题目：PL-SQL 经典试题

0. 准备工作:

```
set serveroutput on
```

hellowrold 程序

```
begin
dbms_output.put_line('hello world');
end;
/
```

[语法格式]

```
--declare
   --声明的变量、类型、游标
begin
   --程序的执行部分（类似于 java 里的 main()方法）
   dbms_output.put_line('helloworld');
--exception
   --针对 begin 块中出现的异常，提供处理的机制
   --when .... then ...
   --when    .... then ...
end;
```

****

基本语法

****

1. 使用一个变量

```
declare
   --声明一个变量
   v_name varchar2(25);
begin
   --通过 select ... into ... 语句为变量赋值
  select last_name into v_name
  from employees
  where employee_id = 186;

  -- 打印变量的值
  dbms_output.put_line(v_name);
end;
```

2. 使用多个变量

```
declare
   --声明变量
   v_name varchar2(25);
   v_email varchar2(25);
   v_salary number(8, 2);
   v_job_id varchar2(10);
begin
   --通过  select ... into ...  语句为变量赋值
   --被赋值的变量与 SELECT 中的列名要一一对应
  select last_name, email, salary, job_id into v_name, v_email, v_salary, v_job_id
  from employees
  where employee_id = 186;

  --  打印变量的值
  dbms_output.put_line(v_name || ', ' || v_email || ', ' ||    v_salary || ', ' ||    v_job_id);
end;
```

记录类型

3.1 自定义记录类型

```
declare
   --定义一个记录类型
   type customer_type is record(
      v_cust_name varchar2(20),
      v_cust_id number(10));

   --声明自定义记录类型的变量
   v_customer_type customer_type;
begin
   v_customer_type.v_cust_name := '刘德华';
   v_customer_type.v_cust_id := 1001;

   dbms_output.put_line(v_customer_type.v_cust_name||','||v_customer_type.v_cust_id);
end;
```

3.2 自定义记录类型

```
declare
```

```
  --定义一个记录类型
  type emp_record is record(
     v_name varchar2(25),
     v_email varchar2(25),
     v_salary number(8, 2),
     v_job_id varchar2(10));

  --声明自定义记录类型的变量
  v_emp_record emp_record;
begin
  --通过 select ... into ... 语句为变量赋值
 select last_name, email, salary, job_id into v_emp_record
 from employees
 where employee_id = 186;

 -- 打印变量的值
 dbms_output.put_line(v_emp_record.v_name || ', ' || v_emp_record.v_email || ', ' ||
v_emp_record.v_salary || ', ' ||  v_emp_record.v_job_id);
end;
```

4. 使用 %type 定义变量，动态的获取数据的声明类型

```
declare
  --定义一个记录类型
  type emp_record is record(
     v_name employees.last_name%type,
     v_email employees.email%type,
     v_salary employees.salary%type,
     v_job_id employees.job_id%type);

  --声明自定义记录类型的变量
  v_emp_record emp_record;
begin
  --通过 select ... into ... 语句为变量赋值
 select last_name, email, salary, job_id into v_emp_record
 from employees
 where employee_id = 186;

 -- 打印变量的值
 dbms_output.put_line(v_emp_record.v_name || ', ' || v_emp_record.v_email || ', ' ||
v_emp_record.v_salary || ', ' ||  v_emp_record.v_job_id);
end;
```

5. 使用 %rowtype

```
declare
--声明一个记录类型的变量
   v_emp_record employees%rowtype;
begin
   --通过 select ... into ... 语句为变量赋值
  select * into v_emp_record
  from employees
  where employee_id = 186;

  -- 打印变量的值
  dbms_output.put_line(v_emp_record.last_name || ', ' || v_emp_record.email || ', ' ||
v_emp_record.salary || ', ' ||   v_emp_record.job_id   || ', ' || v_emp_record.hire_date);
end;
```

6.1 赋值语句：通过变量实现查询语句

```
declare
   v_emp_record employees%rowtype;
   v_employee_id employees.employee_id%type;
begin
   --使用赋值符号位变量进行赋值
   v_employee_id := 186;

   --通过 select ... into ... 语句为变量赋值
  select * into v_emp_record
  from employees
  where employee_id = v_employee_id;

  -- 打印变量的值
  dbms_output.put_line(v_emp_record.last_name || ', ' || v_emp_record.email || ', ' ||
v_emp_record.salary || ', ' ||   v_emp_record.job_id   || ', ' || v_emp_record.hire_date);
end;
```

6.2 通过变量实现 DELETE、INSERT、UPDATE 等操作

```
declare
   v_emp_id employees.employee_id%type;
```

```
begin
   v_emp_id := 109;
   delete from employees
   where employee_id = v_emp_id;
   --commit;
end;
```

****

流程控制

****

条件判断

---

7. 使用 IF ... THEN ... ELSIF ... THEN ...ELSE ... END IF;

要求：查询出 150 号 员工的工资，若其工资大于或等于 10000 则打印 'salary >= 10000'; 若在 5000 到 10000 之间，则打印 '5000<= salary < 10000'; 否则打印 'salary < 5000'

(方法一)

```
declare
   v_salary employees.salary%type;
begin
   --通过 select ... into ... 语句为变量赋值
  select salary into v_salary
  from employees
  where employee_id = 150;

  dbms_output.put_line('salary: ' || v_salary);

  -- 打印变量的值
  if v_salary >= 10000 then
       dbms_output.put_line('salary >= 10000');
  elsif v_salary >= 5000 then
       dbms_output.put_line('5000 <= salary < 10000');
  else
       dbms_output.put_line('salary < 5000');
  end if;
```

(方法二)

```
declare
        v_emp_name employees.last_name%type;
        v_emp_sal employees.salary%type;
        v_emp_sal_level varchar2(20);
begin
        select last_name,salary into v_emp_name,v_emp_sal from employees where employee_id
```

```
= 150;

        if(v_emp_sal >= 10000) then v_emp_sal_level := 'salary >= 10000';
        elsif(v_emp_sal >= 5000) then v_emp_sal_level := '5000<= salary < 10000';
        else v_emp_sal_level := 'salary < 5000';
        end if;

        dbms_output.put_line(v_emp_name||','||v_emp_sal||','||v_emp_sal);
end;
```

7+ 使用 CASE ... WHEN ... THEN ...ELSE ... END 完成上面的任务

```
declare
        v_sal employees.salary%type;
        v_msg varchar2(50);
begin
        select salary into v_sal
        from employees
        where employee_id = 150;

        --case 不能向下面这样用
        /*
        case v_sal when salary >= 10000 then v_msg := '>=10000'
                        when salary >= 5000 then v_msg := '5000<= salary < 10000'
                        else v_msg := 'salary < 5000'
        end;
        */

        v_msg :=
                case trunc(v_sal / 5000)
                        when 0 then 'salary < 5000'
                        when 1 then '5000<= salary < 10000'
                        else 'salary >= 10000'
                end;

        dbms_output.put_line(v_sal ||','||v_msg);
end;
```

8. 使用 CASE ... WHEN ... THEN ... ELSE ... END;

要求: 查询出 122 号员工的 JOB_ID, 若其值为 'IT_PROG', 则打印 'GRADE: A';
'AC_MGT', 打印 'GRADE B',
'AC_ACCOUNT', 打印 'GRADE C';

否则打印 'GRADE D'

```
declare
        --声明变量
        v_grade char(1);
        v_job_id employees.job_id%type;
begin
        select job_id into v_job_id
        from employees
        where employee_id = 122;

        dbms_output.put_line('job_id: ' || v_job_id);

        --根据 v_job_id 的取值, 利用 case 字句为 v_grade 赋值
        v_grade :=
                case v_job_id when 'IT_PROG' then 'A'
                                when 'AC_MGT' then 'B'
                                when 'AC_ACCOUNT' then 'C'
                                else 'D'
                 end;

        dbms_output.put_line('GRADE: ' || v_grade);
end;
```

循环结构

---

9. 使用循环语句打印 1 - 100.（三种方式）

1). LOOP ... EXIT WHEN ... END LOOP

```
declare
        --初始化条件
        v_i number(3) := 1;
begin
        loop
        --循环体
          dbms_output.put_line(v_i);
                                                        --循环条件
          exit when v_i = 100;
                                                        --迭代条件
          v_i := v_i + 1;
        end loop;
end;
```

2). WHILE ... LOOP ... END LOOP

```
declare
        --初始化条件
        v_i number(3) := 1;
begin
        --循环条件
        while v_i <= 100 loop
                                                        --循环体
                dbms_output.put_line(v_i);
                                                        --迭代条件
                v_i := v_i + 1;
        end loop;
end;
```

3).

```
begin
        for i in 1 .. 100 loop
                dbms_output.put_line(i);
        end loop;
end;
```

10. 综合使用 if, while 语句, 打印 1 - 100 之间的所有素数
(素数: 有且仅用两个正约数的整数, 2, 3, 5, 7, 11, 13, ...).

```
declare
   v_flag number(1):=1;
   v_i number(3):=2;
   v_j number(2):=2;
begin

   while (v_i<=100) loop
            while v_j <= sqrt(v_i) loop
                    if (mod(v_i,v_j)=0) then v_flag:= 0;end if;
                    v_j :=v_j +1;
            end loop;
            if(v_flag=1) then dbms_output.put_line(v_i);end if;

            v_flag :=1;
            v_j := 2;
            v_i :=v_i +1;
     end loop;
```

```
end;
```

(法二)使用 for 循环实现 1-100 之间的素数的输出

```
declare
   --标记值，若为 1 则是素数，否则不是
   v_flag number(1) := 0;
begin
    for i in 2 .. 100 loop

          v_flag := 1;

          for j in 2 .. sqrt(i) loop
               if i mod j = 0 then
                    v_flag := 0;
               end if;
          end loop;

          if v_flag = 1 then
               dbms_output.put_line(i);
          end if;

    end loop;
end;
```

11. 使用 goto

```
declare
   --标记值，若为 1 则是素数，否则不是
   v_flag number(1) := 0;
begin
    for i in 2 .. 100 loop
          v_flag := 1;

          for j in 2 .. sqrt(i) loop
               if i mod j = 0 then
                    v_flag := 0;
                    goto label;
               end if;
          end loop;

          <<label>>
          if v_flag = 1 then
```

```
            dbms_output.put_line(i);
        end if;

    end loop;
end;
```

11+.打印 1——100 的自然数，当打印到 50 时，跳出循环，输出“打印结束”

(方法一)

```
begin
  for i in 1..100 loop
        dbms_output.put_line(i);
        if(i = 50) then
        goto label;
        end if;
  end loop;

        <<label>>
        dbms_output.put_line('打印结束');

end;
```

(方法二)

```
begin
  for i in 1..100 loop
        dbms_output.put_line(i);
        if(i mod 50 = 0) then dbms_output.put_line('打印结束');
        exit;
        end if;
  end loop;
end;
```

****

游标的使用

****

12.1 使用游标

要求: 打印出 80 部门的所有的员工的工资:salary: xxx

```
declare
  --1. 定义游标
  cursor salary_cursor is select salary from employees where department_id = 80;
  v_salary employees.salary%type;
begin
 --2. 打开游标
 open salary_cursor;
```

```
 --3. 提取游标
 fetch salary_cursor into v_salary;

 --4. 对游标进行循环操作: 判断游标中是否有下一条记录
while salary_cursor%found loop
        dbms_output.put_line('salary: ' || v_salary);
        fetch salary_cursor into v_salary;
end loop;

 --5. 关闭游标
 close   salary_cursor;
end;
```

12.2 使用游标

要求: 打印出 80 部门的所有的员工的工资: Xxx 's salary is: xxx

```
declare
   cursor sal_cursor is select salary ,last_name from employees where department_id = 80;
   v_sal number(10);
   v_name varchar2(20);
begin
   open sal_cursor;

   fetch sal_cursor into v_sal,v_name;

   while sal_cursor%found loop
           dbms_output.put_line(v_name||'`s salary is '||v_sal);
           fetch sal_cursor into v_sal,v_name;
   end loop;

   close sal_cursor;

end;
```

13. 使用游标的练习:

打印出 manager_id 为 100 的员工的 last_name, email, salary 信息(使用游标, 记录类型)

```
declare
            --声明游标
            cursor emp_cursor is select last_name, email, salary from employees where
manager_id = 100;
```

```
		--声明记录类型
		type emp_record is record(
			name employees.last_name%type,
			email employees.email%type,
			salary employees.salary%type
		);

		-- 声明记录类型的变量
		v_emp_record emp_record;
begin
		--打开游标
		open emp_cursor;

		--提取游标
		fetch emp_cursor into v_emp_record;

		--对游标进行循环操作
		while emp_cursor%found loop
			dbms_output.put_line(v_emp_record.name || ', ' || v_emp_record.email
|| ', ' || v_emp_record.salary );
			fetch emp_cursor into v_emp_record;
		end loop;

		--关闭游标
		close emp_cursor;
end;
(法二：使用 for 循环)
declare

	cursor emp_cursor is
	select last_name,email,salary
	from employees
	where manager_id = 100;

begin


	for v_emp_record in emp_cursor loop

dbms_output.put_line(v_emp_record.last_name||','||v_emp_record.email||','||v_emp_record.
salary);
```

```
        end loop;
end;
```

14. 利用游标，调整公司中员工的工资：

工资范围	调整基数
0 - 5000	5%
5000 - 10000	3%
10000 - 15000	2%
15000 -	1%

```
declare
    --定义游标
    cursor emp_sal_cursor is select salary, employee_id from employees;

    --定义基数变量
    temp number(4, 2);

    --定义存放游标值的变量
    v_sal employees.salary%type;
    v_id employees.employee_id%type;
begin
    --打开游标
    open emp_sal_cursor;

    --提取游标
    fetch emp_sal_cursor into v_sal, v_id;

    --处理游标的循环操作
    while emp_sal_cursor%found loop
        --判断员工的工资，执行 update 操作
        --dbms_output.put_line(v_id || ': ' || v_sal);

        if v_sal <= 5000 then
            temp := 0.05;
        elsif v_sal<= 10000 then
            temp := 0.03;
        elsif v_sal <= 15000 then
            temp := 0.02;
        else
            temp := 0.01;
        end if;
```

```
            --dbms_output.put_line(v_id || ': ' || v_sal || ', ' || temp);
            update employees set salary = salary * (1 + temp) where employee_id = v_id;

            fetch emp_sal_cursor into v_sal, v_id;
    end loop;
    --关闭游标
    close emp_sal_cursor;
end;
```

使用 SQL 中的 decode 函数

```
update employees set salary = salary * (1 + (decode(trunc(salary/5000), 0, 0.05,
                                                                        1, 0.03,
                                                                        2, 0.02,
                                                                        0.01)))
```

15. 利用游标 for 循环完成 14.

```
declare
    --定义游标
    cursor emp_sal_cursor is select salary, employee_id id from employees;

    --定义基数变量
    temp number(4, 2);
begin
    --处理游标的循环操作
    for c in emp_sal_cursor loop
            --判断员工的工资，执行 update 操作
            --dbms_output.put_line(c.employee_id || ': ' || c.salary);

            if c.salary <= 5000 then
                temp := 0.05;
            elsif c.salary <= 10000 then
                temp := 0.03;
            elsif c.salary <= 15000 then
                temp := 0.02;
            else
                temp := 0.01;
            end if;

            --dbms_output.put_line(v_id || ': ' || v_sal || ', ' || temp);
```

```
            update employees set salary = salary * (1 + temp) where employee_id = c.id;
    end loop;
end;
```

16*. 带参数的游标

```
declare
    --定义游标
    cursor emp_sal_cursor(dept_id number, sal number) is
            select salary + 1000 sal, employee_id id
            from employees
            where department_id = dept_id and salary > sal;

    --定义基数变量
    temp number(4, 2);
begin
    --处理游标的循环操作
    for c in emp_sal_cursor(sal => 4000, dept_id => 80) loop
            --判断员工的工资，执行 update 操作
            --dbms_output.put_line(c.id || ': ' || c.sal);

            if c.sal <= 5000 then
                temp := 0.05;
            elsif c.sal <= 10000 then
                temp := 0.03;
            elsif c.sal <= 15000 then
                temp := 0.02;
            else
                temp := 0.01;
            end if;

            dbms_output.put_line(c.sal || ': ' || c.id || ', ' || temp);
            --update employees set salary = salary * (1 + temp) where employee_id = c.id;
    end loop;
end;
```

17. 隐式游标：更新指定员工 salary(涨工资 10)，如果该员工没有找到，则打印”查无此人”信息

```
begin
            update employees set salary = salary + 10 where employee_id = 1005;
```

```
        if sql%notfound then
            dbms_output.put_line('查无此人!');
        end if;
end;

*********************************************************************************
                                    异常处理
*********************************************************************************
[预定义异常]
declare

   v_sal employees.salary%type;
begin
   select salary into v_sal
   from employees
   where employee_id >100;

   dbms_output.put_line(v_sal);

exception
   when Too_many_rows then dbms_output.put_line('输出的行数太多了');
end;

[非预定义异常]
declare

   v_sal employees.salary%type;
   --声明一个异常
   delete_mgr_excep exception;
   --把自定义的异常和 oracle 的错误关联起来
   PRAGMA EXCEPTION_INIT(delete_mgr_excep,-2292);
begin
   delete from employees
   where employee_id = 100;

   select salary into v_sal
   from employees
   where employee_id >100;

   dbms_output.put_line(v_sal);

exception
```

```
   when Too_many_rows then dbms_output.put_line('输出的行数太多了');
   when delete_mgr_excep then dbms_output.put_line('Manager 不能直接被删除');
end;

[用户自定义异常]
declare

   v_sal employees.salary%type;
   --声明一个异常
   delete_mgr_excep exception;
   --把自定义的异常和 oracle 的错误关联起来
   PRAGMA EXCEPTION_INIT(delete_mgr_excep,-2292);

   --声明一个异常
   too_high_sal exception;
begin

   select salary into v_sal
   from employees
   where employee_id =100;

   if v_sal > 1000 then
       raise too_high_sal;
   end if;

   delete from employees
   where employee_id = 100;

   dbms_output.put_line(v_sal);

exception
   when Too_many_rows then dbms_output.put_line('输出的行数太多了');
   when delete_mgr_excep then dbms_output.put_line('Manager 不能直接被删除');
   --处理异常
   when too_high_sal then dbms_output.put_line('工资过高了');
end;
```

18. 异常的基本程序:
通过 select ... into ... 查询某人的工资，若没有查询到，则输出 "未找到数据"

```
declare
   --定义一个变量
```

```
   v_sal employees.salary%type;
begin
   --使用  select ... into ...  为  v_sal  赋值
   select salary into v_sal from employees where employee_id = 1000;
   dbms_output.put_line('salary:    ' || v_sal);
exception
   when No_data_found then
         dbms_output.put_line('未找到数据');
end;
```

或

```
declare
   --定义一个变量
   v_sal employees.salary%type;
begin
   --使用  select ... into ...  为  v_sal  赋值
   select salary into v_sal from employees;
   dbms_output.put_line('salary:    ' || v_sal);
exception
   when No_data_found then
         dbms_output.put_line('未找到数据!');
   when Too_many_rows then
         dbms_output.put_line('数据过多!');
end;
```

19. 更新指定员工工资，如工资小于 300，则加 100；对 NO_DATA_FOUND 异常，TOO_MANY_ROWS 进行处理.

```
declare
    v_sal employees.salary%type;
begin
    select salary into v_sal from employees where employee_id = 100;

    if(v_sal < 300) then update employees set salary = salary + 100 where employee_id = 100;
    else dbms_output.put_line('工资大于 300');
    end if;
exception
    when no_data_found then dbms_output.put_line('未找到数据');
     when too_many_rows then dbms_output.put_line('输出的数据行太多');
end;
```

20. 处理非预定义的异常处理: "违反完整约束条件"

```
declare
   --1. 定义异常
   temp_exception exception;

   --2. 将其定义好的异常情况，与标准的 ORACLE 错误联系起来，使用 EXCEPTION_INIT 语句
   PRAGMA EXCEPTION_INIT(temp_exception, -2292);
begin
   delete from employees where employee_id = 100;

exception
   --3. 处理异常
   when temp_exception then
         dbms_output.put_line('违反完整性约束!');
end;
```

21. 自定义异常：更新指定员工工资，增加 100；若该员工不存在则抛出用户自定义异常：no_result

```
declare
   --自定义异常
   no_result exception;
begin
   update employees set salary = salary + 100 where employee_id = 1001;

   --使用隐式游标，抛出自定义异常
   if sql%notfound then
      raise no_result;
   end if;

exception

   --处理程序抛出的异常
   when no_result then
      dbms_output.put_line('更新失败');
end;
```

****

存储函数和过程

****

[存储函数：有返回值，创建完成后，通过 select function() from dual;执行]

[存储过程：由于没有返回值，创建完成后，不能使用 select 语句，只能使用 pl/sql 块执行]

[格式]
--函数的声明(有参数的写在小括号里)
create or replace function func_name(v_param varchar2)
--返回值类型
return varchar2
is
--PL/SQL 块变量、记录类型、游标的声明(类似于前面的 declare 的部分)
begin
--函数体(可以实现增删改查等操作，返回值需要 return)
	return 'helloworld'|| v_param;
end;

22.1 函数的 helloworld: 返回一个 "helloworld" 的字符串

```
create or replace function hello_func
return varchar2
is
begin
	return 'helloworld';
end;
```

执行函数

```
begin
    dbms_output.put_line(hello_func());
end;
```

或者： select hello_func() from dual;

22.2 返回一个"helloworld: atguigu"的字符串，其中 atguigu 由执行函数时输入。

```
--函数的声明(有参数的写在小括号里)
create or replace function hello_func(v_logo varchar2)
--返回值类型
return varchar2
is
--PL/SQL 块变量的声明
begin
--函数体
	return 'helloworld'|| v_logo;
end;
```

22.3 创建一个存储函数，返回当前的系统时间

```
create or replace function func1
return date
is
--定义变量
v_date date;
begin
-函数体
--v_date := sysdate;
	select sysdate into v_date from dual;
		dbms_output.put_line('我是函数哦');

		return v_date;
end;
```

执行法 1：

```
select func1 from dual;
```

执行法 2：

```
declare
  v_date date;
begin
  v_date := func1;
  dbms_output.put_line(v_date);
end;
```

23. 定义带参数的函数：两个数相加

```
create or replace function add_func(a number, b number)
return number
is
begin
		return (a + b);
end;
```

执行函数

```
begin
	dbms_output.put_line(add_func(12, 13));
end;
```

或者

```
	select add_func(12,13) from dual;
```

24. 定义一个函数: 获取给定部门的工资总和, 要求:部门号定义为参数, 工资总额定义为返回值.

```
create or replace function sum_sal(dept_id number)
        return number
        is

        cursor sal_cursor is select salary from employees where department_id = dept_id;
        v_sum_sal number(8) := 0;
begin
        for c in sal_cursor loop
            v_sum_sal := v_sum_sal + c.salary;
        end loop;

        --dbms_output.put_line('sum salary: ' || v_sum_sal);
        return v_sum_sal;
end;
```

执行函数

```
begin
    dbms_output.put_line(sum_sal(80));
end;
```

25. 关于 OUT 型的参数: 因为函数只能有一个返回值, PL/SQL 程序可以通过 OUT 型的参数实现有多个返回值

要求: 定义一个函数: 获取给定部门的工资总和 和 该部门的员工总数(定义为 OUT 类型的参数).
要求: 部门号定义为参数, 工资总额定义为返回值.

```
create or replace function sum_sal(dept_id number, total_count out number)
        return number
        is

        cursor sal_cursor is select salary from employees where department_id = dept_id;
        v_sum_sal number(8) := 0;
begin
        total_count := 0;

        for c in sal_cursor loop
```

```
            v_sum_sal := v_sum_sal + c.salary;
            total_count := total_count + 1;
        end loop;

        --dbms_output.put_line('sum salary: ' || v_sum_sal);
        return v_sum_sal;
end;
```

执行函数:

```
delare
   v_total number(3) := 0;

begin
      dbms_output.put_line(sum_sal(80, v_total));
      dbms_output.put_line(v_total);
end;
```

26*. 定义一个存储过程: 获取给定部门的工资总和(通过 out 参数), 要求:部门号和工资总额定义为参数

```
create or replace procedure sum_sal_procedure(dept_id number, v_sum_sal out number)
        is

        cursor sal_cursor is select salary from employees where department_id = dept_id;
begin
        v_sum_sal := 0;

        for c in sal_cursor loop
            --dbms_output.put_line(c.salary);
            v_sum_sal := v_sum_sal + c.salary;
        end loop;

        dbms_output.put_line('sum salary: ' || v_sum_sal);
end;
[执行]
declare
       v_sum_sal number(10) := 0;
begin
       sum_sal_procedure(80,v_sum_sal);
end;
```

27*. 自定义一个存储过程完成以下操作:
对给定部门(作为输入参数)的员工进行加薪操作, 若其到公司的时间在
(? , 95) 期间, 为其加薪 %5
[95 , 98) %3
[98, ?) %1
得到以下返回结果: 为此次加薪公司每月需要额外付出多少成本(定义一个 OUT 型的输出参数).

```
create or replace procedure add_sal_procedure(dept_id number, temp out number)

is

        cursor sal_cursor is select employee_id id, hire_date hd, salary sal from employees where department_id = dept_id;
        a number(4, 2) := 0;
begin
        temp := 0;

        for c in sal_cursor loop
            a := 0;

            if c.hd < to_date('1995-1-1', 'yyyy-mm-dd') then
                a := 0.05;
            elsif c.hd < to_date('1998-1-1', 'yyyy-mm-dd') then
                a := 0.03;
            else
                a := 0.01;
            end if;

            temp := temp + c.sal * a;
            update employees set salary = salary * (1 + a) where employee_id = c.id;
        end loop;
end;
```

****

触发器

****

一个 helloworld 级别的触发器

```
create or replace trigger hello_trigger
after
update on employees
--for each row
```

```
begin
    dbms_output.put_line('hello...');
    --dbms_output.put_line('old.salary:'|| :OLD.salary||',new.salary'||:NEW.salary);
end;
```

然后执行：update employees set salary = salary + 1000;

28. 触发器的 helloworld: 编写一个触发器，在向 emp 表中插入记录时，打印 'helloworld'

```
create or replace trigger emp_trigger
after
insert on emp
for each row
begin
        dbms_output.put_line('helloworld');
end;
```

29. 行级触发器: 每更新 employees 表中的一条记录，都会导致触发器执行

```
create or replace trigger employees_trigger
after
update on employees
for each row
begin
        dbms_output.put_line('修改了一条记录!');
end;
```

语句级触发器: 一个 update/delete/insert 语句只使触发器执行一次

```
create or replace trigger employees_trigger
after
update on employees
begin
        dbms_output.put_line('修改了一条记录!');
end;
```

30. 使用 :new, :old 修饰符

```
create or replace trigger employees_trigger
after
update on employees
for each row
begin
```

```
        dbms_output.put_line('old salary: ' || :old.salary || ', new salary: ' || :new.salary);
end;
```

31. 编写一个触发器，在对 my_emp 记录进行删除的时候，在 my_emp_bak 表中备份对应的记录

1). 准备工作:

```
create table my_emp as select employee_id id, last_name name, salary sal from employees;

create table my_emp_bak as select employee_id id, last_name name, salary sal from employees
where 1 = 2
```

2).

```
create or replace trigger bak_emp_trigger
        before delete on my_emp
        for each row

begin
        insert into my_emp_bak values(:old.id, :old.name, :old.sal);
end;
```

Hibernate 面试题分析

1. Hibernate 的检索方式有哪些 ？

① 导航对象图检索
② OID 检索
③ HQL 检索
④ QBC 检索
⑤ 本地 SQL 检索

2. 在 Hibernate 中 Java 对象的状态有哪些 ？

①. 临时状态（transient）：不处于 Session 的缓存中。OID 为 null 或等于 id 的 unsaved-value 属性值
②. 持久化状态（persistent）：加入到 Session 的缓存中。
③. 游离状态(detached)：已经被持久化，但不再处于 Session 的缓存中。

3. Session 的清理和清空有什么区别？

清理缓存调用的是 session.flush() 方法. 而清空调用的是 session.clear() 方法.
Session 清理缓存是指按照缓存中对象的状态的变化来同步更新数据库，但不清空缓存；清空是把 Session 的缓存置空，但不同步更新数据库；

4. load()和 get()的区别

①：如果数据库中，没有 OID 指定的对象。通过 get 方法加载，则返回的是一个 null；通过 load 加载，则返回一个代理对象，如果后面代码如果调用对象的某个属性会抛出异常：org.hibernate.ObjectNotFoundException；
②：load 支持延迟加载，get 不支持延迟加载。

5. hibernate 优缺点

①. 优点：
> 对 JDBC 访问数据库的代码做了封装，简化了数据访问层繁琐的重复性代码
> 映射的灵活性，它支持各种关系数据库，从一对一到多对多的各种复杂关系.
> 非侵入性、移植性会好
> 缓存机制：提供一级缓存和二级缓存

②. 缺点：

> 无法对 SQL 进行优化
> 框架中使用 ORM 原则，导致配置过于复杂
> 执行效率和原生的 JDBC 相比偏差：特别是在批量数据处理的时候
> 不支持批量修改、删除

6. 描述使用 Hibernate 进行大批量更新的经验.

直接通过 JDBC API 执行相关的 SQl 语句或调用相关的存储过程是最佳的方式

7. Hibernate 的 OpenSessionView 问题

①. 用于解决懒加载异常，主要功能就是把 Hibernate Session 和一个请求的线程绑定在一起，直到页面完整输出，这样就可以保证页面读取数据的时候 Session 一直是开启的状态，如果去获取延迟加载对象也不会报错。

②. 问题：如果在业务处理阶段大批量处理数据，有可能导致一级缓存里的对象占用内存过多导致内存溢出，另外一个是连接问题： Session 和数据库 Connection 是绑定在一起的，如果业务处理缓慢也会导致数据库连接得不到及时的释放，造成连接池连接不够. 所以在并发量较大的项目中不建议使用此种方式，可以考虑使用迫切左外连接 (LEFT OUTER JOIN FETCH) 或手工对关联的对象进行初始化.

③. 配置 Filter 的时候要放在 Struts2 过滤器的前面，因为它要页面完全显示完后再退出.

8. Hibernate 中 getCurrentSession() 和 openSession() 的区别 ?

①. getCurrentSession() 它会先查看当前线程中是否绑定了 Session，如果有则直接返回，如果没有再创建. 而 openSession() 则是直接 new 一个新的 Session 并返回。

②. 使用 ThreadLocal 来实现线程 Session 的隔离。

③. getCurrentSession() 在事务提交的时候会自动关闭 Session，而 openSession() 需要手动关闭.

9.如何调用原生 SQL ?

调用 Session 的 doWork() 方法.

10. 说说 Hibernate 的缓存：

Hibernate 缓存包括两大类：Hibernate 一级缓存和 Hibernate 二级缓存：

1）. Hibernate 一级缓存又称为“Session 的缓存”，它是内置的，不能被卸载。由于 Session 对象的生命周期通常对应一个数据库事务或者一个应用事务，

因此它的缓存是事务范围的缓存。在第一级缓存中，持久化类的每个实例都具有唯一的 OID。

2）. Hibernate 二级缓存又称为“SessionFactory 的缓存”，由于 SessionFactory 对象的生命周期和应用程序的整个过程对应，因此 Hibernate 二级缓存是进程范围或者集群范围的缓存，有可能出现并发问题，因此需要采用适当的并发访问策略，该策略为被缓存的数据提供了事务隔离级别。第二级缓存是可选的，是一个可配置的插件，在默认情况下，SessionFactory 不会启用这个插件。

当Hibernate根据ID访问数据对象的时候，首先从Session一级缓存中查；查不到，如果配置了二级缓存，那么从二级缓存中查；如果都查不到，再查询数据库，把结果按照 ID 放入到缓存删除、更新、增加数据的时候，同时更新缓存。

Spring 面试题分析

1. 开发中主要使用 Spring 的什么技术 ？

①. IOC 容器管理各层的组件
②. 使用 AOP 配置声明式事务
③. 整合其他框架.

2. 简述 AOP 和 IOC 概念

AOP: Aspect Oriented Program, 面向(方面)切面的编程;Filter(过滤器)也是一种 AOP. AOP 是一种新的方法论, 是对传统 OOP(Object-Oriented Programming, 面向对象编程) 的补充. AOP 的主要编程对象是切面(aspect), 而切面模块化横切关注点.可以举例通过事务说明.

IOC: Invert Of Control, 控制反转. 也成为 DI(依赖注入)其思想是反转资源获取的方向. 传统的资源查找方式要求组件向容器发起请求查找资源.作为回应, 容器适时的返回资源. 而应用了 IOC 之后, 则是容器主动地将资源推送给它所管理的组件,组件所要做的仅是选择一种合适的方式来接受资源. 这种行为也被称为查找的被动形式

3. 在 Spring 中如何配置 Bean ？

Bean 的配置方式: 通过全类名(反射)、通过工厂方法(静态工厂方法 & 实例工厂方法)、FactoryBean

4. IOC 容器对 Bean 的生命周期:

①. 通过构造器或工厂方法创建 Bean 实例
②. 为 Bean 的属性设置值和对其他 Bean 的引用
③. 将 Bean 实例传递给 Bean 后置处理器的 postProcessBeforeInitialization 方法
④. 调用 Bean 的初始化方法(init-method)
⑤. 将 Bean 实例传递给 Bean 后置处理器的 postProcessAfterInitialization 方法
⑦. Bean 可以使用了
⑧. 当容器关闭时, 调用 Bean 的销毁方法(destroy-method)

5. Spring 如何整合 Struts2 ？

整合 Struts2, 即由 IOC 容器管理 Struts2 的 Action:

> 安装 Spring 插件: 把 struts2-spring-plugin-2.2.1.jar 复制到当前 WEB 应用的 WEB-INF/lib 目录下
> 在 Spring 的配置文件中配置 Struts2 的 Action 实例
> 在 Struts 配置文件中配置 action, 但其 class 属性不再指向该 Action 的实现类, 而是指向 Spring 容器中 Action 实例的 ID

6. Spring 如何整合 Hibernate

整合 Hibernate, 即由 IOC 容器生成 SessionFactory 对象, 并使用 Spring 的声明式事务

> 利用 LocalSessionFactoryBean 工厂 Bean, 声明一个使用 XML 映射文件的 SessionFactory 实例.
> 利用 HibernateTransactionManager 配置 Hibernate 的事务管理器

7. Spring MVC 比较 Struts2

①. Spring MVC 的入口是 Servlet, 而 Struts2 是 Filter
②. Spring MVC 会稍微比 Struts2 快些. Spring MVC 是基于方法设计, 而 Sturts2 是基于类, 每次发一次请求都会实例一个 Action.
③. Spring MVC 使用更加简洁, 开发效率 Spring MVC 确实比 struts2 高: 支持 JSR303, 处理 ajax 的请求更方便
④. Struts2 的 OGNL 表达式使页面的开发效率相比 Spring MVC 更高些.

8. Spring MVC 的运行流程

①. 在整个 Spring MVC 框架中， DispatcherServlet 处于核心位置，负责协调和组织不同组件以完成请求处理并返回响应的工作
②. SpringMVC 处理请求过程：
> 若一个请求匹配 DispatcherServlet 的请求映射路径(在 web.xml 中指定), WEB 容器将该请求转交给 DispatcherServlet 处理
>DispatcherServlet 接收到请求后, 将根据请求信息(包括 URL、HTTP 方法、请求头、请求参数、Cookie 等)及 HandlerMapping 的配置找到处理请求的处理器(Handler). 可将 HandlerMapping 看成路由控制器，将 Handler 看成目标主机。
> 当 DispatcherServlet 根据 HandlerMapping 得到对应当前请求的 Handler 后，通过 HandlerAdapter 对 Handler 进行封装，再以统一的适配器接口调用 Handler。
> 处理器完成业务逻辑的处理后将返回一个 ModelAndView 给

DispatcherServlet, ModelAndView 包含了视图逻辑名和模型数据信息

> DispatcherServlet 借助 ViewResoler 完成逻辑视图名到真实视图对象的解析

> 得到真实视图对象 View 后, DispatcherServlet 使用这个 View 对 ModelAndView 中的模型数据进行视图渲染

9. 说出 Spring MVC 常用的 5 个注解:

@RequestMapping 、 @PathVariable 、 @RequestParam 、 @RequestBoy 、 @ResponseBody

10. 如何使用 SpringMVC 完成 JSON 操作：

①. 配置 MappingJacksonHttpMessageConverter
②. 使用 @RequestBody 注解或 ResponseEntity 作为返回值

Struts2 面试题分析

1. 简述 Struts2 的工作流程:

①. 请求发送给 StrutsPrepareAndExecuteFilter
②. StrutsPrepareAndExecuteFilter 判定该请求是否是一个 Struts2 请求
③. 若该请求是一个 Struts2 请求，则 StrutsPrepareAndExecuteFilter 把请求的处理交给 ActionProxy
④. ActionProxy 创建一个 ActionInvocation 的实例，并进行初始化
⑤. ActionInvocation 实例在调用 Action 的过程前后，涉及到相关拦截器（Intercepter）的调用。
⑥. Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。调用结果的 execute 方法，渲染结果。
⑦. 执行各个拦截器 invocation.invoke() 之后的代码
⑧. 把结果发送到客户端

2. Struts2 拦截器 和 过滤器 的区别：

①、过滤器依赖于 Servlet 容器，而拦截器不依赖于 Servlet 容器。
②、Struts2 拦截器只能对 Action 请求起作用，而过滤器则可以对几乎所有请求起作用。
③、拦截器可以访问 Action 上下文(ActionContext)、值栈里的对象(ValueStack)，而过滤器不能.
④、在 Action 的生命周期中，拦截器可以多次调用，而过滤器只能在容器初始化时被调用一次。

3. 为什么要使用 Struts2 & Struts2 的优点：

①. 基于 MVC 架构，框架结构清晰。
②. 使用 OGNL: OGNL 可以快捷的访问值栈中的数据、调用值栈中对象的方法
③. 拦截器: Struts2 的拦截器是一个 Action 级别的 AOP, Struts2 中的许多特性都是通过拦截器来实现的, 例如异常处理，文件上传，验证等。拦截器是可配置与重用的
④. 多种表现层技术. 如：JSP、FreeMarker、Velocity 等

4. Struts2 如何访问 HttpServletRequest、HttpSession、ServletContext 三个域对象 ？

①. 与 Servlet API 解耦的访问方式

> 通过 ActionContext 访问域对象对应的 Map 对象
> 通过实现 Aware 接口使 Struts2 注入对应的 Map 对象

②. 与 Servlet API 耦合的访问方式
> 通过 ServletActionContext 直接获取 Servlet API 对象
> 通过实现 ServletXxxAware 接口的方式使 Struts2 注入对应的对象

5. Struts2 中的默认包 struts-default 有什么作用？

①. struts-default 包是 struts2 内置的，它定义了 struts2 内部的众多拦截器和 Result 类型，而 Struts2 很多核心的功能都是通过这些内置的拦截器实现，如：从请求中把请求参数封装到 action、文件上传和数据验证等等都是通过拦截器实现的。当包继承了 struts-default 包才能使用 struts2 为我们提供的这些功能。

②.struts-default 包是在 struts-default.xml 中定义，struts-default.xml 也是 Struts2 默认配置文件。Struts2 每次都会自动加载 struts-default.xml 文件。

③. 通常每个包都应该继承 struts-default 包。

6. 说出 struts2 中至少 5 个的默认拦截器

exception; fileUpload; i18n; modelDriven; params; prepare; token; tokenSession; validation 等

7. 谈谈 ValueStack：

①. ValueStack 贯穿整个 Action 的生命周期，保存在 request 域中，所以 ValueStack 和 request 的生命周期一样. 当 Struts2 接受一个请求时，会迅速创建 ActionContext，ValueStack，Action. 然后把 Action 存放进 ValueStack，所以 Action 的实例变量可以被 OGNL 访问。请求来的时候，Action、ValueStack 的生命开始；请求结束，Action、ValueStack 的生命结束

②. 值栈是多实例的，因为 Action 是多例的(和 Servlet 不一样，Servelt 是单例的)，而每个 Action 都有一个对应的值栈，Action 对象默认保存在栈顶；

③. ValueStack 本质上就是一个 ArrayList(查看源代码得到)；

④. 使用 OGNL 访问值栈的内容时，不需要#号，而访问 request、session、application、attr 时，需要加#号；

⑤. Struts2 重写了 request 的 getAttribute 方法，所以可以使用 EL 直接访问值栈中的内容

8. ActionContext、ServletContext、pageContext 的区别 ？

①. ActionContext Struts2 的 API：是当前的 Action 的上下文环境
②. ServletContext 和 PageContext 是 Servlet 的 API

9. Struts2 有哪几种结果类型 ？

参看 struts-default.xml 中的相关配置：dispatcher、chain、redirect 等.

10. 拦截器的生命周期与工作过程 ？

每个拦截器都是需要实现 Interceptor 接口

> init()：在拦截器被创建后立即被调用，它在拦截器的生命周期内只被调用一次. 可以在该方法中对相关资源进行必要的初始化；

> intercept(ActionInvocation invocation)：每拦截一个动作请求，该方法就会被调用一次；

> destroy：该方法将在拦截器被销毁之前被调用，它在拦截器的生命周期内也只被调用一次；

11. 如何在 Struts2 中使用 Ajax 功能 ？

①. JSON plugin
②. DOJO plugin
③. DWR plugin
④. 使用 Stream 结果类型.

Android 面试题大全

一． Android 入门

1、描述一下 android 的系统架构

android 系统架构分从下往上为 linux 内核层、运行库、应用程序框架层、和应用程序层。

Linux kernel:

负责硬件的驱动程序、网络、电源、系统安全以及内存管理等功能。

Libraries 和 android Runtime:

Libraries：即 c/c++函数库部分，大多数都是开放源代码的函数库，例如 webkit（引擎），该函数库负责 android 网页浏览器的运行，例如标准的 c 函数库 libc、openssl、sqlite 等，当然也包括支持游戏开发 2dsgl 和 3dopengles，在多媒体方面有 mediaframework 框架来支持各种影音和图形文件的播放与显示，例如 mpeg4、h.264、mp3、 aac、amr、jpg 和 png 等众多的多媒体文件格式。

Runtime：负责解释和执行生成的 dalvik 格式的字节码。

Application framework（应用软件架构）：

java 应用程序开发人员主要是使用该层封装好的 api 进行快速开发。

applications:该层是 java 的应用程序层，android 内置 googlemaps、e-mail、即时通信工具、浏览器、mp3 播放器等处于该层，java 开发人员开发的程序也处于该层，而且和内置的应用程序具有平等的位置，可以调用内置的应用程序，也可以替换内置的应用程序。

应用程序层：

android 应用程序使用框架的 api 并在框架下运行，这就带来了程序开发的高度一致性，另一方面也告诉我们，要想写出优质高效的程序就必须对整个 applicationframework 进行非常深入的理解。精通 applicationframework，你就可以真正的理解 android 的设计和运行机制，也就更能够驾驭整个应用层的开发。

总结：

下层为上层服务，上层需要下层的支持，调用下层的服务，这种严格分层的方式带来的极大的稳定性、灵活性和可扩展性，使得不同层的开发人员可以按照规范专心特定层的开发。

2、Dalvik 和标准 Java 虚拟机之间的主要差别？

Dalvik 和标准 Java 虚拟机（JVM）之间的首要差别之一，就是 Dalvik 基于寄存器，而 JVM 基于栈。

Dalvik 和 Java 之间的另外一大区别就是运行环境—Dalvik 经过优化，允许在有限的内存中同时运行多个虚拟机的实例，并且每一个 Dalvik 应用作为一个独立的 Linux 进程执行。

（1）虚拟机很小，使用的空间也小；

（2）Dalvik 没有 JIT 编译器；

（3）常量池已被修改为只使用 32 位的索引，以简化解释器；

（4）它使用自己的字节码，而非 Java 字节码。

3、Manifest.xml 文件中主要包括哪些信息？

答：manifest：根节点，描述了 package 中所有的内容。
User-sdk：指定支持的手机系统的最小版本
application：包含 package 中 application 级别组件声明的根节点。
activity：Activity 是用来与用户交互的主要工具。
receiver：IntentReceiver 能使的 application 获得数据的改变或者发生的操作，即使它当前不在运行。
service：Service 是能在后台运行任意时间的组件。

provider：ContentProvider 是用来管理持久化数据并发布给其他应用程序使用的组件。
uses-permission：请求你的 package 正常运作所需赋予的安全许可。
permission： 声明了安全许可来限制哪些程序能你 package 中的组件和功能。
instrumentation：声明了用来测试此 package 或其他 package 指令组件的代码。

二． Android 的四大组件

Activity 相关

1、什么是 Activity?

Activity 是一个负责与**用户交互**的组件，Activity 中所有操作都与用户密切相关，可以通过 setContentView(View)来**显示指定控件**。

在一个 android 应用中，一个 Activity 通常就是一个单独的屏幕，它上面可以显示一些控件也可以监听并处理用户的事件做出响应。

2、请描述一下 Activity 生命周期。

onCreate(Bundle savedInstanceState)：
创建 activity 时调用。设置在该方法中，还以 Bundle 的形式提供对以前储存的任何状态的访问！
onStart()：
activity 变为在屏幕上对用户可见时调用。
onResume()：
activity 开始与用户交互时调用（无论是启动还是重新启动一个活动，该方法总是被调用的）。
onPause()：
activity 被暂停或收回 cpu 和其他资源时调用，该方法用于保存活动状态的，也是保护现场，压栈吧！
onStop()：
activity 被停止并转为不可见阶段及后续的生命周期事件时调用。
onRestart()：
重新启动 activity 时调用。该活动仍在栈中，而不是启动新的活动。
onDestroy()：
activity 被完全从系统内存中移除时调用，该方法被调用

3、如何退出 Activity？如何安全退出已调用多个 Activity 的 Application？

在 Android 中退出程序比较麻烦，尤其是在多个 Activity 的程序中，在 2.2 之前可以采用如下代码退出程序：

```
1. ActivityManager am = (ActivityManager)getSystemService (Context.ACTIVIT
   Y_SERVICE);
2. am.restartPackage(getPackageName());
```

此种方法是一种最方便和最简单的退出程序的办法，但是在 2.2 和 2.2 之后就不能用了，一种常用的方法是自定义一个 Activity 的栈，在程序退出时将栈中的所有的 Activity 进行 finish。
还有一些其他的方式，在这
http://alex-yang-xiansoftware-com.iteye.com/blog/1099207 可查看。

4、如果后台的 Activity 由于某原因被系统回收了，如何在被系统回收之前保存当前状态？

答：重写 onSaveInstanceState()方法，在此方法中保存需要保存的数据，该方法将会在 activity 被回收之前调用。通过重写 onRestoreInstanceState()方法可以从中提取保存好的数据

5、 activity 在屏幕旋转时的生命周期

答:不设置 Activity 的 android:configChanges 时,切屏会重新调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次；设置 Activity 的 android:configChanges="orientation"时，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次；设置 Activity 的 android:configChanges="orientation|keyboardHidden"时，切屏不会重新调用各个生命周期，只会执行 onConfigurationChanged 方法。

6、 activity 的启动模式有哪些？是什么含义？

答：在 android 里，有 4 种 activity 的启动模式，分别为：
“standard”（默认）
“singleTop”
“singleTask”
“singleInstance”

当应用运行起来后就会开启一条线程，线程中会运行一个任务栈，当 Activity 实例创建后就会放入任务栈中。Activity 启动模式的设置在 AndroidManifest.xml 文件中，通过配置 Activity 的属性 android:launchMode=""设置。

1. Standared 模式（默认）

我们平时直接创建的 Activity 都是这种模式的 Activity，这种模式的 Activity 的特点是：只要你创建了 Activity 实例，一旦激活该 Activity，则会向任务栈中加入新创建的实例，退出 Activity 则会在任务栈中销毁该实例。

2. SingleTop 模式

这种模式会考虑当前要激活的 Activity 实例在任务栈中是否正处于栈顶，如果处于栈顶则无需重新创建新的实例，会重用已存在的实例，否则会在任务栈中创建新的实例。

3. SingleTask 模式

如果任务栈中存在该模式的 Activity 实例，则把栈中该实例以上的 Activity 实例全部移除，调用该实例的 newInstance()方法重用该 Activity，使该实例处於栈顶位置，否则就重新创建一个新的 Activity 实例。

4. SingleInstance 模式

当该模式 Activity 实例在任务栈中创建后，只要该实例还在任务栈中，即只要激活的是该类型的 Activity，都会通过调用实例的 newInstance()方法重用该 Activity，此时使用的都是同一个 Activity 实例，它都会处于任务栈的栈顶。此模式一般用于加载较慢的，比较耗性能且不需要每次都重新创建的 Activity。

7、跟 activity 和 Task 有关的 Intent 启动方式有哪些？其含义？

核心的 Intent Flag 有：

FLAG_ACTIVITY_NEW_TASK

FLAG_ACTIVITY_CLEAR_TOP

FLAG_ACTIVITY_SINGLE_TOP

FLAG_ACTIVITY_RESET_TASK_IF_NEEDED

FLAG_ACTIVITY_NEW_TASK

如果设置，这个 Activity 会成为历史 stack 中一个新 Task 的开始。一个 Task（从启动它的 Activity 到下一个 Task 中的 Activity）定义了用户可以迁移的 Activity 原子组。Task 可以移动到前台和后台；在某个特定 Task 中的所有 Activity

总是保持相同的次序。这个标志一般用于呈现“启动”类型的行为：它们提供用户一系列可以单独完成的事情，与启动它们的 Activity 完全无关。

FLAG_ACTIVITY_CLEAR_TOP

如果设置，并且这个 Activity 已经在当前的 Task 中运行，因此，不再是重新启动一个这个 Activity 的实例，而是在这个 Activity 上方的所有 Activity 都将关闭，然后这个 Intent 会作为一个新的 Intent 投递到老的 Activity（现在位于顶端）中。

FLAG_ACTIVITY_SINGLE_TOP

如果设置，并且这个 Activity 已经在当前的 Task 中运行，因此，不再是重新启动一个这个 Activity 的实例，而是在这个 Activity 上方的所有 Activity 都将关闭，然后这个 Intent 会作为一个新的 Intent 投递到老的 Activity（现在位于顶端）中。

FLAG_ACTIVITY_RESET_TASK_IF_NEEDED

如果设置这个标志，这个 activity 不管是从一个新的栈启动还是从已有栈推到栈顶，它都将以 the front door of the task 的方式启动。这就讲导致任何与应用相关的栈都讲重置到正常状态（不管是正在讲 activity 移入还是移除），如果需要，或者直接重置该栈为初始状态。

Service 相关

1、 如何开发一个 Service 组件？

服务的开发比较简单，如下：
第一步：继承 Service 类 public class SMSService extends Service {}
第二步：在 AndroidManifest.xml 文件中的<application>节点里对服务进行配置:<service android:name=".SMSService" />
第三步：启动服务
方法一：context.startService()：调用者与服务之间没有关连，即使调用者退出了，服务仍然运行
方法二：context.bindService()：调用者与服务绑定在了一起，调用者一旦退出，服务也就终止，大有“不求同时生，必须同时死”的特点

2、Service 的生命周期？

onCreate()：
该方法在服务被创建时调用，该方法只会被调用一次，无论调用多少次 startService()或 bindService()方法，服务也只被创建一次。
onDestroy()：

该方法在服务被终止时调用。与采用 Context.startService()方法启动服务有关的生命周期方法

onStart():

只有采用 Context.startService()方法启动服务时才会回调该方法。该方法在服务开始运行时被调用。多次调用 startService()方法尽管不会多次创建服务，但 onStart() 方法会被多次调用。

onBind():

只有采用 Context.bindService()方法启动服务时才会回调该方法。该方法在调用者与服务绑定时被调用，当调用者与服务已经绑定，多次调用 Context.bindService()方法并不会导致该方法被多次调用。

onUnbind():

只有采用 Context.bindService()方法启动服务时才会回调该方法。该方法在调用者与服务解除绑定时被调用

3、 Service 和 Thread 的区别？

答：servie 是系统的组件，它由系统进程托管（service manager）；它们之间的通信类似于 client 和 server，是一种轻量级的 ipc 通信，这种通信的载体是 binder，它是在 linux 层交换信息的一种 ipc。而 thread 是由本应用程序托管。

1）. Thread：Thread 是程序执行的最小单元，它是分配 CPU 的基本单位。可以用 Thread 来执行一些异步的操作。

2）. Service：Service 是 android 的一种机制，当它运行的时候如果是 Local Service，那么对应的 Service 是运行在主进程的 main 线程上的。如：onCreate，onStart 这些函数在被系统调用的时候都是在主进程的 main 线程上运行的。如果是 Remote Service，那么对应的 Service 则是运行在独立进程的 main 线程上。

既然这样，那么我们为什么要用 Service 呢？其实这跟 android 的系统机制有关，我们先拿 Thread 来说。Thread 的运行是独立于 Activity

的，也就是说当一个 Activity 被 finish 之后，如果你没有主动停止 Thread 或者 Thread 里的 run 方法没有执行完毕的话，Thread 也会一直执行。因此这里会出现一个问题：当 Activity 被 finish 之后，你不再持有该 Thread 的引用。另一方面，你没有办法在不同的 Activity 中对同一 Thread 进行控制。

举个例子：如果你的 Thread 需要不停地隔一段时间就要连接服务器做某种同步的话，该 Thread 需要在 Activity 没有 start 的时候也在运行。这个时候当你 start 一个 Activity 就没有办法在该 Activity 里面控制之前创建的 Thread。因此你便需要创建并启动一个 Service ，在 Service 里面创建、运行并控制该 Thread，这样便解决了该问题（因为任何 Activity 都可以控制同一 Service，而系统也只会创建一个对应 Service 的实例）。

因此你可以把 Service 想象成一种消息服务，而你可以在任何有 Context 的地方调用 Context.startService、Context.stopService、Context.bindService，Context.unbindService，来控制它，你也可以在 Service 里注册 BroadcastReceiver，在其他地方通过发送 broadcast 来控制它，当然这些都是 Thread 做不到的。

4、 简单描述 AIDL

答：由于每个应用程序都运行在自己的进程空间，并且可以从应用程序 UI 运行另一个服务进程，而且经常会在不同的进程间传递对象。在 Android 平台，一个进程通常不能访问另一个进程的内存空间，所以要想对话，需要将对象分解成操作系统可以理解的基本单元，并且有序的通过进程边界。

通过代码来实现这个数据传输过程是冗长乏味的，Android 提供了 AIDL 工具来处理这项工作。

AIDL (Android Interface Definition Language)是一种 IDL 语言，用于生成可以在 Android 设备上两个进程之间进行进程间通信(IPC)的代码。如果在一个进程中（例如 Activity）要调用另一个进程中（例如 Service）对象的操作，就可以使用 AIDL 生成可序列化的参数。

详细查看 http://www.cnblogs.com/over140/archive/2011/03/08/1976890.html

AIDL 支持的数据类型：

1．不需要 import 声明的简单 Java 编程语言类型(int,boolean 等)

2．String，CharSequence 不需要特殊声明

3．List，Map 和 Parcelables 类型，这些类型内所包含的数据成员也只能是简单数据类型，String 等其他比支持的类型.

BroadCastReceiver 相关

1、请描述一下 Broadcast Receiver。

Broadcast Receiver 用于接收并处理广播通知(broadcast announcements)。多数的广播是系统发起的，如地域变换、电量不足、来电来信等。程序也可以播放一个广播。程序可以有任意数量的 broadcast receivers 来响应它觉得重要的通知。broadcast receiver 可以通过多种方式通知用户：启动 activity、使用 NotificationManager、开启背景灯、振动设备、播放声音等，最典型的是在状态栏显示一个图标，这样用户就可以点它打开看通知内容。通常我们的某个应用或系统本身在某些事件（电池电量不足、来电来短信）来临时会广播一个 Intent 出去，我们可以利用注册一个 Broadcast Receiver 来监听到这些 Intent 并获取 Intent 中的数据。

2、注册广播有几种方式，这些方式有何优缺点？

答：首先写一个类要继承 BroadcastReceiver

第一种:在清单文件中声明,添加

```
<receive android:name=".IncomingSMSReceiver " >
<intent-filter>
   <action
android:name="android.provider.Telephony.SMS_RECEIVED")
<intent-filter>
```

```
    <receiver>
```

第二种使用代码进行注册如：

```
    IntentFilter filter =  new
    IntentFilter("android.provider.Telephony.SMS_RECEIVED");
    IncomingSMSReceiver receiver = new IncomgSMSReceiver();
    registerReceiver(receiver.filter);
```

两种注册类型的区别是：

1)第一种不是常驻型广播，也就是说广播跟随程序的生命周期。

2)第二种是常驻型，也就是说当应用程序关闭后，如果有信息广播来，程序也会被系统调用自动运行。

ContentProvider 相关

1、请介绍下 ContentProvider 是如何实现数据共享的。

一个程序可以通过实现一个 Content provider 的抽象接口将自己的数据完全暴露出去，而且 Content providers 是以类似数据库中表的方式将数据暴露。Content providers 存储和检索数据，通过它可以让所有的应用程序访问到，这也是应用程序之间唯一共享数据的方法。

要想使应用程序的数据公开化，可通过 2 种方法：创建一个属于你自己的 Content provider 或者将你的数据添加到一个已经存在的 Content provider 中，前提是有相同数据类型并且有写入 Content provider 的权限。

如何通过一套标准及统一的接口获取其他应用程序暴露的数据？Android 提供了 ContentResolver，外界的程序可以通过 ContentResolver 接口访问 ContentProvider 提供的数据。

Intent 和 Intent Filter。

1、请描述一下 Intent 和 Intent Filter

Intent 在 Android 中被翻译为"意图"，熟语来讲就是目的，他们是三种应用程序基本组件—activity，service 和 broadcast receiver 之间互相激活的手段。 在调用 Intent 名称时使用 ComponentName 也就是类的全名时为显示调用。这种方式一般用于应用程序的内部调用，因为你不一定会知道别人写的类的全名。我们来看看隐式 Intent 怎么用？ 首先我们先配置我们的 Activity 的 Intent Filter

```
<intent-filter>
    <action android:name="com.example.project.SHOW_CURRENT" />
</intent-filter>
```

这样在调用的时候指定 Intent 的 action，系统就是自动的去对比是哪个 intent-filter 符合我们的 Activity，找到后就会启动 Activity。 一个 intent filter 是 IntentFilter 类的实例，但是它一般不出现在代码中,而是出现在 android Manifest 文件中，以<intent-filter>的形式．（有一个例外是 broadcast receiver 的 intent filter 是使用 Context.registerReceiver()来动态设定的，其 intent filter 也是在代码中创建的.） 一个 filter 有 action，data，category 等字段．一个隐式 intent 为了能被某个 intent filter 接受，必须通过 3 个测试．一个 intent 为了被某个组件接受，则必须通过它所有的 intent filter 中的一个．

详解：http://blog.csdn.net/lmhit/article/details/5576250

综合

1、 Android 的四大组件是哪些，它们的作用？

Activity:

Activity 是 Android 程序与用户交互的窗口,是 Android 构造块中最基本的一种，它需要为保持各界面的状态，做很多持久化的事情，妥善管理生命周期以及一些跳转逻辑

service:

后台服务于 Activity，封装有一个完整的功能逻辑实现，接受上层指令，完成相关的事物，定义好需要接受的 Intent 提供同步和异步的接口

Content Provider:

是 Android 提供的第三方应用数据的访问方案，可以派生 Content Provider 类，对外提供数据，可以像数据库一样进行选择排序，屏蔽内部数据的存储细节，向外提供统一的接口模型，大大简化上层应用，对数据的整合提供了更方便的途径

BroadCast Receiver:

接受一种或者多种 Intent 作触发事件，接受相关消息，做一些简单处理，转换成一条 Notification，统一了 Android 的事件广播模型

三． Android 的 UI

1、简单介绍一下 Android 中的 View 和 ViewGroup

1、View

在 Andorid 应用程序中，UI 元素称为 View，它们都继承了 android.view.View 类。View 有众多的子类，包括 ViewGroup、基础控件、高级控件和布局。

基础控件主要包括：Button、ImageButton、ToggleButton、TextView、RadioButton、CheckBox、ImageView、ProgressBar、SeekBar 等。

2、ViewGroup

android.view.ViewGroup 类是 android.view.View 重要的子类，ViewGroup 类通常叫做“容器”，它就是由个控件组成的复杂控件，因为它也是 View 类的子类，所以本身也是控件。

ViewGroup 是高级控件的和布局的父类，高级控件是和布局与基础控件一样都是不指具体那个类，而是一类容器的总称。

高级控件都直接或者间接的继承了 android.view.ViewGroup 类，常用的高级控件主要包括：AutoCompleteTextView、Spinner、ListView、GridView、Gallery 等。

2、请介绍下 Android 中常用的五种布局。

常用五种布局方式，分别是：FrameLayout（框架布局），LinearLayout （线性布局），AbsoluteLayout（绝对布局），RelativeLayout（相对布局），TableLayout（表格布局）。

FrameLayout：

所有东西依次都放在左上角，会重叠，这个布局比较简单，也只能放一点比较简单的东西。

LinearLayout：

线性布局，每一个 LinearLayout 里面又可分为垂直布局和水平布局。当垂直布局时，每一行就只有一个元素，多个元素依次垂直往下；水平布局时，只有一行，每一个元素依次向右排列。

RelativeLayout：

相对布局可以理解为某一个元素为参照物，来定位的布局方式。主要属性有：相对于某一个元素 android:layout_below、android:layout_toLeftOf 相对于父元素的地方 android:layout_alignParentLeft、android:layout_alignParentRigh；

TableLayout：

表格布局，每一个 TableLayout 里面有表格行 TableRow，TableRow 里面可以具体定义每一个元素。每一个布局都有自己适合的方式，这五个布局元素可以相互嵌套应用，做出美观的界面。

AbsoluteLayout：

绝对布局用 X,Y 坐标来指定元素的位置，这种布局方式也比较简单，但是在屏幕旋转时，往往会出问题，而且多个元素的时候，计算比较麻烦。

3、android 中的动画有哪几类，它们的特点和区别是什么

答：两种，一种是 Tween 动画、还有一种是 Frame 动画。Tween 动画，这种实现方式可以使视图组件移动、放大、缩小以及产生透明度的变化；另一种 Frame 动画，传统的动画方法，通过顺序的播放排列好的图片来实现，类似电影。

4、ListView 的优化方案

1、如果自定义适配器，那么在 getView 方法中要考虑方法传进来的参数 contentView 是否为 null，如果为 null 就创建 contentView 并返回，如果不为 null 则直接使用。在这个方法中尽可能少创建 view。

2、给 contentView 设置 tag（setTag（））， 传入一个 viewHolder 对象，用于缓存要显示的数据，可以达到图像数据异步加载的效果。

3、如果 listview 需要显示的 item 很多，就要考虑分页加载。比如一共要显示 100 条或者更多的时候，我们可以考虑先加载 20 条，等用户拉到列表底部的时候再去加载接下来的 20 条。

5、View 的刷新：

postinvaliate()可以在分线程刷新
invaliate()只能在主线程中执行

6、NotifactionManager 使用原理

1． 通过 getSystemService 方法获得一个 NotificationManager 对象。

2． 创建一个 Notification 对象。每一个 Notification 对应一个 Notification 对象。在这一步需要设置显示在屏幕上方状态栏的通知消息、通知消息前方的图像资源 ID 和发出通知的时间。一般为当前时间。

3． 由于 Notification 可以与应用程序脱离。也就是说，即使应用程序被关闭，Notification 仍然会显示在状态栏 中。当应用程序再次启动后，又可以重新控制这些 Notification。如清除或替换它们。因此，需要创建一个 PendingIntent 对象。该对象由 Android 系统负责维护，因此，在应用程序关闭后，该对象仍然不会被释放。

4． 使用 Notification 类的 setLatestEventInfo 方法设置 Notification 的详细信息。

5． 使用 NotificationManager 类的 notify 方法显示 Notification 消息。在这一步需要指定标识 Notification 的唯一 ID。这个 ID 必须相对于同一个 NotificationManager 对象是唯一的，否则就会覆盖相同 ID 的 Notificaiton。

7、view 如何刷新？简述什么是双缓冲？

android 中实现 view 的更新有两个方法，一个是 invalidate，另一个是 postInvalidate，其中前者是在 UI 线程自身中使用，而后者在非 UI 线程中使用。

双缓冲

闪烁是图形编程的一个常见问题。当进行复杂的绘制操作时会导致呈现的图像闪烁或具有其他不可接受的外观。双缓冲的使用解决这些问题。双缓冲使用内存缓冲区来解决由多重绘制操作造成的闪烁问题。当使用双缓冲时，首先在内存缓冲区里完成所有绘制操作，而不是在屏幕上直接进行绘图。当所有绘制操作完成后，把内存缓冲区完成的图像直接复制到屏幕。因为在屏幕上只执行一个图形操作，所以消除了由复杂绘制操作造成的图像闪烁问题。

在 android 中实现双缓冲,可以使用一个后台画布 backcanvas,先把所有绘制操作都在这上面进行。等图画好了，然后在把 backcanvas 拷贝到

与屏幕关联的 canvas 上去,如下：

```
Bitmap bitmapBase = new Bitmap()

Canvas backcanvas = new Canvas(bitmapBase)

backcanvas.draw()...//画图

Canvas c = lockCanvas(null);

c.drawbitmap(bitmapBase);//把已经画好的图像输出到屏幕上

unlock(c)....
```

四． Android 数据存储与解析

1、请介绍下 Android 的数据存储方式。

Android 提供了 5 种方式存储数据：
（1）使用 SharedPreferences 存储数据；它是 Android 提供的用来存储一些简单配置信息的一种机制，采用了 XML 格式将数据存储到设备中。只能在同一个包内使用，不能在不同的包之间使用。
（2）文件存储数据；文件存储方式是一种较常用的方法，在 Android 中读取/写入文件

的方法，与 Java 中实现 I/O 的程序是完全一样的，提供了 openFileInput()和 openFileOutput()方法来读取设备上的文件。

（3）SQLite 数据库存储数据；SQLite 是 Android 所带的一个标准的数据库，它支持 SQL 语句，它是一个轻量级的嵌入式数据库。

（4）使用 ContentProvider 存储数据；主要用于应用程序之间进行数据交换，从而能够让其他的应用保存或读取此 Content Provider 的各种数据类型。

（5）网络存储数据；通过网络上提供给我们的存储空间来上传(存储)和下载(获取)我们存储在网络空间中的数据信息。

2、android 中有哪几种解析 xml 的类，官方推荐哪种？以及它们的原理和区别。

方式一：DOM 解析

优点：

XML 树在内存中完整存储，因此可以直接修改其数据和结构。 2.可以通过该解析器随时访问 XML 树中的任何一个节点。 3.DOM 解析器的 API 在使用上也相对比较简单。

缺点：

如果 XML 文档体积比较大时，将文档读入内存是非常消耗系统资源的。

使用场景：

DOM 是用与平台和语言无关的方式表示 XML 文档的官方 W3C 标准。DOM 是以层次结构组织的节点的集合。这个层次结构允许开发人员在树中寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构，然后才能进行任何工作。DOM 是基于对象层次结构的。

方式二：SAX 解析

优点：

SAX 对内存的要求比较低，因为它让开发人员自己来决定所要处理的标签。特别是当开发人员只需要处理文档中所包含的部分数据时，SAX 这种扩展能力得到了更好的体现。

缺点：

用 SAX 方式进行 XML 解析时，需要顺序执行，所以很难访问到同一文档中的不同数据。此外，在基于该方式的解析编码过程也相对复杂。

使用场景：

对于含有数据量十分巨大，而又不用对文档的所有数据进行遍历或者分析的时候，使用

该方法十分有效。该方法不用将整个文档读入内存，而只需读取到程序所需的文档标签处即可。

方式三：Xmlpull 解析

android SDK 提供了 xmlpull api，xmlpull 和 sax 类似，是基于流（stream）操作文件，然后根据节点事件回调开发者编写的处理程序。因为是基于流的处理，因此 xmlpull 和 sax 都比较节约内存资源，不会象 dom 那样要把所有节点以对橡树的形式展现在内存中。

xmlpull 比 sax 更简明，而且不需要扫描完整个流。

五． Android 核心机制

消息与异步通信机制

1、请解释下在单线程模型中 Message、Handler、Message Queue、Looper 之间的关系。

简单的说，Handler 获取当前线程中的 looper 对象，looper 用来从存放 Message 的 MessageQueue 中取出 Message，再有 Handler 进行 Message 的分发和处理。

Message Queue（消息队列）：用来存放通过 Handler 发布的消息，通常附属于某一个创建它的线程，可以通过 Looper.myQueue()得到当前线程的消息队列

Handler：可以发布或者处理一个消息或者操作一个 Runnable，通过 Handler 发布消息，消息将只会发送到与它关联的消息队列，然也只能处理该消息队列中的消息。

Looper：是 Handler 和消息队列之间通讯桥梁，程序组件首先通过 Handler 把消息传递给 Looper，Looper 把消息放入队列。Looper 也把消息队列里的消息广播给所有的

Handler：Handler 接受到消息后调用 handleMessage 进行处理

Message：消息的类型，在 Handler 类中的 handleMessage 方法中得到单个的消息进行处理

在单线程模型下，为了线程通信问题，Android 设计了一个 Message Queue（消息队列），线程间可以通过该 Message Queue 并结合 Handler 和 Looper 组件进行信息交换。下面将对它们进行分别介绍：

1．Message

Message 消息，理解为线程间交流的信息，处理数据后台线程需要更新 UI，则发送 Message 内含一些数据给 UI 线程。

2．Handler

Handler 处理者，是 Message 的主要处理者，负责 Message 的发送，Message 内容的执行处理。后台线程就是通过传进来的 Handler 对象引用来 sendMessage(Message)。而使用 Handler，需要 implement 该类的 handleMessage(Message)方法，它是处理这些 Message 的操作内容，例如 Update UI。通常需要子类化 Handler 来实现 handleMessage 方法。

3．Message Queue

Message Queue 消息队列，用来存放通过 Handler 发布的消息，按照先进先出执行。

每个 message queue 都会有一个对应的 Handler。Handler 会向 message queue 通过两种方法发送消息：sendMessage 或 post。这两种消息都会插在 message queue 队尾并按先进先出执行。但通过这两种方法发送的消息执行的方式略有不同：通过 sendMessage 发送的是一个 message 对象，会被 Handler 的 handleMessage()函数处理；而通过 post 方法发送的是一个 runnable 对象，则会自己执行。

4．Looper

Looper 是每条线程里的 Message Queue 的管家。Android 没有 Global 的 Message Queue，而 Android 会自动替主线程(UI 线程)建立 Message Queue，但在子线程里并没有建立 Message Queue。所以调用 Looper.getMainLooper()得到的主线程的 Looper 不为 NULL，但调用 Looper.myLooper() 得到当前线程的 Looper 就有可能为 NULL。对于子线程使用 Looper，API Doc 提供了正确的使用方法：这个 Message 机制的大概流程：

1．在 Looper.loop()方法运行开始后，循环地按照接收顺序取出 Message Queue 里面的非 NULL 的 Message。

2．一开始 Message Queue 里面的 Message 都是 NULL 的。当 Handler.sendMessage(Message)到 Message Queue，该函数里面设置了那个 Message 对象的 target 属性是当前的 Handler 对象。随后 Looper 取出了那个 Message，则调用 该 Message 的 target 指向的 Hander 的 dispatchMessage 函数对 Message 进行处理。在 dispatchMessage 方法里，如何处理 Message 则由用户指定，三个判断，优先级从高到低：

1) 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；

2) Handler 里面的 mCallback 指向的一个实现了 Callback 接口的对象，由其 handleMessage 进行处理；

3) 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。

3．Handler 处理完该 Message (update UI) 后，Looper 则设置该 Message 为 NULL，以便回收！

在网上有很多文章讲述主线程和其他子线程如何交互，传送信息，最终谁来执行处理信息之类的，个人理解是最简单的方法—判断 Handler 对象里面的 Looper 对象是属于哪条线程的，则由该线程来执行！

1．当 Handler 对象的构造函数的参数为空，则为当前所在线程的 Looper；

2．Looper.getMainLooper()得到的是主线程的 Looper 对象，Looper.myLooper()得到的是当前线程的 Looper 对象。

2、说说你对 AsyncTask 的理解

在开发 Android 移动客户端的时候往往要使用多线程来进行操作，我们通常会将耗时的操作放在单独的线程执行，避免其占用主线程而给用户带来不好的用户体验。但是在子线程中无法去操作主线程(UI 线程)，在子线程中操作 UI 线程会出现错误。因此 android 提供了一个类 Handler 来在子线程中来更新 UI 线程，用发消息的机制更新 UI 界面，呈

现给用户。这样就解决了子线程更新 UI 的问题。但是费时的任务操作总会启动一些匿名的子线程，太多的子线程给系统带来巨大的负担，随之带来一些性能问题。因此 android 提供了一个工具类 AsyncTask，顾名思义异步执行任务。这个 AsyncTask 生来就是处理一些后台的比较耗时
的任务，给用户带来良好用户体验的，从编程的语法上显得优雅了许多，不再需要子线程和
Handler 就可以完成异步操作并且刷新用户界面。

触摸机制

其它

1、说说 mvc 模式的原理，它在 android 中的运用。

mvc 是 model,view,controller 的缩写，mvc 包含三个部分：

模型（model）对象：是应用程序的主体部分，所有的业务逻辑都应该写在该层。

视图（view）对象：是应用程序中负责生成用户界面的部分。也是在整个 mvc 架构中用户唯一可以看到的一层，接收用户的输入，显示处理结果。

控制器（control）对象：是根据用户的输入，控制用户界面数据显示及更新 model 对象状态的部分，控制器更重要的一种导航功能，响应用户出发的相关事件，交给 m 层处理。

android 鼓励弱耦合和组件的重用，在 android 中 mvc 的具体体现如下：

1)视图层（view）：一般采用 xml 文件进行界面的描述，使用的时候可以非常方便的引入，当然，如果你对 android 了解的比较的多了话，就一定可以想到在 android 中也可以使用 javascript+html 等的方式作为 view 层，当然这里需要进行 java 和 javascript 之间的通信，幸运的是，android 提供了它们之间非常方便的通信实现。

2)控制层（controller）：android 的控制层的重任通常落在了众多的 acitvity 的肩上，这句话也就暗含了不要在 acitivity 中写代码，要通过 activity 交割 model 业务逻辑层处理,这样做的另外一个原因是 android 中的 acitivity 的响应时间是 5s，如果耗时的操作放在这里，程序就很容易被回收掉。

3)模型层（model）：对数据库的操作、对网络等的操作都应该在 model 里面处理，当然对业务计算等操作也是必须放在的该层的。

六． Android 应用开发相关

1、什么情况会导致 Force Close ？如何避免？能否捕获导致其的异常？

答：程序出现异常，比如 nullpointer。
避免：编写程序时逻辑连贯，思维缜密。能捕获异常，在 logcat 中能看到异常信息

2、Android 本身的 api 并未声明会抛出异常，则其在运行时有无可能抛出 runtime 异常，你遇到过吗？诺有的话会导致什么问题？如何解决？

答：会，比如 nullpointerException。我遇到过，比如 textview.setText()时，textview 没有初始化。会导致程序无法正常运行出现 forceclose。打开控制台查看 logcat 信息找出异常信息并修改程序。

3、DDMS 和 TraceView 的区别？

DDMS 是一个程序执行查看器，在里面可以看见线程和堆栈等信息
TraceView 是程序性能分析器 。

4、Android 中内存泄露出现情况有哪些？

1. 数据库的 cursor 没有关闭， 可以使用 startManagerCursor(cursor)
2. 构造 adapter 时,没有使用缓存 contentview,衍生 listview 的优化问题-----减少创建 view 的对象,充分使用 contentview,可以使用一静态类来优化处理 getview 的过程
3. Bitmap 对象不使用时没有释放，可以通过调用 bitmap.recycle()释放内存

5、什么是 ANR，如何避免它？

ANR：Application Not Responding。在 Android 中，活动管理器和窗口管理器这两个系统服务负责监视应用程序的响应，当用户操作的在 5s 内应用程序没能做出反应，

BroadcastReceiver 在 10 秒内没有执行完毕，就会出现应用程序无响应对话框，这既是 ANR。

避免方法：Activity 应该在它的关键生命周期方法（如 onCreate()和 onResume()）里尽可能少的去做创建操作。潜在的耗时操作，例如网络或数据库操作，或者高耗时的计算如改变位图尺寸，应该在子线程里（或者异步方式）来完成。主线程应该为子线程提供一个 Handler，以便完成时能够提交给主线程。

6、DDMS 中 traceview 的作用？

Traceview 是 android 平台配备一个很好的性能分析的工具。它可以通过图形化的方式让我们了解我们要跟踪的程序的性能，并且能具体到 method。
http://wbdban.javaeye.com/blog/564309

7、根据自己的理解描述下 Android 数字签名。

(1)所有的应用程序都必须有数字证书，Android 系统不会安装一个没有数字证书的应用程序
(2)Android 程序包使用的数字证书可以是自签名的，不需要一个权威的数字证书机构签名认证
(3)如果要正式发布一个 Android ，必须使用一个合适的私钥生成的数字证书来给程序签名，而不能使用 adt 插件或者 ant 工具生成的调试证书来发布。
(4)数字证书都是有有效期的，Android 只是在应用程序安装的时候才会检查证书的有效期。如果程序已经安装在系统中，即使证书过期也不会影响程序的正常功能。

七．其它

1、Android dvm 的进程和 Linux 的进程，应用程序的进程是否为同一个概念

答：DVM 指 dalivk 的虚拟机。每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。而每一个 DVM 都是在 Linux 中的一个进程，所以说可以认为是同一个概念。

2、sim 卡的 EF 文件是什么？有何作用

答：sim 卡的文件系统有自己规范，主要是为了和手机通讯，sim 本 身可以有自己的操作系统，EF 就是作存储并和手机通讯用的

3、什么是嵌入式实时操作系统，Android 操作系统属于实时操作系统吗？

嵌入式实时操作系统是指当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对处理系统作出快速响应，并控制所有实时任务协调一致运行的嵌入式操作系统。主要用于工业控制、 军事设备、 航空航天等领域对系统的响应时间有苛刻的要求，这就需要使用实时系统。又可分为软实时和硬实时两种，而 android 是基于 linux 内核的，因此属于软实时。

4、一条最长的短信息约占多少 byte？

中文 70(包括标点)，英文 160，160 个字节。

5、谈谈 Android 的 IPC（进程间通信）机制

IPC 是内部进程通信的简称， 是共享"命名管道"的资源。Android 中的 IPC 机制是为了让 Activity 和 Service 之间可以随时的进行交互，故在 Android 中该机制，只适用于 Activity 和 Service 之间的通信，类似于远程方法调用，类似于 C/S 模式的访问。通过定义 AIDL 接口文件来定义 IPC 接口。Servier 端实现 IPC 接口，Client 端调用 IPC 接口本地代理。

6、NDK 是什么？

NDK 是一些列工具的集合，NDK 提供了一系列的工具，帮助开发者迅速的开发 C/C++的动态库，并能自动将 so 和 java 应用打成 apk 包。
NDK 集成了交叉编译器，并提供了相应的 mk 文件和隔离 cpu、平台等的差异，开发人员只需简单的修改 mk 文件就可以创建出 so

7、Android 平台手机 5 大优势：

一、开放性

在优势方面，Android 平台首先就是其开发性，开发的平台允许任何移动终端厂商加入到 Android 联盟中来。显著的开放性可以使其拥有更多的开发者，随着用户和应用的日益丰富，一个崭新的平台也将很快走向成熟

开发性对于 Android 的发展而言，有利于积累人气，这里的人气包括消费者和厂商，而对于消费者来讲，随大的受益正是丰富的软件资源。开放的平台也会带来更大竞争，如此一来，消费者将可以用更低的价位购得心仪的手机。

二、挣脱运营商的束缚

在过去很长的一段时间，特别是在欧美地区，手机应用往往受到运营商制约，使用什么功能接入什么网络，几乎都受到运营商的控制。从去年 iPhone 上市 ，用户可以更加方便地连接网络，运营商的制约减少。随着 EDGE、HSDPA 这些 2G 至 3G 移动网络的逐步过渡和提升，手机随意接入网络已不是运营商口中的笑谈，当你可以通过手机 IM 软件方便地进行即时聊天时，再回想不久前天价的彩信和图铃下载业务，是不是像噩梦一样？

互联网巨头 Google 推动的 Android 终端天生就有网络特色，将让用户离互联网更近。

三、丰富的硬件选择

这一点还是与 Android 平台的开放性相关，由于 Android 的开放性，众多的厂商会推出千奇百怪，功能特色各具的多种产品。功能上的差异和特色，却不会影响到数据同步、甚至软件的兼容，好比你从诺基亚 Symbian 风格手机 一下改用苹果 iPhone ，同时还可将 Symbian 中优秀的软件带到 iPhone 上使用、联系人等资料更是可以方便地转移，是不是非常方便呢？

四、不受任何限制的开发商

Android 平台提供给第三方开发商一个十分宽泛、自由的环境，不会受到各种条条框框的阻扰，可想而知，会有多少新颖别致的软件会诞生。但也有其两面性，血腥、暴力、情色方面的程序和游戏如可控制正是留给 Android 难题之一。

五、无缝结合的 Google 应用

如今叱诧互联网的 Google 已经走过 10 年度历史，从搜索巨人到全面的互联网渗透，Google 服务如地图、邮件、搜索等已经成为连接用户和互联网的重要纽带，而 Android 平台手机将无缝结合这些优秀的 Google 服务。

8、Android 的 5 大不足：

一、安全和隐私

由于手机 与互联网的紧密联系，个人隐私很难得到保守。除了上网过程中经意或不经意留下的个人足迹，Google 这个巨人也时时站在你的身后，洞穿一切，因此，互联网的深入将会带来新一轮的隐私危机。

二、首先开卖 Android 手机的不是最大运营商

众所周知，T-Mobile 在 23 日，于美国纽约发布 了 Android 首款手机 G1。但是在北美市场，最大的两家运营商乃 AT&T 和 Verizon，而目前所知取得 Android 手机销售权的仅有 T-Mobile 和 Sprint，其中 T-Mobile 的 3G 网络相对于其他三家也要逊色不少，因此，用户可以买账购买 G1，能否体验到最佳的 3G 网络服务则要另当别论了！

三、运营商仍然能够影响到 Android 手机

在国内市场，不少用户对购得移动定制机不满，感觉所购的手机被人涂画了广告一般。这样的情况在国外市场同样出现。Android 手机的另一发售运营商 Sprint 就将在其机型中内置其手机商店程序。

四、同类机型用户减少

在不少手机论坛 都会有针对某一型号的子论坛，对一款手机的使用心得交流，并分享软件资源。而对于 Android 平台手机，由于厂商丰富，产品类型多样，这样使用同一款机型的用户越来越少，缺少统一机型的程序强化。举个稍显不当的例子，现在山寨机泛滥，品种各异，就很少有专门针对某个型号山寨机的讨论和群组，除了哪些功能异常抢眼、颇受追捧的机型以外。

五、过分依赖开发商缺少标准配置

在使用 PC 端的 Windows Xp 系统的时候，都会内置微软 Windows Media Player 这样一个浏览器程序，用户可以选择更多样的播放器，如 Realplay 或暴风影音等。但入手开始使用默认的程序同样可以应付多样的需要。在 Android 平台中，由于其开放性，软件更多依赖第三方厂商，比如 Android 系统的 SDK 中就没有内置音乐 播放器，全部依赖第三方开发，缺少了产品的统一性。

9、在 android 中，请简述 jni 的调用过程。

1) 安装和下载 Cygwin，下载 AndroidNDK

2)在 ndk 项目中 JNI 接口的设计

3)使用 C/C++实现本地方法

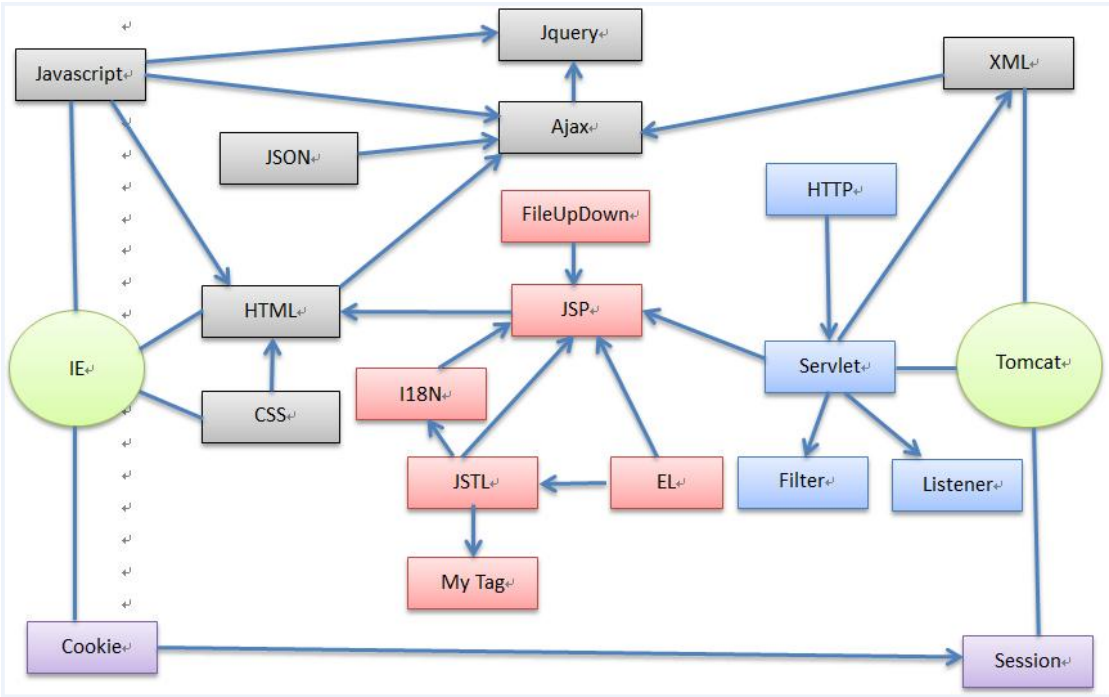
4)JNI 生成动态链接库.so 文件

5)将动态链接库复制到 java 工程，在 java 工程中调用，运行 java 工程即可

JavaWeb 面试题

1.Javaweb 技术的结构

1.1. Javaweb 技术结构图



1.2. 结构图说明:

整体分为四个部分:

1. 黑线: JavaScript 相关技术路线
2. 蓝线: Servlet 相关技术路线
3. 红线: Jsp 相关技术路线
4. 紫线: Web 会话相关技术路线

2.JavaScript 相关技术路线(黑线)

此部分包括: JavaScript, Jquery, Ajax, XML, JSON 和 HTML 等技术.

2.1. 列举 BOM 中常用的几个全局变量和全局方法?

全局对象: window

全局变量： document location history navigator screen

全局方法：alert() confirm() prompt() open() close()

2.2. 在 js 中如何创建一个对象?

```
var p1 = {name: "Tom", "my age" : 12};

function Person(name, age) {
    this.name = name;
    this.age = age;
}
var p2 = new Person("Jack", 14);
```

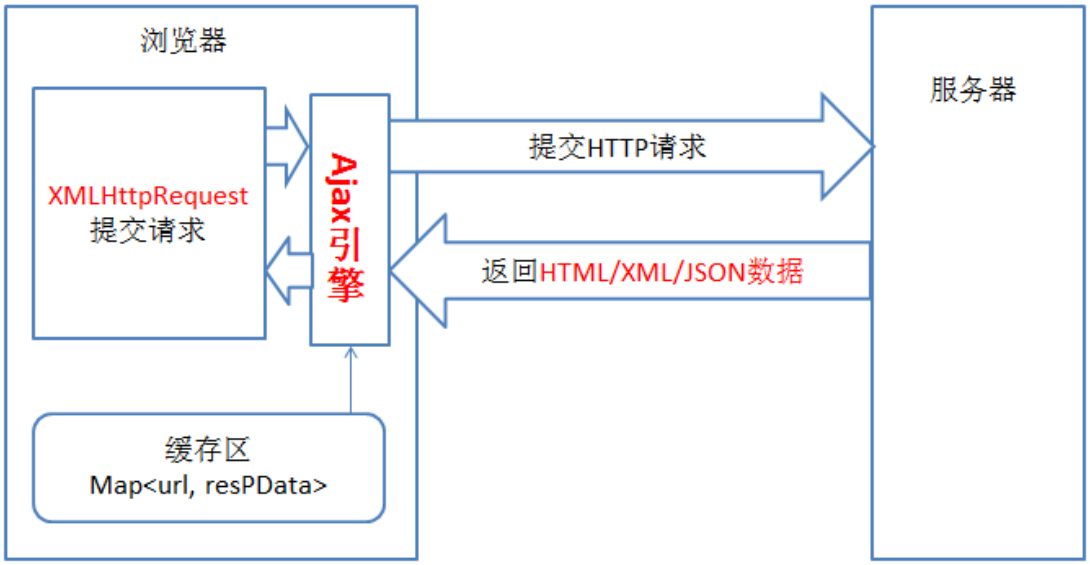
2.3. 在 js 中如何得到对象的属性?

```
var age = p2.age
//alert(age);
age = p1["my age"];
alert(age);
```

2.4. 谈谈 Ajax 技术

Ajax 原理

Ajax 的原理简单来说通过 XmlHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后用 javascript 来操作 DOM 而更新页面的局部显示。



Ajax 的优点：

1.最大的一点是页面无刷新，给用户的体验非常好。

2.使用异步方式与服务器通信，不需要打断用户操作，具有更加迅速的响应能力。

3.ajax 的原则是“按需取数据”，最大程度的减少冗余请求，减少服务器的负荷。

Ajax 的缺点：

1.破坏浏览器后退按钮的正常行为。在动态更新页面后，用户无法回到前一个页面的状态.

2.使用 JavaScript 作 Ajax 的引擎，JavaScript 的兼容性和 Debug 本身就让人头大。

Ajax 的应用场景：

1.文本输入提示（自动完成）的场景(注册)

2.对数据进行联动过滤的场景(三级联动)

2.5. 你觉得 jquery 有哪些好处？

jQuery 是轻量级的 javascript 框架

强大的选择器

出色的 DOM 操作的封装

可靠的事件处理机制

完善的 ajax 封装

出色的浏览器的兼容性
支持链式操作，隐式迭代
支持丰富的插件

jquery 的文档也非常的丰富

2.6. jquery 对象和 dom 对象如何转换？

1. jquery 转 DOM 对象:

jQuery 对象是一个数组对象，可以通过[index]的丰富得到 DOM 对象还可以

通过 get[index]去得到相应的 DOM 对象。

2. DOM 对象转 jQuery 对象:

$(DOM 对象)

2.7. jquery 中$.get()提交和$.post()提交的区别？

1. $.get() 方法使用 GET 方式提交请求,而$.post()使用 POST 方式。
2. GET 方式传输的数据大小不能超过 2KB 而 POST 要大的多
3. GET 方式请求的数据会被浏览器缓存起来，因此有安全问题。

2.8. $(document).ready()方法和 window.onload 区别？

答: 两个方法有相似的功能，但是在实行时机方面是有区别的。

1 window.onload 方法是在网页中所有的元素(包括元素的所有关联文件)完全加载到浏览器后才执行的。

2 $(document).ready() 方法可以在 DOM 载入就绪时就对其进行操纵，并调用执行绑定的函数。

2.9. xml 有哪些解析技术?区别是什么?

答:有 DOM,DOM4j,SAX,PULL 等

DOM: 一次性将整个文档加载到内存中, 生成一个对象树, 在处理大型文件时其性能下降的非常厉害。

DOM4J: 对 DOM 的进一步封装, API 使用更简洁

SAX:基于事件驱动的方法回调机制。每读取一小部分数据时就会回调事件处理器对象的方法, 但解析一旦开始就不能停止.

PULL: 也是基于事件驱动, 只是需要手动控制读取下一部分数据,这样得到想要的数据后就可以停止解析.

2.10. 你在项目中用到了 xml 技术的哪些方面?如何实现的?

答:用到了数据存贮，信息配置两方面。在做数据交换平台时，将不能数据源的数据组装成 XML 文件，然后将 XML 文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同 XML 文件中还原相关信息进行处理。在做软件配置时，利用 XML 可以很方便的进行，软件的各种配置参数都存贮在 XML 文件中。

2.11. 说说你对 JSON 的理解

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于标准 JavaScript 的一个子集,是一个 Js 对象或数组结构的字符串

JSON 有三类数据

1. 单个数据

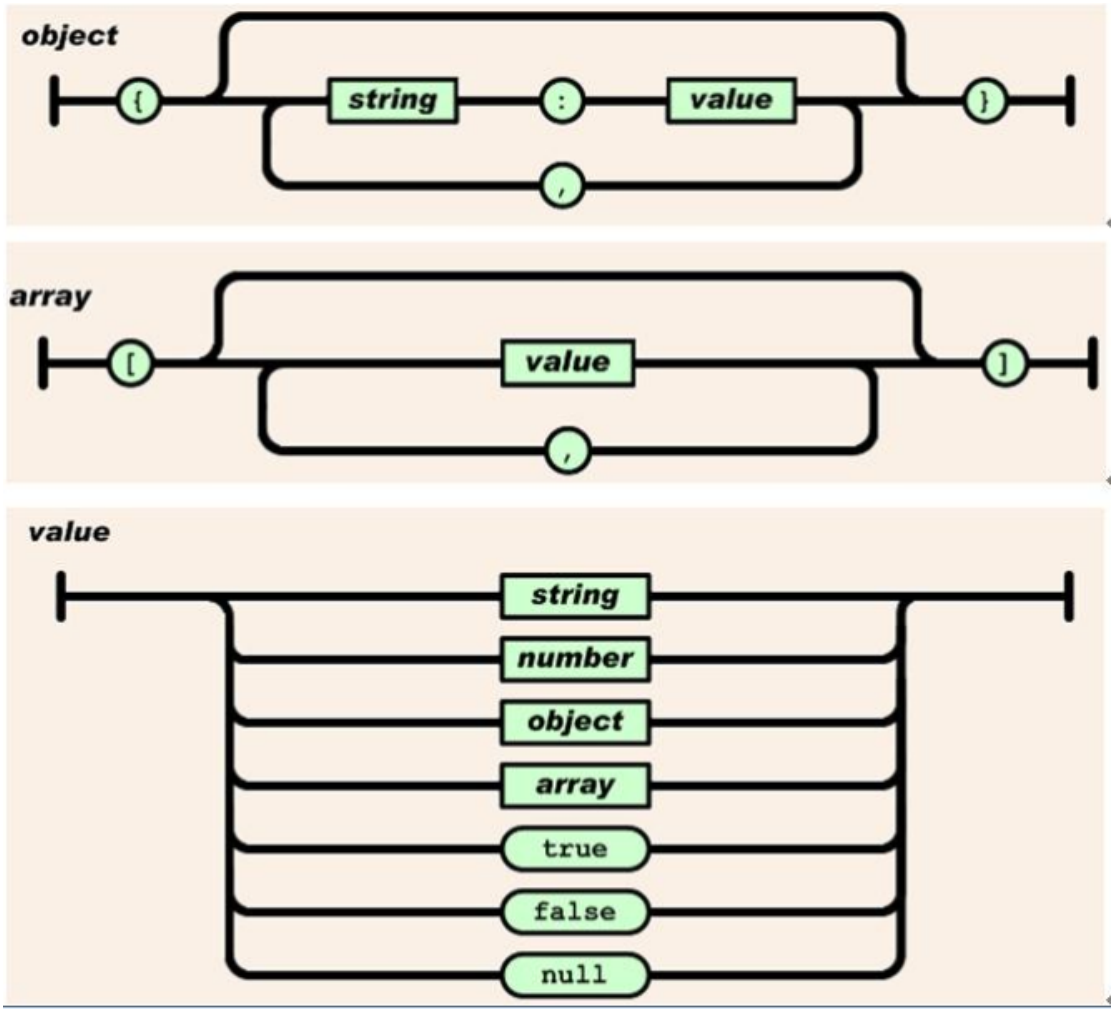
   有 number, string, boolean 和 null 四种类型数据

2. 多个有序的数据: 数组

   用[]包含起来, 其元素可以是三类数据中的任意一种, 元素之间用,号隔开

3. 多个无序的数据: 对象

   用{}包含起来, 其元素必须由 key-value 组成, key 是一个字符串, value 可以是任意类型数据, key 与 value 之间用:号隔开, 两个 key-value 之间用,号隔开.



3.Servlet 相关技术路线(蓝线)

此部分包括: Servlet, Filter, Listener 和 HTTP 协议

3.1. 解释一下什么是 servlet?

答: 我们可以从下面二个方面去看 Servlet:

1. API: 有一个接口 Servlet, 它是 Servlet 规范中定义的用来处理客户端请求的程序需要实现的顶级接口
2. 组件: 服务器端用来处理客户端请求的组件, 需要在 web.xml 请求中配置

3.2. 说一说 Servlet 的生命周期?

答: Servlet 生命周期分为三个阶段:

1，初始化阶段 调用 init()方法

2，响应客户请求阶段 调用 service()方法-→doGet/doPost()

3，终止阶段 调用 destroy()方法

3.3. 区别请求的转发与重定向?

答: 可以从以下三个方面进行比较

1.地址栏:

转发: 显示的是请求的 URL

重定向: 显示的不是请求的 URL, 而是重定向指向的新的 URL

2.浏览器发了几次请求?

转发: 1 次请求

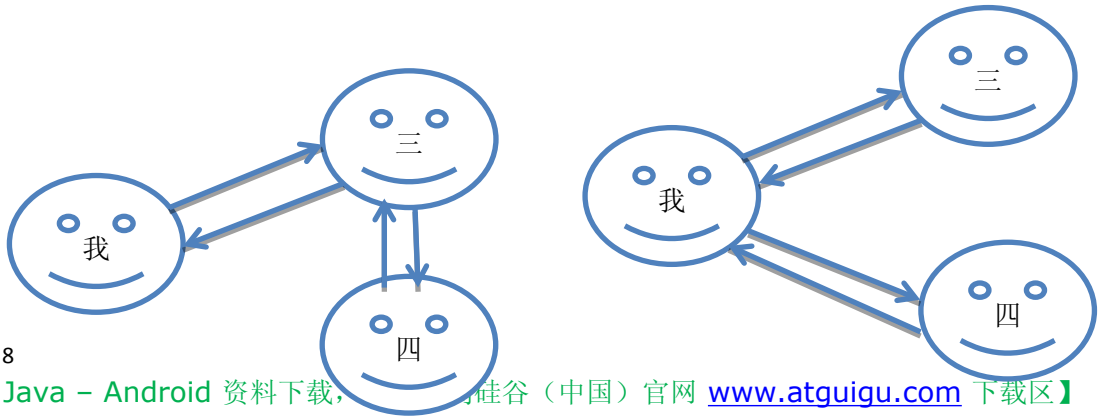
重定向: 2 次请求

3. 是否可以进行 Request 的数据共享?

转发: 两个资源之间是同一个 request 对象, 可以共享 request 中的数据

重定向: 两个资源之间不是同一个 request 对象, 不可以共享

经典现实案例:



3.4. HTTP 请求的 GET 与 POST 方式的区别

答: 可以从以下几个方面去回答:

1. 携带请求参数的方式

   GET: 通过请求行携带参数, 参数会显示在地址栏

   POST: 通过请求体来携带参数, 参数不会显示在地址栏

2. 服务器端处理请求的方法

   GET: 会调用 Servlet 的 doGet()来处理请求

   POST: 会调用 Servlet 的 doPost()来处理请求

3. 数据大小与安全性

   GET: 大小有限制(小于 2k), 不安全

   POST: 大小没有限制, 安全

3.5. 比较一下 Servlet 与 Filter

1. Filter 是一种特别的 Servlet, 它们的作用是完全不一样的. Servlet 是用来处理请求的, 而 Filter 是用来过滤检查请求的.

经典现实案例:

假如我们要去坐地铁去天安门, 我们需要先在检票机上刷票后才能进站坐上地铁, 请求问: 在这个实际业务中, 哪个是 Servlet?哪个是 Filter 呢?

4. Jsp 相关技术路线(红线)

此部分包括: JSP, EL, JSTL, My Tag, I18N, FileUpDown

3.1. jsp 有哪些内置对象?作用分别是什么?

答:JSP 共有以下 9 个内置的对象：

request: 用户端请求，此请求会包含来自 GET/POST 请求的参数

response: 网页传回用户端的回应

pageContext: 网页的属性是在这里管理

session: 与请求有关的会话期

application: 与当前应用对应的 ServletContext 对象, 应用中只有一个

out: 用来传送回应的输出 {}<%=%>

config: 与 jsp 配置对象的对象, 一般无用

page: jsp 对应的 Servlet 对象

exception: 针对错误网页，未捕捉的异常对象

3.2. jsp 有哪些动作?作用分别是什么?

答:JSP 共有以下 6 种基本动作

jsp:include：在页面被请求的时候引入一个文件。

jsp:forward：把请求转到一个新的页面。

jsp:useBean：寻找或者实例化一个 JavaBean。

jsp:setProperty：设置 JavaBean 的属性。

jsp:getProperty：输出某个 JavaBean 的属性。

jsp:plugin：根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记

3.3. JSP 的常用指令

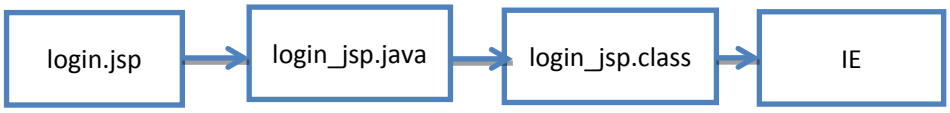
答:主要有下面 3 种指令

1. page 指令: 指定页面的的一些属性, 常用属性:

   contentType="text/html; charset=utf-8" //向浏览器端输出数据的编码

   pageEncoding="utf-8" //jsp 文件被编译成 java 文件时所用的编码

   session="true" //是否自动创建 session



2. include 指令: 包含别一个 jsp 页面
3. taglib 指令: 引入一个标签库

3.4. JSP 中动态 INCLUDE 与静态 INCLUDE 的区别?

答:

1. 动态包含: 用<jsp:include>, 包含的动作是在 jsp 对应的 Serlet 处理请求时去执行的,每次请求都会执行.
2. 静态包含: 用 include 指令, 包含的动作是在 jsp 被编译成 java 文件时执行的, 只有第一次请求时执行.

4.5. JSP 和 Servlet 有哪些相同点和不同点，他们之间的联系是什么?

答:

JSP 的优点是擅长于网页制作，生成动态页面比较直观，缺点是不容易跟踪与排错。

Servlet 是纯 Java 语言，擅长于处理流程和业务逻辑，缺点是生成动态网页不直观。

3.5. EL 的功能, 为什么要用 EL?

EL 的功能包括:

1. 从四个域对象中取出属性数据显示

2. 取出请求参数数据显示

为什么要用 EL?

在页面中用 jsp 脚本和 jsp 表达式来获取数据显示比较麻烦

1. 需要条件判断
2. 可能需要强转

3.6. JSTL 的功能, 为什么要用 JSTL?

JSTL 的功能

JSTL 全名为 JavaServer Pages Standard Tag Library, 主要用于基本输入输出、流程控制、循环、XML 文件剖析、数据库查询及国际化和文字格式标准化的应用等

为什么要用 JSTL?

在 jsp 页面做条件判断或循环操作并输出时, 比较费力

3.7. 为什么要用自定义标签?, MyTag 如何实现?

为什么要用?

1. 不想在 Jsp 中编写 java 代码
2. JSTL 标签库不能满足实际项目的需求

自定义标签定义和使用的流程

1. 编写标签处理器类(SimpleTagSupport 的实现类)
   a) 重写 doTag()
2. 编写标签库文件(WEB-INF/xxx.tld)
   a) 整个文件的定义: <short-name> <uri>
   b) 标签的定义: <tag>
3. 在 jsp 页面使用标签:
   a) 导入标签库(xxx.tld/)
   b) 使用标签

5. Web 会话相关技术路线(紫线)

此部分包括: Cookie 和 Session 技术

5.1. 说说你对 Cookie 与 Session 技术的理解?

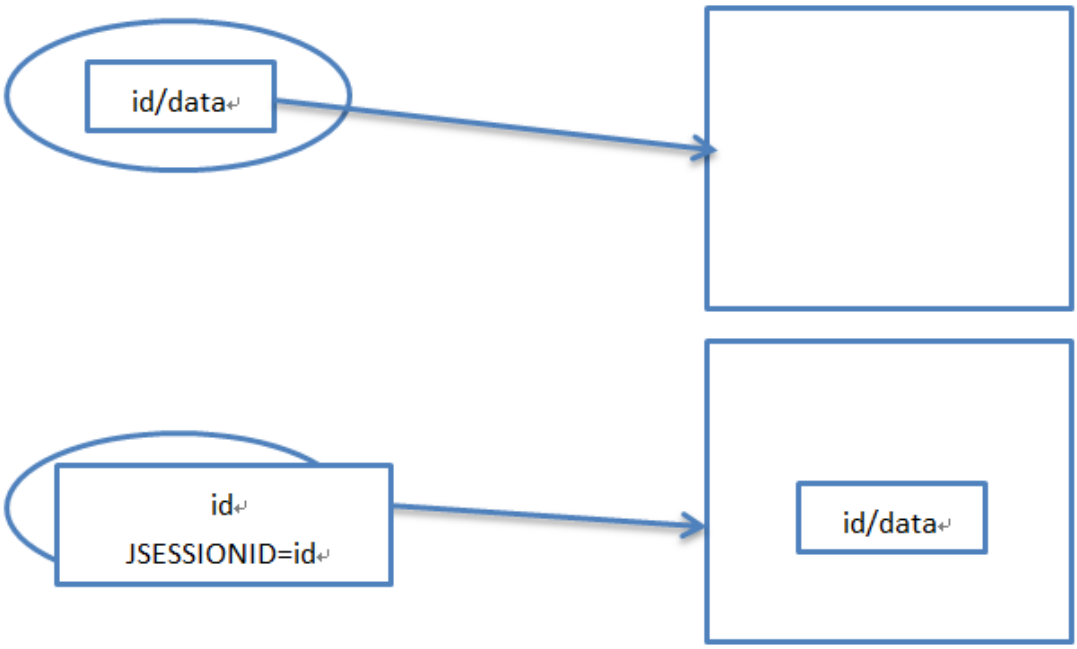
1. cookie 是一种浏览器端的缓存技术, 而 Session 是一种服务器端的缓存技术(依赖 cookie)

经典现实案例:

某咖啡厅推出了一个优惠活动：累计喝五杯咖啡可以免费赠送一杯。他们该如何实现呢？

方法一: 咖啡厅办卡(id,count), 交给消费者, 消费者下次再来消费时, 必须带上卡, 消费一次由咖啡厅来更新卡上的数据, 再次交给消费者

方法二: 咖啡厅办卡(id), id 和 count 都保存在咖啡厅的电脑中的表中, 将卡(id)交给消费者;消费者下次再来消费时, 必须带上卡, 消费一次由咖啡厅来更新表中的数据, 再次交给消费者



5.2. 说说自动登陆功能的编码实现?

1. 登陆功能是用 Session 实现的,就是向 Session 对象中保存当前用户的对象
2. 自动的功能用 Cookie 实现, 就是登陆时将用户的信息保存为持久化 Cookie
3. 下次访问时, 读取请求中如果有用户信息的 Cookie 就可以自动登陆

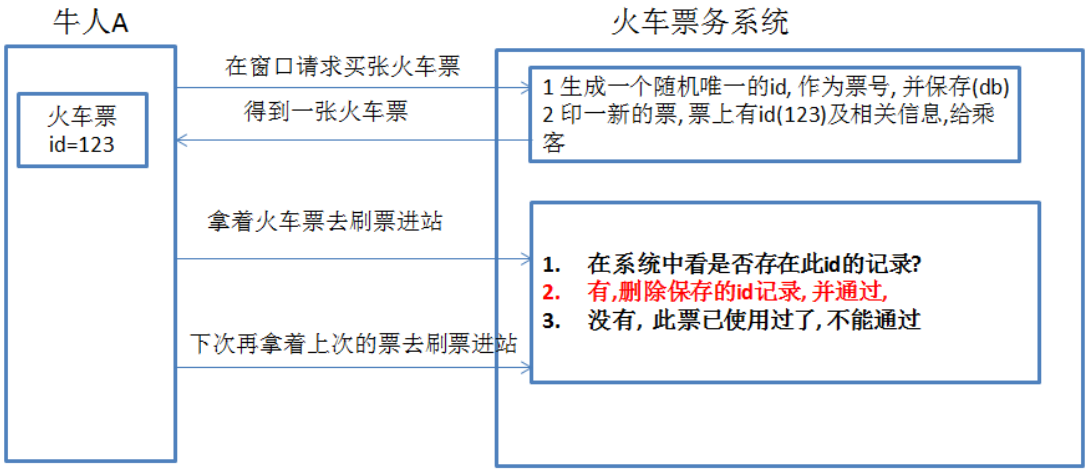
5.3. 如何防止表单重复提交?

答: 使用 Session 技术:

1. 在 regist.jsp 页面中生成一个唯一随机值，将其保存到 Session 中，同时将其保存为表单的隐藏域的值
2. 在处理注册的请求时,获取 Session 中值,获取请求参数的值,比较两者是否相同, 如果相同说明不是重复提交,请求通过同时删除 session 中保存的值, 如果不相同则是重复提交, 不能通过.

经典现实案例:

一位乘客在北京火车站买了一张去天津的火车票(直接刷的那种),他刷票进站坐火车去了天津，回来后过了几天，他又需要去天津这次他不想再买票，直接拿上次的票去进站口刷，检票机提示“此火车票已使用过了”, 不能进站.



6. 其它

此部分包括: MVC, WebService 和 Mybatis

6.1. MVC 的各个部分都有那些技术来实现?如何实现?

答: MVC 是 Model—View—Controller 的简写。

Model 代表的是应用的业务逻辑（通过 JavaBean，EJB 组件实现），

View 是应用的表示面（由 JSP 页面产生），

Controller 是提供应用的处理过程控制（一般是一个 Servlet），

通过这种设计模型把应用逻辑，处理过程和显示逻辑分成不同的组件实现。这些组件可以进行交互和重用。

6.2. WEB SERVICE 相关名词解释

Web Service

Web Service 是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得 Web Service 能与其他兼容的组件进行互操作。

JAXM(Java API for XML Messaging)

是为 SOAP 通信提供访问方法和传输机制的 API。

WSDL:

是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。这种格式首先对操作和消息进行抽象描述，然后将其绑定到具体的网络协议和消息格式上以定义端点。相关的具体端点即组合成为抽象端点（服务）。

SOAP:

即简单对象访问协议(Simple Object Access Protocol)，它是用于交换 XML 编码信息的轻量级协议。

6.3. WebService 技术的本质是使用哪几种技术实现的?

HTTP + XML + Schema

6.4. 如何编码发布一个 WebService?

1. 定义 SEI: 使用@Webservice 和@Webmethod
2. 定义 SEI 的实现类: 使用@Webservice
3. 发布:使用 JDK 中的 Endpoint,或者使用 CXF 框架基于 Spring 的配置来发布

6.5. 如何编码请求一个 WebService?

1. 根据 wsdl 文档生成客户端代码.
2. 利用客户端代码编写调用 webservice 的代码.

6.6. 比较一下 JDBC, dbutils, Mybatis 和 Hibernate

1. JDBC: 原生访问数据库的方式, 其它三个都是对 JDBC 不同程度的包装

访问数据库比较麻烦, 代码重复度极高

2. dbutils: 是对 jdbc 进行了相对简单的包装, 主要就是能自动封装查询结构集, 需要在代码中写 sql 语句
3. Mybatis: 进一步封装 jdbc, Sql 语句写在配置文件中, 面向对象操作, 有一二级缓存功能
4. Hibernate: 对 jdbc 封装得最彻底的框架, 纯面向对象, 可以不用写 SQL

题目：115 个 Java 面试题和答案——终极（上）

本文我们将要讨论 Java 面试中的各种不同类型的面试题，它们可以让雇主测试应聘者的 Java 和通用的面向对象编程的能力。下面的章节分为上下两篇，第一篇将要讨论面向对象编程和它的特点，关于 Java 和它的功能的常见问题，Java 的集合类，垃圾收集器，第二篇主要讨论异常处理，Java 小应用程序，Swing，JDBC，远程方法调用(RMI)，Servlet 和 JSP。

目录

面向对象编程（OOP）

Java 是一个支持并发、基于类和面向对象的计算机编程语言。下面列出了面向对象软件开发的优点：

代码开发模块化，更易维护和修改。
代码复用。
增强代码的可靠性和灵活性。
增加代码的可理解性。
面向对象编程有很多重要的特性，比如：封装，继承，多态和抽象。下面的章节我们会逐个分析这些特性。

封装

封装给对象提供了隐藏内部特性和行为的能力。对象提供一些能被其他对象访问的方法来改变它内部的数据。在 Java 当中，有 3 种修饰符：public，private 和 protected。每一种修饰符给其他的位于同一个包或者不同包下面对象赋予了不同的访问权限。

下面列出了使用封装的一些好处：

通过隐藏对象的属性来保护对象内部的状态。
提高了代码的可用性和可维护性，因为对象的行为可以被单独的改变或者是扩展。
禁止对象之间的不良交互提高模块化。

参考这个文档获取更多关于封装的细节和示例。

多态

多态是编程语言给不同的底层数据类型做相同的接口展示的一种能力。一个多态类型上的操作可以应用到其他类型的值上面。

继承

继承给对象提供了从基类获取字段和方法的能力。继承提供了代码的重用行，也可以在不修改类的情况下给现存的类添加新特性。

抽象

抽象是把想法从具体的实例中分离出来的步骤，因此，要根据他们的功能而不是实现细节来创建类。Java 支持创建只暴漏接口而不包含方法实现的抽象的类。这种抽象技术的主要目的是把类的行为和实现细节分离开。

抽象和封装的不同点

抽象和封装是互补的概念。一方面，抽象关注对象的行为。另一方面，封装关注对象行为的细节。一般是通过隐藏对象内部状态信息做到封装，因此，封装可以看成是用来提供抽象的一种策略。

常见的 Java 问题

1.什么是 Java 虚拟机？为什么 Java 被称作是“平台无关的编程语言”？

Java 虚拟机是一个可以执行 Java 字节码的虚拟机进程。Java 源文件被编译成能被 Java 虚拟机执行的字节码文件。

Java 被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台单独重写或者是重新编译。Java 虚拟机让这个变为可能，因为它知道底层硬件平台的指令长度和其他特性。

2.JDK 和 JRE 的区别是什么？

Java 运行时环境(JRE)是将要执行 Java 程序的 Java 虚拟机。它同时也包含了执行 applet 需要的浏览器插件。Java 开发工具包(JDK)是完整的 Java 软件开发包，包含了 JRE，编译器和其他的工具(比如：JavaDoc，Java 调试器)，可以让开发者开发、编译、执行 Java 应用程序。

3.”static”关键字是什么意思？Java 中是否可以覆盖(override)一个 private 或者是 static 的方法？

“static”关键字表明一个成员变量或者是成员方法可以在没有所属的类的实例变量的情况下被访问。
Java 中 static 方法不能被覆盖，因为方法覆盖是基于运行时动态绑定的，而 static 方法是编译时静态绑定的。static 方法跟类的任何实例都不相关，所以概念上不适用。

4.是否可以在 static 环境中访问非 static 变量？

static 变量在 Java 中是属于类的，它在所有的实例中的值是一样的。当类被 Java 虚拟机载入的时候，会对 static 变量进行初始化。如果你的代码尝试不用实例来访问非 static 的变量，编译器会报错，因为这些变量还没有被创建出来，还没有跟任何实例关联上。

5.Java 支持的数据类型有哪些？什么是自动拆装箱？

Java 语言支持的 8 中基本数据类型是：

byte
short
int
long
float
double
boolean
char
自动装箱是 Java 编译器在基本数据类型和对应的对象包装类型之间做的一个转化。比如：把 int 转化成 Integer，double 转化成 double，等等。反之就是自动拆箱。

6.Java 中的方法覆盖(Overriding)和方法重载(Overloading)是什么意思？

Java 中的方法重载发生在同一个类里面两个或者是多个方法的方法名相同但是参数不同的情况。与此相对，方法覆盖是说子类重新定义了父类的方法。方法覆盖必须有相同的方法名，参数列表和返回类型。覆盖者可能不会限制它所覆盖的方法的访问。

7.Java 中，什么是构造函数？什么是构造函数重载？什么是复制构造函数？

当新对象被创建的时候，构造函数会被调用。每一个类都有构造函数。在程序员没有给类提供构造函数的情况下，Java 编译器会为这个类创建一个默认的构造函数。

Java 中构造函数重载和方法重载很相似。可以为一个类创建多个构造函数。每一个构造函数必须有它自己唯一的参数列表。

Java 不支持像 C++中那样的复制构造函数，这个不同点是因为如果你不自己写构造函数的情况下，Java 不会创建默认的复制构造函数。

8.Java 支持多继承么？

不支持，Java 不支持多继承。每个类都只能继承一个类，但是可以实现多个接口。

9.接口和抽象类的区别是什么？

Java 提供和支持创建抽象类和接口。它们的实现有共同点，不同点在于：

接口中所有的方法隐含的都是抽象的。而抽象类则可以同时包含抽象和非抽象的方法。
类可以实现很多个接口，但是只能继承一个抽象类
类如果要实现一个接口，它必须要实现接口声明的所有方法。但是，类可以不实现抽象类声明的所有方法，当然，在这种情况下，类也必须得声明成是抽象的。
抽象类可以在不提供接口方法实现的情况下实现接口。
Java 接口中声明的变量默认都是 final 的。抽象类可以包含非 final 的变量。
Java 接口中的成员函数默认是 public 的。抽象类的成员函数可以是 private，protected 或者是 public。
接口是绝对抽象的，不可以被实例化。抽象类也不可以被实例化，但是，如果它包含 main 方法的话是可以被调用的。
也可以参考 JDK8 中抽象类和接口的区别

10.什么是值传递和引用传递？

对象被值传递，意味着传递了对象的一个副本。因此，就算是改变了对象副本，也不会影响源对象的值。

对象被引用传递，意味着传递的并不是实际的对象，而是对象的引用。因此，外部对引用对象所做的改变会反映到所有的对象上。

Java 线程

11.进程和线程的区别是什么？

进程是执行着的应用程序，而线程是进程内部的一个执行序列。一个进程可以有多个线程。线程又叫做轻量级进程。

12.创建线程有几种不同的方式？你喜欢哪一种？为什么？

有三种方式可以用来创建线程：

继承 Thread 类
实现 Runnable 接口
应用程序可以使用 Executor 框架来创建线程池
实现 Runnable 接口这种方式更受欢迎，因为这不需要继承 Thread 类。在应用设计中已经继承了别的对象的情况下，这需要多继承（而 Java 不支持多继承），只能实现接口。同时，线程池也是非常高效的，很容易实现和使用。

13.概括的解释下线程的几种可用状态。

线程在执行过程中，可以处于下面几种状态：

就绪(Runnable):线程准备运行，不一定立马就能开始执行。
运行中(Running)：进程正在执行线程的代码。
等待中(Waiting):线程处于阻塞的状态，等待外部的处理结束。
睡眠中(Sleeping)：线程被强制睡眠。
I/O 阻塞(Blocked on I/O)：等待 I/O 操作完成。
同步阻塞(Blocked on Synchronization)：等待获取锁。
死亡(Dead)：线程完成了执行。

14.同步方法和同步代码块的区别是什么？

在 Java 语言中，每一个对象有一把锁。线程可以使用 synchronized 关键字来获取对象上的锁。synchronized 关键字可应用在方法级别(粗粒度锁)或者是代码块级别(细粒度锁)。

15.在监视器(Monitor)内部，是如何做线程同步的？程序应该做哪种级别的同步？

监视器和锁在 Java 虚拟机中是一块使用的。监视器监视一块同步代码块，确保一次只有一个线程执行同步代码块。每一个监视器都和一个对象引用相关联。线程在获取锁之前不允许执行同步代码。

16.什么是死锁(deadlock)？

两个进程都在等待对方执行完毕才能继续往下执行的时候就发生了死锁。结果就是两个进程都陷入了无限的等待中。

17.如何确保 N 个线程可以访问 N 个资源同时又不导致死锁？

使用多线程的时候，一种非常简单的避免死锁的方式就是：指定获取锁的顺序，并强制线程按照指定的顺序获取锁。因此，如果所有的线程都是以同样的顺序加锁和释放锁，就不会出现死锁了。

Java 集合类

18.Java 集合类框架的基本接口有哪些？

Java 集合类提供了一套设计良好的支持对一组对象进行操作的接口和类。Java 集合类里面最基本的接口有：

Collection：代表一组对象，每一个对象都是它的子元素。
Set：不包含重复元素的 Collection。

List：有顺序的 collection，并且可以包含重复元素。
Map：可以把键(key)映射到值(value)的对象，键不能重复。

19.为什么集合类没有实现 Cloneable 和 Serializable 接口？

集合类接口指定了一组叫做元素的对象。集合类接口的每一种具体的实现类都可以选择以它自己的方式对元素进行保存和排序。有的集合类允许重复的键，有些不允许。

20.什么是迭代器(Iterator)？

Iterator 接口提供了很多对集合元素进行迭代的方法。每一个集合类都包含了可以返回迭代器实例的
迭代方法。迭代器可以在迭代的过程中删除底层集合的元素。

克隆(cloning)或者是序列化(serialization)的语义和含义是跟具体的实现相关的。因此，应该由集合类的具体实现来决定如何被克隆或者是序列化。

21.Iterator 和 ListIterator 的区别是什么？

下面列出了他们的区别：

Iterator 可用来遍历 Set 和 List 集合，但是 ListIterator 只能用来遍历 List。
Iterator 对集合只能是前向遍历，ListIterator 既可以前向也可以后向。
ListIterator 实现了 Iterator 接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引，等等。

22.快速失败(fail-fast)和安全失败(fail-safe)的区别是什么？

Iterator 的安全失败是基于对底层集合做拷贝，因此，它不受源集合上修改的影响。java.util 包下面的所有的集合类都是快速失败的，而 java.util.concurrent 包下面的所有的类都是安全失败的。快速失败的迭代器会抛出 ConcurrentModificationException 异常，而安全失败的迭代器永远不会抛出这样的异常。

23.Java 中的 HashMap 的工作原理是什么？

Java 中的 HashMap 是以键值对(key-value)的形式存储元素的。HashMap 需要一个 hash 函数，它使用 hashCode()和 equals()方法来向集合/从集合添加和检索元素。当调用 put()方法的时候，HashMap 会计算 key 的 hash 值，然后把键值对存储在集合中合适的索引上。如果 key 已经存在了，value 会被更新成新值。HashMap 的一些重要的特性是它的容量(capacity)，负载因子(load factor)和扩容极限(threshold resizing)。

24.hashCode()和 equals()方法的重要性体现在什么地方？

Java 中的 HashMap 使用 hashCode()和 equals()方法来确定键值对的索引，当根据键获取值的

时候也会用到这两个方法。如果没有正确的实现这两个方法，两个不同的键可能会有相同的 hash 值，因此，可能会被集合认为是相等的。而且，这两个方法也用来发现重复元素。所以这两个方法的实现对 HashMap 的精确性和正确性是至关重要的。

25.HashMap 和 Hashtable 有什么区别？

HashMap 和 Hashtable 都实现了 Map 接口，因此很多特性非常相似。但是，他们有以下不同点：
HashMap 允许键和值是 null，而 Hashtable 不允许键或者值是 null。
Hashtable 是同步的，而 HashMap 不是。因此，HashMap 更适合于单线程环境，而 Hashtable 适合于多线程环境。
HashMap 提供了可供应用迭代的键的集合，因此，HashMap 是快速失败的。另一方面，Hashtable 提供了对键的列举(Enumeration)。
一般认为 Hashtable 是一个遗留的类。

26.数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用 Array 而不是 ArrayList？

下面列出了 Array 和 ArrayList 的不同点：

Array 可以包含基本类型和对象类型，ArrayList 只能包含对象类型。
Array 大小是固定的，ArrayList 的大小是动态变化的。
ArrayList 提供了更多的方法和特性，比如：addAll()，removeAll()，iterator()等等。
对于基本类型数据，集合使用自动装箱来减少编码工作量。但是，当处理固定大小的基本数据类型的时候，这种方式相对比较慢。

27.ArrayList 和 LinkedList 有什么区别？

ArrayList 和 LinkedList 都实现了 List 接口，他们有以下的不同点：

ArrayList 是基于索引的数据接口，它的底层是数组。它可以以 O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList 是以元素列表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是 O(n)。
相对于 ArrayList，LinkedList 的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。
LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。
也可以参考 ArrayList vs. LinkedList。

28.Comparable 和 Comparator 接口是干什么的？列出它们的区别。

Java 提供了只包含一个 compareTo()方法的 Comparable 接口。这个方法可以个给两个对象排序。具体来说，它返回负数，0，正数来表明输入对象小于，等于，大于已经存在的对象。

Java 提供了包含 compare()和 equals()两个方法的 Comparator 接口。compare()方法用来给两

个输入参数排序，返回负数，0，正数表明第一个参数是小于，等于，大于第二个参数。equals()方法需要一个对象作为参数，它用来决定输入参数是否和 comparator 相等。只有当输入参数也是一个 comparator 并且输入参数和当前 comparator 的排序结果是相同的时候，这个方法才返回 true。

29.什么是 Java 优先级队列(Priority Queue)？

PriorityQueue 是一个基于优先级堆的无界队列，它的元素是按照自然顺序(natural order)排序的。在创建的时候，我们可以给它提供一个负责给元素排序的比较器。PriorityQueue 不允许 null 值，因为他们没有自然顺序，或者说他们没有任何的相关联的比较器。最后，PriorityQueue 不是线程安全的，入队和出队的时间复杂度是 O(log(n))。

30.你了解大 O 符号(big-O notation)么？你能给出不同数据结构的例子么？

大 O 符号描述了当数据结构里面的元素增加的时候，算法的规模或者是性能在最坏的场景下有多么好。
大 O 符号也可用来描述其他的行为，比如：内存消耗。因为集合类实际上是数据结构，我们一般使用大 O 符号基于时间，内存和性能来选择最好的实现。大 O 符号可以对大量数据的性能给出一个很好的说明。

31.如何权衡是使用无序的数组还是有序的数组？

有序数组最大的好处在于查找的时间复杂度是 O(log n)，而无序数组是 O(n)。有序数组的缺点是插入操作的时间复杂度是 O(n)，因为值大的元素需要往后移动来给新元素腾位置。相反，无序数组的插入时间复杂度是常量 O(1)。

32.Java 集合类框架的最佳实践有哪些？

根据应用的需要正确选择要使用的集合的类型对性能非常重要，比如：假如元素的大小是固定的，而且能事先知道，我们就应该用 Array 而不是 ArrayList。
有些集合类允许指定初始容量。因此，如果我们能估计出存储的元素的数目，我们可以设置初始容量来避免重新计算 hash 值或者是扩容。
为了类型安全，可读性和健壮性的原因总是要使用泛型。同时，使用泛型还可以避免运行时的 ClassCastException。
使用 JDK 提供的不变类(immutable class)作为 Map 的键可以避免为我们自己的类实现 hashCode()和 equals()方法。
编程的时候接口优于实现。
底层的集合实际上是空的情况下，返回长度是 0 的集合或者是数组，不要返回 null。

33.Enumeration 接口和 Iterator 接口的区别有哪些？

Enumeration 速度是 Iterator 的 2 倍，同时占用更少的内存。但是，Iterator 远远比 Enumeration 安全，因为其他线程不能够修改正在被 iterator 遍历的集合里面的对象。同时，Iterator 允许调用者删除底层集合里面的元素，这对 Enumeration 来说是不可能的。

34.HashSet 和 TreeSet 有什么区别？

HashSet 是由一个 hash 表来实现的，因此，它的元素是无序的。add()，remove()，contains()方法的时间复杂度是 O(1)。

另一方面，TreeSet 是由一个树形的结构来实现的，它里面的元素是有序的。因此，add()，remove()，contains()方法的时间复杂度是 O(logn)。

垃圾收集器(Garbage Collectors)

35.Java 中垃圾回收有什么目的？什么时候进行垃圾回收？

垃圾回收的目的是识别并且丢弃应用不再使用的对象来释放和重用资源。

36.System.gc()和 Runtime.gc()会做什么事情？

这两个方法用来提示 JVM 要进行垃圾回收。但是，立即开始还是延迟进行垃圾回收是取决于 JVM 的。

37.finalize()方法什么时候被调用？析构函数(finalization)的目的是什么？

在释放对象占用的内存之前，垃圾收集器会调用对象的 finalize()方法。一般建议在该方法中释放对象持有的资源。

38.如果对象的引用被置为 null，垃圾收集器是否会立即释放对象占用的内存？

不会，在下一个垃圾回收周期中，这个对象将是可被回收的。

39.Java 堆的结构是什么样子的？什么是堆中的永久代(Perm Gen space)?

JVM 的堆是运行时数据区，所有类的实例和数组都是在堆上分配内存。它在 JVM 启动的时候被创建。对象所占的堆内存是由自动内存管理系统也就是垃圾收集器回收。

堆内存是由存活和死亡的对象组成的。存活的对象是应用可以访问的，不会被垃圾回收。死亡的对象是应用不可访问尚且还没有被垃圾收集器回收掉的对象。一直到垃圾收集器把这些对象回收掉之前，他们会一直占据堆内存空间。

40.串行(serial)收集器和吞吐量(throughput)收集器的区别是什么？

吞吐量收集器使用并行版本的新生代垃圾收集器，它用于中等规模和大规模数据的应用程序。而串行收集器对大多数的小应用(在现代处理器上需要大概 100M 左右的内存)就足够了。

41.在 Java 中，对象什么时候可以被垃圾回收？

当对象对当前使用这个对象的应用程序变得不可触及的时候，这个对象就可以被回收了。

42.JVM 的永久代中会发生垃圾回收么？

垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值，会触发完全垃圾回收(Full GC)。如果你仔细查看垃圾收集器的输出信息，就会发现永久代也是被回收的。这就是为什么正确的永久代大小对避免 Full GC 是非常重要的原因。请参考下 Java8：从永久代到元数据区
(译者注：Java8 中已经移除了永久代，新加了一个叫做元数据区的 native 内存区)

题目：115 个 Java 面试题和答案——终极（下）

第一篇讨论了面向对象编程和它的特点，关于 Java 和它的功能的常见问题，Java 的集合类，垃圾收集器，本章主要讨论异常处理，Java 小应用程序，Swing，JDBC，远程方法调用(RMI)，Servlet 和 JSP。

异常处理

Java 小应用程序(Applet)

Swing

JDBC

远程方法调用（RMI）

Servlet

JSP

异常处理

43.Java 中的两种异常类型是什么？他们有什么区别？

Java 中有两种异常：受检查的(checked)异常和不受检查的(unchecked)异常。不受检查的异常不需要在方法或者是构造函数上声明，就算方法或者是构造函数的执行可能会抛出这样的异常，并且不受检查的异常可以传播到方法或者是构造函数的外面。相反，受检查的异常必须要用 throws 语句在方法或者是构造函数上声明。这里有 Java 异常处理的一些小建议。

44.Java 中 Exception 和 Error 有什么区别？

Exception 和 Error 都是 Throwable 的子类。Exception 用于用户程序可以捕获的异常情况。Error 定义了不期望被用户程序捕获的异常。

45.1 throw 和 throws 有什么区别？

throw 关键字用来在程序中明确的抛出异常，相反，throws 语句用来表明方法不能处理的异常。每一个方法都必须要指定哪些异常不能处理，所以方法的调用者才能够确保处理可能发生的异常，多个异常是用逗号分隔的。

45.2 异常处理的时候，finally 代码块的重要性是什么？

无论是否抛出异常，finally 代码块总是会被执行。就算是没有 catch 语句同时又抛出异常的

情况下，finally 代码块仍然会被执行。最后要说的是，finally 代码块主要用来释放资源，比如：I/O 缓冲区，数据库连接。

46.异常处理完成以后，Exception 对象会发生什么变化？

Exception 对象会在下一个垃圾回收过程中被回收掉。

47.finally 代码块和 finalize()方法有什么区别？

无论是否抛出异常，finally 代码块都会执行，它主要是用来释放应用占用的资源。finalize()方法是 Object 类的一个 protected 方法，它是在对象被垃圾回收之前由 Java 虚拟机来调用的。

Java 小应用程序(Applet)

48.什么是 Applet？

java applet 是能够被包含在 HTML 页面中并且能被启用了 java 的客户端浏览器执行的程序。Applet 主要用来创建动态交互的 web 应用程序。

49.解释一下 Applet 的生命周期

applet 可以经历下面的状态：

Init：每次被载入的时候都会被初始化。
Start：开始执行 applet。
Stop：结束执行 applet。
Destroy：卸载 applet 之前，做最后的清理工作。

50.当 applet 被载入的时候会发生什么？

首先，创建 applet 控制类的实例，然后初始化 applet，最后开始运行。

51.Applet 和普通的 Java 应用程序有什么区别？

applet 是运行在启用了 java 的浏览器中，Java 应用程序是可以在浏览器之外运行的独立的 Java 程序。但是，它们都需要有 Java 虚拟机。

进一步来说，Java 应用程序需要一个有特定方法签名的 main 函数来开始执行。Java applet 不需要这样的函数来开始执行。

最后，Java applet 一般会使用很严格的安全策略，Java 应用一般使用比较宽松的安全策略。

52.Java applet 有哪些限制条件？

主要是由于安全的原因，给 applet 施加了以下的限制：

applet 不能够载入类库或者定义本地方法。
applet 不能在宿主机上读写文件。
applet 不能读取特定的系统属性。
applet 不能发起网络连接，除非是跟宿主机。
applet 不能够开启宿主机上其他任何的程序。

53.什么是不受信任的 applet？

不受信任的 applet 是不能访问或是执行本地系统文件的 Java applet，默认情况下，所有下载的 applet 都是不受信任的。

54.从网络上加载的 applet 和从本地文件系统加载的 applet 有什么区别？

当 applet 是从网络上加载的时候，applet 是由 applet 类加载器载入的，它受 applet 安全管理器的限制。

当 applet 是从客户端的本地磁盘载入的时候，applet 是由文件系统加载器载入的。

从文件系统载入的 applet 允许在客户端读文件，写文件，加载类库，并且也允许执行其他程序，但是，却通不过字节码校验。

55.applet 类加载器是什么？它会做哪些工作？

当 applet 是从网络上加载的时候，它是由 applet 类加载器载入的。类加载器有自己的 java 名称空间等级结构。类加载器会保证来自文件系统的类有唯一的名称空间，来自网络资源的类有唯一的名称空间。

当浏览器通过网络载入 applet 的时候，applet 的类被放置于和 applet 的源相关联的私有的名称空间中。然后，那些被类加载器载入进来的类都是通过了验证器验证的。验证器会检查类文件格式是否遵守 Java 语言规范，确保不会出现堆栈溢出(stack overflow)或者下溢(underflow)，传递给字节码指令的参数是正确的。

56.applet 安全管理器是什么？它会做哪些工作？

applet 安全管理器是给 applet 施加限制条件的一种机制。浏览器可以只有一个安全管理器。安全管理器在启动的时候被创建，之后不能被替换覆盖或者是扩展。

Swing

57.弹出式选择菜单(Choice)和列表(List)有什么区别

Choice 是以一种紧凑的形式展示的，需要下拉才能看到所有的选项。Choice 中一次只能选中

一个选项。List 同时可以有多个元素可见，支持选中一个或者多个元素。

58.什么是布局管理器？

布局管理器用来在容器中组织组件。

59.滚动条(Scrollbar)和滚动面板(JScrollPane)有什么区别？

Scrollbar 是一个组件，不是容器。而 ScrollPane 是容器。ScrollPane 自己处理滚动事件。

60.哪些 Swing 的方法是线程安全的？

只有 3 个线程安全的方法： repaint(), revalidate(), and invalidate()。

61.说出三种支持重绘(painting)的组件。

Canvas, Frame, Panel,和 Applet 支持重绘。

62.什么是裁剪(clipping)？

限制在一个给定的区域或者形状的绘图操作就做裁剪。

63.MenuItem 和 CheckboxMenuItem 的区别是什么？

CheckboxMenuItem 类继承自 MenuItem 类，支持菜单选项可以选中或者不选中。

64.边缘布局(BorderLayout)里面的元素是如何布局的？

BorderLayout 里面的元素是按照容器的东西南北中进行布局的。

65.网格包布局(GridBagLayout)里面的元素是如何布局的？

GridBagLayout 里面的元素是按照网格进行布局的。不同大小的元素可能会占据网格的多于 1 行或一列。因此，行数和列数可以有不同的大小。

66.Window 和 Frame 有什么区别？

Frame 类继承了 Window 类，它定义了一个可以有菜单栏的主应用窗口。

67.裁剪(clipping)和重绘(repainting)有什么联系？

当窗口被 AWT 重绘线程进行重绘的时候，它会把裁剪区域设置成需要重绘的窗口的区域。

68.事件监听器接口(event-listener interface)和事件适配器(event-adapter)有什么关系？

事件监听器接口定义了对特定的事件，事件处理器必须要实现的方法。事件适配器给事件监听器接口提供了默认的实现。

69.GUI 组件如何来处理它自己的事件？

GUI 组件可以处理它自己的事件，只要它实现相对应的事件监听器接口，并且把自己作为事件监听器。

70.Java 的布局管理器比传统的窗口系统有哪些优势？

Java 使用布局管理器以一种一致的方式在所有的窗口平台上摆放组件。因为布局管理器不会和组件的绝对大小和位置相绑定，所以他们能够适应跨窗口系统的特定平台的不同。

71.Java 的 Swing 组件使用了哪种设计模式？

Java 中的 Swing 组件使用了 MVC(视图-模型-控制器)设计模式。

JDBC

72.什么是 JDBC？

JDBC 是允许用户在不同数据库之间做选择的一个抽象层。JDBC 允许开发者用 JAVA 写数据库应用程序，而不需要关心底层特定数据库的细节。

73.解释下驱动(Driver)在 JDBC 中的角色。

JDBC 驱动提供了特定厂商对 JDBC API 接口类的实现，驱动必须要提供 java.sql 包下面这些类的实现：Connection, Statement, PreparedStatement,CallableStatement, ResultSet 和 Driver。

74.Class.forName()方法有什么作用？

这个方法用来载入跟数据库建立连接的驱动。

75.PreparedStatement 比 Statement 有什么优势？

PreparedStatements 是预编译的，因此，性能会更好。同时，不同的查询参数值，PreparedStatement 可以重用。

76.什么时候使用 CallableStatement？用来准备 CallableStatement 的方法是什么？

CallableStatement 用来执行存储过程。存储过程是由数据库存储和提供的。存储过程可以接受输入参数，也可以有返回结果。非常鼓励使用存储过程，因为它提供了安全性和模块化。准备一个 CallableStatement 的方法是：

CallableStament.prepareCall();

77.数据库连接池是什么意思？

像打开关闭数据库连接这种和数据库的交互可能是很费时的，尤其是当客户端数量增加的时候，会消耗大量的资源，成本是非常高的。可以在应用服务器启动的时候建立很多个数据库连接并维护在一个池中。连接请求由池中的连接提供。在连接使用完毕以后，把连接归还到池中，以用于满足将来更多的请求。

远程方法调用(RMI)

78.什么是 RMI？

Java 远程方法调用(Java RMI)是 Java API 对远程过程调用(RPC)提供的面向对象的等价形式，支持直接传输序列化的 Java 对象和分布式垃圾回收。远程方法调用可以看做是激活远程正在运行的对象上的方法的步骤。RMI 对调用者是位置透明的，因为调用者感觉方法是执行在本地运行的对象上的。看下 RMI 的一些注意事项。

79.RMI 体系结构的基本原则是什么？

RMI 体系结构是基于一个非常重要的行为定义和行为实现相分离的原则。RMI 允许定义行为的代码和实现行为的代码相分离，并且运行在不同的 JVM 上。

80.RMI 体系结构分哪几层？

RMI 体系结构分以下几层：

存根和骨架层(Stub and Skeleton layer)：这一层对程序员是透明的，它主要负责拦截客户端发出的方法调用请求，然后把请求重定向给远程的 RMI 服务。

远程引用层(Remote Reference Layer)：RMI 体系结构的第二层用来解析客户端对服务端远程对象的引用。这一层解析并管理客户端对服务端远程对象的引用。连接是点到点的。

传输层(Transport layer)：这一层负责连接参与服务的两个 JVM。这一层是建立在网络上机器间的 TCP/IP 连接之上的。它提供了基本的连接服务，还有一些防火墙穿透策略。

81.RMI 中的远程接口(Remote Interface)扮演了什么样的角色？

远程接口用来标识哪些方法是可以被非本地虚拟机调用的接口。远程对象必须要直接或者是间接实现远程接口。实现了远程接口的类应该声明被实现的远程接口，给每一个远程对象定义构造函数，给所有远程接口的方法提供实现。

82.java.rmi.Naming 类扮演了什么样的角色？

java.rmi.Naming 类用来存储和获取在远程对象注册表里面的远程对象的引用。Naming 类的

每一个方法接收一个 URL 格式的 String 对象作为它的参数。

83.RMI 的绑定(Binding)是什么意思？

绑定是为了查询找远程对象而给远程对象关联或者是注册以后会用到的名称的过程。远程对象可以使用 Naming 类的 bind()或者 rebind()方法跟名称相关联。

84.Naming 类的 bind()和 rebind()方法有什么区别？

bind()方法负责把指定名称绑定给远程对象，rebind()方法负责把指定名称重新绑定到一个新的远程对象。如果那个名称已经绑定过了，先前的绑定会被替换掉。

85.让 RMI 程序能正确运行有哪些步骤？

为了让 RMI 程序能正确运行必须要包含以下几个步骤：

编译所有的源文件。
使用 rmic 生成 stub。
启动 rmiregistry。
启动 RMI 服务器。
运行客户端程序。

86.RMI 的 stub 扮演了什么样的角色？

远程对象的 stub 扮演了远程对象的代表或者代理的角色。调用者在本地 stub 上调用方法，它负责在远程对象上执行方法。当 stub 的方法被调用的时候，会经历以下几个步骤：

初始化到包含了远程对象的 JVM 的连接。
序列化参数到远程的 JVM。
等待方法调用和执行的结果。
反序列化返回的值或者是方法没有执行成功情况下的异常。
把值返回给调用者。

87.什么是分布式垃圾回收(DGC)？它是如何工作的？

DGC 叫做分布式垃圾回收。RMI 使用 DGC 来做自动垃圾回收。因为 RMI 包含了跨虚拟机的远程对象的引用，垃圾回收是很困难的。DGC 使用引用计数算法来给远程对象提供自动内存管理。

88.RMI 中使用 RMI 安全管理器(RMISecurityManager)的目的是什么？

RMISecurityManager 使用下载好的代码提供可被 RMI 应用程序使用的安全管理器。如果没有设置安全管理器，RMI 的类加载器就不会从远程下载任何的类。

89.解释下 Marshalling 和 demarshalling。

当应用程序希望把内存对象跨网络传递到另一台主机或者是持久化到存储的时候，就必须要把对象在内存里面的表示转化成合适的格式。这个过程就叫做 Marshalling，反之就是 demarshalling。

90.解释下 Serialization 和 Deserialization。

Java 提供了一种叫做对象序列化的机制，他把对象表示成一连串的字节，里面包含了对象的数据，对象的类型信息，对象内部的数据的类型信息等等。因此，序列化可以看成是为了把对象存储在磁盘上或者是从磁盘上读出来并重建对象而把对象扁平化的一种方式。反序列化是把对象从扁平状态转化成活动对象的相反的步骤。

Servlet

91.什么是 Servlet？

Servlet 是用来处理客户端请求并产生动态网页内容的 Java 类。Servlet 主要是用来处理或者是存储 HTML 表单提交的数据，产生动态内容，在无状态的 HTTP 协议下管理状态信息。

92.说一下 Servlet 的体系结构。

所有的 Servlet 都必须要实现的核心的接口是 javax.servlet.Servlet。每一个 Servlet 都必须要直接或者是间接实现这个接口，或者是继承 javax.servlet.GenericServlet 或者 javax.servlet.http.HTTPServlet。最后，Servlet 使用多线程可以并行的为多个请求服务。

93.Applet 和 Servlet 有什么区别？

Applet 是运行在客户端主机的浏览器上的客户端 Java 程序。而 Servlet 是运行在 web 服务器上的服务端的组件。applet 可以使用用户界面类，而 Servlet 没有用户界面，相反，Servlet 是等待客户端的 HTTP 请求，然后为请求产生响应。

94.GenericServlet 和 HttpServlet 有什么区别？

GenericServlet 是一个通用的协议无关的 Servlet，它实现了 Servlet 和 ServletConfig 接口。继承自 GenericServlet 的 Servlet 应该要覆盖 service()方法。最后，为了开发一个能用在网页上服务于使用 HTTP 协议请求的 Servlet，你的 Servlet 必须要继承自 HttpServlet。这里有 Servlet 的例子。

95.解释下 Servlet 的生命周期。

对每一个客户端的请求，Servlet 引擎载入 Servlet，调用它的 init()方法，完成 Servlet 的初始化。然后，Servlet 对象通过为每一个请求单独调用 service()方法来处理所有随后来自客户端的请求，最后，调用 Servlet(译者注：这里应该是 Servlet 而不是 server)的 destroy()方法把 Servlet

删除掉。

96.doGet()方法和 doPost()方法有什么区别？

doGet：GET 方法会把名值对追加在请求的 URL 后面。因为 URL 对字符数目有限制，进而限制了用在客户端请求的参数值的数目。并且请求中的参数值是可见的，因此，敏感信息不能用这种方式传递。

doPOST：POST 方法通过把请求参数值放在请求体中来克服 GET 方法的限制，因此，可以发送的参数的数目是没有限制的。最后，通过 POST 请求传递的敏感信息对外部客户端是不可见的。

97.什么是 Web 应用程序？

Web 应用程序是对 Web 或者是应用服务器的动态扩展。有两种类型的 Web 应用：面向表现的和面向服务的。面向表现的 Web 应用程序会产生包含了很多种标记语言和动态内容的交互的 web 页面作为对请求的响应。而面向服务的 Web 应用实现了 Web 服务的端点(endpoint)。一般来说，一个 Web 应用可以看成是一组安装在服务器 URL 名称空间的特定子集下面的 Servlet 的集合。

98.什么是服务端包含(Server Side Include)？

服务端包含(SSI)是一种简单的解释型服务端脚本语言，大多数时候仅用在 Web 上，用 servlet 标签嵌入进来。SSI 最常用的场景把一个或多个文件包含到 Web 服务器的一个 Web 页面中。当浏览器访问 Web 页面的时候，Web 服务器会用对应的 servlet 产生的文本来替换 Web 页面中的 servlet 标签。

99.什么是 Servlet 链(Servlet Chaining)？

Servlet 链是把一个 Servlet 的输出发送给另一个 Servlet 的方法。第二个 Servlet 的输出可以发送给第三个 Servlet，依次类推。链条上最后一个 Servlet 负责把响应发送给客户端。

100.如何知道是哪一个客户端的机器正在请求你的 Servlet？

ServletRequest 类可以找出客户端机器的 IP 地址或者是主机名。getRemoteAddr()方法获取客户端主机的 IP 地址，getRemoteHost()可以获取主机名。看下这里的例子。

101.HTTP 响应的结构是怎么样的？

HTTP 响应由三个部分组成：

状态码(Status Code)：描述了响应的状态。可以用来检查是否成功的完成了请求。请求失败的情况下，状态码可用来找出失败的原因。如果 Servlet 没有返回状态码，默认会返回成功的状态码 HttpServletResponse.SC_OK。

HTTP 头部(HTTP Header)：它们包含了更多关于响应的信息。比如：头部可以指定认为响应过期的过期日期，或者是指定用来给用户安全的传输实体内容的编码格式。如何在 Serlet 中检索 HTTP 的头部看这里。

主体(Body)：它包含了响应的内容。它可以包含 HTML 代码，图片，等等。主体是由传输在 HTTP 消息中紧跟在头部后面的数据字节组成的。

102.什么是 cookie？session 和 cookie 有什么区别？

cookie 是 Web 服务器发送给浏览器的一块信息。浏览器会在本地文件中给每一个 Web 服务器存储 cookie。以后浏览器在给特定的 Web 服务器发请求的时候，同时会发送所有为该服务器存储的 cookie。下面列出了 session 和 cookie 的区别：

无论客户端浏览器做怎么样的设置，session 都应该能正常工作。客户端可以选择禁用 cookie，但是，session 仍然是能够工作的，因为客户端无法禁用服务端的 session。
在存储的数据量方面 session 和 cookies 也是不一样的。session 能够存储任意的 Java 对象，cookie 只能存储 String 类型的对象。

103.浏览器和 Servlet 通信使用的是什么协议？
浏览器和 Servlet 通信使用的是 HTTP 协议。

104.什么是 HTTP 隧道？

HTTP 隧道是一种利用 HTTP 或者是 HTTPS 把多种网络协议封装起来进行通信的技术。因此，HTTP 协议扮演了一个打通用于通信的网络协议的管道的包装器的角色。把其他协议的请求掩盖成 HTTP 的请求就是 HTTP 隧道。

105.sendRedirect()和 forward()方法有什么区别？

sendRedirect()方法会创建一个新的请求，而 forward()方法只是把请求转发到一个新的目标上。重定向(redirect)以后，之前请求作用域范围以内的对象就失效了，因为会产生一个新的请求，而转发(forwarding)以后，之前请求作用域范围以内的对象还是能访问的。一般认为 sendRedirect()比 forward()要慢。

106.什么是 URL 编码和 URL 解码？

URL 编码是负责把 URL 里面的空格和其他的特殊字符替换成对应的十六进制表示，反之就是解码。

JSP

107.什么是 JSP 页面？

JSP 页面是一种包含了静态数据和 JSP 元素两种类型的文本的文本文档。静态数据可以用任何基于文本的格式来表示，比如：HTML 或者 XML。JSP 是一种混合了静态内容和动态产生的内容的技术。这里看下 JSP 的例子。

108.JSP 请求是如何被处理的？

浏览器首先要请求一个以.jsp 扩展名结尾的页面，发起 JSP 请求，然后，Web 服务器读取这个请求，使用 JSP 编译器把 JSP 页面转化成一个 Servlet 类。需要注意的是，只有当第一次请求页面或者是 JSP 文件发生改变的时候 JSP 文件才会被编译，然后服务器调用 servlet 类，处理浏览器的请求。一旦请求执行结束，servlet 会把响应发送给客户端。这里看下如何在 JSP 中获取请求参数。

109.JSP 有什么优点？

下面列出了使用 JSP 的优点：

JSP 页面是被动态编译成 Servlet 的，因此，开发者可以很容易的更新展现代码。
JSP 页面可以被预编译。
JSP 页面可以很容易的和静态模板结合，包括：HTML 或者 XML，也可以很容易的和产生动态内容的代码结合起来。
开发者可以提供让页面设计者以类 XML 格式来访问的自定义的 JSP 标签库。
开发者可以在组件层做逻辑上的改变，而不需要编辑单独使用了应用层逻辑的页面。

110.什么是 JSP 指令(Directive)？JSP 中有哪些不同类型的指令？

Directive 是当 JSP 页面被编译成 Servlet 的时候，JSP 引擎要处理的指令。Directive 用来设置页面级别的指令，从外部文件插入数据，指定自定义的标签库。Directive 是定义在<%@ 和 %>之间的。下面列出了不同类型的 Directive：

包含指令(Include directive)：用来包含文件和合并文件内容到当前的页面。
页面指令(Page directive)：用来定义 JSP 页面中特定的属性，比如错误页面和缓冲区。
Taglib 指令： 用来声明页面中使用的自定义的标签库。

111.什么是 JSP 动作(JSP action)？

JSP 动作以 XML 语法的结构来控制 Servlet 引擎的行为。当 JSP 页面被请求的时候，JSP 动作会被执行。它们可以被动态的插入到文件中，重用 JavaBean 组件，转发用户到其他的页面，或者是给 Java 插件产生 HTML 代码。下面列出了可用的动作：

jsp:include-当 JSP 页面被请求的时候包含一个文件。
jsp:useBean-找出或者是初始化 Javabean。
jsp:setProperty-设置 JavaBean 的属性。
jsp:getProperty-获取 JavaBean 的属性。
jsp:forward-把请求转发到新的页面。
jsp:plugin-产生特定浏览器的代码。

112.什么是 Scriptlets？

JSP 技术中，scriptlet 是嵌入在 JSP 页面中的一段 Java 代码。scriptlet 是位于标签内部的所有的东西，在标签与标签之间，用户可以添加任意有效的 scriplet。

113.声明(Decalaration)在哪里？

声明跟 Java 中的变量声明很相似，它用来声明随后要被表达式或者 scriptlet 使用的变量。添加的声明必须要用开始和结束标签包起来。

114.什么是表达式(Expression)？

JSP 表达式是 Web 服务器把脚本语言表达式的值转化成一个 String 对象，插入到返回给客户端的数据流中。表达式是在<%=和%>这两个标签之间定义的。

115.隐含对象是什么意思？有哪些隐含对象？

JSP 隐含对象是页面中的一些 Java 对象，JSP 容器让这些 Java 对象可以为开发者所使用。开发者不用明确的声明就可以直接使用他们。JSP 隐含对象也叫做预定义变量。下面列出了 JSP 页面中的隐含对象：

application
page
request
response
session
exception
out
config
pageContext

祝：编程快乐！

题目：Java 程序员的 10 道 XML 面试题

包括 web 开发人员的 Java 面试在内的各种面试中，XML 面试题在各种编程工作的面试中很常见。XML 是一种成熟的技术，经常作为从一个平台到其他平台传输数据的标准。XML 面试问题包括用于转换 XML 文件的 XSLT 技术，XPATH，XQuery 等各种 XML 技术和 XML 基础知识，比如 DTD 或者 Schema。

本文将看到 10 道常见的 XML 面试问答题。这些问题大部分在 Java 面试中会问到，同时在 C，C++，Scala 或其他语言的编程面试中同样很有用处。XML 并不依赖于其他编程语言，同 SQL 一样是编程人员所需要的技能之一，因此在任何技术工作面试之前准备一些 XML 问题是很有意义的。

XML 面试问答

下面是我列出的关于 XML 技术经常会问到的面试题。这些问题并不很难但涵盖了 XML 技术的一些重要领域，比如 DTD，XML Schema，XSLT 转换，XPATH 检索，XML 绑定，XML 解析器以及 XML 的基本知识，比如命名空间，校验，属性，元素等。

问题 1：XML 是什么？

答：XML 即可扩展标记语言（Extensible Markup language），你可以根据自己的需要扩展 XML。XML 中可以轻松定义<books>, <orders>等自定义标签，而在 HTML 等其他标记语言中必须使用预定义的标签，比如<p>，而不能使用用户定义的标签。使用 DTD 和 XML Schema 标准化 XML 结构。XML 主要用于从一个系统到另一系统的数据传输，比如企业级应用的客户端与服务端。

问题 2：DTD 与 XML Schema 有什么区别？

答：DTD 与 XML Schema 有以下区别：DTD 不使用 XML 编写而 XML Schema 本身就是 xml 文件，这意味着 XML 解析器等已有的 XML 工具可以用来处理 XML Schema。而且 XML Schema 是设计于 DTD 之后的，它提供了更多的类型来映射 xml 文件不同的数据类型。DTD 即文档类型描述(Document Type definition)是定义 XML 文件结构的传统方式。

问题 3：XPath 是什么？

答：XPath 是用于从 XML 文档检索元素的 XML 技术。XML 文档是结构化的，因此 XPath 可以从 XML 文件定位和检索元素、属性或值。从数据检索方面来说，XPath 与 SQL 很相似，但是它有自己的语法和规则。了解更多查看怎样使用 XPath 从 XML 文档中检索数据。

问题 4：XSLT 是什么？

答：XSLT 也是常用的 XML 技术，用于将一个 XML 文件转换为另一种 XML，HTML 或者其他的格式。XSLT 为转换 XML 文件详细定义了自己的语法，函数和操作符。通常由 XSLT 引擎完

成转换，XSLT 引擎读取 XSLT 语法编写的 XML 样式表或者 XSL 文件的指令。XSLT 大量使用递归来执行转换。一个常见 XSLT 使用就是将 XML 文件中的数据作为 HTML 页面显示。XSLT 也可以很方便地把一种 XML 文件转换为另一种 XML 文档。

问题 5：什么是 XML 元素和属性

答：最好举个例子来解释。下面是简单的 XML 片断。

```
<Orders>
    <Order id="123">
        <Symbol>6758.T</Symbol>
        <Price>2300</Price>
    <Order>
<Orders>
```

例子中 id 是元素的一个属性，其他元素都没有属性。

问题 6：什么是格式良好的 XML

答：这个问题经常在电话面试中出现。一个格式良好的 XML 意味着该 XML 文档语法上是正确的，比如它有一个根元素，所有的开放标签合适地闭合，属性值必须加引号等等。如果一个 XML 不是格式良好的，那么它可能不能被各种 XML 解析器正确地处理和解析。

问题 7：XML 命名空间是什么？它为什么很重要？

答：XML 命名空间与 Java 的 package 类似，用来避免不同来源名称相同的标签发生冲突。XML 命名空间在 XML 文档顶部使用 xmlns 属性定义，语法为 xmlns:prefix=’URI’。prefix 与 XML 文档中实际标签一起使用。下面例子为 XML 命名空间的使用。

```
<root xmlns:inst="http://instruments.com/inst"
    <inst:phone>
        <inst:number>837363223</inst:number>
    </inst:phone>
</root>
```

问题 8：DOM 和 SAX 解析器有什么区别

答：这又是一道常见面试题，不仅出现在 XML 面试题中，在 Java 面试中也会问到。DOM 和 SAX 解析器的主要区别在于它们解析 XML 文档的方式。使用 DOM 解析时，XML 文档以树形结构的形式加载到内存中，而 SAX 是事件驱动的解析器。这个问题更详细的回答查看 DOM 和 SAX 解析器之间的区别。

问题 9：XML CDATA 是什么

答：这道题很简单也很重要，但很多编程人员对它的了解并不深。CDATA 是指字符数据，它

有特殊的指令被 XML 解析器解析。XML 解析器解析 XML 文档中所有的文本，比如<name>This is name of person</name>，标签的值也会被解析，因为标签值也可能包含 XML 标签，比如<name><firstname>First Name</firstname></name>。CDATA 部分不会被 XML 解析器解析。CDATA 部分以<![CDATA[开始，以]]>结束。

问题 10：Java 的 XML 数据绑定是什么

答：Java 的 XML 绑定指从 XML 文件中创建类和对象，使用 Java 编程语言修改 XML 文档。XML 绑定的 Java API，JAXB 提供了绑定 XML 文档和 Java 对象的便利方式。另一个可选的 XML 绑定方法是使用开源库，比如 XML Beans。Java 中 XML 绑定的一个最大的优势就是利用 Java 编程能力创建和修改 XML 文档。

以上的 XML 面试问答题收集自很多编程人员，但它们对于使用 XML 技术的每个人都是有用的。由于 XML 具有平台独立的特性，XPath，XSLT，XQuery 等 XML 技术越来越重要，XML 广泛用于跨平台数据传输。尽管 XML 有冗余和文档体积大等缺点，但它在 web 服务以及带宽、速率作为次要考虑因素的系统间数据传输起很大作用。

题目：PL-SQL 经典试题

0. 准备工作:

```
set serveroutput on
```

hellowrold 程序

```
begin
dbms_output.put_line('hello  world');
end;
/
```

[语法格式]

```
--declare
   --声明的变量、类型、游标
begin
   --程序的执行部分（类似于 java 里的 main()方法）
   dbms_output.put_line('helloworld');
--exception
   --针对 begin 块中出现的异常，提供处理的机制
   --when .... then ...
   --when    .... then ...
end;
```

****

基本语法

****

1. 使用一个变量

```
declare
   --声明一个变量
   v_name varchar2(25);
begin
   --通过  select ... into ...  语句为变量赋值
  select last_name into v_name
  from employees
  where employee_id = 186;

  --  打印变量的值
  dbms_output.put_line(v_name);
end;
```

2. 使用多个变量

```
declare
   --声明变量
   v_name varchar2(25);
   v_email varchar2(25);
   v_salary number(8, 2);
   v_job_id varchar2(10);
begin
   --通过  select ... into ...  语句为变量赋值
   --被赋值的变量与 SELECT 中的列名要一一对应
  select last_name, email, salary, job_id into v_name, v_email, v_salary, v_job_id
  from employees
  where employee_id = 186;

  -- 打印变量的值
  dbms_output.put_line(v_name  || ', ' || v_email || ', ' ||    v_salary || ', ' ||    v_job_id);
end;
```

记录类型

3.1 自定义记录类型

```
declare
   --定义一个记录类型
   type customer_type is record(
      v_cust_name varchar2(20),
      v_cust_id number(10));

   --声明自定义记录类型的变量
   v_customer_type customer_type;
begin
   v_customer_type.v_cust_name := '刘德华';
   v_customer_type.v_cust_id := 1001;

   dbms_output.put_line(v_customer_type.v_cust_name||','||v_customer_type.v_cust_id);
end;
```

3.2 自定义记录类型

```
declare
   --定义一个记录类型
   type emp_record is record(
      v_name varchar2(25),
      v_email varchar2(25),
```

```
    v_salary number(8, 2),
    v_job_id varchar2(10));

  --声明自定义记录类型的变量
  v_emp_record emp_record;
begin
  --通过 select ... into ... 语句为变量赋值
 select last_name, email, salary, job_id into v_emp_record
 from employees
 where employee_id = 186;

 -- 打印变量的值
 dbms_output.put_line(v_emp_record.v_name || ', ' || v_emp_record.v_email || ', ' ||
v_emp_record.v_salary || ',' ||  v_emp_record.v_job_id);
end;
```

4. 使用 %type 定义变量，动态的获取数据的声明类型

```
declare
  --定义一个记录类型
  type emp_record is record(
    v_name employees.last_name%type,
    v_email employees.email%type,
    v_salary employees.salary%type,
    v_job_id employees.job_id%type);

  --声明自定义记录类型的变量
  v_emp_record emp_record;
begin
  --通过 select ... into ... 语句为变量赋值
 select last_name, email, salary, job_id into v_emp_record
 from employees
 where employee_id = 186;

 -- 打印变量的值
 dbms_output.put_line(v_emp_record.v_name || ', ' || v_emp_record.v_email || ', ' ||
v_emp_record.v_salary || ',' ||  v_emp_record.v_job_id);
end;
```

5. 使用 %rowtype

```
declare
--声明一个记录类型的变量
```

```
   v_emp_record employees%rowtype;
begin
   --通过  select ... into ...  语句为变量赋值
  select * into v_emp_record
  from employees
  where employee_id = 186;

  -- 打印变量的值
  dbms_output.put_line(v_emp_record.last_name  ||  ',  '  ||  v_emp_record.email  ||  ',  '  ||
v_emp_record.salary || ', ' ||    v_emp_record.job_id    || ', ' || v_emp_record.hire_date);
end;
```

6.1 赋值语句：通过变量实现查询语句

```
declare
   v_emp_record employees%rowtype;
   v_employee_id employees.employee_id%type;
begin
   --使用赋值符号位变量进行赋值
   v_employee_id := 186;

   --通过  select ... into ...  语句为变量赋值
  select * into v_emp_record
  from employees
  where employee_id = v_employee_id;

  -- 打印变量的值
  dbms_output.put_line(v_emp_record.last_name  || ', ' || v_emp_record.email || ', ' ||
v_emp_record.salary || ', ' ||    v_emp_record.job_id    || ', ' || v_emp_record.hire_date);
end;
```

6.2　通过变量实现 DELETE、INSERT、UPDATE 等操作

```
declare
   v_emp_id employees.employee_id%type;

begin
   v_emp_id := 109;
   delete from employees
   where employee_id = v_emp_id;
   --commit;
end;
```

****

****

条件判断

---

7. 使用 IF ... THEN ... ELSIF ... THEN ...ELSE ... END IF;

要求: 查询出 150 号 员工的工资, 若其工资大于或等于 10000 则打印 'salary >= 10000'; 若在 5000 到 10000 之间, 则打印 '5000<= salary < 10000'; 否则打印 'salary < 5000'

(方法一)

```
declare
   v_salary employees.salary%type;
begin
   --通过 select ... into ... 语句为变量赋值
  select salary into v_salary
  from employees
  where employee_id = 150;

  dbms_output.put_line('salary: ' || v_salary);

  -- 打印变量的值
  if v_salary >= 10000 then
       dbms_output.put_line('salary >= 10000');
  elsif v_salary >= 5000 then
       dbms_output.put_line('5000 <= salary < 10000');
  else
       dbms_output.put_line('salary < 5000');
  end if;
```

(方法二)

```
declare
        v_emp_name employees.last_name%type;
        v_emp_sal employees.salary%type;
        v_emp_sal_level varchar2(20);
begin
        select last_name,salary into v_emp_name,v_emp_sal from employees where employee_id
= 150;

        if(v_emp_sal >= 10000) then v_emp_sal_level := 'salary >= 10000';
        elsif(v_emp_sal >= 5000) then v_emp_sal_level := '5000<= salary < 10000';
        else v_emp_sal_level := 'salary < 5000';
        end if;

        dbms_output.put_line(v_emp_name||','||v_emp_sal||','||v_emp_sal);
end;
```

7+ 使用 CASE ... WHEN ... THEN ...ELSE ... END 完成上面的任务

```
declare
        v_sal employees.salary%type;
        v_msg varchar2(50);
begin
        select salary into v_sal
        from employees
        where employee_id = 150;

        --case 不能向下面这样用
        /*
        case v_sal when salary >= 10000 then v_msg := '>=10000'
                        when salary >= 5000 then v_msg := '5000<= salary < 10000'
                        else v_msg := 'salary < 5000'
        end;
        */

        v_msg :=
                case trunc(v_sal / 5000)
                        when 0 then 'salary < 5000'
                        when 1 then '5000<= salary < 10000'
                        else 'salary >= 10000'
                end;

        dbms_output.put_line(v_sal ||','||v_msg);
end;
```

8. 使用 CASE ... WHEN ... THEN ... ELSE ... END;
要求: 查询出 122 号员工的 JOB_ID, 若其值为 'IT_PROG', 则打印 'GRADE: A';
'AC_MGT', 打印 'GRADE B',
'AC_ACCOUNT', 打印 'GRADE C';
否则打印 'GRADE D'

```
declare
        --声明变量
        v_grade char(1);
        v_job_id employees.job_id%type;
begin
        select job_id into v_job_id
        from employees
        where employee_id = 122;

        dbms_output.put_line('job_id:  ' || v_job_id);
```

```
        --根据 v_job_id 的取值, 利用 case 字句为 v_grade 赋值
        v_grade :=
                case v_job_id when 'IT_PROG' then 'A'
                              when 'AC_MGT' then 'B'
                              when 'AC_ACCOUNT' then 'C'
                              else 'D'
                 end;

        dbms_output.put_line('GRADE: ' || v_grade);
end;
```

循环结构

---

9. 使用循环语句打印 1 - 100.（三种方式）

```
1).   LOOP ... EXIT WHEN ... END LOOP
declare
        --初始化条件
        v_i number(3) := 1;
begin
        loop
        --循环体
          dbms_output.put_line(v_i);
                                                    --循环条件
          exit when v_i = 100;
                                                    --迭代条件
          v_i := v_i + 1;
        end loop;
end;

2). WHILE ... LOOP ... END LOOP
declare
        --初始化条件
        v_i number(3) := 1;
begin
        --循环条件
        while v_i <= 100 loop
                                                        --循环体
                dbms_output.put_line(v_i);
                                                        --迭代条件
                v_i := v_i + 1;
        end loop;
end;
```

3).

```
begin
        for i in 1 .. 100 loop
                dbms_output.put_line(i);
        end loop;
end;
```

10. 综合使用 if, while 语句, 打印 1 - 100 之间的所有素数(素数: 有且仅用两个正约数的整数, 2, 3, 5, 7, 11, 13, ...).

```
declare
   v_flag number(1):=1;
   v_i number(3):=2;
   v_j number(2):=2;
begin

   while (v_i<=100) loop
            while v_j <= sqrt(v_i) loop
                    if (mod(v_i,v_j)=0) then v_flag:= 0;end if;
                    v_j :=v_j +1;
            end loop;
            if(v_flag=1) then dbms_output.put_line(v_i);end  if;

            v_flag :=1;
            v_j := 2;
            v_i :=v_i +1;
     end loop;

end;
```

(法二)使用 for 循环实现 1-100 之间的素数的输出

```
declare
   --标记值, 若为 1 则是素数, 否则不是
   v_flag number(1) := 0;
begin
     for i in 2 .. 100 loop

        v_flag := 1;

        for j in 2 .. sqrt(i) loop
            if i mod j = 0 then
                v_flag := 0;
            end if;
```

```
        end loop;

        if v_flag = 1 then
            dbms_output.put_line(i);
        end if;

    end loop;
end;
```

11. 使用 goto

```
declare
  --标记值，若为 1 则是素数，否则不是
  v_flag number(1) := 0;
begin
    for i in 2 .. 100 loop
        v_flag := 1;

        for j in 2 .. sqrt(i) loop
            if i mod j = 0 then
                v_flag := 0;
                goto label;
            end if;
        end loop;

        <<label>>
        if v_flag = 1 then
            dbms_output.put_line(i);
        end if;

    end loop;
end;
```

11+.打印 1——100 的自然数，当打印到 50 时，跳出循环，输出“打印结束”

(方法一)

```
begin
  for i in 1..100 loop
      dbms_output.put_line(i);
      if(i = 50) then
      goto label;
      end if;
  end loop;

      <<label>>
      dbms_output.put_line('打印结束');
```

```
end;
(方法二)
begin
   for i in 1..100 loop
         dbms_output.put_line(i);
         if(i mod 50 = 0) then dbms_output.put_line('打印结束');
         exit;
         end if;
   end loop;
end;
```

****

游标的使用

****

12.1 使用游标

要求: 打印出 80 部门的所有的员工的工资:salary: xxx

```
declare
   --1. 定义游标
   cursor salary_cursor is select salary from employees where department_id = 80;
   v_salary employees.salary%type;
begin
  --2. 打开游标
  open salary_cursor;

  --3. 提取游标
  fetch salary_cursor into v_salary;

  --4. 对游标进行循环操作: 判断游标中是否有下一条记录
while salary_cursor%found loop
         dbms_output.put_line('salary: ' || v_salary);
         fetch salary_cursor into v_salary;
end loop;

  --5. 关闭游标
  close   salary_cursor;
end;
```

12.2 使用游标

要求: 打印出 80 部门的所有的员工的工资: Xxx 's salary is: xxx

```
declare
   cursor sal_cursor is select salary ,last_name from employees where department_id = 80;
   v_sal number(10);
```

```
   v_name varchar2(20);
begin
   open sal_cursor;

   fetch sal_cursor into v_sal,v_name;

   while sal_cursor%found  loop
            dbms_output.put_line(v_name||'`s  salary is '||v_sal);
            fetch sal_cursor into v_sal,v_name;
   end loop;

   close sal_cursor;

end;
```

13. 使用游标的练习:
打印出 manager_id 为 100 的员工的 last_name, email, salary 信息(使用游标, 记录类型)

```
declare
            --声明游标
            cursor emp_cursor is select last_name, email, salary from employees where
manager_id = 100;

            --声明记录类型
            type emp_record is record(
                  name employees.last_name%type,
                  email employees.email%type,
                  salary employees.salary%type
            );

            -- 声明记录类型的变量
            v_emp_record emp_record;
begin
            --打开游标
            open emp_cursor;

            --提取游标
            fetch emp_cursor into v_emp_record;

            --对游标进行循环操作
            while emp_cursor%found  loop
                  dbms_output.put_line(v_emp_record.name || ', ' || v_emp_record.email
|| ', ' || v_emp_record.salary );
                  fetch emp_cursor into v_emp_record;
```

```
        end loop;

        --关闭游标
        close emp_cursor;
end;
```

(法二：使用 for 循环)

```
declare

    cursor emp_cursor is
    select last_name,email,salary
    from employees
    where manager_id = 100;

begin


    for v_emp_record in emp_cursor loop

dbms_output.put_line(v_emp_record.last_name||','||v_emp_record.email||','||v_emp_record.
salary);
    end loop;
end;
```

14. 利用游标，调整公司中员工的工资:

工资范围	调整基数
0 - 5000	5%
5000 - 10000	3%
10000 - 15000	2%
15000 -	1%

```
declare
    --定义游标
    cursor emp_sal_cursor is select salary, employee_id from employees;

    --定义基数变量
    temp number(4, 2);

    --定义存放游标值的变量
    v_sal employees.salary%type;
    v_id employees.employee_id%type;
begin
    --打开游标
    open emp_sal_cursor;
```

```
    --提取游标
    fetch emp_sal_cursor into v_sal, v_id;

    --处理游标的循环操作
    while emp_sal_cursor%found loop
        --判断员工的工资，执行 update 操作
        --dbms_output.put_line(v_id || ': ' || v_sal);

        if v_sal <= 5000 then
            temp := 0.05;
        elsif v_sal<= 10000 then
            temp := 0.03;
        elsif v_sal <= 15000 then
            temp := 0.02;
        else
            temp := 0.01;
        end if;

        --dbms_output.put_line(v_id || ': ' || v_sal || ', ' || temp);
        update employees set salary = salary * (1 + temp) where employee_id = v_id;

        fetch emp_sal_cursor into v_sal, v_id;
    end loop;
    --关闭游标
    close emp_sal_cursor;
end;
```

使用 SQL 中的 decode 函数

```
update employees set salary = salary * (1 + (decode(trunc(salary/5000), 0, 0.05,
                                                                        1, 0.03,
                                                                        2, 0.02,
                                                                        0.01)))
```

15. 利用游标 for 循环完成 14.

```
declare
    --定义游标
    cursor emp_sal_cursor is select salary, employee_id id from employees;

    --定义基数变量
    temp number(4, 2);
begin
```

```
    --处理游标的循环操作
    for c in emp_sal_cursor loop
        --判断员工的工资，执行 update 操作
        --dbms_output.put_line(c.employee_id || ': ' || c.salary);

        if c.salary <= 5000 then
            temp := 0.05;
        elsif c.salary <= 10000 then
            temp := 0.03;
        elsif c.salary <= 15000 then
            temp := 0.02;
        else
            temp := 0.01;
        end if;

        --dbms_output.put_line(v_id || ': ' || v_sal || ', ' || temp);
        update employees set salary = salary * (1 + temp) where employee_id = c.id;
    end loop;
end;
```

16*. 带参数的游标

```
declare
    --定义游标
    cursor emp_sal_cursor(dept_id number, sal number) is
        select salary + 1000 sal, employee_id id
        from employees
        where department_id = dept_id and salary > sal;

    --定义基数变量
    temp number(4, 2);
begin
    --处理游标的循环操作
    for c in emp_sal_cursor(sal => 4000, dept_id => 80) loop
        --判断员工的工资，执行 update 操作
        --dbms_output.put_line(c.id || ': ' || c.sal);

        if c.sal <= 5000 then
            temp := 0.05;
        elsif c.sal <= 10000 then
            temp := 0.03;
        elsif c.sal <= 15000 then
            temp := 0.02;
        else
```

```
            temp := 0.01;
        end if;

        dbms_output.put_line(c.sal || ': ' || c.id || ', ' || temp);
        --update employees set salary = salary * (1 + temp) where employee_id = c.id;
    end loop;
end;
```

17. 隐式游标：更新指定员工 salary(涨工资 10)，如果该员工没有找到，则打印”查无此人”信息

```
begin
        update employees set salary = salary + 10 where employee_id = 1005;

        if sql%notfound then
            dbms_output.put_line('查无此人!');
        end if;
end;
```

****

异常处理

****

[预定义异常]

```
declare

  v_sal employees.salary%type;
begin
  select salary into v_sal
  from employees
  where employee_id >100;

  dbms_output.put_line(v_sal);

exception
  when Too_many_rows then dbms_output.put_line('输出的行数太多了');
end;
```

[非预定义异常]

```
declare

  v_sal employees.salary%type;
  --声明一个异常
  delete_mgr_excep exception;
  --把自定义的异常和 oracle 的错误关联起来
```

```
   PRAGMA EXCEPTION_INIT(delete_mgr_excep,-2292);
begin
   delete from employees
   where employee_id = 100;

   select salary into v_sal
   from employees
   where employee_id >100;

   dbms_output.put_line(v_sal);

exception
   when Too_many_rows then dbms_output.put_line('输出的行数太多了');
   when delete_mgr_excep then dbms_output.put_line('Manager 不能直接被删除');
end;

[用户自定义异常]
declare

   v_sal employees.salary%type;
   --声明一个异常
   delete_mgr_excep exception;
   --把自定义的异常和 oracle 的错误关联起来
   PRAGMA EXCEPTION_INIT(delete_mgr_excep,-2292);

   --声明一个异常
   too_high_sal exception;
begin

   select salary into v_sal
   from employees
   where employee_id =100;

   if v_sal > 1000 then
       raise too_high_sal;
   end if;

   delete from employees
   where employee_id = 100;

   dbms_output.put_line(v_sal);

exception
   when Too_many_rows then dbms_output.put_line('输出的行数太多了');
```

```
  when delete_mgr_excep then dbms_output.put_line('Manager 不能直接被删除');
  --处理异常
  when too_high_sal then dbms_output.put_line('工资过高了');
end;
```

18. 异常的基本程序:
通过 select ... into ... 查询某人的工资，若没有查询到，则输出 "未找到数据"

```
declare
  --定义一个变量
  v_sal employees.salary%type;
begin
  --使用 select ... into ... 为 v_sal 赋值
  select salary into v_sal from employees where employee_id = 1000;
  dbms_output.put_line('salary:    ' || v_sal);
exception
  when No_data_found then
       dbms_output.put_line('未找到数据');
end;
```

或

```
declare
  --定义一个变量
  v_sal employees.salary%type;
begin
  --使用 select ... into ... 为 v_sal 赋值
  select salary into v_sal from employees;
  dbms_output.put_line('salary:    ' || v_sal);
exception
  when No_data_found then
       dbms_output.put_line('未找到数据!');
  when Too_many_rows then
       dbms_output.put_line('数据过多!');
end;
```

19. 更新指定员工工资，如工资小于 300，则加 100；对 NO_DATA_FOUND 异常，TOO_MANY_ROWS 进行处理.

```
declare
    v_sal employees.salary%type;
begin
    select salary into v_sal from employees where employee_id = 100;

    if(v_sal < 300) then update employees set salary = salary + 100 where employee_id = 100;
```

```
    else dbms_output.put_line('工资大于 300');
    end if;
exception
    when no_data_found then dbms_output.put_line('未找到数据');
     when too_many_rows then dbms_output.put_line('输出的数据行太多');
end;
```

20. 处理非预定义的异常处理: "违反完整约束条件"

```
declare
  --1. 定义异常
  temp_exception exception;

  --2. 将其定义好的异常情况，与标准的 ORACLE 错误联系起来，使用 EXCEPTION_INIT 语句
  PRAGMA EXCEPTION_INIT(temp_exception, -2292);
begin
  delete from employees where employee_id = 100;

exception
  --3. 处理异常
  when temp_exception then
        dbms_output.put_line('违反完整性约束!');
end;
```

21. 自定义异常: 更新指定员工工资，增加 100；若该员工不存在则抛出用户自定义异常: no_result

```
declare
  --自定义异常
  no_result exception;
begin
  update employees set salary = salary + 100 where employee_id = 1001;

  --使用隐式游标, 抛出自定义异常
  if sql%notfound then
      raise no_result;
  end if;

exception

  --处理程序抛出的异常
  when no_result then
      dbms_output.put_line('更新失败');
```

```
end;
```

****

存储函数和过程

****

[存储函数：有返回值，创建完成后，通过 select function() from dual;执行]
[存储过程：由于没有返回值，创建完成后，不能使用 select 语句，只能使用 pl/sql 块执行]

[格式]

```
--函数的声明(有参数的写在小括号里)
create or replace function func_name(v_param varchar2)
--返回值类型
return varchar2
is
--PL/SQL 块变量、记录类型、游标的声明(类似于前面的 declare 的部分)
begin
--函数体(可以实现增删改查等操作，返回值需要 return)
        return 'helloworld'|| v_param;
end;
```

22.1 函数的 helloworld: 返回一个 "helloworld" 的字符串

```
create or replace function hello_func
return varchar2
is
begin
        return 'helloworld';
end;
```

执行函数

```
begin
    dbms_output.put_line(hello_func());
end;
```

或者： select hello_func() from dual;

22.2 返回一个"helloworld: atguigu"的字符串，其中 atguigu 由执行函数时输入。

```
--函数的声明(有参数的写在小括号里)
create or replace function hello_func(v_logo varchar2)
--返回值类型
return varchar2
is
--PL/SQL 块变量的声明
```

```
begin
--函数体
        return 'helloworld'|| v_logo;
end;
```

22.3 创建一个存储函数，返回当前的系统时间

```
create or replace function func1
return date
is
--定义变量
v_date date;
begin
-函数体
--v_date := sysdate;
    select sysdate into v_date from dual;
        dbms_output.put_line('我是函数哦');

        return v_date;
end;
```

执行法 1：

```
select func1 from dual;
```

执行法 2：

```
declare
  v_date date;
begin
  v_date := func1;
  dbms_output.put_line(v_date);
end;
```

23. 定义带参数的函数：两个数相加

```
create or replace function add_func(a number, b number)
return number
is
begin
        return (a + b);
end;
```

执行函数

```
begin
    dbms_output.put_line(add_func(12, 13));
end;
```

或者

```
select add_func(12,13) from dual;
```

24. 定义一个函数: 获取给定部门的工资总和, 要求:部门号定义为参数, 工资总额定义为返回值.

```
create or replace function sum_sal(dept_id number)
        return number
        is

        cursor sal_cursor is select salary from employees where department_id = dept_id;
        v_sum_sal number(8) := 0;
begin
        for c in sal_cursor loop
            v_sum_sal := v_sum_sal + c.salary;
        end loop;

        --dbms_output.put_line('sum salary: ' || v_sum_sal);
        return v_sum_sal;
end;
```

执行函数

```
begin
    dbms_output.put_line(sum_sal(80));
end;
```

25. 关于 OUT 型的参数: 因为函数只能有一个返回值, PL/SQL 程序可以通过 OUT 型的参数实现有多个返回值

要求: 定义一个函数: 获取给定部门的工资总和 和 该部门的员工总数(定义为 OUT 类型的参数).
要求: 部门号定义为参数, 工资总额定义为返回值.

```
create or replace function sum_sal(dept_id number, total_count out number)
        return number
        is

        cursor sal_cursor is select salary from employees where department_id = dept_id;
        v_sum_sal number(8) := 0;
begin
        total_count := 0;

        for c in sal_cursor loop
```

```
            v_sum_sal := v_sum_sal + c.salary;
            total_count := total_count + 1;
        end loop;

        --dbms_output.put_line('sum  salary: ' || v_sum_sal);
        return v_sum_sal;
end;
```

执行函数:

```
delare
   v_total number(3) := 0;

begin
      dbms_output.put_line(sum_sal(80,  v_total));
      dbms_output.put_line(v_total);
end;
```

26*. 定义一个存储过程: 获取给定部门的工资总和(通过 out 参数), 要求:部门号和工资总额定义为参数

```
create or replace procedure sum_sal_procedure(dept_id  number, v_sum_sal out number)
        is

        cursor sal_cursor is select salary from employees where department_id = dept_id;
begin
        v_sum_sal := 0;

        for c in sal_cursor loop
            --dbms_output.put_line(c.salary);
            v_sum_sal := v_sum_sal + c.salary;
        end loop;

        dbms_output.put_line('sum  salary: ' || v_sum_sal);
end;
[执行]
declare
      v_sum_sal number(10) := 0;
begin
      sum_sal_procedure(80,v_sum_sal);
end;
```

27*. 自定义一个存储过程完成以下操作:
对给定部门(作为输入参数)的员工进行加薪操作, 若其到公司的时间在

(? , 95) 期间, 为其加薪 %5
[95 , 98) %3
[98, ?) %1
得到以下返回结果: 为此次加薪公司每月需要额外付出多少成本(定义一个 OUT 型的输出参数).

```
create or replace procedure add_sal_procedure(dept_id number, temp out number)

is

        cursor sal_cursor is select employee_id id, hire_date hd, salary sal from employees
where department_id = dept_id;
        a number(4, 2) := 0;
begin
        temp := 0;

        for c in sal_cursor loop
            a := 0;

            if c.hd < to_date('1995-1-1', 'yyyy-mm-dd') then
                a := 0.05;
            elsif c.hd < to_date('1998-1-1', 'yyyy-mm-dd') then
                a := 0.03;
            else
                a := 0.01;
            end if;

            temp := temp + c.sal * a;
            update employees set salary = salary * (1 + a) where employee_id = c.id;
        end loop;
end;
```

****

触发器

****

一个 helloworld 级别的触发器

```
create or replace trigger hello_trigger
after
update on employees
--for each row
begin
    dbms_output.put_line('hello...');
    --dbms_output.put_line('old.salary:'||  :OLD.salary||',new.salary'||:NEW.salary);
end;
```

然后执行：update employees set salary = salary + 1000;

28. 触发器的 helloworld: 编写一个触发器，在向 emp 表中插入记录时，打印 'helloworld'

```
create or replace trigger emp_trigger
after
insert on emp
for each row
begin
        dbms_output.put_line('helloworld');
end;
```

29. 行级触发器: 每更新 employees 表中的一条记录，都会导致触发器执行

```
create or replace trigger employees_trigger
after
update on employees
for each row
begin
        dbms_output.put_line('修改了一条记录!');
end;
```

语句级触发器: 一个 update/delete/insert 语句只使触发器执行一次

```
create or replace trigger employees_trigger
after
update on employees
begin
        dbms_output.put_line('修改了一条记录!');
end;
```

30. 使用 :new, :old 修饰符

```
create or replace trigger employees_trigger
after
update on employees
for each row
begin
        dbms_output.put_line('old  salary: ' || :old.salary || ', new salary: ' || :new.salary);
end;
```

31. 编写一个触发器，在对 my_emp 记录进行删除的时候，在 my_emp_bak 表中备份对应的记录

1). 准备工作:

```
create table my_emp as select employee_id id, last_name name, salary sal from employees;

create table my_emp_bak as select employee_id id, last_name name, salary sal from employees
where 1 = 2
```

2).

```
create or replace trigger bak_emp_trigger
        before delete on my_emp
        for each row

begin
        insert into my_emp_bak values(:old.id, :old.name, :old.sal);
end;
```

SQL 面试用题

employees 表：

EMPLOYEE_ID	NUMBER(6)
FIRST_NAME	VARCHAR2(20)
LAST_NAME	VARCHAR2(25)
EMAIL	VARCHAR2(25)
PHONE_NUMBER	VARCHAR2(20)
HIRE_DATE	DATE
JOB_ID	VARCHAR2(10)
SALARY	NUMBER(8,2)
COMMISSION_PCT	NUMBER(2,2)
MANAGER_ID	NUMBER(6)
DEPARTMENT_ID	NUMBER(4)

departments 表：

DEPARTMENT_ID	NUMBER(4)
DEPARTMENT_NAME	VARCHAR2(30)
MANAGER_ID	NUMBER(6)
LOCATION_ID	NUMBER(4)

locations 表：

LOCATION_ID	NUMBER(4)
STREET_ADDRESS	VARCHAR2(40)
POSTAL_CODE	VARCHAR2(12)
CITY	VARCHAR2(30)
STATE_PROVINCE	VARCHAR2(25)
COUNTRY_ID	CHAR(2)

jobs 表：

JOB_ID	VARCHAR2(10)
JOB_TITLE	VARCHAR2(35)
MIN_SALARY	NUMBER(6)
MAX_SALARY	NUMBER(6)

job_grades 表：

GRADE_LEVEL	VARCHAR2(3)
LOWEST_SAL	NUMBER
HIGHEST_SAL	NUMBER

1. 查询每个月倒数第 2 天入职的员工的信息.

2. 查询出 last_name 为 'Chen' 的 manager 的信息.

3. 查询平均工资高于 8000 的部门 id 和它的平均工资.

4. 查询工资最低的员工信息: last_name, salary

5. 查询平均工资最低的部门信息

6. 查询平均工资最低的部门信息和该部门的平均工资

7. 查询平均工资最高的 job 信息

8. 查询平均工资高于公司平均工资的部门有哪些?

9. 查询出公司中所有 manager 的详细信息.

10．各个部门中 最高工资中最低的那个部门的 最低工资是多少

11．查询平均工资最高的部门的 manager 的详细信息：last_name, department_id, email, salary

12．查询 1999 年来公司的人所有员工的最高工资的那个员工的信息.

13．返回其它部门中比 job_id 为‘IT_PROG’部门所有工资都低的员工的员工号、姓名、job_id 以及 salary

**************************answers*************************

1．查询每个月倒数第 2 天入职的员工的信息.

```
select last_name, hire_date
from employees
where hire_date = last_day(hire_date) - 1
```

2．查询出 last_name 为 'Chen' 的 manager 的信息.

1)．通过两条 sql 查询:

```
select manager_id
from employees
where lower(last_name) = 'chen' --返回的结果为 108

select *
from employees
where employee_id = 108
```

2)．通过一条 sql 查询(自连接):

```
select m.*
from employees e, employees m
where e.manager_id = m.employee_id and e.last_name
= 'Chen'
```

3). 通过一条 sql 查询(子查询):

```
select *
from employees
where employee_id = (
                    select manager_id
                    from employees
                    where last_name = 'Chen'
                   )
```

3. 查询平均工资高于 8000 的部门 id 和它的平均工资.

```
SELECT department_id, avg(salary)
FROM employees e
GROUP BY department_id
HAVING avg(salary) > 8000
```

4. 查询工资最低的员工信息: last_name, salary

```
SELECT last_name, salary
FROM employees
WHERE salary = (
   SELECT min(salary)
   FROM employees
)
```

5. 查询平均工资最低的部门信息

```
SELECT *
FROM departments
WHERE department_id = (
   SELECT department_id
   FROM employees
   GROUP BY department_id
   HAVING avg(salary) = (
      SELECT min(avg(salary))
      FROM employees
      GROUP BY department_id
   )
)
```

6．查询平均工资最低的部门信息和该部门的平均工资

```
select d.*, (select avg(salary) from employees where
department_id = d.department_id)
from departments d
where d.department_id = (
     SELECT department_id
     FROM employees
     GROUP BY department_id
     HAVING avg(salary) = (
          SELECT min(avg(salary))
          FROM employees
          GROUP BY department_id
           )
     )
```

7．查询平均工资最高的 job 信息

1)．按 job_id 分组，查询最高的平均工资

```
SELECT max(avg(salary))
FROM employees
GROUP BY job_id
```

2)．查询出平均工资等于 1) 的 job_id

```
SELECT job_id
FROM employees
GROUP BY job_id
HAVING avg(salary) = (
   SELECT max(avg(salary))
   FROM employees
   GROUP BY job_id
)
```

3)．查询出 2) 对应的 job 信息

```
SELECT *
FROM jobs
```

```
WHERE job_id = (
   SELECT job_id
   FROM employees
   GROUP BY job_id
   HAVING avg(salary) = (
      SELECT max(avg(salary))
      FROM employees
      GROUP BY job_id
   )
)
```

8．查询平均工资高于公司平均工资的部门有哪些？

1)．查询出公司的平均工资

```
SELECT avg(salary)
FROM employees
```

2)．查询平均工资高于 1) 的部门 ID

```
SELECT department_id
FROM employees
GROUP BY department_id
HAVING avg(salary) > (
   SELECT avg(salary)
   FROM employees
)
```

9．查询出公司中所有 manager 的详细信息．

1)．查询出所有的 manager_id

```
SELECT distinct manager_id
FROM employeess
```

2)．查询出 employee_id 为 1) 查询结果的那些员工的信息

```
SELECT employee_id, last_name
FROM employees
WHERE employee_id in (
   SELECT distinct manager_id
```

```
    FROM employees
)
```

10．各个部门中 最高工资中最低的那个部门的 最低工资是多少

1)．查询出各个部门的最高工资

```
SELECT max(salary)
FROM employees
GROUP BY department_id
```

2)．查询出 1) 对应的查询结果的最低值：各个部门中最低的最高工资(无法查询对应的 department_id)

```
SELECT min(max(salary))
FROM employees
GROUP BY department_id
```

3)．查询出 2) 所对应的部门 id 是多少：各个部门中最高工资等于 2) 的那个部门的 id

```
SELECT department_id
FROM employees
GROUP BY department_id
HAVING max(salary) = (
    SELECT min(max(salary))
    FROM employees
    GROUP BY department_id
)
```

4)．查询出 3) 所在部门的最低工资

```
SELECT min(salary)
FROM employees
WHERE department_id = (
    SELECT department_id
    FROM employees
    GROUP BY department_id
    HAVING max(salary) = (
```

```
        SELECT min(max(salary))
        FROM employees
        GROUP BY department_id
    )
)
```

11. 查询平均工资最高的部门的 manager 的详细信息：last_name, department_id, email, salary

1). 各个部门中，查询平均工资最高的平均工资是多少

```
SELECT max(avg(salary))
FROM employees
GROUP BY department_id
```

2). 各个部门中，平均工资等于 1) 的那个部门的部门号是多少

```
SELECT department_id
FROM employees
GROUP BY department_id
HAVING avg(salary) = (
    SELECT max(avg(salary))
    FROM employees
    GROUP BY department_id
)
```

3). 查询出 2) 对应的部门的 manager_id

```
SELECT manager_id
FROM departments
WHERE department_id = (
    SELECT department_id
    FROM employees
    GROUP BY department_id
    HAVING avg(salary) = (
        SELECT max(avg(salary))
        FROM employees
        GROUP BY department_id
    )
)
```

4). 查询出 employee_id 为 3) 查询的 manager_id 的员工的 last_name, department_id, email, salary

```
SELECT last_name, department_id, email, salary
FROM employees
WHERE employee_id = (
   SELECT manager_id
   FROM departments
   WHERE department_id = (
      SELECT department_id
      FROM employees
      GROUP BY department_id
      HAVING avg(salary) = (
         SELECT max(avg(salary))
         FROM employees
         GROUP BY department_id
      )
   )
)
```

12. 查询 1999 年来公司的人所有员工的最高工资的那个员工的信息.

1). 查询出 1999 年来公司的所有的员工的 salary

```
SELECT salary
FROM employees
WHERE to_char(hire_date, 'yyyy') = '1999'
```

2). 查询出 1) 对应的结果的最大值

```
SELECT max(salary)
FROM employees
WHERE to_char(hire_date, 'yyyy') = '1999'
```

3). 查询工资等于 2) 对应的结果且 1999 年入职的员工信息

```
SELECT *
FROM employees
WHERE to_char(hire_date, 'yyyy') = '1999' AND salary = (
      SELECT max(salary)
```

```
            FROM employees
            WHERE to_char(hire_date, 'yyyy') = '1999'
        )
```

13．返回其它部门中比 job_id 为‘IT_PROG’部门所有工资都低的员工的员工号、姓名、job_id 以及 salary

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

**********高级子查询**********

- 书写多列子查询
- 在 FROM 子句中使用子查询
- 在 SQL 中使用单列子查询
- 书写相关子查询
- 使用 EXISTS 和 NOT EXISTS 操作符
- 使用子查询更新和删除数据
- 使用 WITH 子句

--多列子查询（不成对比较 & 成对比较）

1． 查询与 141 号或 174 号员工的 manager_id 和 department_id 相同的其他员工的 employee_id, manager_id, department_id

[方式一]

```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   manager_id IN
```

```
                (SELECT  manager_id
                 FROM    employees
                 WHERE   employee_id IN (174,141))
AND     department_id IN
                (SELECT  department_id
                 FROM    employees
                 WHERE   employee_id IN (174,141))
AND   employee_id NOT IN(174,141);
```

[方式二]

```
SELECT   employee_id, manager_id, department_id
FROM  employees
WHERE  (manager_id, department_id) IN
                    (SELECT manager_id, department_id
                     FROM   employees
                     WHERE  employee_id IN (141,174))
AND   employee_id NOT IN (141,174);
```

--在 FROM 子句中使用子查询

2. 返回比本部门平均工资高的员工的 last_name, department_id, salary 及平均工资

[方式一]

```
select last_name,department_id,salary,
(select avg(salary)from employees e3
where e1.department_id = e3.department_id
group by department_id) avg_salary
from employees e1
where salary >
        (select avg(salary)
         from employees e2
         where e1.department_id = e2.department_id
         --group by department_id
         )
```

[方式二]

```
SELECT  a.last_name, a.salary,
       a.department_id, b.salavg
```

```
FROM    employees a, (SELECT   department_id,
                    AVG(salary) salavg
                    FROM     employees
                    GROUP BY department_id) b
WHERE   a.department_id = b.department_id
AND     a.salary > b.salavg;
```

--单列子查询表达式

- Oracle8*i* 只在下列情况下可以使用，例如：
  - SELECT 语句 (FROM 和 WHERE 子句)
  - INSERT 语句中的 VALUES 列表中
- Oracle9*i* 中单列子查询表达式可在下列情况下使用：
  - DECODE 和 CASE
  - SELECT 中除 GROUP BY 子句以外的所有子句中

3. 显式员工的 employee_id,last_name 和 location。其中，若员工 department_id 与 location_id 为 1800 的 department_id 相同，则 location 为'Canada',其余则为'USA'。

```
SELECT employee_id, last_name,
       (CASE department_id
        WHEN  (SELECT department_id FROM departments
                 WHERE location_id = 1800)
        THEN 'Canada' ELSE 'USA' END) location
FROM   employees;
```

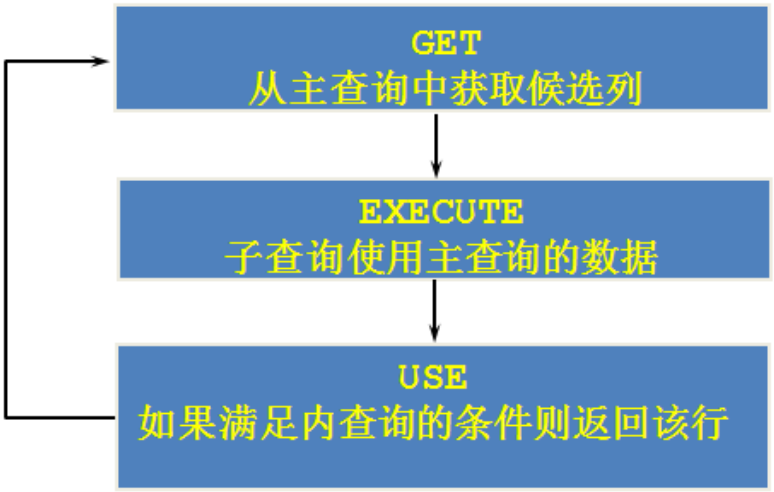
4. 查询员工的 employee_id,last_name，要求按照员工的 department_name 排序

```
SELECT   employee_id, last_name
FROM     employees e
```

```
ORDER BY (SELECT department_name
          FROM departments d
          WHERE e.department_id = d.department_id);
```

--相关子查询

相关子查询按照一行接一行的顺序执行，主查询的每一行都执行一次子查询




```
SELECT column1, column2, ...
FROM   table1  outer
WHERE  column1 operator
                    (SELECT  colum1, column2
                     FROM    table2
                     WHERE   expr1 =
                                outer.expr2);
```

5.查询员工中工资大于本部门平均工资的员工的 last_name，

salary 和其 department_id

```
SELECT last_name, salary, department_id
FROM   employees outer
WHERE  salary >
```

```
                    (SELECT AVG(salary)
                     FROM   employees
                     WHERE  department_id =
                            outer.department_id) ;
```

6. 若 employees 表中 employee_id 与 job_history 表中 employee_id 相同的数目不小于 2，输出这些相同 id 的员工的 employee_id,last_name 和其 job_id

```
SELECT e.employee_id, last_name,e.job_id
FROM   employees e
WHERE  2 <= (SELECT COUNT(*)
             FROM   job_history
             WHERE  employee_id = e.employee_id);
```

--EXISTS 操作符

- EXISTS 操作符检查在子查询中是否存在满足条件的行
- **如果在子查询中存在满足条件的行：**
  - 不在子查询中继续查找
  - 条件返回 TRUE

7. 查询公司管理者的 employee_id,last_name,job_id, department_id 信息

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE  EXISTS ( SELECT 'X'
                FROM   employees
                WHERE  manager_id =
                       outer.employee_id);
```

8. 查询 departments 表中，不存在于 employees 表中的部门的

department_id 和 department_name

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM   employees
                  WHERE  department_id
                         = d.department_id);
```

--关于数据更新

9.修改表 employees,添加 department_name 列，赋予 department_id 相应的部门名称。

```
ALTER TABLE employees
ADD(department_name VARCHAR2(14));


UPDATE employees e
SET    department_name =
             (SELECT department_name
          FROM   departments d
          WHERE  e.department_id = d.department_id);
```

--关于数据删除

10.删除表 employees 中，其与 emp_history 表皆有的数据

```
DELETE FROM employees E
WHERE employee_id in
          (SELECT employee_id
           FROM   emp_history
           WHERE  employee_id = E.employee_id);
```

--WITH 子句

11. **查询公司中各部门的总工资大于公司中各部门的平均总工资的部门信息**

```
WITH
```

```
dept_costs  AS (
   SELECT  d.department_name, SUM(e.salary) AS dept_total
   FROM    employees e, departments d
   WHERE   e.department_id = d.department_id
   GROUP BY d.department_name),
avg_cost    AS (
   SELECT SUM(dept_total)/COUNT(*) AS dept_avg
   FROM   dept_costs)
SELECT *
FROM   dept_costs
WHERE  dept_total >
       (SELECT dept_avg
        FROM avg_cost)
ORDER BY department_name;
```

附加题目：

12.查询员工的 last_name, department_id, salary.其中员工的 salary,department_id 与有奖金的任何一个员工的 salary, department_id 相同即可

```
select last_name, department_id, salary
from employees
where (salary,department_id) in (
select salary,department_id
from employees
where commission_pct is not null
                            )
```

13.选择工资大于所有 JOB_ID = 'SA_MAN'的员工的工资的员工的 last_name, job_id, salary

```
select last_name, job_id, salary
from employees
where salary > all(
                select salary
```

```
                from employees
                where job_id = 'SA_MAN'
                )
```

14.选择所有没有管理者的员工的 last_name

```
select last_name
from employees e1
where not exists (
            select 'A'
            from employees e2
            where e1.manager_id = e2.employee_id
                )
```

15. 查询 10，50，20 号部门的 job_id，department_id 并且 department_id 按 10，50，20 的顺序排列

```
Column dummy noprint;
select job_id , department_id ,1 dummy
from employees
where department_id = 10
union
select job_id , department_id , 2
from employees
where department_id = 50
union
select job_id , department_id , 3
from employees
where department_id= 20
order by 3
```