# 尚硅谷——Java面试题

## 一 Java基础阶段

### 1.1 面向对象

#### 1. 面向对象的特征有哪些方面?

面向对象的特征主要有以下几个方面：

1)抽象：抽象就是忽略一个主题中与当前目标无关的那些方面，以便更充分地 注意与当前目标有关的方面。抽象并不打算了解全部问题，而只是选择其中的一 部分，暂时不用部分细节。抽象包括两个方面，一是过程抽象,二是数据抽象。

2)继承：继承是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了 一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称 为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原 始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量 ， 并且类可以修改或增加新的方法使之更适合特殊的需要。

3)封装：封装是把过程和数据包围起来，对数据的访问只能通过已定义的界面 。 面向对象计算始于这个基本概念，即现实世界可以被描绘成一系列完全自治、封 装的对象,这些对象通过一个受保护的接口访问其他对象。

4)多态性：多态性是指允许不同类的对象对同一消息作出响应。多态性包括参 数化多态性和包含多态性。多态性语言具有灵活、抽象、行为共享、代码共享的 优势，很好的解决了应用程序函数同名问题

#### 2. Java多态的具体体现

面向对象编程有四个特征：抽象，封装，继承，多态。

多态有四种体现形式：

1. 接口和接口的继承。

2. 类和类的继承。

3. 重载。

4. 重写。

其中重载和重写为核心。

重载：重载发生在同一个类中，在该类中如果存在多个同名方

法，但是方法的参数类型和个数不一样，那么说明该方法被重

载了。

重写：重写发生在子类继承父类的关系中，父类中的方法被子

类继承，方法名，返回值类型，参数完全一样，但是方法体不

一样，那么说明父类中的该方法被子类重写了。

#### 3、什么时候用 assert？

答：assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都 支持这种机制。一般来说，assertion 用于保证程序最基本、关键的正确性。 assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion 检查通常是关闭的。在实现中，断言是一个包含布尔表达式的语句， 在执行这个语句时假定该表达式为 true；如果表达式计算为 false，那么系统 会报告一个 Assertionerror。 断言用于调试目的： assert(a > 0); // throws an Assertionerror if a <= 0 断言可以有两种形式： assert Expression1 ; assert Expression1 : Expression2 ; Expression1 应该总是产生一个布尔值。 Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调 试信息的 String 消息。 断言在默认情况下是禁用的，要在编译时启用断言，需使用 source 1.4 标记 ： javac -source 1.4 Test.java 要在运行时启用断言，可使用 -enableassertions 或者 -ea 标记。 要在运行时选择禁用断言，可使用 -da 或者 -disableassertions 标记。 要在系统类中启用断言，可使用 -esa 或者 -dsa 标记。还可以在包的基础上 启用或者禁用断言。可以在预计正常情况下不会到达的任何位置上放置断言。断 言可以用于验证传递给私有方法的参数。不过，断言不应该用于验证传递给公有 方法的参数，因为不管是否启用了断言，公有方法都必须检查其参数。不过，既 可以在公有方法中，也可以在非公有方法中利用断言测试后置条件。另外，断言 不应该以任何方式改变程序的状态。

### 1.2 字符串相关

#### 1. StringBuffer StringBuilder String 区别

String 字符串常量 不可变 使用字符串拼接时是不同的2个空间

StringBuffer 字符串变量 可变 线程安全 字符串拼接直接在字符串后追加

StringBuilder 字符串变量 可变 非线程安全 字符串拼接直接在字符串后追加

1.StringBuilder执行效率高于StringBuffer高于String.

2.String是一个常量，是不可变的，所以对于每一次+=赋值都会创建一个新的对象， StringBuffer和StringBuilder都是可变的，当进行字符串拼接时采用append方 法，在原来的基础上进行追加，所以性能比String要高，又因为StringBuffer 是 线程安全的而StringBuilder是线程非安全的，所以StringBuilder的效率高于 StringBuffer.

3.对于大数据量的字符串的拼接，采用StringBuffer,StringBuilder.

#### 2、数组有没有length()方法?String 有没有length()方法？

答：数组没有length()方法，有length 的属性。String 有length()方法。JavaScript中，获得字符串的长度是通过length属性得到的，这一点容易和Java混淆。

#### 3、char 型变量中能不能存贮一个中文汉字?为什么?

答：char类型可以存储一个中文汉字，因为Java中使用的编码是Unicode（不选择任何特定的编码，直接使用字符在字符集中的编号，这是统一的唯一方法），一个char类型占2个字节（16bit），所以放一个中文是没问题的。

#### 4、Java中如何实现序列化，有什么意义？

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决对象流读写操作时可能引发的问题（如果不进行序列化可能会存在数据乱序的问题）。

要实现序列化，需要让一个类实现Serializable接口，该接口是一个标识性接口，标注该类对象是可被序列化的，然后使用一个输出流来构造一个对象输出流并通过writeObject(Object obj)方法就可以将实现对象写出(即保存其状态)；如果需要反序列化则可以用一个输入流建立对象输入流，然后通过readObject方法从流中读取对象。

### 1.3 集合

#### 1、几种重要的接口和类简介

1、List（有序、可重复）

List里存放的对象是有序的，同时也是可以重复的，List关注的是索引，拥有一系列和索引相关的方法，查询速度快。因为往list集合里插入或删除数据时，会伴随着后面数据的移动，所有插入删除数据速度慢。

2、Set（无序、不能重复）

Set里存放的对象是无序，不能重复的，集合中的对象不按特定的方式排序，只是简单地把对象加入集合中。

3、Map（键值对、键唯一、值不唯一）

Map集合中存储的是键值对，键不能重复，值可以重复。根据键得到值，对map集合遍历时先得到键的set集合，对set集合进行遍历，得到相应的值。

对比如下：

　 　 是否有序 是否允许元素重复

Collection 否 是

List 是 是

Set AbstractSet 否 否

　 HashSet

　 TreeSet 是（用二叉排序树）

Map AbstractMap 否 使用key-value来映射和存储数据，key必须唯一，value可以重复

　 HashMap

　 TreeMap 是（用二叉排序树）

#### 2、Vector与ArrayList

1.Vector是线程同步的，所以它也是线程安全的，而Arraylist是线程异步的，是不安全的。如果不考虑到线程的安全因素，一般用Arraylist效率比较高。

2.如果集合中的元素的数目大于目前集合数组的长度时，vector增长率为目前数组长度的100%，而Arraylist增长率为目前数组长度的50%。如果在集合中使用数据量比较大的数据，用vector有一定的优势。

3.如果查找一个指定位置的数据，vector和arraylist使用的时间是相同的，如果频繁的访问数据，这个时候使用vector和arraylist都可以。而如果移动一个指定位置会导致后面的元素都发生移动，这个时候就应该考虑到使用linklist,因为它移动一个指定位置的数据时其它元素不移动。

ArrayList 和Vector是采用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，都允许直接序号索引元素，但是插入数据要涉及到数组元素移动等内存操作，所以索引数据快，插入数据慢，Vector由于使用了synchronized方法（线程安全）所以性能上比ArrayList要差，LinkedList使用双向链表实现存储，按序号索引数据需要进行向前或向后遍历，但是插入数据时只需要记录本项的前后项即可，所以插入数度较快。

#### 3、Arraylist和Linkedlist

1.ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。

2.对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。

3.对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据。 这一点要看实际情况的。若只对单条数据插入或删除，ArrayList的速度反而优于LinkedList。但若是批量随机的插入删除数据，LinkedList的速度大大优于ArrayList. 因为ArrayList每插入一条数据，要移动插入点及之后的所有数据。

#### 4、HashMap与TreeMap

1、HashMap通过hashcode对其内容进行快速查找，而TreeMap中所有的元素都保持着某种固定的顺序，如果你需要得到一个有序的结果你就应该使用TreeMap（HashMap中元素的排列顺序是不固定的）。

2、在Map 中插入、删除和定位元素，HashMap是最好的选择。但如果您要按自然顺序或自定义顺序遍历键，那么TreeMap会更好。使用HashMap要求添加的键类明确定义了hashCode()和 equals()的实现。

两个map中的元素一样，但顺序不一样，导致hashCode()不一样。

同样做测试：

在HashMap中，同样的值的map,顺序不同，equals时，false;

而在treeMap中，同样的值的map,顺序不同,equals时，true，说明，treeMap在equals()时是整理了顺序了的。

#### 5、HashTable与HashMap

1、同步性:Hashtable是线程安全的，也就是说是同步的，而HashMap是线程序不安全的，不是同步的。

2、HashMap允许存在一个为null的key，多个为null的value 。

3、hashtable的key和value都不允许为null。

#### 6、如何解决HashMap的线程不安全问题？

1.替换成Hashtable，Hashtable通过对整个表上锁实现线程安全，因此效率比较低

2.使用Collections类的synchronizedMap方法包装一下。方法如下：

public static <K,V> Map<K,V> synchronizedMap(Map<K,V> m) 返回由指定映射支持的同步（线程安全的）映射

3.使用ConcurrentHashMap，它使用分段锁来保证线程安全

通过前两种方式获得的线程安全的HashMap在读写数据的时候会对整个容器上锁，而ConcurrentHashMap并不需要对整个容器上锁，它只需要锁住要修改的部分就行了

#### 7、Collection 和 Collections 的区别？

Collection 是 java.util 下的接口，它是各种集合的父接口，继承于它的 接口主要有 Set 和 List；Collections 是个 java.util 下的类，是针对集合的 帮助类，提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作 。

#### 8、List、Map、Set 三个接口，存取元素时，各有什么特点？

List 以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部 排序。Map 保存 key-value 值，value 可多值。

#### 9、Set 里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还 是 equals()? 它们有何区别?

Set 里的元素是不能重复的，用 equals ()方法来区分重复与否。覆盖 equals()方法用来判断对象的内容是否相同，而”==”判断地址是否相等,用来 决定引用值是否指向同一对象。

#### 10、TreeMap和TreeSet在排序时如何比较元素？Collections工具类中的sort()方法如何比较元素？

答：TreeSet要求存放的对象所属的类必须实现Comparable接口，该接口提供了比较元素的compareTo()方法，当插入元素时会回调该方法比较元素的大小。TreeMap要求存放的键值对映射的键必须实现Comparable接口从而根据键对元素进行排序。Collections工具类的sort方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象比较实现Comparable接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是Comparator接口的子类型（需要重写compare方法实现元素的比较），相当于一个临时定义的排序规则，其实就是是通过接口注入比较元素大小的算法，也是对回调模式的应用

#### 11、Map集合

实现类：HashMap、Hashtable、LinkedHashMap和TreeMap

HashMap是最常用的Map，它根据键的HashCode值存储数据，根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。因为键对象不可以重复，所以HashMap最多只允许一条记录的键为Null，允许多条记录的值为Null，是非同步的

Hashtable与HashMap类似，是HashMap的线程安全版，它支持线程的同步，即任一时刻只有一个线程能写Hashtable，因此也导致了Hashtale在写入时会比较慢，它继承自Dictionary类，不同的是它不允许记录的键或者值为null，同时效率较低。

ConcurrentHashMap

线程安全，并且锁分离。ConcurrentHashMap内部使用段(Segment)来表示这些不同的部分，每个段其实就是一个小的hash table，它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并发进行。

LinkedHashMap保存了记录的插入顺序，在用Iteraor遍历LinkedHashMap时，先得到的记录肯定是先插入的，在遍历的时候会比HashMap慢，有HashMap的全部特性。

TreeMap实现SortMap接口，能够把它保存的记录根据键排序，默认是按键值的升序排序（自然顺序），也可以指定排序的比较器，当用Iterator遍历TreeMap时，得到的记录是排过序的。不允许key值为空，非同步的；

#### 12、ArrayList扩容机制【必考】

发生扩容的条件：

根据传入的最小需要容量minCapacity来和数组的容量长度对比，若minCapactity大于或等于数组容量，则需要进行扩容。（如果实际存储数组是空数组，则最小需要容量就是默认容量）

实现扩容：

jdk7中采用>>位运算，右移动一位。 容量相当于扩大了1.5倍；

举例说明：添加20个元素到ArrayList中

当第一次插入元素时才分配10（默认）个对象空间。之后扩容会按照1.5倍增长。

也就是当添加第11个数据的时候，Arraylist继续扩容变为10\*1.5=15；

当添加第16个数据时，继续扩容变为15 \* 1.5 =22个；

#### 13、HashMap扩容机制

三个常量

static final int DEFAULT\_INITIAL\_CAPACITY = 16;

static final int MAXIMUM\_CAPACITY = 1 << 30;

static final float DEFAULT\_LOAD\_FACTOR = 0.75f;

三个常量中可以看出，默认的容器大小是16，最大长度是2的30次方，load factor默认是0.75，扩充的临界值是16\*0.75=12

HashMap进行扩容

当HashMap中的元素个数超过数组大小乘以负载因子(loadFactor)时，就会进行数组扩容，loadFactor的默认值为0.75，

也就是说，默认情况下，数组大小为16，

那么当HashMap中元素个数超过16\*0.75=12的时候，

就把数组的大小扩展为2\*16=32，即扩大一倍，

然后重新计算每个元素在数组中的位置。

### 1.4 异常

#### 1、Java 中的异常处理机制的简单原理和应用？

当 JAVA 程序违反了 JAVA 的语义规则时，JAVA 虚拟机就会将发生的错误表 示为一个异常。违反语义规则包括 2 种情况。一种是 JAVA 类库内置的语义检查。 例如数组下标越界,会引发 IndexOutOfBoundsException;访问 null 的对象时会 引发 NullPointerException。另一种情况就是 JAVA 允许程序员扩展这种语义检 查，程序员可以创建自己的异常，并自由选择在何时用 throw 关键字引发异常。 所有的异常都是 java.lang.Thowable 的子类。

#### 2、error 和 exception 有什么区别?

error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困 难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况 ； exception 表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现 问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

#### 3、try {}里有一个return语句，那么紧跟在这个try后的finally {}里的code 会不会被执行，什么时候被执行，在 return 前还是后?

会执行，在 return 前执行。

#### 4、JAVA 语言如何进行异常处理，关键字：throws,throw,try,catch,finally 分别代表什么意义？在 try 块中可以抛出异常吗？

Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并

提供了良好的接口。在 Java 中，每个异常都是一个对象，它是 Throwable 类或 其它子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有 异常信息，调用这个对象的方法可以捕获到这个异常并进行处理。Java 的异常 处理是通过 5 个关键词来实现的：try、catch、throw、throws 和 finally。一 般情况下是用 try 来执行一段程序，如果出现异常，系统会抛出（throws）一个 异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺 省处理器来处理； try 用来指定一块预防所有“异常”的程序； catch 子句紧跟在 try 块后面，用来指定你想要捕捉的“异常”的类型； throw 语句用来明确地抛出一个“异常”； throws 用来标明一个成员函数可能抛出的各种“异常”； Finally 为确保一段代码不管发生什么“异常”都被执行一段代码； 可以在一个成员函数调用的外面写一个 try 语句，在这个成员函数内部写另一 个 try 语句保护其他代码。每当遇到一个 try 语句，“异常”的框架就放到堆栈 上面，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种“异常” 进行处理，堆栈就会展开，直到遇到有处理这种“异常”的 try 语句。

#### 5、运行时异常与一般异常有何异同？

异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的 通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须 声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行 时异常。

#### 6、给我一个你最常见到的 runtime exception？

ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IndexOutOfBoundsException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException, ProviderException, RasterFormatException, SecurityException, SystemException, UndeclaredThrowableException, UnmodifiableSetException, UnsupportedOperationExceptio

### 1.5 IO流

#### 1、什么是比特(Bit),什么是字节(Byte),什么是字符(Char),它们长度是多少,各有什么区别

Bit最小的二进制单位 ，是计算机的操作部分 取值0或者1  
Byte是计算机操作数据的最小单位由8位bit组成 取值（-128-127）  
Char是用户的可读写的最小单位，在[**Java**](http://lib.csdn.net/base/java)里面由16位bit组成 取值（0-65535）

Bit 是最小单位 计算机 只能认识 0或者1

8个字节 是给计算机看的  
字符 是看到的东西  一个字符=二个字节

#### 2、什么是流,按照传输的单位,分成哪两种流,并且他们的父类叫什么流是指数据的传输

字节流，字符流   
字节流：InputStream OutputStream  
字符流：Reader Writer

#### 3、流按照传输的方向可以分为哪两种,分别举例说明

输入输出相对于程序  
输入流InputStream  
，输出流OutputStream

#### 4、按照实现功能分为哪两种,分别举例说明

答案

节点流，处理流  
节点流：OutputStream  
处理流： OutputStreamWriter

#### 5、BufferedReader属于哪种流,它主要是用来做什么的,它里面有那些经典的方法

**答案**

属于处理流中的缓冲流，可以将读取的内容存在内存里面，有readLine（）方法

#### 6、什么是节点流,什么是处理流,它们各有什么用处,处理流的创建有什么特征

节点流 直接与数据源相连，用于输入或者输出  
处理流：在节点流的基础上对之进行加工，进行一些功能的扩展  
处理流的构造器必须要 传入节点流的子类

#### 7、如果我要对字节流进行大量的从硬盘读取,要用那个流,为什么

**BufferedInputStream 使用缓冲流能够减少对硬盘的损伤**

#### 8、如果我要打印出不同类型的数据到数据源,那么最适合的流是那个流,为什么

**Printwriter 可以打印各种数据类型**

#### 9、怎么样把我们控制台的输出改成输出到一个文件里面,这个技术叫什么

**SetOut（printWriter,printStream）重定向**

#### 10、怎么样把输出字节流转换成输出字符流,说出它的步骤

**使用 转换处理流OutputStreamWriter 可以将字节流转为字符流  
New OutputStreamWriter（new FileOutputStream（File file））;**

#### 11、把包括基本类型在内的数据和字符串按顺序输出到数据源，或者按照顺序从数据源读入，一般用哪两个流

**DataInputStream DataOutputStream**

#### 12、把一个对象写入数据源或者从一个数据源读出来,用哪两个流

**ObjectInputStream ObjectOutputStream**

#### 13、什么叫对象序列化，什么是反序列化，实现对象序列化需要做哪些工作

**对象序列化，将对象以二进制的形式保存在硬盘上  
反序列化；将二进制的文件转化为对象读取  
实现serializable接口**

不想让字段放在硬盘上就加transient

#### 14、如果在对象序列化的时候不想给一个字段的数据保存在硬盘上面,采用那个关键字?

**transient关键字**

#### 15、在实现序列化接口是时候一般要生成一个serialVersionUID字段,它叫做什么,一般有什么用

**是版本号，要保持版本号的一致 来进行序列化**

**为了防止序列化出错**

#### 16、InputStream里的read()返回的是什么,read(byte[] data)是什么意思,返回的是什么值

**返回的是所读取的字节的int型（范围0-255）  
read（byte [ ] data）将读取的字节储存在这个数组  
返回的就是传入数组参数个数**

**Read  字节读取字节  字符读取字符**

#### 17、OutputStream里面的write()是什么意思,write(byte b[], int off, int len)这个方法里面的三个参数分别是什么意思

**write将指定字节传入数据源  
Byte b[ ]是byte数组  
b[off]是传入的第一个字符  
b[off+len-1]是传入的最后的一个字符   
len是实际长度**

#### 18、流一般需要不需要关闭,如果关闭的话在用什么方法,一般要在那个代码块里面关闭比较好，处理流是怎么关闭的，如果有多个流互相调用传入是怎么关闭的？

**流一旦打开就必须关闭，使用close方法  
放入finally语句块中（finally 语句一定会执行）  
调用的处理流就关闭处理流  
多个流互相调用只关闭最外层的流**

#### ****19、****Java中的所有的流可以分为几大类,它们的名字是什么,各代表什么

**分为 字节输入流 InputStream**

**字节输出流 OutputStream**

**字符输入流 Reader**

**字符输出流 Writer**

**所有流都是这四个流的子类**

#### 20、说下常用的io流

InputStream,OutputStream,

FileInputStream,FileOutputStream,

BufferedInputStream,BufferedOutputStream

Reader,Writer

BufferedReader,BufferedWriter

#### 21、 io流怎样读取文件的？

使用File对象获取文件路径，通过字符流Reader加入文件，使用字符缓存流BufferedReader处理Reader，再定义一个字符串，循环遍历出文件。代码如下：

File file = new File("d:/spring.txt");

try {

Reader reader = new FileReader(file);

BufferedReader buffered = new BufferedReader(reader);

String data = null;

while((data = buffered.readLine())!=null){

System.out.println(data);

}

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

#### 22、说说你对io流的理解

Io流主要是用来处理输入输出问题，常用的io流有InputStream，OutputStream，Reader，Writer等

#### 23、JAVA的IO流和readLine方法

Java的io流用来处理输入输出问题，readLine是BufferedReader里的一个方法，用来读取一行。

#### 24、用什么把对象动态的写入磁盘中，写入要实现什么接口。

ObjectInputStream，需要实现Serializable接口

#### 25、FileInputStream 创建详情，就是怎样的创建不报错，它列出了几种形式

FileInputStream是InputStream的子类，通过接口定义，子类实现创建FileInputStream,

#### 26、请问你在什么情况下会在你得java代码中使用可序列化？ 如何实现java序列化？

把一个对象写入数据源或者从一个数据源读出来，使用可序列化，需要实现Serializable接口

#### 27、PrintStream、BufferedWriter、PrintWriter的比较?

PrintStream类的输出功能非常强大，通常如果需要输出文本内容，都应该将输出流包装成PrintStream后进行输出。它还提供其他两项功能。与其他输出流不同，PrintStream 永远不会抛出 IOException；而是，异常情况仅设置可通过 checkError 方法测试的内部标志。另外，为了自动刷新，可以创建一个 PrintStream

BufferedWriter:将文本写入字符输出流，缓冲各个字符从而提供单个字符，数组和字符串的高效写入。通过write()方法可以将获取到的字符输出，然后通过newLine()进行换行操作。BufferedWriter中的字符流必须通过调用flush方法才能将其刷出去。并且BufferedWriter只能对字符流进行操作。如果要对字节流操作，则使用BufferedInputStream。

PrintWriter的println方法自动添加换行，不会抛异常，若关心异常，需要调用checkError方法看是否有异常发生，PrintWriter构造方法可指定参数，实现自动刷新缓存（autoflush）；

### 1.6 多线程

#### 1、sleep() 和 wait() 有什么区别?

sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，把执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用sleep不会释放对象锁。

wait是Object类的方法，对此对象调用wait方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出notify方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。

#### 2、sleep()和yield()有什么区别?

1）sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；

2）线程执行sleep()方法后转入阻塞（blocked）状态，而执行yield()方法后转入就绪（ready）状态；

3）sleep()方法声明抛出InterruptedException，而yield()方法没有声明任何异常；

4）sleep()方法比yield()方法（跟操作系统相关）具有更好的可移植性。

#### 3、启动一个线程是用run()还是start()?

启动一个线程是调用start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由JVM调度并执行。这并不意味着线程就会立即运行。run()方法可以产生必须退出的标志来停止一个线程

#### 4、请说出与线程同步相关的方法。

1）wait():使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；

2）sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉InterruptedException 异常；

3）notify():唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且与优先级无关；

4）notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争；

5）JDK 1.5通过Lock接口提供了显式(explicit)的锁机制，增强了灵活性以及对线程的协调。Lock接口中定义了加锁（lock()）和解锁(unlock())的方法，同时还提供了newCondition()方法来产生用于线程之间通信的Condition对象；

6）JDK 1.5还提供了信号量(semaphore)机制，信号量可以用来限制对某个共享资源进行访问的线程的数量。在对资源进行访问之前，线程必须得到信号量的许可（调用Semaphore对象的acquire()方法）；在完成对资源的访问后，线程必须向信号量归还许可（调用Semaphore对象的release()方法）。

#### 5、编写多线程程序有几种实现方式？

Java 5以前实现多线程有两种实现方法：一种是继承Thread类；另一种是实现Runnable接口。两种方式都要通过重写run()方法来定义线程的行为，推荐使用后者，因为Java中的继承是单继承，一个类有一个父类，如果继承了Thread类就无法再继承其他类了，显然使用Runnable接口更为灵活。

补充：Java 5以后创建线程还有第三种方式：实现Callable接口，该接口中的call方法可以在线程执行结束时产生一个返回值，代码如下所示：

package com.lovo.demo;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

class MyTask implements Callable<Integer> {

private int upperBounds;

public MyTask(int upperBounds) {

this.upperBounds = upperBounds;

}

@Override

public Integer call() throws Exception {

int sum = 0;

for(int i = 1; i <= upperBounds; i++) {

sum += i;

}

return sum;

}

}

public class Test {

public static void main(String[] args) throws Exception {

List<Future<Integer>> list = new ArrayList<>();

ExecutorService service = Executors.newFixedThreadPool(10);

for(int i = 0; i < 10; i++) {

list.add(service.submit(new MyTask((int) (Math.random() \* 100))));

}

int sum = 0;

for(Future<Integer> future : list) {

while(!future.isDone()) ;

sum += future.get();

}

System.out.println(sum);

}

}

#### 6、synchronized关键字的用法？

synchronized关键字可以将对象或者方法标记为同步，以实现对对象和方法的互斥访问，可以用synchronized(对象) { … }定义同步代码块，或者在声明方法时将synchronized作为方法的修饰符。在第60题的例子中已经展示了synchronized关键字的用法。

#### 7、举例说明同步和异步。

如果系统中存在临界资源（资源数量少于竞争资源的线程数量的资源），例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就必须进行同步存取（数据库操作中的悲观锁就是最好的例子）。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。事实上，所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作。

#### 8、 java线程池概述

**java**线程池的工作原理和数据库连接池的差不多，因为每次重新创建线程

都是很耗资源的操作，所以我们可以建立一个线程池，这样当需要用到线程

进行某些操作时，就可以直接去线程池里面找到空闲的线程，这样就可以直接

使用，而不用等到用到的时候再去创建，用完之后可以把该线程重新放入线程池

供其他请求使用从而提高应用程序的性能。

**线程池的核心流程:**

1.构建一个 ThreadPoolExecutor 并指定默认要创建的线程的数量

2.通过 threadPool.execute()

去添加一个个要执行的线程即实现了Runable接口的java类

3.在实现了Runable接口的java类的run方法中写入具体的业务代码

**线程池的业务场景：**

我在工作的时候，当时一个同事给我提了一个需求，目前有大量的图片

需要处理生产缩略图并进行加水印，因为按照普通的处理方法一个个的

进行处理太慢了，问我有没有好的解决方案，这个时候我就想到了java中

的线程池，我构建了一个线程数为5个线程池，然后采用分段批量提取的

方式每500条为一组数据进行图片信息的提取，然后再把这些通过Threadpool的

execute方法交给线程池中的线程进行处理，即充分使用了CPU硬件资源又加快

了大数据情况下程序的处理效率。

我当时在工作的过程中，认识一个做电商的朋友，他们当时公司才起步，很多

技术都不成熟，所以就常常和我探讨一些技术问题，有次他向我请教一个问题，

问我如何才能提高网站的性能，我根据自己在项目中的经验以及自己以前阅读的

关于优化方面的资料给他提出了很多建议，如用lucene进行全文检索，用memcached

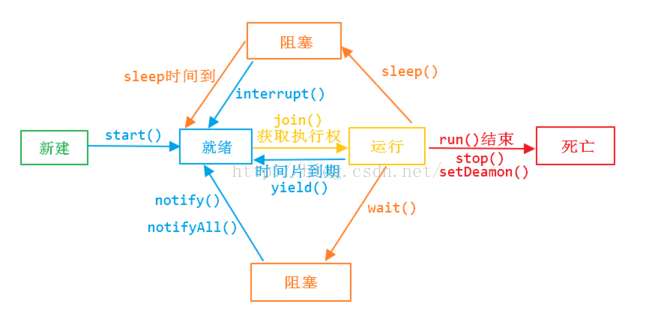
进行分布式缓存，以及通过spring定时器结合freeMarker模板引擎来生成静态

页面，由于要生成的页面的数量比较多，考虑到程序的性能，我建议他结合

java的线程池进行工作，这样就可以充分使用了CPU硬件资源又加快

了大数据情况下程序的处理效率。

#### 9、线程的生命周期



#### 10、线程间通信：

1. 首先，线程A把本地内存A中更新过的共享变量刷新到主内存中去。

2. 然后，线程B到主内存中去读取线程A之前已更新过的共享变量

### 1.7 反射相关

#### 1、Java反射机制的作用？

（1）在运行时判断任意一个对象所属的类。

（2）在运行时判断任意一个类所具有的成员变量和方法。

（3）在运行时任意调用一个对象的方法。

（4）在运行时构造任意一个类的对象。

#### 2、什么是反射机制？

反射就是动态加载对象，并对对象进行剖析。在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法，这种动态获取信息以及动态调用对象方法的功能成为Java反射机制。

#### 3、哪里用到反射机制？

jdbc中，获取数据库驱动对象实例的代码，很多框架都用到反射机制，如hibernate、struts、spring。

#### 4、反射机制的优缺点？

优点：可以动态的创建对象和编译，最大限度发挥了java的灵活性。

缺点：对性能有影响。使用反射基本上一种解释操作，告诉JVM我们要做什么并且满足我们的要求，这类操作总是慢于直接执行java代码。

### 1.8 设计模式

#### 1、Java中的23种设计模式：

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 Responsibleity（责任链模式）

#### 2、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一个类的实例，通常这一组类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。然后需要定义一个工厂类，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。

#### 3、单例模式

单例就是该类只能返回一个实例。

单例所具备的特点：

1.私有化的构造函数

2.私有的静态的全局变量

3.公有的静态的方法

单例分为懒汉式、饿汉式和双层锁式

饿汉式:

public class Singleton1 {

private Singleton1() {};

private static Singleton1 single = new Singleton1();

public static Singleton1 getInstance() {

return single;

}

}

懒汉式：

public class Singleton2 {

private Singleton2() {}

private static Singleton2 single=null;

public tatic Singleton2 getInstance() {

if (single == null) {

single = new Singleton2();

}

return single;

}

}

线程安全：

public class Singleton3 {

private Singleton3() {}

private static Singleton3 single ;

public static Singleton3 getInstance() {

if(null == single){

synchronized(single ){

if(null == single){

single = new Singleton3();

}

}

}

return single;

}

}

参考：

通过双重判断来保证单列设计模式在多线程中的安全性，

并且它在性能方面提高了很多。

synchronized在方法上加锁 (同步锁)

synchronized在代码块内部加锁 （同步代码块）

synchronized(同步锁)

使用synchronized如何解决线程安全的问题？

1.synchronized在方法上加锁

2.synchronized在代码块内部加锁

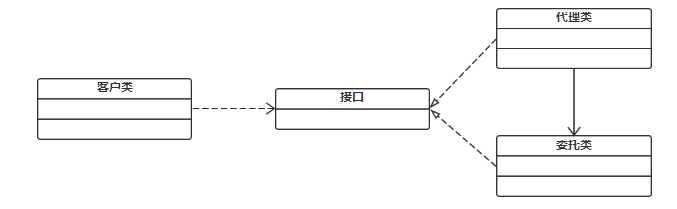
1.懒汉 2.饿汉 3.双重判断

#### 4、代理模式

1.什么是代理模式？

**代理模式的定义：**代理模式给某一个对象提供一个代理对象，并由代理对象控制对原对象的引用。通俗的来讲代理模式就是我们生活中常见的中介。

举个例子来说明：假如说我现在想买一辆二手车，虽然我可以自己去找车源，做质量检测等一系列的车辆过户流程，但是这确实太浪费我得时间和精力了。我只是想买一辆车而已为什么我还要额外做这么多事呢？于是我就通过中介公司来买车，他们来给我找车源，帮我办理车辆过户流程，我只是负责选择自己喜欢的车，然后付钱就可以了。用图表示如下：



2.为什么要用代理模式？

**中介隔离作用：**在某些情况下，一个客户类不想或者不能直接引用一个委托对象，而代理类对象可以在客户类和委托对象之间起到中介的作用，其特征是代理类和委托类实现相同的接口。

**开闭原则，增加功能：**代理类除了是客户类和委托类的中介之外，我们还可以通过给代理类增加额外的功能来扩展委托类的功能，这样做我们只需要修改代理类而不需要再修改委托类，符合代码设计的开闭原则。代理类主要负责为委托类预处理消息、过滤消息、把消息转发给委托类，以及事后对返回结果的处理等。代理类本身并不真正实现服务，而是同过调用委托类的相关方法，来提供特定的服务。真正的业务功能还是由委托类来实现，但是可以在业务功能执行的前后加入一些公共的服务。例如我们想给项目加入缓存、日志这些功能，我们就可以使用代理类来完成，而没必要打开已经封装好的委托类。

3.有哪几种代理模式？

     我们有多种不同的方式来实现代理。如果按照代理创建的时期来进行分类的话， 可以分为两种：静态代理、动态代理。静态代理是由程序员创建或特定工具自动生成源代码，在对其编译。在程序员运行之前，代理类.class文件就已经被创建了。动态代理是在程序运行时通过反射机制动态创建的。

**静态代理**

第一步：创建服务类接口

**第二步：实现服务接口**

**第三步：创建代理类**

**第四步：编写测试类**

**静态代理总结：**

优点：可以做到在符合开闭原则的情况下对目标对象进行功能扩展。

缺点：我们得为每一个服务都得创建代理类，工作量太大，不易管理。同时接口一旦发生改变，代理类也得相应修改。

**动态代理**

　　在动态代理中我们不再需要再手动的创建代理类，我们只需要编写一个动态处理器就可以了。真正的代理对象由JDK再运行时为我们动态的来创建。

**第一步：编写动态处理器**

**第二步：编写测试类**

**动态代理总结：**虽然相对于静态代理，动态代理大大减少了我们的开发任务，同时减少了对业务接口的依赖，降低了耦合度。但是还是有一点点小小的遗憾之处，那就是它始终无法摆脱仅支持interface代理的桎梏，因为它的设计注定了这个遗憾。回想一下那些动态生成的代理类的继承关系图，它们已经注定有一个共同的父类叫Proxy。Java的继承机制注定了这些动态代理类们无法实现对class的动态代理，原因是多继承在Java中本质上就行不通。有很多条理由，人们可以否定对 class代理的必要性，但是同样有一些理由，相信支持class动态代理会更美好。接口和类的划分，本就不是很明显，只是到了Java中才变得如此的细化。如果只从方法的声明及是否被定义来考量，有一种两者的混合体，它的名字叫抽象类。实现对抽象类的动态代理，相信也有其内在的价值。此外，还有一些历史遗留的类，它们将因为没有实现任何接口而从此与动态代理永世无缘。如此种种，不得不说是一个小小的遗憾。但是，不完美并不等于不伟大，伟大是一种本质，Java动态代理就是佐例。

**CGLIB代理**

     JDK实现动态代理需要实现类通过接口定义业务方法，对于没有接口的类，如何实现动态代理呢，这就需要CGLib了。CGLib采用了非常底层的字节码技术，其原理是通过字节码技术为一个类创建子类，并在子类中采用方法拦截的技术拦截所有父类方法的调用，顺势织入横切逻辑。但因为采用的是继承，所以不能对final修饰的类进行代理。JDK动态代理与CGLib动态代理均是实现Spring AOP的基础。

**第一步：创建CGLIB代理类**

**第二步：创建测试类**

**CGLIB代理总结：**CGLIB创建的动态代理对象比JDK创建的动态代理对象的性能更高，但是CGLIB创建代理对象时所花费的时间却比JDK多得多。所以对于单例的对象，因为无需频繁创建对象，用CGLIB合适，反之使用JDK方式要更为合适一些。同时由于CGLib由于是采用动态创建子类的方法，对于final修饰的方法无法进行代理。

#### 5、观察者模式

**意图：**定义对象间的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得到通知并被自动更新。

**主要解决：**一个对象状态改变给其他对象通知的问题，而且要考虑到易用和低耦合，保证高度的协作。

**何时使用：**一个对象（目标对象）的状态发生改变，所有的依赖对象（观察者对象）都将得到通知，进行广播通知。

**如何解决：**使用面向对象技术，可以将这种依赖关系弱化。

**关键代码：**在抽象类里有一个 ArrayList 存放观察者们。

**应用实例：** 1、拍卖的时候，拍卖师观察最高标价，然后通知给其他竞价者竞价。 2、西游记里面悟空请求菩萨降服红孩儿，菩萨洒了一地水招来一个老乌龟，这个乌龟就是观察者，他观察菩萨洒水这个动作。

**优点：** 1、观察者和被观察者是抽象耦合的。 2、建立一套触发机制。

**缺点：** 1、如果一个被观察者对象有很多的直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间。 2、如果在观察者和观察目标之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃。 3、观察者模式没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而仅仅只是知道观察目标发生了变化。

**使用场景：**

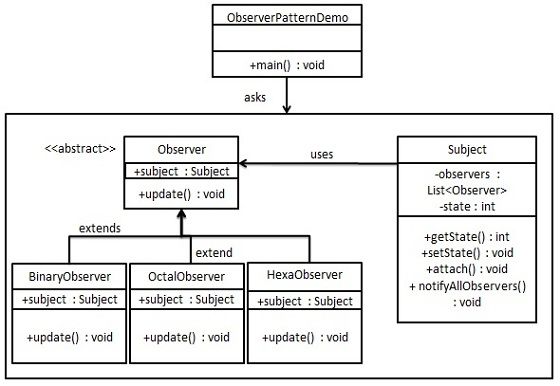
* 一个抽象模型有两个方面，其中一个方面依赖于另一个方面。将这些方面封装在独立的对象中使它们可以各自独立地改变和复用。
* 一个对象的改变将导致其他一个或多个对象也发生改变，而不知道具体有多少对象将发生改变，可以降低对象之间的耦合度。
* 一个对象必须通知其他对象，而并不知道这些对象是谁。
* 需要在系统中创建一个触发链，A对象的行为将影响B对象，B对象的行为将影响C对象……，可以使用观察者模式创建一种链式触发机制。

**注意事项：** 1、JAVA 中已经有了对观察者模式的支持类。 2、避免循环引用。 3、如果顺序执行，某一观察者错误会导致系统卡壳，一般采用异步方式。

**实现**

观察者模式使用三个类 Subject、Observer 和 Client。Subject 对象带有绑定观察者到 Client 对象和从 Client 对象解绑观察者的方法。我们创建 *Subject* 类、*Observer* 抽象类和扩展了抽象类 *Observer* 的实体类。

*ObserverPatternDemo*，我们的演示类使用 *Subject* 和实体类对象来演示观察者模式。



#### 6、适配器模式

适配器模式（Adapter Pattern）是作为两个不兼容的接口之间的桥梁。这种类型的设计模式属于结构型模式，它结合了两个独立接口的功能。

这种模式涉及到一个单一的类，该类负责加入独立的或不兼容的接口功能。举个真实的例子，读卡器是作为内存卡和笔记本之间的适配器。您将内存卡插入读卡器，再将读卡器插入笔记本，这样就可以通过笔记本来读取内存卡。

我们通过下面的实例来演示适配器模式的使用。其中，音频播放器设备只能播放 mp3 文件，通过使用一个更高级的音频播放器来播放 vlc 和 mp4 文件。

介绍

**意图：**将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

**主要解决：**主要解决在软件系统中，常常要将一些"现存的对象"放到新的环境中，而新环境要求的接口是现对象不能满足的。

**何时使用：** 1、系统需要使用现有的类，而此类的接口不符合系统的需要。 2、想要建立一个可以重复使用的类，用于与一些彼此之间没有太大关联的一些类，包括一些可能在将来引进的类一起工作，这些源类不一定有一致的接口。 3、通过接口转换，将一个类插入另一个类系中。（比如老虎和飞禽，现在多了一个飞虎，在不增加实体的需求下，增加一个适配器，在里面包容一个虎对象，实现飞的接口。）

**如何解决：**继承或依赖（推荐）。

**关键代码：**适配器继承或依赖已有的对象，实现想要的目标接口。

**应用实例：** 1、美国电器 110V，中国 220V，就要有一个适配器将 110V 转化为 220V。 2、JAVA JDK 1.1 提供了 Enumeration 接口，而在 1.2 中提供了 Iterator 接口，想要使用 1.2 的 JDK，则要将以前系统的 Enumeration 接口转化为 Iterator 接口，这时就需要适配器模式。 3、在 LINUX 上运行 WINDOWS 程序。 4、JAVA 中的 jdbc。

**优点：** 1、可以让任何两个没有关联的类一起运行。 2、提高了类的复用。 3、增加了类的透明度。 4、灵活性好。

**缺点：** 1、过多地使用适配器，会让系统非常零乱，不易整体进行把握。比如，明明看到调用的是 A 接口，其实内部被适配成了 B 接口的实现，一个系统如果太多出现这种情况，无异于一场灾难。因此如果不是很有必要，可以不使用适配器，而是直接对系统进行重构。 2.由于 JAVA 至多继承一个类，所以至多只能适配一个适配者类，而且目标类必须是抽象类。

**使用场景：**有动机地修改一个正常运行的系统的接口，这时应该考虑使用适配器模式。

**注意事项：**适配器不是在详细设计时添加的，而是解决正在服役的项目的问题。

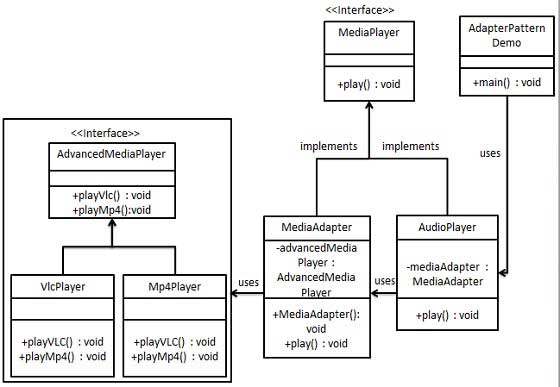
**实现**

我们有一个 *MediaPlayer* 接口和一个实现了 *MediaPlayer* 接口的实体类 *AudioPlayer*。默认情况下，*AudioPlayer* 可以播放 mp3 格式的音频文件。

我们还有另一个接口 *AdvancedMediaPlayer* 和实现了 *AdvancedMediaPlayer* 接口的实体类。该类可以播放 vlc 和 mp4 格式的文件。

我们想要让 *AudioPlayer* 播放其他格式的音频文件。为了实现这个功能，我们需要创建一个实现了 *MediaPlayer* 接口的适配器类 *MediaAdapter*，并使用 *AdvancedMediaPlayer* 对象来播放所需的格式。

*AudioPlayer* 使用适配器类 *MediaAdapter* 传递所需的音频类型，不需要知道能播放所需格式音频的实际类。*AdapterPatternDemo*，我们的演示类使用 *AudioPlayer* 类来播放各种格式。



#### 7、命令模式

命令模式（Command Pattern）是一种数据驱动的设计模式，它属于行为型模式。请求以命令的形式包裹在对象中，并传给调用对象。调用对象寻找可以处理该命令的合适的对象，并把该命令传给相应的对象，该对象执行命令。

**介绍**

**意图：**将一个请求封装成一个对象，从而使您可以用不同的请求对客户进行参数化。

**主要解决：**在软件系统中，行为请求者与行为实现者通常是一种紧耦合的关系，但某些场合，比如需要对行为进行记录、撤销或重做、事务等处理时，这种无法抵御变化的紧耦合的设计就不太合适。

**何时使用：**在某些场合，比如要对行为进行"记录、撤销/重做、事务"等处理，这种无法抵御变化的紧耦合是不合适的。在这种情况下，如何将"行为请求者"与"行为实现者"解耦？将一组行为抽象为对象，可以实现二者之间的松耦合。

**如何解决：**通过调用者调用接受者执行命令，顺序：调用者→接受者→命令。

**关键代码：**定义三个角色：1、received 真正的命令执行对象 2、Command 3、invoker 使用命令对象的入口

**应用实例：**struts 1 中的 action 核心控制器 ActionServlet 只有一个，相当于 Invoker，而模型层的类会随着不同的应用有不同的模型类，相当于具体的 Command。

**优点：** 1、降低了系统耦合度。 2、新的命令可以很容易添加到系统中去。

**缺点：**使用命令模式可能会导致某些系统有过多的具体命令类。

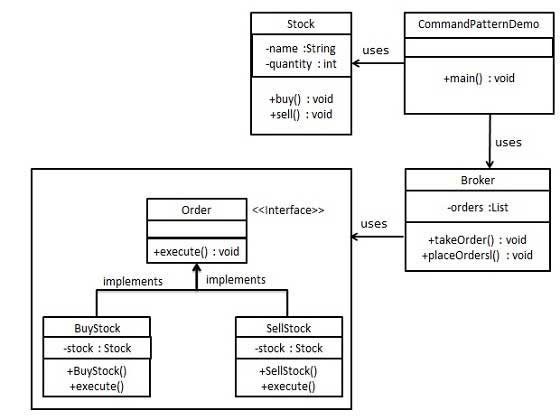
**使用场景：**认为是命令的地方都可以使用命令模式，比如： 1、GUI 中每一个按钮都是一条命令。 2、模拟 CMD。

**注意事项：**系统需要支持命令的撤销(Undo)操作和恢复(Redo)操作，也可以考虑使用命令模式，见命令模式的扩展。

**实现**

我们首先创建作为命令的接口 *Order*，然后创建作为请求的 *Stock* 类。实体命令类 *BuyStock* 和 *SellStock*，实现了 *Order* 接口，将执行实际的命令处理。创建作为调用对象的类 *Broker*，它接受订单并能下订单。

*Broker* 对象使用命令模式，基于命令的类型确定哪个对象执行哪个命令。*CommandPatternDemo*，我们的演示类使用 *Broker* 类来演示命令模式。



## 二 数据库阶段

#### 1、JDBC连接数据库步骤(以MYSQL为例)

1、加载JDBC驱动程序：

通过Class类的forName方法实现，并将驱动地址放进去成功加载后，会将Driver类的实例注册到DriverManager类中。

2、提供JDBC连接的URL 、创建数据库的连接

要连接数据库，需要向java.sql.DriverManager请求并获得Connection对象，该对象就代表一个数据库的连接。

使用DriverManager的getConnectin()方法传入指定的欲连接的数据库的路径、数据库的用户名和密码。

Connection con=DriverManager.getConnection(url , username , password);

"jdbc:mysql://localhost/test?user=root&password=123&useUnicode=true&characterEncoding=utf-8”;

3、创建一个Statement

要执行SQL语句，必须获得java.sql.Statement实例

执行静态SQL语句。通常通过Statement实例实现。

执行动态SQL语句。通常通过PreparedStatement实例实现。

String sql = “”;

Statement st = con.createStatement() ;

PreparedStatement pst = con.prepareStatement(sql) ;

4、执行SQL语句

Statement接口提供了executeQuery、executeUpdate、execute三种方法

executeQuery：执行select语句，返回ResultSet结果集

ResultSet rst = pst.executeQuery();

executeUpdate：执行insert、update、delete语句pst.executeUpdate();

5、关闭JDBC对象

操作完成以后要把所有使用的JDBC对象全都关闭，以释放JDBC资源。

#### 2、数据库连接池

数据库连接池简介：

数据库连接池的基本思想就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。

什么是数据库连接池：

Connection pooling，它是程序启动时建立足够的数据库连接，并将这些连接组成一个连接池，由程序动态地对池中的连接进行申请，使用，释放。

数据库连接池的运行机制：

1.从连接池获取或创建可用的连接；

2.使用完毕之后，把连接返还给连接池；

3.在系统关闭前，断开所有连接并释放连接占用的系统资源；

4.还能够处理无效连接（原来登记为可用的连接，由于某种原因不再可用，如超时、通讯问题），并能限制连接池中的连接总数不低于某个预定值和不超过某个预定值；

#### 3、mysql的数据库导入导出

配置：

首先找到mysql的安装目录，进入bin目录下复制路径

将mysql的bin目录粘贴在计算机环境变量的path中

授权：

登录mysql

将某张表的某个权限赋给某个用户

grant [select,insert,update,delete,create,drop] on [databaseName].[tableName] to [userName]@[userIP] identified by [‘连接口令’]

grant select,insert,update,delete,create,drop on oa\_ssh.user to root@[IP] identified by 'root';

将所有库的所有权限赋给某个用户

grant all privileges on \*.\* to [userName]@[userIp] identified by [‘连接口令’]

grant all privileges on \*.\* to root@[IP] identified by ‘root';

将所有库的所有权限赋给所有用户

grant all privileges on \*.\* to root@'%' identified by ‘root’;

导出本地数据库：

mysqldump -u 用户名 -p 数据库名 > 磁盘：导出的文件名(加后缀)

远程导出数据库：

mysqldump -h IP -u 用户名 -p 数据库名称 >导出的文件名(加后缀)

远程导出数据表：

mysqldump -u root -p -d --add-drop-table 数据库名称 > 导出文件名(加后缀)

导入数据：

mysql -u root -p登录成功后 ==》 source 磁盘：导入的文件名(加后缀)

#### 4、jdbc分段批量提交的时候出现异常怎么处理?

通过Map来解决性能问题。首先在分段批量提交的时候，我们不采用事务，这样就保证了合法的数据就自动提交，不合法的数据就自己自动进行回滚，为了避免不合法数据影响后续合法数据的提交，采用定义业务规则字典表，实现对数据的验证，将不合法的数据记录下来，供用户进行后续处理，而合法的数据就全部提交。

#### 5、jdbc批量处理数据

dbc批量处理数据是通过PreparedStatement对象的 addbatch(), executebatch（） clearbatch()进行和数据库的交互。通常我们使用分段批量处理的方式 这样可以提高程序的性能 ，防止内存溢出。

1.每个sql语句都和数据库交互一次(非批量操作)

2.只和数据库交互一次(批量操作)(内存溢出)

当数据达到一定额度的时候就和数据库进行交互，分多次进行(分段批量操作)(500或者1000)

pst.addBatch();

if (i > 0 && i%1000 == 0) {

pst.executeBatch();

pst.clearBatch();

}

#### 6、Oracle分页

select \* from (select \* from (select s.\*,rownum rn from student s ) where rn<=5) where rn>0

#### 7、Oracle的基本数据类型

Oracle的基本数据类型（常用）：

1、字符型

Char 固定长度字符串 占2000个字节

Varchar2 可变长度字符串 占4000个字节

Nvarchar2 占2000个字符（最多能存2000个字母/中文）

2、大对象型（lob）

Blob ：二进制数据 最大长度4G

Blob 用于存一些图片，视频，文件。

比如：当我们在进行文件上传时，我们一般把上传的文件存在硬盘上，可以不占用 数据库，下载时，如果项目迁移时，文件也要跟着迁移。因此我们可以把用blob把它存在数据库中。但这样也增加了数据库的负担。

Clob ：字符数据 最大长度4G，可以存大字符串 varchar2和nvarchar2都具有一定的局限性，它们长度有限，但数据库中无论用varchar2或nvarchar2类型，还是用clob，在java端都使用String接收。

3、数值型

Integer 整数类型，小的整数。

Float 浮点数类型。

Real 实数类型。

Number（p,s）包含小数位的数值类型。P表示精度，s表示小数后的位数。

Eg: number(10,2) 表示小数点之前可有8位数字，小数点后有2位。

4、日期类型

Date 日期（日-月-年） DD-MM-YY(HH-MI-SS)

Timestamp 跟date比 它可以精确到微秒。精确范围0~9 默认为6.

#### 8、id、rowid、rownum的区别

rowid就是唯一标志记录物理位置的一个id。而id是唯一标识记录的逻辑上的标识，所以rowid速度要快于id,是目前最快的定位一条记录的方式

rowid和rownum都是"伪数列"

rownum用于标记结果集中结果顺序的一个字段，

这个字段被称为“伪数列”，也就是事实上不存在的一个数列。

它的特点是按顺序标记，而且是逐次递加的，

换句话说就是只有有rownum=1的记录，才可能有rownum=2的记录。

rownum关键字只能和<直接关联

如果是>或者=则需要给他起个别名

#### 9、主键和唯一索引的区别？

在创建主键的同时会生成对应的唯一索引，主键在保证数据唯一性的同时不允许为空，而唯一可以有一个为空数据项，一个表中只能有一个主键，但是一个主键可以有多个字段，一个表中可以有多个唯一索引。

#### 10、Prepared statement和statement的区别

用Prepared statement进行开发。Prepared statement是预编译的，而statement不是，在每次执行sql语句的增删改时，如果是一条数据两者没差距，但如果数据量大于1，那么每次执行sql语句statement都要重新编译一次，而Prepared statement不用，Prepared statement的运行效率大于statement；从代码的可维护性和可读性来说，虽然用Prepared statement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来说，都比直接使用statement的代码高很多档次；最重要的一点，从安全角度来说，使用Prepared statement可以打打提高程序的安全性，因为Prepared statement是用‘？’传参，具有安全性，而statement用的是‘+’字符串拼接，安全性较低。

#### 11、数据库三范式

第一范式：数据库表中的所有字段值都是不可分解的原子值。

第二范式：需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）

第三范式：需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关

#### 12、视图概述

视图可以视为“虚拟表”或“存储的查询”

创建视图所依据的表称为“基表”

视图的优点：

提供了另外一种级别的表安全性:隐藏了一些关键的字段

简化的用户的SQL命令

隔离基表结构的改变

#### 13、存储过程概述

存储过程（Stored Procedure）

可以包含逻辑判断的sql语句集合。

是经过预编译，存在于数据库中。

通过调用指定存储过程的名字（可有参，可无参）来执行。

优点：

简化了复杂的业务逻辑，根据需要可重复使用

屏蔽了底层细节，不暴露表信息即可完成操作

降低网络的通信量，多条语句可以封装成一个存储过程来执行

设置访问权限来提高安全性

提高执行效率，因为它是预编译以及存储在数据库中

缺点：

可移植性差，相同的存储过程并不能跨多个数据库进行操作

大量使用存储过程后，首先会使服务器压力增大，而且维护难度逐渐增加

存储过程的语法：

--下面是在oracle数据库下最基本的语法

--仅创建一个名为testProcedure 的无参的存储过程

--IS也可以是AS

--如果已经存在名为 testProcedure 的存储过程，下面的语法会出现 名称已被使用的错误

--解决办法：

--第一句可以写成 create or replace procedure testProcedure

--这样会替换原有的存储过程

--NULL表示任何可以正确执行的sql 语句，但至少一句

create procedure testProcedure IS BEGIN NULL END;

存储过程的参数的分类:

IN

OUT

INOUT

注意：

存储过程之间可相互调用

存储过程一般修改后，立即生效。

#### 14、索引概述

1、索引的概念

索引就是加快查询表中数据的方法。

数据库的索引类似于书籍的索引。

在书籍中，索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息。

在数据库中，索引也允许数据库程序迅速地找到表中的数据，

而不必扫描整个数据库.

2、索引的特点

　　1.索引可以加快数据库的检索速度

　　2.索引降低了数据库插入、修改、删除等维护任务的速度

　　3.索引创建在表上，不能创建在视图上

3、索引的优点

　　1.创建唯一性索引，保证数据库表中每一行数据的唯一性

　　2.大大加快数据的检索速度，这也是创建索引的最主要的原因

　　3.减少磁盘IO（向字典一样可以直接定位）

4、索引的缺点

1.创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加

2.索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定 的物理空间

3.当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，降低 了数据的维护速度

5、索引的分类

　　1.普通索引和唯一性索引

　　 普通索引：CREATE INDEX mycolumn\_index ON mytable (myclumn)

　　 唯一性索引：保证在索引列中的全部数据是唯一的

　　 CREATE unique INDEX mycolumn\_index ON mytable (myclumn)

　　2. 单个索引和复合索引

　　 单个索引：即非复合索引

　　 复合索引：又叫组合索引，在索引建立语句中同时包含多个字段名，

　　 最多16个字段

　　 CREATE INDEX name\_index ON username(firstname,lastname)

　　顺序索引，散列索引,位图索引

#### 15、必背的sql语句

一：oracle 分页

select \* from (select t.\*, rownum rn from (select \* from menu order by id desc) t where rownum < 10) where rn >=5

二: mysql 分页

select \* from music where id limit 5,5

三：oracle中如何快速将一张表的数据复制到另外一张表中(另外一张表不存在，另外一张 表存在，但数据为空)

1、.不存在另一张表时：

create table 新表 as select \* from 将要复制的表

2、存在另一张表时：

insert into 新表名 select 字段 from 将要复制的表名

四：快速删除一张表（不可事物回滚，也就是没有日志记录）

TRUNCATE from 表名

五：inner join

select 查找信息 from 表名 1 inner join 表名2 on 表名1.列名 = 表名2.列名

六：left join

左外连接 select 查找信息 from 表名1 left join 表名2 on 表名1.列名 = 表名2.列名

七：right join

右外连接 select 查找信息 from 表名1 right join 表名2 on 表名1.列名 = 表名2.列名

八：oracle中查询遍历树形结构（start with）

select \* from extmenu

start with pid=0

connect by prior id = pid

九：查询出来60-70,80-90,95-100学生的信息

select \* from stu where chengji between 60 and 70 or between 80 and 90 or between 95 and 100

select \* from stu where chengji > 60 and chengji < 70 or chengji > 80 and chengji < 90 or chengji > 95 and chengji < 100

十：用exists替换in------进行联表查询

select \* from dept where exists(select \* from emp where emp.deptno=dept.deptno);

或

select \* from dept d inner join emp e on d.deptno = e.deptno(只查询出两表共同拥有的字段数据)

十一：删除表中的重复数据：

delete from xin a where a.rowid != (

select max(b.rowid) from xin b

where a.name = b.name

);

#### 16、MySQL的复制原理以及流程

基本原理流程，3个线程以及之间的关联；

1. 主：binlog线程——记录下所有改变了数据库数据的语句，放进master上的binlog中；

2. 从：io线程——在使用start slave 之后，负责从master上拉取 binlog 内容，放进 自己的relay log中；

3. 从：sql执行线程——执行relay log中的语句；

#### 17、MySQL中myisam与innodb的区别，至少5点

(1)、问5点不同；

1>.InnoDB支持事物，而MyISAM不支持事物

2>.InnoDB支持行级锁，而MyISAM支持表级锁

3>.InnoDB支持MVCC, 而MyISAM不支持

4>.InnoDB支持外键，而MyISAM不支持

5>.InnoDB不支持全文索引，而MyISAM支持。

(2)、innodb引擎的4大特性

插入缓冲（insert buffer),二次写(double write),自适应哈希索引(ahi),预读(read ahead)

(3)、2者selectcount(\*)哪个更快，为什么

myisam更快，因为myisam内部维护了一个计数器，可以直接调取。

#### 18、MySQL中varchar与char的区别以及varchar(50)中的50代表的涵义

(1)、varchar与char的区别

char是一种固定长度的类型，varchar则是一种可变长度的类型

(2)、varchar(50)中50的涵义

最多存放50个字符，varchar(50)和(200)存储hello所占空间一样，但后者在排序时会消耗更多内存，因为order by col采用fixed\_length计算col长度(memory引擎也一样)

(3)、int（20）中20的涵义

是指显示字符的长度

但要加参数的，最大为255，比如它是记录行数的id,插入10笔资料，它就显示00000000001 ~~~00000000010，当字符的位数超过11,它也只显示11位，如果你没有加那个让它未满11位就前面加0的参数，它不会在前面加0

20表示最大显示宽度为20，但仍占4字节存储，存储范围不变；

(4)、mysql为什么这么设计对大多数应用没有意义，只是规定一些工具用来显示字符的个数；int(1)和int(20)存储和计算均一样；

#### 19、innodb的事务与日志的实现方式

(1)、有多少种日志；

错误日志：记录出错信息，也记录一些警告信息或者正确的信息。

查询日志：记录所有对数据库请求的信息，不论这些请求是否得到了正确的执行。

慢查询日志：设置一个阈值，将运行时间超过该值的所有SQL语句都记录到慢查询的日志文件中。

二进制日志：记录对数据库执行更改的所有操作。

中继日志：

事务日志：

(2)、事物的4种隔离级别

隔离级别

读未提交(RU)

读已提交(RC)

可重复读(RR)

串行

(3)、事务是如何通过日志来实现的，说得越深入越好。

事务日志是通过redo和innodb的存储引擎日志缓冲（Innodb log buffer）来实现的，当开始一个事务的时候，会记录该事务的lsn(log sequence number)号; 当事务执行时，会往InnoDB存储引擎的日志

的日志缓存里面插入事务日志；当事务提交时，必须将存储引擎的日志缓冲写入磁盘（通过innodb\_flush\_log\_at\_trx\_commit来控制），也就是写数据前，需要先写日志。这种方式称为“预写日志方式”

#### 20、MySQL binlog的几种日志录入格式以及区别

(1)、binlog的日志格式的种类和分别

(2)、适用场景；

(3)、结合第一个问题，每一种日志格式在复制中的优劣。

Statement：每一条会修改数据的sql都会记录在binlog中。

优点：不需要记录每一行的变化，减少了binlog日志量，节约了IO，提高性能。(相比row能节约多少性能 与日志量，这个取决于应用的SQL情况，正常同一条记录修改或者插入row格式所产生的日志量还小于Statement产生的日志量，但是考虑到如果带条 件的update操作，以及整表删除，alter表等操作，ROW格式会产生大量日志，因此在考虑是否使用ROW格式日志时应该跟据应用的实际情况，其所 产生的日志量会增加多少，以及带来的IO性能问题。)

缺点：由于记录的只是执行语句，为了这些语句能在slave上正确运行，因此还必须记录每条语句在执行的时候的 一些相关信息，以保证所有语句能在slave得到和在master端执行时候相同 的结果。另外mysql 的复制,像一些特定函数功能，slave可与master上要保持一致会有很多相关问题(如sleep()函数， last\_insert\_id()，以及user-defined functions(udf)会出现问题).

使用以下函数的语句也无法被复制：

\* LOAD\_FILE()

\* UUID()

\* USER()

\* FOUND\_ROWS()

\* SYSDATE() (除非启动时启用了 --sysdate-is-now 选项)

同时在INSERT ...SELECT 会产生比 RBR 更多的行级锁

2.Row:不记录sql语句上下文相关信息，仅保存哪条记录被修改。

优点： binlog中可以不记录执行的sql语句的上下文相关的信息，仅需要记录那一条记录被修改成什么了。所以rowlevel的日志内容会非常清楚的记录下 每一行数据修改的细节。而且不会出现某些特定情况下的存储过程，或function，以及trigger的调用和触发无法被正确复制的问题

缺点:所有的执行的语句当记录到日志中的时候，都将以每行记录的修改来记录，这样可能会产生大量的日志内容,比 如一条update语句，修改多条记录，则binlog中每一条修改都会有记录，这样造成binlog日志量会很大，特别是当执行alter table之类的语句的时候，由于表结构修改，每条记录都发生改变，那么该表每一条记录都会记录到日志中。

3.Mixedlevel: 是以上两种level的混合使用，一般的语句修改使用statment格式保存binlog，如一些函数，statement无法完成主从复制的操作，则 采用row格式保存binlog,MySQL会根据执行的每一条具体的sql语句来区分对待记录的日志形式，也就是在Statement和Row之间选择 一种.新版本的MySQL中队row level模式也被做了优化，并不是所有的修改都会以row level来记录，像遇到表结构变更的时候就会以statement模式来记录。至于update或者delete等修改数据的语句，还是会记录所有行的 变更。

#### 21、MySQL数据库cpu飙升到500%的话他怎么处理？

(1)、没有经验的，可以不问；

(2)、有经验的，问他们的处理思路。

列出所有进程 show processlist 观察所有进程 多秒没有状态变化的(干掉)

查看超时日志或者错误日志 (做了几年开发,一般会是查询以及大批量的插入会导致cpu与i/o上涨,,,,当然不排除网络状态突然断了,,导致一个请求服务器只接受到一半，比如where子句或分页子句没有发送,,当然的一次被坑经历)

#### 22、数据库中常用术语：

ddl:数据定义语言 Create Drop Alter

dml:数据操纵语言 insert update delete select

dcl:数据控制语言 grant revoke

tcl:事务控制语言 commit rollback

#### 23、备份计划，mysqldump以及xtranbackup的实现原理

(1)、备份计划；

这里每个公司都不一样，您别说那种1小时1全备什么的就行

(2)、备份恢复时间；

这里跟机器，尤其是硬盘的速率有关系，以下列举几个仅供参考

20G的2分钟（mysqldump）

80G的30分钟(mysqldump)

111G的30分钟（mysqldump)

288G的3小时（xtra)

3T的4小时（xtra)

逻辑导入时间一般是备份时间的5倍以上

(3)、xtrabackup实现原理

在InnoDB内部会维护一个redo日志文件，我们也可以叫做事务日志文件。事务日志会存储每一个InnoDB表数据的记录修改。当InnoDB启动时，InnoDB会检查数据文件和事务日志，并执行两个步骤：它应用（前滚）已经提交的事务日志到数据文件，并将修改过但没有提交的数据进行回滚操作。

#### 24、MySQL中InnoDB引擎的行锁是通过加在什么上完成(或称实现)的？为什么是这样子的？

InnoDB是基于索引来完成行锁

例: select \* from tab\_with\_index where id = 1 for update;

for update 可以根据条件来完成行锁锁定,并且 id 是有索引键的列,

如果 id 不是索引键那么InnoDB将完成表锁,,并发将无从谈起

#### 25、说出数据连接池的工作机制是什么?

J2EE 服务器启动时会建立一定数量的池连接，并一直维持不少于此数目的池连接。客户端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量由配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。

#### 26、事务概述

在数据库中,所谓事务是指一组逻辑操作单元即一组sql语句。当这个单元中的一部分操作失败,整个事务回滚，只有全部正确才完成提交。

判断事务是否配置成功的**关键点**在于出现异常时事务是否会回滚

事务的ACID属性

1. 原子性（Atomicity）

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，

要么都不发生。

2. 一致性（Consistency）

事务必须使数据库从一个一致性状态变换到另外一个一致性状态。(数据不被破坏)

3. 隔离性（Isolation）

事务的隔离性是指一个事务的执行不能被其他事务干扰.

4. 持久性（Durability）

持久性是指一个事务一旦被提交，

它对数据库中数据的改变就是永久性的.即使系统重启也不会丢失.

在JDBC中，

事务默认是自动提交的，

每次执行一个 SQL 语句时，如果执行成功，

就会向数据库自动提交，而不能回滚

为了让多个 SQL 语句作为一个事务执行：

（1）执行语句前调用 Connection 对象的 setAutoCommit(false);

以取消自动提交事务

（2）在所有的 SQL 语句都成功执行后，调用 commit(); 方法提交事务

（3）在出现异常时，调用 rollback(); 方法回滚事务。

## 三 JavaWeb阶段

#### 1、Servlet的生命周期，以及它和Tomcat怎么结合工作的 工作原理？

Servlet 加载—>实例化—>服务—>销毁。

1.生命周期详解：

init（）：

在Servlet的生命周期中，仅执行一次init()方法。它是在服务器装入Servlet时执行的，负责初始化Servlet对象。可以配置服务器，以在启动服务器或客户机首次访问Servlet时装入Servlet。无论有多少客户机访问Servlet，都不会重复执行init（）。

service（）：

它是Servlet的核心，负责响应客户的请求。每当一个客户请求一个HttpServlet对象，该对象的Service()方法就要调用，而且传递给这个方法一个“请求”（ServletRequest）对象和一个“响应”（ServletResponse）对象作为参数。在HttpServlet中已存在Service()方法。默认的服务功能是调用与HTTP请求的方法相应的do功能。

destroy（）：

仅执行一次，在服务器端停止且卸载Servlet时执行该方法。当Servlet对象退出生命周期时，负责释放占用的资源。一个Servlet在运行service()方法时可能会产生其他的线程，因此需要确认在调用destroy()方法时，这些线程已经终止或完成。

2.如何与Tomcat 结合工作步骤：

1）. Web Client 向Servlet容器（Tomcat）发出Http请求

2）. Servlet容器接收Web Client的请求

3）. Servlet容器创建一个HttpRequest对象，将Web Client请求的信息封装到这个对象中。

4）. Servlet容器创建一个HttpResponse对象

5）. Servlet容器调用HttpServlet对象的service方法，把HttpRequest对象与HttpResponse对象作为参数传给HttpServlet 对象。

6）. HttpServlet调用HttpRequest对象的有关方法，获取Http请求信息。

7）. HttpServlet调用HttpResponse对象的有关方法，生成响应数据。

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

JSP 是Servlet技术的扩展，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侧重于视图，Servlet主要用于控制逻辑。

#### 3、四种会话跟踪技术

会话作用域ServletsJSP 页面描述

page否是代表与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（可以带有任何的 include 指令，但是没有 include 动作）表示。这既包括 servlet 又包括被编译成 servlet 的 JSP 页面

request是是代表与 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 forward 指令和 include 动作的关系）

session是是代表与用于某个 Web 客户机的一个用户体验相关的对象和属性。一个 Web 会话可以也经常会跨越多个客户机请求

application是是代表与整个 Web 应用程序相关的对象和属性。这实质上是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域

#### 4、forward 和 redirect 的区别?

forward 是容器中控制权的转向，是服务器请求资源，服务器直接访问目标 地址的 URL，把那个 URL 的响应内容读取过来，然后把这些内容再发给浏览器， 浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来 的地址。 redirect 就是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请 求那个地址，一般来说浏览器会用刚才请求的所有参数重新请求，所以 session,request 参数都可以获取，并且从浏览器的地址栏中可以看到跳转后的 链接地址。前者更加高效，在前者可以满足需要时，尽量使用 forward()方法， 并且，这样也有助于隐藏实际的链接；在有些情况下，比如，需要跳转到一个其 它服务器上的资源，则必须使用 sendRedirect()方法。

#### 5、JSP 中动态 INCLUDE 与静态 INCLUDE 的区别？

动态 INCLUDE用jsp:include动作实现 <jsp:include page="included.jsp" flush="true" />它总是会检查所含文件中的变化，适合用于包含动态页面，并 且可以带参数；

静态 INCLUDE 用 include 伪码实现,它不会检查所含文件的变化， 适用于包含静态页面<%@ include file="included.htm" %>

#### 6、jsp 有哪些内置对象?作用分别是什么?

JSP 共有以下 9 种基本内置组件（可与 ASP 的 6 种内部组件相对应）：

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

response：网页传回用户端的回应；

pageContext：网页的属性是在这里管理；

session：与请求有关的会话期；

application：servlet 正在执行的内容；

out：用来传送回应的输出；

config：servlet 的构架部件；

page：JSP 网页本身；

exception：针对错误网页，未捕捉的例外。

#### 7、get 和 post 的区别？

Form 中的 get 和 post 方法，在数据传输过程中分别对应了 HTTP 协议中的 GET 和 POST 方法。

二者主要区别如下：

1）Get 是用来从服务器上获得数据，而 Post 是用来向服务器上传递数据；

2）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，并且两者使用“?”连接，而各个变量之间使用“&”连接；Post 是将表单中的数据放在form 的数据体中，按照变量和值相对应的方式，传递到action 所指向 URL；

3）Get 是不安全的，因为在传输过程，数据被放在请求的 URL 中；Post 的 所有操作对用户来说都是不可见的；

4）Get 传输的数据量小，这主要是因为受 URL 长度限制；而 Post 可以传输 大量的数据，所以在上传文件只能使用 Post；

5）Get限制Form表单的数据集必须为ASCII字符，而 Post支持整个 ISO10646 字符集；

6）Get 是 Form 的默认方法。

#### 8、jsp 的四种范围？

1.page 是代表与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（可以带有任何的 include 指令，但是没有 include 动作） 表示。这既包括 servlet 又包括被编译成 servlet 的 JSP 页面

2.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 forward 指令和 include 动 作的关系）

3.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 会话可以也经常会跨越多个客户机请求

4.application 是代表与整个 Web 应用程序相关的对象和属性。这实质上 是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域。

#### 9、javascript 的优缺点和内置对象

1）优点：简单易用，与 Java 有类似的语法，可以使用任何文本编辑工具编 写，只需要浏览器就可执行程序，并且事先不用编译，逐行执行，无需进行严格 的变量声明，而且内置大量现成对象，编写少量程序可以完成目标；

2）缺点：不适合开发大型应用程序；

3）Javascript 有 11 种内置对象： Array、String、Date、Math、Boolean、Number、 Function、Global、Error、RegExp、Object

#### 10、tomcat Apache Jboss 有什么区别？

Tomcat是Apache鼎力支持的Java Web应用服务器，由于它优秀的稳定性以及丰富的文档资料，广泛的使用人群，从而在开源领域受到最广泛的青睐。¬

实际上Tomcat 部分是Apache 服务 器的扩展，但它是独立运行的，所以当你运行tomcat 时，它实际上作为一个与Apache 独立的进 程单独运行的。

Apache全球应用最广泛的http服务器，免费，出自apache基金组织.apache仅有的不足是，它本身仅提供html静态页面的功能，不能支持jsp、java servlet、asp等功能，

但通过同其他应用服务器一起工作或添加插件来支持。（稳定高效，通常需要与其他服务器软件或插件协同工作）

Jboss作为Java EE应用服务器，它不但是Servlet容器，而且是EJB容器（同时支持EJB的优化版tomcat），从而受到企业级开发人员的欢迎，从而弥补了Tomcat只是一个Servlet容器的缺憾。¬

#### 11、请简述Session 和 Cookie 的区别和联系，如何实现用户的自动登录

区别：cookie数据保存在客户端，session数据保存在服务器端。

详解：

简单的说，当你登录一个网站的时候，如果web服务器端使用的是session,那么所有的数据都保存在服务器上面，客户端每次请求服务器的时候会发送 当前会话的sessionid，服务器根据当前sessionid判断相应的用户数据标志，以确定用户是否登录，或具有某种权限。由于数据是存储在服务器 上面，所以你不能伪造，但是如果你能够获取某个登录用户的sessionid，用特殊的浏览器伪造该用户的请求也是能够成功的。sessionid是服务 器和客户端链接时候随机分配的，一般来说是不会有重复，但如果有大量的并发请求，也不是没有重复的可能性，我曾经就遇到过一次。登录某个网站，开始显示的 是自己的信息，等一段时间超时了，一刷新，居然显示了别人的信息。

联系: Cookies是属于Session对象的一种。但有不同，Cookies不会占服务器资源，是存在客服端内存或者一个cookie的文本文件中；而“Session”则会占用服务器资源。所以，尽量不要使用Session，而使用Cookies。但是我们一般认为cookie是不可靠的，session是可靠地，但是目前很多著名的站点也都以来cookie。有时候为了解决禁用cookie后的页面处理，通常采用url重写技术，调用session中大量有用的方法从session中获取数据后置入页面

如何实现用户的自动登录：

首先，在项目默认页index.jsp，进行重定向操作至logincookie.action。

在logincookie.action方法中，获取当前的cookie和session，如果cookie中存在用户信息（username）且session中存放有已登录用户的信息（logicId，username，logintime等），则自动跳转至登录成功页面(action跳转)，否则进入用户登录页面。值得一提的是，当用户拦截器拦截非法注入等操作的时候，也将跳回index.jsp，这时的sessionId发生变化，需先清除loginUserMap并进入登录页。当用户登录时，新建一个cookie，将username作为一个属性存放在cookie中，设置cookie的失效时间，然后将sessionId存放到cookie的JsessionId属性中，添加cookie。当用户执行安全退出时，清除cookie。这样就能实现session结合cookie的自动登录。

## 四 常见框架阶段

### 4.1 Spring

#### 1、请写出 spring 中 I0C 的三种实现机制。

三种机制为：通过 setter 方法注入、通过构造方法注入和接口注入

#### 2. Spring支持的事务管理类型有哪些？你在项目中使用哪种方式？

Spring支持编程式事务管理和声明式事务管理。许多Spring框架的用户选择声明式事务管理，因为这种方式和应用程序的关联较少，因此更加符合轻量级容器的概念。声明式事务管理要优于编程式事务管理，尽管在灵活性方面它弱于编程式事务管理，因为编程式事务允许你通过代码控制业务。

事务分为全局事务和局部事务。全局事务由应用服务器管理，需要底层服务器JTA支持（如WebLogic、WildFly等）。局部事务和底层采用的持久化方案有关，例如使用JDBC进行持久化时，需要使用Connetion对象来操作事务；而采用Hibernate进行持久化时，需要使用Session对象来操作事务。

Spring提供了如下所示的事务管理器。

事务管理器实现类 目标对象

DataSourceTransactionManager 注入DataSource

HibernateTransactionManager 注入SessionFactory

JdoTransactionManager 管理JDO事务

JtaTransactionManager 使用JTA管理事务

PersistenceBrokerTransactionManager 管理Apache的OJB事务

这些事务的父接口都是PlatformTransactionManager。Spring的事务管理机制是一种典型的策略模式，PlatformTransactionManager代表事务管理接口，该接口定义了三个方法，该接口并不知道底层如何管理事务，但是它的实现类必须提供getTransaction()方法（开启事务）、commit()方法（提交事务）、rollback()方法（回滚事务）的多态实现，这样就可以用不同的实现类代表不同的事务管理策略。使用JTA全局事务策略时，需要底层应用服务器支持，而不同的应用服务器所提供的JTA全局事务可能存在细节上的差异，因此实际配置全局事务管理器是可能需要使用JtaTransactionManager的子类，如：WebLogicJtaTransactionManager（Oracle的WebLogic服务器提供）、UowJtaTransactionManager（IBM的WebSphere服务器提供）等。

#### 3、简述Spring 的理解，对IOC和AOP的理解及实现原理，以及在项目中你们是如何应用的？

spring概述：

首先Spring是一个大的概念，Spring从最开始的一个Library到现在一个系列，其中最主要的包括Spring Framework, Spring Data, Spring Security, Spring Batch等等，以及快速框架Spring Boot，其中最重要的项目是Spring Framework，这个项目包括了IoC, AOP, MVC以及Testing。

第一个需要明白的是Spring的核心思想是什么？

Spring整个系列的最最核心的概念当属IoC, AOP，什么是IoC和AOP就不展开了，简而言之，将对象创建过程的职责赋予容器，通过容器管理对象的生老病死， 将对象创建过程从编译时延期到运行时，即通过配置进行加载，这样一来就解决了不用编译后期选择具体实现，其实就是面向对象的核心理念，针对接口编程。IoC开始就是个factory加上依赖管理罢了，这样一来，一个系统的创建过程就从原先的new改为配置组装，内部通过注入解决了依赖关系，只要满足接口协议即插即用。通过IoC, AOP事实上形成了一个套路，通过这个套路完成了系统的整合。

所以Spring并没有说自己写一个ORM，而是用统一的套路完成了多个ORM的集成，这也是Spring越做越大的基础，慢慢就形成了Spring Way，其实这个才是Spring最有价值的地方。

IOC和AOP的理解：

IOC的作用：产生对象实例，所以它是基于工厂设计模式的

Spring IOC的注入

通过属性进行注入，通过构造函数进行注入，

注入对象数组 注入List集合

注入Map集合 注入Properties类型

Spring IOC 自动绑定模式：

可以设置autowire按以下方式进行绑定

按byType只要类型一致会自动寻找，

按byName自动按属性名称进行自动查找匹配.

AOP 面向方面（切面）编程：

AOP是OOP的延续，是Aspect Oriented Programming的缩写，

意思是面向方面(切面)编程。

注：OOP(Object-Oriented Programming ) 面向对象编程

AOP 主要应用于日志记录，性能统计，安全控制,事务处理（项目中使用的）等方面。

Spring中实现AOP技术：

在Spring中可以通过代理模式来实现AOP

代理模式分为

静态代理：一个接口，分别有一个真实实现和一个代理实现。

动态代理：通过代理类的代理，接口和实现类之间可以不直接发生联系，而可以在运行期（Runtime）实现动态关联。

动态代理有两种实现方式，可以通过jdk的动态代理实现也可以通过cglib

来实现而AOP默认是通过jdk的动态代理来实现的。jdk的动态代理必须要有

接口的支持，而cglib不需要，它是基于类的。

#### 4、请简单介绍Spring支持的常用数据库事务传播属性和事务隔离级别

1 事务的传播属性（Propagation）

1) REQUIRED ，这个是默认的属性

Support a current transaction, create a new one if none exists.

如果存在一个事务，则支持当前事务。如果没有事务则开启一个新的事务。

被设置成这个级别时，会为每一个被调用的方法创建一个逻辑事务域。如果前面的方法已经创建了事务，那么后面的方法支持当前的事务，如果当前没有事务会重新建立事务。

2) MANDATORY

Support a current transaction, throw an exception if none exists.支持当前事务，如果当前没有事务，就抛出异常。

3) NEVER

Execute non-transactionally, throw an exception if a transaction exists.

以非事务方式执行，如果当前存在事务，则抛出异常。

4) NOT\_SUPPORTED

Execute non-transactionally, suspend the current transaction if one exists.

以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

5) REQUIRES\_NEW

Create a new transaction, suspend the current transaction if one exists.

新建事务，如果当前存在事务，把当前事务挂起。

6) SUPPORTS

Support a current transaction, execute non-transactionally if none exists.

支持当前事务，如果当前没有事务，就以非事务方式执行。

7) NESTED

Execute within a nested transaction if a current transaction exists, behave like PROPAGATION\_REQUIRED else.

支持当前事务，新增Savepoint点，与当前事务同步提交或回滚。

嵌套事务一个非常重要的概念就是内层事务依赖于外层事务。外层事务失败时，会回滚内层事务所做的动作。而内层事务操作失败并不会引起外层事务的回滚。

8) PROPAGATION\_NESTED 与PROPAGATION\_REQUIRES\_NEW的区别

它们非常 类似,都像一个嵌套事务，如果不存在一个活动的事务，都会开启一个新的事务。使用PROPAGATION\_REQUIRES\_NEW时，内层事务与外层事务就像两个独立的事务一样，一旦内层事务进行了提交后，外层事务不能对其进行回滚。两个事务互不影响。两个事务不是一个真正的嵌套事务。同时它需要JTA 事务管理器的支持。

使用PROPAGATION\_NESTED时，外层事务的回滚可以引起内层事务的回滚。而内层事务的异常并不会导致外层事务的回滚，它是一个真正的嵌套事务。

2.事务的隔离级别

1) 首先说明一下事务并发引起的三种情况

i. Dirty Reads 脏读

一个事务正在对数据进行更新操作，但是更新还未提交，另一个事务这时也来操作这组数据，并且读取了前一个事务还未提交的数据，而前一个事务如果操作失败进行了回滚，后一个事务读取的就是错误数据，这样就造成了脏读。

ii. Non-Repeatable Reads 不可重复读

一个事务多次读取同一数据，在该事务还未结束时，另一个事务也对该数据进行了操作，而且在第一个事务两次次读取之间，第二个事务对数据进行了更新，那么第一个事务前后两次读取到的数据是不同的，这样就造成了不可重复读。

iii. Phantom Reads 幻像读

第一个数据正在查询符合某一条件的数据，这时，另一个事务又插入了一条符合条件的数据，第一个事务在第二次查询符合同一条件的数据时，发现多了一条前一次查询时没有的数据，仿佛幻觉一样，这就是幻像读。

iv. 非重复度和幻像读的区别

非重复读是指同一查询在同一事务中多次进行，由于其他提交事务所做的修改或删除，每次返回不同的结果集，此时发生非重复读。(A transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data. )

幻像读是指同一查询在同一事务中多次进行，由于其他提交事务所做的插入操作，每次返回不同的结果集，此时发生幻像读。(A transaction reexecutes a query returning a set of rows that satisfies a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition. )

表面上看，区别就在于非重复读能看见其他事务提交的修改和删除，而幻像能看见其他事务提交的插入。

2) DEFAULT （默认）

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

3) READ\_UNCOMMITTED （读未提交）

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。

4) READ\_COMMITTED （读已提交）

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。这种事务隔离级别可以避免脏读出现，但是可能会出现不可重复读和幻像读。

5) REPEATABLE\_READ （可重复读）

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，还保证了不可重复读

6) SERIALIZABLE（串行化）

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。

#### 5、什么是Spring框架？Spring框架有哪些主要模块？

Spring框架是一个为Java应用程序的开发提供了综合、广泛的基础性支持的Java平台。Spring帮助开发者解决了开发中基础性的问题，使得开发人员可以专注于应用程序的开发。Spring框架本身亦是按照设计模式精心打造，这使得我们可以在开发环境中安心的集成Spring框架，不必担心Spring是如何在后台进行工作的。

Spring框架至今已集成了20多个模块。这些模块主要被分如下图所示的核心容器、数据访问/集成,、Web、AOP（面向切面编程）、工具、消息和测试模块。

#### 6、使用Spring框架能带来哪些好处？

下面列举了一些使用Spring框架带来的主要好处：

Dependency Injection(DI) 方法使得构造器和JavaBean properties文件中的依赖关系一目了然。

与EJB容器相比较，IoC容器更加趋向于轻量级。这样一来IoC容器在有限的内存和CPU资源的情况下进行应用程序的开发和发布就变得十分有利。

Spring并没有闭门造车，Spring利用了已有的技术比如ORM框架、logging框架、J2EE、Quartz和JDK Timer，以及其他视图技术。

Spring框架是按照模块的形式来组织的。由包和类的编号就可以看出其所属的模块，开发者仅仅需要选用他们需要的模块即可。

要测试一项用Spring开发的应用程序十分简单，因为测试相关的环境代码都已经囊括在框架中了。更加简单的是，利用JavaBean形式的POJO类，可以很方便的利用依赖注入来写入测试数据。

Spring的Web框架亦是一个精心设计的Web MVC框架，为开发者们在web框架的选择上提供了一个除了主流框架比如Struts、过度设计的、不流行web框架的以外的有力选项。

Spring提供了一个便捷的事务管理接口，适用于小型的本地事物处理（比如在单DB的环境下）和复杂的共同事物处理（比如利用JTA的复杂DB环境）。

#### 7、什么是控制反转(IOC)？什么是依赖注入？

控制反转是应用于软件工程领域中的，在运行时被装配器对象来绑定耦合对象的一种编程技巧，对象之间耦合关系在编译时通常是未知的。在传统的编程方式中，业务逻辑的流程是由应用程序中的早已被设定好关联关系的对象来决定的。在使用控制反转的情况下，业务逻辑的流程是由对象关系图来决定的，该对象关系图由装配器负责实例化，这种实现方式还可以将对象之间的关联关系的定义抽象化。而绑定的过程是通过“依赖注入”实现的。

控制反转是一种以给予应用程序中目标组件更多控制为目的设计范式，并在我们的实际工作中起到了有效的作用。

依赖注入是在编译阶段尚未知所需的功能是来自哪个的类的情况下，将其他对象所依赖的功能对象实例化的模式。这就需要一种机制用来激活相应的组件以提供特定的功能，所以依赖注入是控制反转的基础。否则如果在组件不受框架控制的情况下，框架又怎么知道要创建哪个组件？

在Java中依然注入有以下三种实现方式：

构造器注入

Setter方法注入

接口注入

#### 8、请解释下Spring框架中的IOC？

Spring中的 org.springframework.beans 包和 org.springframework.context包构成了Spring框架IoC容器的基础。

BeanFactory 接口提供了一个先进的配置机制，使得任何类型的对象的配置成为可能。ApplicationContex接口对BeanFactory（是一个子接口）进行了扩展，在BeanFactory的基础上添加了其他功能，比如与Spring的AOP更容易集成，也提供了处理message resource的机制（用于国际化）、事件传播以及应用层的特别配置，比如针对Web应用的WebApplicationContext。

org.springframework.beans.factory.BeanFactory 是Spring IoC容器的具体实现，用来包装和管理前面提到的各种bean。BeanFactory接口是Spring IoC 容器的核心接口。

#### 9、BeanFactory和ApplicationContext有什么区别？

BeanFactory 可以理解为含有bean集合的工厂类。BeanFactory 包含了种bean的定义，以便在接收到客户端请求时将对应的bean实例化。

BeanFactory还能在实例化对象的时生成协作类之间的关系。此举将bean自身与bean客户端的配置中解放出来。BeanFactory还包含了bean生命周期的控制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。

从表面上看，application context如同bean factory一样具有bean定义、bean关联关系的设置，根据请求分发bean的功能。但application context在此基础上还提供了其他的功能。

提供了支持国际化的文本消息

统一的资源文件读取方式

已在监听器中注册的bean的事件

以下是三种较常见的 ApplicationContext 实现方式：

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。

#### 10、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

#### 11、如何用基于XML配置的方式配置Spring？

在Spring框架中，依赖和服务需要在专门的配置文件来实现，我常用的XML格式的配置文件。这些配置文件的格式通常用<beans>开头，然后一系列的bean定义和专门的应用配置选项组成。

SpringXML配置的主要目的时候是使所有的Spring组件都可以用xml文件的形式来进行配置。这意味着不会出现其他的Spring配置类型（比如声明的方式或基于Java Class的配置方式）

Spring的XML配置方式是使用被Spring命名空间的所支持的一系列的XML标签来实现的。Spring有以下主要的命名空间：context、beans、jdbc、tx、aop、mvc和aso。

<beans> <!-- JSON Support -->

<bean name="viewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver"/> <bean name="jsonTemplate" class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"/> <bean id="restTemplate" class="org.springframework.web.client.RestTemplate"/>

</beans>

下面这个web.xml仅仅配置了DispatcherServlet，这件最简单的配置便能满足应用程序配置运行时组件的需求。

<web-app> <display-name>Archetype Created Web Application</display-name> <servlet> <servlet-name>spring</servlet-name> <servlet-class> org.springframework.web.servlet.DispatcherServlet </servlet-class> <load-on-startup>1</load-on-startup> </servlet> <servlet-mapping> <servlet-name>spring</servlet-name> <url-pattern>/</url-pattern> </servlet-mapping> </web-app>

#### 12、如何用基于Java配置的方式配置Spring？

Spring对Java配置的支持是由@Configuration注解和@Bean注解来实现的。由@Bean注解的方法将会实例化、配置和初始化一个新对象，这个对象将由Spring的IoC容器来管理。@Bean声明所起到的作用与<bean/> 元素类似。被@Configuration所注解的类则表示这个类的主要目的是作为bean定义的资源。被@Configuration声明的类可以通过在同一个类的内部调用@bean方法来设置嵌入bean的依赖关系。

最简单的@Configuration 声明类请参考下面的代码：

@Configuration

public class AppConfig {

@Bean

public MyService myService() {

return new MyServiceImpl();

} }

对于上面的@Beans配置文件相同的XML配置文件如下：

<beans>

<bean id="myService" class="com.howtodoinjava.services.MyServiceImpl"/>

</beans>

上述配置方式的实例化方式如下：利用AnnotationConfigApplicationContext 类进行实例化

public static void main(String[] args) {

ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class); MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

要使用组件组建扫描，仅需用@Configuration进行注解即可：

@Configuration @ComponentScan(basePackages = "com.howtodoinjava")

public class AppConfig { ... }

在上面的例子中，com.acme包首先会被扫到，然后再容器内查找被@Component 声明的类，找到后将这些类按照Sring bean定义进行注册。

如果你要在你的web应用开发中选用上述的配置的方式的话，需要用AnnotationConfigWebApplicationContext 类来读取配置文件，可以用来配置Spring的Servlet监听器ContrextLoaderListener或者Spring MVC的DispatcherServlet。

#### 13、怎样用注解的方式配置Spring？

Spring在2.5版本以后开始支持用注解的方式来配置依赖注入。可以用注解的方式来替代XML方式的bean描述，可以将bean描述转移到组件类的内部，只需要在相关类上、方法上或者字段声明上使用注解即可。注解注入将会被容器在XML注入之前被处理，所以后者会覆盖掉前者对于同一个属性的处理结果。

注解装配在Spring中是默认关闭的。所以需要在Spring文件中配置一下才能使用基于注解的装配模式。如果你想要在你的应用程序中使用关于注解的方法的话，请参考如下的配置。

<beans> <context:annotation-config/> <!-- bean definitions go here --> </beans>

在 <context:annotation-config/>标签配置完成以后，就可以用注解的方式在Spring中向属性、方法和构造方法中自动装配变量。

下面是几种比较重要的注解类型：

@Required：该注解应用于设值方法。

@Autowired：该注解应用于有值设值方法、非设值方法、构造方法和变量。

@Qualifier：该注解和@Autowired注解搭配使用，用于消除特定bean自动装配的歧义。

JSR-250 Annotations：Spring支持基于JSR-250 注解的以下注解，@Resource、@PostConstruct 和 @PreDestroy。

#### 14、请解释Spring Bean的生命周期？

Spring Bean的生命周期简单易懂。在一个bean实例被初始化时，需要执行一系列的初始化操作以达到可用的状态。同样的，当一个bean不在被调用时需要进行相关的析构操作，并从bean容器中移除。

Spring bean factory 负责管理在spring容器中被创建的bean的生命周期。Bean的生命周期由两组回调（call back）方法组成。

初始化之后调用的回调方法。

销毁之前调用的回调方法。

Spring框架提供了以下四种方式来管理bean的生命周期事件：

InitializingBean和DisposableBean回调接口

针对特殊行为的其他Aware接口

Bean配置文件中的Custom init()方法和destroy()方法

@PostConstruct和@PreDestroy注解方式

使用customInit()和 customDestroy()方法管理bean生命周期的代码样例如下：

<beans> <bean id="demoBean" class="com.howtodoinjava.task.DemoBean" init-method="customInit" destroy-method="customDestroy"></bean> </beans>

#### 15、Spring Bean的作用域之间有什么区别？

Spring容器中的bean可以分为5个范围。所有范围的名称都是自说明的，但是为了避免混淆，还是让我们来解释一下：

singleton：这种bean范围是默认的，这种范围确保不管接受到多少个请求，每个容器中只有一个bean的实例，单例的模式由bean factory自身来维护。

prototype：原形范围与单例范围相反，为每一个bean请求提供一个实例。

request：在请求bean范围内会每一个来自客户端的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。

Session：与请求范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效。

global-session：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。

#### 16、什么是Spring inner beans？

在Spring框架中，无论何时bean被使用时，当仅被调用了一个属性。一个明智的做法是将这个bean声明为内部bean。内部bean可以用setter注入“属性”和构造方法注入“构造参数”的方式来实现。

比如，在我们的应用程序中，一个Customer类引用了一个Person类，我们的要做的是创建一个Person的实例，然后在Customer内部使用。

public class Customer {

private Person person; //Setters and Getters

}

public class Person {

private String name;

private String address;

private int age; //Setters and Getters

}

内部bean的声明方式如下：

<bean id="CustomerBean" class="com.howtodoinjava.common.Customer">

<property name="person">

<!-- This is inner bean -->

<bean class="com.howtodoinjava.common.Person">

<property name="name" value="lokesh" />

<property name="address" value="India" />

<property name="age" value="34" />

</bean>

</property>

</bean>

#### 17、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”。

#### 18、请举例说明如何在Spring中注入一个Java Collection？

Spring提供了以下四种集合类的配置元素：

<list> : 该标签用来装配可重复的list值。

<set> : 该标签用来装配没有重复的set值。

<map>: 该标签可用来注入键和值可以为任何类型的键值对。

<props> : 该标签支持注入键和值都是字符串类型的键值对。

下面看一下具体的例子：

<beans>

<!-- Definition for javaCollection -->

<bean id="javaCollection" class="com.howtodoinjava.JavaCollection">

<!-- java.util.List -->

<property name="customList">

<list>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>UK</value>

</list>

</property>

<!-- java.util.Set -->

<property name="customSet">

<set>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>UK</value>

</set>

</property>

<!-- java.util.Map -->

<property name="customMap">

<map>

<entry key="1" value="INDIA"/>

<entry key="2" value="Pakistan"/>

<entry key="3" value="USA"/>

<entry key="4" value="UK"/>

</map>

</property>

<!-- java.util.Properties -->

<property name="customProperies">

<props>

<prop key="admin">admin@nospam.com</prop>

<prop key="support">support@nospam.com</prop>

</props>

</property>

</bean>

</beans>

#### 19、如何向Spring Bean中注入一个Java.util.Properties？

第一种方法是使用如下面代码所示的<props> 标签：

<bean id="adminUser" class="com.howtodoinjava.common.Customer">

<!-- java.util.Properties -->

<property name="emails">

<props>

<prop key="admin">admin@nospam.com</prop>

<prop key="support">support@nospam.com</prop>

</props>

</property>

</bean>

也可用”util:”命名空间来从properties文件中创建出一个propertiesbean，然后利用setter方法注入bean的引用。

#### 20、请解释Spring Bean的自动装配？

在Spring框架中，在配置文件中设定bean的依赖关系是一个很好的机制，Spring容器还可以自动装配合作关系bean之间的关联关系。这意味着Spring可以通过向Bean Factory中注入的方式自动搞定bean之间的依赖关系。自动装配可以设置在每个bean上，也可以设定在特定的bean上。

下面的XML配置文件表明了如何根据名称将一个bean设置为自动装配：

<bean id="employeeDAO" class="com.howtodoinjava.EmployeeDAOImpl" autowire="byName" />

除了bean配置文件中提供的自动装配模式，还可以使用@Autowired注解来自动装配指定的bean。在使用@Autowired注解之前需要在按照如下的配置方式在Spring配置文件进行配置才可以使用。

<context:annotation-config />

也可以通过在配置文件中配置AutowiredAnnotationBeanPostProcessor 达到相同的效果。

<bean

class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

配置好以后就可以使用@Autowired来标注了。

@Autowired

public EmployeeDAOImpl ( EmployeeManager manager ) {

this.manager = manager;

}

#### 21、请解释自动装配模式的区别？

在Spring框架中共有5种自动装配，让我们逐一分析。

no：这是Spring框架的默认设置，在该设置下自动装配是关闭的，开发者需要自行在bean定义中用标签明确的设置依赖关系。

byName：该选项可以根据bean名称设置依赖关系。当向一个bean中自动装配一个属性时，容器将根据bean的名称自动在在配置文件中查询一个匹配的bean。如果找到的话，就装配这个属性，如果没找到的话就报错。

byType：该选项可以根据bean类型设置依赖关系。当向一个bean中自动装配一个属性时，容器将根据bean的类型自动在在配置文件中查询一个匹配的bean。如果找到的话，就装配这个属性，如果没找到的话就报错。

constructor：造器的自动装配和byType模式类似，但是仅仅适用于与有构造器相同参数的bean，如果在容器中没有找到与构造器参数类型一致的bean，那么将会抛出异常。

autodetect：该模式自动探测使用构造器自动装配或者byType自动装配。首先，首先会尝试找合适的带参数的构造器，如果找到的话就是用构造器自动装配，如果在bean内部没有找到相应的构造器或者是无参构造器，容器就会自动选择byTpe的自动装配方式。

#### 22、如何开启基于注解的自动装配？

要使用 @Autowired，需要注册 AutowiredAnnotationBeanPostProcessor，可以有以下两种方式来实现：

1、引入配置文件中的<bean>下引入 <context:annotation-config>

<beans> <context:annotation-config /> </beans>

2、在bean配置文件中直接引入AutowiredAnnotationBeanPostProcessor

<beans> <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/> </beans>

#### 23、请举例解释@Required annotation？

在产品级别的应用中，IoC容器可能声明了数十万了bean，bean与bean之间有着复杂的依赖关系。设值注解方法的短板之一就是验证所有的属性是否被注解是一项十分困难的操作。可以通过在<bean>中设置“dependency-check”来解决这个问题。

在应用程序的生命周期中，你可能不大愿意花时间在验证所有bean的属性是否按照上下文文件正确配置。或者你宁可验证某个bean的特定属性是否被正确的设置。即使是用“dependency-check”属性也不能很好的解决这个问题，在这种情况下，你需要使用@Required 注解。

需要用如下的方式使用来标明bean的设值方法。

public class EmployeeFactoryBean extends AbstractFactoryBean<Object> {

private String designation;

public String getDesignation() {

return designation;

}

@Required

public void setDesignation(String designation) {

this.designation = designation;

} //more code here }

RequiredAnnotationBeanPostProcessor是Spring中的后置处理用来验证被@Required 注解的bean属性是否被正确的设置了。在使用RequiredAnnotationBeanPostProcesso来验证bean属性之前，首先要在IoC容器中对其进行注册：

<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" />

但是如果没有属性被用 @Required 注解过的话，后置处理器会抛出一个BeanInitializationException 异常。

#### 24、请举例解释@Autowired注解？

@Autowired注解对自动装配何时何处被实现提供了更多细粒度的控制。@Autowired注解可以像@Required注解、构造器一样被用于在bean的设值方法上自动装配bean的属性，一个参数或者带有任意名称或带有多个参数的方法。

比如，可以在设值方法上使用@Autowired注解来替代配置文件中的 <property>元素。当Spring容器在setter方法上找到@Autowired注解时，会尝试用byType 自动装配。

当然我们也可以在构造方法上使用@Autowired 注解。带有@Autowired 注解的构造方法意味着在创建一个bean时将会被自动装配，即便在配置文件中使用<constructor-arg> 元素。

public class TextEditor {

private SpellChecker spellChecker;

@Autowired

public TextEditor(SpellChecker spellChecker){

System.out.println("Inside TextEditor constructor." ); this.spellChecker = spellChecker;

}

public void spellCheck(){

spellChecker.checkSpelling();

} }

下面是没有构造参数的配置方式：

<beans>

<context:annotation-config/>

<!-- Definition for textEditor bean without constructor-arg --> <bean id="textEditor" class="com.howtodoinjava.TextEditor"></bean> <!-- Definition for spellChecker bean -->

<bean id="spellChecker" class="com.howtodoinjava.SpellChecker"> </bean>

</beans>

#### 25、请举例说明@Qualifier注解？

@Qualifier注解意味着可以在被标注bean的字段上可以自动装配。Qualifier注解可以用来取消Spring不能取消的bean应用。

下面的示例将会在Customer的person属性中自动装配person的值。

public class Customer {

@Autowired

private Person person;

}

下面我们要在配置文件中来配置Person类。

<bean id="customer" class="com.howtodoinjava.common.Customer" />

<bean id="personA" class="com.howtodoinjava.common.Person" >

<property name="name" value="lokesh" />

</bean>

<bean id="personB" class="com.howtodoinjava.common.Person" >

<property name="name" value="alex" />

</bean>

Spring会知道要自动装配哪个person bean么？不会的，但是运行上面的示例时，会抛出下面的异常：

Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No unique bean of type [com.howtodoinjava.common.Person] is defined: expected single matching bean but found 2: [personA, personB]

要解决上面的问题，需要使用 @Quanlifier注解来告诉Spring容器要装配哪个bean：

public class Customer {

@Autowired

@Qualifier("personA")

private Person person;

}

#### 26、构造方法注入和设值注入有什么区别？

请注意以下明显的区别：

在设值注入方法支持大部分的依赖注入，如果我们仅需要注入int、string和long型的变量，我们不要用设值的方法注入。对于基本类型，如果我们没有注入的话，可以为基本类型设置默认值。在构造方法注入不支持大部分的依赖注入，因为在调用构造方法中必须传入正确的构造参数，否则的话为报错。

设值注入不会重写构造方法的值。如果我们对同一个变量同时使用了构造方法注入又使用了设置方法注入的话，那么构造方法将不能覆盖由设值方法注入的值。很明显，因为构造方法尽在对象被创建时调用。

在使用设值注入时有可能还不能保证某种依赖是否已经被注入，也就是说这时对象的依赖关系有可能是不完整的。而在另一种情况下，构造器注入则不允许生成依赖关系不完整的对象。

在设值注入时如果对象A和对象B互相依赖，在创建对象A时Spring会抛出s

ObjectCurrentlyInCreationException异常，因为在B对象被创建之前A对象是不能被创建的，反之亦然。所以Spring用设值注入的方法解决了循环依赖的问题，因对象的设值方法是在对象被创建之前被调用的。

#### 27、Spring框架中有哪些不同类型的事件？

Spring的ApplicationContext 提供了支持事件和代码中监听器的功能。

我们可以创建bean用来监听在ApplicationContext 中发布的事件。ApplicationEvent类和在ApplicationContext接口中处理的事件，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。

public class AllApplicationEventListener implements ApplicationListener < ApplicationEvent > {

@Override

public void onApplicationEvent(ApplicationEvent applicationEvent) {

//process event

} }

Spring 提供了以下5中标准的事件：

上下文更新事件（ContextRefreshedEvent）：该事件会在ApplicationContext被初始化或者更新时发布。也可以在调用ConfigurableApplicationContext 接口中的refresh()方法时被触发。

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。

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。

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。

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。

除了上面介绍的事件以外，还可以通过扩展ApplicationEvent 类来开发自定义的事件。

public class CustomApplicationEvent extends ApplicationEvent {

public CustomApplicationEvent ( Object source, final String msg ) {

super(source);

System.out.println("Created a Custom event");

} }

为了监听这个事件，还需要创建一个监听器：

public class CustomEventListener implements ApplicationListener < CustomApplicationEvent > {

@Override

public void onApplicationEvent(CustomApplicationEvent applicationEvent) {

//handle event

} }

之后通过applicationContext接口的publishEvent()方法来发布自定义事件。

CustomApplicationEvent customEvent = new CustomApplicationEvent(applicationContext, "Test message"); applicationContext.publishEvent(customEvent);

#### 28、Spring 框架中都用到了哪些设计模式？

Spring框架中使用到了大量的设计模式，下面列举了比较有代表性的：

代理模式—在AOP和remoting中被用的比较多。

单例模式—在spring配置文件中定义的bean默认为单例模式。

模板方法—用来解决代码重复的问题。

比如. RestTemplate, JmsTemplate, JpaTemplate。

前端控制器—Srping提供了DispatcherServlet来对请求进行分发。

视图帮助(View Helper )—Spring提供了一系列的JSP标签，高效宏来辅助将分散的代码整合在视图里。

依赖注入—贯穿于BeanFactory / ApplicationContext接口的核心理念。

工厂模式—BeanFactory用来创建对象的实例。

### 4.2 Struts2

#### 1、Struts2概述

Struts 2框架本身大致可以分为3个部分：

核心控制器FilterDispatcher、业务控制器Action和用户实现的企业业务逻辑组件。

核心控制器FilterDispatcher是Struts 2框架的基础，

包含了框架内部的控制流程和处理机制。

业务控制器Action和业务逻辑组件是需要用户来自己实现的。

用户在开发Action和业务逻辑组件的同时，还需要编写相关的配置文件，

供核心控制器FilterDispatcher来使用。

2、Struts 2的工作流程相对于Struts 1要简单，与WebWork框架基本相同，

所以说Struts 2是WebWork的升级版本。基本简要流程如下：

1 、客户端初始化一个指向Servlet容器的请求；

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 、接着FilterDispatcher被调用，

FilterDispatcher询问ActionMapper来决定这个请是否需要调用某个Action

4、如果ActionMapper决定需要调用某个Action，

FilterDispatcher把请求的处理交给ActionProxy

5、ActionProxy通过Configuration Manager询问框架的配置文件，

找到需要调用的Action类

6、ActionProxy创建一个ActionInvocation的实例。

7、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。

8、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

9、响应的返回是通过我们在web.xml中配置的过滤器

10、如果ActionContextCleanUp是当前使用的，则FilterDispatecher将不会清理sreadlocal ActionContext;如果ActionContextCleanUp不使用，则将会去清理sreadlocals。

#### 2、说下Struts的设计模式

MVC模式: web应用程序启动时就会加载并初始化ActionServler。用户提交表单时，一个配置好的ActionForm对象被创建，并被填入表单相应的数据，ActionServler根据Struts-config.xml文件配置好的设置决定是否需要表单验证，如果需要就调用ActionForm的Validate（）验证后选择将请求发送到哪个Action，如果Action不存在，ActionServlet会先创建这个对象，然后调用Action的execute（）方法。Execute（）从ActionForm对象中获取数据，完成业务逻辑，返回一个ActionForward对象，ActionServlet再把客户请求转发给ActionForward对象指定的jsp组件，ActionForward对象指定的jsp生

成动态的网页，返回给客户。

#### 3、拦截器和过滤器的区别

1、拦截器是基于java反射机制的，而过滤器是基于函数回调的。

2、过滤器依赖于servlet容器，而拦截器不依赖于servlet容器。

3、拦截器只能对Action请求起作用，而过滤器则可以对几乎所有请求起作用。

4、拦截器可以访问Action上下文、值栈里的对象，而过滤器不能。

5、在Action的生命周期中，拦截器可以多次调用，而过滤器只能在容器初始化时被调用一次。

#### 4、struts1于struts2的比较

1、Action类:

Struts1要求Action类继承一个抽象基类。Struts1的一个普遍问题是使用抽象类编程而不是接口。

Struts 2 Action类可以实现一个Action接口，也可实现其他接口，使可选和定制的服务成为可能。Struts2提供一个ActionSupport基类去 实现常用的接口。Action接口不是必须的，任何有execute标识的POJO对象都可以用作Struts2的Action对象。

2、线程模式:

Struts1 Action是单例模式并且必须是线程安全的，因为仅有Action的一个实例来处理所有的请求。单例策略限制了Struts1 Action能作的事，并且要在开发时特别小心。Action资源必须是线程安全的或同步的。

Struts2 Action对象为每一个请求产生一个实例，因此没有线程安全问题。（实际上，servlet容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题）

3、Servlet依赖:

Struts1 Action依赖于Servlet API,因为当一个Action被调用时HttpServletRequest 和HttpServletResponse被传递给execute方法。

Struts 2 Action不依赖于容器，允许Action脱离容器单独被测试。如果需要，Struts2 Action仍然可以访问初始的request和response。但是，其他的元素减少或者消除了直接访问HttpServetRequest 和 HttpServletResponse的必要性。

4、可测性:

测试Struts1 Action的一个主要问题是execute方法暴露了servlet API（这使得测试要依赖于容器）。一个第三方扩展－－Struts TestCase－－提供了一套Struts1的模拟对象（来进行测试）。

Struts 2 Action可以通过初始化、设置属性、调用方法来测试，“依赖注入”支持也使测试更容易。

5、捕获输入:

Struts1 使用ActionForm对象捕获输入。所有的ActionForm必须继承一个基类。因为其他JavaBean不能用作ActionForm，开发者经 常创建多余的类捕获输入。动态Bean（DynaBeans）可以作为创建传统ActionForm的选择，但是，开发者可能是在重新描述(创建)已经存 在的JavaBean（仍然会导致有冗余的javabean）。

Struts 2直接使用Action属性作为输入属性，消除了对第二个输入对象的需求。输入属性可能是有自己(子)属性的rich对象类型。Action属性能够通过 web页面上的taglibs访问。Struts2也支持ActionForm模式。rich对象类型，包括业务对象，能够用作输入/输出对象。这种 ModelDriven 特性简化了taglib对POJO输入对象的引用。

6、表达式语言：

Struts1 整合了JSTL，因此使用JSTL EL。这种EL有基本对象图遍历，但是对集合和索引属性的支持很弱。

Struts2可以使用JSTL，但是也支持一个更强大和灵活的表达式语言－－ "Object Graph Notation Language " (OGNL).

7、绑定值到页面（view）:

Struts 1使用标准JSP机制把对象绑定到页面中来访问。

Struts 2 使用 "ValueStack "技术，使taglib能够访问值而不需要把你的页面（view）和对象绑定起来。ValueStack策略允许通过一系列名称相同但类型不同的属性重用页面（view）。

8、类型转换：

Struts 1 ActionForm 属性通常都是String类型。Struts1使用Commons-Beanutils进行类型转换。每个类一个转换器，对每一个实例来说是不可配置的。

Struts2 使用OGNL进行类型转换。提供基本和常用对象的转换器。

9、校验：

Struts 1支持在ActionForm的validate方法中手动校验，或者通过Commons Validator的扩展来校验。同一个类可以有不同的校验内容，但不能校验子对象。

Struts2支持通过validate方法和XWork校验框架来进行校验。XWork校验框架使用为属性类类型定义的校验和内容校验，来支持chain校验子属性

10、Action执行的控制：

Struts1支持每一个模块有单独的Request Processors（生命周期），但是模块中的所有Action必须共享相同的生命周期。

Struts2支持通过拦截器堆栈（Interceptor Stacks）为每一个Action创建不同的生命周期。堆栈能够根据需要和不同的Action一起使用。

#### 5、为什么要使用Struts2

Struts2 是一个相当强大的Java Web开源框架，是一个基于POJO的Action的MVC Web框架。它基于当年的Webwork和XWork框架，继承其优点，同时做了相当的改进。

1.Struts2基于MVC架构，框架结构清晰，开发流程一目了然，开发人员可以很好的掌控开发的过程。

2使用OGNL进行参数传递。

OGNL提供了在Struts2里访问各种作用域中的数据的简单方式，你可以方便的获取Request，Attribute，Application，Session，Parameters中的数据。大大简化了开发人员在获取这些数据时的代码量。

3强大的拦截器

Struts2 的拦截器是一个Action级别的AOP，Struts2中的许多特性都是通过拦截器来实现的，例如异常处理，文件上传，验证等。拦截器是可配置与重用的，可以将一些通用的功能如：登录验证，权限验证等置于拦截器中以完成一些Java Web项目中比较通用的功能。在我实现的的一Web项目中，就是使用Struts2的拦截器来完成了系统中的权限验证功能。

4易于测试

Struts2的Action都是简单的POJO，这样可以方便的对Struts2的Action编写测试用例，大大方便了5Java Web项目的测试。

易于扩展的插件机制在Struts2添加扩展是一件愉快而轻松的事情，只需要将所需要的Jar包放到WEB-INF/lib文件夹中，在struts.xml中作一些简单的设置就可以实现扩展。

6模块化管理

Struts2已经把模块化作为了体系架构中的基本思想，可以通过三种方法来将应用程序模块化：将配置信息拆分成多个文件把自包含的应用模块创建为插件创建新的框架特性，即将与特定应用无关的新功能组织成插件，以添加到多个应用中去。

7全局结果与声明式异常

为应用程序添加全局的Result，和在配置文件中对异常进行处理，这样当处理过程中出现指定异常时，可以跳转到特定页面。

他的如此之多的优点，是很多人比较的青睐，与spring ,Hibernate进行结合，组成了现在比较流行的ssh框架，当然每个公司都要自己的框架，也是ssh变异的产品。

#### 6、struts2有哪些优点？

1）在软件设计上Struts2的应用可以不依赖于Servlet API和struts API。 Struts2的这种设计属于无侵入式设计；

2）拦截器，实现如参数拦截注入等功能；

3）类型转换器，可以把特殊的请求参数转换成需要的类型；

4）多种表现层技术，如：JSP、freeMarker、Velocity等；

5）Struts2的输入校验可以对指定某个方法进行校验；

6）提供了全局范围、包范围和Action范围的国际化资源文件管理实现

#### 7、struts2是如何启动的？

struts2框架是通过Filter启动的，即StrutsPrepareAndExecuteFilter，此过滤器为struts2的核心过滤器；

StrutsPrepareAndExecuteFilter的init()方法中将会读取类路径下默认的配置文件struts.xml完成初始化操作。struts2读取到struts.xml的内容后，是将内容封装进javabean对象然后存放在内存中，以后用户的每次请求处理将使用内存中的数据，而不是每次请求都读取struts.xml文件。

#### 8、struts2框架的核心控制器是什么？它有什么作用？

1）Struts2框架的核心控制器是StrutsPrepareAndExecuteFilter。

2）作用：

负责拦截由<url-pattern>/\*</url-pattern>指定的所有用户请求，当用户请求到达时，该Filter会过滤用户的请求。默认情况下，如果用户请求的路径

不带后缀或者后缀以.action结尾，这时请求将被转入struts2框架处理，否则struts2框架将略过该请求的处理。

可以通过常量"struts.action.extension"修改action的后缀，如：

<constant name="struts.action.extension" value="do"/>

如果用户需要指定多个请求后缀，则多个后缀之间以英文逗号（,）隔开。

<constant name="struts.action.extension" value="do,go"/>

#### 9、struts2配置文件的加载顺序？

struts.xml ——> struts.properties

常量可以在struts.xml或struts.properties中配置，如果在多个文件中配置了同一个常量，则后一个文件中配置的常量值会覆盖前面文件中配置的常量值.

struts.xml文件的作用：通知Struts2框架加载对应的Action资源

#### 10、struts2常量的修改方式？

常量可以在struts.xml或struts.properties中配置，两种配置方式如下：

1）在struts.xml文件中配置常量

<constant name="struts.action.extension" value="do"/>

2）在struts.properties中配置常量（struts.properties文件放置在src下）：

struts.action.extension=do

#### 11、struts2如何访问HttpServletRequest、HttpSession、ServletContext三个域对象？

方案一：

HttpServletRequest request =ServletActionContext.getRequest();

HttpServletResponse response =ServletActionContext.getResponse();

HttpSession session= request.getSession();

ServletContext servletContext=ServletActionContext.getServletContext();

方案二：

类implements ServletRequestAware,ServletResponseAwar，SessionAware，ServletContextAware

注意：框架自动传入对应的域对象

#### 12、struts2是如何管理action的？这种管理方式有什么好处？

struts2框架中使用包来管理Action，包的作用和java中的类包是非常类似的。

主要用于管理一组业务功能相关的action。在实际应用中，我们应该把一组业务功能相关的Action放在同一个包下。

#### 13、struts2中的默认包struts-default有什么作用？

1）struts-default包是由struts内置的，它定义了struts2内部的众多拦截器和Result类型，而Struts2很多核心的功能都是通过这些内置的拦截器实现，如：从请求中

把请求参数封装到action、文件上传和数据验证等等都是通过拦截器实现的。当包继承了struts-default包才能使用struts2为我们提供的这些功能。

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

3）通常每个包都应该继承struts-default包。

#### 14、struts2如何对指定的方法进行验证？

1）validate()方法会校验action中所有与execute方法签名相同的方法；

2）要校验指定的方法通过重写validateXxx()方法实现， validateXxx()只会校验action中方法名为Xxx的方法。其中Xxx的第一个字母要大写；

3）当某个数据校验失败时，调用addFieldError()方法往系统的fieldErrors添加校验失败信息（为了使用addFieldError()方法，action可以继承ActionSupport）， 如果系统 的fieldErrors包含失败信息，struts2会将请求转发到名为input的result；

4）在input视图中可以通过<s:fielderror/>显示失败信息。

5）先执行validateXxxx()->validate()->如果出错了，会转发<result name="input"/>所指定的页面，如果不出错，会直接进行Action::execute()方法

#### 15、struts2默认能解决get和post提交方式的乱码问题吗？

不能。struts.i18n.encoding=UTF-8属性值只能解析POST提交下的乱码问题。

#### 16、请你写出struts2中至少5个的默认拦截器？

fileUpload 提供文件上传功能

i18n 记录用户选择的locale

cookies 使用配置的name,value来是指cookies

checkbox 添加了checkbox自动处理代码，将没有选中的checkbox的内容设定为false，而html默认情况下不提交没有选中的checkbox。

chain 让前一个Action的属性可以被后一个Action访问，现在和chain类型的result（）结合使用。

alias 在不同请求之间将请求参数在不同名字件转换，请求内容不变

#### 17、值栈ValueStack的原理与生命周期？

1）ValueStack贯穿整个 Action 的生命周期，保存在request域中，所以ValueStack和request的生命周期一样。当Struts2接受一个请求时，会迅速创建ActionContext，

ValueStack，action。然后把action存放进ValueStack，所以action的实例变量可以被OGNL访问。 请求来的时候，action、ValueStack的生命开始，请求结束，action、 ValueStack的生命结束；

2）action是多例的，和Servlet不一样，Servelt是单例的；

3）每个action的都有一个对应的值栈，值栈存放的数据类型是该action的实例，以及该action中的实例变量，Action对象默认保存在栈顶；

4）ValueStack本质上就是一个ArrayList；

5）关于ContextMap，Struts 会把下面这些映射压入 ContextMap 中：

parameters : 该 Map 中包含当前请求的请求参数

request : 该 Map 中包含当前 request 对象中的所有属性 session :该 Map 中包含当前 session 对象中的所有属性

application :该 Map 中包含当前 application 对象中的所有属性

attr:该 Map 按如下顺序来检索某个属性: request, session, application

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

7）注意： Struts2中，OGNL表达式需要配合Struts标签才可以使用。如：<s:property value="name"/>

8）在struts2配置文件中引用ognl表达式 ,引用值栈的值 ，此时使用的"$"，而不是#或者%;

#### 18、ActionContext、ServletContext、pageContext的区别？

1）ActionContext是当前的Action的上下文环境，通过ActionContext可以获取到request、session、ServletContext等与Action有关的对象的引用；

2）ServletContext是域对象，一个web应用中只有一个ServletContext，生命周期伴随整个web应用；

3）pageContext是JSP中的最重要的一个内置对象，可以通过pageContext获取其他域对象的应用，同时它是一个域对象，作用范围只针对当前页面，当前页面结束时，pageContext销毁，

生命周期是JSP四个域对象中最小的。

#### 19、result的type属性中有哪几种结果类型？

一共10种：

dispatcher

struts默认的结果类型，把控制权转发给应用程序里的某个资源不能把控制权转发给一个外部资源，若需要把控制权重定向到一个外部资源, 应该使用

redirect结果类型

redirect 把响应重定向到另一个资源（包括一个外部资源）

redirectAction 把响应重定向到另一个 Action

freemarker、velocity、chain、httpheader、xslt、plainText、stream

#### 20、拦截器的生命周期与工作过程？

1）每个拦截器都是实现了Interceptor接口的 Java 类；

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

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

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

5）struts2中有内置了18个拦截器。

#### 21、struts2如何完成文件的上传？

1、JSP页面：

1）JSP页面的上传文件的组件：<s: file name=”upload” />，如果需要一次上传多个文件, 就必须使用多个 file 标签, 但它们的名字必须是相同的，即：

name=“xxx”的值必须一样；

2）必须把表单的enctype属性设置为：multipart/form-data；

3）表单的方法必须为post，因为post提交的数据在消息体中，而无大小限制。

2、对应的action：

1）在 Action 中新添加 3 个和文件上传相关的属性；

2）如果是上传单个文件, uploadImage属性的类型就是 java.io.File, 它代表被上传的文件, 第二个和第三个属性的类型是 String, 它们分别代表上传文

件的文件名和文件类型，定义方式是分别是：

jsp页面file组件的名称+ContentType, jsp页面file组件的名称+FileName

3）如果上上传多个文件, 可以使用数组或 List

### 4.3 Hibernate

#### 1、Hibernate工作原理及优势？

原理：

1.读取并解析配置文件

2.读取并解析映射信息，创建SessionFactory

3.打开Sesssion

4.创建事务Transation

5.持久化操作

6.提交事务

7.关闭Session

8.关闭SesstionFactory

优势：

1. 对JDBC访问数据库的代码做了封装，大大简化了数据访问层繁琐的重复性代码。

2. Hibernate是一个基于JDBC的主流持久化框架，是一个优秀的ORM实现。他很大程度

简化DAO层的编码工作

3. hibernate使用Java反射机制，而不是字节码增强程序来实现透明性。

4. hibernate的性能非常好，因为它是个轻量级框架。映射的灵活性很出色。它支持各种关系数据库，从一对一到多对多的各种复杂关系。

1、什么是Hibernate的并发机制？怎么去处理并发问题？

Hibernate并发机制：

a、Hibernate的Session对象是非线程安全的,对于单个请求,单个会话,单个的工作单元(即单个事务,单个线程),它通常只使用一次, 然后就丢弃。

如果一个Session 实例允许共享的话，那些支持并发运行的,例如Http request,session beans将会导致出现资源争用。

如果在Http Session中有hibernate的Session的话,就可能会出现同步访问Http Session。只要用户足够快的点击浏览器的“刷新”, 就会导致两个并发运行的线程使用同一个Session。

b、多个事务并发访问同一块资源,可能会引发第一类丢失更新，脏读，幻读，不可重复读，第二类丢失更新一系列的问题。

解决方案：设置事务隔离级别。

Serializable：串行化。隔离级别最高

Repeatable Read：可重复读

Read Committed：已提交数据读

Read Uncommitted：未提交数据读。隔离级别最差

设置锁：乐观锁和悲观锁。

乐观锁：使用版本号或时间戳来检测更新丢失,在的映射中设置 optimistic-lock=”all”可以在没有版本或者时间戳属性映射的情况下实现 版本检查，此时Hibernate将比较一行记录的每个字段的状态 行级悲观锁：Hibernate总是使用数据库的锁定机制，从不在内存中锁定对象！只要为JDBC连接指定一下隔 离级别，然后让数据库去搞定一切就够了。类LockMode 定义了Hibernate所需的不同的锁定级别：LockMode.UPGRADE,LockMode.UPGRADE\_NOWAIT,LockMode.READ;

#### 2、update和saveOrUpdate的区别？

update()和saveOrUpdate()是用来对跨Session的PO进行状态管理的。

update()方法操作的对象必须是持久化了的对象。也就是说，如果此对象在数据库中不存在的话，就不能使用update()方法。

saveOrUpdate()方法操作的对象既可以使持久化了的，也可以使没有持久化的对象。如果是持久化了的对象调用saveOrUpdate()则会 更新数据库中的对象；如果是未持久化的对象使用此方法,则save到数据库中。

#### 3、hibernate的三种状态之间如何转换

当对象由瞬时状态(Transient)一save()时，就变成了持久化状态；

当我们在Session里存储对象的时候，实际是在Session的Map里存了一份， 也就是它的缓存里放了一份，然后，又到数据库里存了一份，在缓存里这一份叫持久对象(Persistent)。 Session 一 Close()了,它的缓存也都关闭了,整个Session也就失效了,这个时候，这个对象变成了游离状态(Detached)，但数据库中还是存在的。

当游离状态(Detached)update()时，又变为了持久状态(Persistent)。

当持久状态(Persistent)delete()时，又变为了瞬时状态(Transient), 此时，数据库中没有与之对应的记录。

#### 4、比较hibernate的三种检索策略优缺点

1立即检索；

优点： 对应用程序完全透明，不管对象处于持久化状态，还是游离状态，应用程序都可以方便的从一个对象导航到与它关联的对象；

缺点： 1.select语句太多；2.可能会加载应用程序不需要访问的对象白白浪费许多内存空间；

2延迟检索：

优点： 由应用程序决定需要加载哪些对象，可以避免可执行多余的select语句，以及避免加载应用程序不需要访问的对象。因此能提高检索性能，并且能节省内存空间；

缺点： 应用程序如果希望访问游离状态代理类实例，必须保证他在持久化状态时已经被初始化；

3 迫切左外连接检索

优点： 1对应用程序完全透明，不管对象处于持久化状态，还是游离状态，应用程序都可以方便地冲一个对象导航到与它关联的对象。2使用了外连接，select语句数目少；

缺点： 1 可能会加载应用程序不需要访问的对象，白白浪费许多内存空间；2复杂的数据库表连接也会影响检索性能；

#### 5、如何在控制台看到hibernate生成并执行的sql

在定义数据库和数据库属性的文件applicationConfig.xml里面，把hibernate.show\_sql 设置为true

这样生成的SQL就会在控制台出现了

注意：这样做会加重系统的负担，不利于性能调优

#### 6、hibernate都支持哪些缓存策略

Read-only: 这种策略适用于那些频繁读取却不会更新的数据，这是目前为止最简单和最有效的缓存策略

\* Read/write:这种策略适用于需要被更新的数据，比read-only更耗费资源，在非JTA环境下，每个事务需要在session.close和session.disconnect()被调用

\* Nonstrict read/write: 这种策略不保障两个同时进行的事务会修改同一块数据，这种策略适用于那些经常读取但是极少更新的数据

\* Transactional: 这种策略是完全事务化得缓存策略，可以用在JTA环境下

#### 7、hibernate里面的sorted collection 和ordered collection有什么区别

sorted collection是在内存中通过Java比较器进行排序的

ordered collection是在数据库中通过order by进行排序的

#### 8、Hibernate是如何延迟加载?

当Hibernate在查询数据的时候，数据并没有存在与内存中，当程序真正对数据的操作时，对象才存在与内存中，就实现了延迟加载，他节省了服务器的内存开销，从而提高了服务器的性能。

#### 9、Hibernate中怎样实现类之间的关系?(如：一对多、多对多的关系)

类与类之间的关系主要体现在表与表之间的关系进行操作，它们都是对对象进行操作，我们程序中把所有的表与类都映射在一起，它们通过配置文件中的many-to-one、one-to-many、many-to-many、

#### 10、说下Hibernate的缓存机制

1. 内部缓存存在Hibernate中又叫一级缓存，属于应用事物级缓存

2. 二级缓存：

a) 应用及缓存

b) 分布式缓存

条件：数据不会被第三方修改、数据大小在可接受范围、数据更新频率低、同一数据被系统频繁使用、非关键数据

c) 第三方缓存的实现

13、Hibernate的查询方式

Sql、Criteria,objectcomposition

Hql：

1、 属性查询

2、 参数查询、命名参数查询

3、 关联查询

4、 分页查询

5、 统计函数

#### 11、如何优化Hibernate？

1.使用双向一对多关联，不使用单向一对多

2.灵活使用单向一对多关联

3.不用一对一，用多对一取代

4.配置对象缓存，不使用集合缓存

5.一对多集合使用Bag,多对多集合使用Set

6. 继承类使用显式多态

7. 表字段要少，表关联不要怕多，有二级缓存撑腰

#### 12、Hibernate有哪几种查询数据的方式

3种：hql、条件查询QBC(QueryBy Criteria)、原生sql （通过createSQLQuery建立）

#### 13、谈谈Hibernate中inverse的作用

inverse属性默认是false,就是说关系的两端都来维护关系。

比如Student和Teacher是多对多关系，用一个中间表TeacherStudent维护。Gp)i

如果Student这边inverse=”true”, 那么关系由另一端Teacher维护，就是说当插入Student时，不会操作TeacherStudent表（中间表）。只有Teacher插入或删除时才会触发对中间表的操作。所以两边都inverse=”true”是不对的，会导致任何操作都不触发对中间表的影响；当两边都inverse=”false”或默认时，会导致在中间表中插入两次关系。

#### 14、Detached Object（游离对象）有什么好处

Detached Object（游离对象）可以传递到任何层直到表现层而不是用任何DTO(DataTransfer Objects). 然后你还可以重新把游离对象赋给另外一个Session.

#### 15、JDBC hibernate 和 ibatis 的区别

jdbc:手动

手动写sql

delete、insert、update要将对象的值一个一个取出传到sql中,不能直接传入一个对象。

select:返回的是一个resultset，要从ResultSet中一行一行、一个字段一个字段的取出，然后封装到一个对象中，不直接返回一个对象。

ibatis的特点:半自动化

sql要手动写

delete、insert、update:直接传入一个对象

select:直接返回一个对象

hibernate:全自动

不写sql,自动封装

delete、insert、update:直接传入一个对象

select:直接返回一个对象

#### 16、在数据库中条件查询速度很慢的时候,如何优化?

1.建索引

2.减少表之间的关联

3.优化sql，尽量让sql很快定位数据，不要让sql做全表查询，应该走索引,把数据量大的表排在前面

4.简化查询字段，没用的字段不要，已经对返回结果的控制，尽量返回少量数据

#### 17、什么是SessionFactory,她是线程安全么？

SessionFactory 是Hibrenate单例数据存储和线程安全的，以至于可以多线程同时访问。一个SessionFactory 在启动的时候只能建立一次。SessionFactory应该包装各种单例以至于它能很简单的在一个应用代码中储存.

#### 18、Hibernate的五个核心接口

Configuration 接口：配置Hibernate，根据其启动hibernate，创建

SessionFactory 对象；

SessionFactory 接口：初始化Hibernate，充当数据存储源的代理，创建

session 对象，sessionFactory 是线程安全的，意味着它的同一个实例可以被应

用的多个线程共享，是重量级、二级缓存；

Session 接口：负责保存、更新、删除、加载和查询对象，是线程不安全的，

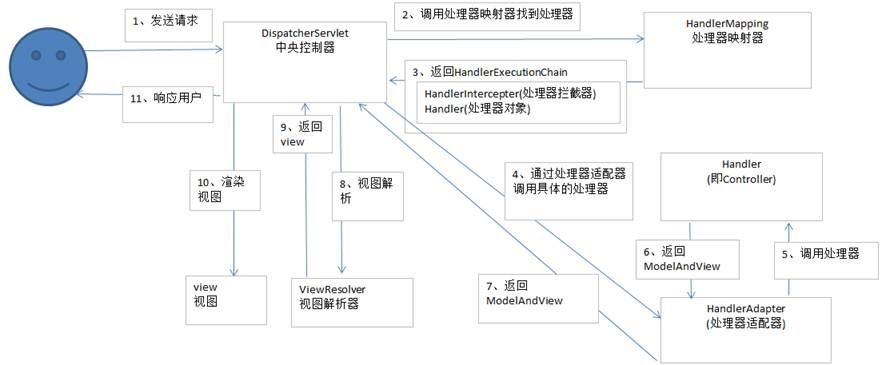
避免多个线程共享同一个session，是轻量级、一级缓存；

Transaction 接口：管理事务；

Query 和Criteria 接口：执行数据库的查询

### 4.4 SpringMVC

#### 1、简单的谈一下SpringMVC的工作流程？



流程

1、用户发送请求至前端控制器DispatcherServlet

2、DispatcherServlet收到请求调用HandlerMapping处理器映射器。

3、处理器映射器找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

4、DispatcherServlet调用HandlerAdapter处理器适配器

5、HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)。

6、Controller执行完成返回ModelAndView

7、HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

8、DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9、ViewReslover解析后返回具体View

10、DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）。

11、DispatcherServlet响应用户

#### 2、如何解决POST请求中文乱码问题，GET的又如何处理呢？

在web.xml中加入：

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

以上可以解决post请求乱码问题。对于get请求中文参数出现乱码解决方法有两个：

修改tomcat配置文件添加编码与工程编码一致，如下：

<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

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

#### 3、SpringMVC与Struts2的主要区别？

①springmvc的入口是一个servlet即前端控制器，而struts2入口是一个filter过虑器。

②springmvc是基于方法开发，传递参数是通过方法形参，可以设计为单例或多例(建议单例)，struts2是基于类开发，传递参数是通过类的属性，只能设计为多例。

③Struts采用值栈存储请求和响应的数据，通过OGNL存取数据， springmvc通过参数解析器是将request对象内容进行解析成方法形参，将响应数据和页面封装成ModelAndView对象，最后又将模型数据通过request对象传输到页面。 Jsp视图解析器默认使用jstl。

### 4.5 MyBatis

#### 1、#{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换。

Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

Mybatis在处理${}时，就是把${}替换成变量的值。

使用#{}可以有效的防止SQL注入，提高系统安全性。

#### 2、当实体类中的属性名和表中的字段名不一样 ，怎么办 ？

第1种： 通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致

<select id=”selectorder” parametertype=”int” resultetype=”me.gacl.domain.order”>

select order\_id id, order\_no orderno ,order\_price price form orders where order\_id=#{id};

</select>

第2种： 通过<resultMap>来映射字段名和实体类属性名的一一对应的关系

<select id="getOrder" parameterType="int" resultMap="orderresultmap">

select \* from orders where order\_id=#{id}

</select>

<resultMap type=”me.gacl.domain.order” id=”orderresultmap”>

<!–用id属性来映射主键字段–>

<id property=”id” column=”order\_id”>

<!–用result属性来映射非主键字段，property为实体类属性名，column为数据表中的属性–>

<result property = “orderno” column =”order\_no”/>

<result property=”price” column=”order\_price” />

</reslutMap>

#### 3、 模糊查询like语句该怎么写?

第1种：在Java代码中添加sql通配符。

string wildcardname = “%smi%”;

list<name> names = mapper.selectlike(wildcardname);

<select id=”selectlike”>

select \* from foo where bar like #{value}

</select>

第2种：在sql语句中拼接通配符，会引起sql注入

string wildcardname = “smi”;

list<name> names = mapper.selectlike(wildcardname);

<select id=”selectlike”>

select \* from foo where bar like "%"#{value}"%"

</select>

#### 4、通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

Dao接口，就是人们常说的Mapper接口，接口的全限名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中MappedStatement的id值，接口方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement，举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个<select>、<insert>、<update>、<delete>标签，都会被解析为一个MappedStatement对象。

Dao接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

Dao接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Dao接口生成代理proxy对象，代理对象proxy会拦截接口方法，转而执行Mappedstatement所代表的sql,然后将sql执行结果返回。

#### 5、Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页，可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

#### 6、Mybatis是如何将sql执行结果封装为目标对象并返回的？都有哪些映射形式？

答：第一种是使用<resultMap>标签，逐一定义列名和对象属性名之间的映射关系。第二种是使用sql列的别名功能，将列别名书写为对象属性名，比如T\_NAME AS NAME，对象属性名一般是name，小写，但是列名不区分大小写，Mybatis会忽略列名大小写，智能找到与之对应对象属性名，你甚至可以写成T\_NAME AS NaMe，Mybatis一样可以正常工作。

有了列名与属性名的映射关系后，Mybatis通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

#### 7、如何执行批量插入?

首先,创建一个简单的insert语句:

<insert id=”insertname”>

insert into names (name) values (#{value})

</insert>

然后在java代码中像下面这样执行批处理插入:

list<string> names = new arraylist();

names.add(“fred”);

names.add(“barney”);

names.add(“betty”);

names.add(“wilma”);

// 注意这里 executortype.batch

sqlsession sqlsession = sqlsessionfactory.opensession(executortype.batch);

try {

namemapper mapper = sqlsession.getmapper(namemapper.class);

for (string name : names) {

mapper.insertname(name);

}

sqlsession.commit();

} finally {

sqlsession.close();

}

#### 8、如何获取自动生成的(主)键值?

insert 方法总是返回一个int值 - 这个值代表的是插入的行数。

而自动生成的键值在 insert 方法执行完后可以被设置到传入的参数对象中。

示例:

<insert id=”insertname” usegeneratedkeys=”true” keyproperty=”id”>

insert into names (name) values (#{name})

</insert>

name name = new name();

name.setname(“fred”);

int rows = mapper.insertname(name);

// 完成后,id已经被设置到对象中

system.out.println(“rows inserted = ” + rows);

system.out.println(“generated key value = ” + name.getid());

#### 9、在mapper中如何传递多个参数?

第1种：

//DAO层的函数

Public UserselectUser(String name,String area);

//对应的xml,#{0}代表接收的是dao层中的第一个参数，#{1}代表dao层中第二参数，更多参数一致往后加即可。

<select id="selectUser"resultMap="BaseResultMap">

select \* fromuser\_user\_t whereuser\_name = #{0} anduser\_area=#{1}

</select>

第2种： 使用 @param 注解:

import org.apache.ibatis.annotations.param;

public interface usermapper {

user selectuser(@param(“username”) string username,

@param(“hashedpassword”) string hashedpassword);

}

然后,就可以在xml像下面这样使用(推荐封装为一个map,作为单个参数传递给mapper):

<select id=”selectuser” resulttype=”user”>

select id, username, hashedpassword

from some\_table

where username = #{username}

and hashedpassword = #{hashedpassword}

</select>

#### 10、Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不？

Mybatis动态sql可以让我们在Xml映射文件内，以标签的形式编写动态sql，完成逻辑判断和动态拼接sql的功能。

Mybatis提供了9种动态sql标签：trim|where|set|foreach|if|choose|when|otherwise|bind。

其执行原理为，使用OGNL从sql参数对象中计算表达式的值，根据表达式的值动态拼接sql，以此来完成动态sql的功能。

#### 11、Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复？

不同的Xml映射文件，如果配置了namespace，那么id可以重复；如果没有配置namespace，那么id不能重复；毕竟namespace不是必须的，只是最佳实践而已。

原因就是namespace+id是作为Map<String, MappedStatement>的key使用的，如果没有namespace，就剩下id，那么，id重复会导致数据互相覆盖。有了namespace，自然id就可以重复，namespace不同，namespace+id自然也就不同。

#### 12、为什么说Mybatis是半自动ORM映射工具？它与全自动的区别在哪里？

Hibernate属于全自动ORM映射工具，使用Hibernate查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而Mybatis在查询关联对象或关联集合对象时，需要手动编写sql来完成，所以，称之为半自动ORM映射工具。

#### 13、 一对一、一对多的关联查询 ？

<mapper namespace="com.lcb.mapping.userMapper">

<!--association 一对一关联查询 -->

<select id="getClass" parameterType="int" resultMap="ClassesResultMap">

select \* from class c,teacher t where c.teacher\_id=t.t\_id and c.c\_id=#{id}

</select>

<resultMap type="com.lcb.user.Classes" id="ClassesResultMap">

<!-- 实体类的字段名和数据表的字段名映射 -->

<id property="id" column="c\_id"/>

<result property="name" column="c\_name"/>

<association property="teacher" javaType="com.lcb.user.Teacher">

<id property="id" column="t\_id"/>

<result property="name" column="t\_name"/>

</association>

</resultMap>

<!--collection 一对多关联查询 -->

<select id="getClass2" parameterType="int" resultMap="ClassesResultMap2">

select \* from class c,teacher t,student s where c.teacher\_id=t.t\_id and c.c\_id=s.class\_id and c.c\_id=#{id}

</select>

<resultMap type="com.lcb.user.Classes" id="ClassesResultMap2">

<id property="id" column="c\_id"/>

<result property="name" column="c\_name"/>

<association property="teacher" javaType="com.lcb.user.Teacher">

<id property="id" column="t\_id"/>

<result property="name" column="t\_name"/>

</association>

<collection property="student" ofType="com.lcb.user.Student">

<id property="id" column="s\_id"/>

<result property="name" column="s\_name"/>

</collection>

</resultMap>

</mapper>

#### 14、MyBatis与Hibernate有哪些不同？

Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句，不过mybatis可以通过XML或注解方式灵活配置要运行的sql语句，并将java对象和sql语句映射生成最终执行的sql，最后将sql执行的结果再映射生成java对象。

Mybatis学习门槛低，简单易学，程序员直接编写原生态sql，可严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套sql映射文件，工作量大。

Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用hibernate开发可以节省很多代码，提高效率。但是Hibernate的缺点是学习门槛高，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡，以及怎样用好Hibernate需要具有很强的经验和能力才行。

总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。

#### 15、Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

Mybatis仅支持association关联对象和collection关联集合对象的延迟加载，association指的就是一对一，collection指的就是一对多查询。在Mybatis配置文件中，可以配置是否启用延迟加载lazyLoadingEnabled=true|false。

它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用a.getB().getName()，拦截器invoke()方法发现a.getB()是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用a.setB(b)，于是a的对象b属性就有值了，接着完成a.getB().getName()方法的调用。这就是延迟加载的基本原理。

当然了，不光是Mybatis，几乎所有的包括Hibernate，支持延迟加载的原理都是一样的。

#### 16、Mybatis比IBatis比较大的几个改进是什么

a.有接口绑定,包括注解绑定sql和xml绑定Sql ,

b.动态sql由原来的节点配置变成OGNL表达式,

c. 在一对一,一对多的时候引进了association,在一对多的时候引入了collection节点,不过都是在resultMap里面配置

#### 17、Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页，可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

举例：select \* from student，

拦截sql后重写为：selectt.\* from （select \* from student）t limit0，10

### 4.6 SVN

#### 1. 简述一下SVN和 Git的区别及对比

git是分布式版本控制系统，SVN是集中式版本控制系统。

1．SVN优缺点

优点：

1、 管理方便，逻辑明确，符合一般人思维习惯。

2、 易于管理，集中式服务器更能保证安全性。

3、 代码一致性非常高。

4、 适合开发人数不多的项目开发。

缺点：

1、 服务器压力太大，数据库容量暴增。

2、 如果不能连接到服务器上，基本上不可以工作，看上面第二步，如果服务器不能连接上，就不能提交，还原，对比等等。

3、 不适合开源开发（开发人数非常非常多，但是Google app engine就是用svn的）。但是一般集中式管理的有非常明确的权限管理机制（例如分支访问限制），可以实现分层管理，从而很好的解决开发人数众多的问题。

2．Git优缺点

优点：

1、适合分布式开发，强调个体。

2、公共服务器压力和数据量都不会太大。

3、速度快、灵活。

4、任意两个开发者之间可以很容易的解决冲突。

5、离线工作。

缺点：

1、学习周期相对而言比较长。

2、不符合常规思维。

3、代码保密性差，一旦开发者把整个库克隆下来就可以完全公开所有代码和版本信息。

#### 2、说一下SVN 的工作原理

在svn服务器上每个源代码文件都有一个版本（1、2、3），假设服务器上的代码文件原始版本是1，张三下载下来，修改后这个版本自动+1，变成了2，他把这个2上传到服务器。李四之前也下载了版本为1的这个文件，李四也修改，版本也是2，这时他要上传，服务器发现这个文件版本2在服务器上已经存在，发出警告：版本冲突了。如何解决？合并。

#### 3、你们用SVN的时候是如何解决冲突的

一 : 以我修改的为准，不管服务器目前的那个版本，比如另外同事和我同时修改了同一个地方，但是我觉得我写的更好：

1）右击冲突文件，选择 Mark Resolved ( 标记为解决 ) , 然后选择使用自己的版本

2) 在弹出框中选择第二个（Resolve the conflict by using my version of the file.），点击确定即可

二 : 选择服务器目前最新的版本，比如我原来想改这个文件，后来发现同事已经改好了，就可以选择这个：

1)右击冲突文件，选择 Mark Resolved：

2)在弹出框中选择第三个（Resolve the conflict by using the incoming of the file.），点击确定即可。

三 : 选择服务器的基础版本，这种情况出现比较少，比如我改着改着，后来别人说这个功能不做了，其他同事修改的也要取消，就可以选择这个：

1)右击冲突文件，选择 Mark Resolved：

2)在弹出框中选择第四个（Resolve the conflict by using base version of the file.），点击确定即可。

四 : 这种情况比较常见，主要是这个文件里面，我同事改的也需要，我改的也需要，这样就要进行编辑冲突，将我们的两个文件合成一个：

1)右击冲突文件，选择 Edit Conflicts：

2)在弹出框中，对两个文件进行合并，以左边的那个为准。合并完以后，保存。

3)在弹出框中选择第一个（Conflicts have been resolved in the file.），点击确定即可。

### 4.7 Maven

#### 1、Maven有哪些优点和缺点

优点如下：

简化了项目依赖管理：

易于上手，对于新手可能一个"mvn clean package"命令就可能满足他的工作

便于与持续集成工具（jenkins）整合

便于项目升级，无论是项目本身升级还是项目使用的依赖升级。

有助于多模块项目的开发，一个模块开发好后，发布到仓库，依赖该模块时可以直接从仓库更新，而不用自己去编译。

maven有很多插件，便于功能扩展，比如生产站点，自动发布版本等

缺点如下：

maven是一个庞大的构建系统，学习难度大

maven采用约定优于配置的策略（convention over configuration），虽然上手容易，但是一旦出了问题，难于调试。

当依赖很多时，m2eclipse 老是搞得Eclipse很卡。

中国的网络环境差，很多repository无法访问，比如google code， jboss 仓库无法访问等。

#### 2、Maven坐标

一般maven使用[groupID,artifactId,version，packaging]来表示一个项目的某个版本，有时还会使用classifier来表示项目的附属构建，常见的附属构建有javadoc和sources包。

#### 3、Maven常见的依赖范围有哪些?

compile:编译依赖，默认的依赖方式，在编译（编译项目和编译测试用例），运行测试用例，运行（项目实际运行）三个阶段都有效，典型地有spring-core等jar。

test:测试依赖，只在编译测试用例和运行测试用例有效，典型地有JUnit。

provided:对于编译和测试有效，不会打包进发布包中，典型的例子为servlet-api,一般的web工程运行时都使用容器的servlet-api。

runtime:只在运行测试用例和实际运行时有效，典型地是jdbc驱动jar包。

system: 不从maven仓库获取该jar,而是通过systemPath指定该jar的路径。

import: 用于一个dependencyManagement对另一个dependencyManagement的继承。

#### 4、Maven的生命周期

maven有三套生命周期，分别为：

1、clean 周期：主要用于清理上一次构建产生的文件，可以理解为删除target目录

2、默认周期，

主要阶段包含:

process-resources 默认处理src/test/resources/下的文件，将其输出到测试的classpath目录中,

compile 编译src/main/java下的java文件，产生对应的class,

process-test-resources 默认处理src/test/resources/下的文件，将其输出到测试的classpath目录中,

test-compile 编译src/test/java下的java文件，产生对应的class,

test 运行测试用例,

package 打包构件，即生成对应的jar, war等,

install将构件部署到本地仓库,

deploy 部署构件到远程仓库

3、site周期

主要阶段包含

site 产生项目的站点文档

site-deploy 将项目的站点文档部署到服务器

#### 5、我们经常使用“Mvn Clean Package”命令进行项目打包，请问该命令执行了哪些动作来完成该任务？

在这个命令中我们调用了maven的clean周期的clean阶段绑定的插件任务，以及default周期的package阶段绑定的插件任务

默认执行的任务有（maven的术语叫goal, 也有人翻译成目标，我这里用任务啦）：

maven-clean-plugin:clean->

maven-resources-plugin:resources->

maven-compile-plugin:compile->

mavne-resources-plugin:testResources->

maven-compile-plugin:testCompile->

maven-jar-plugin:jar

#### 6、依赖的解析机制

解析发布版本：如果本地有，直接使用本地的，没有就向远程仓库请求。

解析快照版本：合并本地和远程仓库的元数据文件-groupId/artifactId/version/maven-metadata.xml，这个文件存的版本都是带时间戳的，将最新的一个改名为不带时间戳的格式供本次编译使用。

解析版本为LATEST,RELEASE，过于复杂，且解析的结果不稳定， 不推荐在项目中使用，感兴趣的同学自己去研究，简而言之就是合并groupId/artifactId/maven-metadata.xml找到对应的最新版本和包含快照的最新版本。

#### 7、插件的解析机制

当我们输入"mvn dependency:tree"这样的指令，解析的步骤为：

解析groupID:

maven使用默认的groupID:"org.apache.maven.plugins"或者"org.codehaus.mojo"

解析artifactId(maven的官方叫做插件前缀解析策略)

合并该groupId在所有仓库中的元数据库文件（maven-metadata-repository.xml），比如maven官方插件的元数据文件所在的目录为org\apache\maven\plugins，该文件下有如下的条目

通过比较这样的条目，我们就将该命令的artifactId解析为maven-dependency-plugin

解析version

如果你在项目的pom中声明了该插件的版本，那么直接使用该版本的插件，否则合并所有仓库中groupId/artifactId/maven-metadata-repository.xml，找到最新的发布版本。

对于非官方的插件，有如下两个方法可以选择：

1）使用groupId:artifactId:version:goal 来运行，好长~~~~~~~~~~

2）在Settings.xml中添加pluginGroup项，这样maven不能在官方的插件库中解析到某个插件，那么就可以去你配置的group下查找啦。

#### 8、多模块如何聚合

配置一个打包类型为pom的聚合模块，然后在该pom中使用<module>元素声明要聚合的模块

#### 9、对于一个多模块项目，如果管理项目依赖的版本

通过在父模块中声明dependencyManagement和pluginManagement， 然后让子模块通过<parent>元素指定父模块，这样子模块在定义依赖是就可以只定义groupId和artifactId，自动使用父模块的version,这样统一整个项目的依赖的版本。

#### 10、一个项目的依赖来源于不同的组织，可能这些依赖还会依赖别的Jar包，如何保证这些传递依赖不会引起版本冲突。

使用<dependency>的<exclusion>元素将会引起冲突的元素排除。

#### 11、常见的Maven私服的仓库类型。

（宿主仓库）hosted repository, （代理仓库）proxy repository, （仓库组）group repository

#### 12、如何查询一个插件有哪些目标（Goal）

mvn help:describe -Dplugin=groupId:artifactId

from: http://www.javacoder.cn/?p=211

### 4.8 Git

#### 1、reset 与 rebase, pull 与 fetch 的区别

git reset 不修改commit相关的东西，只会去修改.git目录下的东西。

git rebase 会试图修改你已经commit的东西，比如覆盖commit的历史等，但是不能使用rebase来修改已经push过的内容，容易出现兼容性问题。rebase还可以来解决内容的冲突，解决两个人修改了同一份内容，然后失败的问题。

git pull pull=fetch+merge,

使用git fetch是取回远端更新，不会对本地执行merge操作，不会去动你的本地的内容。 pull会更新你本地代码到服务器上对应分支的最新版本

#### 2、git merge和git rebase的区别

git merge把本地代码和已经取得的远程仓库代码合并。

git rebase是复位基底的意思，gitmerge会生成一个新的节点，之前的提交会分开显示，而rebase操作不会生成新的操作，将两个分支融合成一个线性的提交。

#### 3、git如何解决代码冲突

git stash

git pull

git stash pop

这个操作就是把自己修改的代码隐藏，然后把远程仓库的代码拉下来，然后把自己隐藏的修改的代码释放出来，让gie自动合并。

如果要代码库的文件完全覆盖本地版本。

git reset –hard

git pull

### 4.9 WebService

#### 1、WEB SERVICE 名词解释；JAXP、JAXM 的解释；SOAP、UDDI,WSDL 解释？

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

JAXP(Java API for XML Parsing)定义了在 Java 中使用 DOM, SAX, XSLT 的通用的接口，这样在你的程序中你只要使用这些通用的接口，当你需要改变具 体的实现时候也不需要修改代码；

JAXM(Java API for XML Messaging)是为 SOAP 通信提供访问方法和传输机 制的 API；

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

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

UDDI 的目的是为电子商务建立标准；UDDI 是一套基于 Web 的、分布式的、 为 Web Service 提供的、信息注册中心的实现标准规范，同时也包含一组使企业 能将自身提供的 Web Service 注册，以使别的企业能够发现的访问协议的实现标准。

1、webService概述

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。

WebService实现不同语言间的调用，是依托于一个标准，webservice是需要遵守WSDL（web服务定义语言）/SOAP（简单请求协议）规范的。

WebService=WSDL+SOAP+UDDI（webservice的注册）

Soap是由Soap的part和0个或多个附件组成，一般只有part，在part中有Envelope和Body。

Web Service是通过提供标准的协议和接口，可以让不同的程序集成的一种SOA架构。

Web Service的优点

(1) 可以让异构的程序相互访问（跨平台）

(2) 松耦合

(3) 基于标准协议（通用语言，允许其他程序访问）

Web Service的基本原理

(1) Service Provider采用WSDL描述服务

(2) Service Provider 采用UDDI将服务的描述文件发布到UDDI服务器（Register server）

(3) Service Requestor在UDDI服务器上查询并 获取WSDL文件

(4) Service requestor将请求绑定到SOAP，并访问相应的服务。

#### 2、webservice是什么？

1.基于WEB的服务，服务端整出一些资源让客户端应用访问（提供数据）

2.webservice是一个跨语言跨平台的规范（抽象）

3.是多个跨语言跨平台的应用间通信整合的方案（实际）

#### 3、webservice相当于什么？

http + xml + schema

#### 4、如何发布一个webservice

1.定义SEI（接口） @webservice（类） @webMethod（暴露的方法）

2.定义SEI的实现

3.发布Endpoint.publish(url,new SEI的实现对象)

#### 5、如何请求一个webservice

1.根据wsdl文档生成客户端代码

jdk wsimport -keep wsdl路径

cxf wsdl2java wsdl路径

2.根据生成的代码调用webservice

找到wsdl文档中service标签的name属性对应的类，找到这个port标签的name属性 调用这个方法

#### 6、wsdl（WebService Definition Language）是什么？

1.webservice定义语言，对应.wsdl文档

2.定义了webservice服务器端和客户端应用交互传递请求数据的格式和过程

3.一个webservice对应一个唯一的wsdl文档

#### 7、SOAP（Simple Object Access Protocal）简单对象访问协议

1.是一种简单的，基于HTTP和XML的协议，用于在WEB交换结构化(XML)的数据

2.SOAP消息：请求消息和响应消息

3.HTTP+XML片断

### 4.10 EasyUI

#### 1、easyui如何实现表单验证

提供了一个validatebox插件来验证一个表单 input表单根据validType属性来应用验证 validType="email"邮箱验证 required="true"必填项

<input class="easyui-validatebox" type="text" name="name" validType="email" required="true"></input>

#### 2、easyui如何实现表格分页

将pagination属性设为true 在数据网格下生成一个分页工具栏 工具栏会发送连个参数到服务器 page当前页 默认1 rows：每页显示行

### 4.11 Bootstrap

#### 1、bootstrap的特点

1.响应式布局

概念：响应式web布局是让用户通过不同尺寸的浏览器都可以获得良好视觉的一种方法。是目前比较流行的一种布局方法。

实现原理：通过CSS3 Media Queries（媒体（设备）查询），媒体查询是让页面内容在不同的媒体环境下运行时可以展示不同的样式（这个样式当然是由我们来书写规定的）。@media是CSS3中规定的属性，它可以实现针对不同媒体设备来设置不同的样式的目的。而且就算是在同一设备中它也可以在你重置浏览器大小的过程中，页面也会根据浏览器的宽度和高度重新渲染页面。

Bootstrap主要用到min-width、max-width，以及and语法，用于在不同的分辨率下设置不同的CSS样式。

@media的语法

@media mediatype and|not|only (media feature) {

CSS-Code;

}

其中mediatype有print（打印设备）、screen（用于电脑屏幕，平板电脑，智能手机等）、speech（应用于屏幕阅读器等发声设备）；media feature则是用来规定如最大宽度或者最小宽度。

我们来看看bootstrap中布局容器的例子：

Bootstrap 需要为页面内容和栅格系统包裹一个 .container 容器。

如下

固定宽度布局

<div class="container">

...

</div>

或者 流式布局

<div class="container-fluid">

...

</div>

在bootstrap的css文档中@media属性

@media (min-width: 768px) {

.container {

width: 750px;

}

}

@media (min-width: 992px) {

.container {

width: 970px;

}

}

@media (min-width: 1200px) {

.container {

width: 1170px;

}

}

……

以上代码实现了随浏览器宽度的变化container容器的宽度也进行变化。

#### 2、移动设备优先

在html文件中head区域加入这样的一个meta标签，name=“viewpoint”是指这个标签对移动设备生效，content中width=device-width是指宽度为设备宽度，initial-scale=1意思是初始缩放比例为1.

<meta name="viewport" content="width=device-width, initial-scale=1">

但是在css文档中没有对超小屏幕定义任何媒体查询相关的代码，因为这在 Bootstrap 中是默认的移动设备优先！

/\* 超小屏幕（手机，小于 768px） \*/

/\* 没有任何媒体查询相关的代码，因为这在 Bootstrap 中是默认的（还记得 Bootstrap 是移动设备优先的吗？） \*/

/\* 小屏幕（平板，大于等于 768px） \*/

@media (min-width: @screen-sm-min) { ... }

/\* 中等屏幕（桌面显示器，大于等于 992px） \*/

@media (min-width: @screen-md-min) { ... }

/\* 大屏幕（大桌面显示器，大于等于 1200px） \*/

@media (min-width: @screen-lg-min) { ... }

#### 3、data-\*属性的使用

data属性是HTML5中定义的一个全局属性，它用来存在页面或者应用程序的私有自定义数据。我们可以为所有 HTML 元素上嵌入自定义 data 属性，自定义（存储）的数据可以被CSS或者JS来使用，来提高用户体验。

data-\* 属性包括两部分：

属性名不应该包含任何大写字母，并且在前缀 “data-” 之后必须有至少一个字符

属性值可以是任意字符串

data属性是bootstrap中应用很多的一个属性，它可以让开发者仅仅通过data属性API就能使用所有Bootstrap中的插件，而且不用写一行JavaScript代码。

bootstap中封装的data-\*属性的API

\*\*css中\*\*

[data-toggle="buttons"] > .btn input[type="radio"],

[data-toggle="buttons"] > .btn-group > .btn input[type="radio"],

[data-toggle="buttons"] > .btn input[type="checkbox"],

[data-toggle="buttons"] > .btn-group > .btn input[type="checkbox"] {

position: absolute;

clip: rect(0, 0, 0, 0);

pointer-events: none;

}

\*\*js中\*\*

[data-toggle="buttons"

[data-toggle="collapse"]

[data-toggle="dropdown"]

[data-toggle="modal"]

[data-toggle="tab"]

[data-toggle="pill"]

#### 4、栅格系统

参考：http://www.jb51.net/css/362199.html

栅格将一个页面可以拆分成多个区块来理解，而正是这些区块共同构成了真个页面的布局。根据不同的屏幕尺寸情况，调整这些区块的排版，就可以实现响应式设计。另外，屏幕宽度较大的时候，区块倾向于水平分布，而屏幕宽度较小的时候，区块倾向于竖直堆叠。

栅格样式库一般是这样做的：将页面划分为若干等宽的列（column），然后推荐你通过等宽列来创建响应式的页面区块。

Bootstrap把它的栅格放在CSS这个分类下，并称它为Gird system。默认分为12列。

要点一：容器、行与列

<div class="container">

<div class="row">

<div class="col-md-6"></div>

<div class="col-md-6"></div>

</div>

</div>

container、row、column必须保持特定的层级关系，栅格系统才可以正常工作。

要点二：断点类型

Bootstrap栅格的column对应的类名形如.col-xx-y。y是数字，表示该元素的宽度占据12列中的多少列。而xx只有特定的几个值可供选择，分别是xs、sm、md、lg，它们就是断点类型。

在Bootstrap栅格的设计中，断点的意义是，当视口（viewport）宽度小于断点时，column将竖直堆叠（display: block的默认表现），而当视口宽度大于或等于断点时，column将水平排列（float的效果）。按照xs、sm、md、lg的顺序，断点像素值依次增大，其中xs表示极小，即认为视口宽度永远不小于xs断点，column将始终水平浮动。

有时候，会需要将多种断点类型组合使用，以实现更细致的响应式设计。此时不同的断点类型之间会有怎样的相互作用呢？

先看看Bootstrap的sass源码是如何定义栅格的：

@include make-grid-columns;

@include make-grid(xs);

@media (min-width: $screen-sm-min) {

@include make-grid(sm);

}

@media (min-width: $screen-md-min) {

@include make-grid(md);

}

@media (min-width: $screen-lg-min) {

@include make-grid(lg);

}

可以看到，用了min-width的写法，而且断点像素值越大的，对应代码越靠后。所以，如果有这样的一些元素：

<div class="container">

<div class="row">

<div class="col-sm-6 col-lg-3">1</div>

<div class="col-sm-6 col-lg-3">2</div>

<div class="col-sm-6 col-lg-3">3</div>

<div class="col-sm-6 col-lg-3">4</div>

</div>

</div>

结合前面的源码，可以想到，在上面这样视口宽度由小变大的过程中，首先是保持默认的竖直堆叠，然后超过了sm的断点，sm的样式生效，变为一行两列的排版，再继续超过lg的断点后，lg的样式也生效，由于lg的样式代码定义在sm之后，所以会覆盖掉sm的样式，从而得到一行四列的排版。

所以，结合使用多个断点类型，就可以引入多个断点变化，把响应式做得更加细致。

### 4.12 Echarts

#### 1、你了解Echarts吗 简单谈谈你是如何使用的

模块化单文件引入

1、新建一个echarts.html文件，为ECharts准备一个具备大小（宽高）的Dom。

2、新建<script>标签引入模块化单文件echarts.js

3、新建<script>标签中为模块加载器配置echarts和所需图表的路径（相对路径为从当前页面链接到echarts.js），引入图表文件见引入ECharts2

4、<script>标签内动态加载echarts和所需图表，回调函数中可以初始化图表并驱动图表的生成，option见API & Doc

### 4.13 Shiro

#### 1、什么是shiro

shiro是Apache的一个开源框架，它将软件系统的安全认证相关的功能抽取出来，实现用户身份认证，权限授权、加密、会话管理等功能，组成了一个通用的安全认证框架。它可以实现如下的功能： 1.验证用户 2.对用户执行访问控制，如：判断用户是否具有角色admin，判断用户是否拥有访问的资源权限。 3.在任何环境下使用SessionAPI。例如C/S程序 4.可以使用多个用户数据源。例如一个是Oracle数据库，另外一个是MySQL数据库。 5.单点登录(SSO)功能 6.”Remember Me”服务，类似于购物车的功能，shiro官方建议开启。

#### 2、为何使用shiro?

因为shiro将安全认证相关的功能抽取出来组成了一个框架，因此使用shiro就可以非常快速的完成认证、授权等功能的开发，降低系统成本。 shiro使用广泛，shiro可以运行在web应用，非web应用，集群分布式应用中越来越多的用户开始使用shiro. Spring中有Spring security(原名Acegi), 是一个权限框架，它和Spring的依赖过于紧密，没有shiro使用简单。shiro不依赖于Spring，而shiro就相对独立最主要是因为shiro使用简单‘灵活，所以现在越来越多的用户选择shiro。

#### 3、shiro核心组成部分

shiro的4大组成部分——身份认证，授权，会话管理和加密 Authentication：身份验证(身份认证)，简称”登录”。 Authorization：授权，给用户给用户分配角色或者权限资源。 Session Manager：用户Session管理器，可以让C/S程序也使用Session来控制权限。 Cryptography：将JDK中复杂的密码加密方式进行封装。 除了以上功能，shiro还提供很多扩展功能： Web Support：主要针对web应用提供一些常用功能。 Caching：缓存可以使程序运行更有效率。 Concurrency：多线程相关功能。 Testing：帮助我们进行测试相关的功能。 “Run As”：一个允许用户假设为另一个用户身份(如果允许)的功能，有时候在管理脚本时很有用。 “Remember Me”：记住用户身份，提供类似购物车的功能。

#### 4、shiro运行大致流程：

Subject——主体，是与程序进行交互的对象，可以是人也可以是服务或其他程序，通常理解为用户。所有的Subject实例都必须绑定到一个SecurityManager上。我们与Subject交互，运行时shiro会自动转化为与SecurityManager交互的特定的subject的交互。SecurityManager——SecurityManager是shiro的核心，初始化时协调各个模块运行。然而，一旦SecurityManager协调完毕，SecurityManager会被单独留下，且我们只需要去操作Subject即可，无需操作SecurityManager。但是需要了解的是当我们与一个Subject进行交互时，实质上是SecurityManager在处理Subject的安全操作。 Realms——Realms在shiro中作为程序和安全数据之间的”桥梁”或”连接器”。它用于获取安全数据来判断subject是否能够登录，subject拥有什么权限。有点类似于DAO。在配置realms时，需要至少一个realm。而且shiro提供了一些常用的Realms来连接数据源，如LDAP数据源的JndiLdapRealm，JDBC数据源的JdbcRealm，ini文件数据源的iniRealm，Properties文件数据源的PropertiesRealm，等等，我们也可以插入自己的Realm实现来代表自定义的数据源。像其他组件一样，Realms也是由SecurityManager控制。

#### 5、shiro架构

Subject：(org.apache.shiro.subject.Subject) 即主体，简称用户，主体既可以是用户也可以是程序，主体访问系统，系统需要对主体进行认证、授权。 外部应用与subject进行交互，Subject记录了当前操作用户，将用户的概念理解为当前操作的主体，可能是一个通过浏览器请求的用户，也可能是一个运行的程序。 Subject在shiro中是一个接口，接口中定义了很多认证授权相关的方法，外部程序通过subject进行认证授权，而Subject是通过SecurityManager安全管理器进行认证授权。 SecurityManager：(org.apache.shiro.mgt.SecurityManager)如上所述，SecurityManager是shiro的核心，协调shiro的各个组件。SecurityManager就是安全管理器，负责对全部的subject进行安全管理。通过SecurityManager可以完成Subject的认证、授权等，实质上SecurityManager是通过Authenticator对主体进行认证，通过Authorizer对主体进行授权，通过SessionManager进行会话管理等等。SecurityManager是一个接口，继承了Authenticator，Authorizer，SessionManager这三个接口。 Authenticator：(org.apache.shiro.authc.Authenticator) 即认证器，对用户身份进行认证，Authenticator是一个接口，shiro提供ModularRealmAuthenticator实现类，通过ModularRealmAuthenticator基本上可以满足大多数需求，也可以自定义认证器。 Authorizer：(org.apache.shiro.authz.Authorizer)即授权器，用户在通过认证器认证通过后，在访问时需要通过授权器判断用户是否有此功能的操作权限。最终是通过认证器对主体进行授权的。 Realm：(org.apache.shiro.realm.Realm)Realm即领域，相当于DataSource数据源，通过Realm存取认证、授权相关数据。SecurityManager通过认证器对主体进行安全认证需要通过Realm获取用户身份数据，比如：如果用户身份数据在数据库，那么Realm就需要从数据库获取用户的身份信息。授权也是如此，也需要通过Realm取出授权相关信息。注意：不要将Realm理解成只是从数据源获取数据，在Realm中还有认证授权校验的相关代码 SessionManager：(org.apache.shiro.session.SessionManager)会话管理。web应用中一般是web容器对Session进行管理，shiro框架定义了一套会话管理，它不依赖于web容器的Session，所以shiro可以使用在非web应用上，也可以将分布式应用的会话集中在一点进行管理，此特性可使它实现单点登录。 SessionDAO：SessionDAO即会话dao，是对Session会话操作的一套接口，比如要将Session存储到数据库，可以通过JDBC将会话存储到数据库。针对个性化的Session数据存储(存到数据库)需要使用SessionDAO。 CacheManager：(org.apahce.shiro.cache.CacheManager)缓存管理器，主要对Session和授权数据进行缓存，比如将授权数据通过cachemanager进行缓存管理，和ehcache集成对缓存数据进行管理，可以减少不必要的后台访问，提高应用效率，增加用户体验。 Cryptography：(org.apache.shiro.crypto.\*)密码管理，提供了一套加密/解密组件，对JDK中的加密解密算法进行了封装，方便开发。比如提供常用的散列、加/解密等功能，比如MD5散列算法。

#### 6、shiro相关jar

与其他Java开源框架类似，将shiro的jar包加入项目就可以使用shiro提供的功能了。shiro-core时核心包必须选用，还提供了与web集成的shiro-web、与Spring集成的shiro-spring、与任务调度quartz集成的shiro-quartz、与ehcache集成的shiro-ehcache。下边是shiro各个jar包的Maven坐标。

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-ehcache</artifactId>

<version>1.2.3</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-quartz</artifactId>

<version>1.2.3</version>

</dependency>

也可以通过引入shiro-all包括shiro所有的包

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-all</artifactId>

<version>1.2.3</version>

</dependency>

#### 7、shiro认证过程

1.应用程序构建了一个终端用户认证信息AuthenticationToken实例后，调用Subject.login方法。 2.Subject的实例通常是DelegatingSubject类(或子类)的实例对象，在认证开始时，会委托应用程序设置的SecurityManager实例调用securityManager.login(token)方法。 3.SecurityManager接受到token(令牌)信息后会委托内置的Authenticator的实例(通常是ModularRealmAuthenticator类的实例)调用authenticator.authenticate(token)。ModularRealmAuthenticator在认证过程中会对设置的一个或多个Realm实例进行适配，它实际上为shiro提供了一个可插拔的认证机制。 4.如果在应用程序中配置类多个Realm，ModularRealmAuthenticator会根据配置的AuthenticationStrategy(认证策略)来进行多Realm的认证过程。在Realm被调用后，AuthenticationStrategy将对每一个Realm的结果做出响应。注意：如果应用程序中仅配置了一个Realm，Realm将被直接调用而无需再配置认证策略。 5.判断每一个Realm是否支持提交的token，如果支持，Realm将调用getAuthenticationInfo(token)；getAuthenticationInfo方法就是实际认证处理，我们通过覆盖Realm的doGetAuthenticationInfo方法来编写我们自定义的认证处理。

#### 8、shiro入门程序(用户登录和退出)

创建Java工程

加入shiro-core的jar包以及依赖包

工程结构如下：

log4j.properties日志配置文件

log4j.rootLogger=debug, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m %n

shiro-first.ini

通过shiro-first.ini配置文件初始化SecurityManager环境，创建SecurityManager工厂。 配置MyEclipse支持ini，添加中文支持插件Properties Editor 地址是 http://propedit.sourceforge.jp/eclipse/updates/

在MyEclipse配置后，在classpath创建shiro.ini配置文件，为了方便测试将用户名和密码配置在shiro-first.ini文件中

#对用户信息进行配置

[users]

#用户账号和密码

zhangsan=123456

lisi=654321

入门程序认证代码

// 用户登录和退出

@Test

public void testLoginAndLogout() {

// 创建SecurityManager工厂。通过ini配置文件创建securityManager

Factory<SecurityManager> factory = new IniSecurityManagerFactory("classpath:shiro-first.ini");

// 创建SecurityManager

SecurityManager securityManager = factory.getInstance();

// 将securityManager设置到当前的运行环境中

SecurityUtils.setSecurityManager(securityManager);

// 从SecurityUtils中创建一个subject

Subject subject = SecurityUtils.getSubject();

// 在认证提交前准备token(令牌)

// 这里的账号和密码 将来是由用户输入进去的

UsernamePasswordToken token = new UsernamePasswordToken("zhangsan", "123456");

//执行认证提交

try {

//执行认证提交

subject.login(token);

} catch (AuthenticationException e) {

e.printStackTrace();

}

// 是否认证通过

boolean isAuthenticated = subject.isAuthenticated();

System.out.println("是否认证通过："+isAuthenticated);

//退出操作

subject.logout();

// 是否认证通过

isAuthenticated = subject.isAuthenticated();

System.out.println("是否认证通过："+isAuthenticated);

}

#### 9、认证执行流程

1.通过ini配置文件创建SecurityManager 2.创建token令牌，token中有用户提交的认证信息即用户名和密码 3.执行subject.login(token)方法提交认证，最终由securityManager通过Authenticator进行认证 4.Authenticator的实现ModularRealmAuthenticator调用realm从ini配置文件取出用户真实的账号和密码，这里使用的是iniRealm(shiro自带) 5.initRealm先根据先根据token中的账号去ini中查找账号，如果查找不到则给ModularRealmAuthenticator返回null，如果查到用户信息，就给ModularRealmAuthenticator返回用户信息(账号和密码) 6.ModularRealmAuthenticator接收IniRealm返回Authentication认证信息，如果返回的认证信息是null，ModularRealmAuthenticator抛出异常（org.apache.shiro.authc.UnknownAccountException）如果返回的认证信息不是null（说明inirealm找到了用户），对IniRealm返回用户密码 （在ini文件中存在）和 token中的密码 进行对比，如果不一致抛出异常（org.apache.shiro.authc.IncorrectCredentialsException）

#### 10、常见的认证异常

UnknownAccountException

账号不存在异常如下：org.apache.shiro.authc.UnknownAccountException: No account found for user…

IncorrectCredentialsException

当输入密码错误会抛此异常，如下：org.apache.shiro.authc.IncorrectCredentialsException: Submitted credentials for token [org.apache.shiro.authc.UsernamePasswordToken – zhangsan, rememberMe=false] did not match the expected credentials.

更多如下：

DisabledAccountException（帐号被禁用）

LockedAccountException（帐号被锁定）

ExcessiveAttemptsException（登录失败次数过多）

ExpiredCredentialsException（凭证过期）等

小结

ModularRealmAuthenticator作用进行认证，需要调用realm查询用户信息（在数据库中存在用户信息）

ModularRealmAuthenticator进行密码对比（认证过程）。

realm：需要根据token中的身份信息去查询数据库（入门程序使用ini配置文件），如果查到用户返回认证信息，如果查询不到返回null

### 4.14 Activiti

#### 1.什么是Activiti?

Activiti是android的四大组件中最常见的部分，是Android用于交互的部分。Activiti有四种状态：Active/Runing、Paused、Stoped、Killed。主要有七个生命周期方法。通过Intent与其他activiti或者组件通信。

#### 2、请描述一下Activiti生命周期。

1、onCreate()

当Activiti被创建时，系统自动调用onCreate()方法进行一些初始化操作，比如创建views，设置数据到list等。该方法提供了一个Bundle类型的变量，该变量中有这个activiti以前的状态信息，前提是以前存过这些信息。这个方法执行完后执行的是onStart()方法;若在onCreate方法中加入finish()方法，onCreate下一个运行onDestroy方法。

以上是官方SDK上的说明，onCreate方法是整个生命周期中唯一必须实现的方法。布局都是在activiti中进行，初始的数据也是在这里进行，较大量的数据需要异步加载handler。

2、onRestart()

当Activitiy被停止时，调用这个方法进行唤醒，该方法优先于onStart()方法。若在onRestart()方法中加入finish()语句，则还是会继续运行onStart及后面的状态方法直到onDestroy运行完。

onRestart（）是activiti的一个生命周期，基类中就只做了一件事情就是把mCalled设置为true代表界面当前使用状态。其他的操作都是在activitimanagerservice中去执行的，service中执行的代码也不是说属于执行这个生命周期的方法，它的意思是服务将这个界面从堆栈中拿出来重新放到前台可见的时候允许客户端程序做一个自定义事情的回调方法。你需要先理解activiti的生命周期的含义，它只是服务端控制activiti状态时允许客户端activiti在相对应的时刻添加一些自己要做的事情的回调方法而已。

onRestart 当处于非栈顶状态的活动需要再次返回栈顶，展现给用户的时候，触发该方法。

也就是说执行了onStop()且没有执行onDestroy()的activiti被重新激活时，就会调用onRestart()方法。

生命周期只有一种：Running状态->onPause()->onStop()->不可见状态->onRestart()->onStart->onResume()->获得焦点可交互状态。

onRestart()方法一般是进行一些数据的初始化操作。onStop方法会将activiti中的资源释放，所以需要在onRestart方法中重新初始化。但是这些资源的申请操作一般在onStart方法中进行，onRestart()方法并不常用。

3、onStart()

当activiti对用户可见时会调用onStart，当activiti在前台显示时，会运行onResume;当activiti还没在前台显示就被隐藏(停止状态)了会运行onStop()，比如在onStart方法中用了finish()方法的话，onStart()之后就会直接运行onStop->onDestroy。

当进入onStart状态时，activiti将可见但是不能交互。可以把onStart和onStop看成一对。

4、onResume()

当activiti开始与用户交互时会调用onResume，并且为了用户操作此时该activiti位于activiti栈的顶部。经过某些操作后该方法执行完后执行的是onPause()。

当activiti被一个透明或者Dialog的activiti覆盖时将处于Pause状态，它仍然可见，但是已经失去了焦点，不可与用户交互。通过调用onResume()方法使activiti从Pause恢复为Active状态。

例如在一个Dialog中保存数据，并刷新Acitivity的数据，则需要在onResume()方法中进行数据查询并显示。

5、onPause()

当一个activiti运行到onResume方法后，不管是这个activiti要销毁还是要暂停或停止，都会调用该方法。这个方法之后有可能是onResume也有可能是onStop，若是在这个activiti-A中打开一个不完全覆盖这个activiti-A的新activiti-B，那么activiti-A就会是onPause状态，当activiti-B退出时，activiti-A就直接运行onResume(前提是onPause的方法执行完了，否则会等onPause方法执行完后才运行onResume方法，所以不建议在这个方法中执行CPU密集的操作)。若是需要退出activiti-A，那么下一个就会执行onStop。onPause()用于提交未保存发生变化了的持久化数据，及停止动画及其他其他比较消耗CPU的事件(比如广播接收器，传感器(比如GPS)，或者消耗电量的资源)，这是为了更好的运行新的activiti。

在onPause方法中可以保存一些状态信息，用于恢复时使用，例如输入框中已经输入的内容，在这个方法中保存。当Acivity恢复时，可以在onRestart()或者onStart()方法中将数据恢复。

6、onStop()

当这个activiti完全看不见的时候，会调用onStop方法，因为另一个activiti会调用onResume并且覆盖这个activiti。以下三种情况都会使这个activiti调用onStop()方法，第一种是一个新的activiti被执行，第二种是一个已经存在的activiti被切换到最前端，第三种是这个activiti要被销毁。如果通过用户召回这个activiti，那么会调用onRestart方法;若这个activiti要被销毁，则调用onDestroy方法。

当用户自己退出程序的时候，建议在onStop方法中保存数据。

7、onDestory()

当activiti销毁前会调用该方法，比如发生如下情况：activiti调用了finish()方法来结束这个activiti，或者因为系统为了节省空间而临时销毁这个activiti，这两个情况可以通过isFinishing()方法判断。

#### 3、两个Activiti之间跳转时必然会执行的是哪几个方法。

这个问题与 从A跳转到B需要执行那些生命周期方法 的意思是相同的。分为两种情况：B不透明时，A执行除onDestory之外的全部方法，B执行到onResume方法；B透明时，A执行到onPause方法，B执行到onResume方法。

#### 4、何将一个activiti设置成窗口的样式。

在AndroidManifest.xml中在你需要显示为窗口的activiti中添加如果属性：android:theme="@style/Theme.Floatactiviti" 即可

#### 5、你后台的activiti被系统回收怎么办？如果后台的activiti由于某原因被系统回收可了，如何在被系统回收之前保存当前状态？

onSaveInstanceState在这个方法中保存信息。在onCreate中判断savedInstanceState是否为空，不为空就去出来。

#### 6.如何退出activiti？如何安全退出已调用多个activiti的Application？

退出单个activiti调用finish()方法。

#### 7.两个activiti之间怎么传递数据？

可以通过Intent，Bundle或者SharedPreferences都可以传递数据。

#### 8.怎么在启动一个activiti时就启动一个service？

在activiti的onCreate里写

startService(xxx);

然后

this.finish();结束自己..

这是最简单的方法 可能会有屏幕一闪的现象，如果UI要求严格的话用AIDL把

根据service与activiti的生命周期，选择在onCreate或onResume中startService;当然要记得stopService.

#### 9.同一个程序，但不同的activiti是否可以放在不同的Task任务栈中？

可以放在不同的Task中。需要为不同的activiti设置不同的taskaffinity属性，启动activiti的Intent需要包含FLAG\_activiti\_NEW\_TASK标记

#### 10.activiti怎么和service绑定，怎么在activiti中启动自己对应的service？

startService()一旦被创建 调用着无关 没法使用service里面的方法

bindService () 把service 与调用者绑定,如果调用者被销毁, service会销毁

bindService() 我们可以使用service 里面的方法

bindService(). 让activiti能够访问到service里面的方法

构建一个intent对象,

Intent service = new Intent(this,MyService.class);

通过bindService的方法去启动一个服务,

bindService(intent, new MyConn(), BIND\_AUTO\_CREATE);

ServiceConnection 对象(重写onServiceConnected和OnServiceDisconnected方法) 和BIND\_AUTO\_CREATE.

private class myconn implements ServiceConnection

{

public void onServiceConnected(ComponentName name,IBinder service) {

// TODO Auto-generated method stub

//可以通过IBinder的对象 去使用service里面的方法

}

public void onServiceDisconnected(ComponentName name) {

// TODO Auto-generated method stub

}

}

### 4.15 Quartz

#### 1、Quartz可以用来做什么？

Quartz是一个任务调度框架。比如你遇到这样的问题

想每月25号，信用卡自动还款

想每年4月1日自己给当年暗恋女神发一封匿名贺卡

想每隔1小时，备份一下自己的爱情动作片 学习笔记到云盘

这些问题总结起来就是：在某一个有规律的时间点干某件事。并且时间的触发的条件可以非常复杂（比如每月最后一个工作日的17:50），复杂到需要一个专门的框架来干这个事。 Quartz就是来干这样的事，你给它一个触发条件的定义，它负责到了时间点，触发相应的Job起来干活。

一个简单的示例

这里面的所有例子都是基于Quartz 2.2.1

package com.test.quartz;

import static org.quartz.DateBuilder.newDate;

import static org.quartz.JobBuilder.newJob;

import static org.quartz.SimpleScheduleBuilder.simpleSchedule;

import static org.quartz.TriggerBuilder.newTrigger;

import java.util.GregorianCalendar;

import org.quartz.JobDetail;

import org.quartz.Scheduler;

import org.quartz.Trigger;

import org.quartz.impl.StdSchedulerFactory;

import org.quartz.impl.calendar.AnnualCalendar;

public class QuartzTest {

public static void main(String[] args) {

try {

//创建scheduler

Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();

//定义一个Trigger

Trigger trigger = newTrigger().withIdentity("trigger1", "group1") //定义name/group

.startNow()//一旦加入scheduler，立即生效

.withSchedule(simpleSchedule() //使用SimpleTrigger

.withIntervalInSeconds(1) //每隔一秒执行一次

.repeatForever()) //一直执行，奔腾到老不停歇

.build();

//定义一个JobDetail

JobDetail job = newJob(HelloQuartz.class) //定义Job类为HelloQuartz类，这是真正的执行逻辑所在

.withIdentity("job1", "group1") //定义name/group

.usingJobData("name", "quartz") //定义属性

.build();

//加入这个调度

scheduler.scheduleJob(job, trigger);

//启动之

scheduler.start();

//运行一段时间后关闭

Thread.sleep(10000);

scheduler.shutdown(true);

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.test.quartz;

import java.util.Date;

import org.quartz.DisallowConcurrentExecution;

import org.quartz.Job;

import org.quartz.JobDetail;

import org.quartz.JobExecutionContext;

import org.quartz.JobExecutionException;

public class HelloQuartz implements Job {

public void execute(JobExecutionContext context) throws JobExecutionException {

JobDetail detail = context.getJobDetail();

String name = detail.getJobDataMap().getString("name");

System.out.println("say hello to " + name + " at " + new Date());

}

}

这个例子很好的覆盖了Quartz最重要的3个基本要素：

Scheduler：调度器。所有的调度都是由它控制。

Trigger： 定义触发的条件。例子中，它的类型是SimpleTrigger，每隔1秒中执行一次（什么是SimpleTrigger下面会有详述）。

JobDetail & Job： JobDetail 定义的是任务数据，而真正的执行逻辑是在Job中，例子中是HelloQuartz。 为什么设计成JobDetail + Job，不直接使用Job？这是因为任务是有可能并发执行，如果Scheduler直接使用Job，就会存在对同一个Job实例并发访问的问题。而JobDetail & Job 方式，sheduler每次执行，都会根据JobDetail创建一个新的Job实例，这样就可以规避并发访问的问题。

#### 2、Quartz API

Quartz的API的风格在2.x以后，采用的是DSL风格（通常意味着fluent interface风格），就是示例中newTrigger()那一段东西。它是通过Builder实现的，就是以下几个。（\*\* 下面大部分代码都要引用这些Builder \*\* )

//job相关的builder

import static org.quartz.JobBuilder.\*;

//trigger相关的builder

import static org.quartz.TriggerBuilder.\*;

import static org.quartz.SimpleScheduleBuilder.\*;

import static org.quartz.CronScheduleBuilder.\*;

import static org.quartz.DailyTimeIntervalScheduleBuilder.\*;

import static org.quartz.CalendarIntervalScheduleBuilder.\*;

//日期相关的builder

import static org.quartz.DateBuilder.\*;

DSL风格写起来会更加连贯，畅快，而且由于不是使用setter的风格，语义上会更容易理解一些。对比一下：

JobDetail jobDetail=new JobDetailImpl("jobDetail1","group1",HelloQuartz.class);

jobDetail.getJobDataMap().put("name", "quartz");

SimpleTriggerImpl trigger=new SimpleTriggerImpl("trigger1","group1");

trigger.setStartTime(new Date());

trigger.setRepeatInterval(1);

trigger.setRepeatCount(-1);/

#### 3、关于name和group

JobDetail和Trigger都有name和group。

name是它们在这个sheduler里面的唯一标识。如果我们要更新一个JobDetail定义，只需要设置一个name相同的JobDetail实例即可。

group是一个组织单元，sheduler会提供一些对整组操作的API，比如 scheduler.resumeJobs()。

#### 6、优先级（Priority）

当scheduler比较繁忙的时候，可能在同一个时刻，有多个Trigger被触发了，但资源不足（比如线程池不足）。那么这个时候比剪刀石头布更好的方式，就是设置优先级。优先级高的先执行。

需要注意的是，优先级只有在同一时刻执行的Trigger之间才会起作用，如果一个Trigger是9:00，另一个Trigger是9:30。那么无论后一个优先级多高，前一个都是先执行。

优先级的值默认是5，当为负数时使用默认值。最大值似乎没有指定，但建议遵循Java的标准，使用1-10，不然鬼才知道看到【优先级为10】是时，上头还有没有更大的值。

#### 7、Misfire(错失触发）策略

类似的Scheduler资源不足的时候，或者机器崩溃重启等，有可能某一些Trigger在应该触发的时间点没有被触发，也就是Miss Fire了。这个时候Trigger需要一个策略来处理这种情况。每种Trigger可选的策略各不相同。

这里有两个点需要重点注意：

MisFire的触发是有一个阀值，这个阀值是配置在JobStore的。比RAMJobStore是org.quartz.jobStore.misfireThreshold。只有超过这个阀值，才会算MisFire。小于这个阀值，Quartz是会全部重新触发。

所有MisFire的策略实际上都是解答两个问题：

已经MisFire的任务还要重新触发吗？

如果发生MisFire，要调整现有的调度时间吗？

比如SimpleTrigger的MisFire策略有：

MISFIRE\_INSTRUCTION\_IGNORE\_MISFIRE\_POLICY

这个不是忽略已经错失的触发的意思，而是说忽略MisFire策略。它会在资源合适的时候，重新触发所有的MisFire任务，并且不会影响现有的调度时间。

比如，SimpleTrigger每15秒执行一次，而中间有5分钟时间它都MisFire了，一共错失了20个，5分钟后，假设资源充足了，并且任务允许并发，它会被一次性触发。

这个属性是所有Trigger都适用。

MISFIRE\_INSTRUCTION\_FIRE\_NOW

忽略已经MisFire的任务，并且立即执行调度。这通常只适用于只执行一次的任务。

MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT

将startTime设置当前时间，立即重新调度任务，包括的MisFire的

MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_REMAINING\_REPEAT\_COUNT

类似MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT，区别在于会忽略已经MisFire的任务

MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_EXISTING\_COUNT

在下一次调度时间点，重新开始调度任务，包括的MisFire的

MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_REMAINING\_COUNT

类似于MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_EXISTING\_COUNT，区别在于会忽略已经MisFire的任务。

MISFIRE\_INSTRUCTION\_SMART\_POLICY

所有的Trigger的MisFire默认值都是这个，大致意思是“把处理逻辑交给聪明的Quartz去决定”。基本策略是，

如果是只执行一次的调度，使用MISFIRE\_INSTRUCTION\_FIRE\_NOW

如果是无限次的调度(repeatCount是无限的)，使用MISFIRE\_INSTRUCTION\_RESCHEDULE\_NEXT\_WITH\_REMAINING\_COUNT

否则，使用MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT

MisFire的东西挺繁杂的，可以参考这篇

#### 8、Calendar

这里的Calendar不是jdk的java.util.Calendar，不是为了计算日期的。它的作用是在于补充Trigger的时间。可以排除或加入某一些特定的时间点。

以”每月25日零点自动还卡债“为例，我们想排除掉每年的2月25号零点这个时间点（因为有2.14，所以2月一定会破产）。这个时间，就可以用Calendar来实现。

例子：

AnnualCalendar cal = new AnnualCalendar(); //定义一个每年执行Calendar，精度为天，即不能定义到2.25号下午2:00

java.util.Calendar excludeDay = new GregorianCalendar();

excludeDay.setTime(newDate().inMonthOnDay(2, 25).build());

cal.setDayExcluded(excludeDay, true); //设置排除2.25这个日期

scheduler.addCalendar("FebCal", cal, false, false); //scheduler加入这个Calendar

//定义一个Trigger

Trigger trigger = newTrigger().withIdentity("trigger1", "group1")

.startNow()//一旦加入scheduler，立即生效

.modifiedByCalendar("FebCal") //使用Calendar !!

.withSchedule(simpleSchedule()

.withIntervalInSeconds(1)

.repeatForever())

.build();

Quartz体贴地为我们提供以下几种Calendar，注意，所有的Calendar既可以是排除，也可以是包含，取决于：

HolidayCalendar。指定特定的日期，比如20140613。精度到天。

DailyCalendar。指定每天的时间段（rangeStartingTime, rangeEndingTime)，格式是HH:MM[:SS[:mmm]]。也就是最大精度可以到毫秒。

WeeklyCalendar。指定每星期的星期几，可选值比如为java.util.Calendar.SUNDAY。精度是天。

MonthlyCalendar。指定每月的几号。可选值为1-31。精度是天

AnnualCalendar。 指定每年的哪一天。使用方式如上例。精度是天。

CronCalendar。指定Cron表达式。精度取决于Cron表达式，也就是最大精度可以到秒。

#### 9、Trigger实现类

Quartz有以下几种Trigger实现:

SimpleTrigger

指定从某一个时间开始，以一定的时间间隔（单位是毫秒）执行的任务。

它适合的任务类似于：9:00 开始，每隔1小时，执行一次。

它的属性有：

repeatInterval 重复间隔

repeatCount 重复次数。实际执行次数是 repeatCount+1。因为在startTime的时候一定会执行一次。\*\* 下面有关repeatCount 属性的都是同理。　\*\*

例子：

simpleSchedule()

.withIntervalInHours(1) //每小时执行一次

.repeatForever() //次数不限

.build();

simpleSchedule()

.withIntervalInMinutes(1) //每分钟执行一次

.withRepeatCount(10) //次数为10次

.build();

CalendarIntervalTrigger

类似于SimpleTrigger，指定从某一个时间开始，以一定的时间间隔执行的任务。 但是不同的是SimpleTrigger指定的时间间隔为毫秒，没办法指定每隔一个月执行一次（每月的时间间隔不是固定值），而CalendarIntervalTrigger支持的间隔单位有秒，分钟，小时，天，月，年，星期。

相较于SimpleTrigger有两个优势：1、更方便，比如每隔1小时执行，你不用自己去计算1小时等于多少毫秒。 2、支持不是固定长度的间隔，比如间隔为月和年。但劣势是精度只能到秒。

它适合的任务类似于：9:00 开始执行，并且以后每周 9:00 执行一次

它的属性有:

interval 执行间隔

intervalUnit 执行间隔的单位（秒，分钟，小时，天，月，年，星期）

例子:

calendarIntervalSchedule()

.withIntervalInDays(1) //每天执行一次

.build();

calendarIntervalSchedule()

.withIntervalInWeeks(1) //每周执行一次

.build();

DailyTimeIntervalTrigger

指定每天的某个时间段内，以一定的时间间隔执行任务。并且它可以支持指定星期。

它适合的任务类似于：指定每天9:00 至 18:00 ，每隔70秒执行一次，并且只要周一至周五执行。

它的属性有:

startTimeOfDay 每天开始时间

endTimeOfDay 每天结束时间

daysOfWeek 需要执行的星期

interval 执行间隔

intervalUnit 执行间隔的单位（秒，分钟，小时，天，月，年，星期）

repeatCount 重复次数

例子:

dailyTimeIntervalSchedule()

.startingDailyAt(TimeOfDay.hourAndMinuteOfDay(9, 0)) //第天9：00开始

.endingDailyAt(TimeOfDay.hourAndMinuteOfDay(16, 0)) //16：00 结束

.onDaysOfTheWeek(MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY) //周一至周五执行

.withIntervalInHours(1) //每间隔1小时执行一次

.withRepeatCount(100) //最多重复100次（实际执行100+1次）

.build();

dailyTimeIntervalSchedule()

.startingDailyAt(TimeOfDay.hourAndMinuteOfDay(9, 0)) //第天9：00开始

.endingDailyAfterCount(10) //每天执行10次，这个方法实际上根据 startTimeOfDay+interval\*count 算出 endTimeOfDay

.onDaysOfTheWeek(MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY) //周一至周五执行

.withIntervalInHours(1) //每间隔1小时执行一次

.build();

CronTrigger

适合于更复杂的任务，它支持类型于Linux Cron的语法（并且更强大）。基本上它覆盖了以上三个Trigger的绝大部分能力（但不是全部）—— 当然，也更难理解。

它适合的任务类似于：每天0:00,9:00,18:00各执行一次。

它的属性只有:

Cron表达式。但这个表示式本身就够复杂了。下面会有说明。

例子：

cronSchedule("0 0/2 8-17 \* \* ?") // 每天8:00-17:00，每隔2分钟执行一次

.build();

cronSchedule("0 30 9 ? \* MON") // 每周一，9:30执行一次

.build();

weeklyOnDayAndHourAndMinute(MONDAY,9, 30) //等同于 0 30 9 ? \* MON

.build();

Cron表达式

位置 时间域 允许值 特殊值

1 秒 0-59 , - \* /

2 分钟 0-59 , - \* /

3 小时 0-23 , - \* /

4 日期 1-31 , - \* ? / L W C

5 月份 1-12 , - \* /

6 星期 1-7 , - \* ? / L C #

7 年份（可选） 1-31 , - \* /

星号()：可用在所有字段中，表示对应时间域的每一个时刻，例如， 在分钟字段时，表示“每分钟”；

问号（?）：该字符只在日期和星期字段中使用，它通常指定为“无意义的值”，相当于点位符；

减号(-)：表达一个范围，如在小时字段中使用“10-12”，则表示从10到12点，即10,11,12；

逗号(,)：表达一个列表值，如在星期字段中使用“MON,WED,FRI”，则表示星期一，星期三和星期五；

斜杠(/)：x/y表达一个等步长序列，x为起始值，y为增量步长值。如在分钟字段中使用0/15，则表示为0,15,30和45秒，而5/15在分钟字段中表示5,20,35,50，你也可以使用\*/y，它等同于0/y；

L：该字符只在日期和星期字段中使用，代表“Last”的意思，但它在两个字段中意思不同。L在日期字段中，表示这个月份的最后一天，如一月的31号，非闰年二月的28号；如果L用在星期中，则表示星期六，等同于7。但是，如果L出现在星期字段里，而且在前面有一个数值X，则表示“这个月的最后X天”，例如，6L表示该月的最后星期五；

W：该字符只能出现在日期字段里，是对前导日期的修饰，表示离该日期最近的工作日。例如15W表示离该月15号最近的工作日，如果该月15号是星期六，则匹配14号星期五；如果15日是星期日，则匹配16号星期一；如果15号是星期二，那结果就是15号星期二。但必须注意关联的匹配日期不能够跨月，如你指定1W，如果1号是星期六，结果匹配的是3号星期一，而非上个月最后的那天。W字符串只能指定单一日期，而不能指定日期范围；

LW组合：在日期字段可以组合使用LW，它的意思是当月的最后一个工作日；

井号(#)：该字符只能在星期字段中使用，表示当月某个工作日。如6#3表示当月的第三个星期五(6表示星期五，#3表示当前的第三个)，而4#5表示当月的第五个星期三，假设当月没有第五个星期三，忽略不触发；

C：该字符只在日期和星期字段中使用，代表“Calendar”的意思。它的意思是计划所关联的日期，如果日期没有被关联，则相当于日历中所有日期。例如5C在日期字段中就相当于日历5日以后的第一天。1C在星期字段中相当于星期日后的第一天。

Cron表达式对特殊字符的大小写不敏感，对代表星期的缩写英文大小写也不敏感。

一些例子：

表示式 说明

0 0 12 \* \* ? 每天12点运行

0 15 10 ? \* \* 每天10:15运行

0 15 10 \* \* ? 每天10:15运行

0 15 10 \* \* ? \* 每天10:15运行

0 15 10 \* \* ? 2008 在2008年的每天10：15运行

0 \* 14 \* \* ? 每天14点到15点之间每分钟运行一次，开始于14:00，结束于14:59。

0 0/5 14 \* \* ? 每天14点到15点每5分钟运行一次，开始于14:00，结束于14:55。

0 0/5 14,18 \* \* ? 每天14点到15点每5分钟运行一次，此外每天18点到19点每5钟也运行一次。

0 0-5 14 \* \* ? 每天14:00点到14:05，每分钟运行一次。

0 10,44 14 ? 3 WED 3月每周三的14:10分到14:44，每分钟运行一次。

0 15 10 ? \* MON-FRI 每周一，二，三，四，五的10:15分运行。

0 15 10 15 \* ? 每月15日10:15分运行。

0 15 10 L \* ? 每月最后一天10:15分运行。

0 15 10 ? \* 6L 每月最后一个星期五10:15分运行。

0 15 10 ? \* 6L 2007-2009 在2007,2008,2009年每个月的最后一个星期五的10:15分运行。

0 15 10 ? \* 6#3 每月第三个星期五的10:15分运行。

#### 10、JobDetail & Job

JobDetail是任务的定义，而Job是任务的执行逻辑。在JobDetail里会引用一个Job Class定义。一个最简单的例子

public class JobTest {

public static void main(String[] args) throws SchedulerException, IOException {

JobDetail job=newJob()

.ofType(DoNothingJob.class) //引用Job Class

.withIdentity("job1", "group1") //设置name/group

.withDescription("this is a test job") //设置描述

.usingJobData("age", 18) //加入属性到ageJobDataMap

.build();

job.getJobDataMap().put("name", "quertz"); //加入属性name到JobDataMap

//定义一个每秒执行一次的SimpleTrigger

Trigger trigger=newTrigger()

.startNow()

.withIdentity("trigger1")

.withSchedule(simpleSchedule()

.withIntervalInSeconds(1)

.repeatForever())

.build();

Scheduler sche=StdSchedulerFactory.getDefaultScheduler();

sche.scheduleJob(job, trigger);

sche.start();

System.in.read();

sche.shutdown();

}

}

public class DoNothingJob implements Job {

public void execute(JobExecutionContext context) throws JobExecutionException {

System.out.println("do nothing");

}

}

从上例我们可以看出，要定义一个任务，需要干几件事：

创建一个org.quartz.Job的实现类，并实现实现自己的业务逻辑。比如上面的DoNothingJob。

定义一个JobDetail，引用这个实现类

加入scheduleJob

Quartz调度一次任务，会干如下的事：

JobClass jobClass=JobDetail.getJobClass()

Job jobInstance=jobClass.newInstance()。所以Job实现类，必须有一个public的无参构建方法。

jobInstance.execute(JobExecutionContext context)。JobExecutionContext是Job运行的上下文，可以获得Trigger、Scheduler、JobDetail的信息。

也就是说，每次调度都会创建一个新的Job实例，这样的好处是有些任务并发执行的时候，不存在对临界资源的访问问题——当然，如果需要共享JobDataMap的时候，还是存在临界资源的并发访问的问题。

JobDataMap

Job都次都是newInstance的实例，那我怎么传值给它？ 比如我现在有两个发送邮件的任务，一个是发给"liLei",一个发给"hanmeimei",不能说我要写两个Job实现类LiLeiSendEmailJob和HanMeiMeiSendEmailJob。实现的办法是通过JobDataMap。

每一个JobDetail都会有一个JobDataMap。JobDataMap本质就是一个Map的扩展类，只是提供了一些更便捷的方法，比如getString()之类的。

我们可以在定义JobDetail，加入属性值，方式有二：

newJob().usingJobData("age", 18) //加入属性到ageJobDataMap

or

job.getJobDataMap().put("name", "quertz"); //加入属性name到JobDataMap

然后在Job中可以获取这个JobDataMap的值，方式同样有二：

public class HelloQuartz implements Job {

private String name;

public void execute(JobExecutionContext context) throws JobExecutionException {

JobDetail detail = context.getJobDetail();

JobDataMap map = detail.getJobDataMap(); //方法一：获得JobDataMap

System.out.println("say hello to " + name + "[" + map.getInt("age") + "]" + " at "

+ new Date());

}

//方法二：属性的setter方法，会将JobDataMap的属性自动注入

public void setName(String name) {

this.name = name;

}

}

对于同一个JobDetail实例，执行的多个Job实例，是共享同样的JobDataMap，也就是说，如果你在任务里修改了里面的值，会对其他Job实例（并发的或者后续的）造成影响。

除了JobDetail，Trigger同样有一个JobDataMap，共享范围是所有使用这个Trigger的Job实例。

Job并发

Job是有可能并发执行的，比如一个任务要执行10秒中，而调度算法是每秒中触发1次，那么就有可能多个任务被并发执行。

有时候我们并不想任务并发执行，比如这个任务要去”获得数据库中所有未发送邮件的名单“，如果是并发执行，就需要一个数据库锁去避免一个数据被多次处理。这个时候一个@DisallowConcurrentExecution解决这个问题。

就是这样

public class DoNothingJob implements Job {

@DisallowConcurrentExecution

public void execute(JobExecutionContext context) throws JobExecutionException {

System.out.println("do nothing");

}

}

注意，@DisallowConcurrentExecution是对JobDetail实例生效，也就是如果你定义两个JobDetail，引用同一个Job类，是可以并发执行的。

JobExecutionException

Job.execute()方法是不允许抛出除JobExecutionException之外的所有异常的（包括RuntimeException)，所以编码的时候，最好是try-catch住所有的Throwable，小心处理。

其他属性

Durability(耐久性？)

如果一个任务不是durable，那么当没有Trigger关联它的时候，它就会被自动删除。

RequestsRecovery

如果一个任务是"requests recovery"，那么当任务运行过程非正常退出时（比如进程崩溃，机器断电，但不包括抛出异常这种情况），Quartz再次启动时，会重新运行一次这个任务实例。

可以通过JobExecutionContext.isRecovering()查询任务是否是被恢复的。

Scheduler

Scheduler就是Quartz的大脑，所有任务都是由它来设施。

Schduelr包含一个两个重要组件: JobStore和ThreadPool。

JobStore是会来存储运行时信息的，包括Trigger,Schduler,JobDetail，业务锁等。它有多种实现RAMJob(内存实现)，JobStoreTX(JDBC，事务由Quartz管理），JobStoreCMT(JDBC，使用容器事务)，ClusteredJobStore(集群实现)、TerracottaJobStore(什么是Terractta)。

ThreadPool就是线程池，Quartz有自己的线程池实现。所有任务的都会由线程池执行。

SchedulerFactory

SchdulerFactory，顾名思义就是来用创建Schduler了，有两个实现：DirectSchedulerFactory和 StdSchdulerFactory。前者可以用来在代码里定制你自己的Schduler参数。后者是直接读取classpath下的quartz.properties（不存在就都使用默认值）配置来实例化Schduler。通常来讲，我们使用StdSchdulerFactory也就足够了。

SchdulerFactory本身是支持创建RMI stub的，可以用来管理远程的Scheduler，功能与本地一样，可以远程提交个Job什么的。

DirectSchedulerFactory的创建接口

/\*\*

\* Same as

\* {@link DirectSchedulerFactory#createScheduler(ThreadPool threadPool, JobStore jobStore)},

\* with the addition of specifying the scheduler name and instance ID. This

\* scheduler can only be retrieved via

\* {@link DirectSchedulerFactory#getScheduler(String)}

\*

\* @param schedulerName

\* The name for the scheduler.

\* @param schedulerInstanceId

\* The instance ID for the scheduler.

\* @param threadPool

\* The thread pool for executing jobs

\* @param jobStore

\* The type of job store

\* @throws SchedulerException

\* if initialization failed

\*/

public void createScheduler(String schedulerName,

String schedulerInstanceId, ThreadPool threadPool, JobStore jobStore)

throws SchedulerException;

StdSchdulerFactory的配置例子， 更多配置，参考Quartz配置指南：

org.quartz.scheduler.instanceName = DefaultQuartzScheduler

org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool

org.quartz.threadPool.threadCount = 10

org.quartz.threadPool.threadPriority = 5

org.quartz.threadPool.threadsInheritContextClassLoaderOfInitializingThread = true

org.quartz.jobStore.class = org.quartz.simpl.RAMJobStor

## 五 新技术阶段

### 5.1 Redis

#### 1、什么是redis?

Redis 是一个基于内存的高性能key-value数据库。

#### 2、Redis支持的数据类型

Redis通过Key-Value的单值不同类型来区分,

以下是支持的类型:Strings

Lists

Sets 求交集、并集

Sorted Set

hashes

#### 3、为什么redis需要把所有数据放到内存中？

Redis为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以redis具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘I/O速度为严重影响redis的性能。在内存越来越便宜的今天，redis将会越来越受欢迎。

如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

#### 4、Reids的特点

　　Redis本质上是一个Key-Value类型的内存数据库，很像memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据flush到硬盘上进行保存。因为是纯内存操作，Redis的性能非常出色，每秒可以处理超过 10万次读写操作，是已知性能最快的Key-Value DB。

　　Redis的出色之处不仅仅是性能，Redis最大的魅力是支持保存多种数据结构，此外单个value的最大限制是1GB，不像 memcached只能保存1MB的数据，因此Redis可以用来实现很多有用的功能，比方说用他的List来做FIFO双向链表，实现一个轻量级的高性 能消息队列服务，用他的Set可以做高性能的tag系统等等。另外Redis也可以对存入的Key-Value设置expire时间，因此也可以被当作一个功能加强版的memcached来用。

　　Redis的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此Redis适合的场景主要局限在较小数据量的高性能操作和运算上。

#### 5、.使用redis有哪些好处？

   (1) 速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)   
   (2) 支持丰富数据类型，支持string，list，set，sorted set，hash   
   (3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行   
   (4) 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

#### 6、redis相比memcached有哪些优势？

(1) memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型

(2) redis的速度比memcached快很多

(3) redis可以持久化其数据

#### 7、Memcache与Redis的区别都有哪些？

1)、存储方式 Memecache把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。 Redis有部份存在硬盘上，这样能保证数据的持久性。   
2)、数据支持类型 Memcache对数据类型支持相对简单。 Redis有复杂的数据类型。   
3)、使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。 Redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

#### 8、redis常见性能问题和解决方案

1).Master写内存快照，save命令调度rdbSave函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以Master最好不要写内存快照。

2).Master AOF持久化，如果不重写AOF文件，这个持久化方式对性能的影响是最小的，但是AOF文件会不断增大，AOF文件过大会影响Master重启的恢复速度。Master最好不要做任何持久化工作，包括内存快照和AOF日志文件，特别是不要启用内存快照做持久化,如果数据比较关键，某个Slave开启AOF备份数据，策略为每秒同步一次。

3).Master调用BGREWRITEAOF重写AOF文件，AOF在重写的时候会占大量的CPU和内存资源，导致服务load过高，出现短暂服务暂停现象。

4). Redis主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave和Master最好在同一个局域网内

#### 9、redis持久化的几种方式

1、快照（snapshots）

缺省情况情况下，Redis把数据快照存放在磁盘上的二进制文件中，文件名为dump.rdb。你可以配置Redis的持久化策略，例如数据集中每N秒钟有超过M次更新，就将数据写入磁盘；或者你可以手工调用命令SAVE或BGSAVE。

　　工作原理

　　． Redis forks.

　　． 子进程开始将数据写到临时RDB文件中。

　　． 当子进程完成写RDB文件，用新文件替换老文件。

　　． 这种方式可以使Redis使用copy-on-write技术。

2、AOF

　　快照模式并不十分健壮，当系统停止，或者无意中Redis被kill掉，最后写入Redis的数据就会丢失。这对某些应用也许不是大问题，但对于要求高可靠性的应用来说，

　　Redis就不是一个合适的选择。

　　Append-only文件模式是另一种选择。

　　你可以在配置文件中打开AOF模式

3、虚拟内存方式

　　当你的key很小而value很大时,使用VM的效果会比较好.因为这样节约的内存比较大.

　　当你的key不小时,可以考虑使用一些非常方法将很大的key变成很大的value,比如你可以考虑将key,value组合成一个新的value.

　　vm-max-threads这个参数,可以设置访问swap文件的线程数,设置最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的.可能会造成比较长时间的延迟,但是对数据完整性有很好的保证.

　　自己测试的时候发现用虚拟内存性能也不错。如果数据量很大，可以考虑分布式或者其他数据库

### 5.2 Nginx

#### ****1、什么是Nginx？****

Nginx是一个高性能的HTTP和反向代理服务器，及电子邮件（IMAP/POP3）代理服务器，同时也是一个非常高效的反向代理、负载平衡。

#### 2、为什么要用Nginx？

优点：

跨平台、配置简单

非阻塞、高并发连接：处理2-3万并发连接数，官方监测能支持5万并发

内存消耗小：开启10个nginx才占150M内存，Nginx采取了分阶段[资源分配](https://baike.so.com/doc/1092243-1155769.html)技术

Nginx处理静态文件好,耗费内存少

内置的健康检查功能：如果有一个服务器宕机，会做一个健康检查，再发送的请求就不会发送到宕机的服务器了。重新将请求提交到其他的节点上。

节省宽带：支持GZIP压缩，可以添加浏览器本地缓存

稳定性高：宕机的概率非常小

 master/worker结构：一个master进程，生成一个或者多个worker进程

接收用户请求是异步的：浏览器将请求发送到nginx服务器，它先将用户请求全部接收下来，再一次性发送给后端web服务器，极大减轻了web服务器的压力

 一边接收web服务器的返回数据，一边发送给浏览器客户端

 网络依赖性比较低，只要ping通就可以负载均衡

 可以有多台nginx服务器

 事件驱动：通信机制采用epoll模型

#### 3、为什么Nginx性能这么高？

 得益于它的事件处理机制：

 异步非阻塞事件处理机制：运用了epoll模型，提供了一个队列，排队解决

#### 4、为什么不使用多线程？

Apache: 创建多个进程或线程，而每个进程或线程都会为其分配cpu和内存（线程要比进程小的多，所以worker支持比perfork高的并发），并发过大会榨干服务器资源。

Nginx: 采用单线程来异步非阻塞处理请求（管理员可以配置Nginx主进程的工作进程的数量）(epoll)，不会为每个请求分配cpu和内存资源，节省了大量资源，同时也减少了大量的CPU的上下文切换。所以才使得Nginx支持更高的并发。

#### 5、Nginx是如何处理一个请求的呢？

首先，nginx在启动时，会解析配置文件，得到需要监听的端口与ip地址，然后在nginx的master进程里面

先初始化好这个监控的socket，再进行listen

然后再fork出多个子进程出来,  子进程会竞争accept新的连接。

此时，客户端就可以向nginx发起连接了。当客户端与nginx进行三次握手，与nginx建立好一个连接后

此时，某一个子进程会accept成功，然后创建nginx对连接的封装，即ngx\_connection\_t结构体

接着，根据事件调用相应的事件处理模块，如http模块与客户端进行数据的交换。

最后，nginx或客户端来主动关掉连接，到此，一个连接就寿终正寝了

#### 6、正向代理

 一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。客户端才能使用正向代理

正向代理总结就一句话：代理端代理的是客户端

#### 7、反向代理

反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后将请求，发给内部网络上的服务器

并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器

反向代理总结就一句话：代理端代理的是服务端

#### 8、动态资源，静态资源分离

 动态资源、静态资源分离是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们就可以根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路

动态资源、静态资源分离简单的概括是：动态文件与静态文件的分离

#### 9、为什么要做动、静分离？

 在我们的软件开发中，有些请求是需要后台处理的（如：.jsp,.do等等），有些请求是不需要经过后台处理的（如：css、html、jpg、js等等文件）

 这些不需要经过后台处理的文件称为静态文件，否则动态文件。因此我们后台处理忽略静态文件。这会有人又说那我后台忽略静态文件不就完了吗， 当然这是可以的，但是这样后台的请求次数就明显增多了。在我们对资源的响应速度有要求的时候，我们应该使用这种动静分离的策略去解决动、静分离将网站静态资源（HTML，JavaScript，CSS，img等文件）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问

 这里我们将静态资源放到nginx中，动态资源转发到tomcat服务器中

#### 10、负载均衡

负载均衡即是代理服务器将接收的请求均衡的分发到各服务器中

负载均衡主要解决网络拥塞问题，提高服务器响应速度，服务就近提供，达到更好的访问质量，减少后台服务器大并发压力。

### 5.3 Spring Boot

#### 1、什么是SpringBoot

描述：Spring Boot是Spring社区发布的一个开源项目，旨在帮助开发者快速并且更简单的构建项目。大多数SpringBoot项目只需要很少的配置文件。

#### 2、SpringBoot核心功能

1、独立运行Spring项目

Spring boot 可以以jar包形式独立运行，运行一个Spring Boot项目只需要通过java -jar xx.jar来运行。

2、内嵌servlet容器

Spring Boot可以选择内嵌Tomcat、jetty或者Undertow,这样我们无须以war包形式部署项目。

3、提供starter简化Maven配置

spring提供了一系列的start pom来简化Maven的依赖加载，例如，当你使用了spring-boot-starter-web，会自动加入如图5-1所示的依赖包。

4、自动装配Spring

SpringBoot会根据在类路径中的jar包，类、为jar包里面的类自动配置Bean，这样会极大地减少我们要使用的配置。当然，SpringBoot只考虑大多数的开发场景，并不是所有的场景，若在实际开发中我们需要配置Bean，而SpringBoot灭有提供支持，则可以自定义自动配置。

5、准生产的应用监控

SpringBoot提供基于http ssh telnet对运行时的项目进行监控。

6、无代码生产和xml配置

SpringBoot不是借助与代码生成来实现的，而是通过条件注解来实现的，这是Spring4.x提供的新特性。

#### 3、SpringBoot优点

1、快速构建项目。

2、对主流开发框架的无配置集成。

3、项目可独立运行，无须外部依赖Servlet容器。

4、提供运行时的应用监控。

5、极大的提高了开发、部署效率。

6、与云计算的天然集成。

#### 4、SpringBoot特性

1、创建独立的Spring项目

2、内置Tomcat和Jetty容器

3、提供一个starter POMs来简化Maven配置

4、提供了一系列大型项目中常见的非功能性特性，如安全、指标，健康检测、外部配置等

5、完全没有代码生成和xml配置文件

#### 6、SpringBoot CLI

SpringBoot CLI 是SpringBoot提供的控制台命令工具。

#### 7、SpringBoot maven 构建项目

spring-boot-starter-parent：是一个特殊Start，它用来提供相关的Maven依赖项，使用它之后，常用的包依赖可以省去version标签。

#### 8、SpringBoot几个常用的注解

（1）@RestController和@Controller指定一个类，作为控制器的注解

（2）@RequestMapping方法级别的映射注解，这一个用过Spring MVC的小伙伴相信都很熟悉

（3）@EnableAutoConfiguration和@SpringBootApplication是类级别的注解，根据maven依赖的jar来自动猜测完成正确的spring的对应配置，只要引入了spring-boot-starter-web的依赖，默认会自动配置Spring MVC和tomcat容器

（4）@Configuration类级别的注解，一般这个注解，我们用来标识main方法所在的类,完成元数据bean的初始化。

（5）@ComponentScan类级别的注解，自动扫描加载所有的Spring组件包括Bean注入，一般用在main方法所在的类上

（6）@ImportResource类级别注解，当我们必须使用一个xml的配置时，使用@ImportResource和@Configuration来标识这个文件资源的类。

（7）@Autowired注解，一般结合@ComponentScan注解，来自动注入一个Service或Dao级别的Bean

（8）@Component类级别注解，用来标识一个组件，比如我自定了一个filter，则需要此注解标识之后，Spring Boot才会正确识别。

### 5.4 Spring Cloud

#### 1、谈谈你对SpringCloud的认识

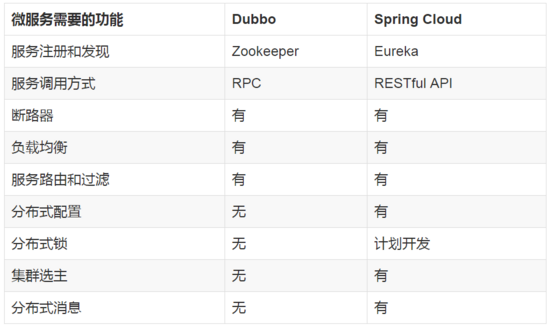
SpringCloud 是Pivotal提供的用于简化分布式系统构建的工具集。Spring Cloud引入了[云平台](http://www.chinacloud.cn/show.aspx?id=2608&cid=14)连接器(CloudConnector)和服务连接器(Service Connector)的概念。云平台连接器是一个接口，需要由云平台提供者进行实现，以便库中的其他模块可以与该云平台协同工作。

Spring Cloud最重要的一点是它可以和Spring Boot一起工作，Spring Boot可以帮助开发者更容易地创建基于Spring的应用程序和服务。

#### 2、Spring Cloud与Dubbo的对比

作为新一代的服务框架，Spring Cloud提出的口号是开发“面向云环境的应用程序”，它为微服务架构提供了更加全面的技术支持。

结合我们一开始提到的微服务的诉求，我们把Spring Cloud与Dubbo进行一番对比：



Spring Cloud抛弃了Dubbo的RPC通信，采用的是基于HTTP的REST方式。严格来说，这两种方式各有优劣。虽然从一定程度上来说，后者牺牲了服务调用的性能，但也避免了上面提到的原生RPC带来的问题。而且REST相比RPC更为灵活，服务提供方和调用方的依赖只依靠一纸契约，不存在代码级别的强依赖，这在强调快速演化的微服务环境下，显得更加合适。

很明显，Spring Cloud的功能比DUBBO更加强大，涵盖面更广，而且作为Spring的拳头项目，它也能够与Spring Framework、Spring Boot、Spring Data、Spring Batch等其他Spring项目完美融合，这些对于微服务而言是至关重要的。前面提到，微服务背后一个重要的理念就是持续集成、快速交付，而在服务内部使用一个统一的技术框架，显然比把分散的技术组合到一起更有效率。更重要的是，相比于Dubbo，它是一个正在持续维护的、社区更加火热的开源项目，这就保证使用它构建的系统，可以持续地得到开源力量的支持。

#### 3、Spring Cloud Netflix 组件

Netflix和Spring Cloud是什么关系呢？Netflix是一家成功实践微服务架构的互联网公司，几年前，Netflix就把它的几乎整个微服务框架栈开源贡献给了社区。Spring背后的Pivotal在2015年推出的Spring Cloud开源产品，主要对Netflix开源组件的进一步封装，方便Spring开发人员构建微服务基础框架。

对于微服务的治理而言，核心就是服务的注册和发现。所以选择哪个组件，很大程度上要看它对于服务注册与发现的解决方案。在这个领域，开源架构很多，最常见的是Zookeeper，但这并不是一个最佳选择。

在分布式系统领域有个著名的CAP定理：C——数据一致性，A——服务可用性，P——服务对网络分区故障的容错性。这三个特性在任何分布式系统中不能同时满足，最多同时满足两个。

Eureka本身是Netflix开源的一款提供服务注册和发现的产品，并且提供了相应的Java封装。在它的实现中，节点之间是相互平等的，部分注册中心的节点挂掉也不会对集群造成影响，即使集群只剩一个节点存活，也可以正常提供发现服务。哪怕是所有的服务注册节点都挂了，Eureka Clients上也会缓存服务调用的信息。这就保证了我们微服务之间的互相调用是足够健壮的。

除此之外，Spring Cloud Netflix背后强大的开源力量，也促使我们选择了Spring Cloud Netflix：

前文提到过，Spring Cloud的社区十分活跃，其在业界的应用也十分广泛（尤其是国外），而且整个框架也经受住了Netflix严酷生产环境的考验。

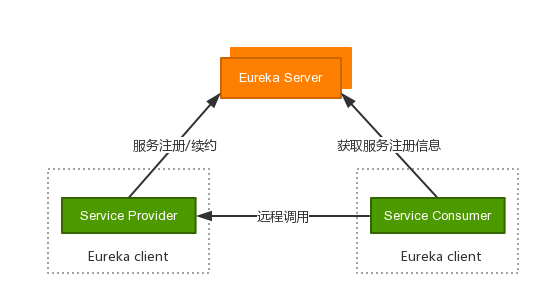
除了服务注册和发现，Spring Cloud Netflix的其他功能也十分强大，包括Ribbon，hystrix，Feign，Zuul等组件，结合到一起，让服务的调用、路由也变得异常容易。

Spring Cloud Netflix作为Spring的重量级整合框架，使用它也意味着我们能从Spring获取到巨大的便利。Spring Cloud的其他子项目，比如Spring Cloud Stream、Spring Cloud Config等等，都为微服务的各种需求提供了一站式的解决方案。

#### 4、服务注册与发现——Eureka

Eureka这个词来源于古希腊语，意为“我找到了！我发现了！”，据传，阿基米德在洗澡时发现浮力原理，高兴得来不及穿上裤子，跑到街上大喊：“Eureka(我找到了)！”。

Eureka由多个instance(服务实例)组成，这些服务实例可以分为两种：Eureka Server和Eureka Client。为了便于理解，我们将Eureka client再分为Service Provider和Service Consumer。如下图所示：



Eureka Server：服务的注册中心，负责维护注册的服务列表。

**Service Provider：**服务提供方，作为一个Eureka Client，向Eureka Server做服务注册、续约和下线等操作，注册的主要数据包括服务名、机器ip、端口号、域名等等。

**Service Consumer：**服务消费方，作为一个Eureka Client，向Eureka Server获取Service Provider的注册信息，并通过远程调用与Service Provider进行通信。

Service Provider和Service Consumer不是严格的概念，Service Consumer也可以随时向Eureka Server注册，来让自己变成一个Service Provider。

Spring Cloud针对服务注册与发现，进行了一层抽象，并提供了三种实现：Eureka、Consul、Zookeeper。目前支持得最好的就是Eureka，其次是Consul，最后是Zookeeper。

**Eureka Server**作为一个独立的部署单元，以REST API的形式为服务实例提供了注册、管理和查询等操作。同时，Eureka Server也为我们提供了可视化的监控页面，可以直观地看到各个Eureka Server当前的运行状态和所有已注册服务的情况。

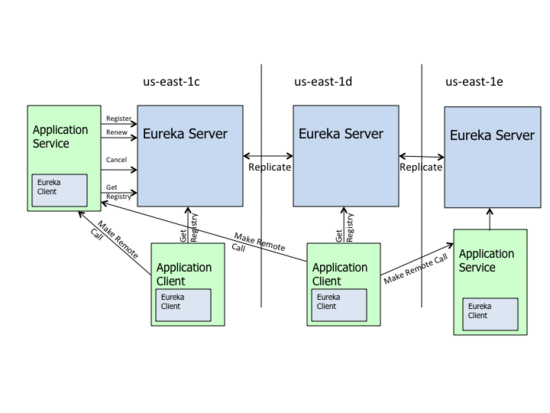
**Eureka Server的高可用集群**

Eureka Server可以运行多个实例来构建集群，解决单点问题，但不同于ZooKeeper的选举leader的过程，Eureka Server采用的是Peer to Peer对等通信。这是一种去中心化的架构，无master/slave区分，每一个Peer都是对等的。在这种架构中，节点通过彼此互相注册来提高可用性，每个节点需要添加一个或多个有效的serviceUrl指向其他节点。每个节点都可被视为其他节点的副本。

如果某台Eureka Server宕机，Eureka Client的请求会自动切换到新的Eureka Server节点，当宕机的服务器重新恢复后，Eureka会再次将其纳入到服务器集群管理之中。当节点开始接受客户端请求时，所有的操作都会进行replicateToPeer（节点间复制）操作，将请求复制到其他Eureka Server当前所知的所有节点中。

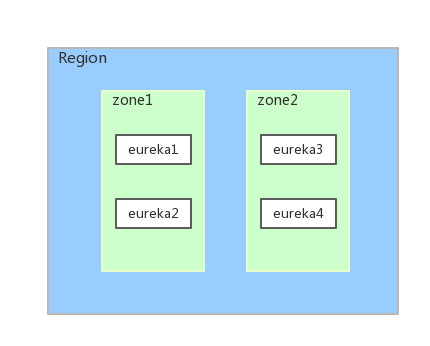
一个新的Eureka Server节点启动后，会首先尝试从邻近节点获取所有实例注册表信息，完成初始化。Eureka Server通过getEurekaServiceUrls()方法获取所有的节点，并且会通过心跳续约的方式定期更新。默认配置下，如果Eureka Server在一定时间内没有接收到某个服务实例的心跳，Eureka Server将会注销该实例（默认为90秒，通过eureka.instance.lease-expiration-duration-in-seconds配置）。当Eureka Server节点在短时间内丢失过多的心跳时（比如发生了网络分区故障），那么这个节点就会进入自我保护模式。下图为Eureka官网的架构图

什么是自我保护模式？默认配置下，如果Eureka Server每分钟收到心跳续约的数量低于一个阈值（instance的数量\*(60/每个instance的心跳间隔秒数)\*自我保护系数），就会触发自我保护。在自我保护模式中，Eureka Server会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该Eureka Server节点就会自动退出自我保护模式。它的设计哲学前面提到过，那就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。该模式可以通过eureka.server.enable-self-preservation = false来禁用，同时eureka.instance.lease-renewal-interval-in-seconds可以用来更改心跳间隔，eureka.server.renewal-percent-threshold可以用来修改自我保护系数（默认0.85）。



Eureka Server的Region、Zone

Eureka的官方文档对Regin、Zone几乎没有提及，由于概念抽象，新手很难理解。因此，我们先来了解一下Region、Zone、Eureka集群三者的关系，如下图所示：



region和zone（或者Availability Zone）均是AWS的概念。在非AWS环境下，我们可以先简单地将region理解为Eureka集群，zone理解成机房。上图就可以理解为一个Eureka集群被部署在了zone1机房和zone2机房中。

**Service Provider**

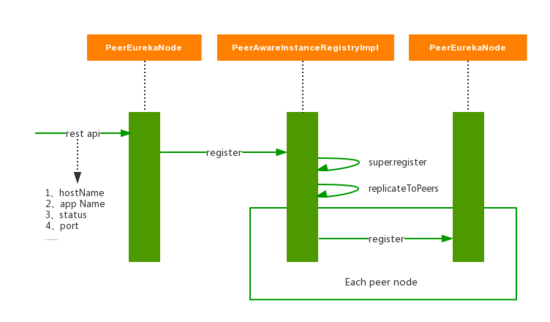
**服务注册**

Service Provider本质上是一个Eureka Client。它启动时，会调用服务注册方法，向Eureka Server注册自己的信息。Eureka Server会维护一个已注册服务的列表，这个列表为一个嵌套的hash map：

第一层，application name和对应的服务实例。

第二层，服务实例及其对应的注册信息，包括IP，端口号等。

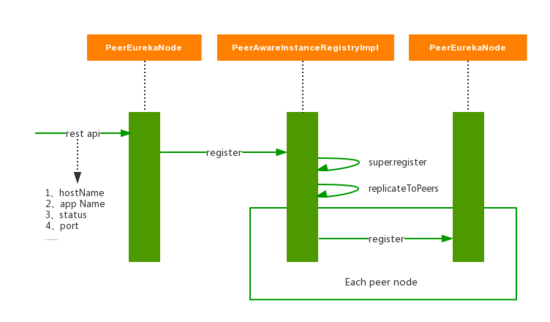
当实例状态发生变化时（如自身检测认为Down的时候），也会向Eureka Server更新自己的服务状态，同时用replicateToPeers()向其它Eureka Server节点做状态同步。



**续约与剔除**

前面提到过，服务实例启动后，会周期性地向Eureka Server发送心跳以续约自己的信息，避免自己的注册信息被剔除。续约的方式与服务注册基本一致：首先更新自身状态，再同步到其它Peer。

如果Eureka Server在一段时间内没有接收到某个微服务节点的心跳，Eureka Server将会注销该微服务节点（自我保护模式除外）。



#### 6、Service Consumer

Service Consumer本质上也是一个Eureka Client（它也会向Eureka Server注册，只是这个注册信息无关紧要罢了）。它启动后，会从Eureka Server上获取所有实例的注册信息，包括IP地址、端口等，并缓存到本地。这些信息默认每30秒更新一次。前文提到过，如果与Eureka Server通信中断，Service Consumer仍然可以通过本地缓存与Service Provider通信。

实际开发Eureka的过程中，有时会遇见Service Consumer获取到Server Provider的信息有延迟，在Eureka Wiki中有这么一段话:

All operations from Eureka client may take some time to reflect in the Eureka servers and subsequently in other Eureka clients. This is because of the caching of the payload on the eureka server which is refreshed periodically to reflect new information. Eureka clients also fetch deltas periodically. Hence, it may take up to 2 mins for changes to propagate to all Eureka clients.

最后一句话提到，服务端的更改可能需要2分钟才能传播到所有客户端，至于原因并没有介绍。这是因为Eureka有三处缓存和一处延迟造成的。

Eureka Server对注册列表进行缓存，默认时间为30s。

Eureka Client对获取到的注册信息进行缓存，默认时间为30s。

Ribbon会从上面提到的Eureka Client获取服务列表，将负载均衡后的结果缓存30s。

如果不是在Spring Cloud环境下使用这些组件(Eureka, Ribbon)，服务启动后并不会马上向Eureka注册，而是需要等到第一次发送心跳请求时才会注册。心跳请求的发送间隔默认是30s。Spring Cloud对此做了修改，服务启动后会马上注册。

基于Service Consumer获取到的服务实例信息，我们就可以进行服务调用了。而Spring Cloud也为Service Consumer提供了丰富的服务调用工具：

Ribbon，实现客户端的负载均衡。

Hystrix，断路器。

Feign，RESTful Web Service客户端，整合了Ribbon和Hystrix。

#### 7、服务调用端负载均衡——Ribbon

Ribbon是Netflix发布的开源项目，主要功能是为REST客户端实现负载均衡。它主要包括六个组件：

ServerList，负载均衡使用的服务器列表。这个列表会缓存在负载均衡器中，并定期更新。当Ribbon与Eureka结合使用时，ServerList的实现类就是DiscoveryEnabledNIWSServerList，它会保存Eureka Server中注册的服务实例表。

ServerListFilter，服务器列表过滤器。这是一个接口，主要用于对Service Consumer获取到的服务器列表进行预过滤，过滤的结果也是ServerList。Ribbon提供了多种过滤器的实现。

IPing，探测服务实例是否存活的策略。

IRule，负载均衡策略，其实现类表述的策略包括：轮询、随机、根据响应时间加权等。

我们也可以自己定义负载均衡策略，比如我们就利用自己实现的策略，实现了服务的版本控制和直连配置。实现好之后，将实现类重新注入到Ribbon中即可。

ILoadBalancer，负载均衡器。这也是一个接口，Ribbon为其提供了多个实现，比如ZoneAwareLoadBalancer。而上层代码通过调用其API进行服务调用的负载均衡选择。一般ILoadBalancer的实现类中会引用一个IRule。

RestClient，服务调用器。顾名思义，这就是负载均衡后，Ribbon向Service Provider发起REST请求的工具。

Ribbon工作时会做四件事情：

1.优先选择在同一个Zone且负载较少的Eureka Server；

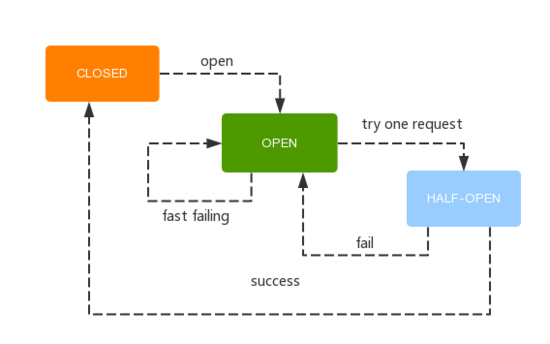
2.定期从Eureka更新并过滤服务实例列表；

3.根据用户指定的策略，在从Server取到的服务注册列表中选择一个实例的地址；

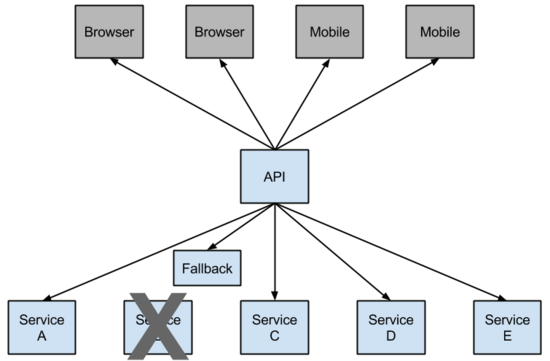
4.通过RestClient进行服务调用。

#### 8、服务调用端熔断——Hystrix

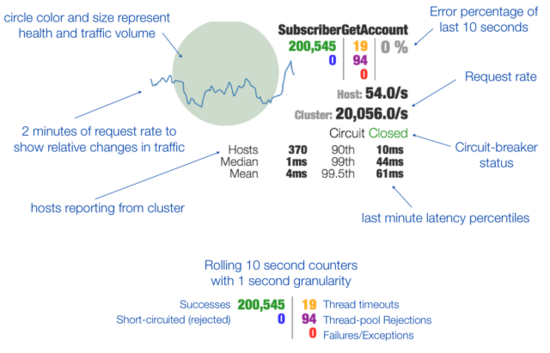
Netflix创建了一个名为Hystrix的库,实现了断路器的模式。“断路器”本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。



当然，在请求失败频率较低的情况下，Hystrix还是会直接把故障返回给客户端。只有当失败次数达到阈值（默认在20秒内失败5次）时，断路器打开并且不进行后续通信，而是直接返回备选响应。当然，Hystrix的备选响应也是可以由开发者定制的。



除了隔离依赖服务的调用以外，Hystrix还提供了准实时的调用监控（Hystrix Dashboard），Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，并以统计报表和图形的形式展示给用户，包括每秒执行多少请求多少成功，多少失败等。Netflix通过hystrix-metrics-event-stream项目实现了对以上指标的监控。Spring Cloud也提供了Hystrix Dashboard的整合，对监控内容转化成可视化界面，Hystrix Dashboard Wiki上详细说明了图上每个指标的含义。



#### 9、服务调用端代码抽象和封装——Feign

Feign是一个声明式的Web Service客户端，它的目的就是让Web Service调用更加简单。它整合了Ribbon和Hystrix，从而让我们不再需要显式地使用这两个组件。Feign还提供了HTTP请求的模板，通过编写简单的接口和插入注解，我们就可以定义好HTTP请求的参数、格式、地址等信息。接下来，Feign会完全代理HTTP的请求，我们只需要像调用方法一样调用它就可以完成服务请求。

Feign具有如下特性：

可插拔的注解支持，包括Feign注解和JAX-RS注解

支持可插拔的HTTP编码器和解码器

支持Hystrix和它的Fallback

支持Ribbon的负载均衡

支持HTTP请求和响应的压缩

以下是一个Feign的简单示例：

@SpringBootApplication

@EnableDiscoveryClient //启用Feign

@EnableFeignClients

public class Application{

public static void main(String[] args){

SpringApplication.run(Application.class, args);

}

}

@FeignClient(name = “elements”, fallback = ElementsFallback.class) //指定feign调用的服务和Hystrix Fallback（name即eureka的application name）

public interface Elements{

@RequestMapping(value = “/index”)

String index();

}

//Hystrix Fallback

@Component

public class ElementsFallback implements Elements{

@Override

public String index(){

return “\*\*\*\*\*\*\*\*\*\*\*\*\*\*”;

}

}

//测试类

@Component

public class TestController {

@Autowired

Elements elements;

@RequestMapping(value = “/testEureka”, method = RequestMethod.GET)

public String testeureka(){

return elements.index();

}

}

### 5.5 Dubbo

#### 1、Dubbo中zookeeper做注册中心，如果注册中心集群都挂掉，发布者和订阅者之间还能通信么？

可以的，启动dubbo时，消费者会从zk拉取注册的生产者的地址接口等数据，缓存在本地。每次调用时，按照本地存储的地址进行调用

注册中心对等集群，任意一台宕掉后，会自动切换到另一台

注册中心全部宕掉，服务提供者和消费者仍可以通过本地缓存通讯

服务提供者无状态，任一台 宕机后，不影响使用

服务提供者全部宕机，服务消费者会无法使用，并无限次重连等待服务者恢复

#### 2、dubbo连接注册中心和直连的区别

在开发及测试环境下，经常需要绕过注册中心，只测试指定服务提供者，这时候可能需要点对点直连，

点对点直联方式，将以服务接口为单位，忽略注册中心的提供者列表，服务注册中心，动态的注册和发现服务，使服务的位置透明，并通过在消费方获取服务提供方地址列表，实现软负载均衡和Failover， 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，服务消费者向注册中心获取服务提供者地址列表，并根据负载算法直接调用提供者，注册中心，服务提供者，服务消费者三者之间均为长连接，监控中心除外，注册中心通过长连接感知服务提供者的存在，服务提供者宕机，注册中心将立即推送事件通知消费者注册中心和监控中心全部宕机，不影响已运行的提供者和消费者，消费者在本地缓存了提供者列表注册中心和监控中心都是可选的，服务消费者可以直连服务提供者

#### 3、Dubbo在安全机制方面是如何解决的

Dubbo通过Token令牌防止用户绕过注册中心直连，然后在注册中心上管理授权。Dubbo还提供服务黑白名单，来控制服务所允许的调用方。

### 5.6 Zookeeper

#### 1、Zookeeper是什么框架

分布式的、开源的分布式应用程序协调服务，原本是Hadoop、HBase的一个重要组件。它为分布式应用提供一致性服务的软件，包括：配置维护、域名服务、分布式同步、组服务等。

#### 2、Zookeeper应用场景

Zookeeper的功能很强大，应用场景很多，结合我实际工作中使用Dubbo框架的情况，Zookeeper主要是做注册中心用。基于Dubbo框架开发的提供者、消费者都向Zookeeper注册自己的URL，消费者还能拿到并订阅提供者的注册URL，以便在后续程序的执行中去调用提供者。而提供者发生了变动，也会通过Zookeeper向订阅的消费者发送通知。

#### 3、Zookeeper有哪几种节点类型

持久：创建之后一直存在，除非有删除操作，创建节点的客户端会话失效也不影响此节点。

持久顺序：跟持久一样，就是父节点在创建下一级子节点的时候，记录每个子节点创建的先后顺序，会给每个子节点名加上一个数字后缀。

临时：创建客户端会话失效（注意是会话失效，不是连接断了），节点也就没了。不能建子节点。

临时顺序：不用解释了吧。

#### 4、Zookeeper对节点的watch监听通知是永久的吗？

不是。官方声明：一个Watch事件是一个一次性的触发器，当被设置了Watch的数据发生了改变的时候，则服务器将这个改变发送给设置了Watch的客户端，以便通知它们。

为什么不是永久的，举个例子，如果服务端变动频繁，而监听的客户端很多情况下，每次变动都要通知到所有的客户端，这太消耗性能了。

一般是客户端执行getData(“/节点A”,true)，如果节点A发生了变更或删除，客户端会得到它的watch事件，但是在之后节点A又发生了变更，而客户端又没有设置watch事件，就不再给客户端发送。

在实际应用中，很多情况下，我们的客户端不需要知道服务端的每一次变动，我只要最新的数据即可。

#### 5、部署方式？集群中的机器角色都有哪些？集群最少要几台机器

单机，集群。Leader、Follower。集群最低3（2N+1）台，保证奇数，主要是为了选举算法。

#### 6、集群如果有3台机器，挂掉一台集群还能工作吗？挂掉两台呢？

记住一个原则：过半存活即可用。

#### 7、集群支持动态添加机器吗？

其实就是水平扩容了，Zookeeper在这方面不太好。两种方式：  
全部重启：关闭所有Zookeeper服务，修改配置之后启动。不影响之前客户端的会话。

逐个重启：顾名思义。这是比较常用的方式。

#### 8、Zookeeper选举过程

当leader崩溃或者leader失去大多数的follower，这时候zk进入恢复模式，恢复模式需要重新选举出一个新的leader，让所有的Server都恢复到一个正确的状态。Zk的选举算法使用ZAB协议：

1、选举线程由当前Server发起选举的线程担任，其主要功能是对投票结果进行统计，并选出推荐的Server；

2、选举线程首先向所有Server发起一次询问(包括自己)；

3、选举线程收到回复后，验证是否是自己发起的询问(验证zxid是否一致)，然后获取对方的id(myid)，并存储到当前询问对象列表中，最后获取对方提议的leader相关信息(id,zxid)，并将这些信息存储到当次选举的投票记录表中；

4、收到所有Server回复以后，就计算出zxid最大的那个Server，并将这个Server相关信息设置成下一次要投票的Server；

5、线程将当前zxid最大的Server设置为当前Server要推荐的Leader，如果此时获胜的Server获得n/2 + 1的Server票数， 设置当前推荐的leader为获胜的Server，将根据获胜的Server相关信息设置自己的状态，否则，继续这个过程，直到leader被选举出来。

通过流程分析我们可以得出：要使Leader获得多数Server的支持，则Server总数最好是奇数2n+1，且存活的Server的数目不得少于n+1

#### 9、master/slave之间通信

Storm：定期扫描   
PtBalancer：节点监听

#### 10、节点变多时，PtBalancer速度变慢

类似问题：根据Netflix的Curator作者所说，ZooKeeper真心不适合做Queue，或者说ZK没有实现一个好的Queue，   
原因有五：

1、ZK有1MB 的传输限制。 实践中ZNode必须相对较小，而队列包含成千上万的消息，非常的大。

2、如果有很多节点，ZK启动时相当的慢。 而使用queue会导致好多ZNode. 你需要显著增大 initLimit 和 syncLimit.

3、ZNode很大的时候很难清理。Netflix不得不创建了一个专门的程序做这事。

4、当很大量的包含成千上万的子节点的ZNode时， ZK的性能变得不好

5、ZK的数据库完全放在内存中。 大量的Queue意味着会占用很多的内存空间。

尽管如此， Curator还是创建了各种Queue的实现。 如果Queue的数据量不太多，数据量不太大的情况下，酌情考虑，还是可以使用的。

#### 11、客户端对ServerList的轮询机制是什么

随机，客户端在初始化( new ZooKeeper(String connectString, int sessionTimeout, Watcher watcher) )的过程中，将所有Server保存在一个List中，然后随机打散，形成一个环。之后从0号位开始一个一个使用。

两个注意点：

1、Server地址能够重复配置，这样能够弥补客户端无法设置Server权重的缺陷，但是也会加大风险。（比如: 192.168.1.1:2181,192.168.1.1:2181,192.168.1.2:2181).

2、如果客户端在进行Server切换过程中耗时过长，那么将会收到SESSION\_EXPIRED. 这也是上面第1点中的加大风险之处。

#### 12、客户端如何正确处理CONNECTIONLOSS(连接断开) 和 SESSIONEXPIRED(Session 过期)两类连接异常

在ZooKeeper中，服务器和客户端之间维持的是一个长连接，在 SESSION\_TIMEOUT 时间内，服务器会确定客户端是否正常连接(客户端会定时向服务器发送heart\_beat),服务器重置下次SESSION\_TIMEOUT时间。因此，在正常情况下，Session一直有效，并且zk集群所有机器上都保存这个Session信息。在出现问题情况下，客户端与服务器之间连接断了（客户端所连接的那台zk机器挂了，或是其它原因的网络闪断），这个时候客户端会主动在地址列表（初始化的时候传入构造方法的那个参数connectString）中选择新的地址进行连接。

好了，上面基本就是服务器与客户端之间维持长连接的过程了。在这个过程中，用户可能会看到两类异常CONNECTIONLOSS(连接断开) 和SESSIONEXPIRED(Session 过期)。

CONNECTIONLOSS发生在上面红色文字部分，应用在进行操作A时，发生了CONNECTIONLOSS，此时用户不需要关心我的会话是否可用，应用所要做的就是等待客户端帮我们自动连接上新的zk机器，一旦成功连接上新的zk机器后，确认刚刚的操作A是否执行成功了。

#### 13、一个客户端修改了某个节点的数据，其它客户端能够马上获取到这个最新数据吗

ZooKeeper不能确保任何客户端能够获取（即Read Request）到一样的数据，除非客户端自己要求：方法是客户端在获取数据之前调用org.apache.zookeeper.AsyncCallback.VoidCallback, java.lang.Object) sync.   
通常情况下（这里所说的通常情况满足：1. 对获取的数据是否是最新版本不敏感，2. 一个客户端修改了数据，其它客户端是否需要立即能够获取最新），可以不关心这点。   
在其它情况下，最清晰的场景是这样：ZK客户端A对 /my\_test 的内容从 v1->v2, 但是ZK客户端B对 /my\_test 的内容获取，依然得到的是 v1. 请注意，这个是实际存在的现象，当然延时很短。解决的方法是客户端B先调用 sync(), 再调用 getData().

#### 14、Zookeeper为什么不提供一个永久性的Watcher注册机制

不支持用持久Watcher的原因很简单，ZK无法保证性能。   
使用watch需要注意的几点

1、Watches通知是一次性的，必须重复注册.

2、发生CONNECTIONLOSS之后，只要在session\_timeout之内再次连接上（即不发生SESSIONEXPIRED），那么这个连接注册的watches依然在。

3、节点数据的版本变化会触发NodeDataChanged，注意，这里特意说明了是版本变化。存在这样的情况，只要成功执行了setData()方法，无论内容是否和之前一致，都会触发NodeDataChanged。

4、对某个节点注册了watch，但是节点被删除了，那么注册在这个节点上的watches都会被移除。

5、同一个zk客户端对某一个节点注册相同的watch，只会收到一次通知。

6、Watcher对象只会保存在客户端，不会传递到服务端。

#### 15、我能否收到每次节点变化的通知

如果节点数据的更新频率很高的话，不能。   
原因在于：当一次数据修改，通知客户端，客户端再次注册watch，在这个过程中，可能数据已经发生了许多次数据修改，因此，千万不要做这样的测试：”数据被修改了n次，一定会收到n次通知”来测试server是否正常工作。（我曾经就做过这样的傻事，发现Server一直工作不正常？其实不是）。即使你使用了GitHub上这个客户端也一样。

#### 16、Zookeeper能为临时节点创建子节点吗

不能。

#### 17、在getChildren(String path, boolean watch)是注册了对节点子节点的变化，那么子节点的子节点变化能通知吗

不能

#### 18、创建的临时节点什么时候会被删除，是连接一断就删除吗？延时是多少？

连接断了之后，Zookeeper不会马上移除临时数据，只有当SESSIONEXPIRED之后，才会把这个会话建立的临时数据移除。因此，用户需要谨慎设置Session\_TimeOut

#### 19、zookeeper是否支持动态进行机器扩容？如果目前不支持，那么要如何扩容呢？

截止3.4.3版本的zookeeper，还不支持这个功能，在3.5.0版本开始，支持动态加机器了。20、ZooKeeper集群中服务器之间是怎样通信的？

Leader服务器会和每一个Follower/Observer服务器都建立TCP连接，同时为每个F/O都创建一个叫做LearnerHandler的实体。LearnerHandler主要负责Leader和F/O之间的网络通讯，包括数据同步，请求转发和Proposal提议的投票等。Leader服务器保存了所有F/O的LearnerHandler。

#### 21、Zookeeper是否会自动进行日志清理？如何进行日志清理？

Zookeeper自己不会进行日志清理，需要运维人员进行日志清理

#### 22、Zookeeper是如何保证事务的顺序一致性的

zookeeper采用了递增的事务Id来标识，所有的proposal都在被提出的时候加上了zxid，zxid实际上是一个64位的数字，高32位是epoch用来标识leader是否发生改变，如果有新的leader产生出来，epoch会自增，低32位用来递增计数。当新产生proposal的时候，会依据数据库的两阶段过程，首先会向其他的server发出事务执行请求，如果超过半数的机器都能执行并且能够成功，那么就会开始执行

#### 23、Zookeeper是如何选取主leader的？

当leader崩溃或者leader失去大多数的follower，这时zk进入恢复模式，

zk中znode类型有四种，持久化目录节点 持久化顺序编号目录节点(有顺序 能够在注册机器等许多场景用到) 临时目录节点 临时顺序编号节点

#### 24、 Zookeeper的通知机制

client端会对某个znode建立一个watcher事件，当该znode发生变化时，这些client会收到zk的通知，然后client可以根据znode变化来做出业务上的改变等。

#### 25、Zookeeper的配置管理

程序分布式的部署在不同的机器上，将程序的配置信息放在zk的znode下，当有配置发生改变时，也就是znode发生变化时，可以通过改变zk中某个目录节点的内容，利用water通知给各个客户端 从而更改配置。

#### 26、Zookeeper的命名服务

命名服务是指通过指定的名字来获取资源或者服务的地址，利用zk创建一个全局的路径，这个路径就可以作为一个名字，指向集群中的集群，提供的服务的地址，或者一个远程的对象等等。

#### 27、Zookeeper分布式通知和协调

对于系统调度来说：操作人员发送通知实际是通过控制台改变某个节点的状态，然后zk将这些变化发送给注册了这个节点的watcher的所有客户端。

对于执行情况汇报：每个工作进程都在某个目录下创建一个临时节点。并携带工作的进度数据，这样汇总的进程可以监控目录子节点的变化获得工作进度的实时的全局情况。

机器中为什么会有master；

在分布式环境中，有些业务逻辑只需要集群中的某一台机器进行执行，其他的机器可以共享这个结果，这样可以大大减少重复计算，提高性能，于是就需要进行master选举。

### 5.7 ActiveMQ

#### 1、ActiveMQ事务控制

在创建session时可以选择开启事务控制。所谓事务控制，即将消息生成发送，或接受，消费操作放入一个事务中。但不能同时控制发送与消费这一整个过程。因为事务都是基于一个session下的操作。

如下代码即开启了事务处理：

ActiveMQSession session =

connection.createSession(

**true**

,Session.CLIENT\_ACKNOWLEDGE);

在事务状态下进行发送操作，消息并未真正投递到中间件，而只有进行session.commit操作之后，消息才会发送到中间件，再转发到适当的消费者进行处理。如果是调用rollback操作，则表明，当前事务期间内所发送的消息都取消掉。此时无论commit或rollback，会重新打开一个事务。 与此同时，在rollback之后，随着新的事务打开，一些持久化的消息会重新接收。原因在于当传送模式处于持久话的状态，产生的消息如若没有被及时签收确认，则消息会被中间件持久化。此时，当客户端重新连接或新的事务开启，消息会被再次发送到客户端。

为什么commit之后，不会有持久的消息重新传送呢?

原因在于commit操作会自动将为签收确认的消息进行签收确认，如果是当前接收但未签收确认的消息，都会被确认处理。因而在commit之后不会有持久化的消息出现。

### 5.8 Solr

#### 1、Lucene的作用

lucene是一个全文检索引擎，在进行模糊匹配的时候，他可以用来替代数据库中的like,从而在匹配准确性以及性能进行大幅度的提高。我在做XX项目的XX模块的时候，就是用lucene来进行全文检索用IK分词器来进行分词。从而实现了高亮显示关键词，分页，排序，多字段，多条件的高性能搜索。在从数据中取数据生成索引的时候，因为表中的数据量比较大，防止一次取出所导致内存溢出问题，我采用了分段批量提取的方式进行，除此之外我们对后续增加的数据根据优先级的不同采取不同的策略，对于那些需要及时显示的数据我们通过spring定时器 在短时间内(30分钟)进行增量索引的生成，对于那些不需要及时展示的数据,我们通过spring定时器在每天晚上凌晨的时候进行索引的重新生成。

### 5.9 Elasticsearch

#### 1、Elasticsearch是如何实现Master选举的？

* Elasticsearch的选主是ZenDiscovery模块负责的，主要包含Ping（节点之间通过这个RPC来发现彼此）和Unicast（单播模块包含一个主机列表以控制哪些节点需要ping通）这两部分；
* 对所有可以成为master的节点（**node.master: true**）根据nodeId字典排序，每次选举每个节点都把自己所知道节点排一次序，然后选出第一个（第0位）节点，暂且认为它是master节点。
* 如果对某个节点的投票数达到一定的值（可以成为master节点数n/2+1）并且该节点自己也选举自己，那这个节点就是master。否则重新选举一直到满足上述条件。
* 补充：master节点的职责主要包括集群、节点和索引的管理，不负责文档级别的管理；data节点可以关闭http功能。

#### 2、Elasticsearch中的节点（比如共20个），其中的10个选了一个master，另外10个选了另一个master，怎么办？

* 当集群master候选数量不小于3个时，可以通过设置最少投票通过数量（**discovery.zen.minimum\_master\_nodes**）超过所有候选节点一半以上来解决脑裂问题；
* 当候选数量为两个时，只能修改为唯一的一个master候选，其他作为data节点，避免脑裂问题。

#### 3、客户端在和集群连接时，如何选择特定的节点执行请求的？

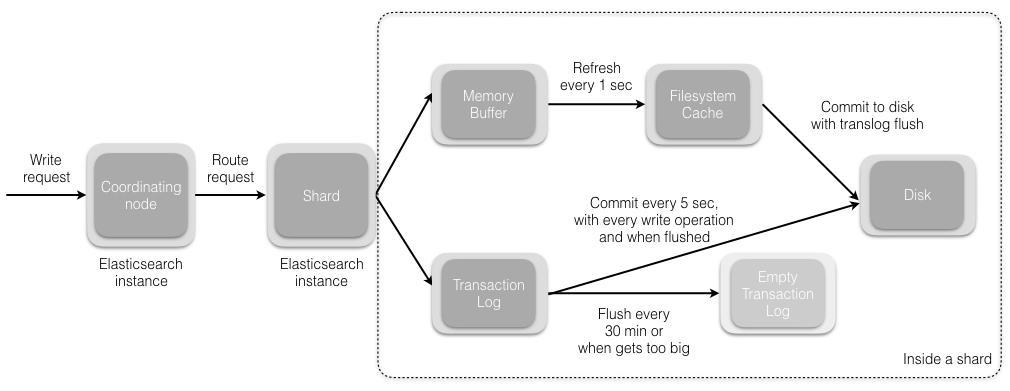
* TransportClient利用transport模块远程连接一个elasticsearch集群。它并不加入到集群中，只是简单的获得一个或者多个初始化的transport地址，并以 **轮询** 的方式与这些地址进行通信。/

#### 4、详细描述一下Elasticsearch索引文档的过程。

* 协调节点默认使用文档ID参与计算（也支持通过routing），以便为路由提供合适的分片。

shard = hash(document\_id) % (num\_of\_primary\_shards)

* 当分片所在的节点接收到来自协调节点的请求后，会将请求写入到Memory Buffer，然后定时（默认是每隔1秒）写入到Filesystem Cache，这个从Momery Buffer到Filesystem Cache的过程就叫做refresh；
* 当然在某些情况下，存在Momery Buffer和Filesystem Cache的数据可能会丢失，ES是通过translog的机制来保证数据的可靠性的。其实现机制是接收到请求后，同时也会写入到translog中，当Filesystem cache中的数据写入到磁盘中时，才会清除掉，这个过程叫做flush；
* 在flush过程中，内存中的缓冲将被清除，内容被写入一个新段，段的fsync将创建一个新的提交点，并将内容刷新到磁盘，旧的translog将被删除并开始一个新的translog。
* flush触发的时机是定时触发（默认30分钟）或者translog变得太大（默认为512M）时；

[](https://upload-images.jianshu.io/upload_images/3709321-2084bd0268a42ae1.jpeg?imageMogr2/auto-orient/strip|imageView2/2/w/1240)

补充：关于Lucene的Segement：

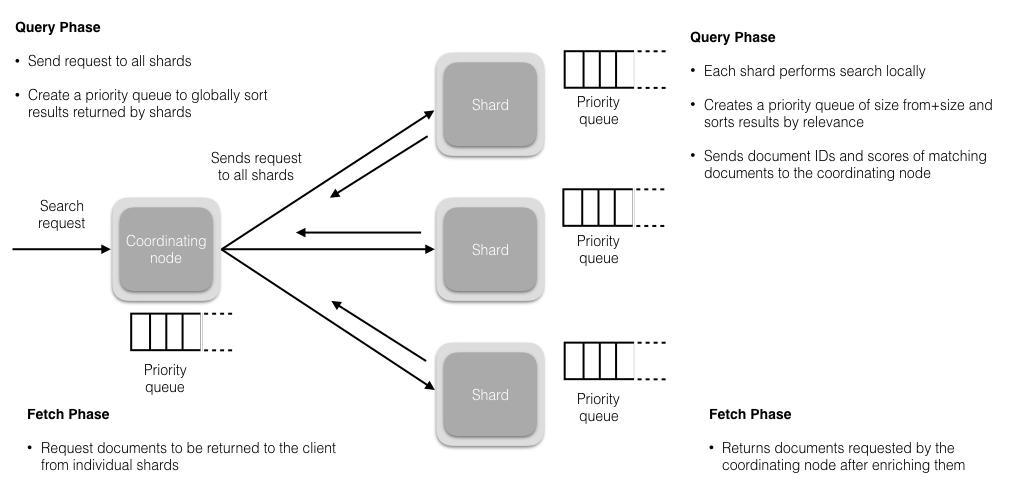
* Lucene索引是由多个段组成，段本身是一个功能齐全的倒排索引。
* 段是不可变的，允许Lucene将新的文档增量地添加到索引中，而不用从头重建索引。
* 对于每一个搜索请求而言，索引中的所有段都会被搜索，并且每个段会消耗CPU的时钟周、文件句柄和内存。这意味着段的数量越多，搜索性能会越低。
* 为了解决这个问题，Elasticsearch会合并小段到一个较大的段，提交新的合并段到磁盘，并删除那些旧的小段。

#### 5、详细描述一下Elasticsearch更新和删除文档的过程。

* 删除和更新也都是写操作，但是Elasticsearch中的文档是不可变的，因此不能被删除或者改动以展示其变更；
* 磁盘上的每个段都有一个相应的.del文件。当删除请求发送后，文档并没有真的被删除，而是在.del文件中被标记为删除。该文档依然能匹配查询，但是会在结果中被过滤掉。当段合并时，在.del文件中被标记为删除的文档将不会被写入新段。
* 在新的文档被创建时，Elasticsearch会为该文档指定一个版本号，当执行更新时，旧版本的文档在.del文件中被标记为删除，新版本的文档被索引到一个新段。旧版本的文档依然能匹配查询，但是会在结果中被过滤掉。/

#### 6、详细描述一下Elasticsearch搜索的过程。

* 搜索被执行成一个两阶段过程，我们称之为 Query Then Fetch；
* 在初始查询阶段时，查询会广播到索引中每一个分片拷贝（主分片或者副本分片）。 每个分片在本地执行搜索并构建一个匹配文档的大小为 from + size 的优先队列。PS：在搜索的时候是会查询Filesystem Cache的，但是有部分数据还在Memory Buffer，所以搜索是近实时的。
* 每个分片返回各自优先队列中 **所有文档的 ID 和排序值** 给协调节点，它合并这些值到自己的优先队列中来产生一个全局排序后的结果列表。
* 接下来就是 取回阶段，协调节点辨别出哪些文档需要被取回并向相关的分片提交多个 GET 请求。每个分片加载并 丰富 文档，如果有需要的话，接着返回文档给协调节点。一旦所有的文档都被取回了，协调节点返回结果给客户端。
* 补充：Query Then Fetch的搜索类型在文档相关性打分的时候参考的是本分片的数据，这样在文档数量较少的时候可能不够准确，DFS Query Then Fetch增加了一个预查询的处理，询问Term和Document frequency，这个评分更准确，但是性能会变差。

[](https://upload-images.jianshu.io/upload_images/3709321-88f589037638c93d.jpeg?imageMogr2/auto-orient/strip|imageView2/2/w/1240)

#### 7、Elasticsearch在部署时，对Linux的设置有哪些优化方法？

64 GB 内存的机器是非常理想的， 但是32 GB 和16 GB 机器也是很常见的。少于8 GB 会适得其反。

如果你要在更快的 CPUs 和更多的核心之间选择，选择更多的核心更好。多个内核提供的额外并发远胜过稍微快一点点的时钟频率。

如果你负担得起 SSD，它将远远超出任何旋转介质。 基于 SSD 的节点，查询和索引性能都有提升。如果你负担得起，SSD 是一个好的选择。

即使数据中心们近在咫尺，也要避免集群跨越多个数据中心。绝对要避免集群跨越大的地理距离。

请确保运行你应用程序的 JVM 和服务器的 JVM 是完全一样的。 在 Elasticsearch 的几个地方，使用 Java 的本地序列化。

通过设置gateway.recover\_after\_nodes、gateway.expected\_nodes、gateway.recover\_after\_time可以在集群重启的时候避免过多的分片交换，这可能会让数据恢复从数个小时缩短为几秒钟。

Elasticsearch 默认被配置为使用单播发现，以防止节点无意中加入集群。只有在同一台机器上运行的节点才会自动组成集群。最好使用单播代替组播。

不要随意修改垃圾回收器（CMS）和各个线程池的大小。

把你的内存的（少于）一半给 Lucene（但不要超过 32 GB！），通过ES\_HEAP\_SIZE 环境变量设置。

内存交换到磁盘对服务器性能来说是致命的。如果内存交换到磁盘上，一个 100 微秒的操作可能变成 10 毫秒。 再想想那么多 10 微秒的操作时延累加起来。 不难看出 swapping 对于性能是多么可怕。

Lucene 使用了大量的文件。同时，Elasticsearch 在节点和 HTTP 客户端之间进行通信也使用了大量的套接字。 所有这一切都需要足够的文件描述符。你应该增加你的文件描述符，设置一个很大的值，如 64,000。

补充：索引阶段性能提升方法

使用批量请求并调整其大小：每次批量数据 5–15 MB 大是个不错的起始点。

存储：使用 SSD

段和合并：Elasticsearch 默认值是 20 MB/s，对机械磁盘应该是个不错的设置。如果你用的是 SSD，可以考虑提高到 100–200 MB/s。如果你在做批量导入，完全不在意搜索，你可以彻底关掉合并限流。另外还可以增加 index.translog.flush\_threshold\_size 设置，从默认的 512 MB 到更大一些的值，比如 1 GB，这可以在一次清空触发的时候在事务日志里积累出更大的段。

如果你的搜索结果不需要近实时的准确度，考虑把每个索引的index.refresh\_interval 改到30s。

如果你在做大批量导入，考虑通过设置index.number\_of\_replicas: 0 关闭副本。

#### 8、对于GC方面，在使用Elasticsearch时要注意什么？

倒排词典的索引需要常驻内存，无法GC，需要监控data node上segment memory增长趋势。

各类缓存，field cache, filter cache, indexing cache, bulk queue等等，要设置合理的大小，并且要应该根据最坏的情况来看heap是否够用，也就是各类缓存全部占满的时候，还有heap空间可以分配给其他任务吗？避免采用clear cache等“自欺欺人”的方式来释放内存。

避免返回大量结果集的搜索与聚合。确实需要大量拉取数据的场景，可以采用scan & scroll api来实现。

cluster stats驻留内存并无法水平扩展，超大规模集群可以考虑分拆成多个集群通过tribe node连接。

想知道heap够不够，必须结合实际应用场景，并对集群的heap使用情况做持续的监控。

#### 9、Elasticsearch对于大数据量（上亿量级）的聚合如何实现？

* Elasticsearch 提供的首个近似聚合是cardinality 度量。它提供一个字段的基数，即该字段的distinct或者unique值的数目。它是基于HLL算法的。HLL 会先对我们的输入作哈希运算，然后根据哈希运算的结果中的 bits 做概率估算从而得到基数。其特点是：可配置的精度，用来控制内存的使用（更精确 ＝ 更多内存）；小的数据集精度是非常高的；我们可以通过配置参数，来设置去重需要的固定内存使用量。无论数千还是数十亿的唯一值，内存使用量只与你配置的精确度相关。

#### 10、在并发情况下，Elasticsearch如果保证读写一致？

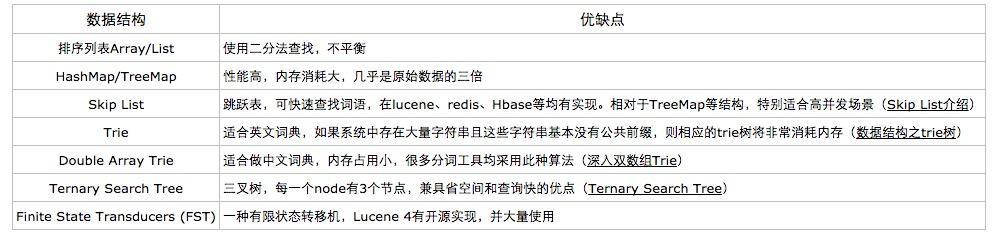
* 可以通过版本号使用乐观并发控制，以确保新版本不会被旧版本覆盖，由应用层来处理具体的冲突；
* 另外对于写操作，一致性级别支持quorum/one/all，默认为quorum，即只有当大多数分片可用时才允许写操作。但即使大多数可用，也可能存在因为网络等原因导致写入副本失败，这样该副本被认为故障，分片将会在一个不同的节点上重建。
* 对于读操作，可以设置replication为sync(默认)，这使得操作在主分片和副本分片都完成后才会返回；如果设置replication为async时，也可以通过设置搜索请求参数\_preference为primary来查询主分片，确保文档是最新版本。/

#### 11、如何监控Elasticsearch集群状态？

* Marvel 让你可以很简单的通过 Kibana 监控 Elasticsearch。你可以实时查看你的集群健康状态和性能，也可以分析过去的集群、索引和节点指标。

#### 12、是否了解字典树？

* 常用字典数据结构如下所示：

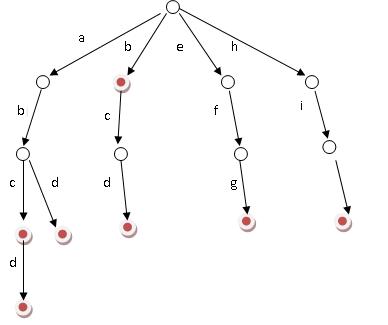
[](https://upload-images.jianshu.io/upload_images/3709321-7b6f0fab6f412f51.png?imageMogr2/auto-orient/strip|imageView2/2/w/1240)

* Trie的核心思想是空间换时间，利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。它有3个基本性质：

根节点不包含字符，除根节点外每一个节点都只包含一个字符。

从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。

每个节点的所有子节点包含的字符都不相同。

[](https://github.com/julycoding/The-Art-Of-Programming-By-July/raw/master/ebook/images/8/8.4/1.jpg)

* 可以看到，trie树每一层的节点数是26^i级别的。所以为了节省空间，我们还可以用动态链表，或者用数组来模拟动态。而空间的花费，不会超过单词数×单词长度。
* 实现：对每个结点开一个字母集大小的数组，每个结点挂一个链表，使用左儿子右兄弟表示法记录这棵树；
* 对于中文的字典树，每个节点的子节点用一个哈希表存储，这样就不用浪费太大的空间，而且查询速度上可以保留哈希的复杂度O(1)。

#### 13、拼写纠错是如何实现的？

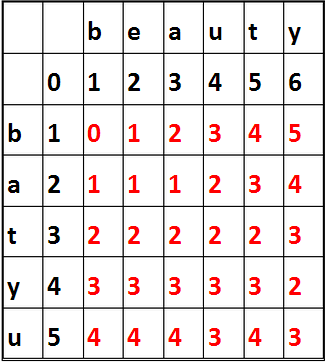
* 拼写纠错是基于编辑距离来实现；编辑距离是一种标准的方法，它用来表示经过插入、删除和替换操作从一个字符串转换到另外一个字符串的最小操作步数；
* 编辑距离的计算过程：比如要计算batyu和beauty的编辑距离，先创建一个7×8的表（batyu长度为5，coffee长度为6，各加2），接着，在如下位置填入黑色数字。其他格的计算过程是取以下三个值的最小值：

如果最上方的字符等于最左方的字符，则为左上方的数字。否则为左上方的数字+1。（对于3,3来说为0）

左方数字+1（对于3,3格来说为2）

上方数字+1（对于3,3格来说为2）

最终取右下角的值即为编辑距离的值3。

[](https://upload-images.jianshu.io/upload_images/3709321-31bef8a5bbf14a13.png?imageMogr2/auto-orient/strip|imageView2/2/w/1240)

* 对于拼写纠错，我们考虑构造一个度量空间（Metric Space），该空间内任何关系满足以下三条基本条件：

d(x,y) = 0 -- 假如x与y的距离为0，则x=y

d(x,y) = d(y,x) -- x到y的距离等同于y到x的距离

d(x,y) + d(y,z) >= d(x,z) -- 三角不等式

* 根据三角不等式，则满足与query距离在n范围内的另一个字符转B，其与A的距离最大为d+n，最小为d-n。
* BK树的构造就过程如下：每个节点有任意个子节点，每条边有个值表示编辑距离。所有子节点到父节点的边上标注n表示编辑距离恰好为n。比如，我们有棵树父节点是”book”和两个子节点”cake”和”books”，”book”到”books”的边标号1，”book”到”cake”的边上标号4。从字典里构造好树后，无论何时你想插入新单词时，计算该单词与根节点的编辑距离，并且查找数值为d(neweord, root)的边。递归得与各子节点进行比较，直到没有子节点，你就可以创建新的子节点并将新单词保存在那。比如，插入”boo”到刚才上述例子的树中，我们先检查根节点，查找d(“book”, “boo”) = 1的边，然后检查标号为1的边的子节点，得到单词”books”。我们再计算距离d(“books”, “boo”)=2，则将新单词插在”books”之后，边标号为2。
* 查询相似词如下：计算单词与根节点的编辑距离d，然后递归查找每个子节点标号为d-n到d+n（包含）的边。假如被检查的节点与搜索单词的距离d小于n，则返回该节点并继续查询。比如输入cape且最大容忍距离为1，则先计算和根的编辑距离d(“book”, “cape”)=4，然后接着找和根节点之间编辑距离为3到5的，这个就找到了cake这个节点，计算d(“cake”, “cape”)=1，满足条件所以返回**cake**，然后再找和cake节点编辑距离是0到2的，分别找到cape和cart节点，这样就得到**cape**这个满足条件的结果。

### 5.10 Memcached

#### 1、Memcached介绍

Memcached是一个用C语言开发的分布式的缓存，内部基于类似hashMap的结构。

它的优点是协议简单，内置内存存储，并且他的

分布式算法是在客户端完成的，不需要服务器端进行

通信，我们当时在做项目的时候因为考虑到项目

的高可用性高扩展性，因此在服务器部署方面采用

了apache+jk+tomcat这种负载均衡的方式，但是也带来了一个问题

就是session共享的问题，虽然可以通过session复制来解决这个

问题，但是在性能方面存在缺陷，所以最后我们采用了

用memcached来存储session，这样既解决了session共享

问题，也解决了session复制那种方式所产生的性能问题。

了解(不必主动说，但别人问的话一定要知道)

memcached是以KEY-VALUE的方式进行数据存储的，

KEY的大小限制：Key（max）<=250个字符；

VALUE在存储时有限制：Value（max）<= 1M；

根据最近最少使用原则删除对象即LRU.

memcached默认过期时间：ExpiresTime（max）= 30（days）

### 5.11、FreeMarker

#### 1、FreeMarker描述

FreeMarker是一个用Java语言编写的模板引擎，它是基于模板来生成文本输出的通用工具。Freemarker可以生成HTML， XML，JSP或Java等多种文本输出。

工作原理：定义模板文件，嵌入数据源，通过模板显示准备的数据

（数据 + 模板 = 输出）

　　我们在使用模板中发现freemarker具有许多优点，它彻底的分离表现层和业务逻辑，模板只负责数据在页面中的表现，不涉及任何的逻辑代码，所以使得开发过程中的人员分工更加明确，作为界面开发人员，只需专心创建HTML文件、图像以及Web页面的其他可视化方面，不用理会数据；而程序开发人员则专注于系统实现，负责为页面准备要显示的数据。

如果使用jsp来展示，开发阶段进行功能调适时，需要频繁的修改JSP，每次修改都要编译和转换，浪费了大量时间，FreeMarker模板技术不存在编译和转换的问题，在开发过程中，我们在不必在等待界面设计开发人员完成页面原型后再来开发程序。由此使用freemarker还可以大大提高开发效率。

## 六 Linux

#### 1、如何对一台[Linux](http://lib.csdn.net/base/linux)服务器进行系统性能调优，列举出参数。

答：主机的存活状态、系统负载值、CPU的使用率、磁盘IO量，swap进出量，网络流量，[数据库](http://lib.csdn.net/base/mysql)的主从复制状态、LVS+keepalived的VIP。

#### 2、DNS服务器的工作原理。

答：首先客户机的应用进程将待解析的域名放在DNS请求报文中，以UDP用户数据报方式发给本地域名服务器（使用UDP是因为它无连接，开销小），本地域名服务器查看缓存是否对对应域名和IP地址的映射关系，如果有，直接返回IP地址给主机，如果没有，就将向转发器发出查询请求，转发器是网络上的DNS服务器，目的是提高解析效率，如果连转发器也没有的话，就进行递归解析，转发报文从根域名服务器-顶级域名服务器-二级域名服务器等一直找到完整的域名后，进行反向回溯给交本地域名服务器，缓存后再由本地域名服务器将对应的IP地址放在应答报文中，从而主机得到IP地址即可通信。

#### 5、修改第一块网卡的路径是什么。

/etc/sysconfig/network-scripts/ifcfg-eth 0

#### 6、 简述I节点是什么，并说出[linux](http://lib.csdn.net/base/linux) 文件系统通过i 节点把文件的逻辑结构和物理结构转换的工作过程？

答：I节点是一个多级索引的[**数据结构**](http://lib.csdn.net/base/datastructure)，它的基本思想是给每个文件赋予一张成为I节点的小表，在这张小表中列出了文件属性以及文件各数据块中在磁盘中的地址。那么它的工作工程：当需要访问文件时，内核查看文件目录项找到对象的文件名和I节点号，那么再通过这个I节点号来找磁盘的地址，实现逻辑结构和物理结构的转换，返回给数据给用户显示。

#### 7、使用shell，建立classl用户组，再批量建立stu 1--stu30的用户，并指定用户组为classl。

答：vi yonghu.sh

#!/bin/bash

groupadd class1

for((i=1;i<=30;i=i+1))

do

useradd stu$i -G class1

done

#### 8、日常监控都需要监控哪些？

1)硬件:CPU:cat /proc/cpuinfo,内存: cat /proc/meminfo，硬盘:fdisk -l（dmesg是用来查看硬件信息的）

2)系统:负载:cat /proc/loadavg, uptime查看实时load average、swap，虚拟内存:vmstat(参数一s;  2 4)，vmstat 5 5（表示在5秒时间内进行5次采样)。

系统日志:tail -f /var/log/messages

logwatch一print一range Today一service SSHD一service pam\_unix

3)网络:Host\_ Alive,Ping，端口，连接

#### 9、如何将本地80端口的请求转发到8080端口，当前主机IP为192.168.16.1，其中本地网卡eth0。

答：#iptables -t nat -A PREROUTING -p tcp -i eth0 -d 192.168.16.1--dport

80 -j REDIRECT --to-port 8080

#### 10、包过滤防火墙与代理应用防火墙有什么区别，能列举几种相应的产品吗?

答：①包过滤防火墙工作在网络协议IP层，它只对IP包的源地址、目标地址及相应端口进行处理，因此速度比较快，能够处理的并发连接比较多，缺点是对应用层的攻击无能为力。②应用代理防火墙将收到的IP包解封装成高层协议的通讯数据，比如http连接信息，因此能够对基于高层协议的攻击进行拦截。缺点是处理速度比较慢，能够处理的并发数比较少。事实上还有状态监测防火墙（ip\_conntrack）：状态监测防火墙是在当今企业中部署最为常见的。他们建立在基于防火墙保持每个活动连接状态信息的包过滤上。当有一个新的包到达防火墙时，过滤机制首先检查这个包是否是当前活动连接（前面已经授权过的）的一部分。只有当这个包没有出现在当前的活动连接列表里时，防火墙才会以它的规则库评估这个包。状态监测防火墙之所以如此常见是因为：它们是效率最高、性价比最高的防火墙，并且广泛适用于保护网络的边界。ASA、PIX、firewall

#### 11、iptables是否支持time时间控制用户行为，如有请写出具体操作步骤

iptables -A INPUT -p ICMP --icmp-type 8 -m time --timestart 00:00:00 --timestop 23:59:59 --weekdays Mon -j DROP

限制在每个星期一的0点0分0秒到23点59分59秒这个时间段内，所有icmp协议报文的应答都会拒绝，

#### 12、Linux常见打包工具并写相应解压缩参数?

答：①用tar调用bzip2将file1,file2,file3生成压缩包

        tar -jcvf file.tar.bz2 file{1,2,3}  解压：tar -jxvf file.tar.bz2

 ②用tar调用gzip将file1,file2,file3生成压缩包

        tar -zcvf file.tar.gz file{1,2,3}  解压：tar -zxvf file.tar.gz

③.用tar调用xz将file1,file2,file3生成压缩包

        tar -Jcvf file.tar.xz file{1,2,3} 解压：tar -Jxvf file.tar.xz

#### 13、当用户在浏览器当中输入一个网站，说说计算机对dns解释经过那些流程?注:本地dns还没有缓存。

答:a.用户输入网址到浏览器

b.浏览器发出DNS请求报文

c.计算机首先查询本机HOSTS文件，看是否存在，存在直接返回结果，不存在，继续转发请求报文

d.计算机按照本地DNS的顺序，向合法dns服务器查询IP结果，

e.合法dns返回dns结果给本地dns，管理该域的本地dns服务器缓存本结果，直到缓存时间过期，才再次查询此结果。

#### 14、DNS什么使用TCP，什么时候使用UDP，为什么这么设计？

答：DNS在区域复制的使用TCP，因为主域名服务器传输数据到辅助域名服务器，必须得保证我们传输的数据是可靠的，是没有差错的；那么在浏览器发出DNS请求报文的时候使用UDP，原因是为了减少它的开销，无连接，把这个数据包扔了就走，什么都不用管。

#### 15、请解释FTP控制连接的工作模式？

答：①在主动模式下，FTP客户端随机开启一个大于1024的端口N向服务器的TCP 21号端口发起连接，然后开放N+1号端口进行监听，并向服务器发出PORT N+1命令（用来告诉服务器连接它的哪个新端口）。服务器接收到命令后，会用其本地的FTP数据端口(通常是20)来连接客户端指定的端口N+1，进行数据传输。

②在被动模式下，FTP客户端随机开启一个大于1024的端口N向服务器的21号端口发起连接，同时会开启N+1号端口。然后向服务器发送PASV命令，通知服务器自己处于被动模式（此端口是空闲的、可用的）。服务器收到命令后，会开放一个大于1024的端口P进行监听，然后用PORT命令通知客户端，自己的数据端口是P。客户端收到命令后，会通过N+1号端口连接服务器的端口P，然后在两个端口之间进行数据传输。

总的来说，主动模式的FTP是指服务器主动连接客户端的数据端口，被动模式的FTP是指服务器被动地等待客户端连接自己的数据端口。

附：传输模式是二进制模式和ASCII文本模式，常用ftp命令有get、put、ls、dir

#### 16、vi file

50 16 \* \* \* rm -r /abc/\* #每天下午4:40删除

55 17 \* \* \*　umount /dev/hdc #卸载CDROM

0 8-18/1 \* \* \* cut -f 1 /xyz/x1 >> /backup/bak01.txt

#每天的8:00-18:00时间段内每隔1小时，都会截取/xyz/x1的第一列（每行第一个域）的数据以追加的方式放入到/backup/bak01.txt文件中

50 5 \* \* 1 tar -zcvf backup.tar.gz /data 每周星期一的下午5:50将/data目录下所有文件打包并压缩

root@yhc#: crontab file #这样也能进行定时任务而不用edit

#### 17、 什么是符号链接，什么是硬链接？符号链接与硬链接的区别是什么？

答：链接分硬链接和符号链接。符号链接可以建立对于文件和目录的链接。符号链接可以跨磁盘分区，符号链接的文件类型位是I，链接文件具有新的i节点；硬链接不可以跨磁盘分区。它只能建立对文件的链接，硬链接的文件类型位是-号，且硬链接文件的i节点同被链接文件的i节点相同。

#### 18、简述进程的启动、终止的方式以及如何进行进程的査看。

在 Linux中启动一个进程有手工启动和调度启动两种方式：

(1) 手工启动

用户在输入端发出命令，直接启动一个进程的启动方式。可以分为：

①前台启动：直接在SHELL中输入命令进行启动。

②后台启动：启动一个目前并不紧急的进程，如打印进程。

(2) 调度启动

系统管理员根据系统资源和进程占用资源的情况，事先进行调度安排，指定任务运行的时间和场合，到时候系统会自动完成该任务。

经常使用的进程调度命令为：at、batch、crontab。进程查看ps -aux | grep httpd，进程终止kill-9 进程号。

#### 19、Linux硬盘分了多少个块？

答：linux硬盘组织方式为:引导区、超级块、索引结点、数据块、目录块；其中超级块中包含了关于该硬盘或分区上的文件系统的整体信息，如文件系统的大小等，超级决包含了i节点表和空闲块表等的文件系统信息；超级块后面的数据结构是索引结点，包含了针对某一个具体文件的几乎全部信息，如文件的存取权、所有者、大小、建立时间以及对应的目录块和数据块等:数据块是真正存储文件内蓉的位置。但是索引结点中不包括文件的名字，文件名是放在目录块里的。目录块里包含有文件的名字以及此文件的索引结点编号。

#### 20、NTFS与FAT的区别？

答：NTFS的特性相比FAT有很多优点。①提供文件和文件夹的安全性，支持加密和设置权限②高可靠性，NTFS是一种可恢复的文件系统，并且支持坏簇映射，检测出现错误的磁盘区域 ，标记并将数据存储在其他区域③NTFS支持磁盘配额管理，另外支持对文件、文件夹以及分区的压缩。④NTFS采用了更小的簇，4KB左右，避免了对磁盘空间的；浪费。那么FAT记录文件分配表的格式，缺点是仅用于单用户系统，而且没有防止碎片的措施，并且易受损害。

#### 21、如何挂载windows共享的目录？

答：mount -t cifs -o username=administrator,password=admin //192.168.1.1/share/ /windows/ #optional非强制的，可选的，CIFS是公共的或开放的[SMB](http://baike.baidu.com/view/262410.htm)协议版本，Common Internet File System通用网络文件系统，并由Microsoft使用，/windows/是我们的挂载点。

#### 22、如何查看当前Linux 系统的状态,如CPU 使用,内存使用,负载情况等.

答：Linux 系统中/proc是个伪文件目录,不占用系统空间，及时的反应出内存现在使用的进程情况......其中许多文件都保存系统运行状态和相关信息

对于/proc中文件可使用文件查看命令浏览cat其内容，文件中包含系统特定信息：

cpuinfo 主机CPU 信息

filesystems 文件系统信息

meminfo 主机内存信息

version Linux 内存版本信息

diskstatus 磁盘负载情况

另外top 命令可以动态的显示服务器健康状况的使用情况. free 命令呢可以查看真实使用的内存 一般用free -m 。使用lsof (list open files)是一个列出当前系统打开文件的工具使用lsof -i:8080, -i 用以显示符合条件的进程情况

ps -aux 可以查看详细的每个进程的使用状况,还有uptime查看负载命令

dmesg 也是常用来查看系统性能及硬件的命令

#### 23、解释top命令和vmstat命令

答：top命令：查看CPU状态、内存状态、进程状态、交换分区状态，系统启动时间以及负载情况。vmstat 2 1现给定时间间隔的服务器的状态值,包括服务器的CPU使用率，内存使用，虚拟内存交换情况,IO读写情况，2表示每个两秒采集一次服务器状态，1表示只采集一次。如果不加1就一直采集。

vmstat 5 6

procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------  
r b swpd free buff cache si so bi bo in cs us sy id wa st  
0 0 0 3029876 199616 690980 0 0 0 2 3 2 0 0 100 0 0

#### 24、用sed命令和cat将test.txt所有回车替换为空格？

答：cat test.txt | tr "\n" " "   ；sed 's/\n/ /g' test.txt； route add -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.1.1 ；

 sed '1c Hi' ab                #第一行代替为Hi

 sed '1,2c Hi' ab             #第一行到第二行代替为Hi

 sed s/\'//g  替换单引号为空

格式：sed 's/要替换的字符串/新的字符串/g'   （要替换的字符串可以用正则表达式）

#### 25、如何看到用户登录日志？

答：more /var/log/secure是记录了所有用户登陆信息 ，who /var/log/wtmp可以干了些什么？root账户下输入su - username，切换到username下输入history，能看到这个用户历史命令，默认最近的1000条。还可以用last命令显示自从文件第一次创建以来登录过的用户

39、常用命令实例：

（1）打印出含有空行的行号：grep -n ^$ b.bash |awk 'BEGIN{FS=":"}{print$1}'

其中分隔符等价于awk -F ":"  '{print $1}’

（2）打印文件的第2行到第5行：sed -n '2,5p' b.bash

（3）把文件中第10行的sum改为yhc：sed '10s/sum/yhc/g'  b.bash

（4）在匹配式样“done”的行之后插入一空行：sed  '/done/G' b.bash （G附加暂存空间到模式空间）

（5）在文件第10行首添加5个空格：sed  '10s/^/     /g'   b.bash

（6）从a.log文件中提取包含“warning”或者“fatal”，同时不包含“IGNOR”的行，然后提取“：”分割的第五个字段

 egrep 'WARNING|FATAL' a.log | egrep -v 'IGNOR' | awk -F ":" '{print $5}'

40、在Shell环境下,如何查看远程Linux系统运行了多少时间？

答：ssh user@被监控主机ip "uptime"

在每个月的第一天备份并压缩/etc目录下的所有内容，存放在/root/backup目录下，文件名为yymmdd\_etc,shell程序fileback.sh存放在/usr/bin目录下？

 #!/bin/bash

   filename=`date +%y%m%d`\_etc.tar.gz    #date +%y%m%d直接就能输出160612

   cd /etc

   tar -zcvf $filename \*

   mv $filename /root/backup/   #必须先要创建这个文件才行。

#echo \* \* 1 \* \* root ./fileback.sh & >>/etc/crontab

41、介绍下LVS负载模式和调度[**算法**](http://lib.csdn.net/base/datastructure)？

答：（1）DNAT模式：画图说明

（2）隧道模式：调度器在数据报文的头部再添加一个IP地址的，类似于在VPN中， 公有IP封装在外，私有IP在内，RS收到后再进行解封装，这种情况主要用于远程站点，比如说有台服务器部署在美国，他要求调度器和RS都要支持隧道协议。

（3）DR直接路由模式，要求RS和调度器共享一个VIP，并且他们的真实IP都处于同一个网段，以便发出ARP请求，在调度器的出接口将目的MAC地址替换为RS的目的MAC地址并转发出去，RS收到数据包处理后，直接返回客户端，不经过调度器了。

调度算法分为静态调度和动态调度，比如轮询和加权最少连接。

42、linux上进程有5种状态:  
1. 运行(正在运行或在运行队列中等待)  
2. 中断(休眠中, 受阻, 在等待某个条件的形成或接受到信号)  
3. 不可中断(收到信号不唤醒和不可运行, 进程必须等待直到有中断发生)  
4. 僵死(进程已终止, 但进程描述符存在, 直到父进程调用wait4()系统调用后释放)  
5. 停止(进程收到SIGSTOP, SIGSTP, SIGTIN, SIGTOU信号后停止运行运行)

43、30道面试题

（1）、查看http的并发连接数与其TCP连接状态

netstat -n | awk '/^tcp/ {++b[$NF]} END {for(a in b) print a, b[a]}'

（2）、linux如何挂载windows的共享文件夹

mount -t cifs -o username=WindowsLogin,password=passwordinWindows//16.187.190.50/test    /mnt/   （-o optional 可选的）

（3）统计出apache的access.log中访问量最多的5个IP:

 cat access\_log | awk  '{print $1}' | sort | uniq -c | sort -n -r | head -5

(uniq命令可以去除排序过的文件中的重复行,-c  ：进行计数,sort -n,你有没有遇到过10比2小的情况。我反正遇到过。出现这种情况是由于排序程序将这些数字按字符来排序了，排序程序会先比较1和2，显然1小，所以就将10放在2前面。我们如果想改变这种现状，就要使用-n选项，来告诉sort，“要以数值来排序”,-r代表降序排序)

（4）ps aux 中的VSZ代表什么意思，RSS代表什么意思

VSZ:虚拟内存集,进程占用的虚拟内存空间

RSS:物理内存集,进程占用实际物理内存空间

a 显示所有与终端相关的进程,由终端发起的.  
x 显示所有与终端无关的进程.  
u 显示用户导向的用户列表.

（5）Linux系统的开机启动顺序

加载BIOS–>读取MBR–>Boot Loader–>加载内核–>用户层init根据inittab文件来设定系统运行的等级(一般3或者5，3是多用户命令行，5是界面)–>init进程执行rc.syninit–>启动内核[模块](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http://zhangge.net/1986.html&p=baidu&c=news&n=10&t=tpclicked3_hc&q=31072150_cpr&k=%C4%A3%BF%E9&k0=%C4%A3%BF%E9&k1=%B6%CB%BF%DA&k2=%BA%AF%CA%FD&k3=%CE%C4%BC%FE%CF%B5%CD%B3&k4=mbr&k5=etc&sid=c3a8a398a4cec66a&ch=0&tu=u1682464&jk=c8fabfe794451dcc&cf=29&fv=11&stid=9&urlid=0&luki=6&seller_id=1&di=128)–>执行不同级别运行的脚本程序–>执行/[etc](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http://zhangge.net/1986.html&p=baidu&c=news&n=10&t=tpclicked3_hc&q=31072150_cpr&k=etc&k0=etc&k1=%BF%CD%BB%A7%B6%CB&k2=%B7%FE%CE%F1%C6%F7&k3=%B4%C5%C5%CC&k4=tcp&k5=%C4%A3%BF%E9&sid=c3a8a398a4cec66a&ch=0&tu=u1682464&jk=c8fabfe794451dcc&cf=29&fv=11&stid=9&urlid=0&luki=1&seller_id=1&di=128)/rc.d/rc.local(本地运行服务)–>执行/bin/login,就可以登录了。

（6）保存当前磁盘分区的分区表

dd 命令（diskcopy）：指定大小的块拷贝一个文件，在复制的同时进行转换。

dd if=/dev/sda   of=./[mbr](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http://zhangge.net/1986.html&p=baidu&c=news&n=10&t=tpclicked3_hc&q=31072150_cpr&k=mbr&k0=mbr&k1=etc&k2=%BF%CD%BB%A7%B6%CB&k3=%B7%FE%CE%F1%C6%F7&k4=%B4%C5%C5%CC&k5=tcp&sid=c3a8a398a4cec66a&ch=0&tu=u1682464&jk=c8fabfe794451dcc&cf=29&fv=11&stid=9&urlid=0&luki=10&seller_id=1&di=128).txt  bs=1 count=512

（7）符号链接与硬链接的区别

我们可以把符号链接，也就是软连接 当做是 windows系统里的 快捷方式。

硬链接 就好像是 又复制了一份.

ln 3.txt 4.txt   这是硬链接，相当于复制，不可以跨分区，但修改3,4会跟着变，若删除3,4不受任何影响。

ln -s 3.txt 4.txt  这是软连接，相当于快捷方式。修改4,3也会跟着变，若删除3,4就坏掉了。不可以用了。

（8）如何查看二进制文件的内容

我们一般通过hexdump命令 来查看二进制文件的内容。

（9）在1-39内取随机数

echo $[$RAMDOM %40] ，RANDOM 随机数，%39 取余数

（10）显示/etc/inittab中以#开头，且后面跟了一个或者多个空白字符，而后又跟了任意非空白字符的行

grep "^# \{1,\} [^  ]"  /etc/initab

（11）显示/etc/inittab中包含了:一个数字:(即两个冒号中间一个数字)的行

grep “\:[0-9]\{1\}}\:” /etc/inittab   #{1}确定前面的匹配一次

（12）怎么把自己写的脚本添加到服务里面，即可以使用service命令来调用？

mv test /etc/init.d/    chmod +x /etc/init.d/test   chkconfig –add  test   service test start

写一个脚本，命名为test，最后chkconfig test on所有等级运行时都开启test服务

（13）1)第一个参数为URL，即可下载的文件；第二个参数为目录，即下载后保存的位置；2)如果用户给的目录不存在，则提示用户是否创建；如果创建就继续执行，否则，[函数](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http://zhangge.net/1986.html&p=baidu&c=news&n=10&t=tpclicked3_hc&q=31072150_cpr&k=%BA%AF%CA%FD&k0=%BA%AF%CA%FD&k1=%CE%C4%BC%FE%CF%B5%CD%B3&k2=mbr&k3=etc&k4=%BF%CD%BB%A7%B6%CB&k5=%B7%FE%CE%F1%C6%F7&sid=c3a8a398a4cec66a&ch=0&tu=u1682464&jk=c8fabfe794451dcc&cf=29&fv=11&stid=9&urlid=0&luki=8&seller_id=1&di=128)返回一个51的错误值给调用脚本；3)如果给的目录存在，则下载文件；下载命令执行结束后[**测试**](http://lib.csdn.net/base/softwaretest)文件下载成功与否；如果成功，则返回0给调用脚本，否则，返回52给调用脚本

#!/bin/bash

#function:download a onefile

url=$1

dir=$2

download()

{

[ ! -d $dir ] && read -p "no such file,do you want to creat it?（y/n）" answer

if[ $answer eq "y"|"Y" ];then

mkdir ./dir/

else

exit 51

fi

cd $dir  #进入该目录就能自动保存文件

wget $1>&/dev/null

[ $? eq 0 ]&&exit 0 || exit 52

}

download $url $dir  #调用此函数

[ $# -ne 2 ]&& echo "Usage:sh $0 URL,directory"

（14）、将文件file1的前20行、文件file2的最后15行合并成一个test？

head -20 file1>test && tail -15 file2 >> test

（15）、移动.txt文件，并按大小排序显示

#!/bin/bash

for file in `ls ./\*.txt`

do

mv $file /tmp/

done

ls -al /tmp/ | sort -nr -k5   key位置

（16）设置DNS、网关

echo "nameserver 202.16.53.68" >> /etc/resolv.conf

route add default gw 192.168.1.1

（17）比如两个文件file1和file2，删除两个文件中共同的部分，留下两个文件中独自有的部分

第一种：grep -v -f file1 file2 && grep -v -f file2 file1   -v表示反向匹配invert match，前一部分输出文件2中的内容，但是剔除包含在文件1中的内容，f指定范本文件，其内容有一个或多个范本样式，让grep查找符合范本条件的文件内容，格式为每一列的范本样式

第二种：awk '{print $0}' file1 file2|sort|uniq -u >1.txt表示依次读取file1 file2，打印出每行，NR是行号,$0代表整行。命令uniq，进行报告或者忽略重复的行，参数-u，表示只是打印出唯一的行

第三种：comm -3 file1 file2 默认输出有三列：只在file1中有的行、只在file2中有的行、在file1和file2中共有的行。有参数-1 -2 -3，分别来抑制输出对应的列，解决一切疑难杂症

（18）获取远程服务器的主机名

ssh 192.168.1.1 cat /etc/sysconfig/network |awk -F “=”’/HOSTNAME/ {print $2}’

（19）删除指定时间之前的文件

第一种：location=”/root/sqlback”,find $location -mtime +30 -type f |xrags rm -f删除30天前的东西。再加入定时任务文件crontab -e :10 4 1 \* \* /bin/bash deletesqlback.sh

第二种：find /root/sqlback/ -mtime +30 -type f -name \*.gz -exec rm -f {} \;

（20）取出两个文件的并集(重复的行只保留一份)  
cat file1 file2 | sort | uniq  
取出两个文件的交集(只留下同时存在于两个文件中的文件)  
cat file1 file2 | sort | uniq -d（repeated只输出重复的行）  
删除交集，留下其他的行  
cat file1 file2 | sort | uniq –u(uniq只显示唯一的行)

（21）通过apache访问日志access.log 统计IP和每个地址访问的次数，按访问量列出前10名。

cat /var/log/httpd/access\_log |awk ‘{print $1}’|sort -r |head -10 |uniq -c

uniq  -c :uniq本身具有输出功能，并且-c选项对输出结果相同行的次数统计

（22）、编写IPTABLES使用内网某台机器的80端口可以在公网访问，假设公网IP为10.10.1.1 ，实现192.168.1.0/24段的NAT？

iptables -t nat  -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j SNAT --to 10.10.1.1

（  1.PREROUTING:可以在这里定义进行目的NAT的规则，因为路由器进行路由时只检查数据包的目的ip地址，所以为了使数据包得以正确路由，我们必须在路由之前就进行目的NAT; 2.POSTROUTING:可以在这里定义进行源NAT的规则，系统在决定了数据包的路由以后在执行该链中的规则。）

44、你觉得服务器通常会出现哪些故障，请试着举例说出解决思路？

答：（1）系统无法正常启动

a、grub损坏（MBR损坏、grub配置文件丢失）

b、某文件系统无法正常挂载、驱动不兼容

c、因为某个自启服务故障，从而导致无法启动系统

d、用户无法登陆系统（bash程序损坏）

e、开机发出滋滋滋的响声（内存条接触不好，硬盘扇区损坏）

（2）机器频繁重启、风扇故障、电源不供电

（3）外网无法访问自己（涉及到路由交换、DNS、物理线路、防火墙策略问题）

解决思路：①观察服务器故障现象，注意报错消息②查看相应的系统日志，排障由广到深，

逐步缩小范围③理解故障发生的原因最好的办法是人为使其重现故障，涉及到修改文件时，必须备份原文件，或者将这个故障机器的文件与正常运作的机器的文件进行对比，最好还原为缺省配置。平时做的准备工作也很重要，不要经常更新为好，备份工作的习惯更是要深入到灵魂，条件反射。

45、你能说运维工程师主要做什么工作吗？

答：一是网络基础设施:保障物理上的安全，如机房、服务器、路由交换设备、用户PC正常运作;二是网络操作系统:防漏洞、防入侵、使用权限、密钥管理;三是网络服务：用户身份认证、安装配置、排障；四是用户操作使用培训教育；五是制定完善的网络运营维护文档、日志管理；六是系统备份、数据灾难恢复机制，这也是最难的一点。

46、有多台服务器需部署相同应用文件，文件会持续更新，你用什么方式实现不同服务器间的文件同步？

答：这个问题很常见，可是又没有很完美的解决方案。最直观的想法是写入第三方云平台，只要联网，知道用户密码，就能实现多点登录这个账号；第二个方案，参考一下DNS的主域名服务器和辅助域名服务器的区域复制功能，能不能有所想法。第三个方案，我想这个同步问题早就被人研究了，一定有其他大公司开发出了同步备份系统软件，我们只需要去搜一下就可以去使用了，可能要花点租金，但是值得。比如filegee企业文件同步备份系统、rsync软件。

47、64位和32位操作系统有何不同？

答：（1）很明显，运算速度不同，即CPU处理能力64位比32位高一倍

（2）64位操作系统只能安装在64位的计算机上，而32位操作系统既能安装在32位计算机，又能安装在64位的计算机上

（3）计算机的地址位宽会影响虚拟内存的大小和寻址能力

\* 如果要对32位系统进行扩展内存，使用PAE物理地址扩展技

48、chroot /mnt/sysimage做何理解？有何作用？

答：将/mnt/sysimage作为根目录，原有系统的根目录都被挂载到了/mnt/sysimage目录下。在经过 chroot 之后，在新根下将访问不到旧系统的根目录结构和文件，这样就增强了系统的安全性。chroot 的作用就是切换系统的根位置，而这个作用最为明显的是在系统初始引导磁盘的处理过程中使用，从初始 RAM 磁盘 (initrd) 切换系统的根位置并执行真正的 init。另外，当系统出现一些问题时，我们也可以使用 chroot 来切换到一个临时的内存系统。

49、Nginx相对于Apache优点:

1)高并发响应性能非常好，官方Nginx处理静态文件并发5w/s

2)反向代理性能非常好。(可用于负载均衡)

3)内存和cpu占用率低。(为Apache的1/5-1/10。)

4)功能较Apache少(常用功能均有)

5)对[**php**](http://lib.csdn.net/base/php)可使用cgi方式和fastcgi方式。

50、cgi和fastgcgi的区别？

答：首先CGI是用来web服务器进程和动态网页程序交互的一种协议，fastcgi和其原理一样，不同的是fastcgi处理完一个请求后不会终止整个进程，而是出于挂起状态等待下一个请求的到来，这样它就避免了像cgi那样频繁创建进程，从而提高它的访问效率。

51、多进程与多线程，请问有什么区别？

答：（1）进程：子进程是父进程的复制品。  
（2）线程：一个进程可以容纳多个线程，相对进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享资源，但拥有自己的栈空间，拥有独立的执行序列。两者都可以提高程序的并发度，提高程序运行效率和响应时间。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。根本区别就一点：用多进程每个进程有自己的地址空间(address space)，线程则共享地址空间。所有其它区别都是由此而来的：  
（1）速度：线程产生的速度快，线程间的通讯快、切换快等，因为他们在同一个地址空间内。（2）资源利用率：线程的资源利用率比较好也是因为他们在同一个地址空间内。  
（3）同步问题：线程使用公共变量/内存时需要使用同步机制还是因为他们在同一个地址空间内。

52、time\_filename.txt 改成filename\_time.txt。例如20111111\_me.txt改成me\_201111111.txt要如何修改？

答：#! /bin/sh

for eachfile in `ls -l`

do

 filename=${eachfile%.txt}

 filehead=`echo $filename | awk -F \_ '{print $1 }'`

 filelast=`echo $filename | awk -F \_ '{print $2 }'`

 mv $filename.txt ${filelast}\_$filehead.txt

done

53、线程的优缺点是什么？什么情况不用线程？什么情况使用线程提高效率？

答：优点：相比进程，线程是一个更接近执行体的概念，一个进程内可以拥有多个线程，他们可以并发运行，共享地址空间，而且线程产生的速度快，线程调度之间切换也快，那么因为共享地址空间体现的优点是：系统开销小，资源率利用比较高。

缺点：线程之间的同步机制非常繁琐，（互斥锁、读写锁、条件变量、线程信号），线程之间也会发生死锁，一个线程崩溃的时候  ，会牵连到其他线程。

使用线程的情况：线程适合在对称多处理机上运行

54、buffer和cache有何区别？

答：缓存是用来加速从硬盘中读取数据的，一个程序读取了一个数据，先放在缓存里，下一个程序再来需要的时候，直接调用缓存即可，因为很显然内存的存取速度要大于硬盘嘛，而缓冲是用来加速向硬盘写入数据的，我保存数据到硬盘，不是立刻生效，而是在内存缓冲积累到一定程度后，再全部放入硬盘，避免硬盘被频繁操作，从而影响系统的运行情况。他们二者本质都是内存，不能被进程调用，只能被Linux内核使用。

55、 有一web服务器，一天某用户投诉上某个网站速度很慢，如果你是服务器的管理员，请问 你如何查找原因？

答：首先我自己要试一下访问这个网站的速度如何？从而判断是客户端那边有问题还是服务端有问题，因为客户端涉及到网络拥塞和DNS解析的问题。那么我主要负责服务端这边的问题，如果我这边也不行的话，先从这台主机的系统本身故障入手，先看日志/var/log/messages，如果日志没什么反应的话，我再去看web服务日志的情况，看是不是有恶意访问，比如DDOS攻击，造成正常用户无法得到应答，如果这边也没问题的话，就要涉及到PHP响应慢和MySQL响应慢的问题了。

## 七 Java优化相关

### 7.1 JVM

#### 1、jvm的相关概述

当List放了大量的数据超过jvm中所能容纳的内存后，就会发生堆溢出。

当递归调用没有临界退出条件就会出现栈溢出。

当批量导入大量数据或者用dom4j解析大的xml文件的时候，

会出现 堆溢出，这个时候可以通过分段批量提交以及用

sax代替dom4j来解决问题。

Java 虚拟机有一个堆，堆是运行时数据区域，

所有类实例和数组的内存均从此处分配。

堆是在 Java 虚拟机启动时创建的。”

“在JVM中堆之外的内存称为非堆内存(Non-heap memory)”。

可以看出JVM主要管理两种类型的内存：堆和非堆。

简单来说堆就是Java代码可及的内存，是留给开发人员使用的；

非堆就是JVM留给自己用的。

#### 2、jvm的内存结构

堆： 逻辑上是连续,物理上可以处于不连续的内存空间中，

里面存储的是对象实例以及数组。可以细分为新生代，老生代。

通过-Xmx和-Xms控制大小。

虚拟机栈：基本数据类型，对象引用(地址，指针)。

本地方法栈（了解）:它与虚拟机栈发挥的作用差不多，区别在于虚拟机栈为java方法

的执行提供服务，而本地方法栈为虚拟机使用到的Native(本地)方法服务。

方法区：放了所加载的类的信息（名称、修饰符等）、类中的静态变量、

类中定义为final类型的常量、类中的Field信息、类中的方法信息

在Sun JDK中这块区域对应的为PermanetGeneration，又称为持久代，

默认为64M，可通过-XX:PermSize以及-XX:MaxPermSize来指定其大小

在服务器启动的时候报内存溢出是因为方法区太小，也就相当于持久代的内存太小。

通过-XX:PermSize以及-XX:MaxPermSize来指定其大小，可以解决这个问题。

常量池是方法区的一部分,用来存储常量信息。如String就存储在常量池中。

计数器（了解）:通过该计数器的值来选取下一条要执行的字节码指令。

#### 3、存泄露和内存溢出

内存泄露 (memory leak)，是指应用程序在申请内存后，

无法释放已经申请的内存空间.一次内存泄露危害可以忽略，

但如果任其发展最终会导致内存溢出(out of memory).

如读取文件后流要进行及时的关闭以及对数据库连接的释放。

内存溢出（out of memory）是指应用程序在申请内存时，

没有足够的内存空间供其使用。

如我们在项目中对于大批量数据的导入，采用分段批量提交的方式。

#### 4、GC是什么？为什么要有GC？

答：GC是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。Java程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或Runtime.getRuntime().gc() ，但JVM可以屏蔽掉显示的垃圾回收调用。

垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在Java诞生初期，垃圾回收是Java最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今Java的垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得iOS的系统比[**Android**](http://lib.csdn.net/base/15)系统有更好的用户体验，其中一个深层次的原因就在于Android系统中垃圾回收的不可预知性。

补充：垃圾回收机制有很多种，包括：分代复制垃圾回收、标记垃圾回收、增量垃圾回收等方式。标准的Java进程既有栈又有堆。栈保存了原始型局部变量，堆保存了要创建的对象。Java平台对堆内存回收和再利用的基本算法被称为标记和清除，但是Java对其进行了改进，采用“分代式垃圾收集”。这种方法会跟Java对象的生命周期将堆内存划分为不同的区域，在垃圾收集过程中，可能会将对象移动到不同区域：

（1）伊甸园（Eden）：这是对象最初诞生的区域，并且对大多数对象来说，这里是它们唯一存在过的区域。

（2）幸存者乐园（Survivor）：从伊甸园幸存下来的对象会被挪到这里。

（3）终身颐养园（Tenured）：这是足够老的幸存对象的归宿。年轻代收集（Minor-GC）过程是不会触及这个地方的。当年轻代收集不能把对象放进终身颐养园时，就会触发一次完全收集（Major-GC），这里可能还会牵扯到压缩，以便为大对象腾出足够的空间。

与垃圾回收相关的JVM参数：

* -Xms / -Xmx --- 堆的初始大小 / 堆的最大大小
* -Xmn --- 堆中年轻代的大小
* -XX:-DisableExplicitGC --- 让System.gc()不产生任何作用
* -XX:+PrintGCDetail --- 打印GC的细节
* -XX:+PrintGCDateStamps --- 打印GC操作的时间戳

#### 5、描述一下JVM加载class文件的原理机制?

JVM中类的装载是由ClassLoader和它的子类来实现的,Java ClassLoader 是一个重要的Java运行时系统组件。它负责在运行时查找和装入类文件的类。

#### 6、类的实例化顺序，比如父类静态数据，构造函数，字段，子类静态数据，构造函数，字段，他们的执行顺序

答：先静态、先父后子。

先静态：父静态 > 子静态

优先级：父类 > 子类 静态代码块 > 非静态代码块 > 构造函数

一个类的实例化过程：

1，父类中的static代码块，当前类的static

2，顺序执行父类的普通代码块

3，父类的构造函数

4，子类普通代码块

5，子类（当前类）的构造函数，按顺序执行。

6，子类方法的执行，

#### 7、Java 8的内存分代改进

从永久代到元空间，在小范围自动扩展永生代避免溢出

#### 8、JVM垃圾回收机制，何时触发MinorGC等操作

分代垃圾回收机制：不同的对象生命周期不同。把不同生命周期的对象放在不同代上，不同代上采用最合适它的垃圾回收方式进行回收。

JVM中共划分为三个代：年轻代、年老代和持久代，

年轻代：存放所有新生成的对象；

年老代：在年轻代中经历了N次垃圾回收仍然存活的对象，将被放到年老代中，故都是一些生命周期较长的对象；

持久代：用于存放静态文件，如Java类、方法等。

新生代的垃圾收集器命名为“minor gc”，老生代的GC命名为”Full Gc 或者Major GC”.其中用System.gc()强制执行的是Full Gc.

判断对象是否需要回收的方法有两种：

1.引用计数

当某对象的引用数为0时，便可以进行垃圾收集。

2.对象引用遍历

果某对象不能从这些根对象的一个（至少一个）到达，则将它作为垃圾收集。在对象遍历阶段，gc必须记住哪些对象可以到达，以便删除不可到达的对象，这称为标记（marking）对象。

触发GC（Garbage Collector）的条件：

1)GC在优先级最低的线程中运行，一般在应用程序空闲即没有应用线程在运行时被调用。

2)Java堆内存不足时，GC会被调用。

#### 9、jvm中一次完整的GC流程（从ygc到fgc）是怎样的，重点讲讲对象如何晋升到老年代等

答：对象优先在新生代区中分配，若没有足够空间，Minor GC；

大对象（需要大量连续内存空间）直接进入老年态；长期存活的对象进入老年态。如果对象在新生代出生并经过第一次MGC后仍然存活，年龄+1，若年龄超过一定限制（15），则被晋升到老年态。

#### 10、Eden和Survivor的比例分配等

默认比例8:1。

大部分对象都是朝生夕死。

复制算法的基本思想就是将内存分为两块，每次只用其中一块，当这一块内存用完，就将还活着的对象复制到另外一块上面。复制算法不会产生内存碎片。

#### 11、深入分析了Classloader，双亲委派机制

ClassLoader：类加载器（class loader）用来加载 Java 类到 Java 虚拟机中。Java 源程序（.java 文件）在经过 Java 编译器编译之后就被转换成 Java 字节代码（.class 文件）。类加载器负责读取 Java 字节代码，并转换成 java.lang.Class 类的一个实例。

双亲委派机制：某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父类加载器，依次递归，如果父类加载器可以完成类加载任务，就成功返回；只有父类加载器无法完成此加载任务时，才自己去加载。

#### 12、对Java内存模型的理解，以及其在并发中的应用

Java内存模型的主要目标: 定义程序中各个变量的访问规则。

Java线程之间的通信由Java内存模型（本文简称为JMM）控制。

所有变量的存储都在主内存，每条线程还都有自己的工作内存，线程的工作内存中保存了被该线程使用到的变量的主内存副本拷贝，线程对变量的所有操作必须在工作内存完成，而不能直接读取主内存中的变量。不同的线程直接无法访问对方工作内存中的变量，线程间变量的传递均需要通过主内存来完成。

这里写图片描述

#### 13、JVM常用参数

JVM主要参数：堆设置、回收器选择（串行、并行、并发收集器）

#### 14、volatile的语义，它修饰的变量一定线程安全吗

一个变量被定义为volatile之后，具备两重语义：①保证此变量对所有线程的可见性，即当一条线程修改了这个值，新值对于其他所有线程来说是立即得知的，普通变量需要通过主内存传递。②禁止指令重排序优化。

Volatile修饰的变量不一定是线程安全的，eg非原子操作a++等

#### 15、g1和cms区别,吞吐量优先和响应优先的垃圾收集器选择

CMS收集器：一款以获取最短回收停顿时间为目标的收集器，是基于“标记-清除”算法实现的，分为4个步骤：初始标记、并发标记、重新标记、并发清除。

G1收集器：面向服务端应用的垃圾收集器，过程：初始标记；并发标记；最终标记；筛选回收。整体上看是“标记-整理”，局部看是“复制”，不会产生内存碎片。

吞吐量优先的并行收集器：以到达一定的吞吐量为目标，适用于科学技术和后台处理等。

响应时间优先的并发收集器：保证系统的响应时间，减少垃圾收集时的停顿时间。适用于应用服务器、电信领域等。

#### 16、说一说你对环境变量classpath的理解？如果一个类不在classpath下，为什么会抛出ClassNotFoundException异常，如果在不改变这个类路径的前期下，怎样才能正确加载这个类？

classpath是javac编译器的一个环境变量。它的作用与import、package关键字有关。package的所在位置，就是设置CLASSPATH当编译器面对import packag这个语句时，它先会查找CLASSPATH所指定的目录，并检视子目录java/util是否存在，然后找出名称吻合的已编译文件（.class文件）。如果没有找到就会报错！

动态加载包

#### 17、说一下强引用、软引用、弱引用、虚引用以及他们之间和gc的关系

强引用：new出的对象之类的引用，

只要强引用还在，永远不会回收

软引用：引用但非必须的对象，内存溢出异常之前，回收

弱引用：非必须的对象，对象能生存到下一次垃圾收集发生之前。

虚引用：对生存时间无影响，在垃圾回收时得到通知。

### 7.2 JUC

#### 1、简述synchronized和java.util.concurrent.locks.Lock的异同？

主要相同点：Lock能完成synchronized所实现的所有功能

主要不同点：Lock有比synchronized更精确的线程语义和更好的性能。synchronized会自动释放锁，而Lock一定要求程序员手工释放，并且必须在finally从句中释放。

### 7.3 Tomcat高级

#### 1、tomcat结构，类加载器流程

目录结构：

? /bin：存放windows或Linux平台上启动和关闭Tomcat的脚本文件

? /conf：存放Tomcat服务器的各种全局配置文件，其中最重要的是server.xml和web.xml

? /doc：存放Tomcat文档

? /server：包含三个子目录：classes、lib和webapps

? /server/lib：存放Tomcat服务器所需的各种JAR文件

? /server/webapps：存放Tomcat自带的两个WEB应用admin应用和 manager应用

? /common/lib：存放Tomcat服务器以及所有web应用都可以访问的jar文件

? /shared/lib：存放所有web应用都可以访问的jar文件（但是不能被Tomcat服务器访问）

? /logs：存放Tomcat执行时的日志文件

? /src：存放Tomcat的源代码

? /webapps：Tomcat的主要Web发布目录，默认情况下把Web应用文件放于此目录

? /work：存放JSP编译后产生的class文件

类加载器模式，双亲委派模式：

#### 2、Tomcat优化

增大内存(堆，持久代)并开启server模式

我在做XXX项目时,用到了poi导入和导出数据,由于公司的业务比较繁多,数据量很大,测试时报内存溢出,经过我的分析再结合上网查阅资料,发现可能是tomcat内存不足,需要增大,修改配置文件后测试不再报错.

tomcat增大内存的方式通过修改tomcat配置文件

window下， 在bin/catalina.bat文件中最前面添加：

set JAVA\_OPTS=-XX:PermSize=64M -XX:MaxPermSize=128m –Xms1024m -Xmx1024m

linux下，在catalina.sh最前面增加：

JAVA\_OPTS="-XX:PermSize=64M -XX:MaxPermSize=128m –Xms1024m -Xmx1024m "

-client –service

当我们在cmd中运行-java时,黑窗口会出现-client -service这两参数.其作用是设置虚拟机运行模式;client模式启动比较快，但运行时性能和内存管理效率不如server模式，通常用于客户端应用程序。server模式启动比client慢，但可获得更高的运行性能。Windows默认为client，如果要使用server模式，就需要在启动虚拟机时加-server参数，以获得更高性能，对服务器端应用，推荐采用server模式，尤其是多个CPU的系统。在Linux，Solaris上,默认值为server模式.

JDK版本

影响虚拟机还有JDK的版本,JDK分为32位,64位两种版本,32位装在32位系统,64位系统可以装32位和64位JDK.64位JDK性能优于32位JDK.

测试的命令 java -xmx数值m –version 报错配置大小失败,反之成功

**增加Tomcat最大连接数**

使用场景

我在做完一个XXX项目后,测试时发现并发数量增加到一定程度就会很卡,于是我想到了是不是tomcat最大连接数设置有限制.果不其然,配置文件中最大值才500,于是我更改了最大连接数,根据业务我修改了连接数为2000,完美的解决了这个问题;

修改方法在conf/service.xml中默认值

<Connector port="8080" maxHttpHeaderSize="8192" maxThreads="1500"

minSpareThreads="30" maxSpareThreads="75" enableLookups="false"

redirectPort="8443" acceptCount="100" connectionTimeout="20000"

disableUploadTimeout="true" />,修改maxthreads的值即可

tomcat进行gzip压缩从而降低网络传输量

tomcat 压缩设置tomcat压缩gzip启用

HTTP 压缩可以大大提高浏览网站的速度，它的原理是，

在客户端请求服务器对应资源后，从服务器端将资源文件压缩，

再输出到客户端，由客户端的浏览器负责解压缩并浏览。

相对于普通的浏览过程HTML ,CSS,Javascript , Text ，

它可以节省60%左右的流量。更为重要的是，它可以对动态生成的，

包括CGI、PHP , JSP , ASP , Servlet,SHTML等输出的网页也能进行压缩，

压缩效率也很高。

**启用tomcat 的gzip压缩**

要使用gzip压缩功能，你需要在Connector节点中加上如下属性

记住来源：http://www.qi788.com/info-42.html

compression="on" 打开压缩功能

compressionMinSize="50" 启用压缩的输出内容大小，默认为2KB

noCompressionUserAgents="gozilla, traviata" 对于以下的浏览器，不启用压缩

compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"　哪些资源类型需要压缩

<Connector port="80" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" executor="tomcatThreadPool" URIEncoding="utf-8"

compression="on"

compressionMinSize="50" noCompressionUserAgents="gozilla, traviata"

compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain" />

### 7.4 mysql 高级

#### 1、sql优化

1、SELECT子句中避免使用 \*， 尽量应该根据业务需求按字段进行查询

2、尽量多使用COMMIT如对大数据量的分段批量提交释放了资源，减轻了服务器压力

3、在写sql语句的话，尽量保持每次查询的sql语句字段用大写，因为oracle总是先解析 sql语句，把小写的字母转换成大写的再执行

4、用UNION-ALL 替换UNION，因为UNION-ALL不会过滤重复数据，所执行效率 要快于UNION,并且UNION可以自动排序，而UNION-ALL不会

5、避免在索引列上使用计算和函数,这样索引就不能使用

**Sql优化精简版：**

1.(重点)(必须说) SELECT语句中避免使用 \*，

尽量应该根据业务需求按字段进行查询

举例：如果表中有个字段用的是clob或者是blob这种大数据字段的话，

他们的查询应该根据业务需要来进行指定字段的查询，切记勿直接用\*

2.(重点) 删除重复记录(oracle)：

最高效的删除重复记录方法 ( 因为使用了ROWID)例子：

DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID)

FROM EMP X WHERE X.EMP\_NO = E.EMP\_NO);

3. 用>=替换>

如一个表有100万记录，一个数值型字段A，

A=0时，有30万条；

A=1时，有30万条；

A=2时，有39万条；

A=3时，有1万记录。

那么执行 A>2 与 A>=3 的效果就有很大的区别了，因为 A>2 时，

ORACLE会先找出为2的记录索引再进行比较，

而A>=3时ORACLE则直接找到=3的记录索引。

4.(重点)尽量多使用COMMIT

如对大数据量的分段批量提交

5. (重点)用NOT EXISTS 或（外连接+判断为空）方案 替换 NOT IN操作符

此操作是强列推荐不使用的，因为它不能应用表的索引。

推荐方案：用NOT EXISTS 或（外连接+判断为空）方案代替

6.(重点 必须说)LIKE操作符(大数据的全文检索使用luncene)(solr)

因为使用like不当，会导致性能问题，原因是like在左右两边都有

%的时候，不会使用索引。

如LIKE '%5400%' 这种查询不会引用索引，

而LIKE 'X5400%' 则会引用范围索引。

一个实际例子：

查询营业编号 YY\_BH LIKE '%5400%' 这个条件会产生全表扫描，

如果改成 YY\_BH LIKE 'X5400%' OR YY\_BH LIKE 'B5400%'

则会利用 YY\_BH 的索引进行两个范围的查询，性能肯定大大提高。

7.(重点,必须说)避免在索引列上使用计算和函数,这样索引就不能使用

举例:

低效：

SELECT … FROM DEPT WHERE SAL \* 12 > 25000;

高效:

SELECT … FROM DEPT WHERE SAL > 25000/12;

8.(重点 必须说)用UNION-ALL 替换UNION，

因为UNION-ALL不会过滤重复数据而且不会自动排序，

所执行效率要快于UNION。

9. （优化,重点,3个方面 a.缓存 b.分段批量 c.存储过程）减少访问数据库的次数

举例:如果批量删除多条数据，可以用 delete from tableName where id

in (1,2,3)

而不要用多条delete语句进行删除

10.（重点 必须说）用TRUNCATE替代DELETE

TRUNCATE不记录日志，DELETE记录日志，所以TRUNCATE要快于DELETE

但是一旦用TRUNCATE进行删除就不能进行恢复,TRUNCATE是删除整张表的数据

不能加where条件。

==================================================================

mysql,sqlserver中如果

id为自增类型，那么如果用TRUNCATE删除，则id字段再插入数据时从1开始，

如果delete删除的话，则从删除之前的id的值继续增长。

**(1)、explain出来的各种item的意义；**

select\_type

表示查询中每个select子句的类型

type

表示MySQL在表中找到所需行的方式，又称“访问类型”

possible\_keys

指出MySQL能使用哪个索引在表中找到行，查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询使用

key

显示MySQL在查询中实际使用的索引，若没有使用索引，显示为NULL

key\_len

表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度

ref

表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值

Extra

包含不适合在其他列中显示但十分重要的额外信息

**(2)、profile的意义以及使用场景；**

查询到 SQL 会执行多少时间, 并看出 CPU/Memory 使用量, 执行过程中 Systemlock, Table lock 花多少时间等等

#### 2、防sql注入

针对防sql注入，我们通常是这样做的：

首先在前台页面对用户输入信息进行js验证，对一些特殊字符进行屏蔽，

比如：or ,单引号，--，= ，还有就是限制用户名输入的长度，我们一般将其限制在6---13位。另外，对于用户的敏感信息我们进行Md5加密，还有，为了增加用户体验度和用户友好度，为了不使用户看到一些详细的异常信息我们会进行错误信息页面的定制，像404,500错误。另一个我层面讲，这样做也是为了保护我们的一些重要信息。此外，我们会给特定的人分配定定的权限，而不是给其分配管理员权限！

**sql注入**

所谓SQL注入，就是通过一些含有特殊字符的sql语句发送到服务器欺骗服务器并进行攻击。（特殊字符：or, 单引号，--，空格）

**Sql注入的防护**

1.永远不要信任用户的输入。对用户的输入进行校验，可以通过正则表达式（js正则或者java后台正则），或限制长度；对单引号和双"-"进行转换等。

2.永远不要使用动态拼装sql，使用参数化的sql。（永远不要使用+号拼接sql字符串，而是使用？传参的方式进行）

3.不要给用户太高的权限而根据需求进行赋权

4.对敏感信息进行加密 如md5(单向加密不可逆转)。

5.自定义错误页面。目的是为了不把我们的程序的bug暴露在别有用心的人的面前。而去不会让用户看到报错的页面，也提高了用户的体验度。

**SQL注入防范**

使用参数化的过滤性语句

　　要防御SQL注入，用户的输入就绝对不能直接被嵌入到SQL语句中。恰恰相反，用户的输入必须进行过滤，或者使用参数化的语句。参数化的语句使用参数而不是将用户输入嵌入到语句中。在多数情况中，SQL语句就得以修正。然后，用户输入就被限于一个参数。

**输入验证**

　　检查用户输入的合法性，确信输入的内容只包含合法的数据。数据检查应当在客户端和服务器端（java代码）都执行之所以要执行服务器端验证，是为了弥补客户端验证机制脆弱的安全性。

　　在客户端，攻击者完全有可能获得网页的源代码，修改验证合法性的脚本（或者直接删除脚本），然后将非法内容通过修改后的表单提交给服务器。因此，要保证验证操作确实已经执行，唯一的办法就是在服务器端也执行验证。你可以使用许多内建的验证对象，例如Regular Expression Validator，它们能够自动生成验证用的客户端脚本，当然你也可以插入服务器端的方法调用。如果找不到现成的验证对象，你可以通过Custom Validator自己创建一个。

**错误消息处理**

　　防范SQL注入，还要避免出现一些详细的错误消息，因为黑客们可以利用这些消息。要使用一种标准的输入确认机制来验证所有的输入数据的长度、类型、语句、企业规则等。

加密处理

将用户登录名称、密码等数据加密保存。加密用户输入的数据，然后再将它与数据库中保存的数据比较，这相当于对用户输入的数据进行了“消毒”处理，用户输入的数据不再对数据库有任何特殊的意义，从而也就防止了攻击者注入SQL命令。

存储过程来执行所有的查询

　　SQL参数的传递方式将防止攻击者利用单引号和连字符实施攻击。此外，它还使得数据库权限可以限制到只允许特定的存储过程执行，所有的用户输入必须遵从被调用的存储过程的安全上下文，这样就很难再发生注入式攻击了。

使用专业的漏洞扫描工具

　　攻击者们目前正在自动搜索攻击目标并实施攻击，其技术甚至可以轻易地被应用于其它的Web架构中的漏洞。企业应当投资于一些专业的漏洞扫描工具，如大名鼎鼎的Acunetix的Web漏洞扫描程序等。一个完善的漏洞扫描程序不同于网络扫描程序，它专门查找网站上的SQL注入式漏洞。最新的漏洞扫描程序可以查找最新发现的漏洞。

**确保数据库安全**

　　锁定你的数据库的安全，只给访问数据库的web应用功能所需的最低的权限，撤销不必要的公共许可，使用强大的加密技术来保护敏感数据并维护审查跟踪。如果web应用不需要访问某些表，那么确认它没有访问这些表的权限。如果web应用只需要只读的权限，那么就禁止它对此表的 drop 、insert、update、delete 的权限，并确保数据库打了最新补丁。

**安全审评**

在部署应用系统前，始终要做安全审评。建立一个正式的安全过程，并且每次做更新时，要对所有的编码做审评。开发队伍在正式上线前会做很详细的安全审评，然后在几周或几个月之后他们做一些很小的更新时，他们会跳过安全审评这关， “就是一个小小的更新，我们以后再做编码审评好了”。请始终坚持做安全审评。

### 7.5 你用过的网站前端优化的技术有哪些？

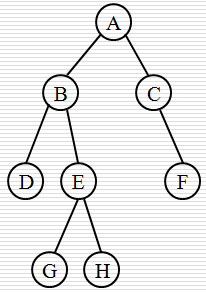
①浏览器访问优化：   
- 减少HTTP请求数量：合并CSS、合并[javascript](http://lib.csdn.net/base/javascript)、合并图片（CSS Sprite）   
- 使用浏览器缓存：通过设置HTTP响应头中的Cache-Control和Expires属性，将CSS、JavaScript、图片等在浏览器中缓存，当这些静态资源需要更新时，可以更新HTML文件中的引用来让浏览器重新请求新的资源   
- 启用压缩   
- CSS前置，JavaScript后置  
- 减少Cookie传输  
② CDN加速：CDN（ContentDistribute Network）的本质仍然是缓存，将数据缓存在离用户最近的地方，CDN通常部署在网络运营商的机房，不仅可以提升响应速度，还可以减少应用服务器的压力。当然，CDN缓存的通常都是静态资源。   
③反向代理：反向代理相当于应用服务器的一个门面，可以保护网站的安全性，也可以实现负载均衡的功能，当然最重要的是它缓存了用户访问的热点资源，可以直接从反向代理将某些内容返回给用户浏览器。

### 7.6 你使用过的应用服务器优化技术有哪些？

①分布式缓存：缓存的本质就是内存中的哈希表，如果设计一个优质的哈希函数，那么理论上哈希表读写的渐近时间复杂度为O(1)。缓存主要用来存放那些读写比很高、变化很少的数据，这样应用程序读取数据时先到缓存中读取，如果没有或者数据已经失效再去访问数据库或文件系统，并根据拟定的规则将数据写入缓存。对网站数据的访问也符合二八定律（Pareto分布，幂律分布），即80%的访问都集中在20%的数据上，如果能够将这20%的数据缓存起来，那么系统的性能将得到显著的改善。当然，使用缓存需要解决以下几个问题：   
- 频繁修改的数据；   
- 数据不一致与脏读；   
- 缓存雪崩（可以采用分布式缓存服务器集群加以解决，[memcached](http://memcached.org)是广泛采用的解决方案）；   
- 缓存预热；   
- 缓存穿透（恶意持续请求不存在的数据）。   
②异步操作：可以使用消息队列将调用异步化，通过异步处理将短时间高并发产生的事件消息存储在消息队列中，从而起到削峰作用。电商网站在进行促销活动时，可以将用户的订单请求存入消息队列，这样可以抵御大量的并发订单请求对系统和数据库的冲击。目前，绝大多数的电商网站即便不进行促销活动，订单系统都采用了消息队列来处理。   
③使用集群。  
④代码优化：  
- 多线程：基于Java的Web开发基本上都通过多线程的方式响应用户的并发请求，使用多线程技术在编程上要解决线程安全问题，主要可以考虑以下几个方面：A. 将对象设计为无状态对象（这和面向对象的编程观点是矛盾的，在面向对象的世界中被视为不良设计），这样就不会存在并发访问时对象状态不一致的问题。B. 在方法内部创建对象，这样对象由进入方法的线程创建，不会出现多个线程访问同一对象的问题。使用ThreadLocal将对象与线程绑定也是很好的做法，这一点在前面已经探讨过了。C. 对资源进行并发访问时应当使用合理的锁机制。   
- 非阻塞I/O：使用单线程和非阻塞I/O是目前公认的比多线程的方式更能充分发挥服务器性能的应用模式，基于[Node.js](http://lib.csdn.net/base/nodejs)构建的服务器就采用了这样的方式。Java在JDK 1.4中就引入了NIO（Non-blockingI/O）,在Servlet3规范中又引入了异步Servlet的概念，这些都为在服务器端采用非阻塞I/O提供了必要的基础。   
- 资源复用：资源复用主要有两种方式，一是单例，二是对象池，我们使用的数据库连接池、线程池都是对象池化技术，这是典型的用空间换取时间的策略，另一方面也实现对资源的复用，从而避免了不必要的创建和释放资源所带来的开销。

## 八 算法与数据结构

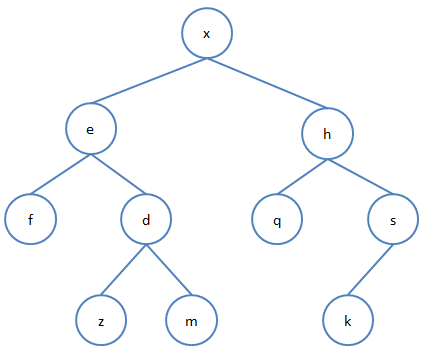
#### 1、给出下面的二叉树先序、中序、后序遍历的序列？



答：先序序列：ABDEGHCF；中序序列：DBGEHACF；后序序列：DGHEBFCA。

补充：二叉树也称为二分树，它是树形结构的一种，其特点是每个结点至多有二棵子树，并且二叉树的子树有左右之分，其次序不能任意颠倒。二叉树的遍历序列按照访问根节点的顺序分为先序（先访问根节点，接下来先序访问左子树，再先序访问右子树）、中序（先中序访问左子树，然后访问根节点，最后中序访问右子树）和后序（先后序访问左子树，再后序访问右子树，最后访问根节点）。如果知道一棵二叉树的先序和中序序列或者中序和后序序列，那么也可以还原出该二叉树。

例如，已知二叉树的先序序列为：xefdzmhqsk，中序序列为：fezdmxqhks，那么还原出该二叉树应该如下图所示：



#### 2、你知道的排序算法都哪些？用Java写一个排序系统。

答：稳定的排序算法有：插入排序、选择排序、冒泡排序、鸡尾酒排序、归并排序、二叉树排序、基数排序等；不稳定排序算法包括：希尔排序、堆排序、快速排序等。

下面是关于排序算法的一个列表：



下面按照策略模式给出一个排序系统，实现了冒泡、归并和快速排序。

Sorter.java

|  |
| --- |
| package com.jackfrued.util;  import java.util.Comparator;  /\*\*  \* 排序器接口(策略模式: 将算法封装到具有共同接口的独立的类中使得它们可以相互替换)  \* @author骆昊  \*  \*/  public interface Sorter {    /\*\*  \* 排序  \* @param list 待排序的数组  \*/  public <T extends Comparable<T>> void sort(T[] list);    /\*\*  \* 排序  \* @param list 待排序的数组  \* @param comp 比较两个对象的比较器  \*/  public <T> void sort(T[] list, Comparator<T> comp);  } |

BubbleSorter.java

|  |
| --- |
| package com.jackfrued.util;  import java.util.Comparator;    /\*\*  \* 冒泡排序  \* @author骆昊  \*  \*/  public class BubbleSorter implements Sorter {    @Override  public <T extends Comparable<T>> void sort(T[] list) {  boolean swapped = true;  for(int i = 1; i < list.length && swapped;i++) {  swapped= false;  for(int j = 0; j < list.length - i; j++) {  if(list[j].compareTo(list[j+ 1]) > 0 ) {  T temp = list[j];  list[j]= list[j + 1];  list[j+ 1] = temp;  swapped= true;  }  }  }  }    @Override  public <T> void sort(T[] list,Comparator<T> comp) {  boolean swapped = true;  for(int i = 1; i < list.length && swapped; i++) {  swapped = false;  for(int j = 0; j < list.length - i; j++) {  if(comp.compare(list[j], list[j + 1]) > 0 ) {  T temp = list[j];  list[j]= list[j + 1];  list[j+ 1] = temp;  swapped= true;  }  }  }  }  } |

MergeSorter.java

|  |
| --- |
| package com.jackfrued.util;    import java.util.Comparator;    /\*\*  \* 归并排序  \* 归并排序是建立在归并操作上的一种有效的排序算法。  \* 该算法是采用分治法（divide-and-conquer）的一个非常典型的应用，  \* 先将待排序的序列划分成一个一个的元素，再进行两两归并，  \* 在归并的过程中保持归并之后的序列仍然有序。  \* @author骆昊  \*  \*/  public class MergeSorter implements Sorter {    @Override  public <T extends Comparable<T>> void sort(T[] list) {  T[] temp = (T[]) new Comparable[list.length];  mSort(list,temp, 0, list.length- 1);  }    private <T extends Comparable<T>> void mSort(T[] list, T[] temp, int low, int high) {  if(low == high) {  return ;  }  else {  int mid = low + ((high -low) >> 1);  mSort(list,temp, low, mid);  mSort(list,temp, mid + 1, high);  merge(list,temp, low, mid + 1, high);  }  }    private <T extends Comparable<T>> void merge(T[] list, T[] temp, int left, int right, int last) {  int j = 0;  int lowIndex = left;  int mid = right - 1;  int n = last - lowIndex + 1;  while (left <= mid && right <= last){  if (list[left].compareTo(list[right]) < 0){  temp[j++] = list[left++];  } else {  temp[j++] = list[right++];  }  }  while (left <= mid) {  temp[j++] = list[left++];  }  while (right <= last) {  temp[j++] = list[right++];  }  for (j = 0; j < n; j++) {  list[lowIndex + j] = temp[j];  }  }    @Override  public <T> void sort(T[] list, Comparator<T> comp) {  T[]temp = (T[])new Comparable[list.length];  mSort(list,temp, 0, list.length- 1, comp);  }    private <T> void mSort(T[] list, T[] temp, int low, int high, Comparator<T> comp) {  if(low == high) {  return ;  }  else {  int mid = low + ((high -low) >> 1);  mSort(list,temp, low, mid, comp);  mSort(list,temp, mid + 1, high, comp);  merge(list,temp, low, mid + 1, high, comp);  }  }    private <T> void merge(T[] list, T[]temp, int left, int right, int last, Comparator<T> comp) {  int j = 0;  int lowIndex = left;  int mid = right - 1;  int n = last - lowIndex + 1;  while (left <= mid && right <= last){  if (comp.compare(list[left], list[right]) <0) {  temp[j++] = list[left++];  } else {  temp[j++] = list[right++];  }  }  while (left <= mid) {  temp[j++] = list[left++];  }  while (right <= last) {  temp[j++] = list[right++];  }  for (j = 0; j < n; j++) {  list[lowIndex + j] = temp[j];  }  }  } |

QuickSorter.java

|  |
| --- |
| package com.jackfrued.util;  import java.util.Comparator;    /\*\*  \* 快速排序  \* 快速排序是使用分治法（divide-and-conquer）依选定的枢轴  \* 将待排序序列划分成两个子序列，其中一个子序列的元素都小于枢轴，  \* 另一个子序列的元素都大于或等于枢轴，然后对子序列重复上面的方法，  \* 直到子序列中只有一个元素为止  \* @author Hao  \*  \*/  public class QuickSorter implements Sorter {    @Override  public <T extends Comparable<T>> void sort(T[] list) {  quickSort(list, 0, list.length- 1);  }    @Override  public <T> void sort(T[] list, Comparator<T> comp) {  quickSort(list, 0, list.length- 1, comp);  }    private <T extends Comparable<T>> void quickSort(T[] list, int first, int last) {  if (last > first) {  int pivotIndex = partition(list, first, last);  quickSort(list, first, pivotIndex - 1);  quickSort(list, pivotIndex, last);  }  }    private <T> void quickSort(T[] list, int first, int last,Comparator<T> comp) {  if (last > first) {  int pivotIndex = partition(list, first, last, comp);  quickSort(list, first, pivotIndex - 1, comp);  quickSort(list, pivotIndex, last, comp);  }  }    private <T extends Comparable<T>> int partition(T[] list, int first, int last) {  T pivot = list[first];  int low = first + 1;  int high = last;    while (high > low) {  while (low <= high && list[low].compareTo(pivot) <= 0) {  low++;  }  while (low <= high && list[high].compareTo(pivot) >= 0) {  high--;  }  if (high > low) {  T temp = list[high];  list[high]= list[low];  list[low]= temp;  }  }    while (high > first&& list[high].compareTo(pivot) >= 0) {  high--;  }  if (pivot.compareTo(list[high])> 0) {  list[first]= list[high];  list[high]= pivot;  return high;  }  else {  return low;  }  }    private <T> int partition(T[] list, int first, int last, Comparator<T> comp) {  T pivot = list[first];  int low = first + 1;  int high = last;    while (high > low) {  while (low <= high&& comp.compare(list[low], pivot) <= 0) {  low++;  }  while (low <= high&& comp.compare(list[high], pivot) >= 0) {  high--;  }  if (high > low) {  T temp = list[high];  list[high] = list[low];  list[low]= temp;  }  }    while (high > first&& comp.compare(list[high], pivot) >= 0) {  high--;  }  if (comp.compare(pivot,list[high]) > 0) {  list[first]= list[high];  list[high]= pivot;  return high;  }  else {  return low;  }  }  } |

#### 3、写一个二分查找（折半搜索）的算法。

答：折半搜索，也称二分查找算法、二分搜索，是一种在有序数组中查找某一特定元素的[搜索算法](http://zh.wikipedia.org/w/index.php?title=%E6%90%9C%E7%B4%A2%E7%AE%97%E6%B3%95&action=edit&redlink=1)。搜素过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜素过程结束；如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组为空，则代表找不到。这种搜索算法每一次比较都使搜索范围缩小一半。

|  |
| --- |
| package com.jackfrued.util;  import java.util.Comparator;    public class MyUtil {    public static <T extends Comparable<T>> int binarySearch(T[] x, T key) {  return binarySearch(x, 0, x.length- 1, key);  }    public static <T> int binarySearch(T[] x, T key, Comparator<T> comp) {  int low = 0;  int high = x.length - 1;  while (low <= high) {  int mid = (low + high) >>> 1;  int cmp = comp.compare(x[mid], key);  if (cmp < 0) {  low = mid + 1;  }  else if (cmp > 0) {  high = mid - 1;  }  else {  return mid;  }  }  return -1;  }    private static <T extends Comparable<T>> int binarySearch(T[] x, int low, int high, T key) {  if(low <= high) {  int mid = low + ((high -low) >> 1);  if(key.compareTo(x[mid]) == 0) {  return mid;  }  else if(key.compareTo(x[mid])< 0) {  return binarySearch(x,l ow, mid - 1, key);  }  else {  return binarySearch(x, mid + 1, high, key);  }  }  return -1;  }  } |

说明：两个版本一个用递归实现，一个用循环实现。需要注意的是计算中间位置时不应该使用(high+ low) / 2的方式，因为加法运算可能导致整数越界，这里应该使用一下三种方式之一：low+ (high – low) / 2或low + (high – low) >> 1或（low + high） >>> 1（注：>>>是逻辑右移，不带符号位的右移）

#### 4、统计一篇英文文章中单词个数。

答：

|  |
| --- |
| import java.io.FileReader;    public class WordCounting {    public static void main(String[] args) {  try(FileReader fr = new FileReader("a.txt")) {  int counter = 0;  boolean state = false;  int currentChar;  while((currentChar= fr.read()) != -1) {  if(currentChar== ' ' || currentChar == '\n'  || currentChar == '\t' || currentChar == '\r') {  state = false;  }  else if(!state) {  state = true;  counter++;  }  }  System.out.println(counter);  }  catch(Exceptione) {  e.printStackTrace();  }  }  } |

补充：这个程序可能有很多种写法，这里选择的是Dennis M. Ritchie和Brian W. Kernighan老师在他们不朽的著作《The C Programming Language》中给出的代码，向两位老师致敬。下面的代码也是如此。

#### 5、输入年月日，计算该日期是这一年的第几天。

答：

|  |
| --- |
| import java.util.Scanner;    public class DayCounting {    public static void main(String[] args) {  int[][] data = {  {31,28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},  {31,29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}  };  Scanner sc = newScanner(System.in);  System.out.print("请输入年月日(1980 11 28): ");  int year = sc.nextInt();  int month = sc.nextInt();  int date = sc.nextInt();  int[] daysOfMonth = data[(year % 4 == 0 && year % 100 != 0 || year % 400 == 0)?1 : 0];  int sum = 0;  for(int i = 0; i < month -1; i++) {  sum += daysOfMonth[i];  }  sum += date;  System.out.println(sum);  sc.close();  }  } |

#### 6、约瑟夫环：

15个基督教徒和15个非教徒在海上遇险，必须将其中一半的人投入海中，其余的人才能幸免于难，于是30个人围成一圈，从某一个人开始从1报数，报到9的人就扔进大海，他后面的人继续从1开始报数，重复上面的规则，直到剩下15个人为止。结果由于上帝的保佑，15个基督教徒最后都幸免于难，问原来这些人是怎么排列的，哪些位置是基督教徒，哪些位置是非教徒。

答：

|  |
| --- |
| public class Josephu {  private static final int DEAD\_NUM = 9;    public static void main(String[] args) {  boolean[] persons = new boolean[30];  for(int i = 0; i < persons.length; i++) {  persons[i] = true;  }    int counter = 0;  int claimNumber = 0;  int index = 0;  while(counter < 15) {  if(persons[index]) {  claimNumber++;  if(claimNumber == DEAD\_NUM) {  counter++;  claimNumber= 0;  persons[index]= false;  }  }  index++;  if(index >= persons.length) {  index= 0;  }  }  for(boolean p : persons) {  if(p) {  System.out.print("基");  }  else {  System.out.print("非");  }  }  }  } |

#### 7、回文素数：

所谓回文数就是顺着读和倒着读一样的数(例如：11，121，1991…)，回文素数就是既是回文数又是素数(只能被1和自身整除的数)的数。编程找出11～9999之间的回文素数。

答：

|  |
| --- |
| public class PalindromicPrimeNumber {    public static void main(String[] args) {  for(int i = 11; i <= 9999; i++) {  if(isPrime(i) && isPalindromic(i)) {  System.out.println(i);  }  }  }    public static boolean isPrime(int n) {  for(int i = 2; i <= Math.sqrt(n); i++) {  if(n % i == 0) {  return false;  }  }  return true;  }    public static boolean isPalindromic(int n) {  int temp = n;  int sum = 0;  while(temp > 0) {  sum= sum \* 10 + temp % 10;  temp/= 10;  }  return sum == n;  }  } |

#### 8、全排列：

给出五个数字12345的所有排列。

答：

|  |
| --- |
| public class FullPermutation {    public static void perm(int[] list) {  perm(list,0);  }    private static void perm(int[] list, int k) {  if (k == list.length) {  for (int i = 0; i < list.length; i++) {  System.out.print(list[i]);  }  System.out.println();  }else{  for (int i = k; i < list.length; i++) {  swap(list, k, i);  perm(list, k + 1);  swap(list, k, i);  }  }  }    private static void swap(int[] list, int pos1, int pos2) {  int temp = list[pos1];  list[pos1] = list[pos2];  list[pos2] = temp;  }    public static void main(String[] args) {  int[] x = {1, 2, 3, 4, 5};  perm(x);  }  } |

#### 9、对于一个有N个整数元素的一维数组，找出它的子数组（数组中下标连续的元素组成的数组）之和的最大值。

答：下面给出几个例子（最大子数组用粗体表示）：

1）数组：{ 1, -2, **3**,**5**, -3, 2 }，结果是：8

2）数组：{ 0, -2, **3**, **5**, **-1**, **2** }，结果是：9

3）数组：{ -9, **-2**,-3, -5, -3 }，结果是：-2

可以使用动态规划的思想求解：

|  |
| --- |
| public class MaxSum {    private static int max(int x, int y) {  return x > y? x: y;  }    public static int maxSum(int[] array) {  int n = array.length;  int[] start = new int[n];  int[] all = new int[n];  all[n - 1] = start[n - 1] = array[n - 1];  for(int i = n - 2; i >= 0;i--) {  start[i] = max(array[i], array[i] + start[i + 1]);  all[i] = max(start[i], all[i + 1]);  }  return all[0];  }    public static void main(String[] args) {  int[] x1 = { 1, -2, 3, 5,-3, 2 };  int[] x2 = { 0, -2, 3, 5,-1, 2 };  int[] x3 = { -9, -2, -3,-5, -3 };  System.out.println(maxSum(x1)); // 8  System.out.println(maxSum(x2)); // 9  System.out.println(maxSum(x3)); //-2  }  } |

#### 10、用递归实现字符串倒转

答：

|  |
| --- |
| public class StringReverse {    public static String reverse(String originStr) {  if(originStr == null || originStr.length()== 1) {  return originStr;  }  return reverse(originStr.substring(1))+ originStr.charAt(0);  }    public static void main(String[] args) {  System.out.println(reverse("hello"));  }  } |

#### 11、输入一个正整数，将其分解为素数的乘积。

答：

|  |
| --- |
| public class DecomposeInteger {    private static List<Integer> list = newArrayList<Integer>();    public static void main(String[] args) {  System.out.print("请输入一个数: ");  Scanner sc = newScanner(System.in);  int n = sc.nextInt();  decomposeNumber(n);  System.out.print(n + " = ");  for(int i = 0; i < list.size() - 1; i++) {  System.out.print(list.get(i) + " \* ");  }  System.out.println(list.get(list.size() - 1));  }    public static void decomposeNumber(int n) {  if(isPrime(n)) {  list.add(n);  list.add(1);  }  else {  doIt(n, (int)Math.sqrt(n));  }  }    public static void doIt(int n, int div) {  if(isPrime(div) && n % div == 0) {  list.add(div);  decomposeNumber(n / div);  }  else {  doIt(n, div - 1);  }  }    public static boolean isPrime(int n) {  for(int i = 2; i <= Math.sqrt(n);i++) {  if(n % i == 0) {  return false;  }  }  return true;  }  } |

#### 12、一个有n级的台阶，一次可以走1级、2级或3级，问走完n级台阶有多少种走法。

答：可以通过递归求解。

|  |
| --- |
| public class GoSteps {    public static int countWays(int n) {  if(n < 0) {  return 0;  }  else if(n == 0) {  return 1;  }  else {  return countWays(n - 1) + countWays(n - 2) + countWays(n -3);  }  }    public static void main(String[] args) {  System.out.println(countWays(5)); // 13  }  } |

#### 13、写一个算法判断一个英文单词的所有字母是否全都不同（不区分大小写）。

答：

|  |
| --- |
| public class AllNotTheSame {    public static boolean judge(String str) {  String temp = str.toLowerCase();  int[] letterCounter = new int[26];  for(int i = 0; i <temp.length(); i++) {  int index = temp.charAt(i)- 'a';  letterCounter[index]++;  if(letterCounter[index] > 1) {  return false;  }  }  return true;  }    public static void main(String[] args) {  System.out.println(judge("hello"));  System.out.print(judge("smile"));  }  } |

#### 14、有一个已经排好序的整数数组，其中存在重复元素，请将重复元素删除掉，例如，A= [1, 1, 2, 2, 3]，处理之后的数组应当为A= [1, 2, 3]。

答：

|  |
| --- |
| import java.util.Arrays;    public class RemoveDuplication {    public static int[] removeDuplicates(int a[]) {  if(a.length <= 1) {  return a;  }  int index = 0;  for(int i = 1; i < a.length; i++) {  if(a[index] != a[i]) {  a[++index] = a[i];  }  }  int[] b = new int[index + 1];  System.arraycopy(a, 0, b, 0, b.length);  return b;  }    public static void main(String[] args) {  int[] a = {1, 1, 2, 2, 3};  a = removeDuplicates(a);  System.out.println(Arrays.toString(a));  }  } |

#### 15、给一个数组，其中有一个重复元素占半数以上，找出这个元素。

答：

|  |
| --- |
| public class FindMost {    public static <T> T find(T[] x){  T temp = null;  for(int i = 0, nTimes = 0; i< x.length;i++) {  if(nTimes == 0) {  temp= x[i];  nTimes= 1;  }  else {  if(x[i].equals(temp)) {  nTimes++;  }  else {  nTimes--;  }  }  }  return temp;  }    public static void main(String[] args) {  String[]strs = {"hello","kiss","hello","hello","maybe"};  System.out.println(find(strs));  }  } |

## 九 项目

#### 1. 大型网站在架构上应当考虑哪些问题？

1.分层：分层是处理任何复杂系统最常见的手段之一，将系统横向切分成若干个层面，每个层面只承担单一的职责，然后通过下层为上层提供的基础设施和服务以及上层对下层的调用来形成一个完整的复杂的系统。计算机网络的开放系统互联参考模型（OSI/RM）和Internet的TCP/IP模型都是分层结构，大型网站的软件系统也可以使用分层的理念将其分为持久层（提供数据存储和访问服务）、业务层（处理业务逻辑，系统中最核心的部分）和表示层（系统交互、视图展示）。需要指出的是：（1）分层是逻辑上的划分，在物理上可以位于同一设备上也可以在不同的设备上部署不同的功能模块，这样可以使用更多的计算资源来应对用户的并发访问；（2）层与层之间应当有清晰的边界，这样分层才有意义，才更利于软件的开发和维护。   
2.分割：分割是对软件的纵向切分。我们可以将大型网站的不同功能和服务分割开，形成高内聚低耦合的功能模块（单元）。在设计初期可以做一个粗粒度的分割，将网站分割为若干个功能模块，后期还可以进一步对每个模块进行细粒度的分割，这样一方面有助于软件的开发和维护，另一方面有助于分布式的部署，提供网站的并发处理能力和功能的扩展。   
3.分布式：除了上面提到的内容，网站的静态资源（JavaScript、CSS、图片等）也可以采用独立分布式部署并采用独立的域名，这样可以减轻应用服务器的负载压力，也使得浏览器对资源的加载更快。数据的存取也应该是分布式的，传统的商业级关系型数据库产品基本上都支持分布式部署，而新生的NoSQL产品几乎都是分布式的。当然，网站后台的业务处理也要使用分布式技术，例如查询索引的构建、数据分析等，这些业务计算规模庞大，可以使用Hadoop以及MapReduce分布式计算框架来处理。   
4.集群：集群使得有更多的服务器提供相同的服务，可以更好的提供对并发的支持。   
5.缓存：所谓缓存就是用空间换取时间的技术，将数据尽可能放在距离计算最近的位置。使用缓存是网站优化的第一定律。我们通常说的CDN、反向代理、热点数据都是对缓存技术的使用。   
6.异步：异步是实现软件实体之间解耦合的又一重要手段。异步架构是典型的生产者消费者模式，二者之间没有直接的调用关系，只要保持数据结构不变，彼此功能实现可以随意变化而不互相影响，这对网站的扩展非常有利。使用异步处理还可以提高系统可用性，加快网站的响应速度（用Ajax加载数据就是一种异步技术），同时还可以起到削峰作用（应对瞬时高并发）。&quot；能推迟处理的都要推迟处理"是网站优化的第二定律，而异步是践行网站优化第二定律的重要手段。   
7.冗余：各种服务器都要提供相应的冗余服务器以便在某台或某些服务器宕机时还能保证网站可以正常工作，同时也提供了灾难恢复的可能性。冗余是网站高可用性的重要保证。

#### 2、你在项目中用到了xml技术的哪些方面?如何实现的?

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

#### 3、数据备份是怎么做的？有没有做读写分离？

主从(一主多从，主要是备份主)，每天备份，备份的文件不要放到数据库服务器上，可以 FTP。要检查有效否。读写分离自己查一下，分库分表做过。

你们使用什么做支付的？如果使用易宝做支付，请求超时了怎么处理？

①重试，一般三次，每次重试都要停顿一会，比如，以第一次停顿 1 秒，第二次停顿 2 秒，第三次停顿 3秒；

②给订单标识付款异常状态，并且发出警告(邮件 JavaMail、短信)给相关人员。

③写个定时任务，定时处理异常状态的订单。(一般在凌晨比较空闲的时间)

如果用户一直向购物车添加商品怎么办？并且他添加一次你查询一次数据库？互联网上用户那么多 ，这样会对数据库造成很大压力你怎么办？( 购物车 redis 存储 )

首先我们使用 redis 作为一个缓冲的内存数据库作为存储用户购物车的信息,当用户登陆以后,讲 redis信息写入关系型数据库.

第二当用户是登录状态的情况下,也是使用 redis 作为缓存,当用户做出关闭当前页面操作的时候,将 redis 的内容统一写入到数据库之中.为了?高 redis 的运行效率我没还可以使用 redis 中的hash 结构.来具体的找到某个字段,做小范围的修改.也可以减轻 redis 的压力.

## 十 众筹项目

#### 1、架构是什么

|  |  |
| --- | --- |
| 分类 | 名称 |
| **表现层框架** | SpringMVC4 |
| **业务（逻辑）层框架** | Spring4 |
| **持久层框架** | Mybatis3 |
| **权限模型** | RBAC |
| **JS框架** | JQuery2.X |
| **HTML5（前端）框架** | Bootstrap3 |
| **流程框架** | Activiti5 |
| **分布式框架** | SpringBoot,SpringCloud |

计算机生成了可选文字:
系统部署
Firew日11
防火场
LO日dB日lanCe
负载均衡
＿匕
奋～－一一－今
心－---－今
Firewall
防火姗
LO日dB日l日nCe
负载均衡
Nginx
朴态资源服务器
Nginx
睁态资源服务器
TOmCat
web服务器
TomCat
web服务器
TOmCat
web服务器
TOmCat
应用服务器
TOm(at
应用服务器
TOmCat
应用服务器
一
}s^N、｝
SANI
SAN…
…丽｝
MySQLI
(oracle)
MySQLZ
(oracle)

#### 2、架构演变过程

|  |
| --- |
| 单一应用架构  当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是关键。 |
| 垂直应用架构  当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的Web框架(MVC)是关键。 |
| 分布式服务架构  当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键。 |
| 流动计算架构  当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的资源调度和治理中心(SOA)是关键。 |

#### 3、数据量有多少

10万

#### 4、多少张表

25张流程框架表;

23张业务表;

9张邮件服务表;

一共57张表;

#### 5、表与表之间的关系

①一对一

例如 : 丈夫和妻子

[1] 让其中一张表的主键,同时充当外键.

共享主键.

|  |  |
| --- | --- |
| husband | Wife |
| Id(PK) name  1 zhangsan  2 lisi | Id(PK,FK) name  1 lily 2 rose |

[2]在其中一张表中额外定义一个字段,作为外键,引用对端表的主键.

|  |  |
| --- | --- |
| husband | Wife |
| Id(PK) name  1 zhangsan  2 lisi | Id(PK) name hid(FK,unique)  1 lily 2 2 rose 1 |

②一对多,多对一

例如:部门和员工

[1]在多的一端表中增加外键,引用一的一端主键.

为什么将外键定义在多的一端,多的一端维护关系更加容易,方便.

|  |  |
| --- | --- |
| Dept | Emp |
| Id(PK) name  1 DEV  2 TEST | Id(PK) name deptid(FK)  1 zhangsan 1 2 lisi 1 |

[2] 菜单(一对多和多对一关系,用一张表来进行描述-自关联)

自关联表的外键,不能设置not null

|  |
| --- |
| menu |
| Id name pid(fk) |
| 1 系统权限菜单 |
| 2 控制面板 1 |
| 3 消息管理 1 |
| 4 权限管理 1 |
| 5 用户管理 4 |
| 6 角色管理 4 |

 ③多对多

在数据库中不能直接描述多对多关联关系的.

一般是借助于一张中间表,含有两个外键,分别引用多对多两个表的主键.

中间表可以有自己的主键,主键可以是自动增长的.

中间表也可以没有自己独立的主键,采用两个外键来充当主键,称为联合主键(表中多个主键,联合表示数据唯一性.).

中间表除了两个外键,可能含有其他的业务字段.例如:学生和课程之间多对多关系,中间表-成绩表,含有成绩信息.

|  |  |
| --- | --- |
| user | role |
| Id(PK) name  1 zhangsan  2 lisi | Id(PK) name  1 Dev  2 Test |

|  |
| --- |
| User\_role |
| Id userid(FK) roleid(FK) xxxx |
| 1 1 1 |
| 2 1 2 |
| 3 2 1 |

计算机生成了可选文字:
用户表
角色衷
鱼色U坦翅那五立址
角色名V月忆H幻论（加）
币
权限表
坦脚助血脸
识V灿忙H氏胶＜的）
饭
F份，:
才
,－即U
F－加比叮J汗～业，J肛打U写厄
用户角色关联衰
用厂吐D
角色D
扣川砂n
抓川B厄E
＜几l>
＜压2>
角色权限关联裹
角色江D翻月日厄R＜众l>
权阵杠D和月石！R＜月忍＞

定义表与表之间的关系:

* PK: Primary key 主键
  + 代理主键:

表示由数据库来分配主键. 一般是没有业务含义的.只是用于标识一条数据,表示数据的唯一性.

例如:

MySQL : auto\_increment

Oracle : sequence

* + 自然主键:

表示业务来维护主键.

例如:注册用户时,采用用户名称作为主键.保存用户前要验证用户名称主键值在数据库中是否存在.

学号,身份证号,订单号

* FK:Foreign Key 外键

#### 6、多少模块?核心模块?

30个左右

后台系统:

权限管理:用户管理,角色管理,许可管理

广告审核,实名认证审核,项目审核

业务模块:项目分类管理,项目管理,流程管理,广告管理,标签管理

系统模块:参数管理

前台系统:

广告,项目分配,发布项目,实名认证,投资项目,支付操作

#### 7、描述一下众筹的业务

企业用户注册登录系统,经过实名认证后,可发布众筹项目;

个人用户注册登录系统,经过实名认证后,可浏览项目,根据分类进程查询项目;可关注项目, 可投资众筹项目;查看投资回报信息等

管理员用户可对系统权限进行分配,控制权限;审核人员可以审核发布的众筹项目,广告,实名认证等;业务管理员可以维护项目分类,查看项目信息,维护广告,维护资质及分类等.参数管理员可设置系统参数.

#### 8、项目进行了怎样的权限控制

未登录系统不能访问系统资源,开发登录权限验证的拦截器进行控制;

登录后没有进行授权的许可资源不能够进行访问,开发访问权限验证拦截器进行权限控制.

首页资源,游客可以任意访问,不需要进行权限控制.

#### 9、项目中使用了那些jQuery插件

1. layer-v3.0.1
2. jquery-form
3. pagination\_zh
4. zTree\_v3

#### 10、项目开发时遇到哪些问题

1. 使用pagination插件死循环 – 注释掉最后一行代码
2. 查询流程定义时,集合数据不能转换为json串,出现死循环 – 转换为map封装数据解决
3. 依赖于activiti流程框架jar包需要存放到mybatis框架与springboot集成jar包的后面,否则报Mapper类找不到异常
4. 需要在类路径下建立processes文件夹

#### 11、量操作时如何提交数据以及如何接收数据

* 第1种方法：表单提交，以字段数组接收；
* 第2种方法：表单提交，以BeanListModel接收；

第1种方法：表单提交，以字段数组接收

* HTML代码如下：

|  |
| --- |
| <h1>submitUserList\_1</h1>  <form action="${pageContext.request.contextPath }/customer/saveCustomer" method="post">  用户名称：<input type="text" name="custName"/><br><br>  年龄：<input type="text" name="custAge"/><br><br>    用户名称：<input type="text" name="custName"/><br><br>  年龄：<input type="text" name="custAge"/><br><br>  <input type="submit" value="保存"><br><br>  </form> |

* Java代码如下：

|  |
| --- |
| @RequestMapping(value="/customer/saveCustomer",method=RequestMethod.POST)  public String saveCustomer(  @Param("custName") String[] custName,  @Param("custAge") Integer[] custAge) throws Exception{  for (int i = 0; i < custAge.length; i++) {  System.out.println(custAge[i]);  }  for (int i = 0; i < custName.length; i++) {  System.out.println(custName[i]);  }  return "redirect:/customer/listCustomer";  } |

 第2种方法：表单提交，以BeanListModel接收

* HTML代码如下：

|  |
| --- |
| <h1>submitUserList\_2</h1>  <form action="${pageContext.request.contextPath }/customer/saveCustomer2" method="post">  用户名称：<input type="text" name="customerList[0].custName"/><br><br>  年龄：<input type="text" name="customerList[0].custAge"/><br><br>    用户名称：<input type="text" name="customerList[1].custName"/><br><br>  年龄：<input type="text" name="customerList[1].custAge"/><br><br>  <input type="submit" value="保存"><br><br>  </form> |

* Java代码如下：

|  |
| --- |
| @RequestMapping(value="/customer/saveCustomer2",method=RequestMethod.POST)  public String saveCustomer2(@RequestParam("customerList") CustomerModel customerModel) throws Exception{  System.out.println(customerModel.getCustomerList());  return "redirect:/customer/listCustomer";  } |
| public class CustomerModel {  private List<Customer> customerList ;  public List<Customer> getCustomerList() {  return customerList;  }  public void setCustomerList(List<Customer> customerList) {  this.customerList = customerList;  }  } |

#### 12、数据显示有哪几种方式

1.表格

计算机生成了可选文字:
名称
PM-
项目经理
SE-
PG-
TL-
GL-
6口QA-
口QC.
口SA-
口〕口口
圳牛架构师
CMO1CMS一配置誉里员
操作
口口a
口．a
口．口
口．口
口．口
口口a
口．口
口口口
口口口
护页．,,'5~

2.树状图

计算机生成了可选文字:
荟兰权限菜单列表
.＊击系统权限菜单
甲控制面板
龟消息管理
一优权限管理
‘用户管理
食角色管理
口许可管理
-．资质管理
邑邑分类誉里
．资质管理
丈i叙呈管理
一二．a审核管理
O实名认证人工审核

3.图表

计算机生成了可选文字:
七日年化收益率（%)
5
基全（）股票
最大值
4
3
气j4
4,.
22
卜饭
又丫Z二二二
沙
1
1
0
一1
二夕
嘛
一2二
2017一05一16
2017一05一18
2017一05一20
2017一05一22

计算机生成了可选文字:
400
300
200
100
基全
定期理财
变现货
口口口．

计算机生成了可选文字:
口口直接访问
口口邮件营销
口口联盟广告
口口视频广告
搜索引擎
某站点用户访问来源
纯属虚构
，巨嘴
搜索引擎一
邮件营销
联盟广告
视频广告

#### 13、权限都控制哪些内容?

①控制菜单/路径是否能够访问

②控制页面按钮是否能够访问

③控制字段是否能够访问

#### 14、文件上传表单开发要求

1. 前台

* 使用表单文件域完成文件上传<input type="file" name="xxxx">
* 表单必须采用POST方式提交数据，因为文件数据放置在请求体中传递，GET方式处理不了，没有请求体
* 文件上传采用特殊的方式传递数据，而不是普通的名值对方式，
* 采用多部件方式提交, 所以在表单标签中需要增加特定属性 enctype="multipart/form-data"

1. 配置

springMVC框架需要提供文件上传解析器，需要在配置文件中进行配置

springMVC框架对请求进行了封装，所以可以通过封装后的请求对象来获取上传的文件数据

#### 15、Freemarker页面如何获取域的数据,常用的表达式是什么

${Request.member.name}

${Session.member.name}

${Application.member.name}

${RequestParameters['id']} 取请求参数值

#### 16、流程框架中常用的服务对象

画出**ProcessEngine**这个类是Activiti5的核心，所有的服务都需要通过ProcessEngine来创建，该类是线程安全的

Activiti5的持久化层采用的是Mybatis，这样移植性好

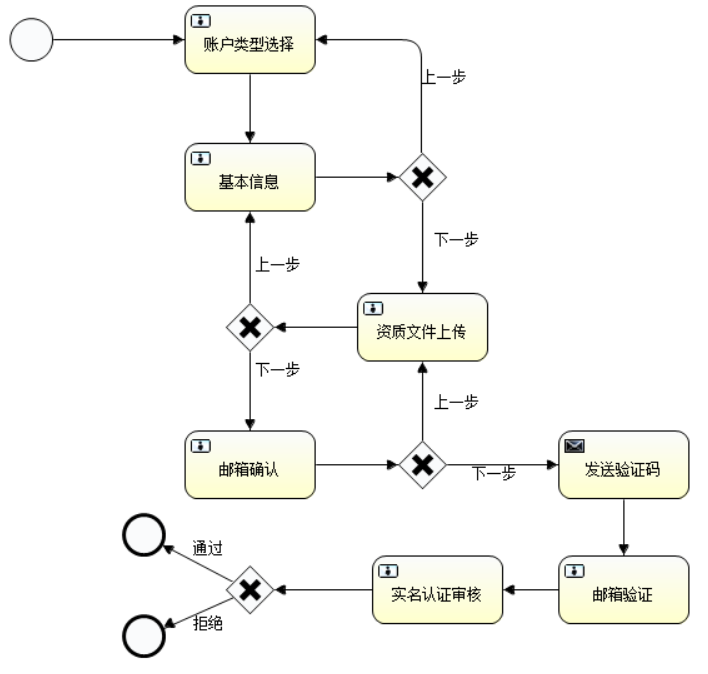
Activiti5主要包括7个Service，这些Service都是通过ProcessEngine创建

* repositoryService（持久化服务） 与数据库进行交互的
* runtimeService（运行时服务） 与运行流程有关
* formService（表单服务）
* identityService（身份信息）
* taskService（任务服务） 与流程中的每一个步骤有关
* historyService（历史信息） 查看历史的流程步骤
* managementService（管理定时任务）在固定的时间点完成固定的任务

#### 17、工作流中有哪些概念



#### 18、实名认证申请的业务流程



#### 19、列出项目中使用的技术清单

|  |  |
| --- | --- |
| 技术 | 说明 |
| Java | 编程语言 |
| Spring | Java集成框架 |
| SpringMVC | Web应用框架 |
| Mybatis | 数据库持久层框架 |
| Activiti5 | 工作流程框架 |
| SVN | 资源版本控制器 |
| maven | 软件项目管理工具 |
| Bootstrap | 前端框架 |
| JQuery | JS脚本框架 |
| JQuery插件 |  |
| HTML5 | 前端网页编程语言 |
| James | 邮件服务器 |
| RBAC权限模型 |  |
| NGINX | web负载均衡服务器 |
| Memcached/Redis | 分布式内存对象缓存系统 |
| ztree | 前端树形结构控件 |
| UML | 统一建模工具（业务） |
| PowerDesigner | 数据库建模工具 |
| MySQL | 数据库 |

#### 20、Springboot和springCloud开发中常用注解

|  |  |
| --- | --- |
| @EnableDiscoveryClient  @SpringBootApplication  @Configuration  @RestController  @EnableFeignClients  @EnableDiscoveryClient | @FeignClient  @ServletComponentScan  @MapperScan  @EnableTransactionManagement  @Bean |