# 1．Java的基础部分

## 1.1.java的跨平台原理

java代码编译后为.class文件，称为字节码文件，jvm就是将字节码文件翻译成特定平台下的机器码然后运行。不同平台有不同的jvm，在对应的平台安装对应的jvm就可以实现“一次编译到处运行”。

## 1.2.java面向对象的特征

*封装、抽象、继承、多态*

封装：就是将信息隐藏在一个类中，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问

抽象：就是将一类实体的共同特性抽取出来，封装在一个抽象类中，所以抽象在面向对象语言是由抽象类来体现的。

继承：就向我们现实生活中的父子关系，儿子可以遗传父亲的一些特征，面向对象中，就是一个类可以继承另一个类的一些特性，从而可以代码重用。

多态：就是指父类通过引用不同的子类对象表现出不同的行为。

## 1.3.多态

### 1.3.1.多态的好处

允许不同类对象对同一消息做出响应，即同一消息可以根据发送对象的不同而采用多种不同的行为方式(发送消息就是函数调用)。主要有以下优点：

1.可替换性：多态对已存在代码具有可替换性

2.可扩充性：增加新的子类不影响已经存在的类结构

3.接口性：多态是超类通过方法签名,向子类提供一个公共接口,由子类来完善或者重写它来实现的。

4.灵活性

5.简化性

### 1.3.2 代码中如何实现多态

实现多态主要有以下三种方式：  
1. 接口实现   
2. 继承父类重写方法   
3. 同一类中进行方法重载

### 1.3.3.虚拟机是如何实现多态的

动态绑定技术(dynamic binding)，执行期间判断所引用对象的实际类型，根据实际类型调用对应的方法。

## 1.4.接口和抽象类

### 1.4.1 接口的意义

接口的意义用三个词就可以概括：规范，扩展，回调。

### 1.4.2.接口的使用规则：

（1）定义一个接口，使用interface关键字

（2）在一个接口中，只能定义常量、抽象方法，JDK1.8后可以定义默认的实现方法

（3）接口可以继承多个接口：extends xxx,xxx

（4）一个具体类实现接口使用implements关键字

（5）一个类可以实现多个接口

（6）抽象类实现接口可以不实现接口的方法

（7）在接口中定义的方法没有声明 访问修饰符，默认为public

（8）接口不能有构造方法

（9）接口不能被实例化

### 1.4.3.抽象类的意义

抽象类的意义可以用三句话来概括：

1.为其他子类提供一个公共的类型

2.封装子类中重复定义的内容

3.定义抽象方法,子类虽然有不同的实现，但是定义是一致的

### 1.4.4. 抽象类的规则

1、抽象类可以没有抽象方法，有抽象方法的类必须是抽象类

2、非抽象类继承抽象类必须实现所有抽象方法

3、抽象类可以继承抽象类，可以不实现父类抽象方法。

4、抽象类可以有方法实现和属性

5、抽象类不能被实例化

6、抽象类不能声明为final

7、抽象类可以有构造方法

### 1.4.5.接口和抽象类的区别

| 比较 | 抽象类 | 接口 |
| --- | --- | --- |
| 默认方法 | 抽象类可以有默认的方法实现 | java 8之前,接口中不存在方法的实现. |
| 实现方式 | 子类使用extends关键字来继承抽象类.如果子类不是抽象类,子类需要提供抽象类中所声明方法的实现. | 子类使用implements来实现接口,需要提供接口中所有声明的实现. |
| 构造器 | 抽象类中可以有构造器, | 接口中不能 |
| 和正常类区别 | 抽象类不能被实例化 | 接口则是完全不同的类型 |
| 访问修饰符 | 抽象方法可以有public,protected和default等修饰 | 接口默认是public,不能使用其他修饰符 |
| 多继承 | 一个子类只能存在一个父类 | 一个子类可以存在多个接口 |
| 添加新方法 | 想抽象类中添加新方法,可以提供默认的实现,因此可以不修改子类现有的代码 | 如果往接口中添加新方法,则子类中需要实现该方法. |

## 1.5. 父类的静态方法能否被子类重写

不能。重写只适用于实例方法,不能用于静态方法，而子类当中含有和父类相同签名的静态方法，我们一般称之为隐藏。

## 1.6. 什么是不可变对象

不可变对象指对象一旦被创建，状态就不能再改变。任何修改都会创建一个新的对象，如 String、Integer及其它包装类。

## 1.7. 静态变量和实例变量的区别?

静态变量存储在方法区，属于类所有。实例变量存储在堆当中，其引用存在当前线程栈。

## 1.8. 能否创建一个包含可变对象的不可变对象?

当然可以创建一个包含可变对象的不可变对象的，你只需要谨慎一点，不要共享可变对象的引用就可以了，如果需要变化时，就返回原对象的一个拷贝。最常见的例子就是对象中包含一个日期对象的引用。

## 1.9. java 创建对象的几种方式

1、采用new

2、通过反射



3、采用clone(对象的克隆)



4、通过序列化机制（对象反序列化，从文件中读取内容，还原成对象）



前2者都需要显式地调用构造方法。造成耦合性最高的恰好是第一种，因此你发现无论什么框架，只要涉及到解耦必先减少new的使用。

## 1.10. switch中能否使用string做参数

在idk 1.7之前，switch只能支持byte, short, char, int或者其对应的封装类以及Enum类型。从idk 1.7之后switch开始支持String。

## 1.11. switch能否作用在byte, long上?

可以用在byte上，但是不能用在long上。

## 1.12. String s1=”ab”, String s2=”a”+”b”, String s3=”a”, String s4=”b”, s5=s3+s4请问s5==s2返回什么？

返回false。在编译过程中，编译器会将s2直接优化为”ab”，会将其放置在常量池当中，s5则是被创建在堆区，相当于s5=new String(“ab”);

## 1.13. 你对String对象的intern()熟悉么?

intern()方法会首先从常量池中查找是否存在该常量值，如果常量池中不存在则现在常量池中创建，如果已经存在则直接返回。  
比如   
String s1=”aa”;   
String s2=s1.intern();   
System.out.print(s1==s2);//返回true

## 1.14. Object中有哪些公共方法?

1.equals()

2.clone()(克隆)

3.getClass()（类的对象）

4.notify()（唤醒等待的线程）

notifyAll()（唤醒所有线程）

wait()（线程进入等待状态）

5.toString

## 1.15. java当中的四种引用

强引用，软引用，弱引用，虚引用。不同的引用类型主要体现在GC上:

1.强引用：如果一个对象具有强引用，它就不会被垃圾回收器回收。即使当前内存空间不足，JVM也不会回收它，而是抛出 OutOfMemoryError 错误，使程序异常终止。如果想中断强引用和某个对象之间的关联，可以显式地将引用赋值为null，这样一来的话，JVM在合适的时间就会回收该对象。

2.软引用：在使用软引用时，如果内存的空间足够，软引用就能继续被使用，而不会被垃圾回收器回收，只有在内存不足时，软引用才会被垃圾回收器回收。

3.弱引用：具有弱引用的对象拥有的生命周期更短暂。因为当 JVM 进行垃圾回收，一旦发现弱引用对象，无论当前内存空间是否充足，都会将弱引用回收。不过由于垃圾回收器是一个优先级较低的线程，所以并不一定能迅速发现弱引用对象。

4.虚引用：顾名思义，就是形同虚设，如果一个对象仅持有虚引用，那么它相当于没有引用，在任何时候都可能被垃圾回收器回收。

## 1.16. WeakReference与SoftReference的区别?

这点在四种引用类型中已经做了解释,这里简单说明一下即可:   
虽然 WeakReference 与 SoftReference 都有利于提高 GC 和 内存的效率，但是 WeakReference ，一旦失去最后一个强引用，就会被 GC 回收，而软引用虽然不能阻止被回收，但是可以延迟到 JVM 内存不足的时候。

## 1.17. 为什么要有不同的引用类型

不像C语言，我们可以控制内存的申请和释放，在Java中有时候我们需要适当的控制对象被回收的时机，因此就诞生了不同的引用类型，可以说不同的引用类型实则是对GC回收时机不可控的妥协。有以下几个使用场景可以充分的说明：

1、利用软引用和弱引用解决OOM问题：用一个HashMap来保存图片的路径和相应图片对象关联的软引用之间的映射关系，在内存不足时，JVM会自动回收这些缓存图片对象所占用的空间，从而有效地避免了OOM的问题.

2、通过软引用实现Java对象的高速缓存:比如我们创建了一Person的类，如果每次需要查询一个人的信息,哪怕是几秒中之前刚刚查询过的，都要重新构建一个实例，这将引起大量Person对象的消耗，并且由于这些对象的生命周期相对较短，会引起多次GC影响性能。此时，通过软引用和 HashMap 的结合可以构建高速缓存，提供性能。

## 1.18．java中==和eqauls()的区别

==是运算符，用于比较两个变量是否相等，而equals是Object类的方法，用于比较两个对象是否相等。默认Object类的equals方法是比较两个对象的地址，此时和==的结果一样。换句话说：基本类型比较用==，比较的是他们的值。默认下，对象用==比较时，比较的是内存地址，如果需要比较对象内容，需要重写equal方法。

## 1.19. equals()和hashcode()的联系

hashCode()是Object类的一个方法，返回一个哈希值。如果两个对象根据equal()方法比较相等，那么调用这两个对象中任意一个对象的hashCode()方法必须产生相同的哈希值。  
如果两个对象根据eqaul()方法比较不相等，那么产生的哈希值不一定相等(碰撞的情况下还是会相等的。)

## 1.20. a.hashCode()有什么用?与a.equals(b)有什么关系

hashCode() 方法是相应对象整型的 hash 值。它常用于基于 hash 的集合类，如 Hashtable、HashMap、LinkedHashMap等等。它与 equals() 方法关系特别紧密。根据 Java 规范，使用 equal() 方法来判断两个相等的对象，必须具有相同的 hashcode。

将对象放入到集合中时，首先判断要放入对象的hashcode是否已经在集合中存在，不存在则直接放入集合。如果hashcode相等，然后通过equal()方法判断要放入对象与集合中的任意对象是否相等：如果equal()判断不相等，直接将该元素放入集合中，否则不放入。

## 1.21. 有没有可能两个不相等的对象有相同的hashcode

有可能，两个不相等的对象可能会有相同的 hashcode 值，这就是为什么在 hashmap 中会有冲突。如果两个对象相等，必须有相同的hashcode 值，反之不成立。

## 1.22. 可以在hashcode中使用随机数字吗?

不行，因为同一对象的 hashcode 值必须是相同的

## 1.23. a==b与a.equals(b)有什么区别

如果a 和b 都是对象，则 a==b 是比较两个对象的引用，只有当 a 和 b 指向的是堆中的同一个对象才会返回 true，而 a.equals(b) 是进行逻辑比较，所以通常需要重写该方法来提供逻辑一致性的比较。例如，String 类重写 equals() 方法，所以可以用于两个不同对象，但是包含的字母相同的比较。

## 1.24. 3\*0.1==0.3返回值是什么

false，因为有些浮点数不能完全精确的表示出来。

## 1.25. a=a+b与a+=b有什么区别吗?

+=操作符会进行隐式自动类型转换，此处a+=b隐式的将加操作的结果类型强制转换为持有结果的类型，而a=a+b则不会自动进行类型转换。如：  
byte a = 127;   
byte b = 127;   
b = a + b; // error : cannot convert from int to byte   
b += a; // ok   
（译者注：这个地方应该表述的有误，其实无论 a+b 的值为多少，编译器都会报错，因为 a+b 操作会将 a、b 提升为 int 类型，所以将 int 类型赋值给 byte 就会编译出错）

## 1.26. short s1= 1; s1 = s1 + 1; 该段代码是否有错,有的话怎么改？

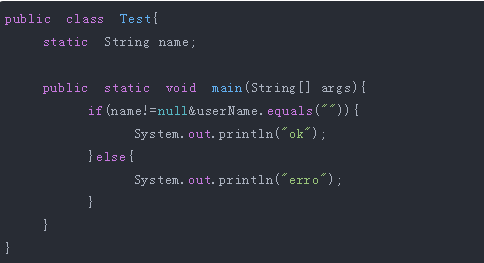
有错误，short类型在进行运算时会自动提升为int类型，也就是说s1+1的运算结果是int类型。

## 1.27. short s1= 1; s1 += 1; 该段代码是否有错，有的话怎么改？

+=操作符会自动对右边的表达式结果强转匹配左边的数据类型，所以没错。

## 1.28. & 和 &&的区别

首先记住&是位操作，而&&是逻辑运算符。另外需要记住逻辑运算符具有短路特性，而&不具备短路特性。



以上代码将会抛出空指针异常。

## 1.29. 一个java文件内部可以有类？(非内部类)

只能有一个public公共类，但是可以有多个default修饰的类。

## 1.30. 如何正确的退出多层嵌套循环？

1、使用标号和break;

2、通过在外层循环中添加标识符

## 1.31. 内部类的作用

内部类可以有多个实例，每个实例都有自己的状态信息，并且与其他外围对象的信息相互独立.在单个外围类当中，可以让多个内部类以不同的方式实现同一接口，或者继承同一个类.创建内部类对象的时刻不依赖于外部类对象的创建。内部类并没有令人疑惑的”is-a”管系，它就像是一个独立的实体。

内部类提供了更好的封装，除了该外围类，其他类都不能访问。

## 1.32. final, finalize和finally的不同之处

final 是一个修饰符，可以修饰变量、方法和类。如果 final 修饰变量，意味着该变量的值在初始化后不能被改变。finalize 方法是在对象被回收之前调用的方法，给对象自己最后一个复活的机会，但是什么时候调用 finalize 没有保证。finally 是一个关键字，与 try 和 catch 一起用于异常的处理。finally 块一定会被执行，无论在 try 块中是否有发生异常。

## 1.33. clone()是哪个类的方法?

java.lang.Cloneable 是一个标示性接口，不包含任何方法，clone 方法在 object 类中定义。并且需要知道 clone() 方法是一个本地方法，这意味着它是由 c 或 c++ 或 其他本地语言实现的。

## 1.34. 深拷贝和浅拷贝的区别是什么?

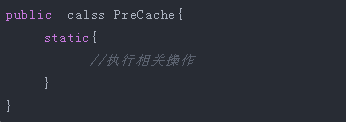
浅拷贝：被复制对象的所有变量都含有与原来的对象相同的值，而所有的对其他对象的引用仍然指向原来的对象。换言之，浅拷贝仅仅复制所考虑的对象，而不复制它所引用的对象。

深拷贝：被复制对象的所有变量都含有与原来的对象相同的值，而那些引用其他对象的变量将指向被复制过的新对象，而不再是原有的那些被引用的对象。换言之，深拷贝把要复制的对象所引用的对象都复制了一遍。

## 1.35. static都有哪些用法?

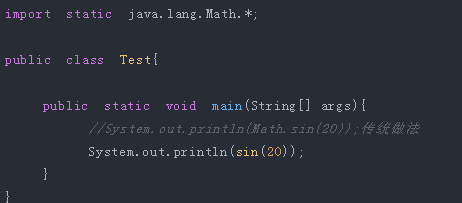
几乎所有的人都知道static关键字这两个基本的用法：静态变量和静态方法。也就是被static所修饰的变量/方法都属于类的静态资源，类实例所共享。

除了静态变量和静态方法之外，static也用于静态块，多用于初始化操作：



此外static也多用于修饰内部类，此时称之为静态内部类。

最后一种用法就是静态导包，即import static.import static是在JDK 1.5之后引入的新特性，可以用来指定导入某个类中的静态资源，并且不需要使用类名。资源名，可以直接使用资源名，比如：



## 1.36. final有哪些用法

final也是很多面试喜欢问的地方，能回答下以下三点就不错了：  
1.被final修饰的类不可以被继承   
2.被final修饰的方法不可以被重写   
3.被final修饰的变量不可以被改变。如果修饰引用，那么表示引用不可变，引用指向的内容可变。  
4.被final修饰的方法，JVM会尝试将其内联，以提高运行效率   
5.被final修饰的常量，在编译阶段会存入常量池中。

回答出编译器对final域要遵守的两个重排序规则更好：  
1.在构造函数内对一个final域的写入，与随后把这个被构造对象的引用赋值给一个引用变量,这两个操作之间不能重排序。  
2.初次读一个包含final域的对象的引用，与随后初次读这个final域,这两个操作之间不能重排序。

## 1.37. 数据类型

### 1.37.1. 64位的JVM当中,int的长度是多少?

Java 中，int 类型变量的长度是一个固定值，与平台无关，都是 32 位。意思就是说，在 32 位 和 64 位 的Java 虚拟机中，int 类型的长度是相同的。

### 1.37.2. int和Integer的区别

Integer是int的包装类型，在拆箱和装箱中，二者自动转换。int是基本类型，直接存数值，而integer是对象，用一个引用指向这个对象。

### 1.37.3. int 和Integer谁占用的内存更多?

Integer 对象会占用更多的内存。Integer是一个对象，需要存储对象的元数据。但是 int 是一个原始类型的数据，所以占用的空间更少。

### 1.37.4. String, StringBuffer和StringBuilder区别

String是字符串常量，final修饰：StringBuffer字符串变量(线程安全)；  
StringBuilder 字符串变量(线程不安全)。

### 1.37.5. String和StringBuffer

String和StringBuffer主要区别是性能：String是不可变对象，每次对String类型进行操作都等同于产生了一个新的String对象，然后指向新的String对象。所以尽量不在对String进行大量的拼接操作，否则会产生很多临时对象，导致GC开始工作，影响系统性能。

StringBuffer是对对象本身操作，而不是产生新的对象，因此在有大量拼接的情况下，我们建议使用StringBuffer。

但是需要注意现在JVM会对String拼接做一定的优化：  
String s=“This is only ”+”simple”+”test”会被虚拟机直接优化成String s=“This is only simple test”，此时就不存在拼接过程。

### 1.37.6. StringBuffer和StringBuilder

StringBuffer是线程安全的可变字符串，其内部实现是可变数组。StringBuilder是jdk 1.5新增的，其功能和StringBuffer类似，但是非线程安全。因此，在没有多线程问题的前提下，使用StringBuilder会取得更好的性能。

### 1.37.7. 什么是编译器常量？使用它有什么风险？

公共静态不可变（public static final ）变量也就是我们所说的编译期常量，这里的 public 可选的。实际上这些变量在编译时会被替换掉，因为编译器知道这些变量的值，并且知道这些变量在运行时不能改变。这种方式存在的一个问题是你使用了一个内部的或第三方库中的公有编译时常量，但是这个值后面被其他人改变了，但是你的客户端仍然在使用老的值，甚至你已经部署了一个新的jar。为了避免这种情况，当你在更新依赖 JAR 文件时，确保重新编译你的程序。

### 1.37.8. java当中使用什么类型表示价格比较好?

如果不是特别关心内存和性能的话，使用BigDecimal，否则使用预定义精度的 double 类型。

### 1.37.9. 如何将byte转为String

可以使用 String 接收 byte[] 参数的构造器来进行转换，需要注意的点是要使用的正确的编码，否则会使用平台默认编码，这个编码可能跟原来的编码相同，也可能不同。



### 1.37.10. 可以将int强转为byte类型么?会产生什么问题?

我们可以做强制转换，但是Java中int是32位的而byte是8 位的，所以,如果强制转化int类型的高24位将会被丢弃，byte 类型的范围是从-128到128

### 1.38. 垃圾回收

### 1.38.1. 你知道哪些垃圾回收算法?

垃圾回收从理论上非常容易理解,具体的方法有以下几种:   
1. 标记-清除   
2. 标记-复制   
3. 标记-整理   
4. 分代回收   
更详细的内容参见深入理解垃圾回收算法：

<http://blog.csdn.net/dd864140130/article/details/50084471>

### 1.38.2. 如何判断一个对象是否应该被回收

这就是所谓的对象存活性判断，常用的方法有两种：1.引用计数法;　2.对象可达性分析。由于引用计数法存在互相引用导致无法进行GC的问题，所以目前JVM虚拟机多使用对象可达性分析算法。

### 1.38.3. 简单的解释一下垃圾回收

Java 垃圾回收机制最基本的做法是分代回收。内存中的区域被划分成不同的世代，对象根据其存活的时间被保存在对应世代的区域中。一般的实现是划分成3个世代：年轻、年老和永久。内存的分配是发生在年轻世代中的。当一个对象存活时间足够长的时候，它就会被复制到年老世代中。对于不同的世代可以使用不同的垃圾回收算法。进行世代划分的出发点是对应用中对象存活时间进行研究之后得出的统计规律。一般来说，一个应用中的大部分对象的存活时间都很短。比如局部变量的存活时间就只在方法的执行过程中。基于这一点，对于年轻世代的垃圾回收算法就可以很有针对性。

### 1.38.4. 调用System.gc()会发生什么?

通知GC开始工作，但是GC真正开始的时间不确定。

## 1.39. 进程,线程

### 1.39.1. 说说进程，线程，协程之间的区别

简而言之，进程是程序运行和资源分配的基本单位，一个程序至少有一个进程，一个进程至少有一个线程。进程在执行过程中拥有独立的内存单元，而多个线程共享内存资源，减少切换次数，从而效率更高。线程是进程的一个实体，是cpu调度和分派的基本单位，是比程序更小的能独立运行的基本单位。同一进程中的多个线程之间可以并发执行。

### 1.39.2. 你了解守护线程吗？它和非守护线程有什么区别

程序运行完毕，jvm会等待非守护线程完成后关闭，但是jvm不会等待守护线程。守护线程最典型的例子就是GC线程。

### 1.39.3. 什么是多线程上下文切换

多线程的上下文切换是指CPU控制权由一个已经正在运行的线程切换到另外一个就绪并等待获取CPU执行权的线程的过程。

### 1.39.4. 创建两种线程的方式?他们有什么区别?

通过实现java.lang.Runnable或者通过扩展java.lang.Thread类。相比扩展Thread，实现Runnable接口可能更优.原因有二：

1、Java不支持多继承。因此扩展Thread类就代表这个子类不能扩展其他类。而实现Runnable接口的类还可能扩展另一个类。

2、类可能只要求可执行即可，因此继承整个Thread类的开销过大。

### 1.39.5. Thread类中的start()和run()方法有什么区别?

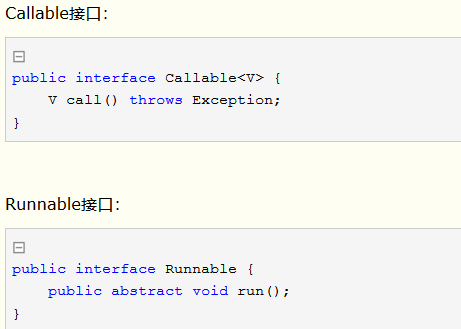
start()方法被用来启动新创建的线程，而且start()内部调用了run()方法，这和直接调用run()方法的效果不一样。当你调用run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。

### 1.39.6. 怎么检测一个线程是否持有对象监视器

Thread类提供了一个holdsLock(Object obj)方法，当且仅当对象obj的监视器被某条线程持有的时候才会返回true，注意这是一个static方法，这意味着”某条线程”指的是当前线程。

### 1.39.7. Runnable和Callable的区别

Runnable接口中的run()方法的返回值是void，它做的事情只是纯粹地去执行run()方法中的代码而已；Callable接口中的call()方法是有返回值的，是一个泛型，和Future、FutureTask配合可以用来获取异步执行的结果。   
这其实是很有用的一个特性，因为多线程相比单线程更难、更复杂的一个重要原因就是因为多线程充满着未知性，某条线程是否执行了？某条线程执行了多久？某条线程执行的时候我们期望的数据是否已经赋值完毕？无法得知，我们能做的只是等待这条多线程的任务执行完毕而已。而Callable+Future/FutureTask却可以方便获取多线程运行的结果，可以在等待时间太长没获取到需要的数据的情况下取消该线程的任务。





### 1.39.8. 什么导致线程阻塞

阻塞指的是暂停一个线程的执行以等待某个条件发生（如某资源就绪），学过操作系统的同学对它一定已经很熟悉了。Java 提供了大量方法来支持阻塞，下面让我们逐一分析。

| 方法 | 说明 |
| --- | --- |
| sleep() | sleep() 允许 指定以毫秒为单位的一段时间作为参数，它使得线程在指定的时间内进入阻塞状态，不能得到CPU 时间，指定的时间一过，线程重新进入可执行状态。 典型地，sleep() 被用在等待某个资源就绪的情形：测试发现条件不满足后，让线程阻塞一段时间后重新测试，直到条件满足为止 |
| suspend() 和 resume() | 两个方法配套使用，suspend()使得线程进入阻塞状态，并且不会自动恢复，必须其对应的resume() 被调用，才能使得线程重新进入可执行状态。典型地，suspend() 和 resume() 被用在等待另一个线程产生的结果的情形：测试发现结果还没有产生后，让线程阻塞，另一个线程产生了结果后，调用 resume() 使其恢复。 |
| yield() | yield() 使当前线程放弃当前已经分得的CPU 时间，但不使当前线程阻塞，即线程仍处于可执行状态，随时可能再次分得 CPU 时间。调用 yield() 的效果等价于调度程序认为该线程已执行了足够的时间从而转到另一个线程 |
| wait() 和 notify() | 两个方法配套使用，wait() 使得线程进入阻塞状态，它有两种形式，一种允许 指定以毫秒为单位的一段时间作为参数，另一种没有参数，前者当对应的 notify() 被调用或者超出指定时间时线程重新进入可执行状态，后者则必须对应的 notify() 被调用。 |

### 1.39.9. wait(),notify()和suspend(),resume()之间的区别

初看起来它们与 suspend() 和 resume() 方法对没有什么分别，但是事实上它们是截然不同的。区别的核心在于，前面叙述的所有方法，阻塞时都不会释放占用的锁（如果占用了的话），而这一对方法则相反。上述的核心区别导致了一系列的细节上的区别。

首先，前面叙述的所有方法都隶属于 Thread 类，但是这一对却直接隶属于 Object 类，也就是说，所有对象都拥有这一对方法。初看起来这十分不可思议，但是实际上却是很自然的，因为这一对方法阻塞时要释放占用的锁，而锁是任何对象都具有的，调用任意对象的 wait() 方法导致线程阻塞，并且该对象上的锁被释放。而调用 任意对象的notify()方法则导致从调用该对象的 wait() 方法而阻塞的线程中随机选择的一个解除阻塞（但要等到获得锁后才真正可执行）。

其次，前面叙述的所有方法都可在任何位置调用，但是这一对方法却必须在 synchronized 方法或块中调用，理由也很简单，只有在synchronized 方法或块中当前线程才占有锁，才有锁可以释放。同样的道理，调用这一对方法的对象上的锁必须为当前线程所拥有，这样才有锁可以释放。因此，这一对方法调用必须放置在这样的 synchronized 方法或块中，该方法或块的上锁对象就是调用这一对方法的对象。若不满足这一条件，则程序虽然仍能编译，但在运行时会出现IllegalMonitorStateException 异常。

wait() 和 notify() 方法的上述特性决定了它们经常和synchronized关键字一起使用，将它们和操作系统进程间通信机制作一个比较就会发现它们的相似性：synchronized方法或块提供了类似于操作系统原语的功能，它们的执行不会受到多线程机制的干扰，而这一对方法则相当于 block 和wakeup 原语（这一对方法均声明为 synchronized）。它们的结合使得我们可以实现操作系统上一系列精妙的进程间通信的算法（如信号量算法），并用于解决各种复杂的线程间通信问题。

关于 wait() 和 notify() 方法最后再说明两点：   
第一：调用 notify() 方法导致解除阻塞的线程是从因调用该对象的 wait() 方法而阻塞的线程中随机选取的，我们无法预料哪一个线程将会被选择，所以编程时要特别小心，避免因这种不确定性而产生问题。

第二：除了 notify()，还有一个方法 notifyAll() 也可起到类似作用，唯一的区别在于，调用 notifyAll() 方法将把因调用该对象的 wait() 方法而阻塞的所有线程一次性全部解除阻塞。当然，只有获得锁的那一个线程才能进入可执行状态。

谈到阻塞，就不能不谈一谈死锁，略一分析就能发现，suspend() 方法和不指定超时期限的 wait() 方法的调用都可能产生死锁。遗憾的是，Java 并不在语言级别上支持死锁的避免，我们在编程中必须小心地避免死锁。

以上我们对 Java 中实现线程阻塞的各种方法作了一番分析，我们重点分析了 wait() 和 notify() 方法，因为它们的功能最强大，使用也最灵活，但是这也导致了它们的效率较低，较容易出错。实际使用中我们应该灵活使用各种方法，以便更好地达到我们的目的。

### 1.39.10. 产生死锁的条件

1.互斥条件：一个资源每次只能被一个进程使用。   
2.请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。   
3.不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。   
4.循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

### 1.39.11. 为什么wait()方法和notify()/notifyAll()方法要在同步块中被调用

这是JDK强制的，wait()方法和notify()/notifyAll()方法在调用前都必须先获得对象的锁

### 1.39.12. wait()方法和notify()/notifyAll()方法在放弃对象监视器时有什么区别

wait()方法和notify()/notifyAll()方法在放弃对象监视器的时候的区别在于：wait()方法立即释放对象监视器，notify()/notifyAll()方法则会等待线程剩余代码执行完毕才会放弃对象监视器。

### 1.39.13. wait()与sleep()的区别

关于这两者已经在上面进行详细的说明,这里就做个概括好了:

* sleep()来自Thread类，和wait()来自Object类。调用sleep()方法的过程中，线程不会释放对象锁。而 调用 wait 方法线程会释放对象锁
* sleep()睡眠后不出让系统资源，wait让其他线程可以占用CPU
* sleep(milliseconds)需要指定一个睡眠时间，时间一到会自动唤醒.而wait()需要配合notify()或者notifyAll()使用

### 1.39.14. 为什么wait, nofity和nofityAll这些方法不放在Thread类当中

一个很明显的原因是JAVA提供的锁是对象级的而不是线程级的，每个对象都有锁，通过线程获得。如果线程需要等待某些锁那么调用对象中的wait()方法就有意义了。如果wait()方法定义在Thread类中，线程正在等待的是哪个锁就不明显了。简单的说，由于wait，notify和notifyAll都是锁级别的操作，所以把他们定义在Object类中因为锁属于对象。

### 1.39.15. 怎么唤醒一个阻塞的线程

如果线程是因为调用了wait()、sleep()或者join()方法而导致的阻塞，可以中断线程，并且通过抛出InterruptedException来唤醒它；如果线程遇到了IO阻塞，无能为力，因为IO是操作系统实现的，Java代码并没有办法直接接触到操作系统。

### 1.39.16. 什么是多线程的上下文切换

多线程的上下文切换是指CPU控制权由一个已经正在运行的线程切换到另外一个就绪并等待获取CPU执行权的线程的过程。

### 1.39.17. 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   
（3）ReentrantLock可以灵活地实现多路通知   
另外，二者的锁机制其实也是不一样的:ReentrantLock底层调用的是Unsafe的park方法加锁，synchronized操作的应该是对象头中mark word。

### 1.39.18. FutureTask是什么

这个其实前面有提到过，FutureTask表示一个异步运算的任务。FutureTask里面可以传入一个Callable的具体实现类，可以对这个异步运算的任务的结果进行等待获取、判断是否已经完成、取消任务等操作。当然，由于FutureTask也是Runnable接口的实现类，所以FutureTask也可以放入线程池中。

### 1.39.19. 一个线程如果出现了运行时异常怎么办?

如果这个异常没有被捕获的话，这个线程就停止执行了。另外重要的一点是：如果这个线程持有某个某个对象的监视器，那么这个对象监视器会被立即释放。

### 1.39.20. Java当中有哪几种锁

1、自旋锁: 自旋锁在JDK1.6之后就默认开启了。基于之前的观察，共享数据的锁定状态只会持续很短的时间，为了这一小段时间而去挂起和恢复线程有点浪费，所以这里就做了一个处理，让后面请求锁的那个线程在稍等一会，但是不放弃处理器的执行时间，看看持有锁的线程能否快速释放。为了让线程等待，所以需要让线程执行一个忙循环也就是自旋操作。在jdk6之后，引入了自适应的自旋锁，也就是等待的时间不再固定了，而是由上一次在同一个锁上的自旋时间及锁的拥有者状态来决定。

2、偏向锁: 在JDK1.之后引入的一项锁优化，目的是消除数据在无竞争情况下的同步原语。进一步提升程序的运行性能。 偏向锁就是偏心的偏，意思是这个锁会偏向第一个获得他的线程，如果接下来的执行过程中，改锁没有被其他线程获取，则持有偏向锁的线程将永远不需要再进行同步。偏向锁可以提高带有同步但无竞争的程序性能，也就是说他并不一定总是对程序运行有利，如果程序中大多数的锁都是被多个不同的线程访问，那偏向模式就是多余的，在具体问题具体分析的前提下，可以考虑是否使用偏向锁。

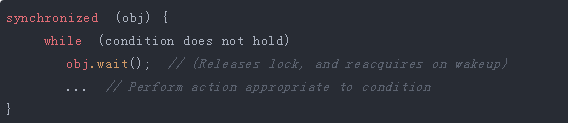
3、轻量级锁: 为了减少获得锁和释放锁所带来的性能消耗，引入了“偏向锁”和“轻量级锁”，所以在Java SE1.6里锁一共有四种状态，无锁状态，偏向锁状态，轻量级锁状态和重量级锁状态，它会随着竞争情况逐渐升级。锁可以升级但不能降级，意味着偏向锁升级成轻量级锁后不能降级成偏向锁。

### 1.39.21. 如何在两个线程间共享数据

通过在线程之间共享对象就可以了，然后通过wait/notify/notifyAll、await/signal/signalAll进行唤起和等待，比方说阻塞队列BlockingQueue就是为线程之间共享数据而设计的。

### 1.39.22. 如何正确的使用wait()？使用if还是while？

wait() 方法应该在循环调用，因为当线程获取到 CPU 开始执行的时候，其他条件可能还没有满足，所以在处理前，循环检测条件是否满足会更好。下面是一段标准的使用 wait 和 notify 方法的代码：



### 1.39.23. 什么是线程局部变量ThreadLocal

线程局部变量是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享。Java提供ThreadLocal类来支持线程局部变量，是一种实现线程安全的方式。但是在管理环境下（如 web 服务器）使用线程局部变量的时候要特别小心，在这种情况下，工作线程的生命周期比任何应用变量的生命周期都要长。任何线程局部变量一旦在工作完成后没有释放，Java 应用就存在内存泄露的风险。

### 1.39.24. ThreadLoal的作用是什么?

简单说ThreadLocal就是一种以空间换时间的做法在每个Thread里面维护了一个ThreadLocal.ThreadLocalMap把数据进行隔离，数据不共享，自然就没有线程安全方面的问题了。

### 1.39.25. 生产者消费者模型的作用是什么?

（1）通过平衡生产者的生产能力和消费者的消费能力来提升整个系统的运行效率，这是生产者消费者模型最重要的作用。  
（2）解耦，这是生产者消费者模型附带的作用，解耦意味着生产者和消费者之间的联系少，联系越少越可以独自发展而不需要收到相互的制约。

### 1.39.26. 写一个生产者-消费者队列

见：java基础

### 1.39.27. 如果你提交任务时，线程池队列已满，这时会发生什么

如果你使用的LinkedBlockingQueue，也就是无界队列的话，没关系，继续添加任务到阻塞队列中等待执行，因为LinkedBlockingQueue可以近乎认为是一个无穷大的队列，可以无限存放任务；如果你使用的是有界队列比方说ArrayBlockingQueue的话，任务首先会被添加到ArrayBlockingQueue中，ArrayBlockingQueue满了，则会使用拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy。

### 1.39.28. 为什么要使用线程池

避免频繁地创建和销毁线程，达到线程对象的重用。另外，使用线程池还可以根据项目灵活地控制并发的数目。

### 1.39.29. java中用到的线程调度算法是什么

抢占式。一个线程用完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。

### 1.39.30. Thread.sleep(0)的作用是什么

由于Java采用抢占式的线程调度算法，因此可能会出现某条线程常常获取到CPU控制权的情况，为了让某些优先级比较低的线程也能获取到CPU控制权，可以使用Thread.sleep(0)手动触发一次操作系统分配时间片的操作，这也是平衡CPU控制权的一种操作。

### 1.39.31. 什么是CAS

CAS，全称为Compare and Swap，即比较-替换。假设有三个操作数：内存值V、旧的预期值A、要修改的值B，当且仅当预期值A和内存值V相同时，才会将内存值修改为B并返回true，否则什么都不做并返回false。当然CAS一定要volatile变量配合，这样才能保证每次拿到的变量是主内存中最新的那个值，否则旧的预期值A对某条线程来说，永远是一个不会变的值A，只要某次CAS操作失败，永远都不可能成功。

### 1.39.32. 什么是乐观锁和悲观锁

乐观锁：乐观锁认为竞争不总是会发生，因此它不需要持有锁，将比较-替换这两个动作作为一个原子操作尝试去修改内存中的变量，如果失败则表示发生冲突，那么就应该有相应的重试逻辑。

悲观锁：悲观锁认为竞争总是会发生，因此每次对某资源进行操作时，都会持有一个独占的锁，就像synchronized，不管三七二十一，直接上了锁就操作资源了。

### 1.39.33. ConcurrentHashMap的并发度是什么?

ConcurrentHashMap的并发度就是segment的大小，默认为16，这意味着最多同时可以有16条线程操作ConcurrentHashMap，这也是ConcurrentHashMap对Hashtable的最大优势，任何情况下，Hashtable能同时有两条线程获取Hashtable中的数据吗？

### 1.39.34. ConcurrentHashMap的工作原理

ConcurrentHashMap在jdk 1.7和jdk 1.8实现原理是不同的。

jdk 1.7:

ConcurrentHashMap是线程安全的，但是与Hashtablea相比，实现线程安全的方式不同。Hashtable是通过对hash表结构进行锁定，是阻塞式的，当一个线程占有这个锁时，其他线程必须阻塞等待其释放锁。ConcurrentHashMap是采用分离锁的方式，它并没有对整个hash表进行锁定，而是局部锁定，也就是说当一个线程占有这个局部锁时，不影响其他线程对hash表其他地方的访问。   
具体实现:ConcurrentHashMap内部有一个Segment.

jdk 1.8

在jdk 8中，ConcurrentHashMap不再使用Segment分离锁，而是采用一种乐观锁CAS算法来实现同步问题，但其底层还是“数组+链表->红黑树”的实现。

### 1.39.35. CyclicBarrier和CountDownLatch区别

这两个类非常类似，都在java.util.concurrent下，都可以用来表示代码运行到某个点上，二者的区别在于：

1、CyclicBarrier的某个线程运行到某个点上之后，该线程即停止运行，直到所有的线程都到达了这个点，所有线程才重新运行；CountDownLatch则不是，某线程运行到某个点上之后，只是给某个数值-1而已，该线程继续运行。

2、CyclicBarrier只能唤起一个任务，CountDownLatch可以唤起多个任务

3、CyclicBarrier可重用，CountDownLatch不可重用，计数值为0该CountDownLatch就不可再用了。

### 1.39.36. java中的++操作符线程安全么?

不是线程安全的操作。它涉及到多个指令，如读取变量值，增加，然后存储回内存，这个过程可能会出现多个线程交差。

### 1.39.37. 你有哪些多线程开发良好的实践?

1、给线程命名

2、最小化同步范围

3、优先使用volatile

4、尽可能使用更高层次的并发工具而非wait和notify()来实现线程通信,如BlockingQueue,Semeaphore

5、优先使用并发容器而非同步容器.

6、考虑使用线程池

## 1.40. volatile关键字

### 1.40.1. 可以创建Volatile数组吗?

Java 中可以创建 volatile类型数组，不过只是一个指向数组的引用，而不是整个数组。如果改变引用指向的数组，将会受到volatile 的保护，但是如果多个线程同时改变数组的元素，volatile标示符就不能起到之前的保护作用了。

### 1.40.2. volatile能使得一个非原子操作变成原子操作吗?

一个典型的例子是在类中有一个 long 类型的成员变量。如果你知道该成员变量会被多个线程访问，如计数器、价格等，你最好是将其设置为 volatile。为什么？因为 Java 中读取 long 类型变量不是原子的，需要分成两步，如果一个线程正在修改该 long 变量的值，另一个线程可能只能看到该值的一半（前 32 位）。但是对一个 volatile 型的 long 或 double 变量的读写是原子。

一种实践是用 volatile 修饰 long 和 double 变量，使其能按原子类型来读写。double 和 long 都是64位宽，因此对这两种类型的读是分为两部分的，第一次读取第一个 32 位，然后再读剩下的 32 位，这个过程不是原子的，但 Java 中 volatile 型的 long 或 double 变量的读写是原子的。volatile 修复符的另一个作用是提供内存屏障（memory barrier），例如在分布式框架中的应用。简单的说，就是当你写一个 volatile 变量之前，Java 内存模型会插入一个写屏障（write barrier），读一个 volatile 变量之前，会插入一个读屏障（read barrier）。意思就是说，在你写一个 volatile 域时，能保证任何线程都能看到你写的值，同时，在写之前，也能保证任何数值的更新对所有线程是可见的，因为内存屏障会将其他所有写的值更新到缓存。

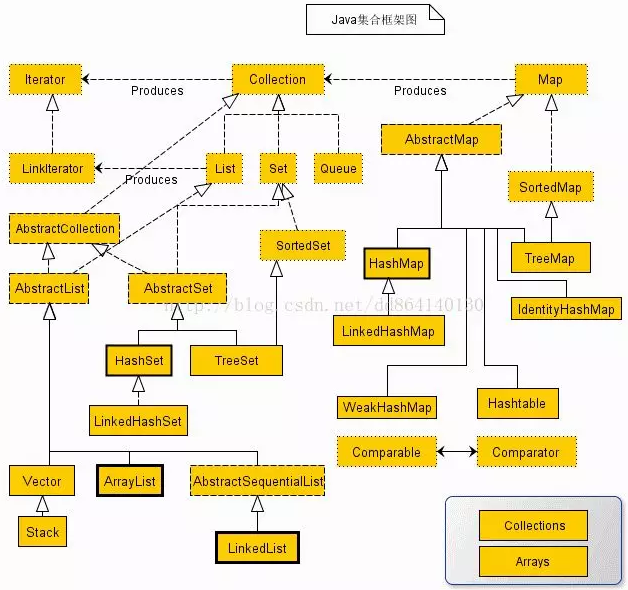
### 1.40.3. volatile类型变量提供什么保证?

volatile 主要有两方面的作用:1.避免指令重排2.可见性保证.例如，JVM 或者 JIT为了获得更好的性能会对语句重排序，但是 volatile 类型变量即使在没有同步块的情况下赋值也不会与其他语句重排序。 volatile 提供 happens-before 的保证，确保一个线程的修改能对其他线程是可见的。某些情况下，volatile 还能提供原子性，如读 64 位数据类型，像 long 和 double 都不是原子的(低32位和高32位)，但 volatile 类型的 double 和 long 就是原子的。

## 1.41. 集合

### 1.41.1. Java中的集合及其继承关系

关于集合的体系是每个人都应该烂熟于心的,尤其是对我们经常使用的List,Map的原理更该如此.这里我们看这张图即可:



更多内容可见集合类总结：

http://write.blog.csdn.net/postedit/40826423

### 1.41.2. poll()方法和remove()方法区别？

poll() 和 remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

### 1.41.3. LinkedHashMap和PriorityQueue的区别

PriorityQueue 是一个优先级队列,保证最高或者最低优先级的的元素总是在队列头部，但是 LinkedHashMap 维持的顺序是元素插入的顺序。当遍历一个 PriorityQueue 时，没有任何顺序保证，但是 LinkedHashMap 课保证遍历顺序是元素插入的顺序。

### 1.41.4. WeakHashMap与HashMap的区别是什么?

WeakHashMap 的工作与正常的 HashMap 类似，但是使用弱引用作为 key，意思就是当 key 对象没有任何引用时，key/value 将会被回收。

### 1.41.5．ArrayList和LinkedList的区别?

最明显的区别是 ArrrayList底层的数据结构是数组，支持随机访问，而 LinkedList 的底层数据结构是双向循环链表，不支持随机访问。使用下标访问一个元素，ArrayList 的时间复杂度是 O(1)，而 LinkedList 是 O(n)。

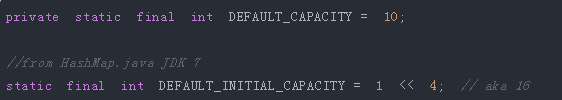
### 1.41.6. ArrayList和Array有什么区别?

1、Array可以容纳基本类型和对象，而ArrayList只能容纳对象。

2、Array是指定大小的，而ArrayList大小是固定的

### 1.41.7. ArrayList和HashMap默认大小?

在 Java 7 中，ArrayList 的默认大小是 10 个元素，HashMap 的默认大小是16个元素（必须是2的幂）。这就是 Java 7 中 ArrayList 和 HashMap 类的代码片段。



### 1.41.8. Comparator和Comparable的区别?

Comparable 接口用于定义对象的自然顺序，而 comparator 通常用于定义用户定制的顺序。Comparable 总是只有一个，但是可以有多个 comparator 来定义对象的顺序。

### 1.41.9. 如何实现集合排序?

你可以使用有序集合，如 TreeSet 或 TreeMap，你也可以使用有顺序的的集合，如 list，然后通过 Collections.sort() 来排序。

### 1.41.10. 如何打印数组内容

你可以使用 Arrays.toString() 和 Arrays.deepToString() 方法来打印数组。由于数组没有实现 toString() 方法，所以如果将数组传递给 System.out.println() 方法，将无法打印出数组的内容，但是 Arrays.toString() 可以打印每个元素。

### 1.41.11. LinkedList的是单向链表还是双向?

双向循环列表，具体实现自行查阅源码。

### 1.41.12. TreeMap是实现原理

采用红黑树实现，具体实现自行查阅源码。

### 1.41.13. 遍历ArrayList时如何正确移除一个元素

该问题的关键在于面试者使用的是 ArrayList 的 remove() 还是 Iterator 的 remove()方法。这有一段示例代码，是使用正确的方式来实现在遍历的过程中移除元素，而不会出现 ConcurrentModificationException 异常的示例代码。

### 1.41.14. 什么是ArrayMap?它和HashMap有什么区别?

ArrayMap是Android SDK中提供的，非Android开发者可以略过。  
ArrayMap是用两个数组来模拟map，更少的内存占用空间,更高的效率。  
具体参考这篇文章：ArrayMap VS HashMap：http://lvable.com/?p=217%5D

### 1.41.15. HashMap的实现原理

1. HashMap概述： HashMap是基于哈希表的Map接口的非同步实现。此实现提供所有可选的映射操作，并允许使用null值和null键。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。   
2. HashMap的数据结构： 在java编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，HashMap也不例外。HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。

当我们往Hashmap中put元素时,首先根据key的hashcode重新计算hash值,根绝hash值得到这个元素在数组中的位置(下标),如果该数组在该位置上已经存放了其他元素,那么在这个位置上的元素将以链表的形式存放,新加入的放在链头,最先加入的放入链尾.如果数组中该位置没有元素,就直接将该元素放到数组的该位置上.

需要注意Jdk 1.8中对HashMap的实现做了优化,当链表中的节点数据超过八个之后,该链表会转为红黑树来提高查询效率,从原来的O(n)到O(logn)

### 1.41.16. 你了解Fail-Fast机制吗？

Fail-Fast即我们常说的快速失败，

更多内容参看fail-fast机制：http://blog.csdn.net/chenssy/article/details/38151189

### 1.41.17. Fail-fast和Fail-safe有什么区别

Iterator的fail-fast属性与当前的集合共同起作用，因此它不会受到集合中任何改动的影响。Java.util包中的所有集合类都被设计为fail->fast的，而java.util.concurrent中的集合类都为fail-safe的。当检测到正在遍历的集合的结构被改变时，Fail-fast迭代器抛出ConcurrentModificationException，而fail-safe迭代器从不抛出ConcurrentModificationException。

## 1.42. 日期

### 1.42.1. SimpleDateFormat是线程安全的吗?

非常不幸，DateFormat 的所有实现，包括 SimpleDateFormat 都不是线程安全的，因此你不应该在多线程序中使用，除非是在对外线程安全的环境中使用，如 将 SimpleDateFormat 限制在 ThreadLocal 中。如果你不这么做，在解析或者格式化日期的时候，可能会获取到一个不正确的结果。因此，从日期、时间处理的所有实践来说，我强力推荐 joda-time 库。

### 1.42.2. 如何格式化日期?

Java 中，可以使用 SimpleDateFormat 类或者 joda-time 库来格式日期。DateFormat 类允许你使用多种流行的格式来格式化日期。参见答案中的示例代码，代码中演示了将日期格式化成不同的格式，如 dd-MM-yyyy 或 ddMMyyyy。

## 1.43. 异常

### 1.43.1. 简单描述java异常体系

相比没有人不了解异常体系,关于异常体系的更多信息可以见白话异常机制：

http://blog.csdn.net/dd864140130/article/details/42504189

### 1.43.2. 什么是异常链

详情直接参见上面的白话异常机制，不做解释了。

### 1.43.3. throw和throws的区别

throw用于主动抛出java.lang.Throwable 类的一个实例化对象，意思是说你可以通过关键字 throw 抛出一个 Error 或者 一个Exception，如：throw new IllegalArgumentException(“size must be multiple of 2″),   
而throws 的作用是作为方法声明和签名的一部分，方法被抛出相应的异常以便调用者能处理。Java 中，任何未处理的受检查异常强制在 throws 子句中声明。

## 1.44. 序列化

### 1.44.1. Java 中，Serializable 与 Externalizable 的区别

Serializable 接口是一个序列化 Java 类的接口，以便于它们可以在网络上传输或者可以将它们的状态保存在磁盘上，是 JVM 内嵌的默认序列化方式，成本高、脆弱而且不安全。Externalizable 允许你控制整个序列化过程，指定特定的二进制格式，增加安全机制。

## 1.45. JVM

### 1.45.1. JVM特性

平台无关性.   
Java语言的一个非常重要的特点就是与平台的无关性。而使用Java虚拟机是实现这一特点的关键。一般的高级语言如果要在不同的平台上运行，至少需要编译成不同的目标代码。而引入Java语言虚拟机后，Java语言在不同平台上运行时不需要重新编译。Java语言使用模式Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言编译程序只需生成在Java虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。

### 1.45.2. 简单解释一下类加载器

有关类加载器一般会问你四种类加载器的应用场景以及双亲委派模型,

更多的内容参看深入理解JVM加载器：

http://blog.csdn.net/dd864140130/article/details/49817357

### 1.45.3. 简述堆和栈的区别

VM 中堆和栈属于不同的内存区域，使用目的也不同。栈常用于保存方法帧和局部变量，而对象总是在堆上分配。栈通常都比堆小，也不会在多个线程之间共享，而堆被整个 JVM 的所有线程共享。

### 1.45.4. 简述JVM内存分配

1、基本数据类型比变量和对象的引用都是在栈分配的。

1. 堆内存用来存放由new创建的对象和数组。

3、类变量（static修饰的变量），程序在一加载的时候就在堆中为类变量分配内存，堆中的内存地址存放在栈中。

4、实例变量：当你使用java关键字new的时候，系统在堆中开辟并不一定是连续的空间分配给变量，是根据零散的堆内存地址，通过哈希算法换算为一长串数字以表征这个变量在堆中的”物理位置”,实例变量的生命周期–当实例变量的引用丢失后，将被GC（垃圾回收器）列入可回收“名单”中，但并不是马上就释放堆中内存。

5、局部变量: 由声明在某方法，或某代码段里（比如for循环），执行到它的时候在栈中开辟内存，当局部变量一但脱离作用域，内存立即释放。

## 1.46. 其他

### 1.46.1. XML解析的几种方式和特点

DOM, SAX, PULL三种解析方式：

* DOM:消耗内存：先把xml文档都读到内存中，然后再用DOM API来访问树形结构，并获取数据。这个写起来很简单，但是很消耗内存。要是数据过大，手机不够牛逼，可能手机直接死机
* SAX:解析效率高，占用内存少，基于事件驱动的：更加简单地说就是对文档进行顺序扫描，当扫描到文档(document)开始与结束、元素(element)开始与结束、文档(document)结束等地方时通知事件处理函数，由事件处理函数做相应动作，然后继续同样的扫描，直至文档结束。
* PULL:与 SAX 类似，也是基于事件驱动，我们可以调用它的next（）方法，来获取下一个解析事件（就是开始文档，结束文档，开始标签，结束标签），当处于某个元素时可以调用XmlPullParser的getAttributte()方法来获取属性的值，也可调用它的nextText()获取本节点的值。

### 1.46.2. JDK 1.7特性

然 JDK 1.7 不像 JDK 5 和 8 一样的大版本，但是，还是有很多新的特性，如 try-with-resource 语句，这样你在使用流或者资源的时候，就不需要手动关闭，Java 会自动关闭。Fork-Join 池某种程度上实现 Java 版的 Map-reduce。允许 Switch 中有 String 变量和文本。菱形操作符(<>)用于类型推断，不再需要在变量声明的右边申明泛型，因此可以写出可读写更强、更简洁的代码。

### 1.46.3. JDK 1.8特性

java 8 在 Java 历史上是一个开创新的版本，下面 JDK 8 中 5 个主要的特性：   
Lambda 表达式，允许像对象一样传递匿名函数   
Stream API，充分利用现代多核 CPU，可以写出很简洁的代码   
Date 与 Time API，最终，有一个稳定、简单的日期和时间库可供你使用   
扩展方法，现在，接口中可以有静态、默认方法。   
重复注解，现在你可以将相同的注解在同一类型上使用多次。

### 1.46.4. Maven和ANT有什么区别?

虽然两者都是构建工具，都用于创建 Java 应用，但是 Maven 做的事情更多，在基于“约定优于配置”的概念下，提供标准的Java 项目结构，同时能为应用自动管理依赖（应用中所依赖的 JAR 文件。

### 1.46.5. JDBC最佳实践

1、优先使用批量操作来插入和更新数据

1. 使用PreparedStatement来避免SQL漏洞

3、使用数据连接池

4、通过列名来获取结果集

### 1.46.6. IO操作最佳实践

1、使用有缓冲的IO类,不要单独读取字节或字符

2、使用NIO和NIO 2或者AIO,而非BIO

1. 在finally中关闭流

4、使用内存映射文件获取更快的IO

## 1.47. 讲一下什么是设计模式？常用的设计模式有哪些

设计模式就是经过前人无数次的实践总结出的，设计过程中可以反复使用的、可以解决特定问题的设计方法。

单例(饱汉模式、饥汉模式)

|  |
| --- |
| //饿汉式  class Singleton {  private Singleton(){};  private static Singleton s = new Singleton();  public static Singleton getInstance() {  return s;  }  public void print() {  System.out.println("测试方法");  }  } |

|  |
| --- |
| //懒汉式  class Singleton {  private Singleton(){};  private static Singleton s;  public static Singleton getInstance() {  if(s == null) {  s = new Singleton();  }  return s;  }  public void print() {  System.out.println("测试方法");  }  } |

1. 构造方法私有化（除了自己类中能创建外其他地方都不能创建）

2、在自己的类中创建一个单实例（饱汉模式是一出来就创建单实例，而饥汉模式需要的时候才创建）

3、提供一个方法获取该实例对象(创建时需要进行方法同步)

工厂模式:Spring IOC就是使用了工厂模式.

对象的创建交给一个工厂去创建。

代理模式:Spring AOP就是使用的动态代理。

# 2.java web部分

## 2.1. http get和post请求的区别

a、get是用来从服务器上获取数据，而post是用来向服务器传递数据；

b、get将表单中数据按照variable=value的形式，添加到action所指向的URL后面，并且两者用"？"连接，变量之间用"&"连接；

而post是将表单中的数据放在form的数据体中，按照变量与值对应的方式，传递到action所指定的URL。

c、get是不安全的，因为在传输过程中，数据是被放在请求的URL中;而post的所有操作对用户来说都是不可见的。

d、get传输的数据量小，这主要应为受url长度限制;而post可以传输大量的数据，所有上传文件只能用post提交。

e、get限制form表单的数据集必须为ASCII字符；而post支持整个IS01 0646字符集。

f、get是form表单的默认方法。

## 2.2. 说一下你对servlet的理解？或者servlet是什么？

**S**ervlet（Server Applet），全称Java Servlet， 是用Java编写的服务器端程序。而这些Sevlet都要实现Servlet这个借口。其主要功能在于交互式地浏览和修改数据，生成动态Web内容。Servlet运行于支持Java的应用服务器中。

HttpServlet 重写doGet和doPost方法或者你也可以重写service方法完成对get和post请求的响应

## 2.3. 简单说一下servlet的生命周期，并说出Servlet和CGI的区别？

servlet有良好的生存期的定义，包括加载和实例化、初始化、处理请求以及服务结束。这个生存期由javax.servlet.Servlet接口的init，service和destroy方法表达。

加载Servlet的class---->实例化Servlet----->调用Servlet的init完成初始化---->响应请求（Servlet的service方法）----->Servlet容器关闭时(Servlet的destory方法)

Servlet启动时，开始加载servlet生命周期开始。Servlet被服务器实例化后，容器运行其init方法，请求到达时运行其service方法，service方法自动派遣运行与请求对应的doXXX方法（doGet，doPost）等，当服务器决定将实例销毁的时候(服务器关闭)调用其destroy方法。

区别：

Servlet处于服务器进程中，它通过多线程运行service()方法，一个实例可以服务于多个请求，并且实例一般不会被销毁；而

CGI对每个请求都产生一个新的进程，服务完成后就销毁，所有效率低于Servlet。

## 2.4. Servlet的基本架构



## 2.5. Servlet API中forward() 与redirect()的区别

1. forward是服务器端的转向而redirect是客户端的跳转。
2. 使用forward浏览器的地址不会发生改变。而redirect会发生改变。
3. Forward是一次请求中完成。而redirect是重新发起请求。
4. Forward是在服务器端完成，而不用客户端重新发起请求，效率较高。

## 2.6. JSP和Servlet有哪些相同点和不同点

JSP是Servlet的扩展，本质上是Servlet的简易方式，更强调应用的外表表达。JSP编译后是"类Servlet"。Servlet和JSP最主要的不同点在于，Servlet的应用逻辑是在java文件中，并且完全从表示层中的HTML里分离开来。而JSP的情况是java和HTML可以组合成一个扩展名为.jsp的文件。JSP偏重于视图，Servlet偏重于业务逻辑。

## 2.7. jsp有哪些内置对象?作用分别是什么?

9个内置的对象：

request负责得到客户端请求的信息，对应类型：javax.servlet.http.HttpServletRequest

response负责向客户端发出响应，对应类型：javax.servlet.http.HttpServletResponse

pageContext 表示这个JSP页面上下文，对应类型：javax.servlet.jsp.pagecontext

session负责保存同一客户端一次会话过程中的一些信息，对应类型：javax.servlet.http.httpsession

application表示整个应用环境的信息，对应类型：javax.servlet.servletcontext

out负责管理对客户端的输出，对应类型：javax.serlvet.jsp.jspwriter

config表示ServletConfig（servlet的构架部件），对应类型：javax.servlet.servletconfig

page 表示当前JSP页面本身

exception 表示页面中发生的异常，可以通过它获得页面异常信息，对应类型：java.lang.exception

四大作用域：

pageContext ：就是设置的属性只能在当前页面有效。通过pageContext的setAttribute()和getAttribute()。

request ：指属性在一次请求范围内有效。如果页面从给一个页面跳转到另一个页面，那么该属性就失效了。这里所指的跳转是指客户端跳转，比如客户单击超链接跳转到其他页面或者通过浏览器地址栏浏览其他页面。如果使用服务器端跳转<jsp:forward>,则该属性仍然生效。同理使用request对象的setAttribute()和getAttribute()。

session ：指客户浏览器与服务器一次会话范围内，如果服务器断开连接，那么属性就失效了。同理通过session对象的setAttribute()和getAttribute()。session范围的例外比如重新打开一个浏览器。

application ：指在整个服务器范围，知道服务器停止以后才会失效。同理通过application对象的setAttribute()和getAttribute()。application范围就是保存的属性只要服务器不重启，就能在任意页面中获取，就算重新打开浏览器也是可以获取属性的。

可以通过jstl（JSP标准标签库）从四大作用域中取值.

Jsp传递值request session application cookie也能传值

## 2.8. 说一下session和cookie的区别？你在项目中都有哪些地方使用了？

Session和cookie都是会话(Seesion)跟踪技术。Cookie通过在客户端记录信息确定用户身份，Session通过在服务器端记录信息确定用户身份。但是Session的实现依赖于Cookie,sessionId(session的唯一标识需要存放在客户端).

cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗，考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能,考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议： 将登陆信息等重要信息存放为SESSION，其他信息如果需要保留，可以放在COOKIE中，比如购物车

购物车最好使用cookie，但是cookie是可以在客户端禁用的，这时候我们要使用cookie+数据库的方式实现，当从cookie中不能取出数据时，就从数据库获取。

## 2.9. 什么情况下调用doGet()和doPost()?

默认情况是调用doGet()方法，JSP页面中的Form表单的method属性设置为post的时候，调用的为doPost()方法；为get的时候，调用deGet()方法。

## 2.10. request.getParameter()和request.getAttribute()的区别？

a、request.getParameter()获取的类型是String；

  request.getAttribute()获取的类型是Object

b、request.getPrameter()获取的是POST/GET传递的参数值和URL中的参数；

  request.getAttribute()获取的是对象容器中的数据值/对象

c、request.setAttribute()和request.getAttribute()可以发送、接收对象；

 request.getParamter()只能接收字符串，官方不开放request.setParamter()（也就是没有这个方法）

  setAttribute()和getAttribute()的传参原理：

  setAttribute()是应用服务器把这个对象放在该页面所对应的一块内存中去，当你的页面服务器重定向到另外一个页面时，

  应用服务器会把这块内存拷贝到另一个页面所对应的那块内存中。这个就可以通过getAttribute()获取到相应的参数值或者对象。

## 2.11. JSP有哪些动作？作用分别是什么？

JSP一共有以下6中基本动作

  JSP：include   （当页面被请求的时候引入一个文件）

  JSP：forward   （将请求转到另一个页面）

  JSP：useBean   （获得JavaBean的一个实例）

  JSP：setProperty（设置JavaBean的属性）

  JSP：getProperty（获得JavaBean的属性）

  JSP：plugin   （根据浏览器类型为Java插件生成object或者embed两种标记）

## 2.12. JSP常用指令有哪些？

page、include、taglib

a、page指令：定义页面的一些属性，常用属性：

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

pageEncoding="utf-8";           JSP编译成java文件时所用的编码

session="true"   是否自动创建session

b、include指令：引入一个静态的JSP页面

c、taglib指令：引入一个标签库

## 2.13. JSP中动态include和静态include的区别？

a、静态include：语法：<%@ include file="文件名" %>，相当于复制，编辑时将对应的文件包含进来，当内容变化时，不会再一次对其编译，不易维护。

b、动态include：语法：<jsp:include page="文件名">,能够自动检查被包含文件，当客户端对JSP文件进行请求时，会重新将对应的文件包含进来，进行实时的更新。

## 2.14. 页面间对象传递的方法？

request、session、application、cookie等。

## 2.15. 详细描述MVC。

基于java的web应用系统采用MVC设计模型，即用Model（模型）、View（视图）和Controller（控制）分离设计，这是目前web应用服务系统的主流设置方向。

Model：处理业务逻辑的模块。

View：负责页面显示，显示Model的处理结果给用户，主要实现数据到页面的转换过程。

Controller：负责每个请求的分发，把Form数据传递给Model进行处理，处理完成后，把处理结果返回给相应的View显示给用户。

## 2.16. MVC的各个部分都有那些技术来实现

M(Model) 代表应用的业务逻辑通过JavaBean、EJB组件实现

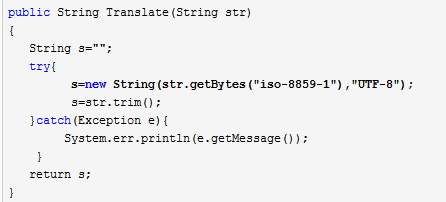
V(View) 视图由JSP产生

C(Control) 控制器 一般是一个Servelt

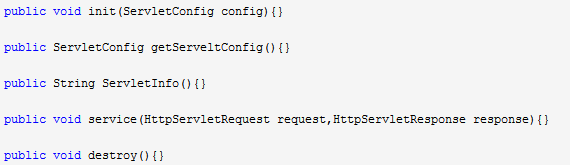
通过这种设计模型把应用逻辑、处理过程和显示逻辑分成不同的组件来实现。这些组件可以交互和重用。

## 2.17. 我们在web应用开发过程中经常遇到输出某种编码的字符，如iso-8859-1等，如何输出一个某种编码的字符串？

下面这个例子是将iso-8859-1转换成utf-8编码方式。



## 2.18. Servlet执行时一般实现哪几个方法？



## 2.19．javascript常用的方面？

数据校验、页面特殊效果等。

## 2.20. 常用的web容器和开发工具？

容器：Tomcat、Weblogic

工具：Eclipse、Jbuilder

## 2.21. JSP的四种范围？

a、page是代表一个页面相关的对象和属性。一个页面由一个编译好的java servlet类（可以带有include指令，但不可以带有include动作）表示。这既包括servlet又包括编译成servlet的jsp页面。

b、request是代表与web客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个web组件（由于forware指令和include动作的关系）

c、session是代表与用于某个web客户机的一个用户体验相关的对象和属性。一个web回话也可以经常跨域多个客户机请求。

d、application是代表与整个web应用程序相关的对象和属性。这实质上是跨域整个web应用程序，包括多个页面、请求和回话的一个全局作用域。

## 2.22. 应用服务器有哪些？

BEA Weblogic Server、

IBM WebSphere Application Server、

Oracle9i Application Server、

JBoss、

Tomcat。

## 2.23. 几种回话跟踪技术？

cookie、url重写、hidden表单隐藏域、session

## 2.24. 简述HttpSession的作用、使用方法，可用代码说明。

HttpSession中可以跟踪并存储用户信息，把值设置到属性中，有两个方法：setAttribute("变量/对象","变量/对象");getAttribute("变量/对象");例如：在一个方法中用session.setAttribute("username",username);在session中设置一个属性名为username的变量或者对象，然后可以在同一个session范围内使用session.getAttribute("username"); 来得到username属性或者对象。

## 2.25. J2EE是什么？

J2EE现在叫做Java EE：是（Java platform,Enterprice Edition 企业版）的简称。企业版本开发和部署服务器应用程序。

J2EE是Sun公司提出的多层（multi-tiered）、分布式（distributed）、基于组件（component-base）的企业级应用模型（Enterprice Application Model）。

在这样一个应用系统中，可按照功能划分不同的组件，这些组件又可在不同的计算机上，并且处理不同的层次（tier）中。所属层次包括客户层（client tier）、web层和组件、business层和组件、企业信息系统（EIS）层。

## 2.26. J2EE是技术、平台还是框架？

J2EE本身是一个标准，一个为企业分布式应用的开发提供的标准平台。

J2EE也是一个框架，包括JDBC、JNDI、RMI、JMS、EJB、JTA等技术。

## 2.27. JSTL是什么？优点有哪些？

JSTL（JSP Standard Tag Library，JSP标准标签库）是一个不断完善的开放源代码的JSP标签库，由四个定制标记库（core、format、xml、sql）和一对通用标记库验证器（ScriptFreeTLV和PermittedTaglibsTLV）组成。优点有：

a、在应用程序服务器之间提供了一致的接口，最大程度的提高了web应用在各应用服务器之间的移植。

b、简化了JSP和web应用程序的开发。

c、以一种统一的方式减少了JSP中scriptlet代码数据，可以达到没有任何scriptlet代码的代码。在我们公司的项目中是不允许任何scriptlet出现在JSP中。

d、允许JSP设计工具与web应用程序开发的进一步集成。相信不久就会有支持JSTL的IDE开发工具出现。

## 2.28. J2EE的优越性主要表现在那些方面？

a、J2EE基于Java技术，与平台无关。

b、J2EE提供了一个优秀的企业级应用程序框架，对快速高质量的开发系统打下了基础。

c、J2EE提供相当专业的通用软件服务。

d、J2EE拥有开放标准，许多大型公司实现了该规范支持的应用服务器。如：BEA、IBM、Oracle等。

## 2.29. JSP页面是如何被执行的？JSP执行效率比Servlet低吗？

当客户端向一个JSP页面发出请求时，Web Container将JSP转换成Servlet的源代码（只是第一次请求时），然后编译转化后的Servlet并加载到内存中执行，执行的结果Response到客户端。

JSP只在第一次执行的时候会转化为Servlet，以后的每次执行Web容器都是直接执行编译后的Servlet，所有JSP和Servlet只有第一次执行的时候不一样，JSP慢一点，以后的执行都是相同的。

## 2.30. JSP如何处理运行时异常（run-time）exception？

可是使用页面的errorPaga属性捕捉没有处理的运行时异常，然后自动转向到一个错误处理页面，代码如下：

<%@page errorPage="错误页面URL"%>

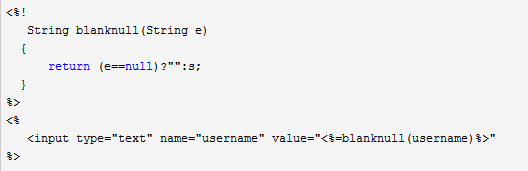
如果在页面请求时出现运行时异常时，以上代码会将代码转向到错误页面，在错误页面里面，可以通过以下代码定义这个页面是错误处理页面：

<%@page isErrorPage="true"%>

这样描述错误信息的Throwable对象就可以在错误页面里面访问到。

## 2.31. 如果JSP表单元素的值为空，如何避免null出现在页面上？

可以写一个简单的函数对空值进行处理，判断值是否为空，如果为空就返回空字符串。实例代码如下：



## 2.32. 如何避免JSP页面自动生成session对象？为什么要这么做？

在默认情况下，在对一个JSP页面发出请求时，如果session还没有建立，那么JSP页面会自动为请求创建一个session对象，但是session是比较消耗资源的，如果没有必要保持和使用session，就不应该创建session，例如一些只用来宣传产品的网页，往往没必要使用session来保存信息，可以在JSP中使用page指令进行设置，避免JSP页面为每个请求都自动创建session。实例代码如下：



## 2.33. 在servlet和JSP之间能共享session对象吗？

Yes



## 2.34. 使用JSP连接数据库连接缓冲池的最好的方法是什么？

a、使用JDBC2.0中带有此服务的Driver

b、使用提供此服务的application server

## 2.35. 在JSP中如何写文本文件？

使用PrintWriter对象：



## 2.36. JSP的缺点？

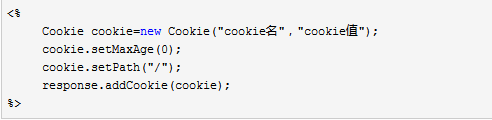
a、大多数的Servlet引擎不支持connection pooling。

b、Servlet引擎没有标准。

c、JSP与其他脚本语言的交互。

d、不易对Java程序进行调试。

## 2.37. 在JSP页面中如何删除一个cookie？



## 2.38. 如何实现servlet的单线程模式？

在JSP中使用page指令进行设置，具体代码如下：



## 2.39. HttpServlet类中主要方法有哪些？



## 2.40. XML文档定义有几种形式？他们之间有何本质区别？解析XML文档有哪几种方式？

a、两种形式:DTD、schema

b、本质区别：schema本身是xml的，可以被XML解析器解析（这也是从dtd上发展schema的根本目的。）

c、有DOM、SAX、STAX等。

DOM：处理大型文件时性能下降的非常厉害。这个问题是由DOM的树结构所造成的，这种结构占用的内存较多，而且DOM必须在解析文件之前就把整个文档装入内存，适合对XML的随机访问；

SAX：是事件驱动型的XML解析方式。它顺序读取XML文件，不需要一次全部装载这个XML文件。当遇到向文件开头、文件结束或者标签开头、标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理XML文件，适合对XML的顺序访问。

STAX：Streaming API for XML

## 2.41. 你在项目中用到了XML技术的哪些方面？如何实现的？

用到了数据存储、信息配置两方面。

在做数据交换平台时，将不能数据源的数据组装成XML文件，然后将XML文件压缩打包加密后通过网络传送给接受者，接收解密与解压缩后再同XML文件中还原相关信息进行处理。

在做软件配置时，利用XML可以和方便的进行，软件的各种配置参数都存储在XML文件中。

## 2.42. EL表达式的功能，为什么要用EL表达式？（Expression Language）

功能：

a、从四个域对象中取出数据数据显示。

b、取出请求参数数据显示。

原因：

在页面中用jsp脚本和jsp表达式来获取数据显示比较麻烦

a、需要判断

b、可能需要强转

## 2.43. JSTL的功能，为什么要用JSTL？（JavaServer Pages Standard Tag Library）

功能：主要用于基本输入输出、流程控制、循环、XML文件剖析、数据库查询及国际化和文字格式标准化的应用等。

原因：在jsp页面做条件判断或者循环操作并输出时，比较费力。

## 2.44. 为什么要使用自定义标签？MyTag如何实现？

原因：

a、不想在jsp中编写java代码

b、JSTL标签库不能满足实际项目的需求

实现的流程：

a、编写标签处理器类（SimpleTagSupport的实现类）

     重写doTag()

b、编写标签库文件（WEB-INF/xxx.tld）

     整个文件的定义：<short-name>  uri

     标签的定义：<tag>

c、在jsp页面中使用标签：

    导入标签库（xxx.tld）

    使用标签

## 2.45. 说说自动登录功能的编码实现？

a、登录功能是用session实现的，就是向session对象中保存当前用户的对象。

b、自动功能是用cookie实现的，就是登录时将用户的信息保存为持久cookie。

c、下次访问时，读取请求中如果有用户信息的cookie就可以自动登录。

## 2.46. 如何防止表单重复提交？

使用session技术：

a、在regist.jsp页面中生成一个为一个随机值，将其保存到session中，同时将其保存为表单的隐藏域的值。

b、在处理注册的请求时，获取session中的值，获取请求参数的值，比较两者是否相同，如果相同说明不是重复提交，请求通过同时删除session中保存的的值，如果不相同则是重复提交，不能通过。

## 2.47. Request对象的主要方法有哪些？

setAttribute(String name,Object)：设置名字为name的request的参数值

getAttribute(String name)：返回由name指定的属性值

getAttributeNames()：返回request对象所有属性的名字集合，结果是一个枚举的实例

getCookies()：返回客户端的所有Cookie对象，结果是一个Cookie数组

getCharacterEncoding()：返回请求中的字符编码方式

getContentLength()：返回请求的Body的长度

getHeader(String name)：获得HTTP协议定义的文件头信息

getHeaders(String name)：返回指定名字的request Header的所有值，结果是一个枚举的实例

getHeaderNames()：返回所以request Header的名字，结果是一个枚举的实例

getInputStream()：返回请求的输入流，用于获得请求中的数据

getMethod()：获得客户端向服务器端传送数据的方法

getParameter(String name)：获得客户端传送给服务器端的有name指定的参数值

getParameterNames()：获得客户端传送给服务器端的所有参数的名字，结果是一个枚举的实例

getParametervalues(String name)：获得有name指定的参数的所有值

getProtocol()：获取客户端向服务器端传送数据所依据的协议名称

getQueryString()：获得查询字符串

getRequestURI()：获取发出请求字符串的客户端地址

getRemoteAddr()：获取客户端的IP地址

getRemoteHost()：获取客户端的名字

getSession([Boolean create])：返回和请求相关Session

getServerName()：获取服务器的名字

getServletPath()：获取客户端所请求的脚本文件的路径

getServerPort()：获取服务器的端口号

removeAttribute(String name)：删除请求中的一个属性

## 2.48. Tomcat根目录下有哪些文件？

a、config：配置文件存放路径

b、webapps：项目部署的目录

c、bin Tomcat运行需要的脚本与jar的目录

d、lib：运行项目时所需要的jar的目录

e、word：部署项目的缓存目录

f、temp：临时文件的存放目录

g、logs：记录日志的目录

## 2.49. 什么是Tomcat，怎样启动停止，配置文件，日志文件的存储？

Tomcat是一种web服务器，java编写的web项目可以部署在上面，用户在客户端请求时，都是将请求发到Tomcat上，Tomcat在将请求发到对应的项目上。

启动Tomcat：

在windows下：进入bin目录，双击startup.bat

在Linux下：cd进入bin目录，sh startup.sh

在开发工具Eclipse下：右键选择debug server或者run server

停止Tomcat：

在windows下：进入bin目录，双击shutdown.bat

在Linux下：cd进入bin目录，sh shutdown.sh

在开发工具Eclipse下：右键选择stop server

配置文件在Tomcat的config目录下

日志文件在Tomcat的logs目录下

## 2.50. 实例化servlet有几种方式？

有两种方式

a、第一次请求时，实例化servlet对象

b、在web.xml文件中，在servlet标签中添加loadon-startup标签，tomcat启动时就会实例化servlet对象。

## 2.51. JSP乱码如何解决？

a、JSP页面乱码

<%@page contentType="text/html;charset=utf-8" %>

b、表单提交时出现乱码

request.setCharacterEncoding("utf-8");

c、数据库出现乱码

jdbc:mysql://localhost:3306:/user?useSSL=false&useUnicode=true&characterEncoding=utf-8;

其实我一般的处理的方法就是配置一个过滤器对每个JSP页面进行字符集处理。

## 2.52. session和application的区别？

a、两者的作用范围不同

session是用户级别的，application是web应用程序级别的，是一个全局作用域。

一个用户一个session，每个用户的session不同，用户所访问的网站多个页面共享同一个session。

一个web应用程序一个application对象，每个web应用程序的application对象不同，但是一个web应用程序的多个用户之间共享一个application。

b、生命周期不同

session：用户首次在网站中发出请求创建session，用户注销、离开网站或者关闭浏览器消亡。

application：启动web服务器创建，关闭web服务器销毁。

## 2.53. 在Eclipse中调试时，怎样查看一个变量的值？

在要查看的变量前先设置断点，然后选中变量，右键选择debug as →Java Application，打开debug视图，这时在Variables窗口中可以看到变量当前的值。

如果是局部变量，也可以在局部变量窗口中查看。要知道一个方法被调用的方法调用链，可以在方法栈中查看。

## 2.54. 你们公司使用的代码配置管理工具是什么？

除了说以前使用的CVS,现在新项目使用SVN了，还简要说一下使用的过程，如果有可能，还说说仓库的概念和如何使用锁之类的细节。

## 2.55. 你们项目总金额多少，多少人开发，总共花了多少个月？

像巴巴运动网这种规模的项目，可以说是4、5个人开发了四五个月，费用则是四五十万。按每人每月两万收入去计算就差不多了。

## 2.56. 什么是重量级？什么是轻量级？

轻量级是指它的创建和销毁不需要消耗太多的资源，意味着可以在程序中经常创建和销毁session对象。

重量级意味着不能随意的创建和销毁它的实例，会占用太多的资源。

## 2.57. 如何从form表单中获取checkbox的值？

可以在页面将checkbox的name属性设置相同，value属性取每个条目的id，在后台使用request.getParametervalues（"name"）;获取一组值。

## 2.58. 过滤器有哪些作用？

a、可以验证客户是否来自可信的网络

b、可以对客户提交的数据进行重写编码

c、可以从系统里获得配置的信息

d、可以过滤掉客户某些不应该出现的词汇

e、验证客户是否登录

f、验证客户的浏览器是否支持当前的应用

g、可以记录系统的日志

## 2.59. 过滤器的用法？（对客户端的请求统一编码和对客户端进行验证）

实现Filter接口，同时重写Filter接口的三个方法：

init(FilterConfig config)    用于获取FilterConfig对象

doFilter(ServletRequest request,ServeltResponse response,FilterChain chain)   进行过滤处理一些业务

destroy()    销毁Filter

## 2.60. C/S与B/S的区别？

有如下8个方面的不同：

a、硬件环境的不同

C/S：一般建立在专用的网络上，小范围里的网络环境，局域网之间再通过专门服务器提供连接和数据交换服务。  
　　B/S：建立在广域网之上的，不必是专门的网络硬件环境，有比C/S更强的适应范围，一般只要有操作系统和浏览器就行。  
b、对安全要求不同   
　　C/S：一般面向相对固定的用户群，对信息安全的控制能力很强。一般高度机密的信息系统采用C/S结构适宜。  
　　B/S：建立在广域网之上，对安全的控制能力相对弱，可能面向不可知的用户。  
c、对程序架构不同   
　　C/S：程序可以更加注重流程，可以对权限多层次校验，对系统运行速度可以较少考虑。  
　　B/S：对安全以及访问速度的多重的考虑，建立在需要更加优化的基础之上。  
d、软件重用不同   
　　C/S：程序可以不可避免的整体性考虑，构件的重用性不如在B/S要求下的构件的重用性好.  
　　B/S：对的多重结构，要求构件相对独立的功能。  
e、系统维护不同    
　　C/S：程序由于整体性，必须整体考察，处理出现的问题以及系统升级。  
　　B/S：构件组成，方面构件个别的更换,实现系统的无缝升级。  
f、处理问题不同   
　　C/S：程序可以处理用户面固定，并且在相同区域，安全要求高需求，与操作系统相关。  
　　B/S：建立在广域网上，面向不同的用户群，分散地域，这是C/S无法作到的。   
g、用户接口不同   
　　C/S：多是建立的Window平台上，表现方法有限，对程序员普遍要求较高。  
　　B/S：建立在浏览器上，有更加丰富和生动的表现方式与用户交流；并且大部分难度减低,减低开发成本。  
h、信息流不同   
　　C/S：程序一般是典型的中央集权的机械式处理，交互性相对低。  
　　B/S：信息流向可变化，B-B B-C B-G等信息、流向的变化，更像交易中心。

## 2.61. 什么是web容器？

给处于其中的应用程序组件（JSP、Servlet）提供一个环境，是JSP、Servlet直接跟容器中的变量交互，不必关注其他系统问题。

主要有web服务器来实现。例如：tomcat、weblogic、sphere、JBoss等。该容器提供的接口严格遵守J2EE规范中的web application标准。

我们把遵守以上标准的web服务器叫做J2EE的web容器。

## 2.62. 什么是EJB容器？

Enterprice Java Bean容器。它提供给运行在其中的组件EJB各种管理功能。只要满足J2EE规范的EJB放入该容器，马上就会被容器

进行高效率的管理。并且可以通过现成的接口来获得系统级别的服务。例如：邮件服务、事务管理……

## 2.63. 开发中都用到了哪些设计模式？用在什么场合？

每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心。通过这种方式，你可以无数次的使用

哪些已有的解决方案，无需再重复相同的工作。主要用到了MVC设计模式。用来开发JSP/Servlet或者J2EE的相关应用。简单工厂模式等。

## 2.64. 说说你所熟悉的或听说过的J2EE中的几种常用模式？及对设计模式的一些看法。

Session Facade Pattern：使用SessionBean访问EntityBean。

Message Facade Pattern：实现异步调用

EJB Command Pattern：使用Command.JavaBeans取代SessionBean，显示轻量级访问。

Data Transfer Object Factory：通过DTO Factory简化EntityBean数据提供特性

Generic Attribute Access:通过AttributeAccess接口简化EntityBean数据提供特性

Business Interface：通过远程（本地）接口和Bean类实现相同接口规范业务逻辑一致性

EJB架构的设计好坏将直接影响系统的性能、可扩展性、可维护性、组件可重用性及开发效率。项目越复杂，项目队伍余越庞大，则越能体现良好设计的重要性。

## 2.65. J2EE常用的设计模式？说明工厂模式。

Java中一共有23中设计模式：

Factory（工厂模式）、Builder（建造模式）、Factory Method（工厂方法模式）、ProtoType（原始模型模式）、Singleton（单例模式）、

Facade（门面模式）、Adapter（适配器模式）、Bridge（桥梁模式）、Composite（合成模式）、Decorator（装饰模式）、

FlyWeight（享元模式）、Proxy（代理模式）、Command（命令模式）、Interpreter（解释器模式）、Visitor（访问者模式）、

Iterator（迭代子模式）、Mediator（调停者模式）、Memento（备忘录模式）、Observer（观察者模式）、State（状态模式）、

Strategy（策略模式）、Template Method（模板方法模式）、Chain Of Responsibility（责任链模式）、

工厂模式：工厂模式是一种经常被使用到的模式，根据工厂模式实现的类可以根据提供的数据生成一组类中某个类的实例，通常一组类中有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作。首先需要定义一个基类，该类的子类通过不同的方法实现了基类中的方法。然后定义一个工厂类，工厂类可以根据条件生成不同的子类实例。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

## 2.66. EJB2.0有哪些内容？分别用在什么场合？EJB2.0和EJB2.1的区别？

规范内容包括Bean提供者，应用程序装配者，EJB容器，EJB配置工具，EJB服务提供者，系统管理员。这里面，EJB容器是EJB之所有能够运行的核心。

EJB容器管理着EJB的创建，撤消，激活，去活，与数据库的连接等等重要的核心工作。

## 2.67. EJB与JAVA BEAN的区别？

Java Bean 是可复用的组件，对Java Bean并没有严格的规范，理论上讲，任何一个Java类都可以是一个Bean。但通常情况下，

由于Java Bean是被容器所创建（如Tomcat)的，所以Java Bean应具有一个无参的构造器，另外，通常Java Bean还要实现

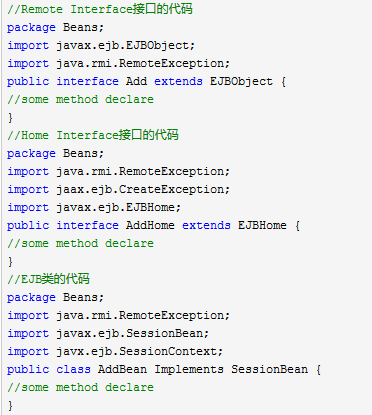
Serializable接口用于实现Bean的持久性。Java Bean实际上相当于微软COM模型中的本地进程内COM组件，它是不能被跨进程访问的。

Enterprise Java Bean 相当于DCOM，即分布式组件。它是基于Java的远程方法调用（RMI）技术的，所以EJB可以被远程访

（跨进程、跨计算机）。但EJB必须被布署在诸如Webspere、WebLogic这样的容器中，EJB客户从不直接访问真正的EJB组件，

而是通过其容器访问。EJB容器是EJB组件的代理，EJB组件由容器所创建和管理。客户通过容器来访问真正的EJB组件。

## 2.68. EJB的基本架构



## 2.69. 存储过程和函数的区别

存储过程是用户定义的一系列sql语句的集合，涉及特定表或者其他对象的任务，用户可以调用存储过程，而函数通常是数据库

已定义的方法，它接受参数并返回某种类型的值并且不涉及特定用户表。

## 2.70. 什么是事务？

事务时作为一个逻辑单元执行的一系列操作，一个逻辑工作单元必须有四个属性，称为ACID（原子性、一致性、隔离性和持久性）属性，

只有这样才能成为一个事务：

原子性：事务必须是原子工作单元，对于其数据修改，要么全都执行，要么全都不执行。

一致性：事务在完成时，必须使所有的数据保持一致的状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如B树索引或双向链表）都必须是正确的。

隔离性：由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务查看数据时数据所处的状态，要么是另一并发事务修改它之前

的状态，要么是另一并发事务修改它之后的状态，事务不会查看中间状态的数据。这称为可串行性，因为它能够重新装载起始数据，并且重播

一系列事务，以使数据结束时的状态与原始事务执行的状态相同。

持久性：事务完成后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。

## 2.71. 触发器分为事前触发和事后触发，两种触发有什么区别？语句级触发和行级触发有什么区别？

事前触发运行于触发事件发生之前，而事后触发运行于触发事件发生之后。通常事前触发器可以获取事件之前和新的字段值。

语句级触发可以在语句执行前或后执行，而行级触发在触发器所影响的每一行触发一次。

## 2.72. EJB容器提供的服务

主要提供声明周期管理、代码产生、持续性管理、安全、事务管理、锁和并发行管理等服务。

## 2.73. EJB的三个角色和三个对象

角色：EJB角色主要包括Bean开发者、应用组装者、部署者、系统管理员、EJB容器提供者、EJB服务器提供者。

对象：Remote（Local）接口、Home（LocalHome）接口、Bean类

## 2.74. EJB的几种类型？

会话（Session） Bean，实体（Entity）Bean、消息驱动的（Message Driven）Bean

会话Bean又可分为有状态（Stateful）和无状态（Stateless）

实体Bean可分为Bean管理的持续性（BMP）和容器管理的持续性（CMP）两种

## 2.75. Bean实例的生命周期

对于Stateless Session Bean、Entity Bean、Message Driven Bean一般存在缓冲池管理，而对于Entity Bean和

Stateful Session Bean存在Cache管理，通常包含创建实例，设置上下文、创建EJB Object（create）、业务方法调用、

remove等过程，对于存在缓冲池管理的Bean，在create之后实例并不从内存清除，而是采用缓冲池机制不断重用实例；

而对于存在Cache管理的Bean，则通过激活或去激活机制保持Bean的状态并限制内存中实例数量。

## 2.76. 激活机制

以Stateful Session Bean为例：其Cache大小决定了内存中可以同时存在的Bean实例的数量，根据MRU或NRU算法，

实例在激活和去激活状态之间迁移，激活机制是当客户端调用某个EJB实例业务方法时，如果对应的EJB Object发现自己

没有捆绑对应的Bean实例，则从其去激活Bean存储中（通过序列化机制存储实例）回复（激活）此实例。状态变迁前会

调用对应的EJB Active 和EJB Passivate方法。

## 2.77. remote接口和home接口的作用

remote接口中定义了业务方法，用于EJB用户调用业务方法。

home是EJB工厂用于创建、移除和查找EJB实例

## 2.78. 客服端调用EJB对象的几个基本步骤

a、设置JNDI服务工厂以及JNDI服务地址系统属性

b、查找home接口

c、从home接口调用create方法创建remote接口

d、通过remote接口调用其业务方法

# 3.数据库部分

## 3.1. 数据库的分类及常用的数据库

数据库分为：关系型数据库和非关系型数据库

关系型：mysql oracle sqlserver等

非关系型：redis,memcache,mogodb,hadoop等

## 3.2. 简单介绍一下关系数据库三范式？

范式就是规范,就是关系型数据库在设计表时，要遵循的三个规范。

要想满足第二范式必须先满足第一范式，要满足第三范式必须先满足第二范式。

第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。列数据的不可分割

二范式（2NF）要求数据库表中的每个行必须可以被唯一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的唯一标识。(主键)

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。(外键)

反三范式,有的时候为了效率，可以设置重复或者可以推导出的字段.

订单（总价）和订单项（单价）

## 3.3. 事务四个基本特征或 ACID 特性。

事务是并发控制的单位，是用户定义的一个操作序列。这些操作要么都做，要么都不做，是一个不可分割的工作单位。

一个转账必须 A账号扣钱成功，B账号加钱成功，才算正真的转账成功。

事务必须满足四大特征:原子性,一致性,隔离性,持久性/持续性

原子性：表示事务内操作不可分割。要么都成功、要么都是失败.

一致性：要么都成功、要么都是失败.后面的失败了要对前面的操作进行回滚。

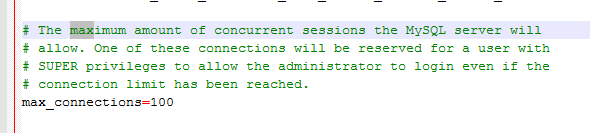
隔离性：一个事务开始后，不能受其他事务干扰。

持久性/持续性：表示事务开始了，就不能终止。

## 3.4. mysql数据库的默认的最大连接数？

100

为什么需要最大连接数？特定服务器上面的数据库只能支持一定数目同时连接，这时候我们一般都会设置最大连接数（最多同时服务多少连接）。在数据库安装时都会有一个默认的最大连接数为100



## 3.5. 说一下msyql的分页？Oracle的分页？

Mysql是使用关键字limit来进行分页的 limit offset,size 表示从多少索引去多少位.

Oracle的分页，大部分情况下，我们是记不住了。说思路，要使用三层嵌套查询。

Oracle的分页有点儿记不住了，只记得一些大概。是使用了三层嵌套查询。如果在工作中使用了，可以到原来的项目中拷贝或上网查询。

mysql:

String sql =

"select \* from students order by id limit " + pageSize\*(pageNumber-1) + "," + pageSize;

oracle:

String sql =

"select \* from " +

(select \*,rownum rid from (select \* from students order by postime desc) where rid<=" + pagesize\*pagenumber + ") as t" +

"where t>" + pageSize\*(pageNumber-1);

## 3.6. 简单讲一下数据库的触发器的使用场景？

触发器，需要有触发条件，当条件满足以后做什么操作。

触发器用处还是很多的，比如校内网、开心网、Facebook，你发一个日志，自动通知好友，其实就是在增加日志时做一个后触发，再向通知表中写入条目。因为触发器效率高。而UCH没有用触发器，效率和数据处理能力都很低。

每插入一个帖子，都希望将版面表中的最后发帖时间，帖子总数字段进行同步更新，用触发器做效率就很高。

|  |
| --- |
| create table board1(id int primary key auto\_increment,name varchar(50),articleCount int);  create table article1(id int primary key auto\_increment,title varchar(50),bid int references board1(id));  delimiter |#把分割符;改成|  create trigger insertArticle\_Trigger after insert on article1 for each row begin  -> update board1 set articleCount=articleCount+1 where id= NEW.bid;  -> end;  -> |  delimiter ;  insert into board1 value (null,'test',0);  insert into article1 value(null,'test',1); |

## 3.7. 简单讲一下数据库的存储过程的使用场景？

数据库存储过程具有如下优点：

1、存储过程只在创建时进行编译，以后每次执行存储过程都不需再重新编译，而一般 SQL 语句每执行一次就编译一次，因此使用存储过程可以大大提高数据库执行速度。

2、通常，复杂的业务逻辑需要多条 SQL 语句。这些语句要分别地从客户机发送到服务器，当客户机和服务器之间的操作很多时，将产生大量的网络传输。如果将这些操作放在一个存储过程中，那么客户机和服务器之间的网络传输就会大大减少，降低了网络负载。

3、存储过程创建一次便可以重复使用，从而可以减少数据库开发人员的工作量。

4、安全性高，存储过程可以屏蔽对底层数据库对象的直接访问，使用 EXECUTE 权限调用存储过程，无需拥有访问底层数据库对象的显式权限。

正是由于存储过程的上述优点，目前常用的数据库都支持存储过程，例如 IBM DB2，Microsoft SQL Server，Oracle，Access 等，开源数据库系统 MySQL 也在 5.0 的时候实现了对存储过程的支持。

定义存储过程:

|  |
| --- |
| create procedure insert\_Student (\_name varchar(50),\_age int ,out \_id int)  begin  insert into student value(null,\_name,\_age);  select max(stuId) into \_id from student;  end;  call insert\_Student('wfz',23,@id);  select @id; |

## 3.8. 用jdbc怎么调用存储过程

贾琏欲执事

加载驱动

获取连接

设置参数

执行

释放连接

|  |
| --- |
| **package** com.huawei.interview.lym;  **import** java.sql.CallableStatement;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.SQLException;  **import** java.sql.Types;  **public** **class** JdbcTest {  /\*\*  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  Connection cn = **null**;  CallableStatement cstmt = **null**;  **try** {  //这里最好不要这么干，因为驱动名写死在程序中了  Class.*forName*("com.mysql.jdbc.Driver");  //实际项目中，这里应用DataSource数据，如果用框架，  //这个数据源不需要我们编码创建，我们只需Datasource ds = context.lookup()  //cn = ds.getConnection();  cn = DriverManager.*getConnection*("jdbc:mysql:///test","root","root");  cstmt = cn.prepareCall("{call insert\_Student(?,?,?)}");  cstmt.registerOutParameter(3,Types.*INTEGER*);  cstmt.setString(1, "wangwu");  cstmt.setInt(2, 25);  cstmt.execute();  //get第几个，不同的数据库不一样，建议不写  System.*out*.println(cstmt.getString(3));  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  **finally**  {  /\*try{cstmt.close();}catch(Exception e){}  try{cn.close();}catch(Exception e){}\*/  **try** {  **if**(cstmt != **null**)  cstmt.close();  **if**(cn != **null**)  cn.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

## 3.9. 简单说一下你对jdbc的理解？

Java database connection java数据库连接.数据库管理系统(mysql oracle等)是很多，每个数据库管理系统支持的命令是不一样的。

Java只定义接口，让数据库厂商自己实现接口，对于我们者而言。只需要导入对应厂商开发的实现即可。然后以接口方式进行调用.(mysql + mysql驱动（实现）+jdbc)

## 3.10. 写一个简单的jdbc的程序。写一个访问oracle数据的jdbc程序?

贾琏欲执事

加载驱动(com.mysql.jdbc.Driver,oracle.jdbc.driver.OracleDriver)

获取连接(DriverManager.getConnection(url,usernam,passord))

设置参数 Statement PreparedStatement

cstmt.setXXX(index, value);

执行 executeQuery executeUpdate

释放连接(是否连接要从小到大，必须放到finnaly)

## 3.11. JDBC中的PreparedStatement相比Statement的好处

大多数我们都使用PreparedStatement代替Statement

1：PreparedStatement是预编译的，比Statement速度快

2：代码的可读性和可维护性

虽然用PreparedStatement来代替Statement会使代码多出几行,但这样的代码无论从可读性还是可维护性上来说.都比直接用Statement的代码高很多档次：

stmt.executeUpdate("insert into tb\_name (col1,col2,col2,col4) values ('"+var1+"','"+var2+"',"+var3+",'"+var4+"')");

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();

不用我多说,对于第一种方法，别说其他人去读你的代码，就是你自己过一段时间再去读,都会觉得伤心。

3：安全性

PreparedStatement可以防止SQL注入攻击，而Statement却不能。比如说：

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。

## 3.12. 数据库连接池作用

1、限定数据库的个数，不会导致由于数据库连接过多导致系统运行缓慢或崩溃

2、数据库连接不需要每次都去创建或销毁，节约了资源

3、数据库连接不需要每次都去创建，响应时间更快。

## 3.13. Mysql的四种隔离级别

（1）Read Uncommitted（读取未提交内容）在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。造成脏读现象。未提交读是最低的隔离级别。

（2）Read Committed（读取提交内容）：一个事务只能看见已经提交事务所做的改变。这种隔离级别 也支持所谓的不可重复读（不可重复读指事务A在事务期间对事务B修改数据并提交前后查询的数据不一样）

（3）Repeatable Read（可重读）：同一事务在启动事务开始，中间无论发生什么样的操作，该事务所读取到的数据行是一样的。这会出现幻读 的问题。

（4）Serializable（可串行化）

serializable完全锁定字段，若一个事务来查询同一份数据就必须等待，直到前一个事务完成并解除锁定为止。是完整的隔离级别，会锁定对应的数据表格，因而会有效率的问题和超时现象。

# 4. 前端部分

## 4.1. 简单说一下html,css,javascript在网页开发中的定位?

HTML 超文本标记语言 定义网页的结构

CSS 层叠样式表，用来美化页面

JavaScript主要用来验证表单，做动态交互(其中ajax)

## 4.2. 简单介绍一下Ajax?

Ajax是异步的javascript和xml

作用是通过AJAX与服务器进行数据交换，AJAX可以使网页实现布局更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

Ajax的实现使用了XmlHttpRequest对象，使用这个对象可以异步向服务器发送请求，并且获取响应，完成局部更新。Ajax还包含Open（打开）请求、send(发送)请求、responseText/responseXml(接收)请求、局部响应

使用场景 登陆失败时不跳转页面，注册时提示用户名是否存在,二级联动等等使用场景

## 4.3. js和jQuery的关系?

jQuery是一个js框架，封装了js的属性和方法，让用户使用起来更加便利。并且增强了js的功能。

原生的js是要处理很多兼容性的问题(比如：注册事件等)，用了jQuery封装了底层，就不用处理兼容性问题了。

原生的js的dom和事件绑定和Ajax等操作非常麻烦，jQuery封装以后操作非常方便。

## 4.4. jQuery的常用选择器？

ID选择器 通过ID获取一个元素

Class选择器 通过类(css)获取元素

标签选择器 通过标签获取元素

通用选择器(\*) 获取所以的元素

div.myCls 获取有myCls这个类的div

层次选择器

儿子选择器 > 获取下面的子元素

后代选择器 空格 获取下面后代，包括儿子、孙子等后代

属性选择器

Tag[attrName=’test’] 获取有属性名为xxxx并且属性的值为test的所有xxx标签

<input type=”checkbox” name=”hobby”/> 吃饭<br/>

<input type=”checkbox” name=”hobby”/> 睡觉<br/>

Input[name=’hobby’],表示获取属性名为name并且name属性值为hobby的的所有input标签元素

## 4.5. jQuery的页面加载完毕事件？

很多时候我们需要获取元素，但是必须等到该元素被加载完成后才能获取。我们可以把js代码放到该元素的后面，但是这样就会造成js在我们的body中存在不好管理。所有页面加载完毕后所有的元素当然已经加载完毕。一般获取元素做操作都要在页面加载完毕后操作。

|  |  |
| --- | --- |
| 第一种:  $(document).ready(function(){  });  $(document)把原生的document这个dom对象转换为jQuery对象，转换完成后才能调用ready方法  ready(fn),表示的是页面结构被加载完毕后执行传入函数fn | 第二种:  $(function(){  });  当页面加载完毕后执行里面的函数,这一种相对简单，用得最多。 |
| window.onload的区别   1. jQuery中的页面加载完毕事件，表示的是页面结构被加载完毕。 2. window.onload 表示的是页面被加载完毕。   <img src=”htttp://baidu.com/1.jpg”/> onload必须等等页面中的图片、声音、图像等远程资源被加载完毕后才调用而jQuery中只需要页面结构被加载完毕，指img标签加载了就可以。 | |

## 4.6. Jquery的Ajax和原生Js实现Ajax有什么关系？

jQuery中的Ajax也是通过原生的js封装的。封装完成后让我们使用起来更加便利，不用考虑底层实现或兼容性等处理。

如果采用原生js实现Ajax是非常麻烦的，并且每次都是一样的。如果我们不使用jQuery我们也要封装Ajax对象的方法和属性。有像jQuery这些已经封装完成，并经过很多企业实际的框架，比较可靠并且开源。我们就不需要封装，直接使用成熟的框架(jQuery)即可.

## 4.7. 简单说一下html5?你对现在的那些新技术有了解?

Html5是最新版本的html,是在原来html4的基础上增强了一些标签。

Html增加一些像画板、声音、视频、web存储等高级功能。但是html5有一个不好的地方，那就是html5太强调语义了，导致开发中都不知道要选择那个标签。

以前在做页面布局时，无论头部、主题、导航等模块都使用div来表示，但是html5的规范，需要使用不同的标签来表示。(header footer等)

## 4.8. 简单说一下css3?

Css3是最新版本的css,是对原理css2的功能增强。

Css3中提供一些原来css2中实现起来比较困难或者不能实现的功能。

1、盒子圆角边框

2、盒子和文字的阴影

3、渐变

4、转换 移动、缩放、旋转等

5、过渡、动画都可以使用动画。

6、可以使用媒体查询实现响应式网站。

Css3最大缺点就是要根据不同的浏览器处理兼容性。对应有一些处理兼容性的工具。不用担心.

## 4.9. bootstrap是什么？

BootStrap是一个移动设备优先的UI框架。我们可以不用谢任何css,js代码就能实现比较漂亮的有交互性的页面。我们程序员对页面的编写是有硬伤的，所有要自己写页面的话就要使用类似于bootstrap这样的UI框架。

平时用得很多的：

1. 模态框
2. 表单，表单项
3. 布局
4. 删格系统

# 5. 框架部分

## 5.1. 什么是框架?

框架（Framework）是一个框子——指其约束性，也是一个架子——指其支撑性。

IT语境中的框架，特指为解决一个开放性问题而设计的具有一定约束性的支撑结构。在此结构上可以根据具体问题扩展、安插更多的组成部分，从而更迅速和方便地构建完整的解决问题的方案。

1）框架本身一般不完整到可以解决特定问题,但是可以帮助您快速解决特定问题；

没有框架所有的工作都从零开始做,有了框架,为我们提供了一定的功能,我们就可以在框 架的基础上开发,极大的解放了生产力。

不同的框架，是为了解决不同领域的问题。一定要为了解决问题才去学习框架。

2）框架天生就是为扩展而设计的；

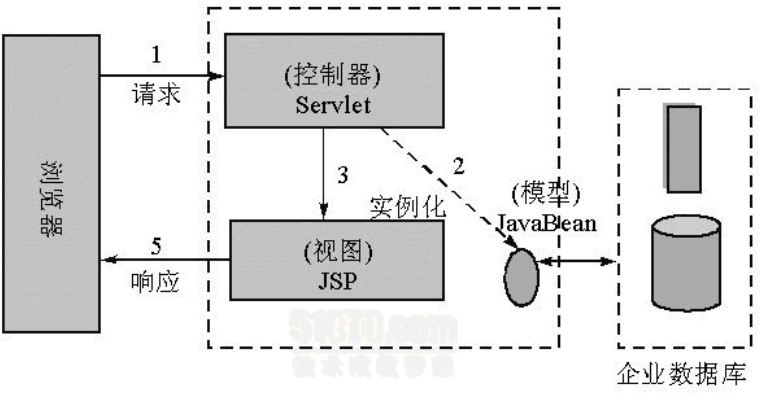
3）框架里面可以为后续扩展的组件提供很多辅助性、支撑性的方便易用的实用工具（utilities），也就是说框架时常配套了一些帮助解决某类问题的库（libraries）或工具（tools）。

java中就是一系列的jar包，其本质就是对jdk功能的扩展.

## 5.2. MVC模式

MVC全名是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

最简单的、最经典就是Jsp(view) +Servlet(controller) + JavaBean(model)



1. 当控制器收到来自用户的请求
2. 控制器调用JavaBean完成业务
3. 完成业务后通过控制器跳转JSP页面的方式给用户反馈信息
4. Jsp给 用户做出响应。

控制器都是核心

## 5.3. MVC框架

MVC框架是为了解决传统MVC模式(Jsp + Servlet + JavaBean)的一些问题而出现的框架。

传统MVC模式问题

1. 所有的Servlet和Servlet映射都要配置在web.xml中，如果项目太大，web.xml就太庞大，并且不能实现模块化管理。
2. Servlet的主要功能就是接受参数、调用逻辑、跳转页面，如果出现像字符编码、文件上传等功能也要写在Servlet中，这个时候servlet就不能只处理主要功能，特例功能也需要处理。

3、接受参数比较麻烦(String name = request.getParameter(“name”),User user=new User user.setName(name))，不能通过model接收，只能单个接收，接收完成后转换封装model.

4、跳转页面方式比较单一(forword,redirect),并且当我的页面名称发生改变时需要修改Servlet源代码.

现在比较常用的MVC框架有：

struts

webwork

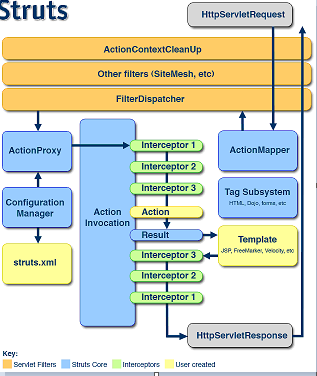
Struts2

Spring MVC

## 5.4．struts2

### 5.4.1．简单讲一下struts2的执行流程？

Struts2的原理?



一个请求在Struts2框架中的处理大概分为以下几个步骤：

1、客户端浏览器发送请求

2、这个请求经过一系列的过滤器（Filter）（这些过滤器中有一个叫做ActionContextCleanUp的可选过滤器，这个过滤器对于Struts2和其他框架的集成很有帮助，例如：SiteMesh Plugin）；

3.最后到达FilterDispatcher(StrutsPrepareAndExecuteFilter)被调用，FilterDispatcher

(StrutsPrepareAndExecuteFilter)询问ActionMapper来决定这个请求是否需要调用某个Action；

1. 如果ActionMapper决定需要调用某个Action，FilterDispatcher

(StrutsPrepareAndExecuteFilter)把请求的处理交给ActionProxy；

1. ActionProxy通过Configuration Manager询问框架的配置文件，找到需要调用的Action类；
2. ActionProxy创建一个ActionInvocation的实例。
3. ActionInvocation实例使用命名模式来调用，在调用Action的过程前后，涉及到相关拦截器（Intercepter）的调用。
4. 一旦Action执行完毕，ActionInvocation负责根据struts.xml中的配置找到对应的返回结果。返回结果通常是（但不总是，也可能是另外的一个Action链）一个需要被表示的JSP或者FreeMarker的模版。在表示的过程中可以使用Struts2框架中继承的标签。在这个过程中需要涉及到ActionMapper。

面试：

1. 浏览器发送请求，经过一系列的过滤器后，到达核心过滤器(StrutsPrepareAndExecuteFilter).
2. StrutsPrepareAndExecuteFilter通过ActionMapper判断当前的请求是否需要某个Action处理,如果不需要，则走原来的流程。如果需要则把请求交给ActionProxy来处理
3. ActionProxy通过Configuration Manager询问框架的配置文件(Struts.xml)，找到需要调用的Action类；
4. 创建一个ActionInvocation实例，来调用Action的对应方法来获取结果集的name,在调用前后会执行相关拦截器。
5. 通过结果集的Name知道对应的结果集来对浏览器进行响应。

拦截、判断、寻找、执行、响应

### 5.4.2. Struts2中的拦截器，你都用它干什么？

java里的拦截器是动态拦截Action调用的对象。它提供了一种机制可以使开发者可以定义在一个action执行的前后执行的代码，也可以在一个action执行前阻止其执行，同时也提供了一种可以提取action中可重用部分的方式。

在AOP（Aspect-Oriented Programming）中拦截器用于在某个方法或字段被访问之前，进行拦截然后在之前或之后加入某些操作。

面试：

struts2中的的功能（参数处理、文件上传、字符编码等）都是通过系统拦截器实现的。

如果业务需要，当然我们也可以自定义拦截器,进行可插拔配置，在执行Action的方法前后、加入相关逻辑完成业务。

使用场景：

1. 用户登录判断，在执行Action的前面判断是否已经登录，如果没有登录的跳转到登录页面。
2. 用户权限判断，在执行Action的前面判断是否具有，如果没有权限就给出提示信息。
3. 操作日志......

4、......

## 5.5．Spring

### 5.5.1. 说一下Spring中的两大核心？

**Spring的核心是控制反转(IOC)和面向切面(AOP)**

IOC

IOC 全称 Inversion Of Control 即控制反转。我的理解就是在之前我们对象都是自己创建的，也就是使用 new 关键字创建的，而现在我们不去自己创建对象了，而是让框架去帮我们创建对象。它的目的就是为了程序解耦，当然它也只能是削减程序的耦合性，并不是完全消除，毕竟不可能完全消除掉程序的耦合性。框架实现的方式就是依赖注入即 DI。

DI

DI 全称 Dependency Injection 依赖注入。我们告诉框架我们要使用的类的全类名，然后在要使用的时候直接通过框架来获取这个类的实例对象。

AOP

AOP：全称是 Aspect Oriented Programming 即：面向切面编程。简单的说它就是把我们程序重复的代码抽取出来，在需要执行的时候，使用动态代理的技术，在不修改源码的基础上，对我们的已有方法进行增强。

### 5.5.2. AOP是什么？你都拿它做什么？

AOP:面向切面编程

核心原理：使用动态代理的设计模式在执行方法前后或出现异常做加入相关逻辑。

我们主要使用AOP来做：

1、事务处理 执行方法前，开启事务、执行完成后关闭事务、出现异常后回滚事务

2、权限判断 在执行方法前，判断是否具有权限

3、日志 在执行前进行日志处理

4、....

### 5.5.3 讲一下Spring的事务传播特性

多个事务存在是怎么处理的策略

1. PROPAGATION\_REQUIRED: 如果存在一个事务，则支持当前事务。如果没有事务则开启  
   2. PROPAGATION\_SUPPORTS: 如果存在一个事务，支持当前事务。如果没有事务，则非事务的执行  
   3. PROPAGATION\_MANDATORY: 如果已经存在一个事务，支持当前事务。如果没有一个活动的事务，则抛出异常。  
   4. PROPAGATION\_REQUIRES\_NEW: 总是开启一个新的事务。如果一个事务已经存在，则将这个存在的事务挂起。  
   5. PROPAGATION\_NOT\_SUPPORTED: 总是非事务地执行，并挂起任何存在的事务。  
   6. PROPAGATION\_NEVER: 总是非事务地执行，如果存在一个活动事务，则抛出异常  
   7. PROPAGATION\_NESTED：如果一个活动的事务存在，则运行在一个嵌套的事务中. 如果没有活动事务, 则按TransactionDefinition.PROPAGATION\_REQUIRED 属性执行

Propagation（传播）

Required 需要 如果存在一个事务，则支持当前事务。如果没有事务则开启

Supports 支持 如果存在一个事务，支持当前事务。如果没有事务，则非事务的执行

Mandatory 必要的 如果已经存在一个事务，支持当前事务。如果没有一个活动的事务，则抛出异常。

required\_new 总是开启一个新的事务。如果一个事务已经存在，则将这个存在的事务挂起。

Not\_support 总是非事务地执行，并挂起任何存在的事务。

Never 绝不 总是非事务地执行，如果存在一个活动事务，则抛出异常

Nested 嵌套的 如果有就嵌套、没有就开启事务

### 5.5.4. Spring事务的隔离级别

1. ISOLATION\_DEFAULT： 这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别.

另外四个与JDBC的隔离级别相对应  
2. ISOLATION\_READ\_UNCOMMITTED： 这是事务最低的隔离级别，它允许另外一个事务可以看到这个事务未提交的数据。  
这种隔离级别会产生脏读，不可重复读和幻像读。

3. ISOLATION\_READ\_COMMITTED： 保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据  
4. ISOLATION\_REPEATABLE\_READ： 这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。

它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。  
5. ISOLATION\_SERIALIZABLE 这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。

除了防止脏读，不可重复读外，还避免了幻像读。

在MySQL数据库中，支持上面四种隔离级别，默认的为Repeatable read (可重复读)；而在Oracle数据库中，只支持Serializable (串行化)级别和Read committed (读已提交)这两种级别，其中默认的为Read committed级别。

**其中的一些概念的说明：**

**脏读:** 指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据， 那么另外一 个事务读到的这个数据是脏数据，依据脏数据所做的操作可能是不正确的。

**不可重复读:** 指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。 那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。

**幻觉读:** 指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及 到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。

### 5.5.5.Spring在ssm中起什么作用？

Spring：轻量级框架

作用：Bean工厂，用来管理Bean的生命周期和框架集成。

两大核心：  
①. IOC/DI(控制反转/依赖注入) ：把dao依赖注入到service层，service层反转给action层，Spring顶层容器为BeanFactory。  
②. AOP：面向切面编程

### 5.5.6.Spring的事务？

spring支持编程式事务管理和声明式事务管理两种方式：

①编程式事务管理使用TransactionTemplate。

②声明式事务管理建立在AOP之上的。其本质是通过AOP功能，对方法前后进行拦截，将事务处理的功能编织到拦截的方法中，也就是在目标方法开始之前加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。

声明式事务最大的优点就是不需要在业务逻辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明或通过@Transactional注解的方式，便可以将事务规则应用到业务逻辑中。

声明式事务管理要优于编程式事务管理，这正是spring倡导的非侵入式的开发方式，使业务代码不受污染，只要加上注解就可以获得完全的事务支持。唯一不足地方是，最细粒度只能作用到方法级别，无法做到像编程式事务那样可以作用到代码块级别。

### 5.5.7.IOC在项目中的作用？

作用：Ioc解决对象之间的依赖问题，把所有Bean的依赖关系通过配置文件或注解关联起来，降低了耦合度。

### 5.5.8.Spring的配置文件中的内容？

开启事务注解驱动

事务管理器

开启注解功能，并配置扫描包

配置数据库

配置SQL会话工厂，别名，映射文件

不用编写Dao层的实现类

### 5.5.9.Spring下的注解？

注册：@Controller @Service @Component

注入：@Autowired @Resource

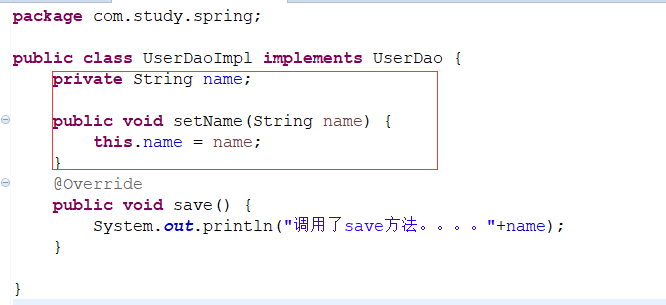
请求地址：@RequestMapping

返回具体数据类型而非跳转：@ResponseBody

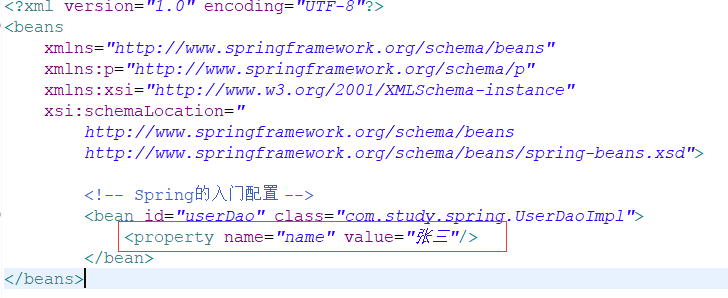
### 5.5.10.Spring DI（依赖注入）的三种方式?

第一种 Setter方法的方式注入属性

在UserDaoImpl.java类添加一个全局属性

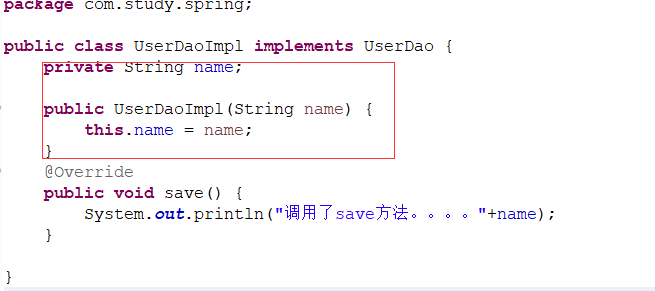


在xml文件中添加依赖属性

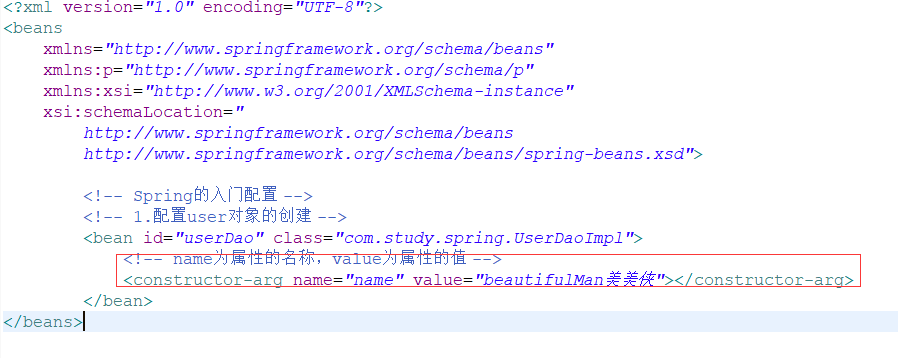


第二种 构造方法的方式注入属性

在UserDaoImpl.java类添加一个全局属性，并设置带参构造方法



在xml文件中添加依赖属性



第三种：接口注入

### 5.5.11.Spring主要使用了什么模式？

（1）工厂模式：BeanFactory就是简单工厂模式的体现，用来创建对象的实例；

（2）单例模式：Bean默认为单例模式。

（3）代理模式：Spring的AOP功能用到了JDK的动态代理和CGLIB字节码生成技术；

（4）模板方法：用来解决代码重复的问题。比如. RestTemplate, JmsTemplate, JpaTemplate。

（5）观察者模式：定义对象键一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都会得到通知被制动更新，如Spring中listener的实现--ApplicationListener。

### 5.5.12.IOC，AOP的实现原理？

IOC：通过反射机制生成对象注入；**XML配置文件+SAX解析+工厂设计模式**。

|  |  |  |
| --- | --- | --- |
| 配置文件中可能会有如下配置信息：   |  | | --- | | <bean id="userDaoImpl" class="cn.itcast.dao.impl.UserDaoImpl" /> |   也是要创建一个工厂类，在工厂类中提供一个返回实现类对象的方法，但并不是直接new实现类，而是使用SAX解析配置文件，根据标签bean中的id属性值得到对应的class属性值，使用反射创建实现类对象。   |  | | --- | | public class BeanFactory {  public static Object getBean(String id) {  // 1.使用SAX解析得到配置文件内容  // 直接根据id值userDaoImpl得到class属性值  String classvalue = "class属性值";  // 2.使用反射得到对象  Class clazz = Class.forName(classvalue);  return clazz.newInstance();  }  } | |

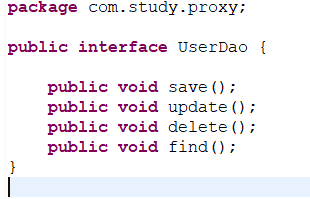
AOP：动态代理

当类实现了接口会默认使用JDK动态代理，不然会自动使用Cglib动态代理

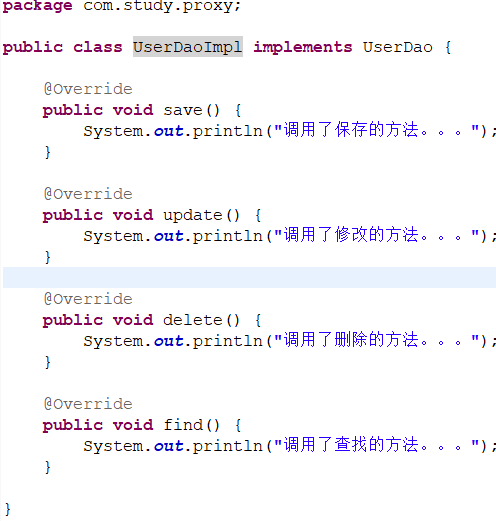
1. JDK动态代理：只能对实现了接口的类产生代理
2. Cglib动态代理（类似于javassist第三方代理技术）：对没有实现接口的类产生代理对象。生成子类对象。

1、JDK动态代理的底层实现

UserDao.java



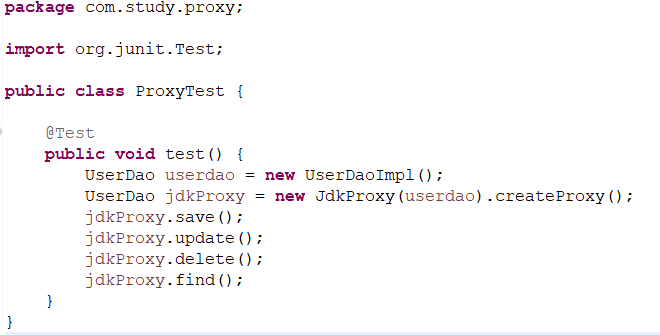
UserDaoImpl.java



JdkProxy.java



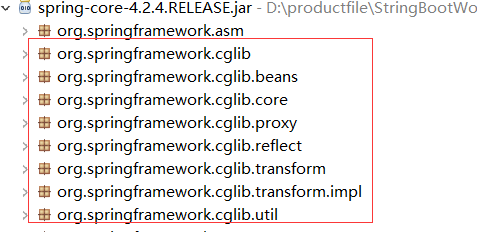
ProxyTest



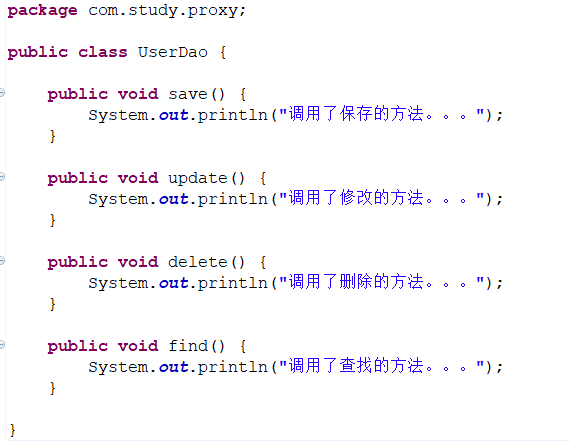
2、Cglib动态代理的底层实现

第三方开源代码生成类库，动态添加类属性和方法

Spring基础jar包中已经引入Cglib



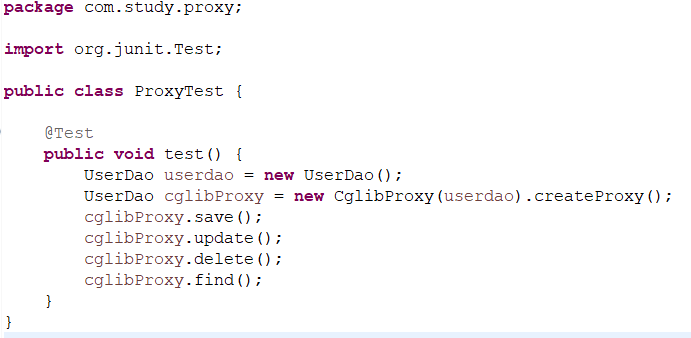
UserDao.java



CglibProxy.java



测试类



### 5.5.13. Spring是什么?

Spring是一个轻量级的IoC和AOP容器框架。是为Java应用程序提供基础性服务的一套框架，目的是用于简化企业应用程序的开发，它使得开发者只需要关心业务需求。常见的配置方式有三种：基于XML的配置、基于注解的配置、基于Java的配置。

主要由以下几个模块组成：

Spring Core：核心类库，提供IOC服务；

Spring Context：提供框架式的Bean访问方式，以及企业级功能（JNDI、定时任务等）；

Spring AOP：AOP服务；

Spring DAO：对JDBC的抽象，简化了数据访问异常的处理；

Spring ORM：对现有的ORM框架的支持；

Spring Web：提供了基本的面向Web的综合特性，例如多方文件上传；

Spring MVC：提供面向Web应用的Model-View-Controller实现。

### 5.5.14. Spring 的优点？

（1）spring属于低侵入式设计，代码的污染极低；

（2）spring的DI机制将对象之间的依赖关系交由框架处理，减低组件的耦合性；

（3）Spring提供了AOP技术，支持将一些通用任务，如安全、事务、日志、权限等进行集中式管理，从而提供更好的复用。

（4）spring对于主流的应用框架提供了集成支持。

### 5.5.15. 为什么说Spring是一个一站式的轻量级开源框架？

EE开发可分成三层架构，针对JavaEE的三层结构，每一层Spring都提供了不同的解决技术：

1. WEB层：SpringMVC
2. 业务层：Spring的Bean管理，Spring声明式事务
3. 持久层：Spring的Jdbc模板，Spring的ORM模块

### 5.5.16. 为什么要使用Spring

1、方便解耦，简化开发。

Spring就是一个大工厂，可以将所有对象的创建和依赖关系的维护，交给Spring管理。

2、AOP编程的支持

Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能。

3、声明式事务的支持

只需要通过配置就可以完成对事务的管理，而无须手动编程。

4、方便程序的测试

Spring对Junit4支持，可以通过注解方便的测试Spring程序。

5、方便集成各种优秀的框架

Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架(如：Struts2、Hibernate、MyBatis、Quartz等)的直接支持。

6、降低JavaEE API的使用难度

Spring对JavaEE开发中非常难用的一些API(JDBC、JavaMail、远程调用等)，都提供了封装，使这些API应用难度大大降低。

### 5.5.17. BeanFactory和ApplicationContext有什么区别？

BeanFactory和ApplicationContext是Spring的两大核心接口，都可以当做Spring的容器。其中ApplicationContext是BeanFactory的子接口。

BeanFactory：是Spring里面最底层的接口，包含了各种Bean的定义，读取bean配置文档，管理bean的加载、实例化，控制bean的生命周期，维护bean之间的依赖关系。ApplicationContext接口作为BeanFactory的派生，除了提供BeanFactory所具有的功能外，还提供了更完整的框架功能：

①继承MessageSource，因此支持国际化。

②统一的资源文件访问方式。

③提供在监听器中注册bean的事件。

④同时加载多个配置文件。



⑤载入多个（有继承关系）上下文 ，使得每一个上下文都专注于一个特定的层次，比如应用的web层。

**以下是三种较常见的 ApplicationContext 实现类**

1、ClassPathXmlApplicationContext：从classpath的XML配置文件中读取上下文，并生成上下文定义。应用程序上下文从程序环境变量中取得。

ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”);

2、FileSystemXmlApplicationContext ：由文件系统中的XML配置文件读取上下文。

ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”);

3、XmlWebApplicationContext：由Web应用的XML文件读取上下文。

### 5.5.18. Spring有几种配置方式？

将Spring配置到应用开发中有以下三种方式：

基于XML的配置  
基于注解的配置  
基于Java的配置

### 5.5.19. 请解释Spring Bean的生命周期？

首先说一下Servlet的生命周期：实例化，初始init，接收请求service，销毁destroy；

Spring上下文中的Bean生命周期也类似，如下：

（1）实例化Bean：

对于BeanFactory容器，当客户向容器请求一个尚未初始化的bean时，或初始化bean的时候需要注入另一个尚未初始化的依赖时，容器就会调用createBean进行实例化。对于ApplicationContext容器，当容器启动结束后，通过获取BeanDefinition对象中的信息，实例化所有的bean。

（2）设置对象属性（依赖注入）：

实例化后的对象被封装在BeanWrapper对象中，紧接着，Spring根据BeanDefinition中的信息 以及 通过BeanWrapper提供的设置属性的接口完成依赖注入。

（3）处理Aware接口：

接着，Spring会检测该对象是否实现了xxxAware接口，并将相关的xxxAware实例注入给Bean：

①如果这个Bean已经实现了BeanNameAware接口，会调用它实现的setBeanName(String beanId)方法，此处传递的就是Spring配置文件中Bean的id值；

②如果这个Bean已经实现了BeanFactoryAware接口，会调用它实现的setBeanFactory()方法，传递的是Spring工厂自身。

③如果这个Bean已经实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文；

（4）BeanPostProcessor：

如果想对Bean进行一些自定义的处理，那么可以让Bean实现了BeanPostProcessor接口，那将会调用postProcessBeforeInitialization(Object obj, String s)方法。由于这个方法是在Bean初始化结束时调用的，所以可以被应用于内存或缓存技术；

（5）InitializingBean 与 init-method：

如果Bean在Spring配置文件中配置了 init-method 属性，则会自动调用其配置的初始化方法。

（6）如果这个Bean实现了BeanPostProcessor接口，将会调用postProcessAfterInitialization(Object obj, String s)方法；



（7）DisposableBean：

当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean这个接口，会调用其实现的destroy()方法；

（8）destroy-method：

最后，如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其配置的销毁方法。

### 5.5.20. 解释Spring支持的几种bean的作用域。

Spring容器中的bean可以分为5个范围：

（1）singleton：默认，每个容器中只有一个bean的实例，单例的模式由BeanFactory自身来维护。

（2）prototype：为每一个bean请求提供一个实例。

（3）request：为每一个网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收。

（4）session：与request范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效。

（5）global-session：全局作用域，global-session和Portlet应用相关。当你的应用部署在Portlet容器中工作时，它包含很多portlet。如果你想要声明让所有的portlet共用全局的存储变量的话，那么这全局变量需要存储在global-session中。全局作用域与Servlet中的session作用域效果相同。

### 5.5.21. Spring框架中的单例Beans是线程安全的么？

Spring框架并没有对单例bean进行任何多线程的封装处理。关于单例bean的线程安全和并发问题需要开发者自行去搞定。但实际上，大部分的Spring bean并没有可变的状态(比如Serview类和DAO类)，所以在某种程度上说Spring的单例bean是线程安全的。如果你的bean有多种状态的话（比如 View Model 对象），就需要自行保证线程安全。最浅显的解决办法就是将多态bean的作用域由“singleton”变更为“prototype”。

### 5.5.22. Spring如何处理线程并发问题？

在一般情况下，只有无状态的Bean才可以在多线程环境下共享，在Spring中，绝大部分Bean都可以声明为singleton作用域，因为Spring对一些Bean中非线程安全状态采用ThreadLocal进行处理，解决线程安全问题。

ThreadLocal和线程同步机制都是为了解决多线程中相同变量的访问冲突问题。同步机制采用了“时间换空间”的方式，仅提供一份变量，不同的线程在访问前需要获取锁，没获得锁的线程则需要排队。而ThreadLocal采用了“空间换时间”的方式。

ThreadLocal会为每一个线程提供一个独立的变量副本，从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进ThreadLocal。

### 5.5.23. Spring基于xml注入bean的几种方式：

（1）Set方法注入；

（2）构造器注入：①通过index设置参数的位置；②通过type设置参数类型；

（3）静态工厂注入；

（4）实例工厂；

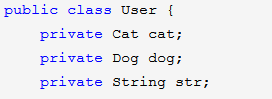
### 5.5.24. Spring的自动装配：

在spring中，对象无需自己查找或创建与其关联的其他对象，由容器负责把需要相互协作的对象引用赋予各个对象，使用autowire来配置自动装载模式。

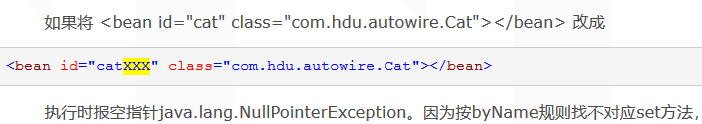
在Spring框架xml配置中共有5种自动装配：

（1）no：默认的方式是不进行自动装配的，通过手工设置ref属性来进行装配bean。

（2）byName：通过bean的名称进行自动装配，如果一个bean的 property 与另一bean 的name 相同，就进行自动装配。





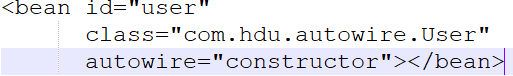




1. byType：通过参数的数据类型进行自动装配。



1. constructor：利用构造函数进行装配，并且构造函数的参数通过byType进行装配。



（5）autodetect：自动探测，如果有构造方法，通过 construct的方式自动装配，否则使用 byType的方式自动装配。

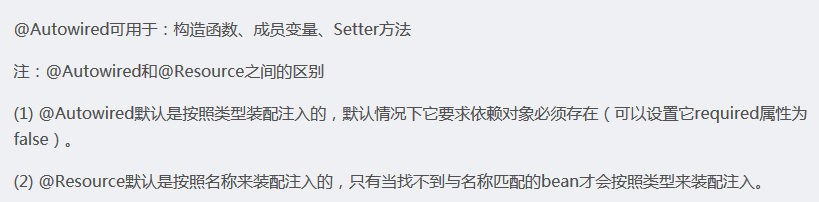
基于注解的方式：

使用@Autowired注解来自动装配指定的bean。在使用@Autowired注解之前需要在Spring配置文件进行配置，<context:annotation-config />。在启动spring IoC时，容器自动装载了一个AutowiredAnnotationBeanPostProcessor后置处理器，当容器扫描到@Autowied、@Resource或@Inject时，就会在IoC容器自动查找需要的bean，并装配给该对象的属性。在使用@Autowired时，首先在容器中查询对应类型的bean：

如果查询结果刚好为一个，就将该bean装配给@Autowired指定的数据；

如果查询的结果不止一个，那么@Autowired会根据名称来查找；

如果上述查找的结果为空，那么会抛出异常。解决方法时，使用required=false。



### 5.5.25. Spring框架中有哪些不同类型的事件？

Spring 提供了以下5种标准的事件：

（1）上下文更新事件（ContextRefreshedEvent）：在调用ConfigurableApplicationContext 接口中的refresh()方法时被触发。

（2）上下文开始事件（ContextStartedEvent）：当容器调用ConfigurableApplicationContext的Start()方法开始/重新开始容器时触发该事件。

（3）上下文停止事件（ContextStoppedEvent）：当容器调用ConfigurableApplicationContext的Stop()方法停止容器时触发该事件。

（4）上下文关闭事件（ContextClosedEvent）：当ApplicationContext被关闭时触发该事件。容器被关闭时，其管理的所有单例Bean都被销毁。

（5）请求处理事件（RequestHandledEvent）：在Web应用中，当一个http请求（request）结束触发该事件。

如果一个bean实现了ApplicationListener接口，当一个ApplicationEvent 被发布以后，bean会自动被通知。

### 5.5.26. 解释一下Spring AOP里面的几个名词：

（1）切面（Aspect）：被抽取的公共模块，可能会横切多个对象。 在Spring AOP中，切面可以使用通用类（基于模式的风格） 或者在普通类中以 @AspectJ 注解来实现。

（2）连接点（Join point）：指方法，在Spring AOP中，一个连接点 总是 代表一个方法的执行。

（3）通知（Advice）：在切面的某个特定的连接点（Join point）上执行的动作。通知有各种类型，其中包括“around”、“before”和“after”等通知。许多AOP框架，包括Spring，都是以拦截器做通知模型， 并维护一个以连接点为中心的拦截器链。

（4）切入点（Pointcut）：切入点是指 我们要对哪些Join point进行拦截的定义。通过切入点表达式，指定拦截的方法，比如指定拦截add\*、search\*。

（5）引入（Introduction）：（也被称为内部类型声明（inter-type declaration））。声明额外的方法或者某个类型的字段。Spring允许引入新的接口（以及一个对应的实现）到任何被代理的对象。例如，你可以使用一个引入来使bean实现 IsModified 接口，以便简化缓存机制。

（6）目标对象（Target Object）： 被一个或者多个切面（aspect）所通知（advise）的对象。也有人把它叫做 被通知（adviced） 对象。 既然Spring AOP是通过运行时代理实现的，这个对象永远是一个 被代理（proxied） 对象。

（7）织入（Weaving）：指把增强应用到目标对象来创建新的代理对象的过程。Spring是在运行时完成织入。

### 5.5.27. Spring通知有哪些类型？

（1）前置通知（Before advice）：在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。

（2）返回后通知（After returning advice）：在某连接点（join point）正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。

（3）抛出异常后通知（After throwing advice）：在方法抛出异常退出时执行的通知。

（4）后通知（After (finally) advice）：当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。

（5）环绕通知（Around Advice）：包围一个连接点（join point）的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。 环绕通知是最常用的一种通知类型。大部分基于拦截的AOP框架，例如Nanning和JBoss4，都只提供环绕通知。

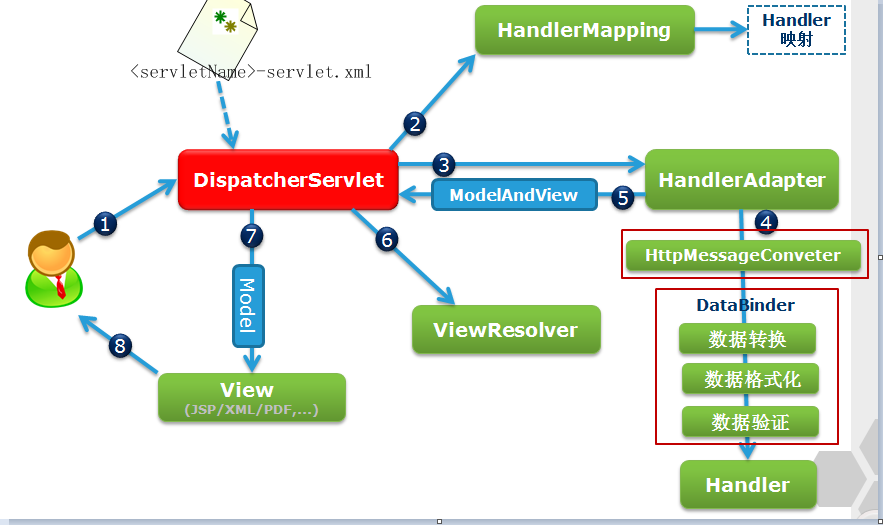


## 5.6. SpringMVC

### 5.6.1. 什么是Spring MVC ？简单介绍下你对springMVC的理解?

Spring MVC是一个基于Java的实现了MVC设计模式的请求驱动类型的轻量级Web框架，通过把Model，View，Controller分离，将web层进行职责解耦，把复杂的web应用分成逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

### 5.6.2. 简单讲一下SpringMVC的执行流程？



1. 用户向服务器发送请求，请求被Spring 前端控制Servelt DispatcherServlet捕获；

2. DispatcherServlet对请求URL进行解析，得到请求资源标识符（URI）。然后根据该URI，调用HandlerMapping获得该Handler配置的所有相关的对象（包括Handler对象以及Handler对象对应的拦截器），最后以HandlerExecutionChain对象的形式返回；

3. DispatcherServlet 根据获得的Handler，选择一个合适的HandlerAdapter。（**附注**：如果成功获得HandlerAdapter后，此时将开始执行拦截器的preHandler(...)方法）

 4.  提取Request中的模型数据，填充Handler入参，开始执行Handler（Controller)。 在填充Handler的入参过程中，根据你的配置，Spring将帮你做一些额外的工作：

  HttpMessageConveter： 将请求消息（如Json、xml等数据）转换成一个对象，将对象转换为指定的响应信息

      数据转换：对请求消息进行数据转换。如String转换成Integer、Double等

数据根式化：对请求消息进行数据格式化。 如将字符串转换成格式化数字或格式化日期等

      数据验证： 验证数据的有效性（长度、格式等），验证结果存储到BindingResult或Error中

5.  Handler执行完成后，向DispatcherServlet 返回一个ModelAndView对象； 6.  根据返回的ModelAndView，选择一个适合的ViewResolver（必须是已经注册到Spring容器中的ViewResolver)返回给DispatcherServlet ；

7. ViewResolver 结合Model和View，来渲染视图

8. 将渲染结果返回给客户端。

面试：

1. 用户向服务器发送请求，请求被Spring 前端控制Servelt DispatcherServlet捕获(捕获)

2、 DispatcherServlet对请求URL进行解析，得到请求资源标识符（URI）。然后根据该URI，调用HandlerMapping获得该Handler配置的所有相关的对象（包括Handler对象以及Handler对象对应的拦截器），最后以HandlerExecutionChain对象的形式返回；(查找handler)

3、 DispatcherServlet 根据获得的Handler，选择一个合适的HandlerAdapter。 提取Request中的模型数据，填充Handler入参，开始执行Handler（Controller), Handler执行完成后，向DispatcherServlet 返回一个ModelAndView对象(执行handler)

4、DispatcherServlet 根据返回的ModelAndView，选择一个适合的ViewResolver（必须是已经注册到Spring容器中的ViewResolver) (选择ViewResolver)

5、通过ViewResolver 结合Model和View，来渲染视图,DispatcherServlet 将渲染结果返回给客户端。（渲染返回）

快速记忆技巧：

核心控制器捕获请求、查找Handler、执行Handler、选择ViewResolver,通过ViewResolver渲染视图并返回

### 5.6.3. Springmvc的优点:

（1）可以支持各种视图技术,而不仅仅局限于JSP；

（2）与Spring框架集成（如IoC容器、AOP等）；

（3）清晰的角色分配：前端控制器(dispatcherServlet) , 请求到处理器映射（handlerMapping), 处理器适配器（HandlerAdapter), 视图解析器（ViewResolver）。

（4） 支持各种请求资源的映射策略。

### 5.6.4. Spring MVC的主要组件？

（1）前端控制器 DispatcherServlet（不需要程序员开发）

作用：接收请求、响应结果，相当于转发器，有了DispatcherServlet 就减少了其它组件之间的耦合度。

（2）处理器映射器HandlerMapping（不需要程序员开发）

作用：根据请求的URL来查找Handler

（3）处理器适配器HandlerAdapter

注意：在编写Handler的时候要按照HandlerAdapter要求的规则去编写，这样适配器HandlerAdapter才可以正确的去执行Handler。

（4）处理器Handler（需要程序员开发）

（5）视图解析器 ViewResolver（不需要程序员开发）

作用：进行视图的解析，根据视图逻辑名解析成真正的视图（view）

（6）视图View（需要程序员开发jsp）

View是一个接口， 它的实现类支持不同的视图类型（jsp，freemarker，pdf等等）

### 5.6.5. 说一下struts2和springMVC有什么不同？

目前企业中使用SpringMvc的比例已经远远超过Struts2,那么两者到底有什么区别，是很多初学者比较关注的问题，下面我们就来对SpringMvc和Struts2进行各方面的比较:

1. 核 心控制器（前端控制器、预处理控制器）：对于使用过mvc框架的人来说这个词应该不会陌生，核心控制器的主要用途是处理所有的请求，然后对那些特殊的请求 （控制器）统一的进行处理(字符编码、文件上传、参数接受、异常处理等等),spring mvc核心控制器是Servlet，而Struts2是Filter。

2.控制器实例：Spring Mvc会比Struts快一些（理论上）。Spring Mvc是基于方法设计，而Sturts是基于对象，每次发一次请求都会实例一个action，每个action都会被注入 属性，而Spring更像Servlet一样，只有一个实例，每次请求执行对应的方法即可(注意：由于是单例实例，所以应当避免全局变量的修改，这样会产生线程安全问题)。

3. 管理方式：大部分的公司的核心架构中，就会使用到spring,而spring mvc又是spring中的一个模块，所以spring对于spring mvc的控制器管理更加简单方便，而且提供了全 注解方式进行管理，各种功能的注解都比较全面，使用简单，而struts2需要采用XML很多的配置参数来管理（虽然也可以采用注解，但是几乎没有公司那 样使用）。

4.参数传递：Struts2中自身提供多种参数接受，其实都是通过（ValueStack）进行传递和赋值，而SpringMvc是通过方法的参数进行接收。

5.学习难度：Struts更加很多新的技术点，比如拦截器、值栈及OGNL表达式，学习成本较高，springmvc 比较简单，很较少的时间都能上手。

6.intercepter（拦截） 的实现机制：struts有以自己的interceptor机制，spring mvc用的是独立的AOP方式。这样导致struts的配置文件量还是比spring mvc大，虽然struts的配置能继承，所以我觉得论使用上来讲，spring mvc使用更加简洁，开发效率Spring MVC确实比struts2高。spring mvc是方法级别的拦截，一个方法对应一个request上下文，而方法同时又跟一个url对应，所以说从架构本身上spring3 mvc就容易实现restful url。struts2是类级别的拦截，一个类对应一个request上下文；实现restful url要费劲，因为struts2 action的一个方法可以对应一个url；而其类属性却被所有方法共享，这也就无法用注解或其他方式标识其所属方法了。spring3 mvc的方法之间基本上独立的，独享request response数据，请求数据通过参数获取，处理结果通过ModelMap交回给框架方法之间不共享变量，而struts2搞的就比较乱，虽然方法之间 也是独立的，但其所有Action变量是共享的，这不会影响程序运行，却给我们编码，读程序时带来麻烦。

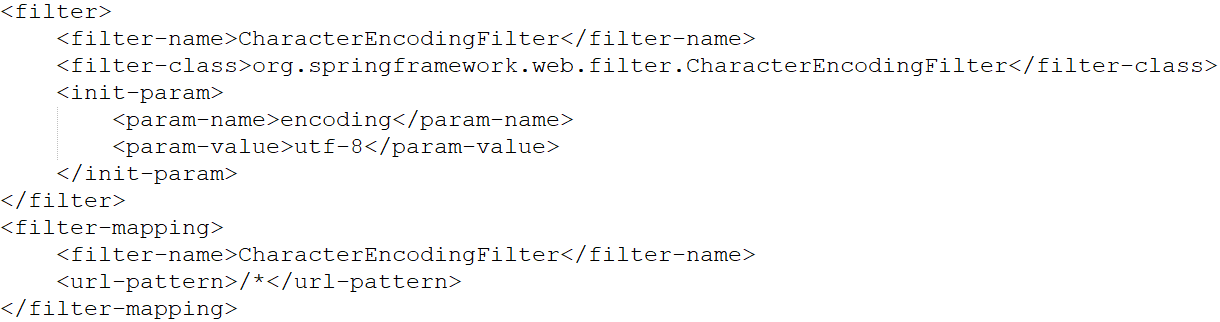
1. spring mvc处理ajax请求,直接通过返回数据，方法中使用注解@ResponseBody，spring mvc自动帮我们对象转换为JSON数据。而struts2是通过插件的方式进行处理

在SpringMVC流行起来之前，Struts2在MVC框架中占核心地位，随着SpringMVC的出现，SpringMVC慢慢的取代struts2,但是很多企业都是原来搭建的框架，使用Struts2较多。

### 5.6.6. 如何解决POST请求中文乱码问题，GET的又如何处理呢？

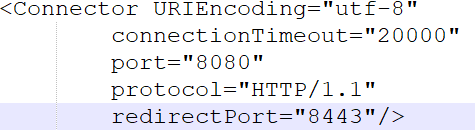
（1）解决post请求乱码问题：

在web.xml中配置一个CharacterEncodingFilter过滤器，设置成utf-8；



（2）get请求中文参数出现乱码解决方法有两个：

①修改tomcat配置文件添加编码与工程编码一致，如下：



②另外一种方法对参数进行重新编码：

String userName = new String(request.getParamter("userName")

.getBytes("ISO8859-1"),"utf-8")

ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码。

### 5.6.7. Spring MVC的异常处理 ？

可以将异常抛给Spring框架，由Spring框架来处理；我们只需要配置简单的异常处理器，在异常处理器中添视图页面即可。

### 5.6.8.SpringMvc 的控制器是不是单例模式，如果是，有什么问题，怎么解决？

是单例模式,所以在多线程访问的时候有线程安全问题,不要用同步,会影响性能的,解决方案是在控制器里面不能写字段。

### 5.6.9. SpringMVC常用的注解有哪些？

@RequestMapping：用于处理请求 url 映射的注解，可用于类或方法上。用于类上，则表示类中的所有响应请求的方法都是以该地址作为父路径。

@RequestBody：注解实现接收http请求的json数据，将json转换为java对象。

@ResponseBody：注解实现将conreoller方法返回对象转化为json对象响应给客户。

### 5.6.10.SpringMvc中控制器的注解？有没有别的注解可以替代？

用@Conntroller注解,表示是表现层,不能用别的注解代替。

### 5.6.11. 如果在拦截请求中，我想拦截get方式提交的方法,怎么配置？

可以在@RequestMapping注解里面加上method=RequestMethod.GET。

### 5.6.12. 怎样在方法里面得到Request,或者Session？

直接在方法的形参中声明request,SpringMvc就自动把request对象传入。

### 5.6.13. 如果想在拦截的方法里面得到从前台传入的参数,怎么得到？

直接在形参里面声明这个参数就可以,但必须名字和传过来的参数一样。

### 5.6.14.前台多个参数，这些参数都是一个对象，快速得到对象？

直接在方法中声明这个对象，SpringMvc就自动把属性赋值到这个对象里面

### 5.6.15.SpringMvc中函数的返回值？

String，ModelAndView，List，Set 等  
一般String，Ajax请求，返回一个List集合

### 5.6.16.SpringMvc中的转发和重定向?

（1）转发：在返回值前面加"forward:"，譬如return："forward:user.do?name=method4"

（2）重定向：在返回值前面加"redirect:"，譬如return："redirect:http://www.baidu.com"

### 5.6.17.SpringMvc和Ajax之间的相互调用？

通过Jackson框架就可以把Java里面的对象直接转化成Js可以识别的Json对象。具体步骤如下 ：

（1）加入Jackson.jar

（2）在配置文件中配置json的映射

（3）在接受Ajax方法里面可以直接返回Object,List等,但方法前面要加上@ResponseBody注解。

### 5.6.18. SpringMvc用什么对象从后台向前台传递数据的？

通过ModelMap对象,可以在这个对象里面调用put方法,把对象加到里面,前台就可以通过el表达式拿到。

### 5.6.19. 怎么样把ModelMap里面的数据放入Session里面？

可以在类上面加上@SessionAttributes注解,里面包含的字符串就是要放入session里面的key。

### 5.6.20. SpringMvc里面拦截器是怎么写的：

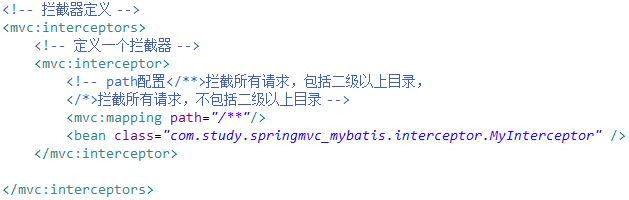
有两种写法,一种是实现HandlerInterceptor接口，另外一种是继承适配器类，接着在接口方法当中，实现处理逻辑；然后在SpringMvc的配置文件中配置拦截器即可：

第一种：





第二种



### 5.6.21. 注解原理：

注解本质是一个继承了Annotation的特殊接口，其具体实现类是Java运行时生成的动态代理类。我们通过反射获取注解时，返回的是Java运行时生成的动态代理对象。通过代理对象调用自定义注解的方法，会最终调用AnnotationInvocationHandler的invoke方法。该方法会从memberValues这个Map中索引出对应的值。而memberValues的来源是Java常量池。

## 5.7. Mybatis

### 5.7.1. Ibatis和Mybatis？

Ibatis：2010年，apache的Ibatis框架停止更新，并移交给了google团队，同时更名为MyBatis。从2010年后Ibatis在没更新过，彻底变成了一个孤儿框架。一个没人维护的框架注定被mybatis拍在沙滩上。

Mybatis：Ibatis的升级版本。

### 5.7.2.什么是Mybatis的接口绑定，有什么好处？

Mybatis实现了DAO接口与xml映射文件的绑定，自动为我们生成接口的具体实现，使用起来变得更加省事和方便。

### 5.7.3.什么情况用注解，什么情况用xml绑定？

注解使用情况：Sql语句简单时

xml绑定使用情况：xml绑定 (@RequestMap用来绑定xml文件)

### 5.7.4.Mybatis在核心处理类叫什么?

SqlSession

### 5.7.5.查询表名和返回实体Bean对象不一致，如何处理？

映射键值对即可



column：数据库中表的列名

property：实体Bean中的属性名

### 5.7.6.Mybatis的好处？

把Sql语句从Java中独立出来。

封装了底层的JDBC，API的调用，并且能够将结果集自动转换成JavaBean对象，简化了Java数据库编程的重复工作。

自己编写Sql语句，更加的灵活。

入参无需用对象封装（或者map封装）,使用@Param注解

### 5.7.7.Mybatis配置一对多？

<collection property="topicComment" column="id" ofType="com.tmf.bbs.pojo.Comment" select="selectComment"/>

property：属性名

column：共同列

ofType：集合中元素的类型

select：要连接的查询

### 5.7.8.Mybatis配置一对一？

<association property="topicType" select="selectType" column="topics\_type\_id" javaType="com.tmf.bbs.pojo.Type"/>

property：属性名

select：要连接的查询

column：共同列

javaType：集合中元素的类型

### 5.7.9.${} 和 #{}的区别？

${}：简单字符串替换，把${}直接替换成变量的值，不做任何转换，这种是取值以后再去编译SQL语句。

#{}：预编译处理，sql中的#{}替换成？，补全预编译语句，有效的防止Sql语句注入，这种取值是编译好SQL语句再取值。

总结：一般用#{}来进行列的代替

### 5.7.10.获取上一次自动生成的主键值？

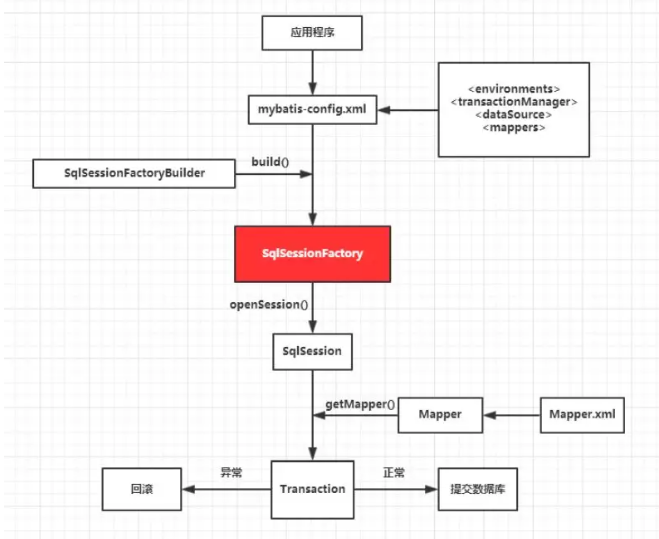


### 5.7.11.Mybatis如何分页，分页原理？

RowBounds对象分页

在Sql内直接书写，带有物理分页

### 5.7.12.Mybatis工作原理？



通过SqlSessionFactoryBuilder从mybatis-config.xml配置文件中构建出SqlSessionFactory。

SqlSessionFactory开启一个SqlSession，通过SqlSession实例获得Mapper对象并且运行Mapper映射的Sql语句。

完成数据库的CRUD操作和事务提交，关闭SqlSession。

## 5.8. Hibernate

### 5.8.1.iBatis(mybatis)与Hibernate有什么不同?

相同点：

都是java中orm框架、屏蔽jdbc api的底层访问细节，使用我们不用与jdbc api打交道，就可以完成对数据库的持久化操作。jdbc api编程流程固定，还将sql语句与java代码混杂在了一起，经常需要拼凑sql语句，细节很繁琐。

ibatis的好处：屏蔽jdbc api的底层访问细节；将sql语句与java代码进行分离;提供了将结果集自动封装称为实体对象和对象的集合的功能.queryForList返回对象集合，用queryForObject返回单个对象；提供了自动将实体对象的属性传递给sql语句的参数。

Hibername的好处：Hibernate是一个全自动的orm映射工具，它可以自动生成sql语句，并执行并返回java结果。

不同点：

1、hibernate要比ibatis功能强大很多。因为hibernate自动生成sql语句。

2、ibatis需要我们自己在xml配置文件中写sql语句，hibernate我们无法直接控制该语句，我们就无法去写特定的高效率的sql。对于一些不太复杂的sql查询，hibernate可以很好帮我们完成，但是，对于特别复杂的查询，hibernate就很难适应了，这时候用ibatis就是不错的选择，因为ibatis还是由我们自己写sql语句。

ibatis可以出来复杂语句，而hibernate不能。

3、ibatis要比hibernate简单的多。ibatis是面向sql的，不会考虑对象间一些复杂的映射关系。

### 5.8.2. Hibernate映射对象的状态

临时状态/瞬时状态(transient)：刚刚用new语句创建，没有被持久化

不处于session中(没有使用session的方法去操作临时对象)。该对象称为临时对象

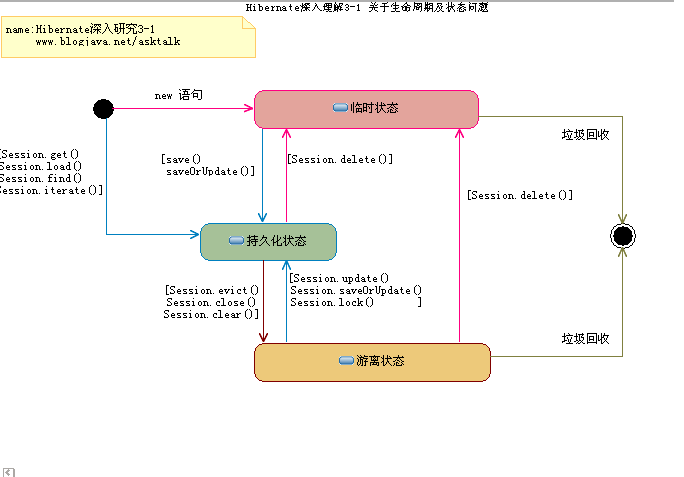
持久化状态/托管状态(persistent)：已经被持久化，加入到session的缓存中。session是没有关闭该状态的对象为持久化对象。

游离状态/脱管状态(detached)：已经被持久化，但不处于session中。

该状态的对象为游离对象。

删除状态(removed)：对象有关联的ID，并且在Session管理下，但是已经被计划(事务提交的时候,commit())删除。如果没有事务就不能删除

相互转换



### 5.8.3. 介绍一下Hibernate的缓存?

一、why（为什么要用Hibernate缓存？）

Hibernate是一个持久层框架，经常访问物理数据库。

为了降低应用程序对物理数据源访问的频次，从而提高应用程序的运行性能。

缓存内的数据是对物理数据源中的数据的复制，应用程序在运行时从缓存读写数据，在特定的时刻或事件会同步缓存和物理数据源的数据。

为了提供访问速度，把磁盘或数据库访问变成内存访问。

二、what（Hibernate缓存原理是怎样的？）Hibernate缓存包括两大类：Hibernate一级缓存和Hibernate二级缓存。

1.Hibernate一级缓存又称为“Session的缓存”。

Session缓存内置不能被卸载，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。

Hibernate提供了org.hibernate.cache.CacheProvider接口,它充当缓存插件与Hibernate之间的适配器。

面试：

Hibernate中的缓存分一级缓存和二级缓存。

一级缓存就是Session级别的缓存，在事务范围内有效是,内置的不能被卸载。二级缓存是SesionFactory级别的缓存，从应用启动到应用结束有效。是可选的，默认没有二级缓存，需要手动开启。

保存数据库后，在内存中保存一份，如果更新了数据库就要同步更新。

什么样的数据适合存放到第二级缓存中？

1. 很少被修改的数据 　帖子的最后回复时间
2. 经常被查询的数据 电商的地点

2) 不是很重要的数据，允许出现偶尔并发的数据

3) 不会被并发访问的数据

4) 常量数据

扩展：hibernate的二级缓存默认是不支持分布式缓存的。使用memcahe,redis等中央缓存来代替二级缓存。

## 5.9. 什么是ORM?

对象关系映射（Object Relational Mapping，简称ORM）模式是一种为了解决面向**对象**与**关系**数据库存在的互不匹配的现象的技术。简单的说，ORM是通过使用描述对象和数据库之间映射的元数据，将程序中的对象自动持久化到关系数据库中。那么，到底如何实现持久化呢？一种简单的方案是采用硬编码方式(jdbc操作sql方式)，为每一种可能的数据库访问操作提供单独的方法。

这种方案存在以下不足：

1.持久化层缺乏弹性。一旦出现业务需求的变更，就必须修改持久化层的接口

2.持久化层同时与域模型与关系数据库模型绑定，不管域模型还是关系数据库模型发生变化，都要修改持久化曾的相关程序代码，增加了软件的维护难度。

ORM提供了实现持久化层的另一种模式，它采用映射元数据来描述对象关系的映射，使得ORM中间件能在任何一个应用的业务逻辑层和数据库层之间充当桥梁。Java典型的ORM框架有:Hibernate,ibatis(mybatis),speedframework。

ORM的方法论基于三个核心原则：

　　 简单：以最基本的形式建模数据。

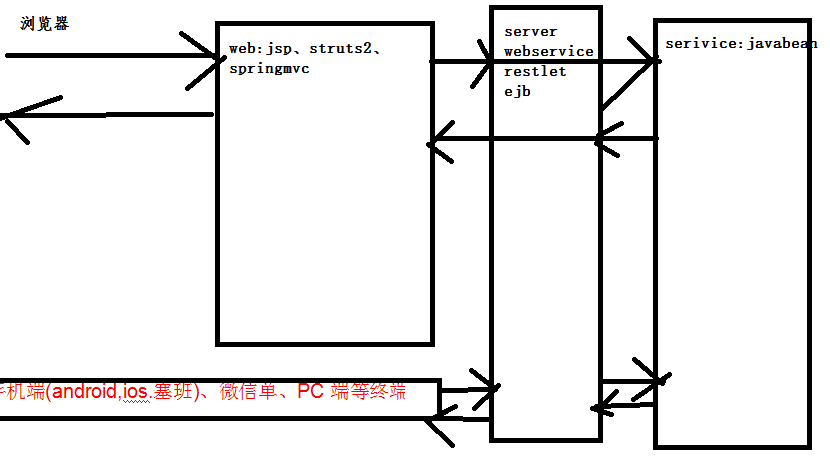
　　 传达性：数据库结构被任何人都能理解的语言文档化。

　　 精确性：基于数据模型创建正确标准化了的结构。

## 5.10. 简单讲一下webservice使用的场景？

webservice是一个SOA（面向服务的编程）的架构，它是不依赖于语言，不依赖于平台，可以实现不同的语言间的相互调用，通过Internet进行基于Http协议的网络应用间的交互。

1. 异构系统(不同语言)的整合
2. 不同客户端的整合 浏览器、手机端(android,ios.塞班)、微信单、PC端等终端来访问



3、实实在在的列子：

天气预报：可以通过实现webservice客户端调用远程天气服务实现的。

单点登录：一个服务实现所有系统的登录

## 5.11. 简单介绍一下activiti?

Activiti是一个业务流程管理(BPM)和工作流系统，适用于开发人员和系统管理员。其核心是超快速，稳定的BPMN2流程引擎。它易于与 Spring集成使用。

主要要在OA中，把线下流程放到线上。 把现实生活中一些流程固话定义到系统中，然后通过输入表单数据完成业务。

他可用在OA系统的流程管理中：

请假流程 小于三天，一级主管审批，大于三天二级才能审批。

报销流程 1000 2000 3000>....

如果你想找专门这方面的工作，要下去复习。

# 6.高级部分

## 6.1. 有没有用过linux?你都用它来做什么？

Linux是一个长时间运行比较稳定的操作系统，所以我们一般会拿它作为服务器(web,db,app等)。

Linux本身具有C的编译环境、我们的一些软件是没有软件包(redis、nginx等)的，需要在Linux的C编译环境编译得到软件包.

## 6.2. 说一下linux下面的一下常用命令？

常用：

Pwd 获取当前路径

Cd 跳转到目录

Su -u 切换到管理员

Ls ls 列举目录

文件操作命令：

文件

tail 查看

rm -rf

vi

文件夹

mkdir

rm -r

## 6.3. 你是使用什么来连接远程的Linux服务器的？

需要依赖于Linux服务器安装ssh服务端，一般这个ssh服务的端口22.

需要依赖于Linux服务器安装sftp服务端，一般这个sftp服务的端口25.

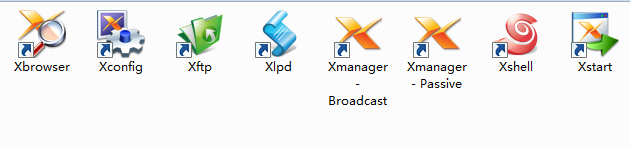
使用ssh客户端连接linux服务器，就有点儿像windows下面的远程连接。但是linux通过ssh连接上以后是没有图形界面，全是命令行。使用的ssh客户端软件：Putty、Xshell

使用sftp客户端来连接sftp服务端，来上传和下载文件.(上传安装包，修改了配置文件上传。)。使用的sftp客户端软件Winscp、xftp

企业中常用的两种组合:

putty+winscp

Xshell+xftp=xmanager



面试：使用xshell、putty等ssh客户端来连接服务器，使用xftp、winscp等sftp客户端来上传和下载文件。连接和上传、下载必须依赖于服务器的ssh、sftp服务，也就是linux服务器需要启动这两个服务。

## 6.4. 有没有使用过云主机？

使用过，在原来的公司，我们没有使用自己的服务器，而是租用阿里的云主机。

没有使用过，但是有所了解。

云主机就是一些云服务运营商(阿里、华为、西部数码、新浪等)，提供的远程的服务器功能，我们开发者或者企业只需按需付费就可以租用对应的服务器。

使用ssh和sftp来进行操作。

## 6.5. 有没有做过数据库优化方面的事情?

做过mysql数据库的优化、其他数据库类似

定位：查找、定位慢查询

优化手段：

* 1. 创建索引:创建合适的索引，我们就可以现在索引中查询，查询到以后直接找对应的记录。
  2. 分表 ：当一张表的数据比较多或者一张表的某些字段的值比较多并且很少使用时，采用水平分表和垂直分表来优化
  3. 读写分离：当一台服务器不能满足需求时，采用读写分离的方式进行集群。
  4. 缓存:使用redis来进行缓存
  5. 一些常用优化技巧

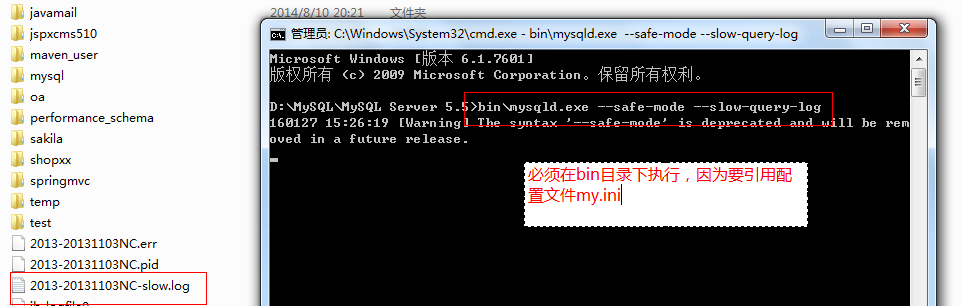
## 6.6. 查找慢查询并定位慢查询？

在项目自验项目转测试之前，在启动mysql数据库时开启慢查询，并且把执行慢的语句写到日志中，在运行一定时间后。通过查看日志找到慢查询语句。

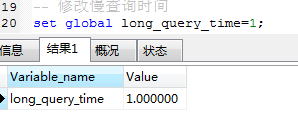
要找出项目中的慢Sql时

1、关闭数据库服务器(关闭服务)

2、把慢查询记录到日志中

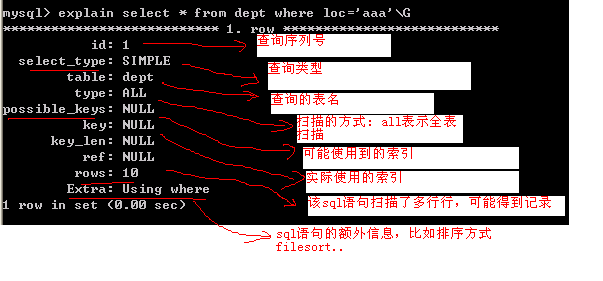


1. 设置慢查询时间



4、找出日志中的慢查询SQL

使用explain 慢查询语句，来详细分析语句的问题.



## 6.7. 数据库优化之遵循范式？

数据库表设计时需要遵循方式

表的范式，是首先符合1NF, 才能满足2NF , 进一步满足3NF

1NF: 即表的列的具有原子性,不可再分解，即列的信息，不能分解.只要数据库是关系型数据库(mysql/oracle/db2/sysbase/sql server)，就自动的满足1NF.关系型数据库中是不允许分割列的。

2NF:表中的记录是唯一的.通常我们设计一个主键来实现

3NF:即表中不要有冗余数据, 就是说，表的信息，如果能够被推导出来，就不应该单独的设计一个字段来存放.(外键)

反3NF :没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是： 在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余**。** 订单和订单项、相册浏览次数和照片的浏览次数

## 6.8. 选择合适的存储引擎

在开发中，我们经常使用的存储引擎 myisam / innodb/ memory

MyISAM存储引擎

如果表对事务要求不高，同时是以查询和添加为主的，我们考虑使用myisam存储引擎.比如 bbs 中的 发帖表，回复表.

INNODB存储引擎:

对事务要求高，保存的数据都是重要数据，我们建议使用INNODB,比如订单表，账号表.

Memory 存储

我们数据变化频繁，不需要入库，同时又频繁的查询和修改，我们考虑使用memory, 速度极快.

问 MyISAM 和 INNODB的区别(主要)

1. 事务安全 myisam不支持事务而innodb支持

2. 查询和添加速度 myisam不用支持事务就不用考虑同步锁，查找和添加和添加的速度快

3. 支持全文索引 myisam支持innodb不支持

4. 锁机制 myisam支持表锁而innodb支持行锁(事务)

5. 外键 MyISAM 不支持外键， INNODB支持外键. (通常不设置外键，通常是在程序中保证数据的一致)



## 6.9. 数据库优化之创建合适的索引?

**索引（Index）是帮助DBMS高效获取数据的数据结构。**

分类：普通索引/唯一索引/主键索引/全文索引

普通索引:允许重复的值出现

唯一索引:除了不能有重复的记录外，其它和普通索引一样(用户名、用户身份证、email,tel)

主键索引：是随着设定主键而创建的，也就是把某个列设为主键的时候，数据库就会給改列创建索引。这就是主键索引.唯一且没有null值

全文索引:用来对表中的文本域(char，varchar，text)进行索引， 全文索引针对MyIsam

explain select \* from articles where match(title,body) against(‘database’);【会使用全文索引】

## 6.10. 索引使用小技巧？

索引弊端

1.占用磁盘空间。

2.对dml(插入、修改、删除)操作有影响，变慢。

使用场景：

a: 肯定在where条件经常使用,如果不做查询就没有意义

b: 该字段的内容不是唯一的几个值(sex)

c: 字段内容不是频繁变化.

具体技巧：

1. 对于创建的多列索引（复合索引），不是使用的第一部分就不会使用索引。

alter table dept add index my\_ind (dname,loc); // dname 左边的列,loc就是右边的列

explain select \* from dept where dname='aaa'\G 会使用到索引

explain select \* from dept where loc='aaa'\G 就不会使用到索引

2. 对于使用like的查询，查询如果是’%aaa’不会使用到索引而‘aaa%’会使用到索引。

explain select \* from dept where dname like '%aaa'\G不能使用索引

explain select \* from dept where dname like 'aaa%'\G使用索引.

所以在like查询时，‘关键字’的最前面不能使用 % 或者 \_这样的字符.，如果一定要前面有变化的值，则考虑使用 全文索引->sphinx.

1. 如果条件中有or，有条件没有使用索引,即使其中有条件带索引也不会使用。换言之，就是要求使用的所有字段,都必须单独使用时能使用索引.
2. 如果列类型是字符串，那一定要在条件中将数据使用引号引用起来。否则不使用索引。

expain select \* from dept where dname=’111’;

expain select \* from dept where dname=111;（数值自动转字符串）

expain select \* from dept where dname=qqq;报错

也就是，如果列是字符串类型，无论是不是字符串数字就一定要用 ‘’ 把它包括起来.

1. 如果mysql估计使用全表扫描要比使用索引快，则不使用索引。

表里面只有一条记录

## 6.11. 数据库优化之分表？

分表分为水平(按行)分表和垂直(按列)分表

根据经验，Mysql表数据一般达到百万级别，查询效率会很低，容易造成表锁，甚至堆积很多连接，直接挂掉；**水平分表**能够很大程度较少这些压力。

按行数据进行分表。

如果一张表中某个字段值非常多(长文本、二进制等)，而且只有在很少的情况下会查询。这时候就可以把字段多个单独放到一个表，通过外键关联起来。

考试详情，一般我们只关注分数，不关注详情。

水平分表策略：

1.按时间分表

这种分表方式有一定的局限性，当数据有较强的实效性，如微博发送记录、微信消息记录等，这种数据很少有用户会查询几个月前的数据，如就可以按月分表。

2.按区间范围分表

一般在有严格的自增id需求上，如按照user\_id水平分表：

table\_1 user\_id从1~100w

table\_2 user\_id从101~200w

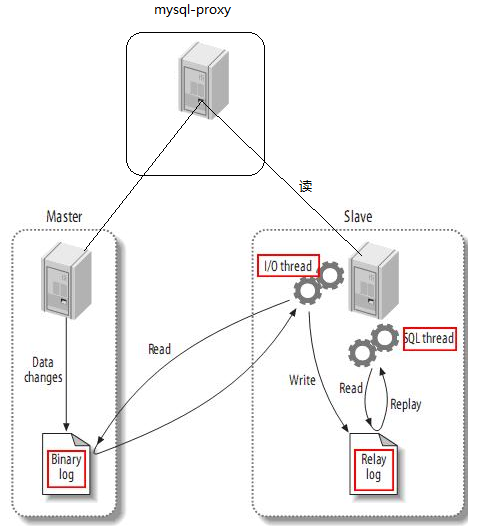
table\_3 user\_id从201~300w

3.hash分表\*\*\*\*\*

通过一个原始目标的ID或者名称通过一定的hash算法计算出数据存储表的表名，然后访问相应的表。

## 6.12. 数据库优化之读写分离

一台数据库支持的最大并发连接数是有限的，如果用户并发访问太多。一台服务器满足不了要求时就可以集群处理。Mysql的集群处理技术最常用的就是读写分离。



主从同步

数据库最终会把数据持久化到磁盘，如果集群必须确保每个数据库服务器的数据是一致的。**能改变数据库数据的操作都往主数据库去写**，**而其他的数据库从主数据库上同步数据。**

读写分离

使用负载均衡来实现写的操作都往主数据去，而读的操作往从服务器去。

## 6.13. 数据库优化之缓存

在持久层(dao)和数据库(db)之间添加一个缓存层，如果用户访问的数据已经缓存起来时，在用户访问时直接从缓存中获取，不用访问数据库。而缓存是在操作内存级，访问速度快。

作用：减少数据库服务器压力，减少访问时间。

Java中常用的缓存有，

1、hibernate的二级缓存。该缓存不能完成分布式缓存。

说明：用户提交数据到A服务器去处理数据，A服务器将数据到数据库服务器中处理完后保存到A服务器中的hibernate的sessionFactory（二级缓存）中。第二次用户获取数据时如果到A服务器可以从缓存中取到数据，如果到B服务器的缓存中是取不到的，只能B服务器先去数据库中把数据保存到缓存中才能取到，这样的话就增加了工作量，达不到缓存的目的。

1. 可以使用redis(memcahe等)来作为中央缓存。对缓存的数据进行集中处理

## 6.14. 语句优化小技巧

DDL优化:

1 、通过禁用索引来提供导入数据性能 。 这个操作主要针对有数据库的表，追加数据

//去除键

alter table test3 DISABLE keys;

//批量插入数据

insert into test3 select \* from test;

//恢复键

alter table test3 ENABLE keys;

2、 关闭唯一校验

set unique\_checks=0 关闭

set unique\_checks=1 开启

3、修改事务提交方式(导入)（变多次提交为一次）

set autocommit=0 关闭

//批量插入

set autocommit=1 开启

DML优化（变多次提交为一次）

insert into test values(1,2);

insert into test values(1,3);

insert into test values(1,4);

//合并多条为一条

insert into test values(1,2),(1,3),(1,4)

DQL优化

Order by优化

1、多用索引排序

1. 普通结果排序（非索引排序）Filesort

group by优化

是使用order by null,取消默认排序

子查询优化

在客户列表找到不在支付列表的客户

#在客户列表找到不在“支付列表”的客户 , 查询没买过东西的客户

explain

select \* from customer where customer\_id not in (select DISTINCT customer\_id from payment); #子查询 -- 这种是基于func外链

explain

select \* from customer c left join payment p on(c.customer\_id=p.customer\_id) where p.customer\_id is null -- 这种是基于“索引”外链

Or优化

在两个独立索引上使用or的性能优于

1、 or两边都是用索引字段做判断，性能好！！

2、 or两边，有一边不用，性能差

3、 如果employee表的name和email这两列是一个复合索引，但是如果是 :name='A' OR email='B' 这种方式，不会用到索引！

limit优化

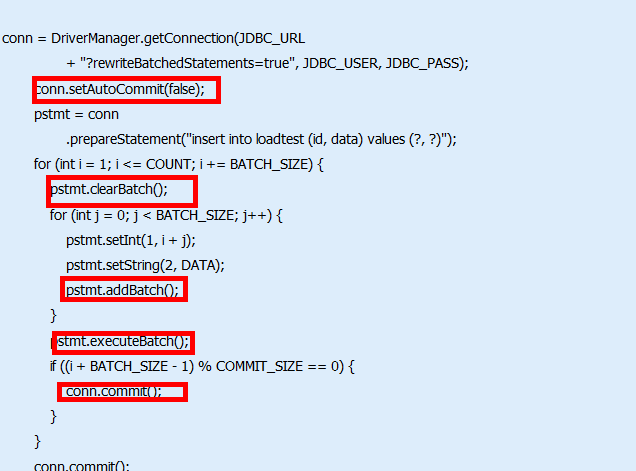
select film\_id,description from film order by title limit 50,5;

select a.film\_id,a.description from film a inner join (select film\_id from film order by title limit 50,5)b on a.film\_id=b.film\_id

## 6.15. jdbc批量插入几百万数据怎么实现？\*\*\*\*\*

1、变多次提交为一次

3、使用批量操作



省出的时间可观。

像这样的批量插入操作能不使用代码操作就不使用，可以使用存储过程来实现。

## 6.16. 有没有使用过redis? Redis是什么

Redis是一个key-value的nosql数据库.先存到内存中，会根据一定的策略持久化到磁盘,即使断电也不会丢失数据。支持的数据类型比较多。

主要用来做缓存数据库的数据和web集群时当做中央缓存存放seesion

## 6.17. Redis和memche的比较?



1、 Redis和Memcache都是将数据存放在内存中，都是内存数据库。不过memcache还可用于缓存其他东西，例如图片、视频等等。

2、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储。

3、虚拟内存--Redis当物理内存用完时，可以将一些很久没用到的value 交换到磁盘

## 6.18. 简单说一下redis的使用场景？

缓存：

把经常需要查询的、很少修改数据，放到读速度很快的空间(内存)，以便下次访问减少时间。减轻压力，减少访问时间.

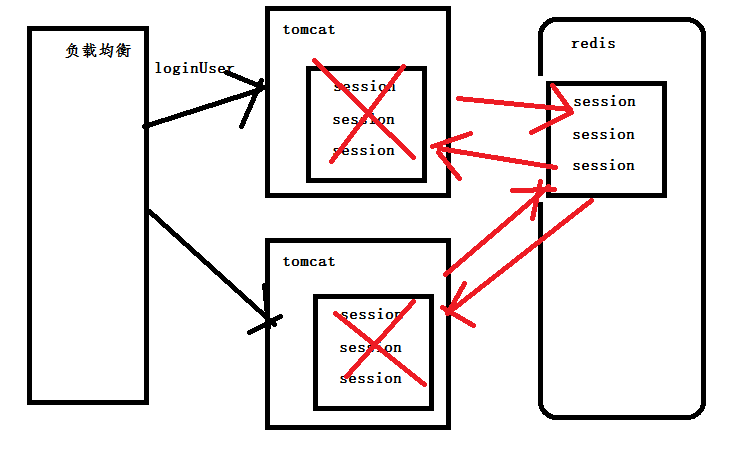
计数器：

redis中的计数器是原子性的内存操作。

可以解决库存溢出问题。进销存、系统库存溢出。

session缓存服务器：

web集群时作为session缓存服务器、缓存队列等



## 6.19. redis对象保存方式？

Json字符串:

需要把对象转换为json字符串，当做字符串处理。直接使用set get来设置或者获取

优点：设置和获取比较简单

缺点：没有提供专门的方法，需要把对象转换为json。(jsonlib)

字节:

需要做序列号，就是把对象序列化为字节保存。

如果是担心JSON转对象会消耗资源的情况，这个问题需要考量几个地方，

第一点：就是使用的JSON转换lib是否就会存在性能问题。

第二点：就是数据的数据量级别，如果是存储百万级的大数据对象，建议采用存储序列化对象方式。如果是少量的数据级对象，或者是数据对象字段不多，还是建议采用JSON转换成String方式。

毕竟redis对存储字符类型这部分优化的非常好。具体采用的方式与方法，还要看你所使用的场景。

## 6.20. Redis数据淘汰机制

在 redis 中，允许用户设置最大使用内存大小 server.maxmemory，在内存限定的情况下是很有用的。譬如，在一台 8G 机子上部署了 4 个 redis 服务点，每一个服务点分配 1.5G 的内存大小，减少内存紧张的情况，由此获取更为稳健的服务。

内存大小有限，需要保存有效的数据?

redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。

redis 提供 6种数据淘汰策略：

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近**最少使用的数据淘汰**

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

**allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰**

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

## 6.21. Java访问Redis

1、使用jedis java客户端来访问redis服务器，有点类似通过jdbc访问mysql一样。

2、当然如果是spring进行集成时，可以使用spring data来访问redis,spring data只是对jedis的二次封装。jdbcTemplate jdbc关系一样

## 6.22. Redis集群

当一台数据无法满足要求，可以使用reids集群来处理，类似于mysql的读写分离。

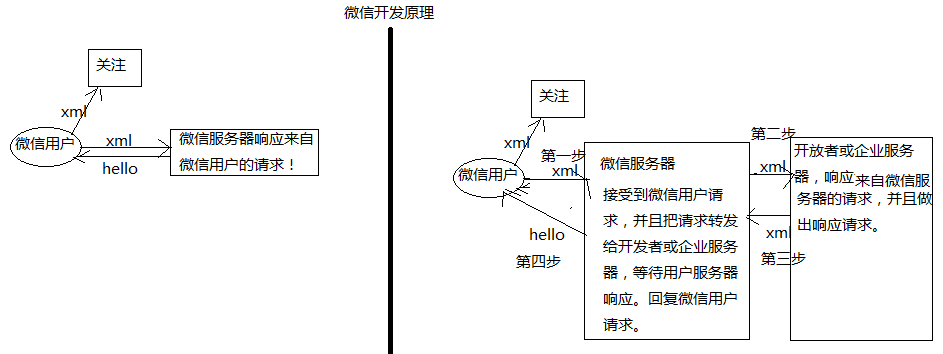
## 6.23. 简单介绍一下微信公共号的分类？

公众号:个人和企业都能申请

服务号：企业才能申请

企业号:企业才能申请

## 6.24. 微信开发原理



微信公众平台开发者，通过接入认证的方式，让我们的服务器能处理来自微信服务器转发的微信用户的请求,处理完成后返回给微信服务器，有微信服务器对用户响应。

## 6.25. 怎么把微信和业务平台绑定？

微信用户和注册用户绑定？

让微信用户也能完成注册用户的功能。

用户注册实体中包含一个微信号的字段，当我进行绑定时就是修改用户的微信号字段。

当然我们在进行菜单跳转到页面后，我们是无法直接获取微信号的。要通过微信网页授权来获取**微信号**。

**第一步：用户同意授权，获取code**

**https://open.weixin.qq.com/connect/oauth2/authorize?appid=APPID&redirect\_uri=REDIRECT\_URI&response\_type=code&scope=SCOPE&state=STATE#wechat\_redirect 若提示“该链接无法访问”，请检查参数是否填写错误，是否拥有scope参数对应的授权作用域权限。**

**第二步：通过code换取网页授权openId也就是我们微信号**

**获取code后，请求以下链接获取access\_token： https://api.weixin.qq.com/sns/oauth2/access\_token?appid=APPID&secret=SECRET&code=CODE&grant\_type=authorization\_code**

**{ "access\_token":"ACCESS\_TOKEN",**

**"expires\_in":7200,**

**"refresh\_token":"REFRESH\_TOKEN",**

**"openid":"OPENID",**

**"scope":"SCOPE" }**

# 7.项目和业务部分

## 7.1.项目分类

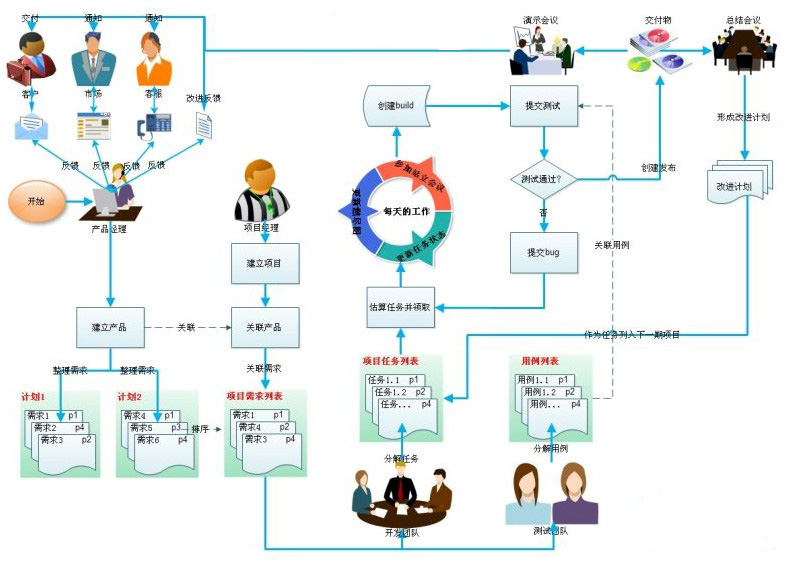
在公司中做的项目可以分为两种产品、项目.

项目：就是给一些公司接的项目，项目开发完成后。就交互，后面这个项目代码就不在维护了

产品：充分考虑扩展性和基本业务，来做一个产品。在这个产品上可以进行定制开发。

## 7.2 项目参与者

产品经理？PM？架构师（SE）？(开发PL? MDE？可能会有多个) (测试PL ？ TSE可能会有多个) UI 资料



开发团队:开始代码能完成需求

测试团队：测试功能

UI:负责界面设计、静态代码的编写

资料：负责界面的文字描述。

QA :通过项目质量监控,和PM,SE同级

开发和UI和资料，协同设计和开发，开发完成后转测试(测试策略)交给测试团队进行测试，测试完成后会出一个测试报告。

## 7.3 项目流程

可行性分析和立项和开工会

需求分析

需求设计

项目开发(多个迭代)

迭代开工会

迭代设计

迭代开发

迭代测试

集成测试

迭代发布

迭代总结

....不断迭代

项目验收

项目总结

## 7.4 业务注意事项

1. 不要多个项目都说你同一个模块。如果要说，就说后面是进行改进。

在pss,crm,shopping都写权限模块

2、多写点业务