## 1.冒泡排序；

public class BubbleSort{

public static void main(String[] args){

int str[] = {67, 69, 75, 87, 89, 90, 99, 100};

for (int i = 0; i &lt; str.lengt -1; i++){ //最多做n-1趟排序

for(int j = 0 ;j &lt; str.length - i - 1; j++){ //对当前无序区间str[0......length-i-1]进行排序(j的范围很关键，这个范围是在逐步缩小的)

if(str[j] &lt; str[j + 1]){ //把小的值交换到后面

int temp = str[j];

str[j] = str[j + 1];

str[j + 1] = temp;

}

}

}

System.out.print("最终排序结果：");

for(int a = 0; a &lt; str.length; a++){

System.out.print(str[a] + "\t");

}

}

}

## 2.多表查询

Seelct \* from User where id in (Select \* from a where id = 1);

Select \* u.username,y.name from user AS u,yong AS y;

SELECT ID,Name FROM Students UNION SELECT ID,Name FROM Teachers

## 3.指针和引用区别

指针指向一块内存，它的内容是所指内存的地址；引用是某块内存的别名。

## 4.http请求和TCP请求

## 1、TCP连接

手机能够使用联网功能是因为手机底层实现了TCP/IP协议，可以使手机终端通过无线网络建立TCP连接。TCP协议可以对上层网络提供接口，使上层网络数据的传输建立在“无差别”的网络之上。

建立起一个**TCP连接需要经**过“三次握手”：

第一次握手：客户端发送syn(SYN(synchronous)是TCP/IP建立连接时使用的握手信号)包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

握 手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连 接之前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连接的请求，断开过程需要经过“四次握手”（过程就不细写 了，就是服务器和客户端交互，最终确定断开）

## 2、HTTP连接

     HTTP协议即超文本传送协议(Hypertext Transfer Protocol )，是Web联网的基础，也是手机联网常用的协议之一，HTTP协议是建立在TCP协议之上的一种应用。

     HTTP连接最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。

     1）在HTTP 1.0中，客户端的每次请求都要求建立一次单独的连接，在处理完本次请求后，就自动释放连接。  
  
     2）在HTTP 1.1中则可以在一次连接中处理多个请求，并且多个请求可以重叠进行，不需要等待一个请求结束后再发送下一个请求。  
  
     由于HTTP在每次请求结束后都会主动释放连接，因此HTTP连接是一种“短连接”，要保持客户端程序的在线状态，需要不断地向服务器发起连接请求。通常的 做法是即时不需要获得任何数据，客户端也保持每隔一段固定的时间向服务器发送一次“保持连接”的请求，服务器在收到该请求后对客户端进行回复，表明知道客 户端“在线”。若服务器长时间无法收到客户端的请求，则认为客户端“下线”，若客户端长时间无法收到服务器的回复，则认为网络已经断开。

**地位对等的协议**，虽然知道TCP是传输层，而http是应用层今天学习了下，知道了 http是要基于TCP连接基础上的，简单的说，TCP就是单纯建立连接，不涉及任何我们需要请求的实际数据，简单的传输。http是用来收发数据，即实际应用上来的。

## 5.进程、线程、端口间关系

进程是指在系统中正在运行的一个应用程序；线程是系统分配处理器时间资源的基本单元，或者说进程之内独立执行的一个单元。对于操 作系统而言，其调度单元是线程。一个进程至少包括一个线程，通常将该线程称为主线程。一个进程从主线程的执行开始进而创建一个或多个附加线程，就是所谓基于多线程的多任务。

端口是计算机与外界通讯的出口，端口将接收到的数据包给到指定的应用程序来处理。进程或是线程可同时监听多个端口。</pre>

3306 mysql

80

oracle 1511

进程是线程的容器，别墅举例，

## 7. 构造器Constructor是否可被override

构造器Constructor不能被继2承，因此不能重写Overriding，但可以被重Overloading。

## 8. collection和collections的区

Collection是集合类的上级接口，继承与他的接口主要有Set 和List. Collections是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

## 9. String与StringBuffer的区别

简单地说，就是一个 变量和常量的关系。StringBuffer对象的内容可以修改；而String对象一旦产生后就不可以被修改，重新赋值其实是两个对象。StringBuffer的内部实现方式和String不同，StringBuffer在进行字符串处理时，不生成新的对象，在内存使用上要优于String类。所以在实际使用时，如果经常需要对一个字符串进行修改，例如插入、删除等操作，使用StringBequffer

在执行速度方面的比较：StringBuilder > StringBuffer

StringBuilder：线程非安全的  
StringBuffer：[线程安全](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%AE%89%E5%85%A8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPW7WPyfdnWm1PjNBP1Rd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjRvnHm3PH04)的

1.如果要操作少量的数据用 = String

2.单线程操作字符串缓冲区 下操作大量数据 = StringBuilder  
3.多线程操作字符串缓冲区 下操作大量数据 = StringBuffer

|  |
| --- |
| **public** **class** StringTest {  **public** **static** **void** main(String[] args) {  **/\*\***  **\* 3、测试字符串是否改变 ,数组会改变，但是 int 和string的不会改变的**  **\*/**  **String str = "abc";**  **StringTest stringTest = new StringTest();**  **int i = 1;**  **int a[] = {1,2,3};**  String abc[] = {"a","b","c"};  stringTest.change(str,i,a,abc);  System.*out*.println(str+i); //a1  System.*out*.println(a[2]); //56  System.*out*.println(abc[2]); } // ef  **void** change(String str){    str = "ghj";  }  }  **void** change(String str,**int** i,**int**[]a,String abc[]){  i = 123;  str = "abc";  a[2] = 56;  abc[2] = "ef";  } |

### 2.测试NULL+”字符串” ，和（）代表整型相加

|  |
| --- |
| /\*\*  \* 2、测试null+字符串的值  \*/  @Test  **public** **void** testnull() {  //  String testString = **null**+"abc";  System.*out*.println(testString);    **int** i = 2222;  String testIntString = "abc"+(i+1);  String testIntStringNo = "abc"+i+1;  System.*out*.println("加括号"+testIntString+"\n"+"不加括号"+testIntStringNo);  }打印输出 ：这里的话就是把null看做了字符串 |

## 10. int和integer 区别

int 是基本数据类型Integer是其包装类，注意是一个类。为什么要提供包装类呢？？？一是为了在各种类型间转化，通过各种方法的调用。否则 你无法直接通过变量转化。比如，现在int要转为String

|  |
| --- |
| **private** **static** **void** IntegerTest() {  String intString = "123";  **int** intA = Integer.*parseInt*(intString); //对象是int类型  Integer intB = Integer.*valueOf*(intString); //对象是 Integer 类型  System.*out*.println((Integer.*valueOf*(intString)).getClass()+""+(Integer.*parseInt*(intString)));  } |

# 12. 运行时异常与一般异常的区

Error和Exception 的父类都是 [**Java**](http://lib.csdn.net/base/17).lang.Throwable到的

Error无法预期的错误因此，这是不可捕捉的，无法采取任何恢复的操作，一般只能显示错误的信息。

va提供了两类主要的异常:运行时异常runtime exception和一般异常checked exception。checked 异常。对于后者这种异常，JAVA要求程序员对其进行catch。所以，面对这种异常不管我们是否愿意，只能自己去写一大堆catch块去处理可能的异常。 运行时异常我们可以不处理。这样的异常由虚拟机接管。出现运行时异常后，系统会把异常一直往上层抛，一直遇到处理代码。如果不对运行时异常进行处理，那么出现运行时异常之后，要么是线程中止，要么是主程序终

## 常见的运行时异常

ArithmeticException、NullPointerException、ClassCaseException、

|  |
| --- |
| /\*\*  \* 1、java.lang.ArithmeticException: / by zero  \*/  @Test  **public** **void** testArithmeticException(){  **int** i = 1/0 ;  }    /\*\*  \* 2、java.lang.ClassCastException  类型转换异常  SonClass SonTwoClass 具有相同的父类，但是他们二者是不能强制转型的  java.lang.ClassCastException: com.hlj.yc.ClassCaseException.SonClass cannot be cast to com.hlj.yc.ClassCaseException.SonTwoClass  \*/  @Test  **public** **void** testClasscastException(){    FatherClass sonClass = **new** SonClass();  FatherClass sonTwoClass = **new** SonTwoClass();    SonTwoClass classOk = (SonTwoClass)sonTwoClass;  //抛出异常 ClassCastException  SonTwoClass classCastExceptionClass = (SonTwoClass)sonClass;      }    /\*\*  3 、NullPointerException  空指针异常  java.lang.NullPointerException at com.hlj.yc.RunningTimeException.testNullPointException(RunningTimeException.java:53)    \*/  @Test  **public** **void** testNullPointException(){    NullPointService nullPointService = **null**;  nullPointService.method();  } |

## 其他的一些场景异常

### 操作[****数据库****](http://lib.csdn.net/base/mysql)异常：SQLException

### 输入输出异常：IOException

### ClassNotFoundException

### FileNotFoundException

## 1只有try没有catch是可以,但是必须有finally

**public** **class** TryTest {

**public** **static** **void** main(String[] args) {

**try** {

System.***out***.println("123s");

} **finally**{

System.***out***.println("456s");

}

}

}

## 2..catch有了return ，finally还执行吗，答案是一定执行

**private** **int** returnSx() {

// **TODO** Auto-generated method stub

**try**

{

a = 2;

**throw** **new** Exception();

}

**catch** (Exception ex)

{

a = 3;

**return** a;

// System.out.println("测试catch的执行"); //不可以写东西了 因为有了return

}

**finally**

{

System.***out***.println("测试finally的执行是不是在return前面");

a = 5;

}

}



**private** **int** returnSx() {

**try**

{

a = 2;

**throw** **new** Exception();

}

**catch** (Exception ex)

{

a = 3;

**return** a;

// System.out.println("测试catch的执行"); //不可以写东西了 因为有了return

}

**finally**

{

System.***out***.println("测试finally的执行是不是在return前面");

a = 5;

**return** a; //如果没有这个则是返回的 3 ，如果加上这个则是返回的5

}

}



## 3. 多个catch执行（子类在前父类在后）



解答:不能编译通过。因为编译的时候会报错:已捕捉到异常 java.io.IOException。  catch(IOException e)这句有错误。  
分析：对于try..catch捕获异常的形式来说，对于异常的捕获，可以有多个catch。对于try里面发生的异常，他会根据发生的异常和catch里面的进行匹配(怎么匹配，按照catch块从上往下匹配)，当它匹配某一个catch块的时候，他就直接进入到这个catch块里面去了，后面在再有catch块的话，它不做任何处理，直接跳过去，全部忽略掉。如果有finally的话进入到finally里面继续执行。换句话说，

如果有匹配的catch，它就会忽略掉这个catch后面所有的catch。对我们这个方法来说，抛出的是IOException，当执行etct.doSomething();时，可能会抛出IOException，一但抛出IOException，它首先进入到catch (Exception e) {}里面，先和Exception匹配，由于OExceptionextends Exception,根据多态的原则，IOException是匹配Exception的，所以程序就会进入到catch (Exception e) {}里面，进入到第一个catch后，后面的catch都不会执行了，所以catch (IOException e) {}永远都执行不到，就给我们报出了前面的错误:已捕捉到异常 java.io.IOException。  
  
总结:在写异常处理的时候，一定要把异常范围小的放在前面，范围大的放在后面，Exception这个异常的根类一定要刚在最后一个catch里面，如果放在前面或者中间，任何异常都会和Exception匹配的，就会报已捕获到...异常的错误。

把子类放前面，因为子类的异常信息比父类更详细

## 4、有了异常会不会继续执行

### 1、有了异常时不会往下执行的，除非捕获catch异常才可以，但是catch中有了return 就不会喽（不包括finally情况）

### 2、写了 throw news 异常，往下写代码会报错，，所以throw之后就不能够写东西了



# 13 hibernate 优点

Hibernate优点 数据持久层   
(1) 对象/关系数据库映射(ORM)  
它使用时只需要操纵对象，使开发更对象化，抛弃了数据库中心的思想，完全的[面向对象思想](https://www.baidu.com/s?wd=%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1%E6%80%9D%E6%83%B3&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnAnzmHDYm1R1m1R4m1T40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWDYPjcLrjfL)  
(2) 透明持久化(persistent)  
带有持久化状态的、具有业务功能的单线程对象，此对象生存期很短。这些对象可能是普通的JavaBeans/POJO，这个对象没有实现第三方框架或者接口，唯一特殊的是他们正与（仅仅一个）Session相关联。一旦这个Session被关闭，这些对象就会脱离持久化状态，这样就可被应用程序的任何层自由使用。（例如，用作跟表示层打交道的数据传输对象。）   
(3) 事务Transaction(org.hibernate.Transaction)  
 应用程序用来指定原子操作单元范围的对象，它是单线程的，生命周期很短。它通过抽象将应用从底层具体的JDBC、JTA以及CORBA事务隔离开。某些情况下，一个Session之内可能包含多个Transaction对象。尽管是否使用该对象是可选的，但无论是使用底层的API还是使用Transaction对象，事务边界的开启与关闭是必不可少的。   
(4) 它没有侵入性，即所谓的[轻量级框架](https://www.baidu.com/s?wd=%E8%BD%BB%E9%87%8F%E7%BA%A7%E6%A1%86%E6%9E%B6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnAnzmHDYm1R1m1R4m1T40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWDYPjcLrjfL)。  
(5) 移植性会很好 可以简单的替换数据库  
(6) 缓存机制，提供[一级缓存](https://www.baidu.com/s?wd=%E4%B8%80%E7%BA%A7%E7%BC%93%E5%AD%98&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvnAnzmHDYm1R1m1R4m1T40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWDYPjcLrjfL)和二级缓存

一级缓存：hibernate的一级缓存是由session提供的

二级缓存：二级缓存的实现原理与一级缓存是一样的。也是通过Key-Value的Map来实现对对象的缓存。二级缓存是作用在SessionFactory范围内的。因此它它可被所有的Session对象所共享。需要注意的是放入缓存中的数据不能有第三方的应用对数据进行修改。  
(7) 简洁的HQL编程    
2. Hibernate缺点   
(1) Hibernate在批量数据处理时有弱势  
(2) 针对单一对象简单的增删查改，适合于Hibernate,而对于批量的修改，删除，不适合用Hibernate,这也是OR框架的弱点；要使用数据库的特定优化机制的时候，不适合用Hibernate **...**

14 Struts的优点有:  
  
    1. 实现MVC模式，   
    2. 自带标签库  
    3.通过一个配置文件，即可把握整个系统各部分之间的联系，这对于后期的维护有着莫大的好处。尤其是当另一批开发者接手这个项目时，这种优势体现得更加明显。  
    4. 提供Exception处理机制 .  
    5. 数据库链接池管理

    缺点:  
  
    五、 对Servlet的依赖性过强. Struts处理Action时必需要依赖ServletRequest 和ServletResponse，所有它摆脱不了Servlet容器。

14AJAX

**异步JavaScript 及XML**（英文：**Asynchronous JavaScript And XML** 缩写**Ajax**）。

是一种基于 [JavaScript](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=2484)和[HTTP](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=26357)请求（HTTP requests），广泛应用在[浏览器](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=5217162)的[网页](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=34859)开发技术

不用刷新页面就可以 连接后台数据库

# 15 iquery

# 16 spring 接管其他所有的框架，实现javaben的注入。

# 17.String 和基本数据类型

Java 中 string 不属于基本的数据类型，只是代表一个类，是一个[引用类型](https://www.baidu.com/s?wd=%E5%BC%95%E7%94%A8%E7%B1%BB%E5%9E%8B&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YdPHc3PjD1rHfvPHFWuhmv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnkP1mkPWf4rjcYn1mYPHmY)。String是[引用类型](https://www.baidu.com/s?wd=%E5%BC%95%E7%94%A8%E7%B1%BB%E5%9E%8B&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YdPHc3PjD1rHfvPHFWuhmv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnkP1mkPWf4rjcYn1mYPHmY)，不过java有[字符串常量](https://www.baidu.com/s?wd=%E5%AD%97%E7%AC%A6%E4%B8%B2%E5%B8%B8%E9%87%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YdPHc3PjD1rHfvPHFWuhmv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnkP1mkPWf4rjcYn1mYPHmY)池机制，声明[字符串常量](https://www.baidu.com/s?wd=%E5%AD%97%E7%AC%A6%E4%B8%B2%E5%B8%B8%E9%87%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YdPHc3PjD1rHfvPHFWuhmv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnkP1mkPWf4rjcYn1mYPHmY)或者[字符串常量](https://www.baidu.com/s?wd=%E5%AD%97%E7%AC%A6%E4%B8%B2%E5%B8%B8%E9%87%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YdPHc3PjD1rHfvPHFWuhmv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnkP1mkPWf4rjcYn1mYPHmY)连接都要用到字符串常量池。如果常量池中有该字符串序列则返回，所以str创建了一个字符串并放入常量池中，str2内的字符序列已在常量池中所以str2直接指向str指向的对象。  
基本数据类型就只有八个，数值型：byte，short，int，long，浮点型：float，double，字符型：char，布尔型：boolean。

# 18 .public privite defult protect



# 19. final

 final修饰的类不能被继承。

 Sting就是一个被final修饰的类，我们只能用，不用继承，可以被实例化

 修饰的方法不能被重写

 final不仅可以修饰类，还可以修饰变量，被final修饰的变量就是一个常量，只能赋值一次

# 20. 抽象类和接口的区别

1.接口是公开的，里面不能有私有的方法或变量，只能使用 public static final，不能够使用private 是用于让别人使用的，而抽象类是可以有私有方法或私有变量的，接口中定义的变量默认是public static final 型，且必须给其初值，（下文有讲解）

**public** **interface** InputConstant {

**public** **static** **final** String *CUSTODIAN\_LICENCETYPE* = "03";  
2.实现接口的一定要实现接口里定义的所有方法，而实现抽象类可以有选择地重写需要用到的方法，一般的应用里，最顶级的是接口，然后是抽象类实现接口，最后才到具体类实现。  
3.还有，接口可以实现[多重继承](https://www.baidu.com/s?wd=%E5%A4%9A%E9%87%8D%E7%BB%A7%E6%89%BF&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYmh7hmHIBuWNBPWu9nWnL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHmLnjmkPW6)，而一个类只能继承一个超类，但可以通过继承多个接口实现[多重继承](https://www.baidu.com/s?wd=%E5%A4%9A%E9%87%8D%E7%BB%A7%E6%89%BF&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYmh7hmHIBuWNBPWu9nWnL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHmLnjmkPW6)，但是可以继承多个接口。常常使用继承单个类和集成多个接口的方式实现类的多重继承。<http://blog.csdn.net/victoryjack/article/details/51615546> 接口还有标识（里面没有任何方法，如Remote接口）和数据共享（里面的变量全是常量）的作

4抽象类不能被实例化

5、接口里面的方法为[抽象方法](https://www.baidu.com/s?wd=%E6%8A%BD%E8%B1%A1%E6%96%B9%E6%B3%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YkmyPbnHNbn1fYPW6dPH790ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHn1rj6LPjRv)，所以写不写abstract 都是一样的。

final修饰的类不能被继承，可以实例化 abstract 修饰的类不能被实例化，能够继承

final修饰的方法不能重写，abstract修饰的方法必须被重写

abstract不能与final并列修饰同一个类。

## 1、接口中的变量默认就是public static final的。而且必须给出值，因为是final的

|  |
| --- |
| **public** **class** InterFaceMain {  **public** **static** **void** main(String[] args) {  /不能重新赋值，因为默认 就是 public static final  //InterfaceMe.a = 5;  System.*out*.println(InterfaceMe.*a*);  }  }  **interface** InterfaceMe{  **int** *a* = 1;  } |

## 2、类实现多重继承

解释：一个类只能继承一个超类，但可以通过继承多个接口实现[多重继承](https://www.baidu.com/s?wd=%E5%A4%9A%E9%87%8D%E7%BB%A7%E6%89%BF&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YYmh7hmHIBuWNBPWu9nWnL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHmLnjmkPW6)

|  |
| --- |
| interface CanFight{  void fight();  }  interface CanSwim{  void swim();  }  interface CanFly{  void fly();  }  class ActionCharacter{  public void fight(){  System.out.println("I can fight!");  }  }  class Hero extends ActionCharacter implements CanFight,CanSwim,CanFly{  public void swim(){  System.out.println("I can swim!");  }  public void fly(){  System.out.println("I can fly!");  }  }  public class Adventure {  public static void t(CanFight x){x.fight();}  public static void u(CanSwim x){x.swim();}  public static void v(CanFly x){x.fly();}  public static void w(ActionCharacter x){x.fight();}  public static void main(String[] args){  Hero h=new Hero();  t(h);  u(h);  v(h);  w(h);  }  } |

## 3、接口和抽象继承接口

### 1、接口

|  |
| --- |
| **public** **interface** AbstractInterface {    **public** **abstract** **int** a();  **int** b();  } |

### 2、抽象类继承接口

|  |
| --- |
| **public** **abstract** **class** AbstractClass **implements** AbstractInterface {  /\*\*  \* 其实这里已经具备了抽象方法 （因为接口中的方法本身就是抽象的）  \*  \*/  } |

### 3、类继承抽象类

|  |
| --- |
| **public** **class** AbstractSon **extends** AbstractClass{  /\*\*  \* 来自接口中的方法（也是来自抽象中的方法）  \*/  @Override  **public** **int** a() {  **return** 0;  }  @Override  **public** **int** b() {  **return** 0;  }  } |

### 4、解释：

#### 1、接口里面的方法为[抽象方法](https://www.baidu.com/s?wd=%E6%8A%BD%E8%B1%A1%E6%96%B9%E6%B3%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YkmyPbnHNbn1fYPW6dPH790ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHn1rj6LPjRv)，所以写不写abstract 都是一样的。

#### 2、一个类中，注意是一个类，不是接口哦，只要有一个抽象方法，这个类必须被声明为抽象类

## 4、abstract方法是不能声明为synchronized的，应该在子类中提供该方法的实现时，再声明synchronized。原因很简单，你在abstract类中尚不知道如何实现该方法，哪里来的同步需求呢？

而且abstract 不能写成native（Oject类中就有） native关键字说明其修饰的方法是一个原生态方法，方法对应的实现不是在当前文件，而是在用其他语言（如C和C++）实现的文件中

# 21 list set map

List特点：元素有放入顺序，元素可重复   
Map特点：元素按键值对存储，无放入顺序 key不能重复，但是value可以重复  
Set特点：元素无放入顺序，元素不可重复（注意：元素虽然无放入顺序，但是元素在set中的位置是有该元素的HashCode决定的，其位置其实是固定的）

## 1、解释list放入必须有顺序

|  |
| --- |
| **public** **class** ArrayListMain {    **public** **static** **void** main(String[] args) {  List<Person> persons= **new** ArrayList<Person>();  Person person = **new** Person(0, "HealerJean");  Person person2 = **new** Person(0, "HealerJean");  persons.add(0,person);  persons.add(3,person2); //报错，因为没有 2  }  } |

# 22.ArrayList和LinkedList的大致区别：

List接口有三个实现类：LinkedList，ArrayList，Vector   
LinkedList：底层基于链表实现，链表内存是散乱的，每一个元素存储本身内存地址的同时还存储下一个元素的地址。链表增删快，查找慢   
ArrayList和Vector的区别：ArrayList是非线程安全的(因为Vector的方法前加了，synchronized 关键字，也就是同步的意思)，效率高；Vector是基于线程安全的，效率低     Vector与ArrayList一样，也是通过数组实现的，不同的是它支持线程的同步

1.ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。   
     2.对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。   
     3.对于新增和删除操作add和remove，LinedList**比较**占优势，因为ArrayList要移动数据。

# 21数组和集合的主要区别

最大的区别就是数组是大小静态的而集合的大小动态的  
数组和集合相比唯一的有点就是速度快..

数组声明了它容纳的元素的类型，而集合不声明。这是由于集合以object形式来存储它们的元素。

22.

# 22. &和&&的区别？     &是位运算符，表示按位与运算，&&是逻辑运算符，表示逻辑与（and）。

# 23 final, finally, finalize的区别？    final 用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。     finally是异常处理语句结构的一部分，表示总是执行。     finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。

# 24. forward 和redirect的区别

1）forward是容器中控制权的转向，是服务器请求资源，服务器直接访问目标地址的URL，把那个URL 的响应内容读取过来，然后把这些内容再发给浏览器，浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来的地址。

2）redirect就是服务器端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址，因此从浏览器的地址栏中可以看到跳转后的链接地址。一般来说浏览器会用刚才请求的所有参数重新请求，所以session,request参数都可以获取。

3）前者更加高效，在前者可以满足需要时，尽量使用转发（通过RequestDispatcher对象的forward方法，RequestDispatcher对象可以通过ServletRequest对象的getRequestDispatcher方法获得），并且，这样也有助于隐藏实际的链接；在有些情况下，比如，需要跳转到一个其它服务器上的资源，则必须使用重定向（通过HttpServletResponse对象调用其sendRedirect方法）。

response.sendRedirect("http://www.baidu.com");

request.getRequestDispatcher("目标页面").forward(response,request);

**request.getRequestDispatcher()是请求转发，前后页面共享一个request  是服务器端跳转   
  
         response.sendRedirect()是重新定向 前后页面不是一个request 是客户端跳转**

　forward直接转发就相当于：“A找B借钱，B说没有，B去找C借，借到借不到都会把消息传递给A”；

　Redirect间接转发就相当于："A找B借钱，B说没有，让A去找C借"。

# 25. .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。

# 26. == 和equals

# 可以把==想象成人的长相是否一样 把equals想象成人的本质是否一样 ==比较两个地址是不是一样的 equals比较两个地址指向的 实际对象的内容 是不是同一个

关键在于。。。你要理解s1=new String("we are students");  
 s2="We are students";  
这两个是不一样的。。。。new是用新建一个对象的方法。。。。而后一种是放在常量池中的。。。你一下次再用这种方法去创建的时候。。。就会去常量池里面找，所以  
s2="We are students";  
 s3="We are students";  
  
这两个是相同的。。。指向常量池中的"We are students";。。。  
  
而s1=new String("we are students");  
 [s4](https://www.baidu.com/s?wd=s4&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YzPWPbP1u9nHwWP1mdnH6d0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmzP1f4rHR3)=new String(s1);  
这两个是不同的对象。。。。。尽管他们的内容是一样的。。。  
就像有两个人。。。他们的名字是相同的。。。。但是并不是一样的人。。。

**public** **static** **void** main(String[] args) {

*dy*();

}

**public** **static** **void** dy(){

String a = "132456";

String a2 = "132456";

String abuilder = **new** String("123456");

String abuilder2 = **new** String(abuilder);

String ab = **new** String(abuilder);

**if**(a==a2){ //a a2 地址相同

System.***out***.println("a==a2"); //成立

}**else** {

System.***out***.println("a !=a2");

}

**if**(a==abuilder){//a abuilder 地址不同

System.***out***.println("a==abuilder");

}**else** {

System.***out***.println("a !=abuilder");//成立

}

**if**(a.equals(abuilder)){ //地址所指向的对象 不同

System.***out***.println("a==abuilder");

}**else** {

System.***out***.println("a !equals =abuilder");//成立

}

**if**(abuilder==abuilder2){ //地址不同

System.***out***.println("abuilder = abuilder2");

}**else** {

System.***out***.println("abuilder != abuilder2"); //成立

}

**if**(abuilder.equals(abuilder2)){

System.***out***.println("abuilder equals= abuilder2"); //成立

}**else** {

System.***out***.println("abuilder !equals= abuilder2");

}

# 详解== 和equels

## 1、首先明白String和对象的equals是不一样的，String重写了Object的equals方法和hashcode方法,String String类中的equals()方法很明显是仅仅进行了对象内容的比较，而没有比较对象存储地址。所以才会出现上面的东西

### 1、String重写的equals方法

|  |
| --- |
| **public** **boolean** equals(Object anObject) {  **if** (**this** == anObject) {  **return** **true**;  }  **if** (anObject **instanceof** String) {  String anotherString = (String) anObject;  **int** n = value.length;  **if** (n == anotherString.value.length) {  **char** v1[] = value;  **char** v2[] = anotherString.value;  **int** i = 0;  **while** (n-- != 0) {  **if** (v1[i] != v2[i])  **return** **false**;  i++;  }  **return** **true**;  }  }  **return** **false**;  } |

### 2、string中的hashcode

|  |
| --- |
| **public** **int** hashCode() {  **int** h = hash;  **if** (h == 0 && value.length > 0) {  **char** val[] = value;  **for** (**int** i = 0; i < value.length; i++) {  h = 31 \* h + val[i];  }  hash = h;  }  **return** h;  } |

## 2、关于Object的equals方法，可以看他的源码，默认情况 下 equals()和hashCode()都是从Object类中继承而来的，而Object中equals()默认实现的是比较两个对象是否==，即默认的equals()和==的效果是一样的，所以当new出对象的之后，很明显地址就是不一样的，所以equals 肯定是不相等的

|  |
| --- |
| **public** **boolean** equals(Object obj) {  **return** (**this** == obj);  } |

## 3、理解hashcode和内存地址，每个对象new出来的时候，内存地址就是不一样的。一般情况下new出来的对象hashcode值也是不一样的，有时候特殊情况（Hashtable），即使hashcode值一样，对象存储的位置可以根据hashcode找到对象所住的房子(单链表)，但是房子里有各种对象，也就是地址不一样。equals相等，hashcode值肯定相同，hashcode值相同，equals不一定相等

**4、 == 比较对象的相等不仅要比较对象内容相等，还要比较对象引用地址是否相等。在我看来Object==了，必然equals**

**5、测试**

|  |
| --- |
| Person person1 = **new** Person();  Person person2 = **new** Person();    System.*out*.println(person1.hashCode()+"\n"+person2.hashCode()); // 不相等  System.*out*.println(person1==person2); //false  System.*out*.println(person1.equals(person2)); //false |

## 27[四种xml的解析方式](http://blog.csdn.net/heaven_of_relaxing/article/details/51627279)

四种解析方式又大致可以分为两种**：一种是基于树结构处理的Dom解析，另外一种是基于事件模型的SAX解析**

### **DOM（Document Object Model)**

      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或信息片断的集合。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，**然后才能做任何工作。由于它是基于信息层次的，因而DOM被认为是基于树或基于对象的。

【优点】  
      ①允许应用程序对数据和结构做出更改。  
      ***②访问是双向的，可以在任何时候在树中***上下导航，获取和操作任意部分的数据。  
【缺点】  
      ①通常需要加载整个XML文档来构造层次结构，消耗资源大。

### **SAX（Simple API for XML)**

     SA**X处理的优点非常类似于流媒体的优点。分析能够立即开始，而不是等待所有的数据被处理。而且，由于应用程序只是在读取数据时检查数据，因此不需要将数据**[**存储**](http://www.storworld.com/)**在内存中。这对于大型文档来说是个巨大的优点**。事实上，应用程序甚至不必解析整个文档；它可以在某个条件得到满足时停止解析。一般来说，SAX还比它的替代者DOM快许多。

①不需要等待所有数据都被处理，分析就能立即开始。  
     ②只在读取数据时检查数据，不需要保存在内存中。  
     ③可以在某个条件得到满足时停止解析，不必解析整个文档。  
     ④效率和性能较高，能解析大于系统内存的文档。

【缺点】  
     ①需要应用程序自己负责TAG的处理逻辑（例如维护父/子关系等），文档越复杂程序就越复杂。  
     ②**单向导航，无法定位文档层次，很难同时访问同一文档的不同部分数据，不支**持XPath。

# 28具体实现XML的三种方式，XSL是可扩展样式表语言的外语缩写

1。第一种方式为：

　　让XML文档和与其关联的XSL级联样表同时被传送到客户端（通常使用的是浏览器）,然后在客户端让XML文档根据XSL定  
义的显示格式显示其内容。

2。第二种方式为：

　　在服务端就使用XSL级联样表转换XML文档为其他的格式（通常为HTML格式）然后在把转换过的文档传送给客户端（一  
般使用浏览器）

3。第三钟方式为：

使用第三方的产品，在将XML文档放到服务端之前就将该文档转换成其他的格式（一般为HTML格式）。然后服务端和客  
户端就和平常处理HTML一样来处理了。

# 29.ioc 和aop

## 40、什么是IoC和DI？DI是如何实现的？  答：IoC叫控制反转，是Inversion of Control的缩写，控制反转是把传统上由程序代码直接操控的对象的调用权交给容器，通过容器来实现对象组件的装配和管理。所谓的"控制反转"就是对组件对象控制权的转移，从程序代码本身转移到了外部容器，由容器来创建对象并管理对象之间的依赖关系。

控制反转——Spring通过一种称作控制反转（IoC）的技术促进了松耦合。当应用了IoC，一个对象依赖的其它对象会通过被动的方式传递进来，而不是这个对象自己创建或者查找依赖对象，是容器在对象初始化时不等对象请求就主动将依赖传递给它。通过IOC反转控制DI依赖注入完成各个层之间的注入，使得层与层之间实现完全脱耦，增加运行效率利于维护。

## 41、解释一下什么叫AOP（面向切面编程）？ 注意结合sino，spring配置文件 答：     spring的AOP面向切面编程，实现在不改变代码的情况下完成对方法的增强。比较常用的就是spring的声明式事务管理，advisor底层通过AOP实现，避免了我们每次都要手动开启事物，提交事务的重复性代码，使得开发逻辑更加清晰。

   简单点解释，比方说你想在你的service层所有类中都加上一个打印‘你好’的功能这你经可以用aop思想来做，你先写个类写个方法，方法经实现打印‘你好’然后你Ioc这个类 ref＝“service.\*”让每个类都注入。

   aop就是面向切面的编程。比如说你每做一次对数据库操作，都要生成一句日志。如果，你对数据库的操作有很多类，那你每一类中都要写关于日志的方法。但是如果你用aop，那么你可以写一个方法，在这个方法中有关于数据库操作的方法，每一次调用这个方法的时候，就加上生成日志的操作。

1.控制反转（IoC）与依赖注入（DI）是同一个概念把创建对象和维护对象的关系的权利，从程序中转移到我们的spring(applicationContext.xml),容器中去。而程序本身不再维护，而传统的时候是需要进行实例化的，不需要再程序中进行配置，学习框架就是学习配置

## 作用：在雨花石学到的

**2.AOP，即面向切面编程，面向切面编程的目标就是分离关注点。什么是关注点呢，就是你要做的事，就是关注点。**假如你是个公子哥，没啥人生目标，天天就是衣来伸手，饭来张口，整天只知道玩一件事！那么，每天你一睁眼，就光想着吃完饭就去玩（你必须要做的事），但是在玩之前，你还需要穿衣服、穿鞋子、叠好被子、做饭等等等等事情，这些事情就是你的关注点，但是你只想吃饭然后玩，那么怎么办呢？这些事情通通交给别人去干。在你走到饭桌之前，有一个专门的仆人A帮你穿衣服，仆人B帮你穿鞋子，仆人C帮你叠好被子，仆人C帮你做饭，然后你就开始吃饭、去玩（这就是你一天的正事），你干完你的正事之后，回来，然后一系列仆人又开始帮你干这个干那个，然后一天就结束了！  
  
AOP的好处就是你只需要干你的正事，其它事情别人帮你干。也许有一天，你想裸奔，不想穿衣服，那么你把仆人A解雇就是了！也许有一天，出门之前你还想带点钱，那么你再雇一个仆人D专门帮你干取钱的活！**这就是AOP。每个人各司其职，灵活组合，达到一种可配置的、可插拔的程序结构。**  
  
**从Spring的角度看，AOP最大的用途就在于提供了事务管理的能力。事务管理就是一个关注点，你的正事就是去访问数据库，而你不想管事务（太烦），所以，Spring在你访问数据库之前，自动帮你开启事务，当你访问数据库结束之后，自动帮你提交/回滚事务！**

## 2.spring ioc有几种注入方式？

（1）setter方法注入

（2）构造器注入

（3）静态工厂注入

（4）无配置文件注入（自动注入）

# 30.hibernate 简单操作

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<!--该文件要清楚地表述出 类 和 表 的对应关系-->

<!-- 映射文件需要一个dtd来制定格式-->

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<!-- package : 表示该类在哪个包下 -->

<hibernate-mapping package=*"com.hsp.domain"*>

<!-- name : 表示类名 table 表示 该类和哪个表映射 -->

<class name=*"Employee"* table=*"employee"*>

<!-- id元素专门用于指定主键是如何生成,hibernate设计者认为，我们每一个表都应该有一个主键 -->

<!-- name:表示类的哪个属性是主键 -->

<id name=*"id"* type=*"java.lang.Integer"*>

<generator class=*"increment"*/>

<!-- 指定主键生成策略 -->

<!--

<generator class="sequence">

<param name="sequence">emp\_seq</param>

</generator> -->

</id>

<property name=*"name"* type=*"java.lang.String"*>

<column name=*"name"* not-null=*"true"*/>

</property>

<property name=*"email"* type=*"java.lang.String"*>

<column name=*"email"* not-null=*"true"*/>

</property>

<property name=*"hiredate"* type=*"java.util.Date"*>

<column name=*"hiredate"* not-null=*"true"*/>

</property>

</class>

</hibernate-mapping>

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<!-- 这个我们自己配置的 -->

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- 将来改变的数据库的时候，只要改了下面这个就是可以了啦 -->

<!-- 该文件用于配置连接数据的种类,用户名，密码,ul ,驱动.. 连接池,二级缓存.. 有点类似strus struts-config.xml -->

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/hiberate1</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>147094</property>

<!-- 配置数据库的方言/ -->

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<!-- 配置显示hibernate生成的 sql ,特别说明，在开发阶段设为true利于调试，在使用项目则设为false--><property name=*"show\_sql"*>true</property>

<!-- 配置管理的对象映射文件 -->

<mapping resource=*"com/hlj/domain/Employee.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

public class TestMain {

public static void main(String[] args) {

// TODO Auto-generated method stub

//对数据持久层操作

//1. 该类主要是用于读取hibernate.cfg.xml

Configuration cf=new Configuration().configure();

//2. 创建一个会话工厂.是一个重量级的对象

SessionFactory sf=cf.buildSessionFactory();

//3. 创建Session(会话)<->Connection

Session session=sf.openSession();

//4.没有下面事务，下面的东西是提交不了，也就是保存不到数据中的 ，

//开始事务

Transaction ts=session.beginTransaction();

//加入一个对象(记录)Employee.java

Employee u=new Employee();

u.setEmail("1318830916@qq.com");

u.setHiredate(new Date());

u.setName("zhangyujin");

//insert

//保存，

session.save(u);//添加该对象到数据库 //等于==inset into 。。被hibernate封装，将来不管用什么数据库都是可以试实现的

ts.commit();//提交事务

session.close();

}

}

# 31.hibernate 小知识

# 1.数据库切换

<!-- 如果数据库中没有表，让这个创建表 -->

2.1<property name="hbm2ddl.auto">create</property>

，这个是每次都会重新创建，有表就是删除再创建

2.2所以应该为update，没有表创建表，有的话，看看结构有没有变化，

<property name="hbm2ddl.auto">update</property>

## 2. openSession()getCurrentSession ()

1. openSession() 是获取一个新的session，必须在commit 和rollback 时候手动关闭，使用它查询时候是不需要事物的，

2.getCurrentSession () 获取和当前线程绑定的session,换言之，在同一个线程中，我们获取的session是同一session,这样可以利于事务控制，而且安全，这个如果要使用的话是需要在后台的配置文件中配置的，但是有了工具类，就不需要配置了，这个的关闭是系统自动的，无需手动操作。使用它必须使用事物。

//获取和线程关联的session

public static Session getCurrentSession(){

Session session=threadLocal.get();

//判断是否得到了session 如果没有得到就开始 让他得到

if(session==null){

session=sessionFactory.openSession();

//把session对象设置到 threadLocal,相当于该session已经和线程绑定

//这样就可以不要cfg 中的配置了

// <property name="current\_session\_context\_class">thread</property>

threadLocal.set(session);

}

return session;

}

## 3. get和load的区别

1.get c查不到就会返回nul，先到缓存中去查，如果查到就返回对象，如果没有就直接查询数据库，

2.load 查不到就会抛出异常，先打缓存中去查，如果查到就返回对象，如果查不到，先不去数据库中查，然后等到真正用的时候再去数据库中查，这个也就是懒加载的使用

总结，如果是我们知道数据库中一定有数据，那么就是使用load，如果不确定就是用get、



## 4.hibernate查询数据库语句

Query query=session.createQuery("from Employee where id='6'");

where 后面的条件可以是类的属性名，也可以是表的字段,安照hibernate规定，我们还是应该使用类的属性名.

## 5.缓存机制

private static void testLoad(){

Session session = MySessionFactory.getSessionFactory().openSession();

//下面这个执行之后会 将这个缓存放到session 缓存/二级缓存中

Employee employee1 = (Employee)session.load(Employee.class,5);

System.out.println(employee1);

//下面这个不会执行db 在二级缓存中查找到的，下面这个缓存之后 如果长时间使用只会，会放到一级缓存中去

Employee employee2 = (Employee)session.load(Employee.class,5);

System.out.println(employee2);

Employee employee3 = (Employee)session.get(Employee.class,5);

System.out.println(employee3);

}

上面的东西 只是执行一个sql语句

## 6.主外键设置

1. 我们的学生表student中出现了下面的东西，就是因为他里面有数据被当做了外键,

**private Set stucourses = new HashSet(0);**

2. 选课表Stucourse的变化最大，外键直接变成对象了，

**p**rivate Integer stuCourseId;

**private Student student;**

**private Course course;**

private Integer grade;

3.学生表的映射文件中出现多对一，因为是多个学生可以选择同一个学生选课表

**<set name="stucourses" inverse="true">**

**<key>**

**<column name="sid" not-null="true" />**

**</key>**

**<one-to-many class="com.hlj.domain.Stucourse" />**

**</set>**

4.学生选课表映射文件中

**<many-to-one name="student" class="com.hlj.domain.Student"fetch="select">**

**<column name="sid" not-null="true" />**

**</many-to-one>**

5.解析其中数据

**createQuery("select student.sname,course.cname,grade from Stucourse where grade>=60").list();**

## 7.解析数据

**1. Iterator<Student> iterator = list.iterator();**

**while** (iterator.hasNext()) {

Student student = iterator.next();

System.*out*.println("iterator姓名"+student.getSname()+"性别"+student.getSsex());

**2. Query query=session.createQuery("from Student");**

List<Student> list=query.list();

for(Student e: list){

System.out.println("姓名"+e.getSname()+"性别"+e.getSsex());

}

## 8.API常用接口

**基本操作类**：用来完成对数据库增、删、改、查的接口。例如：Session、Transaction和Query；

**配置类**：Hibernate用例读取配置文件和映射文件的接口。例如：SessionFactory、Configuration；

**回调接口**：它允许应用程序能对一些事件的发生作出相应的操作。例如：Interceptor、Validatable；

**扩展映射机制接口**：UserType、CompositeUserType、IdentifierGenerator

**Hibernate六个常用接口**

Hibernate的类非常多，但我们常用的也就六个，分别是：

Configuration、SessionFactory、Session、Transaction、Query、Criteria。这六个核心接口是我们必须了解和掌握的。通过这六个接口，我们可以完成对数据的基本操作，包括增、删、改、查，还有事务控制。

这六个接口之间的产生关系如下：

[](http://images2015.cnblogs.com/blog/776887/201510/776887-20151010185752253-2014139636.jpg)

# 32. ：[JSP九大内置对象的作用和用法总结？](http://blog.csdn.net/sona_shi555/article/details/7797068)

request、response、session、application、**out**、pagecontext**、config**、**page、exception**

# 33 类和对象的区别

**类是一个抽象的概念；  
而对象是类抽象概念的实物表达；**  
打个比方：  
比如水果摊进了一批水果（就好比是类）；  
然后就去问卖家有哪些新鲜的水果；  
店家说有苹果、梨、桃等等（这里的苹果、梨、桃就是对象）；  
也就是说对象是类的具体表达；  
而类则是对象的抽象表达。

# 34、面向对象和面向过程

[面向过程](https://www.baidu.com/s?wd=%E9%9D%A2%E5%90%91%E8%BF%87%E7%A8%8B&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nv7hm1TdnWTYrjfLuAnk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW03rH01P0)就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了。   
  
[面向对象](https://www.baidu.com/s?wd=%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nv7hm1TdnWTYrjfLuAnk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW03rH01P0)是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为。

面向过程认为世界是连续的，组成世界的是一个个过程，**对象是这些过程中传递的信息；**  
面向对象认为世界是离散的，组成世界的是一个个对象，**过程是这些对象间发生的关系**

# 35. stringbuffer

charAt(2) indexOf substring replase equalsIgnoreCase lastIndexOf startsWith toLowerCase toUpperCase

两边空格 -- trim();  
中间空格 -- replace(" ","")

**public** **static** **void** Testsplit() {

// **TODO** Auto-generated method stub

String a = "abcdef";

String []aStrings = a.split("");

System.***out***.println(aStrings[0]+"\*"); //输出\* 表示索引为0的时候，不存在数据

System.***out***.println(aStrings[6]); //输出f

**for**(String b:aStrings){

System.***out***.println(b); //a b c d e f

}

}

# 36. static（不能在方法中出现）

**被static修饰的成员变量和成员方法独立于该类的任何对象。也就是说，它不依赖类特定的实例，被类的所有实例共享。（static 不能再static）**

**只要这个类被加载，Java虚拟机就能根据类名在运行时数据区的方法区内定找到他们。因此，static对象可以在它的任何对象创建之前访问，无需引用任何对象。**

**因此静态方法中不能用this和super关键字，不能直接访问所属类的实例变量和实例方法(就是不带static的成员变量和成员成员方法)，只能访问所属类的**[**静态成员**](https://www.baidu.com/s?wd=%E9%9D%99%E6%80%81%E6%88%90%E5%91%98&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvuW9BuHubnHT3PyD1PHcL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWbYPHDvn163)**变量和成员方法。因为static方法独立于任何实例，因此static方法必须被实现，而不能是抽象的abstract。所以叫类变量**

### 2.static执行顺序，因为在虚拟机的生命周期中一个类只被加载一次；又因为static{}是伴随类加载执行的，所以，不管你new多少次对象实例，static{}都只执行一次。

**static{}语句块的执行次序**

**1、**当一个类中**有多个static{}的时候，按照static{}的定义顺序，从**前往后执行；

**2、**先**执行完static{}语句块的内容，才会执行调用语句；**

**示例二**

public class TestStatic{  
    static{  
        System.out.println(1);  
    }  
    static {  
        System.out.println(2);  
    }  
    static {  
        System.out.println(3);  
    }  
    public static void main(String args[]){  
        System.out.println(5);  
    }  
    static {  
        System.out.println(4);  
    }  
}  
**结果**:程序会输出1，2，3，4，5

3、如果静态变量在定义的时候就赋给了初值(如 static int X=100)，那么赋值操作也是在类加载的时候完成的，并且当一个类中既有static{}又有static变量的时候，同样遵循“先定义先执行”的原则；

**示例三**

 class Test{  
 public static int X=300;  
 static{  
  System.out.println(X);  
  X=200;  
  System.out.println(X);  
 }  
}

public class StaticBlockTest{  
 public static void main(String args[]){  
  System.out.println(Test.X);  
 }  
}

结果:程序会依次输出300，200，200，先执行完X=300，再执行static{}语句块。

**public** **class** TestStatic {

**public** **static** **int** *n* = 1;

**int** m = 2;

**public** **static** **void** main(String[] args) {

TestStatic testStatic = **new** TestStatic();

testStatic.b1();

}

**public** **static** **void** a(){

/\* 错误的

\* this.b2();

this.m = 2;\*/

*a3*();

*n* = 2;

//错误的

//m = 3;

System.***out***.println(TestStatic.*n*);

System.***out***.println("a");

}

**public** **static** **void** a3(){

}

**public** **void** a2(){

**this**.a2();

}

**public** **void** b1(){

a2(); 可以直接调用非静态的

*a*();

TestStatic.*a*();

// TestStatic.a2(); //这样是不可以的

**this**.*a*();

**this**.m = 1;

**this**.*n*=2;

*n* = 2;

m = 3;

System.***out***.println("b");

}

**public** **static** **void** b2(){

System.***out***.println("b2");

}

}

# 37.super，this,(构造必须放在第一行)





## super

## 1、父类

|  |
| --- |
| **public** **class** SupperFather {  **public** SupperFather(){  System.*out*.println("SupperFather supper constructor");  }    **public** SupperFather(String str){  System.*out*.println("SupperFather supper Param constructor "+ str);  }    **public** **void** otheMethod(){  System.*out*.println("SupperFather Method Test Supper");  }  } |

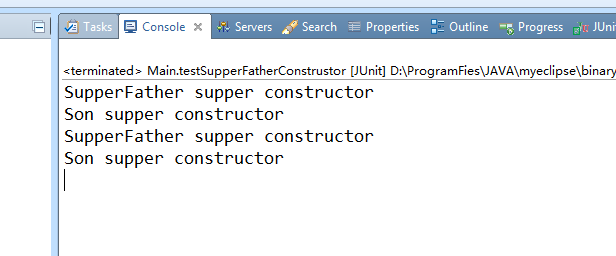
## 2、子类

|  |
| --- |
| **public** **class** Supper **extends** SupperFather{      /\*\*  \* 一、无参构造器  \* 1、写不写 super(); 都会调用父类中的无参构造器  \*/  **public** Supper(){  **super**();  System.*out*.println( "Son supper constructor");  //Supper supper = new Supper(); //死循环 还会调用父类  }    /\*\*  \* 二、有参构造器  1、下面 写super() 和 super(str) 会调用父类中对应的构造器  2、不写 suppr时候，像无参构造器那样自动执行 父类中supper()  \*/  **public** Supper(String str){  // 1、什么都不写会主动调用父类中的 supper()  // 2、super(str); 会调用父类中的有参构造器 ，不会调用无参构造器  System.*out*.println( "Son supper param constructor");  }      /\*\*  \* 三、不是构造器的方法  1、 super() 等 相同类型 只能用在构造器中  2、super.父类方法 可以放在任意方法的任意位置  \*/  **public** **void** otheMethod(){  **super**.otheMethod();  System.*out*.println("Supper Method Test Supper");  **super**.otheMethod();  } |

## 3、main函数测试

### 1、无参构造器测试

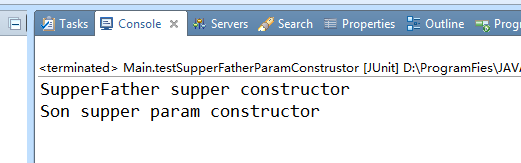
|  |
| --- |
| @Test  **public** **void** testSupperFatherConstrustor() {  Supper supper = **new** Supper(); //和下面是一样的  SupperFather supperFather = **new** Supper();  } |



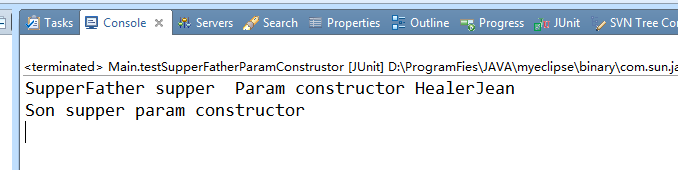
### 2、有参构造器文档

|  |
| --- |
| **public** **class** Main {    @Test  **public** **void** testSupperFatherParamConstrustor() **throws** Exception {  String str = "HealerJean";  SupperFather supperFatherParam = **new** Supper(str);  } |

#### 1、子类中 有super()或者 什么没写super()时

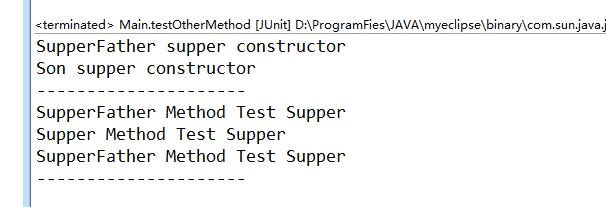


#### 2、子类中 第一行是 super(str)时候



### 3、其他类型的函数，子类中不可以使用super（…）

|  |
| --- |
| @Test  **public** **void** testOtherMethod() {  SupperFather supperFatherParam = **new** Supper();  supperFatherParam.otheMethod();  }  } |



## this 必须放在第一句 ，其实同super()

# 38.类加载

1、用Class.forName()显示加载的时候，如上面的示例一;

2、实例化一个类的时候，如将main()函数的内容改为:Test t=new Test();//这种形式其实和1相比，原理是相同的，都是显示的加载这个类，读者可以验证Test t=new Test();和Test t=(Test)Class.forName().newInstance();这两条语句效果相同。

3、调用类的静态方法的时候，如将main()函数的内容改为:Test.display();

4、调用类的静态变量的时候，如将main()函数的内容改为:System.out.println(Test.X);

# 39. Hibernate一级缓存

1.Hibernate一级缓存又称为“Session的缓存”。

Session内置不能被卸载，Session的缓存是事务范围的缓存（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）。

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，默认下SessionFactory不会启用这个插件。

# 40.java和j2ee

java 只是一个程序设计语言，[J2EE](https://www.baidu.com/s?wd=J2EE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)是基于java为语言的一个[B/S架构](https://www.baidu.com/s?wd=B%2FS%E6%9E%B6%E6%9E%84&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)框架，  
 说的详细点，  
 java是一门程序设计语言，有自己的编译系统和运行在[操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)上的虚拟机，我们可以运用java来编写任何的软件。可以是基于[B/S架构](https://www.baidu.com/s?wd=B%2FS%E6%9E%B6%E6%9E%84&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)，也可以是[C/S架构](https://www.baidu.com/s?wd=C%2FS%E6%9E%B6%E6%9E%84&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)的，也可以是任意的桌面程序。  
 现在的软件越来越向着[B/S架构](https://www.baidu.com/s?wd=B%2FS%E6%9E%B6%E6%9E%84&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)发展，而[J2EE](https://www.baidu.com/s?wd=J2EE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)是一个[java框架](https://www.baidu.com/s?wd=java%E6%A1%86%E6%9E%B6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)，是由开发商（应该是[sun公司](https://www.baidu.com/s?wd=sun%E5%85%AC%E5%8F%B8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)自己开发的）写好的一系列java代码，这样程序员在运用java来开发B/S架构的程序软件时便可以直接运用[J2EE](https://www.baidu.com/s?wd=J2EE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PWf4uWnkn1uhrjuWPHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHf1P1TLnjbs)框架，从而更方便的开发B/S架构软件程序。

# 41 [pageContext、request、session和application区别](http://blog.csdn.net/u010708434/article/details/18183173)

  我们经常会将pageContext、request、session和application混为一谈，因为它们都可以通过setAttribute()和getAttribute()来设定或读取属性。但它们之间是有别的，它们最大的区别在于使用范围。   
  
pageContext对象的范围只适用于当前页面范围，即超过这个页面就不能够使用了。所以使用pageContext对象向其它页面传递参数是不可能的。   
  
request对象的范围是指在一JSP网页发出请求到另一个JSP网页之间，随后这个属性就失效。   
  
session的作用范围为一段用户持续和服务器所连接的时间，但与服务器断线后，这个属性就无效。比如断网或者关闭浏览器。   
  
application的范围在服务器一开始执行服务，到服务器关闭为止。它的范围最大，生存周期最长。

# 42session 和cookie的区别

相同点：cookie和session的共同之处在于：cookie和session都是用来跟踪浏览器用户身份的会话方式。  
不同点 cookie 和session的区别是：cookie数据保存在客户端，session数据保存在服务器端

自己：cookie是明文，不安全，别人利用cookie可以被篡改和攻击；而session存放服务器缓存中并且加密的，其他用户看不到。 。cookie数据保存在客户端，session数据保存在服务器端，其实Session同时存在于客户端与服务器端。但是Session真正的数据是存储在服务器上的 每个session都 对应了一个唯一的sessionId ，在浏览器上使用Cookie来存储这个sessionid的，删除Cookie session失效，关闭浏览器Cookie失效，也就是session失效。所以使用session必须支持cookie

1、cookie是明文，不安全，别人利用cookie可以被篡改和攻击；而session存放服务器缓存中并且加密的，其他用户看不到。

2、虽然Session真正的数据是存储在服务器上的,但每个Session都对应了一个由Web服务器指定的唯一识别符SeesionID,而在[浏览器](https://www.baidu.com/s?wd=%E6%B5%8F%E8%A7%88%E5%99%A8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YznHN-rjR3nHfLuAfYujR40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPH6YrHcLrjcv)里是使用Cookie来存储这个SeesionID的.所以使用Session,浏览器必须支持Cookie.

1.、传说中，Web应用程序中的Session和Application保存服务器端，而cookie保存在客户端。

其实Session同时存在于客户端与服务器端。

开发中如果你已经写了一个Session，打开火狐浏览器，清理cookie的时候你会找到一个session\_id（java开发者会看到JSESSIONID）的cookie，删除他后Session将失效。

如果你打开的是IE浏览器，清理cookie后Session依然存在，我不知道IE是怎么处理的，也许它只从磁盘里去清理。

**2、Session实现原理**

1、创建Session的时候，服务器将生成一个唯一的sessionid，然后用它生成一个关闭浏览器就会失效的cookie。

2、然后再将一个与这个sessionid关联的数据项加入散列表。

    例如这样一段代码:Session["UserName"]=23;

    假设sessionid为123那么散列表中会追加一行

     sessionid          username

     123                  23

3、当浏览器端提交到服务器时，会通过sessionid=123去散列表中寻找属于该用户的Session信息。

# 43.io文件

//引入java文件

File a = **new** File("E:/eclipse工作空间/新类/src/新类/Baozhuanglei.java");

//引入目录

File a1 = **new** File("E:/eclipse工作空间/新类/src");

//通过父的目录引入文件 a1 是父类 (只要是之前的父类以上都是可以的)

File a2 = **new** File(a1,"Calender.java");

//用来判断是不是一个文件或者是一个目录

**if**(a.isFile()){

System.***out***.println("是一个文件");

}

**if**(a1.isDirectory()){

System.***out***.println("是一个目录");

}

System.***out***.println(a.length());

//创建一个java文件

File a3 = **new** File(a1,"Calender22222222.java");

**try** {

a3.createNewFile();

} **catch** (IOException e) {

// **TODO** 自动生成的 catch 块

e.printStackTrace();

}

a3.mkdir()

dir.exists()

//这里是将我们在 f盘的新建文件夹 删除件删除

File a4 = **new** File("F:/新建文件夹");

a4.delete();

//这里是将我们创建的那个文件删除

a3.delete();

//创建一个目录

//下面这个不可以创建（因为有点多了）

File a6 = **new** File("F:/wo/shi/wjo");

a6.mkdir();//当我们要创建的目录前面也有需要创建的目录是后那么我们不可以使用这个创建目录

//下面这个将会创建的

File a5 = **new** File("F:/zhangyujin");

a5.mkdir();

//下面子这个会创建多个目录

File a7 = **new** File("F:/wo/shi/wjo/slasjdhfkhjk");

a7.mkdirs();

}

     if (!file.exists()){                              //文件不存在则创建

                 file.createNewFile();

             }

文件存储：

File file = **new** File(Environment.*getExternalStorageDirectory*(), filename);

outStream.write(content.getBytes());

文件读取

FileInputStream inStream =**new** FileInputStream(filePathStorage);

ByteArrayOutputStream outStream = **new** ByteArrayOutputStream();

**byte**[] buffer = **new** **byte**[1024];

**int** len = 0;

**while**((len = inStream.read(buffer)) != -1){

outStream.write(buffer, 0, len);

}

**byte**[] data = outStream.toByteArray();

**return** **new** String(data);

# 44 i++ ，i--

**public** **static** **void** a(**int** i){

**int** a = i++; //直接赋值的话，还是保留原来的数字。所以a等于1

System.***out***.println(i); //2 //这里i的值就是2了

System.***out***.println(a); //1

System.***out***.println(i); //因为使用之后了所以此时就是 加1 就是 2

System.***out***.println(i++); //2 //不赋值就是加1 ，此时相当于 是a = i++ ，还是保留

System.***out***.println(i);//3 //i使用之后了，所以加1 等于3

**int** b = ++i; //直接赋值，所以使用之前加1 就是 4

System.***out***.println(b); //4

System.***out***.println(i); //4

System.***out***.println(++i); //5

**//总结i++ 原值 ++i 加1**

**// 但是i的值都会+1**

}

}

# 45数组

**int[] a = new int[3]**; //定义一个数组

**int** i,j,n = 5,m = 0;

**int** a[][] = **new** **int**[5][5];

**for** (i = 0; i < n; i++) {

**for** (j = 0; j < n; j++) {

a[i][j] =m;

m++;

System.***out***.println( a[i][j]);

}

}

}

# 46.List 遍历

**public** **boolean** CheckUser(User user){

**boolean** a = **false**;

String sql ="select \* from lu where user = ? and password = ?";

String paras[] = {user.getUser(),user.getPassword()};

SqlHelper sqlHelper = **new** SqlHelper();

ArrayList arrayList = sqlHelper.executeQuery(sql, paras);

**if**(arrayList.size()!=0){

//使用对象将我们想要得到的东西

Object objects[] = (Object[])arrayList.get(0);

user.setEmil((String)objects[4]);

user.setGrade((String)objects[5]);

user.setPhone((String)objects[3]);

a =**true**;

}

**return** a ;

List list=query.list();

**for**(**int** i = 0 ; i < list.size();i++){

Object []object = (Object[]) list.get(i);

System.*out*.println(object[0].toString()+""+object[1]);

}

ArrayList<User> arraylist = new ArrayList<User>();

User user = new User();

while (rs.next()) {

user.setId(rs.getInt(1));

user.setUser(rs.getString(2));

user.setPassword(rs.getString(3));

user.setPhone(rs.getString(4));

user.setEmil(rs.getString(5));

user.setGrade(rs.getString(6));

arraylist.add(user);

List<Student> list=query.list();

**for**(Student e: list){

System.*out*.println("姓名"+e.getSname()+"性别"+e.getSsex());

}

# 47.map

Map<String,String> canshu = **new** HashMap<String,String>();

SharedPreferences preferences =context.getSharedPreferences("shareprefernces",Context.*MODE\_PRIVATE*);

//第一个我们的参数，第二个是如果没有这个参数的时候，默认返回的是blank或者自己写入

canshu.put("name",preferences.getString("name", "dage"));

canshu.put("age",String.*valueOf*(preferences.getInt("age",0)));

Map<String, String> params = service.getPreferences();

/\* 添加默认数据\*/

nameText.setText(params.get("name"));

ageText.setText(params.get("age"));

# 48.set

Set<Stucourse> stucourses = s.getStucourses();

**for**(Stucourse stc : stucourses){

System.*out*.println(s.getSname()+"选择了"+stc.getCourse().getCname());

}

}

**private** Set stucourses = **new** HashSet(0);

**public** Set getStucourses() {

**return** **this**.stucourses;

}

**public** **void** setStucourses(Set stucourses) {

**this**.stucourses = stucourses;

}

# 49容器

**Java容器类包含List、ArrayList、Vector及map、HashTable、HashMap**

保存java的对象的类。

# 50.线程

继承

MyThred **implements** Runnable{

MyThred **extends** Thread

## 1.程序中只要是多线程，都是实现runnable 接口为正统操作，

## 2.避免了单继承的局限，一个类可以实现多个接口，

## 3.适合于资源的共享 ，售票

## 4. main （）方法本身也是线程， 所以上面的线程是3，但是main方法比较简单，一个时间片就会运行完毕，所以他的执行结果先显示出来

runnbale

MyThred myThred1 = **new** MyThred("zhang");

Thread thread1 = **new** Thread(myThred1);

Thread thread2 = **new** Thread(myThred1);

thread 直接new出来就可以了

MyThred myThred1 = **new** MyThred("zhang");

MyThred myThred2 = **new** MyThred("gao");

## 例题

**public** **class** Testthread **implements** Runnable{

@Override

**public** **void** run() {

System.***out***.println("run.");

**throw** **new** RuntimeException("Problem");

}

**public** **static** **void** main(String[] args) {

Thread thread = **new** Thread(**new** Testthread());

thread.start();

System.***out***.println("End of method");

}

}

结果 因为main也是线程，里面的方法比较简单，所以先打出来

End of method

run.

Exception in thread "Thread-0" java.lang.RuntimeException: Problem

at com.hlj.thread.Testthread.run(Testthread.java:7)

at java.lang.Thread.run(Unknown Source)

# 51继承

## 1.简单讲继承

class A extends B,C  
不能这样写，因为java不支持多继承，  
但是你可以像下面这样实现继承多个类  
class A extends B  
class C extends A  
这样C就同时继承了B和A两个类

接口的实现，

class 类 implements 接口1,接口2,...

接口与类之间是实现关系，一个类可以实现多个接口，一个类只能继承一个父类

interface IP1 {  }

interface IP2 { }

public interface ISub extends IP1, IP2 {}

## 2.内部类实现多继承

**class** Example1 {

**public** String name()

{

**return** "liutao";

}

}

**class** Example2 {

**public** **int** age()

{

**return** 25;

}

}

**class** DjcTest

{

**private** **class** test1 **extends** Example1

{

**public** String name()

{

**return** **super**.name();

}

}

**private** **class** test2 **extends** Example2

{

**public** **int** age()

{

**return** **super**.age();

}

}

**public** String name()

{

**return** **new** test1().name();

}

**public** **int** age()

{

**return** **new** test2().age();

}

**public** **static** **void** main(String args[])

{

DjcTest mi = **new** DjcTest();

System.***out***.println("姓名:" + mi.name());

System.***out***.println("年龄:" + mi.age());

}

}

里面分别实现了两个内部类 test1,和test2 ，test1类又继承了Example1，test2继承了Example2，这样我们的类三DjcTest就拥有了Example1和Example2的方法和属性，也就间接地实现了多继承。

## 2. 在使用多态的时候，继承时候他的变量不是继承的，在使用多态的时候,父类是private，子类不同继承，而且不能用父类.atring

**class** A{

// private String aString = "父亲变量";

String aString = "父亲变量";

**public** **void** aString(){

System.***out***.println("父亲方法");

}

}

**public** **class** TestBl **extends** A{

String aString = "儿子变量";

**public** **void** aString(){

aString = "方法儿子"; //没有改变string的值

System.***out***.println("儿子方法");

}

**public** **static** **void** main(String[] args) {

//多态

System.***out***.println("\*\*\*\*\*多态\*\*\*\*\*\*");

A testB = **new** TestBl();

System.***out***.println(testB.aString );

testB.aString();

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*");

TestBl testBl = **new** TestBl();

System.***out***.println(testBl.aString );

testBl.aString() ;

}

}



## 3多态（动态和静态）

静态多态性指的是**在程序执行前方法已经被绑定**，程序在编译时，系统就能决定调用哪个函数，如重载。

动态多态性指在**在运行时根据具体对象的类型进行绑定，**运行中才能动态确定操作指针所指的对象，主要通过虚函数和重写来实现。

## 静态多态性

add(int a);   
add(int a,int b);   
add(double a);   
add(double a,double b);   
  
动态多态性   
public class A{ }   
  
public class AB extends A{ }   
  
public class AC extends A{ }   
  
public class test{   
 go(A a);

## 补充： 子类不能强制转化父类，只有父类才能强制转化子类

# 52. Jdbc

Connnection cn = null;

PreparedStatement ps = null;

ResultSet rs = null;

Statement st = null;

Class.*forName*("com.mysql.jdbc.Driver");

cn =DriverManager.*getConnection*("jdbc:mysql://localhost:3306/shouru","root","147094");

ps = cn.prepareStatement("select password from lu where user = ?");

//st *= cn..creatStatement();*

ps.setString(1, name);

***//rs = st.executeQuery(*"select password from lu where user = ?"*);***

rs = ps.executeQuery(); executeUpdate

if(rs.next()){

String p = rs.getString(1);

}

**//关闭连接，依次是 ResultSet Statement Connection**

**if(rs!=null){rs.close();}**

**if(sm!=null){sm.close();}**

**if(cn!=null){cn.close();}**

# 53 sql语句面试

## 1.

1. 用一条SQL 语句 查询出每门课都大于80 分的学生姓名

name   kecheng   fenshu   
张三     语文       81  
张三     数学       75  
李四     语文       76  
李四     数学       90  
王五     语文       81  
王五     数学       100  
王五     英语       90  
  
select distinct name from table where name not in (select distinct name from table where fenshu<=80)  
select name from table group by name having min(fenshu)>80

## 2.



select name ,SUM(distinct score)  from Table\_1 group by name having SUM(distinct score)>100

使用distinct 防止重复

# 54 float double

因为int 是整数，整数除整数就是整数double是小数。小数除小数才是小数double yy=4/10;yy得的值是0?解释 4和10都是整数int类型的，所以4/10结果也是int的，就是0，把结果赋值给double类型，才会转 换成double，就是0.0d;若果想得到完整的结果0.4，那就需要在除法前先将4或者10转换成double类型比如（double）4/10；int i1 = 123;int i2 = 456; double d1 = (i1+i2)\*1.2;float f1 = (float)((i1+i2)\*1.2)比如这个，为什么((i1+i2)\*1.2)的结果是double类型的，需要强制转换成float类型　double 和 float 的区别是double精度高，有效数字16位，float精度7位。但double消耗内存是 float的两倍，double的运算速度比float慢得多，C语言中数学函数名称double 和 float不同，不要写 错，能用单精度时不要用双精度（以省内存，加快运算速度）java中常量 的浮点数字就被认为是double型的，而且不能直接赋给float型的变量像float a=1.2; 这样的编绎直接出错要写成float a=1.2f; float a = 1;这里的1是整形，当赋值给一个float时，存在一个隐式转换，不损失精度。float a = 1f;（上面的说明，但是其实这样写上去是错误的）1f就是float类型的。如果你这样定义 float f = 1.0; 肯定会出错。 因为1.0默认是double类型的。double范围比float大。为了不损失精度，不会自动转换 。这是必须这样写 float f = 1.0f; float a = 1.0f; float b = 2.0f; printf("%f",a/b); //结果为

|  |
| --- |
| Double a = 2.4;  Double b = 1.2;  **int** aint = 2;  **int** bint = 3;  Double aDouble = 1.11;  Double bDouble = 1.32;  //尽可能得到16位，如果确实没有16位则将保留0  System.*out*.println("double除法取16为小数"+aDouble/bDouble);  System.*out*.println("double除法取16，但是不够16位，则只保留0或者最后的结果"+a/b);  System.*out*.println("整数除法取余数"+aint/bint);  System.*out*.println("double除以int 得到的结果为double"+a/aint);  **float** fa = 2.4f;  **float** fb = 1.2f;  **float** af = 1.11f;  **float** bf = 1.32f;  System.*out*.println(fa/fb); //2.0  System.*out*.println(af/bf); //0.84090906 //8位 |



## 2、short

|  |
| --- |
| **short** s1 = 2;  // s1=s1 +1; 错误  s1 += 2; //正确  **int** a = 1;  a = a+1; |

# 55.数据库事物的AICD性质原子性，持久性，隔离性，一致性

A 事务的原子性

是指事务的所有操纵在数据库中要么全部正确反映出来，要么全部不反映。

B事物持久性。

事物一旦提交，对数据库俄改变是永久的指的是持久性。

C事物隔离性。

一个事物内部的操作及使用的数据对并发的其他事务是隔离的指的是隔离性。

D 事务一致性

事务必须是使数据库从一个一致的状态变到另一个一致的状态指的是一致性。

# 56.排序方法

：[插入排序](http://baike.baidu.com/view/396887.htm)，[冒泡排序](http://baike.baidu.com/view/254413.htm)，[堆排序](http://baike.baidu.com/view/157305.htm)，[归并排序](http://baike.baidu.com/view/90797.htm)，[选择排序](http://baike.baidu.com/view/547263.htm)，[计数排序](http://baike.baidu.com/view/1209480.htm)，[基数排序](http://baike.baidu.com/view/1170573.htm)，[桶排序](http://baike.baidu.com/view/1784217.htm)，[快速排序](http://baike.baidu.com/view/115472.htm)

## 1.插入排序

## 2.快速排序

# 57[Struts的validate框架是如何验证的？](http://blog.csdn.net/jxq0816/article/details/17806675)

方法一：Action类继承ActionSupport类，其中ActionSupport实现了Validateable接口，并定义了validation方法，只要在用户自定义的Action类中重写该方法就可以实现该功能。

其中错误信息可以通过addFieldError方法将错误信息保存起来。

方法二：在Action类的文件的路径下，添加一个校验文件ActionName-validation.xml，并配置校验器。

# 59 spring 和springmvc的区别

Spring是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。  
  
Spring MVC类似于struts的一个MVC开框架，其实都是属于Spring ，Spring MVC需要有Spring 的架包作为支撑才能跑起来

MVC是三个单词的缩写,分别为： 模型(Model),视图(View)和控制Controller)。 [MVC模式](https://www.baidu.com/s?wd=MVC%E6%A8%A1%E5%BC%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YzPHNWrANhrjR1P1ubmWNW0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW6krj6dnHmd)的目的就是实现Web系统的职能分工。

Model层实现系统中的[业务逻辑](https://www.baidu.com/s?wd=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YzPHNWrANhrjR1P1ubmWNW0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW6krj6dnHmd)，通常可以用JavaBean或EJB来实现。 View层用于与用户的交互，通常用JSP来实现。

Controller层是Model与View之间沟通的桥梁，它可以分派用户的请求并选择恰当的视图以用于显示，同时它也可以解释用户的输入并将它们映射为模型层可执行的操作

# 60 Struts的跳转

## Struts1

**return** mapping.findForward("ok");

## Struts2

return "message";

[struts2](https://www.baidu.com/s?wd=struts2&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YkrHf1m1T4n1cYujDknyFB0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjDYrjfdrjDv) 和struts1不同的血统，struts1.x基于Servlet 的侵入式设计，[struts2](https://www.baidu.com/s?wd=struts2&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YkrHf1m1T4n1cYujDknyFB0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjDYrjfdrjDv).x基于Filter的无侵入式设计。。

# 96.自定义异常（抛出异常还会继续执行，但是不抛出是不会执行的）

/\*下面做了归纳总结,欢迎批评指正\*/

/\*自定义异常\*/

**class** ChushulingException **extends** Exception

{

**public** ChushulingException(String msg)

{

**super**(msg);

}

}

**class** ChushufuException **extends** Exception

{

**public** ChushufuException(String msg)

{

**super**(msg);

}

}

/\*自定义异常 End\*/

**class** Numbertest

{

**public** **int** shang(**int** x,**int** y) **throws** ChushulingException,ChushufuException

{

**if**(y<0)

{

**throw** **new** ChushufuException("您输入的是"+y+",规定除数不能为负数!");//抛出异常 设置异常的信息

}

**if**(y==0)

{

**throw** **new** ChushulingException("您输入的是"+y+",除数不能为0!");

}

**int** m=x/y;

**return** m;

}

}

**public** **class** Rt001

{

**public** **static** **void** main(String[]args)

{

Numbertest n=**new** Numbertest();

//捕获异常

**try**

{

System.***out***.println("商="+n.shang(1,-3));

}

**catch**(ChushulingException yc)

{

System.***out***.println(yc.getMessage());

yc.printStackTrace();

}

**catch**(ChushufuException yx)

{

System.***out***.println(yx.getMessage()+"hello");

yx.printStackTrace();

}

**catch**(Exception y)

{

System.***out***.println(y.getMessage());

y.printStackTrace();

}

**finally**{ System.***out***.println("finally!");} ////finally不管发没发生异常都会被执行

}

}

/\*

’

[总结]

1.自定义异常:

class 异常类名 extends Exception

{

public 异常类名(String msg)

{

super(msg);

}

}

2.标识可能抛出的异常:

throws 异常类名1,异常类名2

3.捕获异常:

try{}

catch(异常类名 y){}

catch(异常类名 y){}

4.方法解释

getMessage() //输出异常的信息

printStackTrace() //输出导致异常更为详细的信息

运行结果

第一行 黑字表示的是异常的信息

第二、四、五行是输出的更为详细的信息，可以表明在哪里发生了异常

第二行 表示finally中执行的东西



# 62.hashmap和hashtable的区别



HashMap线程不安全的，Hashtable线程安全。

hashmap是map的一个实现类（也就是继承了AbstractMap），hashtable继承的是Drictionary

当需要同步时，用Hashtable，反之用HashMap。但是，因为在需要时，HashMap可以被同步，HashMap的功能比Hashtable的功能更多，而且它不是基于一个陈旧的类的，所以有人认为，在各种情况下，HashMap都优先于Hashtable。

# 63.[Servlet生命周期与工作原理](http://www.cnblogs.com/cuiliang/archive/2011/10/21/2220671.html)

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

　　3，终止阶段　　调用destroy()方法

## init()

在 Servlet 的生命期中，仅执行一次 init() 方法。它是在服务器装入 Servlet 时执行的。 可以配置服务器，以在启动服务器或客户机首次访问 Servlet 时装入 Servlet。 无论有多少客户机访问 Servlet，都不会重复执行 init() 。在Servlet的整个生命周期内，init()方法只被调用一次。

## service（）Servlet响应请求阶段：

　　对于用户到达Servlet的请求，HttpServlet类实现了该service方法并且要求调用下列的方法之一：service方法会调用与请求对应的doGet或doPost等方法 Servlet容器会创建特定于这个请求的HttpServletRequest对象和HttpServletRequest对象，然后调用Servlet的service方法。service方法从ServletRequest对象获得客户请求信息，处理该请求，并通过ServletResponse对象向客户返回响应信息。

## destroy（）Servlet终止阶段：

　　当WEB应用被终止，或Servlet容器终止运行，或Servlet容器重新装载Servlet新实例时，Servlet容器会先调用Servlet的destroy()方法，在destroy()方法中可以释放掉Servlet所占用的资源。

# 64.get和post区别

get方式安全性低，post方式较安全。但是post方式执行效率要比get方式差一些。

GET方式提交的数据最多只能是1024字节，理论上POST没有限制，

GET请求的数据会附在URL之后（**就是把数据放置在HTTP协议头**中），以?分割URL和传输数据，参数之间以&相连，如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0%E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密，得出如：%E4%BD%A0%E5%A5%BD，其中％XX中的XX为该符号以16进制表示的ASCII。

**POST把提交的数据则放置在是HTTP包的包体中。**

# 65. checkbox传递多个值



# 66. [堆区与栈区的区别](http://blog.csdn.net/sjxbf/article/details/6441730)

1.栈区（stack）— 由编译器自动分配释放，存放函数的参数值，局部变量的值（基本类型的变量和对象的引用变量都在函数的栈内存中分配）等。其操作方式类似于数据结构中的栈。

2.堆区（heap） — 一般由程序员分配释放，若程序员不释放，堆内存用来存放由new创建的对象和数组。程序结束时可能由os回收

# 67.反射机制的工作原理

反射机制概念：JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法；这种动态获取的信息以及动态调用对象的方法的功能称为[java语言](https://www.baidu.com/s?wd=java%E8%AF%AD%E8%A8%80&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Yknjmzm1FbujcLmWN9nAPb0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHTknH0snHDs)的反射机制。

有了反射机制，那么就能逆向代码了，反编译器就是这个原理。2、像流行的框架，Spring,Hibernate等，就是利用反射机制，要不然怎么可能你配置下文件就完事呢3、动态生成类啊，当然spring里面都用到了。希望我的回答对你有帮助！

# 68. [ibatis 中 $与#的区别](http://blog.csdn.net/kiss_vicente/article/details/7602900)

select \* from table where id = #id# ，其中如果字段id为字符型，那么#id#表示的就是'id'类型，如果id为整型，那么#id#就是id类型。

select \* from table where id = $id$ ，如果字段id为整型，Sql语句就不会出错，但是如果字段id为字符型，那么Sql语句应该写成 select \* from table where id = '$id$'

# 69.spring 注解扫描

## 1.1 配置EmployeeService对象通过注解的方式来注入属性值,这是一个知识点

<!-- 配置EmployeeService对象通过注解的方式来注入属性值,这是一个知识点 -->

<bean id=*"employeeService"* class=*"com.hsp.service.imp.EmployeeService"*/>

## 1.2.到了EmployeeService中，下面的setSessionFactory是不会调用了

//当我们给某个属行 resource 时候 spring 就会通过byName 自动注入属行值

@Resource

**public** SessionFactory sessionFactory;

## 1.3.然后启用注解扫描

<!-- 启用注解扫描 -->

<context:annotation-config/>

<!-- 配置EmployeeService对象通过注解的方式来注入属性值,这是一个知识点 -->

<bean id=*"employeeService"* class=*"com.hsp.service.imp.EmployeeService"*/>

# 2.action中也使用 注解的方式来注入

**public** **class** LoginAction **extends** DispatchAction {

//当Struts 配spring接管之后就可以通过这种方式来被调用了

@Resource

EmployeeServiceInter employeeService;

# 106、在浏览器中输入www.baidu.com后执行的全部过程

**48、html访问全过程**

A）解析Web页面的URL，得到Web服务器的域名

查询本地的DNS缓存，当用户在浏览器中输入一个需要登录的网址时，系统会首先自动从Hosts文件中寻找对应的IP地址，以找到域名对应的主机IP地址（若有，则跳到4）  
查询远程**域名根DNS，找**到IP地址  
 B）通过DNS服务器获得Web服务器的IP地址

C）与Web服务器建立TCP连接  
D）与Web服务器建立HTTP连接

向远程IP地址的服务器发送请求（若请求失败且未经过2，则返回2，若再再次失败，返回[错误代码](https://www.baidu.com/s?wd=%E9%94%99%E8%AF%AF%E4%BB%A3%E7%A0%81&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YkP1uhnWcsn1fzuADLrHIW0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1Rzn1RznjmL)）

E）从Web服务器获得URL指定的文档  
 F）浏览器解释页面文档，并显示在屏幕

# 81.单例模式

## 1添加一个单态工具静态代码块如果要使用对象，则上面的对象什么不能使用final

**final** **public** **class** ApplicaionContextUtil {

**private** **static** ApplicationContext *ac*=**null**;

**private** ApplicaionContextUtil(){

}

**static**{

*ac*=**new** ClassPathXmlApplicationContext("applicationContext.xml");

}

**public** **static** ApplicationContext getApplicationContext(){

**return** *ac*;

}

}

## 2.恶汉式单例模式

**Public class Singleton{**

**Private static final Singleton singleton = new Singleton();**

**Private Singleton(){}**

**Public static Singleton getInstance(){**

**Return singleton;**

**}**

**}**

## 2.懒汉式

**public** **class** lazySingleton {

**private** **static** lazySingleton *lazySingleton*;//未初始化

**private** lazySingleton(){}

**public** **static** lazySingleton getInstance(){

**if**(*lazySingleton* == **null**) {

*lazySingleton* = **new** lazySingleton();

}

**return** *lazySingleton*;

}

}

## 3.总结

懒汉式也是通过一个类的静态变量实现的。但是并没有直接初始化。而是在函数getInstance()中实例化的，也就是每次想用这个实例的时候初始化的，如果已经初始化了，那么就不用初始化了，这样也很好啊，对的，其实懒汉式是比较常用的实现方式。

# 83 ER图和实体的关系

一对一联系

一对多联系

多对多联系

## 1.第一步



## 2.画出相应的ER图



# 84.webservice

## 1.soap：simple object access protocal

|  |
| --- |
| 直译: 简单对象访问协议   1. 是一种简单的、基于**HTTP和**[**XML**](http://baike.baidu.com/view/63.htm)**的协议**, 用于在WEB上交换**结构化的数据** 2. soap消息：**请求消息**和**响应消息** 3. **http+xml片断** |

## 2.WSDL：web service definition language

|  |
| --- |
| 直译 : WebService定义语言    1. 对应一种类型的文件**.wsdl**  2. **定义了web service的服务器端与客户端应用交互传递请求和响应数据的格式和方式**  3. 一个web service对应一个**唯一的wsdl文档** |

# 85. public final abstract static

final和abstract，private和abstract，static和abstract这些是不能放在一起的修饰符

abstract 只能与public或protected连用，

类的成员不写访问修饰时默认为default

抽象类不一定有抽象方法，但是抽象方法一定在抽象类中

比如要定义一个变量： 下面的顺序都是一样的  
final static public int i；  
public final static int i；  
static final public int i；  
static public final int i；  
  
public final static

接口里面的方法为[抽象方法](https://www.baidu.com/s?wd=%E6%8A%BD%E8%B1%A1%E6%96%B9%E6%B3%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YkmyPbnHNbn1fYPW6dPH790ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHn1rj6LPjRv)，实现该接口的类都必须实现接口里定义的方法，比如定义： public void test()；其实际为：public abstract void test()。

**abstract** **class** TestAbstract {

**abstract** **void** a();

**abstract** **void** b();

}

**public** **class** TestExtendAbstract **extends** TestAbstract{

**protected** **void** a() {}

**public** **void** b() { }

}

abstract 默认是public 公开的，但是在实现的时候protected 也是可以的

# 86、递归算法

|  |
| --- |
| \* **@Description**:1 1 2 3 5 8 13  \*/  **public** **static** **int** Fribonacci(**int** n){  **if**(n<=2)  **return** 1;  **else**  **return** *Fribonacci*(n-1)+*Fribonacci*(n-2);  }  /\*\*  \* **@Description**: n的阶乘 0的阶乘是1  \*/  **public** **static** **int** f(**int** n){    **if**(n<=1){  **return** 1;  }**else** {  **return** n \* *f*(n-1);  }  } |

# 88、for循环 参数问题

**public** **class** ForTest {

**public** **static** **boolean** foo (**char** c)

{

System.***out***.print(c);

**return** **true**;

}

**public** **static** **void** main(String[] args) {

**int** i = 0;

**for** ( *foo*('A'); *foo*('B') &&(i < 2); *foo*('C')) {

i++ ;

*foo*('D');

}

}

//本循环 只会执行2次 也就是 ABDC BDC

//后面的B 是foo(B)进行判断时候执行的 ，不计入循环次数

//最后面的foo(C)是在循环每次执行完毕的时候执行的

//foo('A') 只会执行一次，因为这个是初始化条件

//运行结果为ABDC BDCB

}

# 89、[PreparedStatement和Statement的区别](http://www.cnblogs.com/codecup/p/3793371.html)

PreparedStatement用于处理动态SQL语句，在执行前会有一个预编译过程，这个过程是有时间开销的，虽然相对数据库的操作，该时间开销可以忽略不计，但是PreparedStatement的预编译结果会被缓存，下次执行相同的预编译语句时，就不需要编译，只要将参数直接传入编译过的语句执行代码中就会得到执行，所以，对于批量处理可以大大提高效率。

2>Statement每次都会执行SQL语句，相关数据库都要执行SQL语句的编译。

3>作为开发者，应该尽可能以PreparedStatement代替Statement，其原因如下:

(1)代码的可读性和可维护性。从代码比较中就应该能看出来；

[复制代码](javascript:void(0);)

stmt.executeUpdate("insert into tb\_name (col1,col2,col2,col4) values ('"+var1+"','"+var2+"',"+var3+",'"+var4+"')");//stmt是Statement对象实例

perstmt = con.prepareStatement("insert into tb\_name (col1,col2,col2,col4) values (?,?,?,?)");

perstmt.setString(1,var1);

perstmt.setString(2,var2);

perstmt.setString(3,var3);

perstmt.setString(4,var4);

perstmt.executeUpdate(); //prestmt是 PreparedStatement 对象实例

[复制代码](javascript:void(0);)

(2)PreparedStatement尽最大可能提高性能。

(3)极大的提高了安全性。

恶意SQL语法

String sql = "select \* from tb\_name where name= '"+varname+"' and passwd='"+varpasswd+"'";   
如果我们把[' or '1' = '1]作为varpasswd传入进来.用户名随意,看看会成为什么?   
select \* from tb\_name = '随意' and passwd = '' or '1' = '1';   
因为'1'='1'肯定成立,所以可以任何通过验证.更有甚者:   
把[';drop table tb\_name;]作为varpasswd传入进来,则:   
select \* from tb\_name = '随意' and passwd = '';drop table tb\_name;有些数据库是不会让你成功的,但也有很多数据库就可以使这些语句得到执行.   
而如果你使用预编译语句.你传入的任何内容就不会和原来的语句发生任何匹配的关系.只要全使用预编译语句,你就用不着对传入的数据做任何过虑.而如果使用普通的statement,有可能要对drop,;等做费尽心机的判断和过虑.

# 90、[java实现文件复制功能](http://www.cnblogs.com/haha12/p/6197731.html)

public void copyFile(File fromFile,File toFile) throws IOException{

FileInputStream ins = new FileInputStream(fromFile);

FileOutputStream out = new FileOutputStream(toFile);

byte[] b = new byte[1024];

int n=0;

while((n=ins.read(b))!=-1){

out.write(b, 0, n);

}

ins.close();

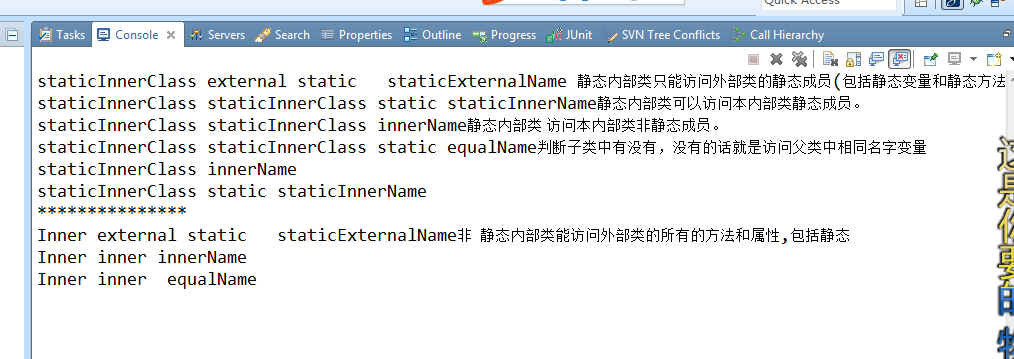
out.close();

}

# 91 [静态内部类和非静态内部类的区别](http://blog.csdn.net/u012123938/article/details/46684839)（外面的类可以有private变量）



|  |
| --- |
| /\*\*  想要理解static应用于内部类时的含义，你就必须记住，  普通的内部类对象隐含地保存了一个引用，指向创建它的外围类对象。  然而，当内部类是static的时，就不是这样了。  \*/  **public** **class** StaticAndNoStaticClass{  **private** **static** String *staticExternalName* = "external static staticExternalName";  **private** String externaNamr = "external externaNamr";    /\*\*  \* 测试相同名字的内部类调用  \*/  **private** **static** String *staticEqualName* = "external static equalName";  **private** String equalName = "external equalName";  // 静态内部类可以用public,protected,private修饰  **static** **class** StaticClass {    **private** **static** String *staticInnerName* = "staticInnerClass static staticInnerName";    // 静态内部类中可以定义静态或者非静态的成员  **private** String innerName = "staticInnerClass innerName";  **private** **static** String *staticEqualName* = "staticInnerClass static equalName";  **public** **void** display() {    //报错 静态内部类不能访问外部类的非静态成员(包括非静态变量和非静态方法)  // System.out.println(externaNamr);  // 静态内部类只能访问外部类的静态成员(包括静态变量和静态方法)  System.*out*.println("staticInnerClass " +*staticExternalName* + " 静态内部类只能访问外部类的静态成员(包括静态变量和静态方法)");  // 静态内部类 可以访问本内部类静态成员  System.*out*.println("staticInnerClass " + *staticInnerName*  + "静态内部类可以访问本内部类静态成员。");  // 静态内部类 访问本内部类非静态成员。  System.*out*.println("staticInnerClass " + innerName  + "静态内部类 访问本内部类非静态成员。");    //先判断子类中有没有，没有的话就是访问父类中相同名字变量  System.*out*.println("staticInnerClass "+*staticEqualName*  + "判断子类中有没有，没有的话就是访问父类中相同名字变量" );  }  }    /\*\*  \* 非静态内部类  \*  \*/  **class** NoStaticClass{  //报错 非静态内部类不能有静态的成员和方法  //private static String staticInnerName = "inner static staticInnerName";  **private** String innerName = "inner innerName";  **private** String equalName = "inner equalName";  **public** **void** display() {    //非 静态内部类能访问外部类的所有的方法和属性  System.*out*.println("Inner " + *staticExternalName*  + "非 静态内部类能访问外部类的所有的方法和属性,包括静态");  System.*out*.println("Inner " + innerName);    //先判断子类中有没有，没有的话就是访问父类中相同名字变量  System.*out*.println("Inner "+equalName);  }  }    /\*\*  测试外部类调用内部类 私有方法和属性  \*/  **public** **void** printInfo() {  StaticClass staticClass = **new** StaticClass();  staticClass.display();  System.*out*.println(staticClass.innerName);// 可以访问内部类的私有成员  System.*out*.println(staticClass.*staticInnerName*);// 外部类访问内部类的静态成员：内部类.静态成员    System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  NoStaticClass noStaticClass = **new** NoStaticClass();  noStaticClass.display();  }  **public** **static** **void** main(String[] args) {  StaticAndNoStaticClass staticTest = **new** StaticAndNoStaticClass();  staticTest.printInfo();  }  } |



# 91.2 再其他的外部类中 获得内部类对象不能直接new，（本类中是可以的）

|  |
| --- |
| **public** **class** Main {  **public** **static** **void** main(String[] args) {  //报错 静态内部类 不能直接new出来  //NoStaticClass noStaticClass = new NoStaticClass();    NoStaticClass noStaticClass = **new** StaticAndNoStaticClass().**new** NoStaticClass();  noStaticClass.display();  }  } |

# 91.3 外部调用内部类

**class** DjcTest

{

**public** **int** thisTest(){

**return** 2;

}

**private** **class** test1 **extends** Example1

{

**public** String name()

{

System.***out***.println(DjcTest.**this**.thisTest());

System.***out***.println(**this**.thisTest());

**return** **super**.name();

}

内部类使用this是使用的内部的类，如果使用外部的类，则可以直接使用外部类名.this.方法



# 92 Switch 从出现的开始找

**public** **static** **void** name() {

String i="b";

**switch**(i) {

**case** "a":

System.***out***.println("a");

**case** "b":

System.***out***.println("b");

**case** "c":

System.***out***.println("c");

**default**:

System.***out***.println("default");

**break**;

}

}

}



**public** **static** **void** name() {

String i="b";

**switch**(i) {

**case** "a":

System.***out***.println("a");

**case** "b":

System.***out***.println("b");

**case** "c":

System.***out***.println("c");

**break**;

**default**:

System.***out***.println("default");

**break**;

}

}



# 93 [throw和throws的区别](http://blog.csdn.net/hhy62011980/article/details/5548278)

**throw是语句抛出一个异常。**语法：throw (异常对象);  
         throw e;

**throws是方法可能抛出异常的声明**。(用在声明方法时，表示该方法可能要抛出异常)  
语法：[(修饰符)](返回值类型)(方法名)([参数列表])[throws(异常类)]{......}  
            public void doA(int a) throws Exception1,Exception3{......}

 ： void doA(int a) throws Exception1,Exception3{  
           try{  
                 ......

           }catch(Exception1 e){  
              throw e;  
           }catch(Exception2 e){  
              System.out.println("出错了！");  
           }  
           if(a!=b)  
              throw new  Exception3("自定义异常");  
}

# 94jdbc事务

<http://blog.csdn.net/wzwenhuan/article/details/9193337>

1. /\*\*
2. \* 开始事务
3. \* @param cnn
4. \*/
5. **public** **static** **void** beginTransaction(Connection cnn){
6. **if**(cnn!=**null**){
7. **try** {
8. **if**(cnn.getAutoCommit()){
9. cnn.setAutoCommit(**false**);
10. }
11. } **catch** (SQLException e) {
12. // TODO Auto-generated catch block
13. e.printStackTrace();
14. }
15. }
16. }
18. /\*\*
19. \* 提交事务
20. \* @param cnn
21. \*/
22. **public** **static** **void** commitTransaction(Connection cnn){
23. **if**(cnn!=**null**){
24. **try** {
25. **if**(!cnn.getAutoCommit()){
26. cnn.commit();
27. }
28. } **catch** (SQLException e) {
29. // TODO Auto-generated catch block
30. e.printStackTrace();
31. }
32. }
33. }
35. /\*\*
36. \* 回滚事务
37. \* @param cnn
38. \*/
39. **public** **static** **void** rollBackTransaction(Connection cnn){
40. **if**(cnn!=**null**){
41. **try** {
42. **if**(!cnn.getAutoCommit()){
43. cnn.rollback();
44. }
45. } **catch** (SQLException e) {
46. // TODO Auto-generated catch block
47. e.printStackTrace();
48. }
49. }
50. }

使用

1. **public** **void** funcExample(){
2. Connection cnn=DbUtil.getConnection();
3. **try**{
4. //开启事务
5. DbUtil.beginTransaction(cnn);
7. //调用Dao层多个增删改方法
8. //..........
10. //提交事务
11. DbUtil.commitTransaction(cnn);
12. }**catch**(Exception e){
13. e.printStackTrace();
14. //回滚事务
15. DbUtil.rollBackTransaction(cnn);
16. }**finally**{
17. //还原连接状态
18. DbUtil.resetConnection(cnn);
19. DbUtil.close(pstm);
20. DbUtil.close(cnn);
21. }
23. }

# 95 构造器

**public** **class** TestConstruster {

**public** **void** TestConstruster(){

System.***out***.println("不出错");

}

**public** TestConstruster(){

}

**public** **static** **void** main(String[] args) {

TestConstruster testConstruster = **new** TestConstruster();

testConstruster.TestConstruster();

}

}



# 96、public

## 1、 一个java 文件类 只能有一个public，也就是文件名，顺序随便(内部类可以public)





## 3、只有在public 中的main函数才能运行，

**public** **class** PublicOrderTest {

}

**class** A{

//不能够运行

**public** **static** **void** main(String[] args) {

System.*out*.println("123");

}

}

# 96. final

|  |
| --- |
| **public** **class** FinalTest {  //1、final 定义必须给出值，否则错误  //final static String aString ;  **final** **static** String *aString* = **null** ;    **public** **static** **void** m(){  //2、 aString = "12";// 错误，全局变量final定义的，不能够改变    //3、 方法里面的可以先不给值 ，下面456 赋值  **final** String bString ;  bString = "456";  //4、bString = "456"; 上面已经给值了，不可以重新赋值  }  } |

# 97 多表查询

## 1、having

### 数据库中where与having区别

having 和where 都是用来筛选用的   
having 是筛选组 而where是筛选记录  
1》当分组筛选的时候 用having  
2》其它情况用where  
-----------------------------------------------------  
用having就一定要和group by连用，  
用group by不一有having （它只是一个筛选条件用的）

<http://blog.csdn.net/yangyi22/article/details/7483859>

### 举例

**一、给定一张学生成绩表tb\_grade：学号，姓名，科目号，科目名，分数。（如下表）**

（select \* from tb\_grade order by Sno,Cno;）  
+------+-------+-----+-------+-------+  
| Sno  | Sname | Cno | Cname | score |  
+------+-------+-----+-------+-------+  
| 1001 | 李菲  |   1 | 语文  |    86 |  
| 1001 | 李菲  |   2 | 数学  |    56 |  
| 1001 | 李菲  |   3 | 物理  |    48 |  
| 1001 | 李菲  |   4 | 化学  |    90 |  
| 1001 | 李菲  |   5 | 英语  |    57 |  
| 1002 | 王琪  |   1 | 语文  |    90 |  
| 1004 | 和树  |   3 | 物理  |    67 |  
| 1004 | 和树  |   4 | 化学  |    30 |  
| 1004 | 和树  |   5 | 英语  |    68 |  
+------+-------+-----+-------+-------+

 1、查询不及格科目数大于等于2的学生学号和不及格科目数量：

[**MySQL**](http://lib.csdn.net/base/mysql)> select Sno,count(score) as '不及格科目数' from tb\_grade where score<60 group by Sno having count(score)>=2;

+------+--------------+

## 2、distinct

如何用distinct消除重复记录的同时又能选取多个字段值？

需求是：我要消除name字段值重复的记录，同时又要得到id字段的值，其中id是自增字段。   
select distinct name from t1 能消除重复记录，但只能取一个字段，现在要同时取id,name这2个字段的值。   
select distinct id,name from t1 可以取多个字段，但只能消除这2个字段值全部相同的记录   
最后解决方法：   
SELECT *id*,name FROM t1 WHERE id IN(SELECT MAX(id) FROM t1 GROUP BY name) order by *id* desc

注意开头的 *id* 的 一定要，后面的order by 里有的字段一定要加进select 结果，要不然排序无效

## 3、触发器

mysql> DELIMITER ||

mysql> CREATE TRIGGER trig2 BEFORE DELETE

-> ON work FOR EACH ROW

-> BEGIN

-> INSERT INTO time VALUES(NOW());

-> INSERT INTO time VALUES(NOW());

-> END

-> ||

Query OK, **0** rows affected (**0.06** sec)

mysql> DELIMITER

## 4. SELECT嵌套

嵌套SELECT语句也叫子查询，一个 SELECT 语句的查询结果能够作为另一个语句的输入值。子查询不但能够出现在Where子句中，也能够出现在from子句中，作为一个临时表使用，也能够出现在select list中，作为一个字段值来返回。

1、单行子查询 ：单行子查询是指子查询的返回结果只有一行数据。当主查询语句的条件语句中引用子查询结果时可用单行比较符号（＝, >, <, >=, <=, <>）来进行比较。

例：  
select ename,deptno,sal  
from emp  
where deptno=(select deptno from dept where loc='NEW YORK')；  
  
2、多行子查询：多行子查询即是子查询的返回结果是多行数据。当主查询语句的条件语句中引用子查询结果时必须用多行比较符号（IN，ALL,ANY）来进行比较。其中，IN的含义是匹配子查询结果中的任一个值即可（"IN" 操作符，能够测试某个值是否在一个列表中），ALL则必须要符合子查询的所有值才可，ANY要符合子查询结果的任何一个值即可。而且须注意ALL 和ANY 操作符不能单独使用，而只能与单行比较符（=、>、< 、>= 、<= 、<>）结合使用。

例：

1）.多行子查询使用IN操作符号例子：查询选修了老师名叫Rona(假设唯一)的学生名字

sql> select stName

from Student

where stId in(selectdistinct stId from score where teId=(select teId from teacher where teName='Rona'));

   查询所有部门编号为A的资料：

SELECT ename,job,sal

FROM EMP

WHERE deptno in ( SELECT deptno FROM dept WHERE dname LIKE 'A%')；  
2）.多行子查询使用ALL操作符号例子：查询有一门以上的成绩高于Kaka的最高成绩的学生的名字:

sql> select stName

from Student

where stId in(select distinct stId from score where score >all(select score from score where stId=(select stId from Student where stName= 'Kaka') ));

4、内联视图子查询

例：  
(1)SELECT ename,job,sal,rownum  
FROM (SELECT ename,job,sal FROM EMP ORDER BY sal)；  
(2)SELECT ename,job,sal,rownum  
FROM ( SELECT ename,job,sal FROM EMP ORDER BY sal)  
WHERE rownum<=5；  
  
5、在HAVING子句中使用子查询

例：  
SELECT deptno,job,AVG(sal) FROM EMP GROUP BY deptno,job HAVING AVG(sal)>(SELECT sal FROM EMP WHERE ename='MARTIN')；

让我们再看看一些具体的实例，

# 98 线程

## 1、给线程赋值给名字，main线程不需要赋值

### 1、线程类

//这个就是一个多线程的操作类 ，下面就该启动线程了

**public** **class** SleepRunnable **implements** Runnable{

**public** **void** run() {

// **TODO** Auto-generated method stub

**for**(**int** i=0;i<10;i++) //表示循环 10 次

{

**try** {

//每个休眠500毫秒

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println(Thread.*currentThread*().getName()+"线程正在运行");

}

}

}

### 2、执行线程

**private** **static** **void** Sleet() {

SleepRunnable sleepRunnable = **new** SleepRunnable();

Thread thread = **new** Thread(sleepRunnable, "线程1");

Thread thread2 = **new** Thread(sleepRunnable, "线程2");

Thread thread3 = **new** Thread(sleepRunnable, "线程3");

thread.start();

thread2.start();

thread3.start();

sleepRunnable.run(); 主线程

}

### 3、测试， 主线程发现不需要给值，



## 2、线程休眠和线程中断

### 1、线程类

**public** **class** InteruptRunnable **implements** Runnable{

**public** **void** run() {

// **TODO** Auto-generated method stub

System.***out***.println(" 1. 进入线程");

**try** {

//每个休眠500毫秒//

System.***out***.println("2. 线程将要 休眠");

Thread.*sleep*(5000);

System.***out***.println("3. 线程正在休眠");

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

System.***out***.println("4. 线程中断休眠");

**return**; //一，当线程休眠中断之后就会发生异常 旦有了异常,返回方法调用处

}

System.***out***.println(Thread.*currentThread*().getName()+"5. 线程正在运行");

}

}

### 2、休眠和中断测试

**private** **static** **void** Interupt() {

InteruptRunnable interuptRunnable = **new** InteruptRunnable();

Thread thread = **new** Thread(interuptRunnable, "线程1");

thread.start();

**try** {

System.***out***.println("执行thread.start();");

thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("这里的方法是中端");

thread.interrupt();

System.***out***.println("这里的方法是中端2");

}

### 3、测试1 thread.*sleep*(2000);



### 测试2、 thread.*sleep*(5000);



## 2.2、解释： 我们看到了上面的主方法中的休眠时间如果小于线程中设置的5000如果添加中断，则会让线程中断。

## 如果是等于或者大于5000，则线程不会中断，这是因为线程的设置休眠的时间是重合的，相当于是并发的休眠时间，不会叠加

## 3、线程设置优先级别

### 5.线程优先级别thread1.setPriority

InterRunnable myThred1 = **new** InterRunnable();

Thread thread1 = **new** Thread(myThred1,"线程1");

Thread thread2 = **new** Thread(myThred1,"线程2");

Thread thread3 = **new** Thread(myThred1,"线程3");

//调用线程体

thread1.setPriority(Thread.*NORM\_PRIORITY*);

thread2.setPriority(Thread.*MAX\_PRIORITY*);

thread3.setPriority(Thread.*MIN\_PRIORITY*);

thread1.start();

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*8");

thread2.start();

thread3.start();

}

### 5.1测试结果



### 5.2main的优先级，是普通的优先级

**public** **static** **void** main(String args[])

{

InterRunnable myThred1 = **new** InterRunnable();

Thread thread1 = **new** Thread(myThred1,"线程1");

Thread thread2 = **new** Thread(myThred1,"线程2");

Thread thread3 = **new** Thread(myThred1,"线程3");

//调用线程体

thread1.setPriority(Thread.*NORM\_PRIORITY*);

thread2.setPriority(Thread.*MAX\_PRIORITY*);

thread3.setPriority(Thread.*MIN\_PRIORITY*);

thread1.start();

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*8");

thread2.start();

thread3.start();

System.*out*.println(Thread.*currentThread*().getName());

System.*out*.println(Thread.*MAX\_PRIORITY*+"max");

System.*out*.println(Thread.*MIN\_PRIORITY*+"min");

System.*out*.println(Thread.*NORM\_PRIORITY*+"norm");

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

线程1正在运行

线程1正在运行

线程1正在运行

线程1正在运行

线程1正在运行

main

线程2正在运行

10max

线程2正在运行

1min

线程2正在运行

5norm

线程2正在运行

线程2正在运行

线程3正在运行

线程3正在运行

线程3正在运行

线程3正在运行

线程3正在运行

### 5.3测试

## 3.2、解释优先级

首先说 线程优先级，并不能保证优先级高的先运行，也不保证优先级高的更多的分配[CPU时间](https://www.baidu.com/s?wd=CPU%E6%97%B6%E9%97%B4&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y1PjNbuhwhn1PbrHm1uAw90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPH0LP1R4PWfv)，只是对系统的建议而已，到底运行哪个，是操作系统决定的，都不是java说了算的。

但这些优先级需要[操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ydnj6YP17-nj6suy79PWm0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtkrjf3njfYnjn)的支持。不同[操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ydnj6YP17-nj6suy79PWm0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtkrjf3njfYnjn)上优先级并不相同

**线程的优先级仍然无法保障线程的执行次序。只不过，优先级高的线程获取CPU资源的概率较大，优先级低的并非没机会执行。**

所谓线程优先级，个人理解只是对资源竞争的一个辨别依据，就是说当多个线程竞争资源时，具有较高优先级的线程会优先执行。  
楼主的线程执行过程太短、逻辑太简单，基本不存在竞争问题，当然看不出优先级的作用……  
其实所谓优先级只有在长期稳定运行的系统中才有用，这个很难测试的…

## 4、同步和死锁

### 1、这两个方法来自不同的类分别是Thread和Object

### 2、最主要是sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法。这样其他对象才能使用notify哦(个人理解)

### 3、wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在

　　任何地方使用（使用范围）

　　synchronized(x){

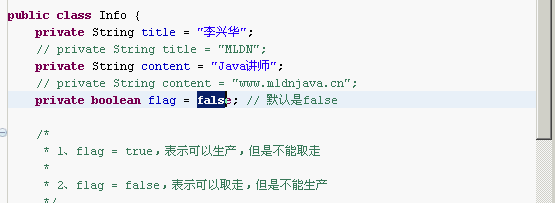
　　x.notify()

　　//或者wait()

}

#### Supper.wait. 让其他的线程先执行

#### Supper.notify 唤醒这个线程



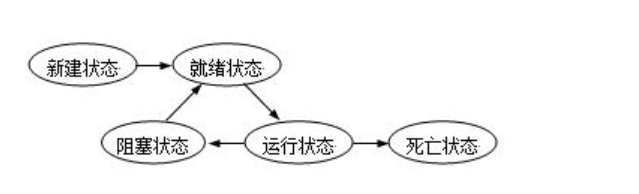




### 4、sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常

### ca5、wait属于Object的成员方法，一旦一个对象调用了wait方法，必须要采用notify()和notifyAll()方法唤醒该进程;如果线程拥有某个或某些对象的同步锁，那么在调用了wait()后，这个线程就会释放它持有的所有同步资源，而不限于这个被调用了wait()方法的对象。wait()方法也同样会在wait的过程中有可能被其他对象调用interrupt()方法而产生

### 5、线程的5中状态



### 6、同步，异步

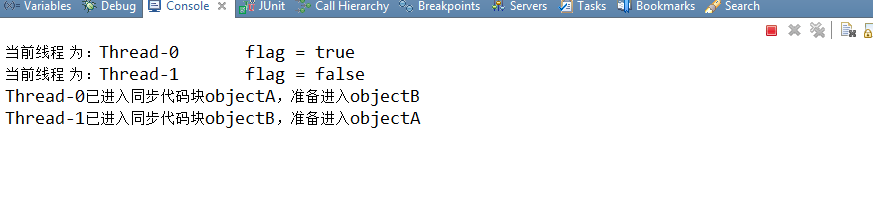
同步交互：指发送一个请求,需要等待返回,然后才能够发送下一个请求，有个等待过程；

异步交互：指发送一个请求,不需要等待返回,随时可以再发送下一个请求，即不需要等待。   
区别：一个需要等待，一个不需要等待，在部分情况下，我们的项目开发中都会优先选择不需要等待的异步交互方式。

哪些情况建议使用同步交互呢？比如银行的转账系统，对数据库的保存操作等等，都会使用同步交互操作，其余情况都优先使用异步交互

### 7、死锁代码，过多的同步造成死锁

|  |
| --- |
| /\*\*  线程thread1占有资源objectA，线程thread2占有资源objectB，  当两个线程发出请求时，由于所请求的资源都在对方手中，从而发生线程阻塞，造成了线程的死锁。  \*/  **public** **class** DeadLockThread **implements** Runnable {  **private** **static** **final** Object *objectA* = **new** Object();  **private** **static** **final** Object *objectB* = **new** Object();  **private** **boolean** flag;  @Override  **public** **void** run() {  String threadName = Thread.*currentThread*().getName();  System.*out*.println("当前线程 为：" + threadName + "\tflag = " + flag);  **if** (flag) {  **synchronized** (*objectA*) {  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  System.*out*.println(threadName + "已进入同步代码块objectA，准备进入objectB");  **synchronized** (*objectB*) {  System.*out*.println(threadName + "已经进入同步代码块objectB");  }  }  } **else** {  **synchronized** (*objectB*) {  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  System.*out*.println(threadName + "已进入同步代码块objectB，准备进入objectA");  **synchronized** (*objectA*) {  System.*out*.println(threadName + "已经进入同步代码块objectA");  }  }  }  }  **public** **static** **void** main(String[] args) {  DeadLockThread deadlock1 = **new** DeadLockThread();  DeadLockThread deadlock2 = **new** DeadLockThread();  deadlock1.flag = **true**;  deadlock2.flag = **false**;  Thread thread1 = **new** Thread(deadlock1);  Thread thread2 = **new** Thread(deadlock2);  thread1.start();  thread2.start();  } |



### 8、死锁产生的原因和避免

#### 1.原因

一是系统提供的资源数量有限，不能满足每个进程的使用；二是多道程序运行时，进程推进顺序不合理。三是资源分配不当等

#### 2.死锁的四个必要条件

1） 互斥条件：（2） 请求与保持条件：（3） 不剥夺条件:（4） 循环等待

#### 3.解除死锁

1、资源剥夺法；2、撤消进程法

### 9、Thread的run（）与start（）的区别

1、Thread.start()方法(native)启动线程，使之进入就绪状态，当cpu分配时间该线程时，由JVM调度执行run()方法。

2、相当于玩游戏机，只有一个游戏机（cpu），可是有很多人要玩，于是，start是排队！等CPU选中你就是轮到你，你就run（），当CPU的运行的时间片执行完，这个线程就继续排队，等待下一次的run（）。

1.start（）方法来启动线程，真正实现了多线程运行。这时无需等待run方法体代码执行完毕，可以直接继续执行下面的代码；通过调用Thread类的start()方法来启动一个线程， 这时此线程是处于就绪状态， 并没有运行。 然后通过此Thread类调用方法run()来完成其运行操作的， 这里方法run()称为线程体，它包含了要执行的这个线程的内容， Run方法运行结束， 此线程终止。然后CPU再调度其它线程。  
2.run（）方法当作普通方法的方式调用。程序还是要顺序执行，要等待run方法体执行完毕后，才可继续执行下面的代码； 程序中只有主线程——这一个线程， 其程序执行路径还是只有一条， 这样就没有达到写线程的目的。

记住：多线程就是分时利用CPU，宏观上让所有线程一起执行 ，也叫并发

### 10、Thread.yield( ) 谦让

Java线程中有一个Thread.yield( )方法，很多人翻译成线程让步。顾名思义，使当前线程从执行状态（运行状态）变为可执行态（就绪状态）。就是说当一个线程使用了这个方法之后，它就会把自己CPU执行的时间让掉，让自己或者其它的线程运行。（也就是谁先抢到谁执行）

### 11、线程的好处

资源利用率更好

### 12、synchronized和ReentrantLock的区别

synchronized是和if、else、for、while一样的关键字，

ReentrantLock是类，这是二者的本质区别。既然ReentrantLock是类，那么它就提供了比synchronized更多更灵活的特性，可以被继承、可以有方法、可以有各种各样的类变量，ReentrantLock比synchronized的扩展性体现在几点上：

（1）ReentrantLock可以对获取锁的等待时间进行设置，这样就避免了死锁

（2）ReentrantLock可以获取各种锁的信息,

synchronize obj.wait(),obj.nofiy()的替代品

### 13、死锁与活锁的区别，死锁与饥饿的区别？

死锁就是，线程A占用资源，线程B也占用资源，但是都不释放

活锁，就是A和B 都占用资源，但是又同时释放，这样来来回回就是活锁

饥饿。比如高优先级的线程，一直占用线程，低优先级的一直不能得到线程

# 87、[java创建线程的三种方式及其对比](http://blog.csdn.net/longshengguoji/article/details/41126119)

## 1、runnable和callable的比较

1、两者最大的不同点是：实现Callable接口的任务线程能返回执行结果；而实现Runnable接口的任务线程不能返回结果；

2、Callable接口的call()方法允许抛出异常；而Runnable接口的run()方法的异常只能在内部消化，不能继续上抛；

3、行Callable任务可以拿到一个Future对象，Future 表示异步计算的结果。它提供了检查计算是否完成的方法，以等待计算的完成，并获取计算的结果。计算完成后只能使用 get 方法来获取结果，如果线程没有执行完，Future.get()方法可能会阻塞当前线程的执行

|  |
| --- |
| **public** **class** CallableTest {  @Test  **public** **void** testStart() **throws** InterruptedException, ExecutionException{    Callable<String> callable = **new** CallableTaskWithResult(45);    FutureTask<String> task = **new** FutureTask<String>(callable);    **new** Thread(task).start();    String result = task.get();  System.*out*.println(result);  }    **class** CallableTaskWithResult **implements** Callable<String> {  **private** **int** id;  **public** CallableTaskWithResult(**int** id) {  **this**.id=id;  }    @Override  **public** String call() **throws** Exception {  System.*out*.println(Thread.*currentThread*().getName());  **return** "id为："+id;  }  }  } |

# 99.dwr和ajax

它就是对ajax进行了封装，ajax是通过js发出请求，被Servlet拦截下来，通过Servlet来与持久层打交道，而DWR就是对这一层Servlet进行了封装，使得js可以直接调用持久层的方法。

 ajax:配置少，[**js**](http://lib.csdn.net/base/javascript)代码写得多  ;dwr配置多，js代码很少

DWR是基于ajax技术的开源java技术框架，节省了java端和js端的很多重复性的工作，对于业务量复杂并且基于ajax交互获取数据的业务建议选择dwr技术，企业级项目适合，而对于只有少部分ajax交互的业务就还是用原生ajax技术封装下请求就可以了

# 100、java内存溢出

## 1、原因

1.数据量过于庞大；死循环 ；静态变量和静态方法过多；递归；无法确定是否被引用的对象；

2.虚拟机不回收内存（内存泄漏）；

## 2、解决方法

一个是优化程序代码，如果业务庞大，逻辑复杂，尽量减少全局变量的引用，让程序使用完变量的时候释放该引用能够让垃圾回收器回收，释放资源。  
二就是物理解决，增大物理内存，然后通过：-Xms256m -Xmx256m -XX:MaxNewSize=256m -XX:MaxPermSize=256m的修改

内存溢出和内存泄漏

内存溢出是指已有的数据超过了其获得到的内存所能存储的范围，比如用一个字节存放1000这个数字就属于内存溢出。比如说你申请了一个integer,但给它存了long才能存下的数，那就是内存溢出。

Java内存泄漏就是没有及时清理内存垃圾，导致系统无法再给你提供内存资源（内存资源耗尽）。Java内存泄露是说程序逻辑问题,造成申请的内存无法释放.这样的话无论多少内存,早晚都会被占用光的.最简单的例子就是死循环了.由于程序判断错误导经常发生此事。

# 86、java中会存在内存泄漏吗，请简单描述

**内存泄露**就是指一个不再被程序使用的对象或变量一直被占据在内存中

1.长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露，尽管短生命周期对象已经不再需要，但是因为长生命周期对象持有它的引用而导致不能被回收，这就是java中内存泄露的发生场景，通俗地说，就是程序员可能创建了一个对象，以后一直不再使用这个对象，这个对象却一直被引用，即这个对象无用但是却无法被垃圾回收器回收的，这就是java中可能出现内存泄露的情况，**例如，缓存系统，我们加载了一个对象放在缓存中(例如放在一个全局map对象中)，然后一直不再使用它，这个对象一直被缓存引用，但却不再被使用。**

检查java中的内存泄露，一定要让程序将各种分支情况都完整执行到程序结束，然后看某个对象是否被使用过，如果没有，则才能判定这个对象属于内存泄露。

# 101 File

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** IOException {    /\*\*  \* 1、已知路径创建 相关文件 前提，必须存在路径  \*/  // createFile();    /\*\*  \* 1.2、创建有父亲目录的文件  \*  \*/  // createHavaDirectoryFile();    /\*\*  \* 2、判断是不是文件或者目录  \* 如果上述文件、目录不存在，也不会报错  \*/  // judgeFileOrDirectory(directoryName);    /\*\*  \* 3、  \* 判断是不是目录  \* 创建目录，即使路径不存在，也会创建相关路径，因为是mkdirs  \*/  // createDirectory();    /\*\*  \* 4、复制文件  \* 第一个文件必须存在，  \* 第二个文件可以不存在,但是路径必须存在  \*/  // copyFile("D:/test/healerjean/file.txt","D:/test/copyFile/file2.txt");      /\*\*  \* 5、删除目录  \*  \*/  // File file = new File("D:/test/healerjean");  // delteFile(file);  }      /\*\*  \* 1、  \* 已知路径创建 相关文件  \* 前提，必须存在路径  \*/  **private** **static** **void** createFile() **throws** IOException {    File fileNameTxt = **new** File("D:/test/healerjean/file.txt");    **if**(!fileNameTxt.exists()){  fileNameTxt.createNewFile();  }  }      /\*\*  \* 1.2、创建有目录的文件  \*  \*/  **private** **static** **void** createHavaDirectoryFile() **throws** IOException {  File directoryName = **new** File("D:/test/healerjean/file");  //通过父的目录引入文件 a1 是父类 (只要是之前的父类以上都是可以的)  File directoryNameTxt = **new** File(directoryName,"directoryName.txt");    **if**(!directoryNameTxt.exists()){  directoryNameTxt.createNewFile();  }  System.*out*.println("创建有目录的文件成功");  }    /\*\*  \* 2、  \* 判断是不是文件或者目录  \* 如果上述文件、目录不存在，则以下都不执行，同时也不会报错  \*/  **private** **static** **void** judgeFileOrDirectory(File path) {    System.*out*.println("测试开始");  **if**(path.isFile()){  System.*out*.println(path.getPath()+"是一个文件");  }  **else** **if**(path.isDirectory()){  System.*out*.println(path.getPath()+"是一个目录");  }**else** {  System.*out*.println(path.getPath()+"不是文件也不是目录");  }  }        /\*\*  \* 3、  \* 判断是不是目录  \* 创建目录，即使路径不存在，也会创建相关路径，因为是mkdirs  \*/  **private** **static** **void** createDirectory() {  //引入目录  File directoryName = **new** File("D:/test/healerjean/file");    **if**(!directoryName.exists()){  directoryName.mkdirs();  System.*out*.println(directoryName.getPath()+"创建目录成功");  }    }    /\*\*  \* 4、复制文件  \* 第一个文件必须存在，  \* 第二个文件可以不存在,但是路径必须存在 (FileOutputStream)  \*/  **public** **static** **void** copyFile(String string,String string2) **throws** IOException{  FileInputStream ins = **new** FileInputStream(string);  FileOutputStream out = **new** FileOutputStream(string2);  **byte**[] b = **new** **byte**[1024];  **int** n=0;  **while**((n=ins.read(b))!=-1){  out.write(b, 0, n);  }    ins.close();  out.close();  System.*out*.println("复制文件成功");  }  /\*\*  \* 5、删除多个目录  \*/    **public** **static** **void** delteFile(File file){  //file.listFiles()是获取file这个对象也就是file这个目录下面的文件和文件夹的集合  File[] files=file.listFiles();  **for**(File f:files)  {  **if**(f.isDirectory())//递归调用  {  *delteFile*(f);  }  **else** {  f.delete();  }  }  file.delete();  System.*out*.println("删除目录成功");  }  /\*\*  \* 6 、目录下读取文件内容进行匹配  \* **@throws** Exception  \*  \*/  **private** **static** **void** getFileValue(File file,String content) **throws** Exception {  //  //file.listFiles()是获取file这个对象也就是file这个目录下面的文件和文件夹的集合  File[] files=file.listFiles();  **for**(File f:files)  {  **if**(f.isDirectory())//递归调用  {  *getFileValue*(f,content);  }  **else** {  String path = f.getPath();  String xml = path.substring(path.lastIndexOf(".")+1, path.length());  /\* if(xml.equals("xml")){\*/  FileInputStream fInputStream = **new** FileInputStream(f);  ByteArrayOutputStream outStream = **new** ByteArrayOutputStream();    **byte**[] buffer = **new** **byte**[1024];  **int** len = 0;  **while**((len=fInputStream.read(buffer))!=-1){  outStream.write(buffer, 0, len);  }  String str = **new** String(outStream.toByteArray(),"utf-8");  // System.out.println("文件名为"+f.getName()+"/n的内容为"+str);  // if(str.contains("<typecode>9208</typecode>")&&str.contains("<appcode>0000</appcode>")){  // if(str.contains("<typecode>1103</typecode>")&&!str.contains("<appcode>0000</appcode>")){  **if**(str.contains(content)){  System.*out*.println("第"+*i*+"个文件名为\n"+f.getPath());  System.*out*.println("内容为\n"+str);  *i*++;  }  /\* } \*/    }  }  }    /\*\*  \* 7、获取txt文件内容  \*  \*/  **private** **static** String getFileTextValue(String filePath) **throws** Exception {  //  FileInputStream fileInputStream = **new** FileInputStream(filePath);  ByteArrayOutputStream byteOutputStream = **new** ByteArrayOutputStream();  **int** len = 0;  **byte**[] buffer = **new** **byte**[1024];  **while**((len = fileInputStream.read(buffer))!=-1){  byteOutputStream.write(buffer, 0, len);  }  String txtValue = **new** String(byteOutputStream.toByteArray());  System.*out*.println(txtValue);  **return** txtValue;  }  /\*\*  \* 8、按照行,一行一行读取txt内容  \*/  **private** **static** String getFileReadLineTextValue(String filePath) **throws** Exception {    FileInputStream fileInputStream = **new** FileInputStream(filePath);  ByteArrayOutputStream byteOutputStream = **new** ByteArrayOutputStream();  **int** len = 0;  **byte**[] buffer = **new** **byte**[1024];  **while**((len = fileInputStream.read(buffer))!=-1){  byteOutputStream.write(buffer, 0, len);  }  **byte**[] txtByteArray = (byteOutputStream.toByteArray());    BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(**new** ByteArrayInputStream(txtByteArray),"utf-8"));  String lineVal =**null** ;  **int** flagLength = 0;  **while**((lineVal = reader.readLine()) != **null** ){    flagLength++;  **if**(flagLength==1){  System.*out*.println("第"+flagLength+"行的内容为"+lineVal);  }  **else**{  System.*out*.println("第"+flagLength+"行的内容为"+lineVal);  }    }      **return** **null**;  }    /\*\*  \* 9、根据字符串生成内容  \* 只要修改code city testType typecChoice directChoice  \*/  **public** **static** **void** getTxtByTextContent(String textContext,String fileName,String path){  **try** {    File fileDirectory = **new** File(path);  **if**(!fileDirectory.exists()){  fileDirectory.mkdirs();  }  File fileTxt = **new** File(path+"/"+fileName) ;  **if**(!fileTxt.exists()){  fileTxt.createNewFile();  }  FileOutputStream outputStream = **new** FileOutputStream(fileTxt);    **byte**[] buffer = textContext.getBytes("utf-8");    outputStream.write(buffer);  System.*out*.println("成功"+path+"/"+fileName);    } **catch** (FileNotFoundException e) {  //  e.printStackTrace();  }**catch** (UnsupportedEncodingException e) {  //  e.printStackTrace();  } **catch** (IOException e) {  //  e.printStackTrace();  }      }      /\*\*  \* 10、操作系统 路径符号判断  \* **@author** 作者 HealerJean:  \* **@version** 创建时间：2017年11月1日 上午9:32:49  \*  \*/  **public** **static** **void** testSeparator(){  /\*\*  \* 1、判断哪个系统  \*/    File directoryName = **new** File("D:"+File.*separator*+"test/txt.txt");  System.*out*.println(directoryName.getPath());    String os = System.*getProperty*("os.name");  **if**(os.startsWith("Win")){  System.*out*.printf("This system is windows");  System.*out*.println(":::"+os);  }**else** { //linux    }  /\*\*  \* 2、系统路径的代表符号  \* / linux  \* \ windows  \*/  String fileSeparator = System.*getProperty*("file.separator");  System.*out*.println(fileSeparator);    System.*out*.println(""+File.*separator*);  }  } |

# 102、String StringBuffer

## 1、测试string 和byte数组的转化

/\*\*

\* 3、测试string 和byte数组的转化

\*/

**public** **static** **void** testbyteArray(){

String str = "HealerJean";

**byte**[] byteArray;

byteArray = str.getBytes("utf-8");

String strByte = **new** String (byteArray,"UTF-8");

System.*out*.println("byte数组："+byteArray+"\n"+"string:"+strByte);

}

## 2、String 和 StringBuffer之间的转化

**public** **static** **void** strToBuffer() {

//String -> StringBuffer 1、构造方法 2、通过append方法

//创建一个String对象

String str = "Hi Java!";

System.*out*.println(str);

//方式一：构造方法

StringBuffer buffer = **new** StringBuffer(str);

System.*out*.println(buffer);

//方式二：通过append方法

StringBuffer buffer2 = **new** StringBuffer();

buffer2.append(str);

System.*out*.println(buffer2);

StringBuffer sBuffer = **new** StringBuffer("123");

//报错 sBuffer = "12"; 这样就变成了string

//StringBuffer -> String

//创建一个StringBuffer对象 1、构造方法 2、toString方法

StringBuffer buffer3 = **new** StringBuffer();

buffer3.append("Happy birthday Java!");

System.*out*.println(buffer3);

//方式一：通过构造方法

String str2 = **new** String(buffer3);

System.*out*.println(str2);

//方式二：通过toString方法

String str3 = buffer3.toString();

System.*out*.println(str3);

}

## 3、测试string 可不可以 +，不可以加

**private** **static** **void** teststringadd() {

//未提前定义好对象

//String m = m +"abc"; //错误的

//如果提前定义对象，是可以的 ,但是short 则只能 b +

String a = "123";

String b = "";

b += "abc";

b = b + "a";

}

## 4、length 括号 和size，数组没有括号，变量有括号

|  |
| --- |
| **public** **static** **void** lengthAndSize(){  String string = "abcdefg" ;  String strArray[] = {"1","2"};    System.*out*.println(string.length());  System.*out*.println(strArray.length);  //错误 System.out.println(strArray.size);  } |

## 5、****String... parem表示不定参数，也就是调用这个方法的时候这里可以传入多个String对象****

|  |
| --- |
| /\*\*  \* 8、string...  \*/  **public** **static** **void** printString(String... str){  **if**(str.length==0){  System.*out*.println("没有传参数。");  }  **for** (**int** i = 0; i < str.length; i++) {  System.*out*.printf(str[i]);  }  }    **public** **static** **void** printStringMain() {  *printString*();  System.*out*.println("==========");  *printString*(**new** String[]{"我","和","你"});  } |

## ****6、数组的定义****

|  |
| --- |
| **public** **static** **void** array(){  //1、  **int**[] a = **new** **int**[3]; //定义一个数组  a[0] = 1;  a[1] = 2;  a[2] = 3;  // a[4] = 5; //超出限制 报错  //2、动态初始化  **int** b[] ;  // b[1] = 2; 没有给定大小，报错  b = **new** **int**[5];  //3、 静态初始化  **int** c[] = {1,4,2,3};    //4、new 并直接静态赋值，不可写入大小  // int d[] = new int[5]{1,2,3};  **int** d[] = **new** **int**[]{1,2,3};  } |

## ****7、截取字节和char的说明，substring截取的是char****

|  |
| --- |
| /\*\*\*  \* 10、substring 截取的是字节还是字符  \* String类的length()方法是以unicode统一字符集代码单元，换言之就是char的个数为来统计的。、因为java一个char类型可以存放一个汉字（2个字节）。所以使用subString等截取出来的子串都不会出现半个汉字的情况，    \* 而如果以字节byte来截取字符串，就会出现半个汉字的情况。  思路：汉字的unicode编码都是负数，如果这些负数字节在截取的字节数组中成对出现，说明不会截取到半个汉字。  如果不是成对出现，则会截取到半个汉字，需要将最后一个字节舍去。  \* 按字节数截取字符串, 不能截出半个汉字, 如果是半个汉字则舍去!，这里有讲究了，下面的直接数字大小-1就可以了。  \*/  @Test  **public** **void** testSubStringByteOr(){  String source = "我是一abcd个好人cccc";  **int** byteCount = 8; //表示截取多少个  **byte**[] byteArr = source.getBytes();  System.*out*.println("字节大小为"+byteArr.length);  **int** count = 0;  // 统计要截取的那部分字节中负数的个数  **for** (**int** i = 0; i < byteCount; i++) {  **if** (byteArr[i] < 0) {  System.*out*.println(i);  count++;  }  }  System.*out*.println("负数的个数为"+count);    // 负数成对出现 则不会出现半个汉字  **if** (count % 2 == 0)  System.*out*.println(**new** String(byteArr, 0, byteCount));  // 负数个数不是偶数，则有半个汉字  **else**  System.*out*.println(**new** String(byteArr, 0, byteCount - 1));  } |

## **8、String.intern()**

new String都是在堆上创建字符串对象。当调用 intern() 方法时，编译器会将字符串添加到常量池中（stringTable维护），并返回指向该常量的引用。

<https://blog.csdn.net/soonfly/article/details/70147205>

举例说明：

String str2 = new String("str")+new String("01");

str2.intern();

String str1 = "str01";

System.out.println(str2==str1);

在JDK 1.7下，当执行str2.intern();时，因为常量池中没有“str01”这个字符串，所以会在常量池中生成一个对堆中的“str01”的**引用**(***注意这里是引用 ，就是这个区别于JDK 1.6的地方。在JDK1.6下是生成原字符串的拷贝***)，而在进行String str1 = “str01”;字面量赋值的时候，常量池中已经存在一个引用，所以直接返回了该引用，因此str1和str2都指向堆中的同一个字符串，返回true。

String str2 = new String("str")+new String("01");

String str1 = "str01";

str2.intern();

System.out.println(str2==str1);

将中间两行调换位置以后，因为在进行字面量赋值（String str1 = “str01″）的时候，常量池中不存在，所以str1指向的常量池中的位置，而str2指向的是堆中的对象，再进行intern方法时，对str1和str2已经没有影响了，所以返回false。

# 103. SpringMVC与struts2的区别

1、springmvc基于方法开发的，struts2基于类开发的。springmvc将url和controller里的方法映射。映射成功后springmvc生成一个Handler对象，对象中只包括了一个method。方法执行结束，形参数据销毁。springmvc的controller开发类似web service开发。  
2、springmvc可以进行单例开发，并且建议使用单例开发，struts2通过类的成员变量接收参数，无法使用单例，只能使用多例。  
3、经过实际测试，struts2速度慢，在于使用struts标签，如果使用struts建议使用jstl。

# 104、过滤器和拦截器的区别：

**①拦截器是基于**[Java](http://lib.csdn.net/base/java)**的反射机制的，而过滤器是基于函数回调。  
　　②拦截器不依赖与servlet容器，过滤器依赖与servlet容器。  
　　③拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。  
　　④拦截器可以访问action上下文、值栈里的对象，而过滤器不能访问。  
　　⑤在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次。**

**⑥拦截器可以获取IOC容器中的各个bean，而过滤器就不行，这点很重要，在拦截器里注入一个service，可以调用业务逻辑。**

**过滤器，是在java web中，你传入的request,response提前过滤掉一些信息，或者提前设置一些参数，然后再传入servlet或者struts的 action进行业务逻辑，比如过滤掉非法url（不是login.do的地址请求，如果用户没有登陆都过滤掉）,或者在传入servlet或者struts的action前统一设置字符集，或者去除掉一些非法字符.**

**拦截器，是在面向切面编程的就是在你的service或者一个方法前，调用一个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，在你调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出异常的时候做业务逻辑的操作。**

**执行顺序：过滤前 - 拦截前 - Action处理 - 拦截后 - 过滤后。**

**个人认为过滤是一个横向的过程，首先把客户端提交的内容进行过滤(例如未登录用户不能访问内部页面的处理)；过滤通过后，拦截器将检查用户提交数据的验证，做一些前期的数据处理，接着把处理后的数据发给对应的Action；Action处理完成返回后，拦截器还可以做其他过程(还没想到要做啥)，再向上返回到过滤器的后续操作。**

|  |
| --- |
| **public class MyFilter extends HttpServlet implements Filter {**  **private String keywords[];**  **public void doFilter(ServletRequest arg0, ServletResponse arg1,**  **FilterChain arg2) throws IOException, ServletException {**  **String username=arg0.getParameter("username");**  **System.out.println("this is safe filter !");**  **System.out.println(username);**  **if(username!=null){**  **for(String key:keywords){**  **if(key.equals(username)){**  **arg0.setAttribute("err", username+"被第二个管理安全的过滤器拦截了");**  **arg0.getRequestDispatcher("/WEB-INF/tishi.jsp").forward(arg0, arg1);**  **return ;**  **}**  **}**  **}**  **arg2.doFilter(arg0, arg1);**  **}**  **public void init(FilterConfig arg0) throws ServletException {**  **keywords=arg0.getInitParameter("keywords").split(";");**  **}**  然后在web.xml配置过滤器还有设置要过滤的关键字  <init-param>  <param-name>keywords</param-name>  <param-value>xiaomao;xiaoxiao</param-value>  </init-param>  **}** |

# 105、arraylist用法

## 1、测试remove方法

|  |
| --- |
| /\*\*  \* 1、测试remove方法 remove(object 或者是 index)  \*/  @Test  **public** **void** RemoveMethod() {  List<String> empList = **new** ArrayList<String>();  empList.add("String one");  empList.add("String two");    String str1 = "String one";  empList.remove(str1);    System.*out*.println(empList.size()+empList.get(0)); // 1    Iterator it = empList.iterator();  **while**(it.hasNext()){  String emp = (String)it.next();  System.*out*.println(emp.toString());  }    empList.add("String one");  empList.remove(0);  System.*out*.println("remove 0 也就是去掉了 String two");  Iterator it2 = empList.iterator();  **while**(it2.hasNext()){  String emp = (String)it2.next();  System.*out*.println(emp.toString());  }  } |

## 2、size初始化为几 其实是没有意义的

|  |
| --- |
| /\*\*  \* 2、设定arraylist最大的size  \* ArrayList中没有值时，则size为0  \* 不管size初始化为几 其实是没有意义的，只会动态的的得到它的大小  \*/  @Test  **public** **void** NewArraylist(){    List<String> strs = **new** ArrayList<String>(2);  //里面没有值 自然之类得到的 大小为 0  System.*out*.println(strs.size());    strs.add("第一个size");  strs.add("第一个size");  strs.add("第一个size");  strs.add("第一个size");  System.*out*.println(strs.size());    } |

## 3、 list 放入有顺序

|  |
| --- |
| /\*\*  3、 list 放入有顺序  \*/  @Test  **public** **void** listHaveAddSort(){  List<Person> persons= **new** ArrayList<Person>();  Person person = **new** Person(0, "HealerJean");  Person person2 = **new** Person(0, "HealerJean");  persons.add(0,person);  persons.add(3,person2); //报错，因为没有 2  } |

## 4、list 排序 Collections.*sort,* new Comparator

|  |
| --- |
| /\*\*  4、list 排序  \*/  @Test  **public** **void** listSort(){  /\*\*  \* 1、list 内部Integer排序  \*/  List<Integer> nums = **new** ArrayList<Integer>();  nums.add(3);  nums.add(5);  nums.add(1);  nums.add(0);  System.*out*.println(nums);  Collections.*sort*(nums);  System.*out*.println("1、list 内部Integer排序");  System.*out*.println(nums);    /\*\*  \* 1、list 内部对象  \*/    List<SortEntry> sortEntries = **new** ArrayList<SortEntry>();  sortEntries.add(**new** SortEntry(23, 100));  sortEntries.add(**new** SortEntry(27, 98));  sortEntries.add(**new** SortEntry(29, 99));  sortEntries.add(**new** SortEntry(29, 98));  sortEntries.add(**new** SortEntry(22, 89));  Collections.*sort*(sortEntries, **new** Comparator<SortEntry>() {    @Override  **public** **int** compare(SortEntry o1, SortEntry o2) {  **int** i = o1.getScore() - o2.getScore(); //先按照年龄排序  **if**(i == 0){  **return** o1.getAge() - o2.getAge(); //如果年龄相等了再用分数进行排序  }  **return** i;  }  });  **for**(SortEntry stu : sortEntries){  System.*out*.println("score:" + stu.getScore() + ":age" + stu.getAge());  }  }    List<Long> ids = new ArrayList<>(); ids.add(1L); ids.add(3L); ids.add(4l); ids.add(8L); ids.add(5L); ids.add(10L); ids.add(10L); ids.add(19L); ids.add(18L); ids.add(14L); ids.add(12L);  //逆序 [12, 14, 18, 19, 10, 10, 5, 8, 4, 3, 1] Collections.reverse(ids); System.***out***.println("逆序 "+ids);  //默认升序 [1, 3, 4, 5, 8, 10, 10, 12, 14, 18, 19] Collections.sort(ids); System.***out***.println("默认升序"+ids);  //降序Collections.reverseOrder() [19, 18, 14, 12, 10, 10, 8, 5, 4, 3, 1] Collections.sort(ids, Collections.reverseOrder()); System.***out***.println("降序 "+ids); |

## 5、list 去重

|  |
| --- |
| /\*\*  5、list 去重  \*/  @Test  **public** **void** deleteSim(){    deleteSimSet();    delteSimIfContain();    delteSimAddAllSet();  }    /\*\*  1、set集合去重，不打乱顺序  \*/  **public** **void** deleteSimSet(){    System.*out*.println("1、set集合去重，不打乱顺序");  List<String> list = **new** ArrayList<String>();  list.add("aaa");  list.add("bbb");  list.add("aaa");  list.add("aba");  list.add("aaa");  Set set = **new** HashSet();  List newList = **new** ArrayList();  **for** (String cd:list) {  **if**(set.add(cd)){  newList.add(cd);  }  }  System.*out*.println( "去重后的集合： " + newList);  }    /\*\*  2、遍历后判断赋给另一个list集合  \*/  **public** **void** delteSimIfContain(){  System.*out*.println("2、遍历后判断赋给另一个list集合");  List<String> list = **new** ArrayList<String>();  list.add("aaa");  list.add("bbb");  list.add("aaa");  list.add("aba");  list.add("aaa");  List<String> newList = **new** ArrayList<String>();  **for** (String cd:list) {  **if**(!newList.contains(cd)){  newList.add(cd);  }  }  System.*out*.println( "去重后的集合： " + newList);  }    /\*\*  3、set.addAll();  list.add(Set)  \*/  **public** **void** delteSimAddAllSet(){  System.*out*.println("2、遍历后判断赋给另一个list集合");  List<String> list = **new** ArrayList<String>();  list.add("aaa");  list.add("bbb");  list.add("aaa");  list.add("aba");  list.add("aaa");  Set set = **new** HashSet();  List newList = **new** ArrayList();  set.addAll(list);  newList.addAll(set);  System.*out*.println( "去重后的集合： " + newList);  } |

## 6、clear清除list集合

|  |
| --- |
| /\*\*  \*  6、清除list集合 list.clear();  \*/  @Test  **public** **void** clear(){  List<String> list = **new** ArrayList<String>();  list.add("aaa");  list.add("bbb");  list.add("aaa");  list.add("aba");  list.add("aaa");    list.clear();  //结果为0  System.*out*.println( "list.clear list大小 " + list.size());  } |

## 7、 list 并集addAll，交集retainAll，差集removeAll

|  |
| --- |
| /\*\*  \*  7、list 并集addAll，交集retainAll，差集removeAll  \*/  @Test  **public** **void** retainAllAndremoveAll(){    java.util.List<Integer> list = **new** ArrayList<Integer>();  java.util.List<Integer> list1 = **new** ArrayList<Integer>();  list.add(1); list.add(2); list.add(3);    list1.add(2); list1.add(3); list1.add(4);    //list和list1的并集：  // list.addAll(list1); //1 2 3 2 3 4    //list和list1的交集：  // list.retainAll(list1); // 2 3  //list和list1的差集：  list.removeAll(list1); //1  **for** (Integer integer : list) {  System.*out*.println(integer);  }  } |

## 8、set方法根据索引位置插入并且替换数据

|  |
| --- |
| /\*\*  8、list set方法根据索引插入并且替换数据  \*/ @Test public void setInsert(){  String a="A",b="B",c="C",d="D",e="E";  List<String>list=new ArrayList<>();  list.add(a);  list.add(e);  list.add(d);  Iterator<String>first=list.iterator();  System.***out***.printf("修改前集合中的元素是：");  while(first.hasNext()){  System.***out***.print(first.next()+" ");  }   System.***out***.println();  String temp= list.set(1, b); //直接替换了上面的索引为1的元素2,并将原来位置上的数据返回  System.***out***.println(temp); //E  System.***out***.printf("set修改集合后的元素是：");  Iterator second=list.iterator();  while(second.hasNext()){  System.***out***.print(second.next()+" ");  }    System.***out***.println();  list.add(2,c);//在索引为2的位置插入了元素c，然后向后移动一位  Iterator second2=list.iterator();  System.***out***.printf("修改集合后的元素是：");  while(second2.hasNext()){  System.***out***.print(second2.next()+" ");  }  }  控制台  修改前集合中的元素是：A E D  E  set修改集合后的元素是：A B D  修改集合后的元素是：A B C D |

## 9、for遍历

ArrayList、LinkedList、HashMap中都有一个字段叫modCount。这个字段的用途

该字段被Iterator以及ListIterator的实现类所使用，如果该值被意外更改，Iterator或者ListIterator 将抛出ConcurrentModificationException异常，

1、在ArrayList中有个成员变量modCount，继承于AbstractArrAayList

数，每对List对象修改一次，也就每次add或者remove它的值都会加1.

Itr类里有一个成员变量expectedModCount，它的值为创建Itr对象的时候List的modCount值。

用此变量来检验在迭代过程中List对象是否被修改了，如果被修改了则抛出java.util.ConcurrentModificationException异常。

在每次调用Itr对象的next()或者remove方法的时候都会调用checkForComodification()方法进行一次检验，

checkForComodification()方法中做的工作就是比较expectedModCount 和modCount的值是否相等，如果不相等，

就认为还有其他对象正在对当前的List进行操作，那个就会抛出ConcurrentModificationException异常。

将list对象里面的“c"删除了，同时list对象的modCount值加1，但是Itr对象的expectedModCount没有变，他们肯定是不相等了。等再一次执行next()方法的时候调用了checkForComodification()方法，这时候就抛出异常了。

public E remove(int index) {  
 rangeCheck(index);  
  
 modCount++;  
 E oldValue = elementData(index);  
  
 int numMoved = size - index - 1;  
 if (numMoved > 0)  
 System.arraycopy(elementData, index+1, elementData, index,  
 numMoved);  
 elementData[--size] = null; // clear to let GC do its work  
  
 return oldValue;  
}

public static void main(String[] args) {  
 List<String> list = new ArrayList<String>();  
   
 list.add("a");  
 list.add("b");  
 list.add("c");  
 list.add("d");  
 list.add("e");  
 Iterator iterator = list.iterator();  
 while(iterator.hasNext()){  
 String str = (String) iterator.next();  
 if(str.equals("c")){  
  
 list.remove(str);  
 }else{  
 System.***out***.println(str);  
 }  
 }  
}

控制台：

a

Exception in thread "main" java.util.ConcurrentModificationException

b

at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:907)

at java.util.ArrayList$Itr.next(ArrayList.java:857)

at com.hlj.Arraylist.ListForEach.main(ListForEach.java:25)

此时将c变成d，它是通过Itr的对象的cursor（下一个索引）与List对象的size值来判断是否还有未迭代的对象，当遍历完“d"的时候cursor=4，删除”d"的时候，List对象的size就会减1，size首先为5，后来变为4，这时候cursor和size是相等的，hasNext()方法返回的是false，就认为遍历结束了，所以删除以后没有进去执行next()方法了，就没有抛出异常了，当然"e"也没有输出来。

a

b

c

　　为了避免这种异常，我们可以使用CopyOnWriteArrayList来代替ArrayList，CopyOnWriteArrayList支持并发访问，所以同时进行迭代和修改是没有问题的。

@Test  
 public void remove(){  
 List<String> list = new ArrayList();  
 list.add("aaaaaa");  
 list.add("bbbbbb");  
 list.add("cccccc");  
 list.add("dddddd");  
 list.add("eeeeee");  
  
 Iterator it = list.iterator();  
 //it.remove(); //删除的是上一个元素 IllegalStateException  
 int i = 0;  
 String s = null;  
 while(it.hasNext()){  
 if(i==2){  
// list.remove(it.next()); 如果用list.remove(it.next());会报异常checkForComodification

it.remove();  
 }  
 System.***out***.println("第"+i+"个元素"+it.next());  
 i++ ;  
 }  
 System.***out***.println("----------------");  
 Iterator it2 = list.iterator();  
 while(it2.hasNext()){  
 System.***out***.println(it2.next());  
 }  
  
 }

第0个元素aaaaaa

第1个元素bbbbbb

第2个元素cccccc

第3个元素dddddd

第4个元素eeeeee

----------------

aaaaaa

cccccc

dddddd

eeeeee

## 10、list截取长度

//包头不包尾比如1 2 曲子list index 0 1  
List<Long> idParams = ids.subList(key.intValue()-1, map.get(key).intValue());

# **1**07、classloader原理****

## 1．classLoader的介绍及加载过程

与普通程序不同的是，Java程序（class文件）并不是本地的可执行程序。当运行Java程序时，首先运行JVM（Java虚拟机），然后再把Java class加载到JVM里头运行，负责加载Java class的这部分就叫做Class Loader。所以classLoader的目的在于把class文件装入到jvm中。

## 2、那么classLoader又在那里的啦？又由谁调用呢？

其实classLoader只是jvm的一个实现的一部分。

Jvm提供的一个顶级的classLoader（bootStrap classLoader），

bootStrap classLoader负责加载java核心的API(也就是java的自身的jar包)以满足java程序最基本的需求。

Jvm还提供的两个classLoader，

其中Extension ClassLoader负责加载扩展的Java class(jar包)，Application ClassLoader负责加载应用程序自身的类（.class文件）。

而Extension ClassLoader和Application ClassLoader则由bootStrap classLoader加载。

## 3．classLoader加载的基本流程

   当运行一个程序的时候，JVM启动，运行bootstrap classloader，该ClassLoader加载java核心API（ExtClassLoader和AppClassLoader也在此时被加载），然后调用ExtClassLoader加载扩展API，最后AppClassLoader加载CLASSPATH目录下定义的Class，这就是一个程序最基本的加载流程。

## 4．classLoader加载的方式

   其实classLoader在加载class文件的时候就采用的双亲委托模式。每一个自定义ClassLoader都必须继承ClassLoader这个抽象类，而每个ClassLoader都会有一个parent ClassLoader，我们可以看一下ClassLoader这个抽象类中有一个getParent()方法，这个方法用来返回当前ClassLoader的parent。

# 108、continue，break

## 1、continenue 打断的是当前循环也就是J循环，进入下一个j

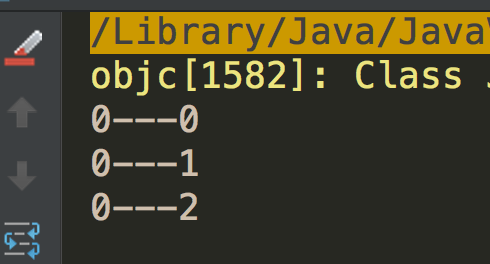
|  |
| --- |
| **public** **class** Continue {  **public** **static** **void** main(String[] args) {  Continue test2 = **new** Continue();  test2.add(5);  }  **private** **void** add(**int** a) {  //  **for** (**int** i = 1 ; i < 3 ; i++){  **for**(**int** j = 1 ; j < 3 ; j++){  **if**(a==5){  System.*out*.println(j);  **continue**;  }  System.*out*.println(i \* j); //不会执行的  }  }  }  // 1 2 1 2    } |

## 2、break，打断的是当前for循环，也就是打断j，直接执行下一个i

|  |
| --- |
| **public** **class** Break {  **public** **static** **void** main(String[] args) {  Break test2 = **new** Break();  test2.add(5);  }  **private** **void** add(**int** a) {  //  **for** (**int** i = 1 ; i < 3 ; i++){  **for**(**int** j = 1 ; j < 3 ; j++){  **if**(a==5){  System.*out*.println(j);  **break**;  }  System.*out*.println(i \* j);  }  }  }  // 1 1    } |

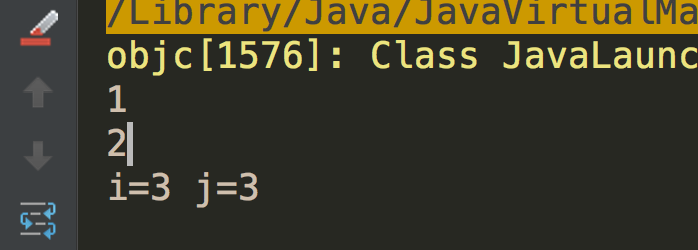
## 3、break 跳出最for外层循环 ，最外层写上标签

private void breakOut(){  
 //标签自己写  
 a:  
 for(int i=0;i<3;i++){  
 for(int j=0;j<3;j++){  
 System.***out***.println(i+"---"+j);  
 //当j=2时结束outer标识符指定的循环  
 if(j==2){  
 break a;  
 }  
 }  
 }  
}



## 4、continue 结束当前继续下一个for，也就是i中的，相当于直接 break

private void continueOut(int a) {  
 //  
 int i = 1;  
 int j = 1;  
 for (; i < 3 ; i++){  
 a: //直接结束a所在位置的当次循环，直接进入下一个循环中  
 for( ; j < 3 ; j++){  
 if(a==5){  
 System.***out***.println(j);  
 continue a;  
 }  
 }  
 }  
 System.***out***.println("i="+i+" j="+j);  
  
}



## 6、return 控制台没有内容

public class While {  
 public static void main(String[] args) {  
 int i= 0;  
 int j = 0;  
 while (i<6) {  
 i++;  
 if(i==4){  
 return; //结束全部的代码,不带返回值的return语句只能用于返回类型为void的函数。  
 }  
  
 }  
 System.***out***.println("i="+i+" j="+j);  
 }  
  
 public String returnTest(){  
  
 //报错 return ;  
 return null;  
 }  
}

# 109、java创建对象的集中方式

<https://www.cnblogs.com/wxd0108/p/5685817.html>

|  |  |
| --- | --- |
| 使用new关键字 | } → 调用了构造函数 |
| 使用Class类的newInstance方法 | } → 调用了构造函数 |
| 使用Constructor类的newInstance方法 | } → 调用了构造函数 |
| 使用clone方法 | } → 没有调用构造函数 |
| 使用反序列化 | } → 没有调用构造函数 |

## 1.使用new关键字

这是最常见也是最简单的创建对象的方式了。通过这种方式，我们可以调用任意的构造函数(无参的和带参数的)。

Employee emp1 = new Employee();

0: new #19 // class org/programming/mitra/exercises/Employee

3: dup

4: invokespecial #21 // Method org/programming/mitra/exercises/Employee."":()V

## 2.使用Class类的newInstance方法

我们也可以使用Class类的newInstance方法创建对象。这个newInstance方法调用无参的构造函数创建对象。

我们可以通过下面方式调用newInstance方法创建对象:

Employee emp2 = (Employee) Class.forName("org.programming.mitra.exercises.Employee").newInstance();

或者

Employee emp2 = Employee.class.newInstance();

51: invokevirtual #70 // Method java/lang/Class.newInstance:()Ljava/lang/Object;

## 3.使用Constructor类的newInstance方法

和Class类的newInstance方法很像， java.lang.reflect.Constructor类里也有一个newInstance方法可以创建对象。我们可以通过这个newInstance方法调用有参数的和私有的构造函数。

Constructor<Employee> constructor = Employee.class.getConstructor();

Employee emp3 = constructor.newInstance();

111: invokevirtual #80 // Method java/lang/reflect/Constructor.newInstance:([Ljava/lang/Object;)Ljava/lang/Object;

这两种newInstance方法就是大家所说的反射。事实上Class的newInstance方法内部调用Constructor的newInstance方法。这也是众多框架，如Spring、Hibernate、Struts等使用后者的原因。想了解这两个newInstance方法的区别，请看这篇[Creating objects through Reflection in Java with Example.](https://programmingmitra.blogspot.in/2016/05/creating-objects-through-reflection-in-java-with-example.html)

## 4.使用clone方法

无论何时我们调用一个对象的clone方法，jvm就会创建一个新的对象，将前面对象的内容全部拷贝进去。用clone方法创建对象并不会调用任何构造函数。

要使用clone方法，我们需要先实现Cloneable接口并实现其定义的clone方法。

Employee emp4 = (Employee) emp3.clone();

162: invokevirtual #87 // Method org/programming/mitra/exercises/Employee.clone ()Ljava/lang/Object;

## 5.使用反序列化

当我们序列化和反序列化一个对象，jvm会给我们创建一个单独的对象。在反序列化时，jvm创建对象并不会调用任何构造函数。  
为了反序列化一个对象，我们需要让我们的类实现Serializable接口

ObjectInputStream in = new ObjectInputStream(new FileInputStream("data.obj"));

Employee emp5 = (Employee) in.readObject();

261: invokevirtual #118 // Method java/io/ObjectInputStream.readObject:()Ljava/lang/Object;

# 110、spring注入的几种方式

**1、使用属性的setter方法注入  这是最常用的方式；**

**2、使用构造器注入；**

**3、静态工厂的方法注入**

**4、实例工厂的方法注入**

**5、使用Filed注入（用于注解方式）.**

## 1、set方式

|  |
| --- |
| 1. **package** com.bless.springdemo.action; 2. **public** **class** SpringAction { 3. //注入对象springDao 4. **private** SpringDao springDao; 5. //一定要写被注入对象的set方法 6. **public** **void** setSpringDao(SpringDao springDao) { 7. **this**.springDao = springDao; 8. } 10. **public** **void** ok(){ 11. springDao.ok(); 12. } 13. } |

随后编写spring的xml文件，<bean>中的name属性是class属性的一个别名，class属性指类的全名，因为在SpringAction中有一个公共属性Springdao，所以要在<bean>标签中创建一个<property>标签指定SpringDao。<property>标签中的name就是SpringAction类中的SpringDao属性名，ref指下面<bean name="springDao"...>，这样其实是spring将SpringDaoImpl对象实例化并且调用SpringAction的setSpringDao方法将SpringDao注入：

**Java代码  [收藏代码](javascript:void())**

1. <!--配置bean,配置后该类由spring管理-->
2. <bean name="springAction" **class**="com.bless.springdemo.action.SpringAction">
3. <!--(1)依赖注入,配置当前类中相应的属性-->
4. <property name="springDao" ref="springDao"></property>
5. </bean>
6. <bean name="springDao" **class**="com.bless.springdemo.dao.impl.SpringDaoImpl"></bean>

## 2、构造器注入

|  |
| --- |
| 1. **public** **class** SpringAction { 2. //注入对象springDao 3. **private** SpringDao springDao; 4. **private** User user; 6. **public** SpringAction(SpringDao springDao,User user){ 7. **this**.springDao = springDao; 8. **this**.user = user; 9. System.out.println("构造方法调用springDao和user"); 10. } 12. **public** **void** save(){ 13. user.setName("卡卡"); 14. springDao.save(user); 15. } 16. } |

在XML文件中同样不用<property>的形式，而是使用<constructor-arg>标签，ref属性同样指向其它<bean>标签的name属性：

**Xml代码  [收藏代码](javascript:void())**

1. <!--配置bean,配置后该类由spring管理-->
2. **<bean** name="springAction" class="com.bless.springdemo.action.SpringAction"**>**
3. <!--(2)创建构造器注入,如果主类有带参的构造方法则需添加此配置-->
4. **<constructor-arg** ref="springDao"**></constructor-arg>**
5. **<constructor-arg** ref="user"**></constructor-arg>**
6. **</bean>**
7. **<bean** name="springDao" class="com.bless.springdemo.dao.impl.SpringDaoImpl"**></bean>**
8. **<bean** name="user" class="com.bless.springdemo.vo.User"**></bean>**

  解决构造方法参数的不确定性，你可能会遇到构造方法传入的两参数都是同类型的，为了分清哪个该赋对应值，则需要进行一些小处理：

下面是设置index，就是参数位置：

## 3、静态工厂的方法注入

静态工厂顾名思义，就是通过调用静态工厂的方法来获取自己需要的对象，为了让spring管理所有对象，我们不能直接通过"工程类.静态方法()"来获取对象，而是依然通过spring注入的形式获取：

1. **package** com.bless.springdemo.factory;
3. **import** com.bless.springdemo.dao.FactoryDao;
4. **import** com.bless.springdemo.dao.impl.FactoryDaoImpl;
5. **import** com.bless.springdemo.dao.impl.StaticFacotryDaoImpl;
7. **public** **class** DaoFactory {
8. //静态工厂
9. **public** **static** **final** FactoryDao getStaticFactoryDaoImpl(){
10. **return** **new** StaticFacotryDaoImpl();
11. }
12. }

同样看关键类，这里我需要注入一个FactoryDao对象，这里看起来跟第一种注入一模一样，但是看随后的xml会发现有很大差别:

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** SpringAction {
2. //注入对象
3. **private** FactoryDao staticFactoryDao;
5. **public** **void** staticFactoryOk(){
6. staticFactoryDao.saveFactory();
7. }
8. //注入对象的set方法
9. **public** **void** setStaticFactoryDao(FactoryDao staticFactoryDao) {
10. **this**.staticFactoryDao = staticFactoryDao;
11. }
12. }

Spring的IOC配置文件，注意看<bean name="staticFactoryDao">指向的class并不是FactoryDao的实现类，而是指向静态工厂DaoFactory，并且配置 factory-method="getStaticFactoryDaoImpl"指定调用哪个工厂方法：

**Xml代码  [收藏代码](javascript:void())**

1. <!--配置bean,配置后该类由spring管理-->
2. **<bean** name="springAction" class="com.bless.springdemo.action.SpringAction" **>**
3. <!--(3)使用静态工厂的方法注入对象,对应下面的配置文件(3)-->
4. **<property** name="staticFactoryDao" ref="staticFactoryDao"**></property>**
5. **</property>**
6. **</bean>**
7. <!--(3)此处获取对象的方式是从工厂类中获取静态方法-->
8. **<bean** name="staticFactoryDao" class="com.bless.springdemo.factory.DaoFactory" factory-method="getStaticFactoryDaoImpl"**></bean>**

## 4、实例工厂的方法注入

实例工厂的意思是获取对象实例的方法不是静态的，所以你需要首先new工厂类，再调用普通的实例方法：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** DaoFactory {
2. //实例工厂
3. **public** FactoryDao getFactoryDaoImpl(){
4. **return** **new** FactoryDaoImpl();
5. }
6. }

那么下面这个类没什么说的，跟前面也很相似，但是我们需要通过实例工厂类创建FactoryDao对象：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** SpringAction {
2. //注入对象
3. **private** FactoryDao factoryDao;
5. **public** **void** factoryOk(){
6. factoryDao.saveFactory();
7. }
9. **public** **void** setFactoryDao(FactoryDao factoryDao) {
10. **this**.factoryDao = factoryDao;
11. }
12. }

最后看spring配置文件：

**Xml代码  [收藏代码](javascript:void())**

1. <!--配置bean,配置后该类由spring管理-->
2. **<bean** name="springAction" class="com.bless.springdemo.action.SpringAction"**>**
3. <!--(4)使用实例工厂的方法注入对象,对应下面的配置文件(4)-->
4. **<property** name="factoryDao" ref="factoryDao"**></property>**
5. **</bean>**
7. <!--(4)此处获取对象的方式是从工厂类中获取实例方法-->
8. **<bean** name="daoFactory" class="com.bless.springdemo.factory.DaoFactory"**></bean>**
9. **<bean** name="factoryDao" factory-bean="daoFactory" factory-method="getFactoryDaoImpl"**></bean>**

## 5、· 使用Filed注入（用于注解方式）.

# 111、mybatis与hibernate（orm对象关系映射）区别

1、对象和数据数据中的关系有完整的映射结构，并且包装了hql可以直接通过对象来 查询，mybatis是需要写sql的，

1、hibernate完全可以通过对象关系模型实现对数据库的操作，拥有完整的JavaBean对象与数据库的映射结构来自动生成sql。而mybatis仅有基本的字段映射，对象数据以及对象实际关系仍然需要通过手写sql来实现和管理。

2. hibernate数据库移植性远大于mybatis。

hibernate通过它强大的映射结构和hql语言，大大降低了对象与数据库（oracle、mysql等）的耦合性，而mybatis由于需要手写sql，因此与数据库的耦合性直接取决于程序员写sql的方法，如果sql不具通用性而用了很多某数据库特性的sql语句的话，移植性也会随之降低很多，成本很高。

3. hibernate拥有完整的日志系统，mybatis则欠缺一些。

hibernate日志系统非常健全，涉及广泛，包括：sql记录、关系异常、优化警告、缓存提示、脏数据警告等；而mybatis则除了基本记录功能外，功能薄弱很多。

4. mybatis相比hibernate需要关心很多细节

hibernate配置要比mybatis复杂的多，学习成本也比mybatis高。但也正因为mybatis使用简单，才导致它要比hibernate关心很多技术细节。mybatis由于不用考虑很多细节，开发模式上与传统jdbc区别很小，因此很容易上手并开发项目，但忽略细节会导致项目前期bug较多，因而开发出相对稳定的软件很慢，而开发出软件却很快。hibernate则正好与之相反。但是如果使用hibernate很熟练的话，实际上开发效率丝毫不差于甚至超越mybatis。

5. sql直接优化上，mybatis要比hibernate方便很多

由于mybatis的sql都是写在xml里，因此优化sql比hibernate方便很多。而hibernate的sql很多都是自动生成的，无法直接维护sql；虽有hql，但功能还是不及sql强大，见到报表等变态需求时，hql也歇菜，也就是说hql是有局限的；hibernate虽然也支持原生sql，但开发模式上却与orm不同，需要转换思维，因此使用上不是非常方便。总之写sql的灵活度上hibernate不及mybatis。

总结：

mybatis：小巧、方便、高效、简单、直接、半自动

hibernate：强大、方便、高效、复杂、绕弯子、全自动

mybatis：

1. 入门简单，即学即用，提供了数据库查询的自动对象绑定功能，而且延续了很好的SQL使用经验，对于没有那么高的对象模型要求的项目来说，相当完美。

2. 可以进行更为细致的SQL优化，可以减少查询字段。

3. 缺点就是框架还是比较简陋，功能尚有缺失，虽然简化了数据绑定代码，但是整个底层数据库查询实际还是要自己写的，工作量也比较大，而且不太容易适应快速数据库修改。

4. 二级缓存机制不佳。

hibernate：

1. 功能强大，数据库无关性好，O/R映射能力强，如果你对Hibernate相当精通，而且对Hibernate进行了适当的封装，那么你的项目整个持久层代码会相当简单，需要写的代码很少，开发速度很快，非常爽。

2. 有更好的二级缓存机制，可以使用第三方缓存。

3. 缺点就是学习门槛不低，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡取得平衡，以及怎样用好Hibernate方面需要你的经验和能力都很强才行。

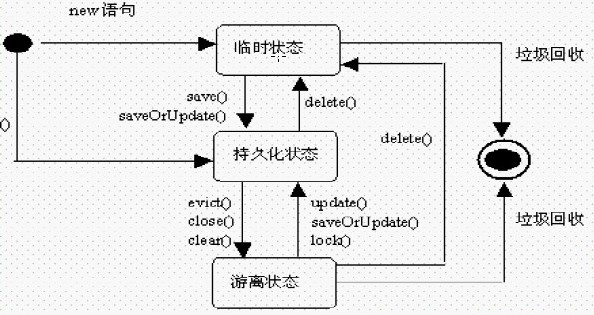
# 112、Hibernate有三种状态

## transient(瞬时状态)

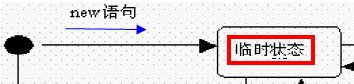
## persistent(持久化状态)

## detached(游离状态)。

　　它们之间有如下转换图来说明：



### 1、transient（瞬时状态）。



　　如上图所示，当new出来一个新对象，还没有保存到数据库中的时候，就是transient状态。

　　代码表示如下：

[复制代码](javascript:void(0);)

1 session=HibernateUtils.getSession();

2 tx=session.beginTransaction();

3 //Transient状态

4 User user=new User();

5 user.setName("Jessica");

6 user.setPassword("123");

7 user.setCreateTime(new Date());

8 user.setExpireTime(new Date());

9 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10 /\*

11 \* 以上user就是一个Transient(瞬时状态),此时user并没有被session进行托管，即在session的

12 \* 缓存中还不存在user这个对象，当执行完save方法后，此时user被session托管，并且数据库中存在了该对象

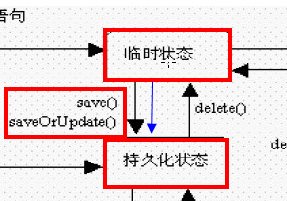
13 \* user就变成了一个Persistent(持久化对象)

14 \*/

15 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[复制代码](javascript:void(0);)

### 2、persistent(持久化状态)。



　　当临时状态的对象被执行save之后，就会被session托管，在session中有一个map存放着user对象，也就是说user对象被session引用着，被session纳入管理了。此时的user就处于持久状态了。

[复制代码](javascript:void(0);)

1 session=HibernateUtils.getSession();

2 tx=session.beginTransaction();

3 //Transient状态

4 User user=new User();

5 user.setName("Jessica");

6 user.setPassword("123");

7 user.setCreateTime(new Date());

8 user.setExpireTime(new Date());

9 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10 /\*

11 \* 以上user就是一个Transient(瞬时状态),此时user并没有被session进行托管，即在session的

12 \* 缓存中还不存在user这个对象，当执行完save方法后，此时user被session托管，并且数据库中存在了该对象

13 \* user就变成了一个Persistent(持久化对象)

14 \*/

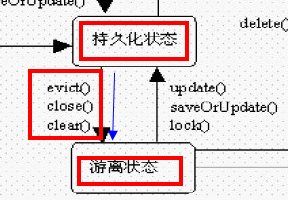
15 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

16 //Persistent状态

17 session.save(user);

[复制代码](javascript:void(0);)

### 3、detached(游离状态)。



　　当持久状态的对象user,在session关闭之后就会变成有游离状态。

[复制代码](javascript:void(0);)

1 Session session=null;

2 Transaction tx=null;

3

4 try {

5 session=HibernateUtils.getSession();

6 tx=session.beginTransaction();

7 //Transient状态

8 User user=new User();

9 user.setName("Jessica");

10 user.setPassword("123");

11 user.setCreateTime(new Date());

12 user.setExpireTime(new Date());

13 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

14 /\*

15 \* 以上user就是一个Transient(瞬时状态),此时user并没有被session进行托管，即在session的

16 \* 缓存中还不存在user这个对象，当执行完save方法后，此时user被session托管，并且数据库中存在了该对象

17 \* user就变成了一个Persistent(持久化对象)

18 \*/

19 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

20 //Persistent状态

21 session.save(user);

22

23 //进行修改操作，修改name=“Juyahong”

24 user.setName("Juyahong");

25 //提交事务

26 tx.commit();

27

28 } catch (Exception e) {

29 e.printStackTrace();

30 if (tx != null) {

31 tx.rollback();

32 }

33 } finally{

34

35 HibernateUtils.closeSession(session);

36 }

37

38 //detached状态

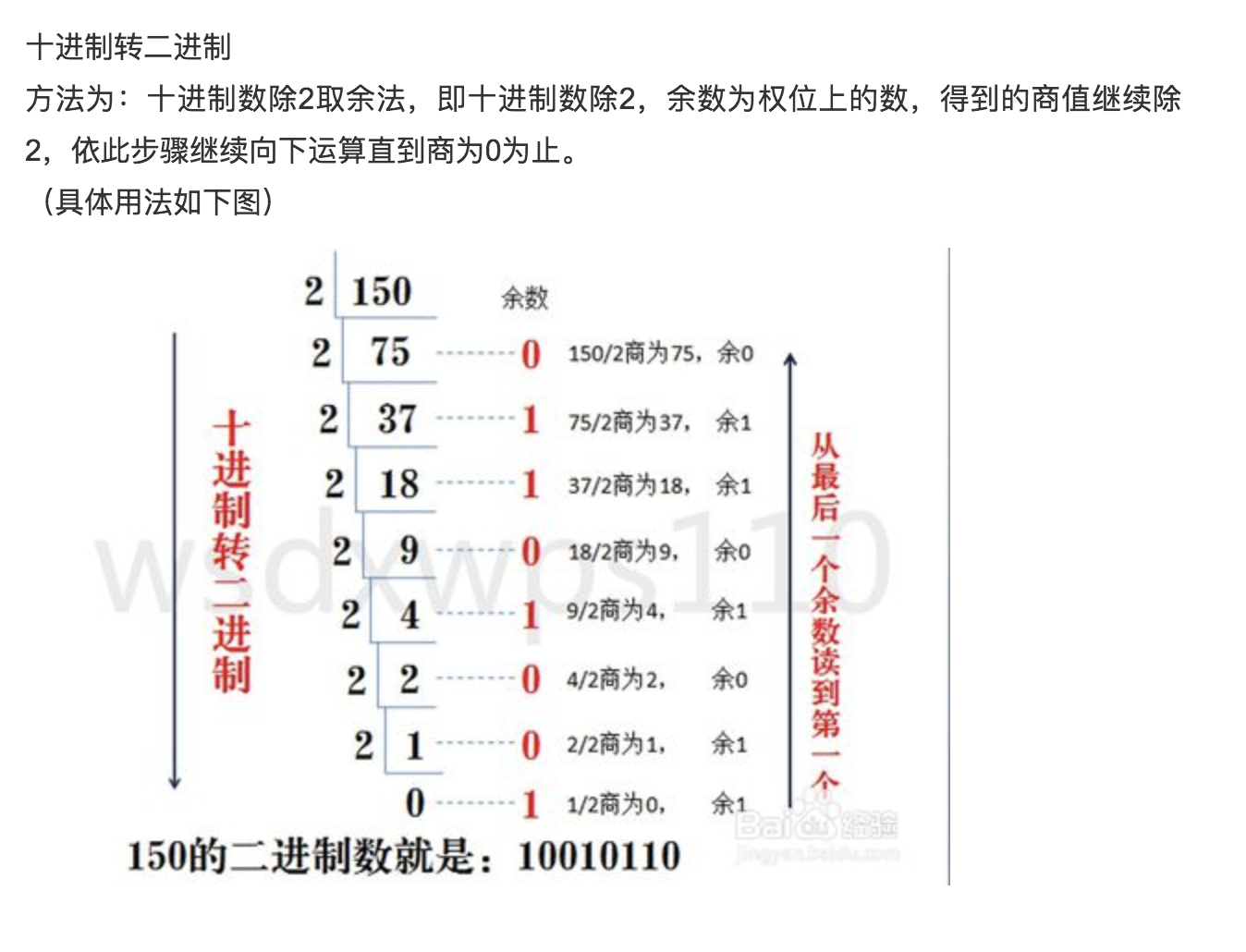
[复制代码](javascript:void(0);)

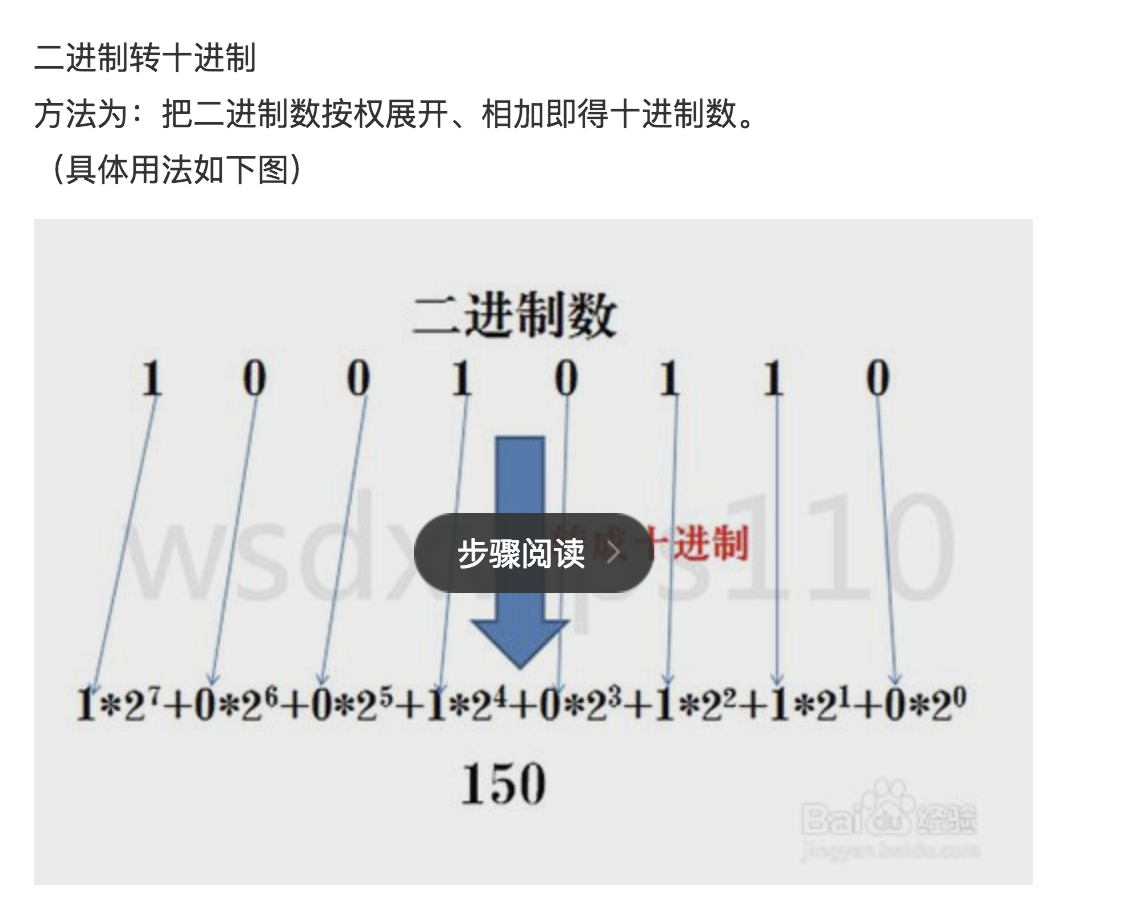
 　　通过以上例子，可以基本理解了Hibernate的三种状态，其实可以举个例子来说明：

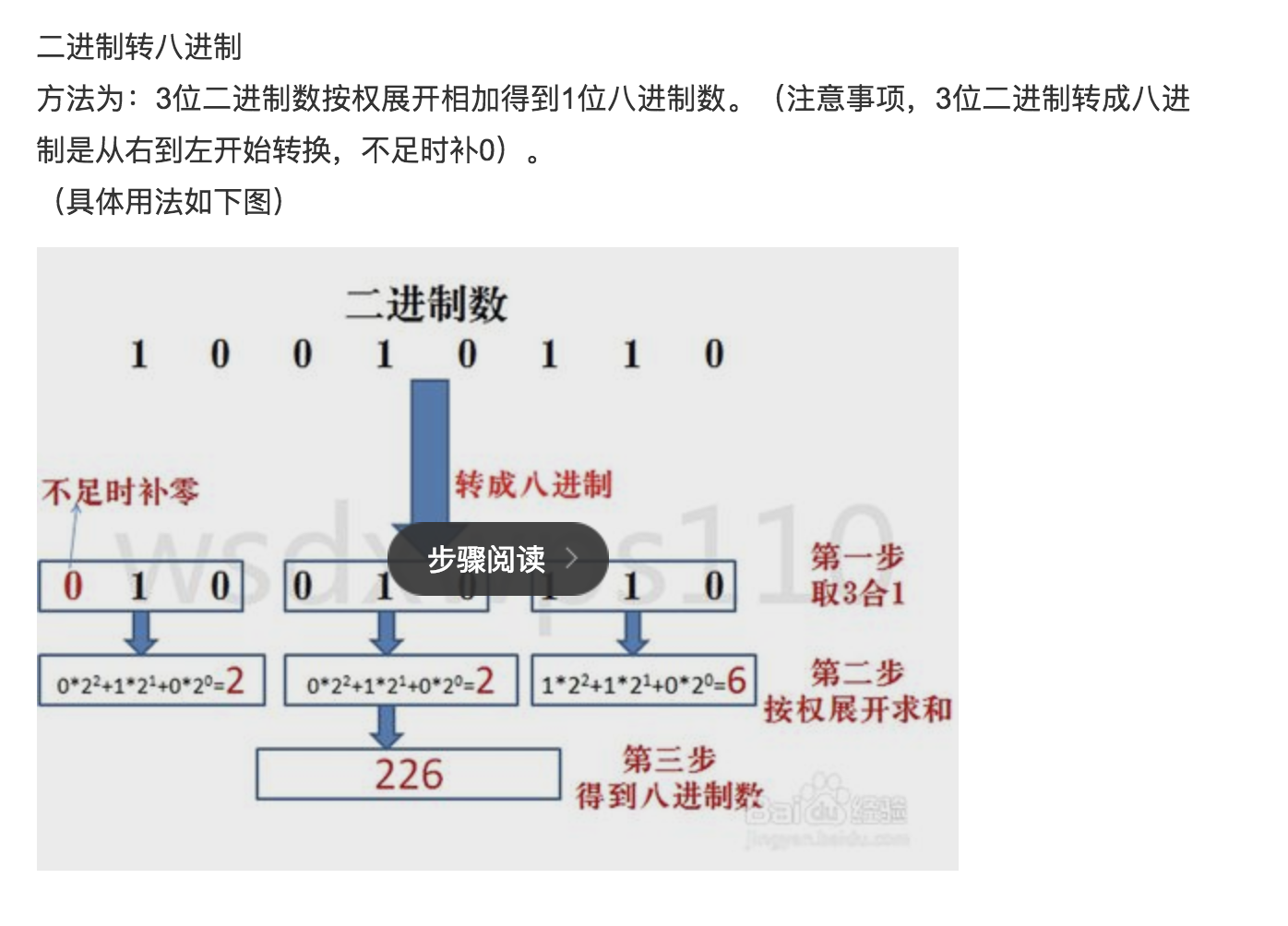
我们把数据库比喻成一个登记簿，当我们把信息登记到登记簿上的时候，这个过程就是持久化的过程。当登记完成之后，登记簿上的数据就是持久态数据，所谓持久态就是一直存在的状态。当我们需要用数据的时候，会从登记簿上查信息，我们查到的信息记录在临时的纸张或者脑海里，然后登记簿会放回原位，这时候临时纸张上或者脑海里的数据就是游离态，随时可能被遗忘，在hibernate中就是随时会被销毁的数据。瞬态数据和游离态数据有点儿类似，但是一点，就是他是存在持久化之前的数据，就是说我需要登记个人信息，我的个人信息还没有登记到登记簿上的时候，信息就是瞬态，一旦登记到登记簿上就变成持久态了，然后再查询到的就是游离态。

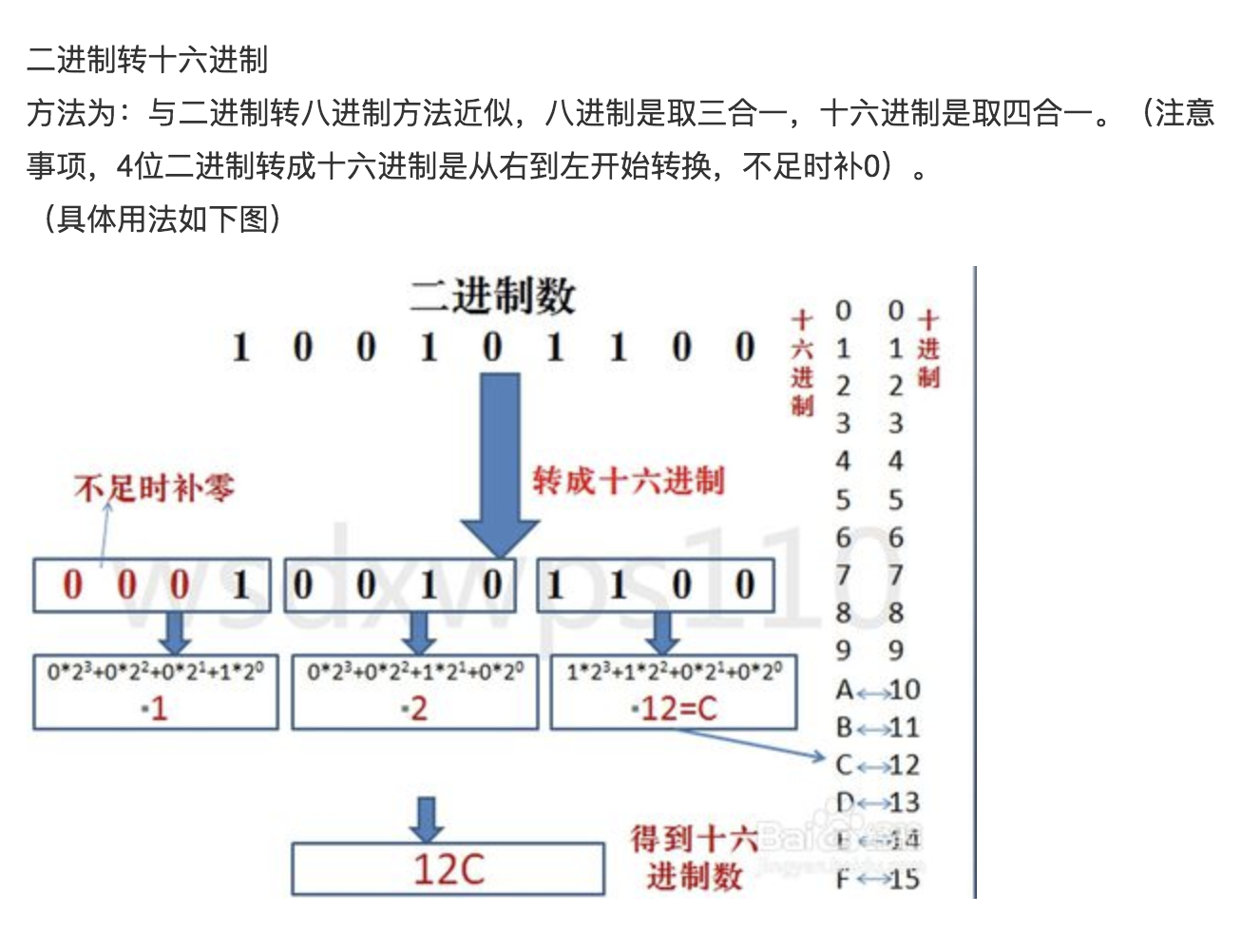
# 113、进制转化

## 前面不足就补0









# 113、hashmap扩容机制

#### 为什么要扩容，因为hashmap底层意识一个entry数组，数组的大小我们都知道是固定的啦，这里它的初始大小为16

## HashMap什么时候进行扩容呢？

当HashMap中的元素个数超过数组大小\*loadFactor时，就会进行数组扩容，loadFactor的默认值为0.75，这是一个折中的取值。也就是说，默认情况下，数组大小为16，那么当HashMap中元素个数超过16\*0.75=12的时候，就把数组的大小扩展为 2\*16=32，即扩大一倍，然后重新计算每个元素在数组中的位置，而这是一个非常消耗性能的操作，在HashMap数组扩容之后，最消耗性能的点就出现了：原数组中的数据必须重新计算其在新数组中的位置，并放进去，这就是resize。

所以如果我们已经预知HashMap中元素的个数，那么预设元素的个数能够有效的提高HashMap的性能。

## 为什么要设置负载因子

当负载因子较大时，去给table数组扩容的可能性就会少，所以相对占用内存较少（空间上较少），但是每条entry链上的元素会相对较多（hash运算，大哥并不是hashcode），查询的时间也会增长（时间上较多）。反之就是，负载因子较少的时候，给table数组扩容的可能性就高，那么内存空间占用就多，但是entry链上的元素就会相对较少，查出的时间也会减少。所以才有了负载因子是时间和空间上的一种折中的说法。所以设置负载因子的时候要考虑自己追求的是时间还是空间上的少。

   负载因子衡量的是一个散列表的空间的使用程度，负载因子越大表示散列表的装填程度越高，反之愈小。对于使用链表法的散列表来说，查找一个元素的平均时间是O(1+a)，因此如果负载因子越大，对空间的利用更充分，然而后果是查找效率的降低；如果负载因子太小，那么散列表的数据将过于稀疏，对空间造成严重浪费。

### 1、当添加数据的时候，size表示当前的数组的大小，如果超过了极限，那么久会扩大为原来的2倍，

threshold = (**int**)(capacity \* loadFactor);  16 \* 0.75

1. **void** addEntry(**int** hash, K key, V value, **int** bucketIndex) {   bucketIndex其实就是下表
2. // 获取指定 bucketIndex 索引处的 Entry
3. Entry<K,V> e = table[bucketIndex];

// 将新创建的 Entry 放入 bucketIndex 索引处，并让新的 Entry 指向原来的 Entry

//添加key到table[bucketIndex]位置，新的元素总是在table[bucketIndex]的第一个元素，原来的元素后移（妈的，这里的下表不是顺序产生的，而是根据hash值产生的）

1. table[bucketIndex] = **new** Entry<K,V>(hash, key, value, e);
2. // 如果 Map 中的 key-value 对的数量超过了极限
3. **if** (size++ >= threshold)
4. // 把 table 对象的长度扩充到原来的2倍。
5. resize(2 \* table.length);
6. }

# 114、ArrayList 扩容

## 1、arrayList底层是数组，长度默认为10

private static final int ***DEFAULT\_CAPACITY*** = 10;

## 2、这个扩容比较简单

在初始化ArrayList时进行赋值，那么开始是不会进行扩容的。

在新增到11个元素的时候，ArrayList内部的elementData数组变成了16个长度的数组。

17时变成25

26时变成38

39时变成58

>>位运算，右移动一位。 整体相当于newCapacity =oldCapacity + 0.5 \* oldCapacity

# 115、[Redis等缓存数据库为什么访问会比较快？](http://www.cnblogs.com/xiohao/p/6971854.html)

mysql是持久化存储，存放在磁盘里面，检索的话，会涉及到一定的IO，为了解决这个瓶颈，于是出现了缓存，比如现在用的最多的 memcached(简称mc)。  
首先，用户访问mc，如果未命中，就去访问mysql，之后像内存和硬盘一样，把数据复制到mc一部分redis和mc都是缓存，并且都是驻留在内存中运行的，这大大提升了高数据量web访问的访问速度。

然而mc只是提供了简单的数据结构，比如 string存储；redis却提供了大量的数据结构，比如string、list、set、hashset、sorted set这些，这使得用户方便了好多，毕竟封装了一层实用的功能，同时  
实现了同样的效果，当然用redis而慢慢舍弃mc。 　　内存和硬盘的关系，硬盘放置主体数据用于持久化存储，而内存则是当前运行的那部分数据，CPU访问内存而不是磁盘，这大大提升了运行的速度，

# 116、activemq比数据库好在哪里

1、削峰填谷：如果短时间内要处理的业务量大于数据库的服务能力，则可能会卡死数据库；使用MQ可以慢慢处理。

4、负载均衡：如果一台机器处理消息不够及时，可以多上几台机器做消息消费。

后来RPC和MQ出现了，现代化的分布式技术的两块基石就此建立起来。后端技术世界就有了如今金碧辉煌的大厦。

### 117、Map

key为对象的时候，一旦对象发送改变，就拿不到值了

public class MapMain {  
  
 public static void main(String[] args) {  
 Map <Object,Object> map = new HashMap<>();  
  
 Object o = new Object();  
 Object o2 = new Object();  
  
 map.put(o,"Stromg");  
 map.put(o2,3);  
  
  
 System.***out***.println(map.get(o));  
 System.***out***.println(map.get(o2));  
  
 o = 456; //对象一旦发送改变了，就拿不到值了  
 System.***out***.println(map.get(o));  
  
  
  
 Person person = new Person(); //这里new出来之后，对象没有发送改变  
 person.setAge(1);  
 map.put(person,3);  
 System.***out***.println(map.get(person));  
 person.setName("HealerJean");  
 System.***out***.println(map.get(person));  
  
  
 }  
}

Stromg

3

null

3

3

# 117、前后端分离的安全性问题

登录验证以及安全问题：   
1、请求接口全部用post方式，在后端判断请求方式是否为post

2、登录密码等敏感信息要加密后传输，如用RSA（支付宝里可下载公私钥生成工具），客户端公钥加密，传到服务器后再用私钥解密：

用户请求登录页面的时候，服务器生成一个token保存在服务器session中和客户端cookie中，登录时客户端带上此token一起提交到服务器，验证此token是否有效，若有效即登录成功并更新此token，重新赋值到session和cookie中；   
登录后的用户在每一次请求时都判断cookie中的token和session中的token是否一样，若一样即登录有效，并且要更新token，下次请求再进行判断，若不一样即登录无效。