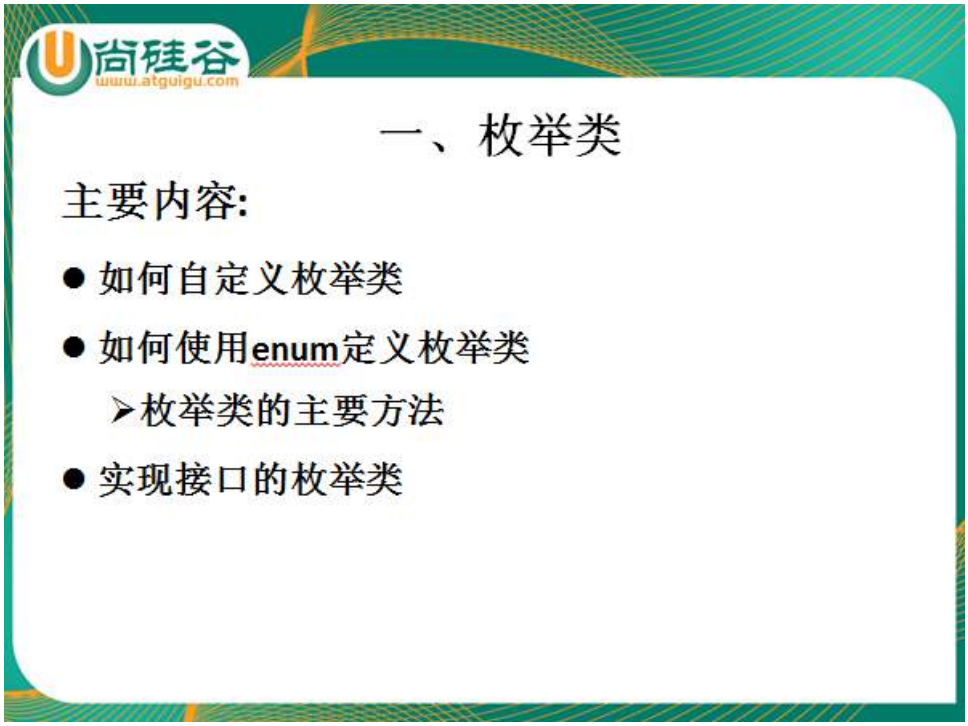
```
Set<Float> set = new Set<Float>;
        set.add(Float f);.....
Set<?> set1 = set;
for(Object obj : set1){
    System.out.println(obj);
}
```

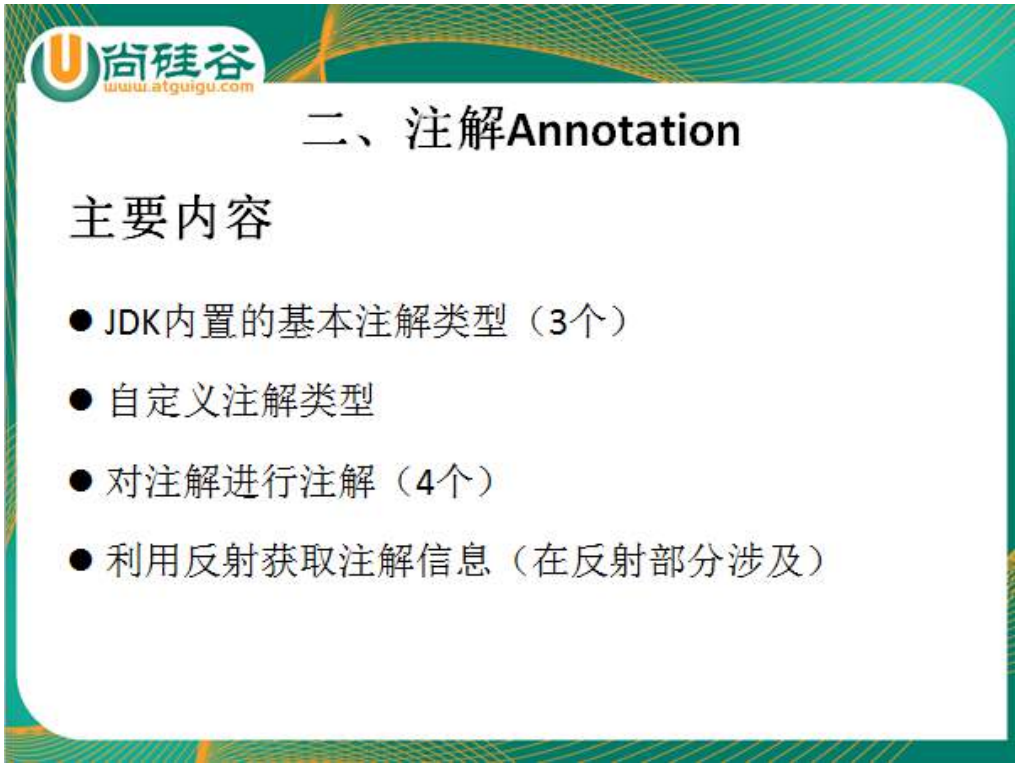
5.有限制的通配符

<? entends Number>　只能够放入 Number 类或其子类类型的数据。

<? super Number>　只能够放入 Number 类或其父类类型的数据。

----------第 9 章：注解 & 枚举 ----------





1.枚举：

1.1 自定义枚举类　　1.2 使用 enum 定义　1.3 values()　　valueOf(String str)

【典型代码 1】

```
class King{
    private final String kingName;
    private final String kingDesc;
    //枚举类的对象
    public static final King KING1 = new King("刘德华","拼命三郎");
    public static final King KING2 = new King("郭富城","舞神");
    public static final King KING3 = new King("黎明","长得帅");
    public static final King KING4 = new King("张学友","歌神");

    private King(String name,String desc){
        this.kingName = name;
        this.kingDesc = desc;
    }
```

```
    public String getKingName() {
        return kingName;
    }

    public String getKingDesc() {
        return kingDesc;
    }

    @Override
    public String toString() {
        return "King [kingName=" + kingName + ", kingDesc=" + kingDesc + "]";
    }
```

【典型代码 2】

```
enum King{
    //枚举类的对象
    KING1("刘德华","拼命三郎"),
    KING2("郭富城","舞神"),
    KING3("黎明","长得帅"),
    KING4("张学友","歌神");
    private final String kingName;
    private final String kingDesc;

    private King(String name,String desc){
        this.kingName = name;
        this.kingDesc = desc;
    }

    public String getKingName() {
        return kingName;
```

```
    }

    public String getKingDesc() {
        return kingDesc;
    }

    @Override
    public String toString() {
        return "King [kingName=" + kingName + ", kingDesc=" + kingDesc + "]";
    }

}
```

2.Annotation 元数据

2.1 JDK 内置的基本注解类型（3 个）

@Override: 限定重写父类方法, 该注释只能用于方法

@Deprecated: 用于表示某个程序元素(类, 方法等)已过时

@SuppressWarnings: 抑制编译器警告

2.2 自定义注解类型(仿照 suppresswarnings 写）

2.3 对注解进行注解（4 个）：元 Annotation

JDK5.0 提供了专门在注解上的注解类型，分别是：

Retention

Target

Documented

Inherited

----------第 10 章：IO 流 ----------

1.File ： 所代表的抽象路径，既可以是一个文件，可以一个是文件目录

1.1 File file = new File(String name);

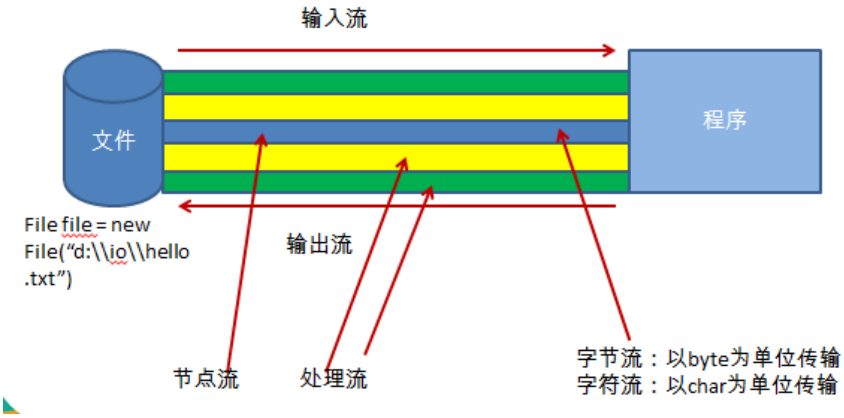
2. IO 流

2.1 IO 流的分类：

按操作数据单位不同分为：字节流(8 bit)，字符流(16 bit)

按数据流的流向不同分为：输入流，输出流

按流的角色的不同分为：节点流，处理流



以及，IO 流的体系

分类	字节输入流	字节输出流	字符输入流	字符输出流
抽象基类	InputStream	OutputStream	Reader	Writer
访问文件	FileInputStream	FileOutputStream	FileReader	FileWriter
访问数组	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
访问管道	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
访问字符串			StringReader	StringWriter
缓冲流	BufferedInputStream	BufferedOuputStream	BufferedReader	BufferedWriter
转换流			InputStreamReader	OutputStreamWriter
对象流	ObjectInputStream	ObjectOutputStream		
	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
打印流		PrintStream		PrintWriter
推回输入流	PushbackInputStream		PushbackReader	
特殊流	DataInputStream	DataOutputStream		

2.2　抽象基类　　节点流（重点）　　缓冲流（重点）

InputStream	FileInputStream	------->	BufferedInputStream
OutputStream	FileOutputStream	------->	BufferedOutputStream
Reader	FileReader	------->	BufferedReader
Writer	FileWriter	------->	BufferedWriter

2.3

【典型代码 1】

```
public void testFileInputStream(){
    InputStream is = null;
    try {
        File file = new File("Sat.txt");
        is = new FileInputStream(file);
        byte[] b = new byte[1024];
        int len;
        //
        while((len = is.read(b)) != -1){
            System.out.print(new String(b,0,len));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }finally{
        try {
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
```

```
    }
```
【典型代码 2】
```
public void testReaderWriter() throws IOException{
    Reader reader = new FileReader("dbcp.txt");
    Writer writer = new FileWriter("dbcp1.txt");
    char[] c = new char[1024];
    int len = 0;
    while((len = reader.read(c))!= -1){
        writer.write(new String(c,0,len));
        writer.flush();
    }

    writer.close();
    reader.close();
}
```
【典型代码 3】
```
//实现文件（文本文件、图片、视频等等）的复制
public static void copyFile(String src ,String des){
    FileInputStream fis = null;
    FileOutputStream fos = null;
    try {
        fis = new FileInputStream(new File(src));
        fos = new FileOutputStream(new File(des));
        byte[] b = new byte[1024];
        int len;
        while ((len = fis.read(b)) != -1) {
            fos.write(b, 0, len);
        }
    } catch (IOException e) {
```

```
            e.printStackTrace();
        } finally {
            try {
                fos.close();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
                    fis.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        }
}
```

【经典代码 4】

```
@Test
public void BufferedReaderWriter() throws IOException{
        Reader reader = new FileReader("dbcp.txt");
        Writer writer = new FileWriter("dbcp3.txt");
        BufferedReader br = new BufferedReader(reader);
        BufferedWriter bw = new BufferedWriter(writer);
        //BufferedReader br = new BufferedReader(new FileReader("dbcp.txt"));
        //BufferedWriter bw = new BufferedWriter(new FileWriter("dbcp1.txt"));
//          char[] c = new char[1024];
        String str = null;
        //int len = 0;
        while((str = br.readLine())!= null){
            bw.write(str);
```

```
            bw.newLine();
            bw.flush();
        }

        bw.close();
        br.close();
    }
```

【经典代码 5】使用缓冲流实现文件（文本文件、图片、视频等等）的复制。并通过实例比较此代码与【经典代码 3】的效率

```
public static void copyBufferedInputOutput(String src, String des)
            throws IOException {
        File file1 = new File(src);
        File file2 = new File(des);
        // 2.
        FileInputStream fis = new FileInputStream(file1);
        FileOutputStream fos = new FileOutputStream(file2);
        // 3.将 FileInputStream 的节点流对象作为形参传递给 BufferedInputStream 构造器。
        BufferedInputStream bis = new BufferedInputStream(fis);
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        // 4.实现复制操作的细节
        byte[] b = new byte[1024];
        int len;
        while ((len = bis.read(b)) != -1) {
            bos.write(b, 0, len);
            bos.flush();
        }

        bos.close();
        bis.close();
```

```
}
```

2.4 转换流： 实现从字节流到字符流的转换： InputStreamReader 和 OutputStreamWriter

```
@Test
public void testConvertStream() throws IOException{
    Reader r = new InputStreamReader(new FileInputStream("dbcp.txt"), "GBK");
    Writer w = new OutputStreamWriter(new FileOutputStream("dbcp4.txt"),"GBK");
    BufferedReader br = new BufferedReader(r);
    BufferedWriter bw = new BufferedWriter(w);
    String str = null;
    while((str = br.readLine()) != null){
        bw.write(str);
        bw.newLine();
        bw.flush();
    }

    bw.close();
    br.close();
    w.close();
    r.close();
}
```

2.5 标准输入输出流、数据流、打印流 （了解）

2.6 对象流 ： ObjectInputStream 和 ObjectOutputStream 可以用来读取和写入基本数据类型的数据以及 JDK 提供类、自定义类的对象。

2.6.1 ObjectInputStream 对应有 readObject()方法：对象的反序列化

ObjectOutputStream 对应有 writeObject(Object obj)方法：对象的序列化

2.6.2 要求存储的对象对应的类一定要实现序列化的机制（implements

Seriliazable) 同时，要求类的属性对应的类也必须实现序列化。

>凡是实现 Serializable 接口的类都有一个表示序列化版本标识符的静态变量：private static final long serialVersionUID = 12L;

>ObjectOutputStream 和 ObjectInputStream 不能序列化 static 和 transient 修饰的成员变量

2.7 RandomAccessFile :支持随机读、取文件，一定有 read 方法和 write 方法

>对于文件的写入，支持文件的开头、中间位置的插入，以及末位的写入。

注意：插入操作实现的对源文件对应位置的“覆盖”，而非“插入”。

----------第 11 章：多线程 ----------

1. 程序 进程 线程 （了解）

2. 如何实现多线程的创建（两种方式）和启动

1.继承 Thread 类

1）子类继承 Thread 类

2）重写 Thread 类的 run()方法？ run()中是创建的多线程要实现的功能。

3）创建子类的对象（创建了几个，就有几个线程）

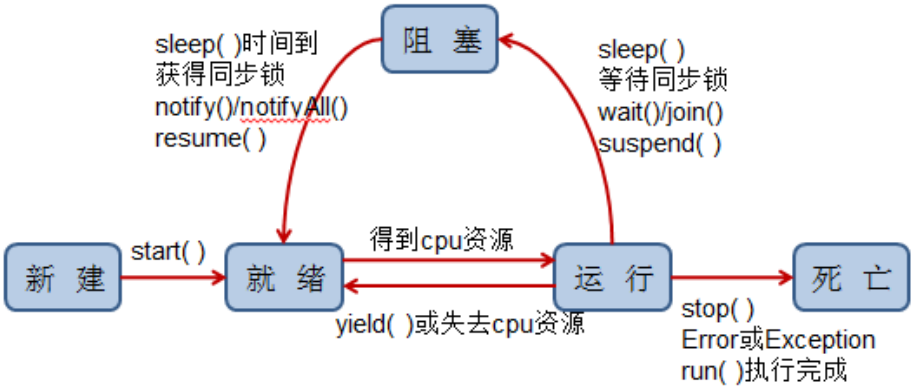
4）启动：对象.start();

2.实现 Runnable 接口

1）子类实现 Runnable 接口
2）重写 Runnable 接口的 run()方法？
3）创建一个子类的实例
4）将子类的实例作为实参传递给 Thread 的构造器中？
5）启动：对象.start();
>比较两种方式的区别：联系，不同点，哪个好

3.涉及 Thread 类的几个方法：

4.多线程的生命周期：



线程状态转换图

5.线程的同步（解决多线程的安全问题）

前提：多个线程操作共享数据
解决方法：在其中一个线程 A 对共享数据修改的时候，其它线程必须在外面等候，当 A 线程执行完共享数据后，其它线程方可参与执行共享数据。

5.1　同步代码块

```
1.synchronized(对象){
 *       //涉及共享数据的代码
 *   }
 *
```

5.2 同步方法

```
 *   2.public synchronized void show(){
 *       //涉及共享数据的代码
 *   }
```

>对于非 static 的同步方法，它对应的锁是 this，即当前对象。
>对于 static 的同步方法，它对应的锁是此方法所在的类。　XxxYyy.class

5.3 死锁（写程序时，需要避免的）

不同的线程分别占用对方需要的同步资源不放弃，都在等待对方放弃自己需要的

同步资源，就形成了线程的死锁

6.线程通信

wait()　notify()　notifyAll()　一定使用在同步代码块或同步方法中。

6.1 会释放同步锁的操作

6.2 不会释放同步锁的操作

----------第 12 章：Java 常用类 ----------

1.String 类　StringBuffer 类　StringBuilder 类

String 类：不可变的字符串序列

1.常用的方法

2.String 类与基本数据类型（或包装类）的转换

1）字符串---->包装类：　如：Integer.parseInt(String str)　2.包装类 ---->字符串：String.valueOf(Xxx xxx)

3.String 类与字符数组、字节数组的转换

* 1.字节数组---->字符串：　String str = new String(byte[] b);

* 2.字符串 ----->字节数组： byte[] b = str.getBytes();

* 3.字符数组 ----->字符串： String str = new String(char[] c);

* 4.字符串 -----> 字符数组：char[] c = str.toCharArray();

StringBuffer 是可变的字符串序列

1.相比 String 类，多了一些方法：append() reverse() insert replace()....

StringBuilder 是可变的字符串序列 : JDK1.5 新加的，处理字符串的效率更高，线程不安全的。

>三者的效率测试 v(StringBuilder) > v(StringBuffer) > v(String)

2.日期类相关的

2.1 System.currentTimemillis();

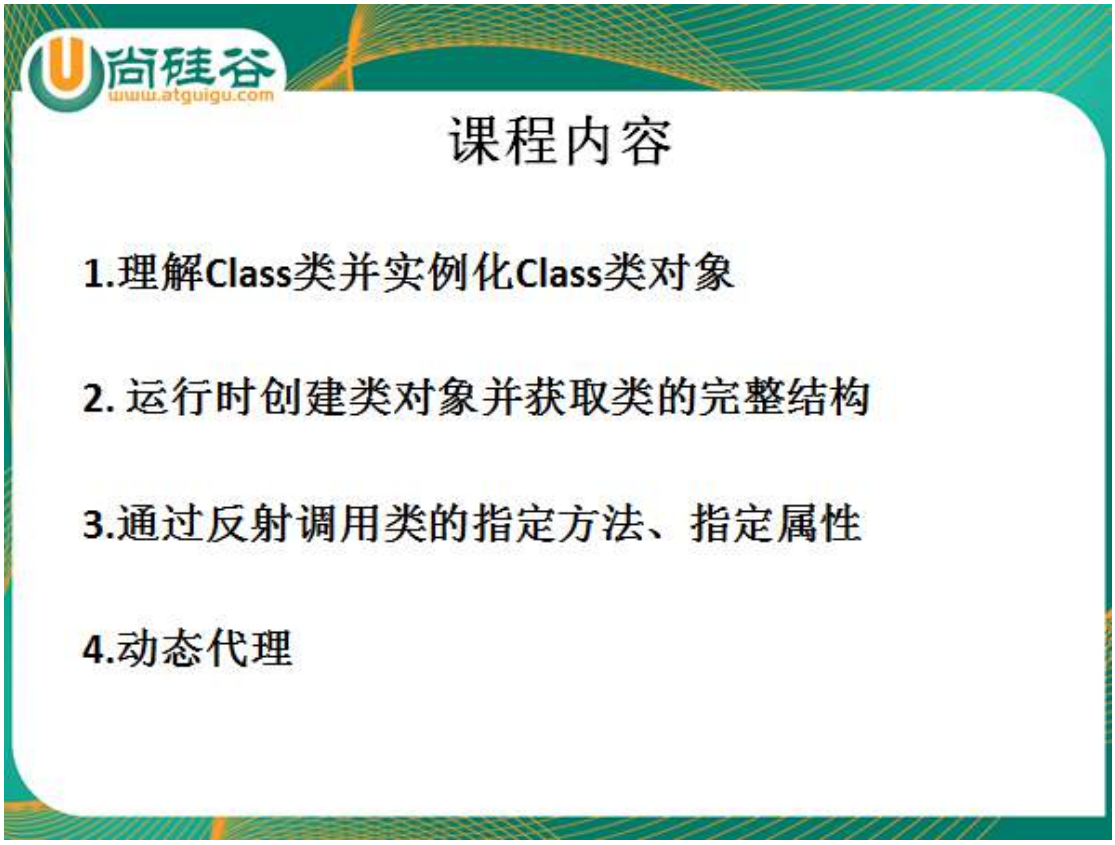
2.2 Date 类 java.util 包下 Date date = new Date(); date.getTime();

2.3 SimpleDateFormat 类 1.格式化：日期 Date ---->文本 2.解析： 文本---->日期 Date

2.4 Calendar 类

3.Math BigInteger BigDecimal

----------第 13 章：Java 反射 ----------



13.1 理解类的加载过程

任何一个类编译后生成一个.class 文件，JVM 的类加载器将此.class 文件加载到内存中。.class 文件就对应有一个 java.lang.Class 类的实例。

可以理解为，加载的类本身就作为 Class 类的实例

>涉及到的包/类：①java.lang.Class 可以看做反射的根源 ②java.lang.refect

13.2 如何实例化 Class 类（掌握）

```
//1.直接调用类的属性.class
Class clazz = Person.class;
clazz = Animal.class;
//2.调用 Class 类的静态方法 forName(全类名),可能抛 ClassNotFoundException 异常
Class clazz2 = Class.forName("com.atguigu.exer.Animal");
System.out.println(clazz2);

//3.通过运行时类的对象的 getClass()方法获取
```

```
Person p = new Person();
Class clazz3 = p.getClass();
System.out.println(clazz3);

//4.了解
ClassLoader classLoader = this.getClass().getClassLoader();
Class clazz4 = classLoader.loadClass("com.atguigu.exer.Animal");
```

13.3 通过 Class 对象可以创建运行时类（即 Class 对象对应的类）的对象，以及获取运行时类完整的类结构

3.1） Person p = (Person)clazz.newInstance();

```
Constructor constructor = clazz.getDeclaredConstructor(String.class,Integer.class);
Person p1 = (Person)constructor.newInstance("小明",23);
```

3.2)通过 Class 对象获取类的完整结构：包名、实现的接口、父类、带泛型的父类、注解（类上、属性、方法、构造器）、属性、方法、构造器、内部类

以及对应的属性的修饰符、声明的类型、名字；方法的修饰符、返回值类型、方法名、形参列表

13.4 创建运行时类的对象，并获取、调用指定的类结构

【典型代码 1】创建运行时类的对象，并获取指定的属性为其赋值

```
@Test
public void getOneField() throws Exception{
    //1.创建运行时类的对象
    Class clazz = Person.class;
    Person p = (Person)clazz.newInstance();
    System.out.println(p);
```

```
        //2.获取运行时类对象的属性
        Field field = clazz.getField("name");
        //如果属性声明为 private,那么只能 getDeclaredField(String str)的方式调用
        Field field1 = clazz.getDeclaredField("age");
        //3.给运行时类的对象的相应属性赋值
        //field.set(Object obj,赋值 a): 给 obj 的对象对应的 field 属性赋值为 a。
        field.set(p, "小丽");
        //当试图对声明为 private 的属性进行修改时，必须先调用属性的 setAccessible(true)，方可对属性进行修改（赋值）
        field1.setAccessible(true);
        field1.set(p, 22);
        //4.输出此对象
        System.out.println(p);
    }
```

【典型代码 2】创建运行时类的对象，并获取指定的方法（考虑给方法的形参赋值）供对象调用。

```
@Test
    public void getOneMethod() throws Exception{
        //1.创建运行时类的对象
        Class clazz = Person.class;
        Person p = (Person)clazz.newInstance();
        //2.获取运行时类对象的方法
        Method method = clazz.getMethod("display", String.class,Integer.class);
        Method method1 = clazz.getDeclaredMethod("show", null);
        //3.调用运行时类的对象的方法，同时若方法有形参，需要给形参赋值。
        //invoke(Object obj,Object...args):给 obj 对象相应方法的形参赋值为 Object...args
        method.invoke(p, "张三",33);
        //System.out.println(method);
```

//当试图对声明为 private 的方法进行调用时，必须先调用方法的setAccessible(true)，方可对方法进行调用

```
        method1.setAccessible(true);
        method1.invoke(p, null);
    }
```

13.5 动态代理

----------第 14 章：网络编程 ----------

14.1 实现网络编程：

1.需要知道 Ip 地址和端口号
2.网络协议

14.2 Ip

用到一个类 InetAddress 类
14.2.1 创建 InetAdress 对象：InetAdredd inetAddress = InetAddress.getByName("www.atguigu.com");
InetAdredd inetAddress =

InetAddress.getByName("192.168.4.32");

14.2.2 调用 InetAddress 的 getLocalHost()方法，获取本机的 Ip 地址。

14.2.3 主要的方法：getHostName()：获取域名 getHostAddress()：获取 IP 地址

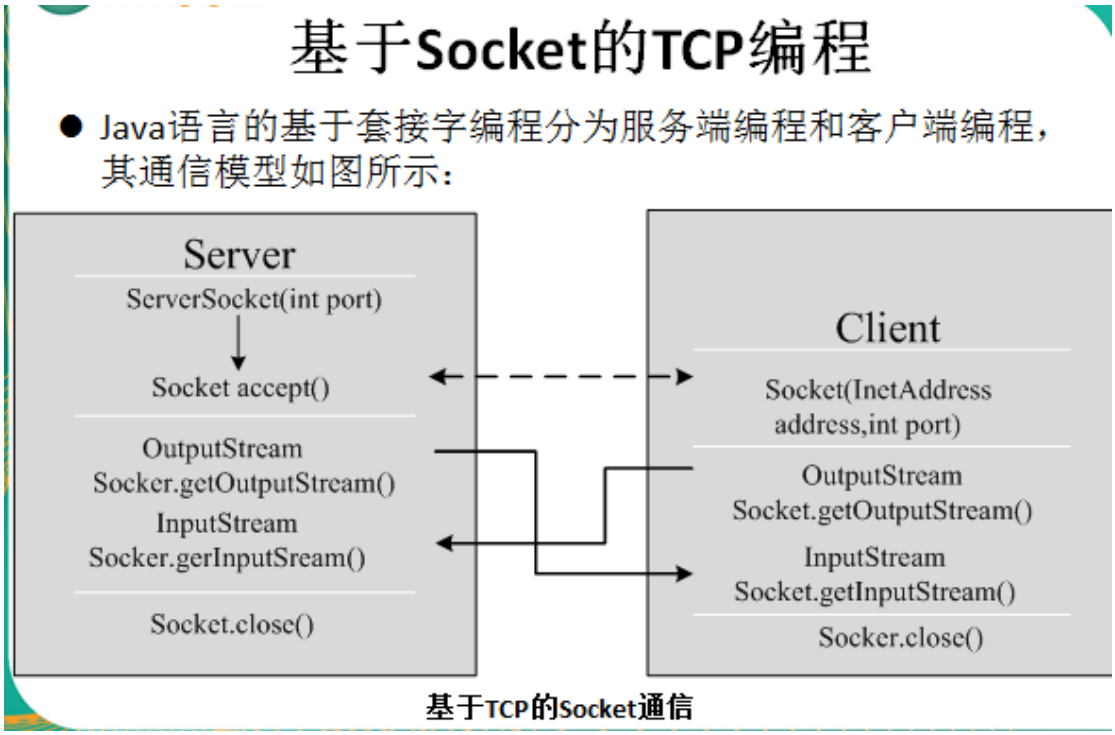
14.3 网络协议

网络通信协议

OSI参考模型	TCP/IP参考模型	TCP/IP参考模型各层对应协议
应用层	应用层	HTTP、ftp、telnet、DNS...
表示层		
会话层		
传输层	传输层	TCP、UDP、...
网络层	网络层	IP、ICMP、ARP...
数据链路层	物理+数据链路层	Link
物理层		

14.4 TCP 协议编程

安全，效率比较低；客户端&服务端；通过 IO 流的形式，实现客户端和服务端的交互



实现客户端和服务端的编程：

1）客户端：　① Socket socket = new Socket(InetAddress address, int port);
或者
Socket socket = new Socket(String host, int port);
② 调用 socket 对象的 getInputStream() 和 getOutputStream()方法实现与服务端的传输
③ 关闭相应的流和 socket 对象
2）服务端：　① ServerSocket　serverSocket = new ServerSocket(int port);
② Socket socket1 = severSocket.accept();
③调用 socket1 对象的 getInputStream() 和 getOutputStream()方法实现与客户端的传输、
④关闭相应的流和 socket 对象、serverSocket 对象

14.5UDP 协议编程

不安全，效率高；传输端&接收端；通过 IO 流的形式，实现客户端和服务端的交互

①创建传输端和接收端的对象　②对于传输端，如何创建数据报：DatagramPacket　,包含着要传输的内容，以及接收端
的 ip 和端口号。

【典型代码】

```
@Test
public void testSend() throws IOException{
    DatagramSocket ds = new DatagramSocket();
    byte[] by = "hello,atguigu.com".getBytes();
    DatagramPacket dp = new DatagramPacket(by,0,by.length,
            InetAddress.getByName("127.0.0.1"),10000);
    ds.send(dp);
    ds.close();
}
@Test
public void testReceive() throws IOException{
    DatagramSocket ds = new DatagramSocket(10000);
    byte[] by = new byte[1024];
    DatagramPacket dp = new DatagramPacket(by,by.length);
    ds.receive(dp);
    String str = new String(dp.getData(),0,dp.getLength());
    System.out.println(str+"--"+dp.getAddress());
    ds.close();

}
```

14.6URL(Uniform Resource Locator)

统一资源定位符，它表示 Internet 上某一资源的地址。

1.URL　url = new URL("http://localhost:8080/examples/myTest.txt");

2.　URL 类常用的方法

public String getProtocol(　)　　获取该 URL 的协议名

public String getHost(　)　　获取该 URL 的主机名

public String getPort(　)　　获取该 URL 的端口号

public String getPath(　)　　获取该 URL 的文件路径

public String getFile(　)　　获取该 URL 的文件名

public String getRef(　)　　获取该 URL 在文件中的相对位置

public String getQuery(　　)　　获取该 URL 的查询名

3.　如果仅需要读取网络资源：URL 的方法 openStream()：能从网络上读取数据

如果希望将网络资源存储在本地：①　调用 URL 的 openConnection()　方法，返回一个 URLConnection 对象

②调用 URLConnection 对象的 getInputSteam()方法

【典型代码】

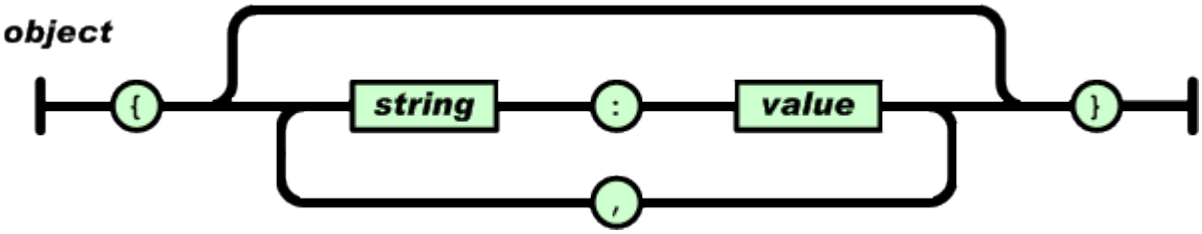
```
//获取 URL 对象的资源
//1.openConnection 的方法，获取 URLConnection 的对象
URLConnection urlconn = url.openConnection();
//2.调用 URLConnection 的 getInputStream()方法，获取输入流
InputStream is = urlconn.getInputStream();

OutputStream os = new FileOutputStream(new File("myTest1.txt"));
//3.实现细节
byte[] b= new byte[1024];
int len;
while((len = is.read(b)) != -1){
    os.write(b, 0, len);
}
```

题目：Android 开发之解析 json

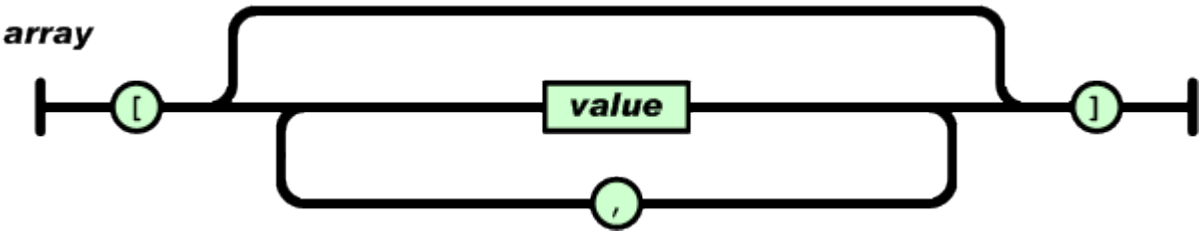
JSON 数据格式，在 Android 中被广泛运用于客户端和网络(或者说服务器)通信，非常有必要系统的了解学习。

1.Json 必知必会

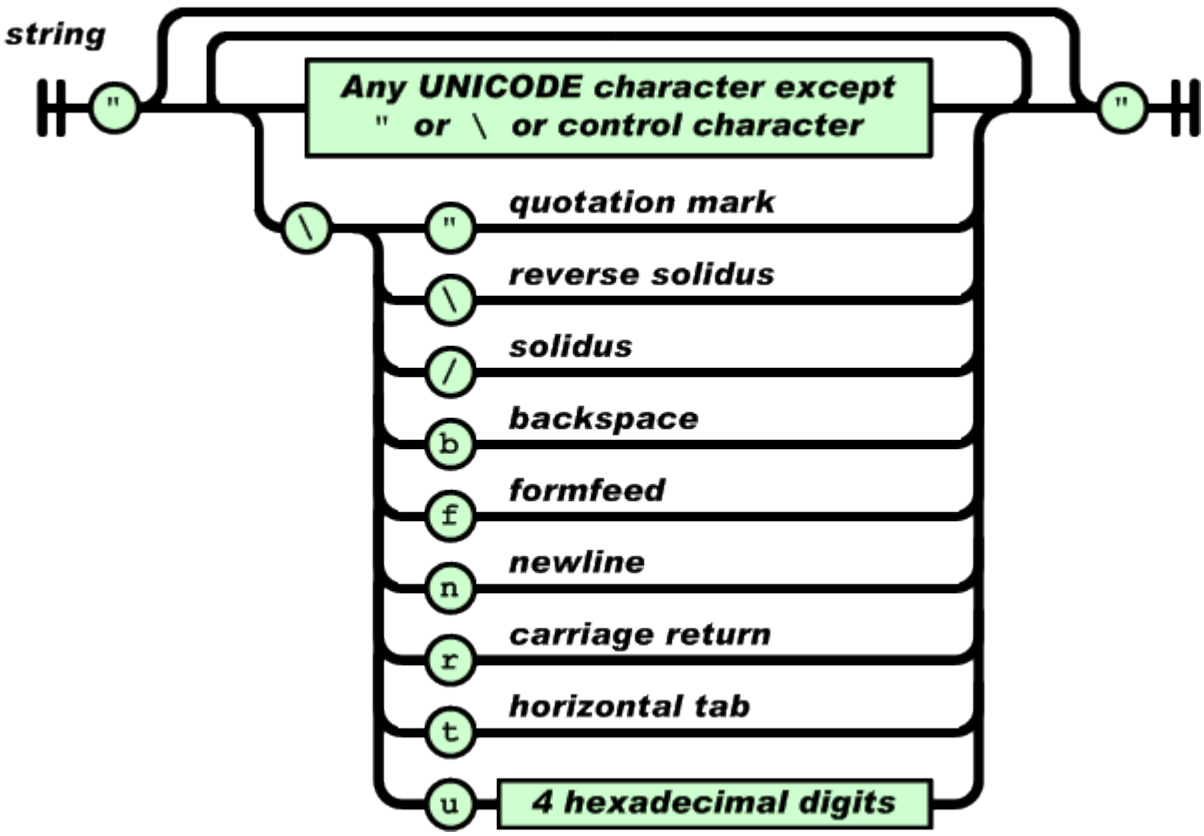
(1).JSON 是一种轻量级的数据交换格式
(2).JSON 基于两种数据结构：Object 和 Array。其中 Object 是“名称/值”对的集合。
(3).对象：大括号，每一组 string-value 结合以","分隔，string 和 value 以冒号分隔。



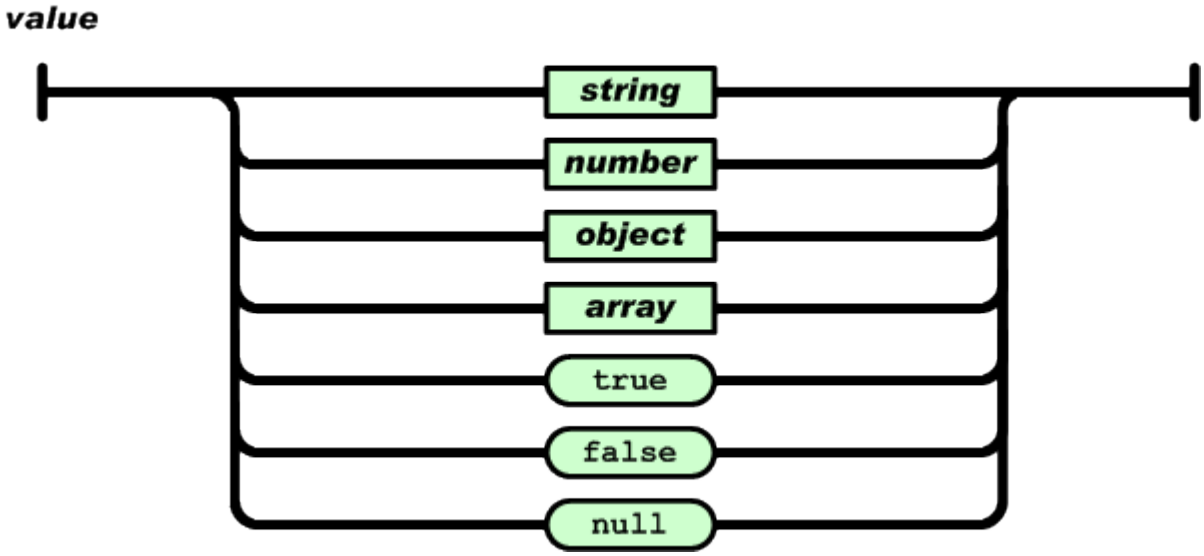
(4).数组：



(5).string 由双引号包围的任意数量 Unicode 字符的集合，使用反斜线转义。



(6).value 可以是双引号括起来的字符串（string）、数值(number)、true、false、 null、对象（object）或者数组（array）。这些结构可以嵌套。



(7)举例：

Object 实例：

```
{
    "Image": {
     "Width":  800,
     "Height": 600,
```

```
    "Title":  "View from 15th Floor",
    "Thumbnail": {
      "Url":   "http://www.atguigu.com/",
      "Height": 125,
      "Width":  "100"
    },
    "IDs": [116, 943, 234, 38793]
  }
}
```

Array 实例：

```
[
  {
    "precision": "zip",
    "Latitude":   37.7668,
    "Longitude": -122.3959,
    "Address":   "",
    "City":     "SAN FRANCISCO",
    "State":    "CA",
    "Zip":      "94107",
    "Country":   "US"
  },
  {
    "precision": "zip",
    "Latitude":   37.371991,
    "Longitude": -122.026020,
    "Address":   "",
    "City":     "SUNNYVALE",
    "State":    "CA",
    "Zip":      "94085",
    "Country":   "US"
  }
]
```

2.Json 的解析:使用原生的 API

(1).解析 Object 之一:

```
1   {"url":"http://www.atguigu.com/android"}
```

解析方法：

```
1 JSONObject demoJson = new JSONObject(jsonString);
2 String url = demoJson.getString("url");
```

(2).解析 Object 之二:

```
1    {"name":"android","version":"6.0"}
```

解析方法:

```
1 JSONObject demoJson = new JSONObject(jsonString);
2 String name = demoJson.getString("name");
3 String version = demoJson.getString("version");
4 System.out.println("name:"+name+",version:"+version);
```

(3).解析 Array 之一:

```
1    {"number":[1,2,3]}
```

解析方法：

```
1 JSONObject demoJson = new JSONObject(jsonString);
2 JSONArray numberList = demoJson.getJSONArray("number");
3 for(int i=0; i<numberList.length(); i++){
4   //因为数组中的类型为 int，所以为 getInt，其他 getString，getLong 同用
5   System.out.println(numberList.getInt(i));
6 }
```

(4).解析 Array 之二:

```
1 {"number":[[1],[2],[3]]}
```

解析方法:

```
1 //嵌套数组遍历
2 JSONObject demoJson = new JSONObject(jsonString);
3 JSONArray numberList = demoJson.getJSONArray("number");
4 for(int i=0; i<numberList.length(); i++){
5    //获取数组中的数组
6    System.out.println(numberList.getJSONArray(i).getInt(0));
7 }
```

(5).解析 Object 和 Array:

```
1    {"mobile":[{"name":"android"},{"name":"iphone"}]}
```

解析方法:

```
1 JSONObject demoJson = new JSONObject(jsonString);
2 JSONArray numberList = demoJson.getJSONArray("mobile");
3 for(int i=0; i<numberList.length(); i++){
4     System.out.println(numberList.getJSONObject(i).getString("name"));
5 }
```

3.Json 的解析:使用 Google 提供的 GSON 的 API

(1) 将 Json 对象转换为 Java 对象

```
String json = "{\"id\":2, \"name\":\"大虾\", \"price\":12.3,
\"imagePath\":\"http://192.168.10.165:8080/L05_Server/images/f1.jpg\"}";

Gson gson = new Gson();
//要求 json 对象的 key 值与 Java 类的属性名必须一致!
//使用了反射创建 ShopInfo 的对象，并给相应的属性赋值
ShopInfo shopInfo = gson.fromJson(json, ShopInfo.class);
```

(2)将 json 数组转换为 Java 对象构成的集合

```
String jsonArrayStr = "[{\"id\":2, \"name\":\"大虾\", \"price\":12.3,
\"imagePath\":\"http://192.168.10.165:8080/L05_Server/images/f1.jpg\"},{\"id\":2,
\"name\":\"小蟹\", \"price\":2.3,
\"imagePath\":\"http://192.168.10.165:8080/L05_Server/images/f1.jpg\"}]";

Gson gson = new Gson();
List<ShopInfo> list = gson.fromJson(jsonArrayStr, new
TypeToken<List<ShopInfo>>(){}.getType());

for(ShopInfo s : list){
    Log.e("TAG", s.toString());
}
```

(3)将 java 对象转换为 Json 对象字符串

```
ShopInfo shopInfo1 = new ShopInfo(1001, "鲍鱼", 23.4, "www.baoyu.com");
Gson gson = new Gson();
String json = gson.toJson(shopInfo1);
```

(4)将 java 对象构成的集合转为 Json 数组

```
ShopInfo shopInfo1 = new ShopInfo(1001, "鲍鱼", 23.4, "www.baoyu.com");
ShopInfo shopInfo2 = new ShopInfo(1002, "鲍鱼 1", 23.42, "www.baoyu.com");
```

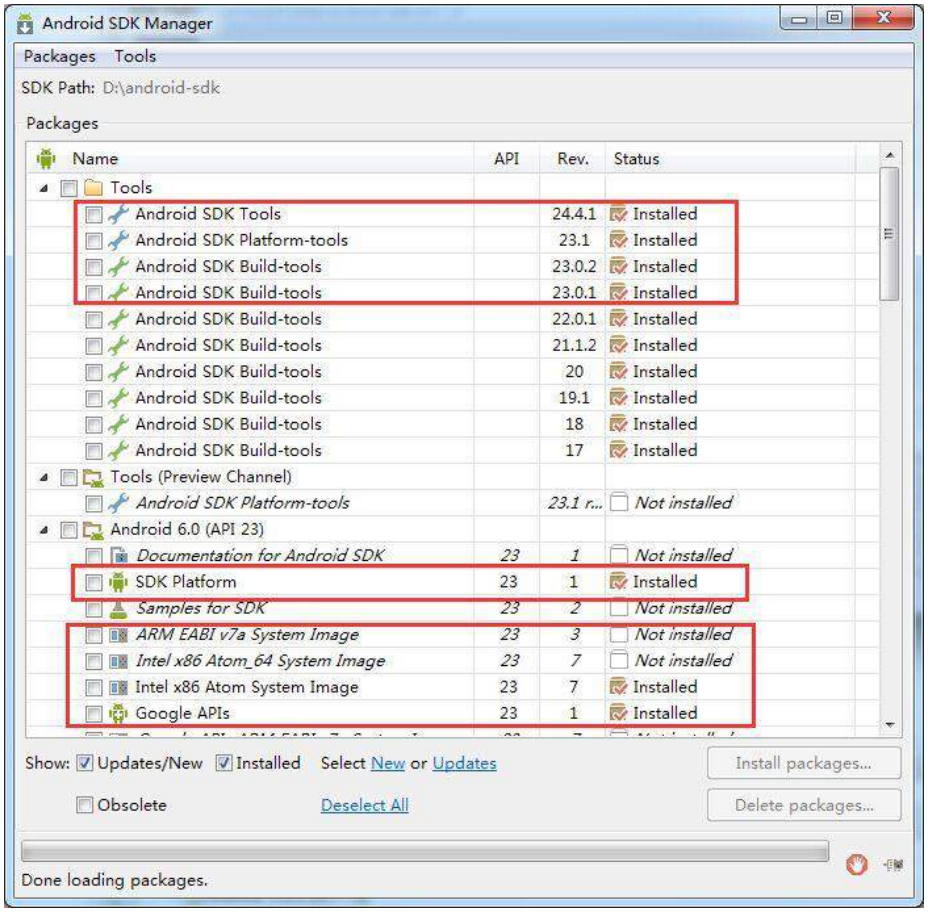
```
List<ShopInfo> list = new ArrayList<ShopInfo>();
list.add(shopInfo1);
list.add(shopInfo2);
Gson gson = new Gson();
String jsonArr = gson.toJson(list);
```

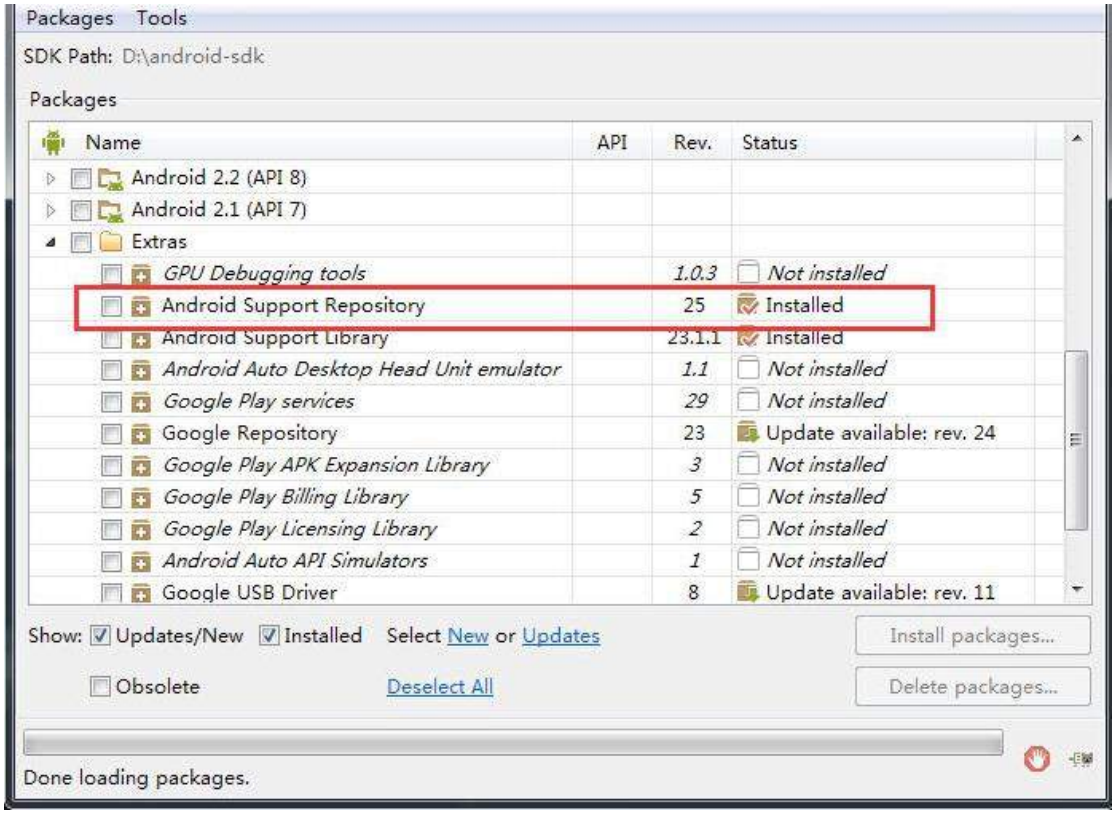
RecatNative 环境配置

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

一 先把 jdk 和 androidSdk 安装上去

（记着配置 ANDROID_HOME 否则安装的时候就会报错）





上面标注的这几项必须要安装上去否则也不行。Android SDK 的安装需要最新的安卓 6.0 的 SDK。下载被墙，可以使用腾讯 Bugly 的镜像来加速下载。

二 下载 nodeJs 并且安装

三 然后打开 CMD 输入以下命令 npm install -g react-native-cli 进行 reactNative 的安装

四 创建 react-native 项目

进入你要放置项目的目录然后执行命令 react-native init RNApp

安装好的目录

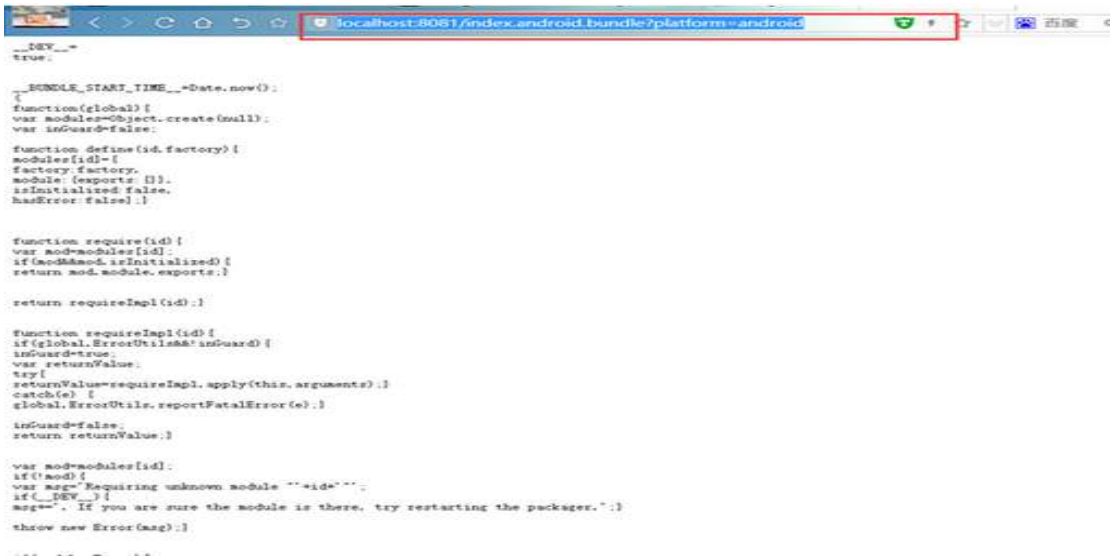
五 启动服务器

#对于 React Native 版本 < 0.14 的
cd MyAwesomeApp
node node_modules/react-native/packager/packager.js
#对于 React Native 版本 >= 0.14 的 (这个版本移除了 packager.js)
cd MyAwesomeApp
进入工作空间执行此命令 react-native start

如果你碰到了 ERROR Watcher took too long to load 的报错，请尝试将这个文件中的 timeout 值改得更大一些 (文件的具体路径是 node_modules/react-native/packager/react-packager/src/FileWatcher/index.js)。

运行这个会启动一个本地 8081 端口的 web 服务，可以使用

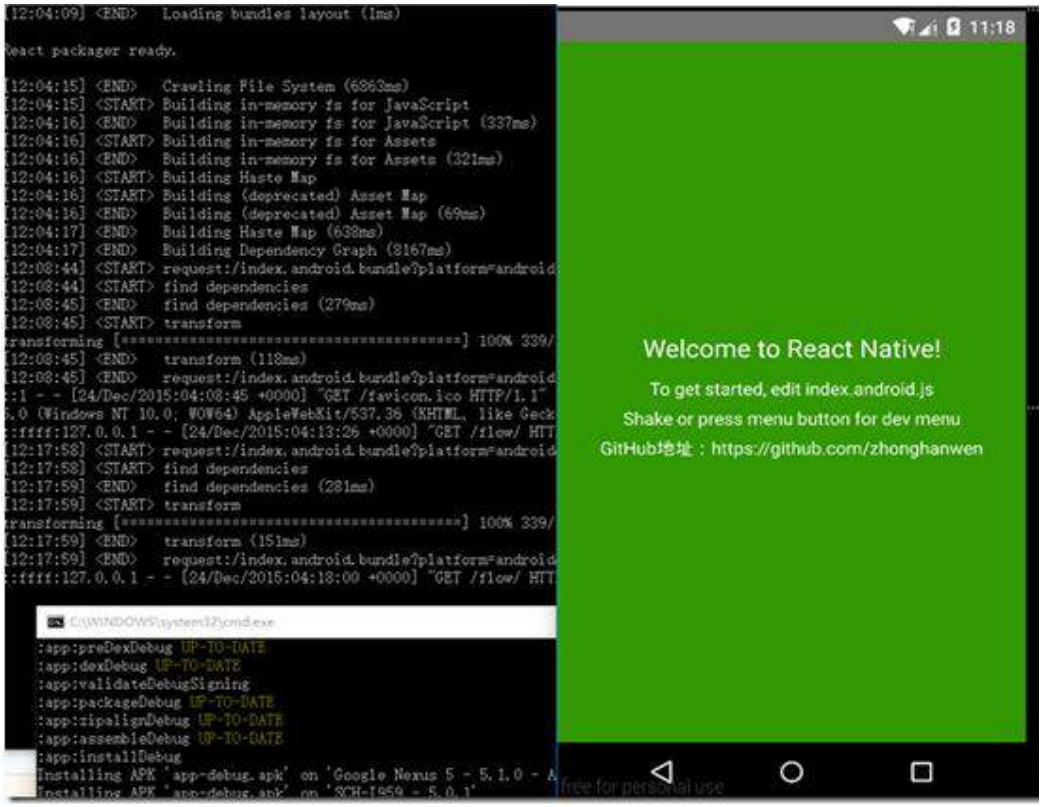
访问 http://localhost:8081/index.android.bundle?platform=android



如果能看到此内容说明服务器启动成功

六 执行 react-native run-android 命令

进行安装程序 第一次速度会慢点要去联网下载一些包这个时候一定要打开 vpn（连上外网）否则会报错



如果能弹出下图就说明安装成功了

（其中里面还是会遇到一些问题的 具体的英文记不太清楚了 翻译过来就是下载失败需要 vpn 或者 ANDROID_HOME 没有配置这样的一些信息，解决后再次执行此命令）

如果用的手机安装可能问题会多一些比如

对于 Android 5.0 以上设备，你可以运行 adb reverse tcp:8081 tcp:8081 来使得你的设备可以

访问到你的电脑。别的可以点击 menu 键 或者摇晃手机 弹出以下菜单

然后点击 devSetting 进行设置 你服务器的 IP：8081 然后返回再些点 menu 键选择 reload js 重新加载

（注意：一般用模拟器比较容易成功因为服务器和模拟器是一台电脑，如果用手机就要保持手机连接 wifi 和电脑是同一个网络 也就是说服务器并不是外网无法访问必须形成一个局域网才可以）

RecycleView

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

有了 ListView、GridView 为什么还需要 RecyclerView 这样的控件？

优点 ：RecyclerView 提供了一种插拔式的体验，高度的解耦，异常的灵活，通过设置它提供的不同 LayoutManager， ItemDecoration ，ItemAnimator 实现 ListView,GridView,瀑布流的效果。

* 你想要控制其显示的方式，请通过布局管理器 LayoutManager
* 你想要控制 Item 间的间隔（可绘制），请通过 ItemDecoration
* 你想要控制 Item 增删的动画，请通过 ItemAnimator

缺点 ：缺少 Item 点击事件，没有添加头部 View 基本使用

简单使用

一初始化控件
mRecyclerView = findView(R.id.id_recyclerview);
二设置布局管理器
mRecyclerView.setLayoutManager(layout);
三设置 adapter
mRecyclerView.setAdapter(adapter)

我们先谈谈布局管理器

RecyclerView.LayoutManager 吧，这是一个抽象类，系统提供了 3 个实现类：
LinearLayoutManager 现行管理器，支持横向、纵向。
GridLayoutManager 网格布局管理器
StaggeredGridLayoutManager 瀑布流式布局管理器

线性布局

rv.setLayoutManager(new LinearLayoutManager(this,LinearLayoutManager.HORIZONTAL,true));

第一个参数是上下文
第二个参数是我们的布局管理器横向展示还是纵向展示
第三个参数是 true 从末尾展示，相反从头展示

网格布局

rv.setLayoutManager(new GridLayoutManager(this,3,GridLayoutManager.HORIZONTAL,true));
第一个参数是上下文
第二个参数相对第三个参数而讲 如果是横向的代表三列 如果是竖直的代表三行
第三个参数我们的布局管理器横向展示还是纵向展示
第四个参数是 true 从末尾展示，相反从头展示

瀑布流管理器
new StaggeredGridLayoutManager(3,StaggeredGridLayoutManager.VERTICAL)
参数同上

注意：在使用瀑布流管理器的时候 需要给每个 item 随机数设置相应高度
我们可以在 viewholder 中对 imageview 动态设置高度

RecyclerView.Adapter 适配器

```
class MyAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>{
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
      return new MyHolder(view);
    }
    @Override
    public void onBindViewHolder(final RecyclerView.ViewHolder holder, int position)
    }
    @Override onBindViewHolder
    public int getItemCount() {
      return list.size();
    }
  }
```

我们先来看泛型

```
public static abstract class Adapter<VH extends ViewHolder> {
    private final AdapterDataObservable mObservable = new AdapterDataObservable();
    private boolean mHasStableIds = false;
```

点击源码可以看到要求我们必须继承 ViewHolder
注意：onCreateViewHolde 的返回类型 onBindViewHolder 的参数 和我们的泛型有直接关系

一我们先看 onCreateViewHolde 让我们返回一个 ViewHolder 那么我们就来看看 ViewHolder

```
class MyHolder extends RecyclerView.ViewHolder{
        public MyHolder(View itemView) {
           super(itemView);

        }
   }
```

这个 ViewHolder 其实就是我们 ListView 的 ViewHolder 的封装 只是我们现在一种类型返回一个 ViewHolder 就可以至于分类型我们下节再讲。
大家可以看到构造器有一个 View 参数 这个就是我们布局，也就是说初始化控件的操作要在我们的 Holder 中执行。

二 getItemCount 同 ListView 的 getItemCount 方法一样

三 onBindViewHolder 用来绑定相应的 viewHolder 比如 分类型的时候 根据不同的值调用不同的 Viewholder

附上代码

```
RecyclerView rv = findViewById(R.id.rv);
rv.setLayoutManager(new LinearLayoutManager(this));
rv.setAdapter(new MyAdapter());

class MyAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>{

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        return new MyHolder(View.inflate(context,R.layout.main_item,null));
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {

        ((MyHolder)holder).setData(list.get(position));
    }

    @Override
    public int getItemCount() {
        return list.size();
    }
}

class MyHolder extends RecyclerView.ViewHolder{

    private   TextView tv;
```

```
    public MyHolder(View itemView) {
        super(itemView);
        TextView tv    = itemView.findViewById(R.id.tv);
    }

    public void setData(String name){
        tv.setText(name);
    }
}
```

简介：在（MaterialDesign 之 RecycleView 的使用-1） 中我们讲解了 RecycleView 的基本使用
这一节我们来讲解
分类型 , item 单击事件

分类型

```
class MyAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>{

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent,
                                                      int viewType) {
        if (viewType==0){
            return new My2Holder(View.inflate(context,R.layout.head_item,null));
        }
        return new MyHolder(View.inflate(context,R.layout.main_item,null));
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {

        if (getItemViewType(position)==0){
```

```
((My2Holder)holder).setData(list.get(position));
        }else{

((MyHolder)holder).setData(list.get(position));
        }

    }

    @Override
    public int getItemCount() {
        return list.size();
    }

    @Override
    public int getItemViewType(int position) {
        if (position==0){
            return 0;
        }else{
            return 1;
        }
    }
}
```

我们先看最后一个方法 是不是和我们的 ListView 的分类型差不多啊
所以我们分类型主要分三步
第一步 getItemViewType() 根据 position 返回不同的类型值（int 值可以是任何数，不用像 listview 那样必须从 0 开始）
第二步 onCreateViewHolder 大家看第二个参数 viewType 就是我们 getItemViewType()返回的类型值，然后根据我们不同的类型值创建我们不同的 ViewHolder

第三步 onBindViewHolder() 从名字上就可以看出就是用来绑定我们的 Holder 通过判断 getItemViewType()的值往相应的 viewholder 传值

Item 点击事件

其实主要原理就是利用我们的回调方法
第一步首先写一个接口

```
public interface OnItemClickListener {

    void onClick(View v,int position);
}
```

第二步 在我们的 Adapter 把我们的接口定为变量，并且定义 set 方法

```
class MyAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>{

    private OnItemClickListener onItemClickListener;

    public void setOnItemClickListener(OnItemClickListener onItemClickListener) {
        this.onItemClickListener = onItemClickListener;
    }
```

第三步 在我们的 onBindViewHolder()通过 holder.itemView 的点击事件来回调我们的方法

```
@Override
public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {

    if (onItemClickListener!=null){
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                onItemClickListener.onClick(v,holder.getLayoutPosition());

            }
        });
    }

}
```

第四步就是调用我们的 adapter 里的 set 方法了

```
adapter.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onClick(View v, int position) {

    }
});
```

Retrofit

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

概述

Retrofit 是一个类型安全的 REST 客户端，Square 提供的开源产品。它可以直接解析 JSON 数据变成 JAVA 对象，甚至支持回调操作，处理不同的结果。
官方地址：http://square.github.io/retrofit/
在官方地址中可以下载 jar 包，也可以在 AndroidStudio 中引入。

使用

1.首先我们要定义一个接口

```
public interface GitHubService {
  @GET("users/list?sort=desc")
  Call<List<User>> listUser (@Path("user") String user);
}
```

下面我来讲解一下这个接口
1.@GET 就是我们的请求方式。主要有 GET,POST,PUT,DELETE,and HEAD
Call 是关键字，
User 是我们的 been 对象，
listUser 是方法名
2．然后看我们括号内的就是我们访问的 URL 地址。再看一下个

```
@GET("group/{id}/users")
Call<List<User>> listUser (@Path("id") int groupId);
```

大括号包含的 id 可以理解为一个变量 = @Path("id") int groupId
等号后面的意思就是 把 groupId 这个参数赋值给我们的 id

```
@GET("group/{id}/users")
Call<List<User>> listUser (@Path("id") int groupId,
@Query("sort") String sort);
```

第一个参数同上，第二个参数的意思就是我们传了一个值 key 是 sort value 是传参过来的 sort

```
@GET("group/{id}/users")
Call<List<User>> listUser (@Path("id") int groupId, @QueryMap
Map<String, String> options);
```

第一个参数同上，第二个参数就好理解了 URL 要带好多参数 每一个参数的 KEY 和 VALUE 都在我们的 Map 里

```
@Headers("Cache-Control: max-age=640000")
@GET("widget/list")
Call<List<Widget>> widgetList();

//添加我们的 header
@Headers({
    "Accept: application/vnd.github.v3.full+json",
    "User-Agent: Retrofit-Sample-App"
})
@GET("users/{username}")
Call<User> getUser(@Path("username") String username);
```

是不是很强大呢 当然还有很多了比如上传文件什么的这里只讲最基本的用法其它的需要大家自己查文档了。

第三步 创建一个 Retrofit 对象：

```
Retrofit retrofit = new Retrofit.Builder()
     //这个 URL 就是和我们接口的 URL 拼接在一起的
    .baseUrl("https://api.github.com")
    .addConverterFactory(GsonConverterFactory.create())
    .build();

//再用这个 Retrofit 对象创建一个 GitHubService 对象：
//GitHubService.class 是我们的接口名字
GitHubService service = retrofit.create(GitHubService.class);
//拿到接口对象然后调用接口里的方法
Call<List<User>> call = service.listRepos("octocat");

//最后你就可以用这个 Githb 对象获得数据了：
call.enqueue(new Callback<List<User>>() {
    @Override
    public void onResponse(Call<List<User>> call,
          Response<List<User>> response) {

    }
    @Override
    public void onFailure(Call<List<User>> call,
```

```
                Throwable throwable) { }
});
```

注意：

URL 的定义方式

```
public interface GitHubService {
    @POST("user")
    Call<User> login();
}
public void getUrl() {
    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http:// atguigu.com /base/home")
            .addConverterFactory(GsonConverterFactory.create())
            .build();
}
//最后的 url 是 http:// atguigu.com /base/user
public interface GitHubService {
    @POST("user")
    Call<User> login();
}
public void getUrl () {
    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://atguigu.com/base/home/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();
}
//最后的 url 是 http://atguigu.com/base/home/user
public interface GitHubService {
    @POST("/user")
    Call<User> login();
}
public void getUrl() {
    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://atguigu.com/base/home/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();
}
//最后的 url 是 http://atguigu.com/user
```

RxJava 的基本使用

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

概述

RxJava 就是一个实现异步操作的库。它可以代替我们的 Handler,AsynTask。

简介

RxJava 最核心的两个东西是 Observables（被观察者，事件源）和 Subscribers（观察者）。

Observables 发出一系列事件，Subscribers 处理这些事件。

一个 Observable 可以发出零个或者多个事件，知道结束或者出错。每发出一个事件，就会调用它的 Subscriber 的 onNext 方法，最后调用 Subscriber.onNext()或者 Subscriber.onError()结束。

注意：Rxjava 的看起来很想设计模式中的观察者模式，但是有一点明显不同，那就是如果一个 Observerble 没有任何的的 Subscriber，那么这个 Observable 是不会发出任何事件的。

HelloWord

首先关联包

```
compile 'io.reactivex:rxandroid:1.2.1'
compile 'io.reactivex:rxjava:1.1.6'
```

```
//定义一个被观察者
Observable<String> myObservable = Observable.create(
    new Observable.OnSubscribe<String>() {
        @Override
        public void call(Subscriber<? super String> sub) {
```

```
            //向观察者发出通知
            sub.onNext("Hello, world!");
            sub.onCompleted();
        }
    }
);
```

这里定义的 Observable 对象仅仅发出一个 Hello World 字符串，然后就结束了。接着我们创建一个 Subscriber 来处理 Observable 对象发出的字符串。

```
//观察者有三个方法
Subscriber<String> mySubscriber = new Subscriber<String>() {
    @Override
    public void onNext(String s) { System.out.println(s); }

    @Override
     public void onCompleted() { }

    @Override
    public void onError(Throwable e) { }
};

//让观察者和被观察者产生关系
 myObservable.subscribe(mySubscriber);
```

//订阅后 myObservable 就是调用 mySubscriber 对象的 onNext 和 onComplete 方法，mySubscriber 就会打印出 Hello World！

HelloWord 简化

```
//RxJava 内置了很多简化创建 Observable 对象的函数，比如 Observable.just 创建只发出一
//个事件就结束的 Observable 对象，可以简化为一行

    Observable<String> myObservable = Observable.just("Hello, world!");

//接下来看看如何简化 Subscriber，上面的例子中，我们其实并不关心 OnComplete 和 OnError，
//我们只需要在 onNext 的时候做一些处理，这时候就可以使用 Action1 类。

    Action1<String> onNextAction = new Action1<String>() {
```

```
    @Override
    public void call(String s) {
        System.out.println(s);
    }
};
```

subscribe 方法有一个重载版本，接受三个 Action1 类型的参数，分别对应 OnNext，OnComplete， OnError 函数。

```
//现在的
myObservable.subscribe(onNextAction);

    //可以再整体简化
    Observable.just("Hello, world!")
    .subscribe(new Action1<String>() {
        @Override
        public void call(String s) {
            System.out.println(s);
        }
    });
```

变换

所谓的变化是指在输出结果之前我们可以利用一些方法对结果做出一些更改

map 方法

```
Observable.just("aaaaaaa") // 输入类型 String
  .map(new Func1<String, String >() {
    @Override
    public String call(String s) { // 参数类型 String
      return s+”bbbbbbbbbbb”; // 返回类型 Bitmap
    }
  })
  .subscribe(new Action1< String >() {
    @Override
```

```
    public void call(String s) { // 参数类型 Bitmap
      Log.i("atguigu.com",s);
    }
  });
```

这里出现了一个叫做 Func1 的类。它和 Action1 非常相似，也是 RxJava 的一个接口，用于包装含有一个参数的方法。
Func1 和 Action 的区别在于， Func1 包装的是有返回值的方法。可以看到，map 方参数中的 String 对象的值在经过 map() 方法后，事件的值从 aaaaaaa 变成了 aaaaaaa bbbbbbbbbbb。还可以更改返回类型呢。比如输入的是一个 url 返回的是一个图片这使得 RxJava 变得非常灵活。

from 方法

```
//Observable.from() 方法，它接收一个集合作为输入，然后每次输出一个元素给 subscriber

Observable.from(new String[]{"hello 5",
    "hello 6"}).subscribe(new Action1<String>() {
      @Override
      public void call(String t) {
        Log.d(TAG, t);//会打印两次依次打印 5， 6
      }
    });
```

flatMap

Observable.flatMap()主要是用来处理一对多的转换 接收一个 Observable 的输出作为输入，同时输出另外一个 Observable
先看这个简单的 通过 flatMap 将集合里的数据转成单个数据

```
List<String> list = new ArrayList<String>();
    list.add("hello 7");
    list.add("hello 8");
```

```
Observable.just(list)
.flatMap(new Func1<List<String>, Observable<String>>() {
    @Override
    public Observable<String> call(List<String> t) {
        return Observable.from(t);
    }

})
.subscribe(new Action1<String>() {

    @Override
    public void call(String t) {
        Log.d(TAG, t); // 依次打印 7,8
    }
});
```

那怎么才能把一个 Student 转化成多个 Course 呢？这个时候，就需要用 flatMap() 了：

```
Student[] students = ...;
Subscriber<Course> subscriber = new Subscriber<Course>() {
    @Override
    public void onNext(Course course) {
        Log.d(tag, course.getName());
    }
    ...
};
//先从 Student 数组里取出每一个学生的对象，然后再把每个学生里的数据单独返回
Observable.from(students)
    .flatMap(new Func1<Student, Observable<Course>>() {
        @Override
        public Observable<Course> call(Student student) {
            return Observable.from(student.getCourses());
        }
    })
     .subscribe(subscriber);
```

flatMap() 和 map() 有一个相同点：它也是把传入的参数转化之后返回另一个对象。但需要注意不同的是， flatMap() 中返回的是个 Observable 对象，并且这个 Observable 对象并不是被直接发送到了 Subscriber 的回调方法中。

过滤

```
.filter(new Func1<String, Boolean>() {

    @Override
    public Boolean call(String t) {
        return t != null;
    }
})
```

take()

take()输出最多指定数量的

doOnNext()

```
//doOnNext()允许我们在每次输出一个元素之前做一些额外的事情
List<String> list = new ArrayList<String>();
    list.add("hello 1");
    list.add("hello 2");
    list.add(null);
    Observable.just(list)
    .flatMap(new Func1<List<String>, Observable<String>>() {

        @Override
        public Observable<String> call(List<String> t) {
            return Observable.from(t);
        }
```

```
})
.filter(new Func1<String, Boolean>() {

    @Override
    public Boolean call(String t) {
        return t != null;
    }
})
.take(2)
.doOnNext(new Action1<String>() {

    @Override
    public void call(String t) {
        Log.e(TAG, t);
    }
})
.subscribe(new Action1<String>() {

    @Override
    public void call(String t) {
        Log.d(TAG, t);
    }
});
```

线程切换

RxJava 在哪个线程调用 subscribe()，就在哪个线程生产事件；在哪个线程生产事件，就在哪个线程消费事件。如果需要切换线程，就需要用到 Scheduler

Schedulers.immediate(): 直接在当前线程运行。这是默认的。

Schedulers.newThread(): 启用新线程，并在新线程执行操作。

Schedulers.io(): I/O 操作（读写文件、读写数据库、网络信息交互等）所使用的 Scheduler。内部实现是是用一个无数量上限的线程池，可以重用空闲的线程，因此多数情况下 io() 比 newThread() 更有效率。不要把计算工作放在 io() 中，可以避免创建不必要的线程。

Schedulers.computation(): 计算所使用的 Scheduler。例如图形的计算。这个 Scheduler 使用的固定的线程池，大小为 CPU 核数。不要把 I/O 操作放在 computation() 中，否则 I/O 操作的等待时间会浪费 CPU。

AndroidSchedulers.mainThread()，Android 专用的 它指定的操作将在 Android 主线程运行。

可以使用 subscribeOn() 和 observeOn() 两个方法来对线程进行控制了。 * subscribeOn(): 指定 subscribe() 所发生的线程，

observeOn(): 指定 Subscriber 所运行在的线程。

```
        //会分别执行里面的参数
        Observable.just(1, 2, 3, 4)
        // 指定 subscribe() 发生在 IO 线程
        .subscribeOn(Schedulers.io())
        // 指定 Subscriber 的回调发生在主线程
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new Action1<Integer>() {
            @Override
            public void call(Integer number) {
                Log.d(tag, "" + number);
        });
```

上两句 subscribeOn(Scheduler.io()) 和 observeOn(AndroidSchedulers.mainThread()) 的使用方式非常常见。

```
int drawableRes = R.drawable.aaa;
ImageView imageView  = new ImageView(this);
Observable.create(new OnSubscribe<Drawable>() {
    @Override
    public void call(Subscriber<? super Drawable> subscriber) {
        Drawable drawable = getTheme().getDrawable(drawableRes));
        subscriber.onNext(drawable);
        subscriber.onCompleted();
    }
})
.subscribeOn(Schedulers.io()) // 指定 subscribe() 发生在 IO 线程
.observeOn(AndroidSchedulers.mainThread()) // 指定 Subscriber 的回调发生在主线程
.subscribe(new Observer<Drawable>() {
    @Override
    public void onNext(Drawable drawable) {
        imageView.setImageDrawable(drawable);
    }

    @Override
    public void onCompleted() {
```

```
    }

    @Override
    public void onError(Throwable e) {
        Toast.makeText(activity, "Error!", Toast.LENGTH_SHORT).show();
    }
});
```

MaterialDesign 之 ToolBar 的使用

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

ToolBar 的使用

1.首先要隐藏原来的 ActionBar

在当前主题里加上这两句话

item name=windowActionBarfalseitem

item name=androidwindowNoTitletrueitem

在布局里加入下面的代码

```
<android.support.v7.widget.Toolbar
    androidid=@+idtoolbar
    androidlayout_height=attractionBarSize
    androidlayout_width=match_parent
android.support.v7.widget.Toolbar>
```

请记得用 support v7 里的 toolbar，不然然只有 API Level 21 也就是 Android 5.0 以上的版本才能使用。

预设常用的几个元素就如图中所示

setNavigationIcon 即设定 up button 的图标

setLogo APP 的图标。

setTitle 主标题。

setSubtitle 副标题。

setOnMenuItemClickListener 设定菜单各按鈕的动作。

注意 ：setNavigationIcon 需要放在 setSupportActionBar 之后才会生效。

菜单部分，需要先在 resmenumenu_main.xml 左定义：

```
<menu xmlnsandroid=httpschemas.android.comapkresandroid
    xmlnsapp=httpschemas.android.comapkres-auto
    xmlnstools=httpschemas.android.comtools
    toolscontext=.MainActivity

  <item androidid=@+idaction_edit
```

```
                androidtitle=@stringaction_edit
                androidorderInCategory=80
                androidicon=@drawableab_edit
                appshowAsAction=ifRoom
menu>
```

Toolbar 和 CollapsingToolbarLayout 结合的使用

```
android.support.design.widget.AppBarLayout
    androidlayout_width=match_parent
    androidlayout_height=256dp
    androidfitsSystemWindows=true
    android.support.design.widget.CollapsingToolbarLayout
        androidid=@+idcollapsing_toolbar_layout
        androidlayout_width=match_parent
        androidlayout_height=match_parent
        appcontentScrim=#30469b
        appexpandedTitleMarginStart=48dp
        applayout_scrollFlags=scrollexitUntilCollapsed
        ImageView
            androidlayout_width=match_parent
            androidlayout_height=match_parent
            androidscaleType=centerCrop
            androidsrc=@mipmapbg
            applayout_collapseMode=parallax
            applayout_collapseParallaxMultiplier=0.7
        android.support.v7.widget.Toolbar
            androidid=@+idtoolbar
            androidlayout_width=match_parent
            androidlayout_height=attractionBarSize
            applayout_collapseMode=pin
    android.support.design.widget.CollapsingToolbarLayout
android.support.design.widget.AppBarLayout
```

我们在 CollapsingToolbarLayout 中设置了一个 ImageView 和一个 Toolbar。并把这个 CollapsingToolbarLayout 放到 AppBarLayout 中作为一个整体。

1、在 CollapsingToolbarLayout 中：

我们设置了 layout_scrollFlags 关于它的值我这里再说一下：
scroll - 想滚动就必须设置这个。

enterAlways - 实现 quick return 效果，当向下移动时，立即显示 View（比如 Toolbar)。
exitUntilCollapsed - 向上滚动时收缩 View，但可以固定 Toolbar 一直在上面。
enterAlwaysCollapsed - 当你的 View 已经设置 minHeight 属性又使用此标志时，你的 View 只能以最小高度进入，只有当滚动视图到达顶部时才扩大到完整高度。

其中还设置了一些属性，简要说明一下：

contentScrim - 设置当完全 CollapsingToolbarLayout 折叠(收缩)后的背景颜色。
expandedTitleMarginStart - 设置扩张时候(还没有收缩时)title 向左填充的距离。

2、在 ImageView 控件中：

我们设置了：
layout_collapseMode (折叠模式) - 有两个值

pin - 设置为这个模式时，当 CollapsingToolbarLayout 完全收缩后，Toolbar 还可以保留在屏幕上。
parallax - 设置为这个模式时，在内容滚动时，CollapsingToolbarLayout 中的 View（比如 ImageView)也可以同时滚动，实现视差滚动效果，通常和 layout_collapseParallaxMultiplier(设置视差因子)搭配使用。
layout_collapseParallaxMultiplier(视差因子) - 设置视差滚动因子，值为：0~1。

3、在 Toolbar 控件中：

我们设置了 layout_collapseMode(折叠模式)：为 pin。

综上分析：当设置了 layout_behavior 的控件响应起了 CollapsingToolbarLayout 中的 layout_scrollFlags 事件时，ImageView 会有视差效果的向上滚动移除屏幕，当开始折叠时 CollapsingToolbarLayout 的背景色(也就是 Toolbar 的背景色)就会变为我们设置好的背景色，Toolbar 也一直会固定在最顶端。