**开发人员面试宝典**

**（Java版）**

目录

[1 基础篇 6](#_Toc414902717)

[1.1 Java介绍 6](#_Toc414902718)

[1.2 J2SE、J2EE、J2ME三者区别 6](#_Toc414902719)

[1.3 JDK、JRE、JVM区别 6](#_Toc414902720)

[1.4 JDK常用的包 7](#_Toc414902721)

[1.5 Java中的基本数据类型 7](#_Toc414902722)

[1.6 Java中switch接受的几种数据类型 8](#_Toc414902723)

[1.7 String、StringBuffer、StringBuilder区别 8](#_Toc414902724)

[1.8 Char型变量能不能存储一个汉字？ 8](#_Toc414902725)

[1.9 ==和equals方法的区别 8](#_Toc414902726)

[1.10 静态变量和实例变量的区别 9](#_Toc414902727)

[1.11 Integer和int的区别 9](#_Toc414902728)

[1.12 Math.round()的使用 9](#_Toc414902729)

[1.13 构造器是否可以被重写 9](#_Toc414902730)

[1.14 抽象类和接口的区别 9](#_Toc414902731)

[1.15 新建对象有几种方式 10](#_Toc414902732)

[1.16 &和&&的区别 10](#_Toc414902733)

[1.17 面向对象什么特性 11](#_Toc414902734)

[1.18 面向对象中多态特性 11](#_Toc414902735)

[1.19 List,set,Collection,Collections 11](#_Toc414902736)

[1.20 线程和进程的区别 11](#_Toc414902737)

[1.21 Jvm内存结构 12](#_Toc414902738)

[1.22 内存泄露和内存溢出 12](#_Toc414902739)

[1.23 解析xml文件的几种技术 12](#_Toc414902740)

[1.24 Dom4j与sax之间的对比 13](#_Toc414902741)

[1.25 java Exception体系结构 13](#_Toc414902742)

[1.26 抽象类和接口的区别 14](#_Toc414902743)

[1.27 修饰符的作用 14](#_Toc414902744)

[1.28 HashMap和Hashtable的区别 14](#_Toc414902745)

[1.29 字节流和字符流的区别 15](#_Toc414902746)

[1.30 运行异常和检查异常有何不同 15](#_Toc414902747)

[1.31 error和exception的区别 15](#_Toc414902748)

[1.32 举例最常用的五个运行时异常 15](#_Toc414902749)

[1.33 Sleep和wait的区别 15](#_Toc414902750)

[1.34 多线程有几种实现方式 16](#_Toc414902751)

[1.35 启动一个线程用run还是start 16](#_Toc414902752)

[1.36 List和Map的区别 16](#_Toc414902753)

[1.37 说出ArrayList,Vector, LinkedList的存储性能和特性 17](#_Toc414902754)

[1.38 Java序列化 17](#_Toc414902755)

[1.39 堆和栈的区别 17](#_Toc414902756)

[1.40 什么时候用断言 17](#_Toc414902757)

[1.41 Java中会有内存泄露吗 18](#_Toc414902758)

[1.42 简述反射的作用 18](#_Toc414902759)

[2 Servlet篇 19](#_Toc414902760)

[2.1 Servlet概述 19](#_Toc414902761)

[2.2 九大隐式对象 20](#_Toc414902762)

[2.3 page和pageContext的区别 20](#_Toc414902763)

[2.4 GET和POST的区别 20](#_Toc414902764)

[2.5 转发与重定向的区别 20](#_Toc414902765)

[2.6 四大作用域介绍 21](#_Toc414902766)

[2.7 session和cookie的区别 21](#_Toc414902767)

[2.8 request和session的区别 21](#_Toc414902768)

[2.9 报错状态码 22](#_Toc414902769)

[2.10 常见协议及默认的端口号 22](#_Toc414902770)

[2.11 对过滤器的理解 22](#_Toc414902771)

[3 脚本篇 23](#_Toc414902772)

[3.1 onready和onload的区别 23](#_Toc414902773)

[3.2 对jQuery简单介绍 23](#_Toc414902774)

[3.3 XML和Json的特点 23](#_Toc414902775)

[3.4 对JQuery中Ajax介绍 24](#_Toc414902776)

[3.5 对Ajax介绍 24](#_Toc414902777)

[4 框架篇 25](#_Toc414902778)

[4.1 Struts1运行原理 25](#_Toc414902779)

[4.2 Struts2运行原理 25](#_Toc414902780)

[4.3 Hibernate运行原理 26](#_Toc414902781)

[4.4 Spring运行原理 27](#_Toc414902782)

[4.5 SpringMVC运行原理 27](#_Toc414902783)

[4.6 Struts1和Struts2区别 28](#_Toc414902784)

[4.7 SpringMVC、Struts1和Struts2区别 29](#_Toc414902785)

[4.8 Struts2在的type类型 29](#_Toc414902786)

[4.9 Struts2的常用标签 29](#_Toc414902787)

[4.10 Hibernate可get和load的区别 30](#_Toc414902788)

[4.11 Hibernate、Ibatis、JDBC区别 30](#_Toc414902789)

[4.12 Hibernate与JDBC区别 30](#_Toc414902790)

[4.13 Hibernate两大配置文件 30](#_Toc414902791)

[4.14 Hibernate中的事务处理 31](#_Toc414902792)

[4.15 Hibernate三种状态及如何转换 31](#_Toc414902793)

[4.16 Hibernate中的五大核心接口 31](#_Toc414902794)

[4.17 Hibernate缓存概述 32](#_Toc414902795)

[4.18 SSH概述 32](#_Toc414902796)

[4.19 分页步骤 33](#_Toc414902797)

[4.20 如何防止表单重复提交 33](#_Toc414902798)

[4.21 通过Struts2来解决表单重复提交 33](#_Toc414902799)

[4.22 拦截器 33](#_Toc414902800)

[5 数据库篇 35](#_Toc414902801)

[5.1 JDBC连接数据库步骤 35](#_Toc414902802)

[5.2 数据库连接池 35](#_Toc414902803)

[5.3 jdbc分段批量提交的时候出现异常怎么处理? 35](#_Toc414902804)

[5.4 JDBC批量处理数据 36](#_Toc414902805)

[5.5 Oracle分页语句 36](#_Toc414902806)

[5.6 MySQL分页语句 36](#_Toc414902807)

[5.7 Oracle基本数据类型 37](#_Toc414902808)

[5.8 id、rowid、rownum的区别 37](#_Toc414902809)

[5.9 主键和唯一索引的区别？ 37](#_Toc414902810)

[5.10 Preparedstatement和statement的区别 38](#_Toc414902811)

[5.11 数据库三范氏 38](#_Toc414902812)

[5.12 视图概述 38](#_Toc414902813)

[5.13 存储过程概述 38](#_Toc414902814)

[5.14 索引概述 39](#_Toc414902815)

[5.15 Ibatis批量 40](#_Toc414902816)

[6 业务场景篇 42](#_Toc414902817)

[6.1 Spring的概述 42](#_Toc414902818)

[6.2 事务概述 43](#_Toc414902819)

[6.3 权限概述 44](#_Toc414902820)

[6.4 OSCache业务场景 44](#_Toc414902821)

[6.5 线程概述 44](#_Toc414902898)

[6.6 Ajax请求Session超时问题 45](#_Toc414902899)

[6.7 Java线程池概述 46](#_Toc414902900)

[6.8 OSCache概述 47](#_Toc414902901)

[6.9 OSCache+autocomplete+单例业务场景 47](#_Toc414902902)

[6.10 缓存概述 47](#_Toc414902903)

[6.11 实现页面静态化业务场景 48](#_Toc414902904)

[6.12 servlet线程安全描述 48](#_Toc414902905)

[6.13 (jbpm4)工作流引擎描述: 49](#_Toc414902906)

[6.14 JPBM业务场景 50](#_Toc414902907)

[6.15 Ant描述 50](#_Toc414902908)

[6.16 FreeMarker描述 51](#_Toc414902909)

[6.17 webService描述 51](#_Toc414902910)

[6.18 oracle索引概述 52](#_Toc414902911)

[6.19 oracle存储过程 53](#_Toc414902912)

[6.20 Junit 业务场景 53](#_Toc414902913)

[6.21 Apache+Tomcat 实现负载均衡及seesion复制 53](#_Toc414902914)

[6.22 Ant业务场景 54](#_Toc414902915)

[6.23 maven业务场景 55](#_Toc414902916)

[6.24 bugfree的操作步骤 55](#_Toc414902917)

[6.25 Axis2 的配置 56](#_Toc414902918)

[6.26 spring定时器 56](#_Toc414902919)

[6.27 Ext概述 57](#_Toc414902920)

[6.28 lucene的概述 57](#_Toc414902921)

[6.29 线程池作用 58](#_Toc414902922)

[6.30 jbpm是如何和spring进行整合 58](#_Toc414902923)

[6.31 Tomcat优化 58](#_Toc414902924)

[6.32 memcached的介绍 60](#_Toc414902925)

[7 优化篇 60](#_Toc414902926)

[7.1 代码优化 60](#_Toc414902927)

[7.2 业务优化 61](#_Toc414902928)

[7.3 sql优化 61](#_Toc414902929)

[7.4 防sql注入 63](#_Toc414902930)

[7.5 数据库中常用术语 65](#_Toc414902931)

[7.6 深入java虚拟机以及大数据 65](#_Toc414902932)

# 基础篇

# Java介绍

Java是面向对象的，跨平台的，它通过java虚拟机来进行跨平台操作，它可以进行自动垃圾回收的（c语言是通过人工进行垃圾回收）。

java还会进行自动分配内存。（c语言是通过指定进行分配内存的），只需要new一个对象，这个对象占用了多少空间，不需要我们来管，java虚拟机负责管这些，用完之后也不需要我们来释放，java虚拟机会自动释放。

# J2SE、J2EE、J2ME三者区别

Java SE = Java Standard Edition=j2se = java 标准版

Java EE = Java Enterprise Edition=j2ee= java 企业版

Java ME = Java Mobile Edition=j2me = java 移动版

**三者特点：**

**J2SE主要用于桌面程序（swing），控制台开发（main程序）。**

**J2EE企业级开发（JSP,EJB,Spring MVC,Struts,hibernate,ibatis等），用于企业级软件开发，网络开发，web开发。**

**J2ME嵌入式开发(手机，小家电，PDA)。[苹果的ios，黑莓]。**

三者之间的关系：

Java SE（Java Platform，Standard Edition，Java标准版）就是基于JDK和JRE的。Java SE为Java EE提供了基础。Java EE除了基于我们这个所谓的Java SE外，还新加了企业应用所需的类库。

# JDK、JRE、JVM区别

Jdk【Java Development ToolKit】就是java开发工具箱， JDK是整个JAVA的核心里边包含了jre，它除了包含jre之外还包含了一些javac的工具类，把java源文件编译成class文件，java文件是用来运行这个程序的，除此之外，里边还包含了java源生的API，java.lang.integer在rt的jar包里边【可以在项目中看到】，通过rt这个jar包来调用我们的这些io流写入写出等。

JDK有以下三种版本：

J2SE，standard edition，标准版，是我们通常用的一个版本

J2EE，enterpsise edtion，企业版，使用这种JDK开发J2EE应用程序

J2ME，micro edtion，主要用于移动设备、嵌入式设备上的java应用程序

Jre【Java Runtime Enviromental】是java运行时环境，那么所谓的java运行时环境，就是为了保证java程序能够运行时，所必备的一基础环境，也就是它只是保证java程序运行的，不能用来开发，而jdk才是用来开发的，所有的Java程序都要在JRE下才能运行。

包括JVM和JAVA核心类库和支持文件。与JDK相比，它不包含开发工具——编译器、调试器和其它工具。Jre里边包含jvm。

Jvm：【Java Virtual Mechinal】因为jre是java运行时环境，java运行靠什么运行，而底层就是依赖于jvm，即java虚拟机，java虚拟机用来加载类文件，java中之所以有跨平台的作用，就是因为我们的jvm。

三者关系：J2se是基于jdk和jre，JDK是整个JAVA的核心里边包含了jre，Jre里边包含jvm。

# JDK常用的包

java.lang: 这个是系统的基础类，比如String、Math、Integer、System和Thread，提供常用功能。

java.io: 这里面是所有输入输出有关的类，比如文件操作等。

java.net: 这里面是与网络有关的类，比如URL,URLConnection等。

java.util : 这个是系统辅助类，特别是集合类Collection,List,Map等。

java.sql: 这个是数据库操作的类，Connection，Statememt，ResultSet等。

# Java中的基本数据类型

|  |  |
| --- | --- |
| **数据类型** | **大小** |
| byte(字节) | 1(8位) |
| shot(短整型) | 2(16位) |
| int(整型) | 4(32位) |
| long(长整型) | 8(32位) **// 64位** |
| float(浮点型) | 4(32位) |
| double(双精度) | 8(64位) |
| char(字符型) | 2(16位) |
| boolean(布尔型) | 1位 |

附加：

String是基本数据类型吗?(String不是基本数据类型)

String的长度是多少，有限制?(长度受内存大小的影响)

# Java中switch接受的几种数据类型

Short， int， byte，char

enum（JDK1.5以后支持）

String（JDK1.7以后支持）

# String、StringBuffer、StringBuilder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。
4. 另一种说法，JDK1.6做了优化，通过String声明的字符串在进行用“+”进行拼接时，底层调用的是StringBuffer，所以性能上基本和后两者没有什么区别。



# Char型变量能不能存储一个汉字？

char型变量是用来存储Unicode编码的字符的，unicode编码字符集中包含了汉字，补充说明：unicode编码占用两个字节，所以，char类型的变量也是占用两个字节。

# ==和equals方法的区别

“==”如果前后比较的是对象，则比较的是对象的地址，如果前后是数字，则比较的数值。

“equals”如果前后是对象，比较的是对象的地址。如果比较的对象重写了equals方法，则有可能比较的是“值”。像八个基本数据类型的封装类、String、File、Date都比较的是值。

# 静态变量和实例变量的区别

在语法定义上的区别：静态变量前要加static关键字，而实例变量前则不加。

在程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量。静态变量不属于某个实例对象，而是属于类，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。

总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

# Integer和int的区别

int是java提供的8种原始数据类型之一，意思整型，占用4字节。

Integer是java为int提供的封装类，是引用数据类型。

int的默认值为0，而Integer的默认值为null，即Integer可以区分出未赋值和值为0的区别，int则无法表达出未赋值的情况。

# Math.round()的使用

Math类中提供了三个与取整有关的方法：ceil、floor、round，这些方法的作用与它们的英文名称的含义相对应，例如，ceil的英文意义是天花板，该方法就表示向上取整，Math.ceil(11.3)的结果为12,Math.ceil(-11.3)的结果是-11；floor的英文意义是地板，该方法就表示向下取整，Math.ceil(11.6)的结果为11,Math.ceil(-11.6)的结果是-12；最难掌握的是round方法，它表示“四舍五入”，算法为Math.floor(x+0.5)，即将原来的数字加上0.5后再向下取整，所以，Math.round(11.5)的结果为12，Math.round(-11.5)的结果为-11。

# 构造器是否可以被重写

构造器Constructor不能被继承，因此不能重写Override，但可以被重载Overload。

# 抽象类和接口的区别

含有abstract修饰符的class即为抽象类，abstract类不能创建的实例对象。含有abstract方法的类必须定义为abstract class，abstract class类中的方法不必是抽象的。abstract class类中定义抽象方法必须在具体(Concrete)子类中实现，所以，**不能有抽象构造方法或抽象静态方法。**如果的子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为abstract类型。

接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为public abstract类型，接口中的成员变量类型默认为public static final。

下面比较一下两者的语法区别：

1. 抽象类可以有构造方法，接口中不能有构造方法。

2. 抽象类中可以有普通成员变量，接口中没有普通成员变量

3. 抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法。

4. 抽象类中的抽象方法的访问类型可以是public，protected和默认类型，但接口中的抽象方法只能是public类型的，并且默认即为public abstract类型。

5. 抽象类中可以包含静态方法**，**接口中不能包含静态方法

6. 抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是public static final类型，并且默认即为public static final类型。

7. 一个类可以实现多个接口，但只能继承一个抽象类。**(单继承，多实现)**

下面接着再说说两者在应用上的区别：

接口更多的是在系统架构设计方法发挥作用，主要用于定义模块之间的通信契约；

而抽象类在代码实现方面发挥作用，可以实现代码的重用。

# 新建对象有几种方式

1. 使用new关键字

2. 使用反射，调用newInstance

3. 使用**clone方法 (未实现)**

4. 使用序列化与反序列化

5. 动态代理(Proxy类和CGLIB)

# &和&&的区别

&和&&都可以用作逻辑与的运算符，&&为短路与，&不是短路与。

另外&可以做为整数的位运算符

例1：对于if(str != null&& !str.equals(“”))表达式，当str为null时，后面的表达式不会执行，所以不会出现NullPointerException如果将&&改为&，则会抛出NullPointerException异常。

例2：If(x==33 &++y>0) y会增长，if(x==33 && ++y>0)不会增长。

备注：这道题先说两者的共同点，再说出&&和&的特殊之处，并列举一些经典的例子来表明自己理解透彻深入、实际经验丰富。

# 面向对象什么特性

对于面向对象的特性，一般有两种说法：一种是有三大特性，分别是封装，继承，多态，一种说法是有四大特性，封装，继承，多态，抽象。讲三大特性的时候更多一些，注意，这里的提问方式是面向对象的特性，而不是Java的特性。

# 面向对象中多态特性

多态是面向对象很重要的一个特性，转型是多态的具体体现。多态还包括重载和重写。

**转型：转型分为向上转型和向下转型。向上转型是子类对象转换成父类对象，直接转就可以，向下转型是父类对象转换成子类对象，这需要强转，在转换过程中要保证此父类对象“本体”是由子类对象实例化的。**

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。

# List,set,Collection,Collections

1. **List和Set都是接口，他们都继承于接口Collection,List是一个有序的可重复的集合，而Set的无序的不可重复的集合。Collection是集合的顶层接口，Collections是一个封装了众多关于集合操作的静态方法的工具类,因为构造方法是私有的，所以不能实例化。**
2. **List接口实现类有ArrayList,LinkedList,Vector。ArrayList和Vector是基于数组实现的,所以查询的时候速度快，而在进行增加和删除的时候速度较慢LinkedList是基于链式存储结构，所以在进行查询的时候速度较慢但在进行增加和删除的时候速度较快。又因为Vector是线程安全的，所以他和ArrayList相比而言，查询效率要低。**

# 线程和进程的区别

1. 进程定义的是应用程序与应用程序之间的边界，通常来说一个进程就代表一个与之对应的应用程序。不同的进程之间不能共享代码和数据空间，而同一进程的不同线程可以共享代码和数据空间。
2. 一个进程可以包括若干个线程，同时创建多个线程来完成某项任务，便是多线程。实现线程的两种方式：继承Thread类，实现Runable接口。

# Jvm内存结构

Java虚拟机的内存结构分为堆(heap)和栈(stack)，堆里面存放是对象实例也就是new出来的对象。栈里面存放的是基本数据类型以及引用数据类型的地址。对于所谓的常量是存储在方法区的常量池里面。

堆是先进先出

栈是先进后出

# 内存泄露和内存溢出

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。

内存泄漏（Memory Leak）是指程序中已动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果。

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。

# 解析xml文件的几种技术

解析xml的几种技术

1. dom4j 、2.sax、3.jaxb、4.jdom、5.dom
2. dom4j

dom4j是一个Java的XML API，类似于jdom，用来读写XML文件的。dom4j是一个非常优秀的Java XML API，具有性能优异、功能强大和极端易用使用的特点，同时它也是一个开放源代码的软件。

1. sax

SAX（simple API for XML）是一种XML解析的替代方法。相比于DOM，SAX是一种速度更快，更有效的方法。它逐行扫描文档，一边扫描一边解析。而且相比于DOM，SAX可以在解析文档的任意时刻停止解析，但任何事物都有其相反的一面，对于SAX来说就是操作复杂。

1. Jaxb

JAXB（Java Architecture for XML Binding) 是一个业界的标准，是一项可以根据XML Schema产生Java类的技术。该过程中，JAXB也提供了将XML实例文档反向生成Java对象树的方法，并能将Java对象树的内容重新写到XML实例文档。从另一方面来讲，JAXB提供了快速而简便的方法将XML模式绑定到Java表示，从而使得Java开发者在Java应用程序中能方便地结合XML数据和处理函数。

# Dom4j与sax之间的对比

dom4j不适合大文件的解析，因为它是一下子将文件加载到内存中，所以有可能出现内存溢出，sax是基于事件来对xml进行解析的，所以他可以解析大文件的xml，也正是因为如此，所以dom4j可以对xml进行灵活的增删改查和导航，而sax没有这么强的灵活性，所以sax经常是用来解析大型xml文件，而要对xml文件进行一些灵活（crud）操作就用dom4j。

# java Exception体系结构

java 异常是程序运行过程中出现的错误。Java把异常当作对象来处理，并定义一个基类java.lang.Throwable作为所有异常的超类。在Java API中定义了许多异常类,分为两大类，错误Error和异常Exception。其中异常类Exception又分为运行时异常(RuntimeException)和非运行时异常(非runtimeException)，也称之为不检查异常（Unchecked Exception）和检查异常（Checked Exception）。

1. Error与Exception

Error是程序无法处理的错误，比如OutOfMemoryError、ThreadDeath等。这些异常发生时，Java虚拟机（JVM）一般会选择线程终止。

Exception是程序本身可以处理的异常，这种异常分两大类运行时异常和非运行时异常。程序中应当尽可能去处理这些异常。

1. 运行时异常和非运行时异常
   1. 运行时异常: 都是RuntimeException类及其子类异常：
      1. IndexOutOfBoundsException 索引越界异常
      2. ArithmeticException：数学计算异常
      3. NullPointerException：空指针异常
      4. ArrayOutOfBoundsException：数组索引越界异常
      5. ClassNotFoundException：类文件未找到异常
      6. ClassCastException：造型异常（类型转换异常）

这些异常是不检查异常（Unchecked Exception），程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引起的。

非运行时异常:是RuntimeException以外的异常，类型上都属于Exception类及其子类。从程序语法角度讲是必须进行处理的异常，如果不处理，程序就不能编译通过。如：

IOException、文件读写异常

FileNotFoundException：文件未找到异常

EOFException：读写文件尾异常

MalformedURLException：URL格式错误异常

SocketException：Socket异常

SQLException：SQL数据库异常

# 抽象类和接口的区别

1. 抽象类由public abstract修饰，接口由public interface修饰。
2. 类（抽象类也是类）只支持单继承，但接口可以实现多个接口。接口不是类，接口和类同属于Java中的一个类型。
3. 抽象类里由构造器、常量、变量、抽象方法、普通方法构成，接口里只包括常量和抽象方法，没有构造器和普通方法。
4. 是否是使用抽象类还是接口，主要看想要实现什么样的事情，如果侧重于描述事务，应该选择抽象类，如果侧重于定义功能，建议选择使用接口。

# 修饰符的作用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰符 | 同一个类中 | 同一个包中 | 子类中 | 全局 |
| private | Yes |  |  |  |
| Default | Yes | Yes |  |  |
| protected | Yes | Yes | Yes |  |
| public | Yes | Yes | Yes | Yes |

# HashMap和Hashtable的区别

1. HashMap和Hashtable是Map接口下的两个实现类，因为Map对象是键值对的，所以此两类也是键值对的。
2. HashMap是线程非安全的，Hashtable是线程安全的，所以HashMap的效率高于Hashtable。
3. HashMap允许键或值为null，键最多只可以有一个为null，值不受限制。而Hashtable键或值都不许为null。

注意：Hashtable中的“t”是小写的。

# 字节流和字符流的区别

以stream结尾都是字节流，以reader和writer结尾都是字符流，两者的区别就是读写的时候一个是按字节读写，一个是按字符。在实际使用时差不多。

在读写文件需要对内容按行处理，比如比较特定字符，处理某一行数据的时候一般会选择字符流。只是读写文件，和文件内容无关的，一般选择字节流。

# 运行异常和检查异常有何不同

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。

# error和exception的区别

error类 和 exception类 。

Error和Exception的区别Error类一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢出等。如java.lang.StackOverFlowError和Java.lang.OutOfMemoryError。对于这类错误，Java编译器不去检查他们。对于这类错误的导致的应用程序中断，仅靠程序本身无法恢复和预防，遇到这样的错误，建议让程序终止。

Exception类表示程序可以处理的异常，可以捕获且可能恢复。遇到这类异常，应该尽可能处理异常，使程序恢复运行，而不应该随意终止异常。

error 表示恢复不是不可能但很困难的情况下的一种严重问题。比如说内存溢出，不可能指望程序能处理这样的情况。exception表示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。

# 举例最常用的五个运行时异常

NullPointerException空指针、ArrayIndexOutOfBoundsException数组越界、ClassCastException 类转换异常、IllegelArgumentException输入输出异常、SecurityException 权限不够。

# Sleep和wait的区别

（网上的答案：sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，给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。）

sleep就是正在执行的线程主动让出cpu，cpu去执行其他线程，在sleep指定的时间过后，cpu才会回到这个线程上继续往下执行，如果当前线程进入了同步锁，sleep方法并不会释放锁，即使当前线程使用sleep方法让出了cpu，但其他被同步锁挡住了的线程也无法得到执行。wait是指在一个已经进入了同步锁的线程内，让自己暂时让出同步锁，以便其他正在等待此锁的线程可以得到同步锁并运行，只有其他线程调用了notify方法（notify并不释放锁，只是告诉调用过wait方法的线程可以去参与获得锁的竞争了，但不是马上得到锁，因为锁还在别人手里，别人还没释放。如果notify方法后面的代码还有很多，需要这些代码执行完后才会释放锁，可以在notfiy方法后增加一个等待和一些代码，看看效果），调用wait方法的线程就会解除wait状态和程序可以再次得到锁后继续向下运行。对于wait的讲解一定要配合例子代码来说明，才显得自己真明白。

# 多线程有几种实现方式

多线程有两种实现方法，分别是继承Thread类与实现Runnable接口

同步的实现方面有五种，分别是synchronized、wait与notify、sleep、suspend、join

synchronized: 一直持有锁，直至执行结束

wait():使一个线程处于等待状态，并从且释放所持有的对象的lock，需捕获异常。

sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，需捕获异常，不释放锁。

notify():唤醒一个处于等待状态的线程，注意的是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，而且不是按优先级。

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。

# 启动一个线程用run还是start

启动一个线程是调用start()方法，使线程就绪状态，以后可以被调度为运行状态，一个线程必须关联一些具体的执行代码，run()方法是该线程所关联的执行代码。

# List和Map的区别

一个是存储单列数据的集合，另一个是存储键和值的双列数据的集合，List中存储的数据是有顺序，并且允许重复；Map中存储的数据是

没有顺序的，其键是不能重复的，它的值是可以有重复的。

1. List有重复值，Map没有重复key，但可以有重复值
2. List有序，Map不一定有序
3. List只能存单列值，Map可以存双列值

# 说出ArrayList,Vector, LinkedList的存储性能和特性

1. ArrayList和Vector使用数组存储元素；LinkedList使用链表存储元素

2. ArrayList和Vector插入删除数据时，需要搬运数据，效率较差；LinkedList使用链表，不需要搬运数据，效率高

3. ArrayList和Vectory查询时，按数组下标查询，不需要遍历，效率高；LinkedList需要遍历，查询效率底

# Java序列化

序列化是把内存Java对象保存到存储介质中，反序列化就是把存储介质中的数据转化为Java对象。Java通过ObjectInputStream和ObjectOutputStream实现序列化和反序列化。需要进行序列化的对象的类必须实现Serializable接口，通常情况下需要满足以下条件：

1. 强烈建议手动生成serialVersionUID常量

2. 如果需要加解密的话，需要实现两个方法readObject和writeObject方法

3. 如果使用Hibernate二级缓存或其它缓存服务器的话，对象必须是可序列化的

4. 如果需要远程调用对象或传值的话，则对像需要序列化

5. 序列化类的可序列化成员必须也是可序列化的，不需要序列化的属性用transient修饰

# 堆和栈的区别

Java的内存分为两类，一类是栈内存，一类是堆内存。

栈中存储的是当前线程的方法调用、基本数据类型和对象的引用，栈是有序的。

堆中存储的是对象的值，堆是无序的。

方法中的局部变量使用final修饰后，放在堆中，而不是栈中。

# 什么时候用断言

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。在实现中，assertion就是在程序中的一条语句，它对一个boolean表达式进行检查，一个正确程序必须保证这个boolean表达式的值为true；如果该值为false，说明程序已经处于不正确的状态下，assert将给出警告或退出。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。

# Java中会有内存泄露吗

会，原因：

如果对象被集合类引用时，如果只是添加，而不删除，会引起内存泄漏，严重时会发出内存溢出。

Java中的内存泄露的情况：长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露。

内存泄露的另外一种情况：当一个对象被存储进HashSet或HashMap中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进HashSet集合中时的哈希值就不同了，在这种情况下，即使在contains方法使用该对象的当前引用作为的参数去HashSet集合中检索对象，也将返回找不到对象的结果，这也会导致无法从HashSet集合中单独删除当前对象，造成内存泄露。

# 简述反射的作用

假如我们有两个程序员，一个程序员在写程序的时候，需要使用第二个程序员所写的类，但第二个程序员并没完成他所写的类。那么第一个程序员的代码能否通过编译呢？这是不能通过编译的。利用Java反射的机制，就可以让第一个程序员在没有得到第二个程序员所写的类的时候，来完成自身代码的编译。

Java的反射机制它知道类的基本结构，这种对Java类结构探知的能力，我们称为Java类的“自审”。大家都用过Jcreator和eclipse。当我们构建出一个对象的时候，去调用该对象的方法和属性的时候。一按点，编译工具就会自动的把该对象能够使用的所有的方法和属性全部都列出来，供用户进行选择。这就是利用了Java反射的原理，是对我们创建对象的探知、自审。

# Servlet篇

# Servlet概述

Servlet是一个web容器，我们通常用的servlet是httpservlet，而httpservlet又是继承于genericservlet，而genericservlet又实现了servlet接口。

servlet的生命周期是 ：先进行实例化，然后是初始化，然后是提高服务，然后销毁，最后不可用，在这五个生命周期，其中，初始化是调用的init方法，这个方法只有一个，而提高服务的时候调用的是service方法，而我们具体在我们所写的这个方法中，因为我们继承了httpservlet，其实就是对应了doGet（），doPost(),这种方法，然后据我了解，servlet是单例的。非线程安全的，我们通常有一下几种方案来解决：

第一种，继承SingleThreadModel但是这样每次都会创建一个新的servlet实例，但这样消耗服务器的内存，降低了性能，并且这个接口现在已经过时了，不推荐使用。

第二种：我们尽量避免使用全局变量，就我个人而言，我比较喜欢使用这种方法。

第三种，我们可以通过使用ThreadLocal， 内部结构是一个Map结构，用当前线程作为key,他会创建多个副本。get,set方。

第四种，我们当然还可以来加锁，进行解决线程问题。

而且我还知道，向我们这种常用的MVC框架，struts1，spring这些MVC框架，都是基于servlet发展而来的，就比如struts1 的核心总控制器是ActionServlet，而springMVC的前端总控制器是dispatchServlet，在项目我们曾经用serlet来生成 图片验证码的，防止用户进行暴力破解

（别人问了，再回答）

servlet的配置文件 web.xml

<servlet>

<servlet-name>ImageCodeServlet</servlet-name> <servlet-class>org.leopard.code.ImageCodeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ImageCodeServlet</servlet-name>

<url-pattern>/d</url-pattern>

</servlet-mapping>

描述：

我在web.xml中，我首先需要写一个servlet标签，servlet标签中有两个子标签，一个叫servlet-name，这个name可以随便起，但是要保证唯一性，除此之外，在这个servlet-name下有一个servlet-class，这个servlet-class对应的就是我后台提高服务的servlet，除此之外还有一个servlet-mapping，这个里边首先有一个servl-name。，这个servl-name首先要保证和上边的servlet-name保持一致，除此之外还有一个url-pattern，这是一个虚拟路径，是用来发送请求的url地址。

# 九大隐式对象

输入/输出对象： request(也属于作用域通信对象) response out

作用域通信对象： session application pageContext

Servlet 对象： page config

错误对象： exception

# page和pageContext的区别

page是servlet对象；使用this关键字，它的作用范围是在同一页面。

pageContext是作用域通信对象；通常使用setAttribute()和getAttribute()来设置和获取存放对象的值。

# GET和POST的区别

1. get是从服务器上获取数据，post是向服务器传送数据。
2. get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制。
3. get安全性非常低，post安全性较高。但是执行效率却比Post方法好。
4. 在进行文件上传时只能使用post而不能是get。

# 转发与重定向的区别

1. 从数据共享上区别：forword是一个请求的延续，可以共享request作用域的数据。redirect开启一个新的请求，不可以共享request作用域的数据，但可以通过URL方式进行数据发送。
2. 从地址栏上区别：forword转发地址栏不发生变化，redirect转发地址栏发生变化。
3. 从性能上区别：forword性能要高于redirect。
4. 常见使用上的区别：因为上性能上有区别，在本系统中请求跳转建议使用forword，如果是跨域访问，建议使用redirect。

# 四大作用域介绍

四个作用域从大到小：appliaction>session>request>page

* application：全局作用范围，整个应用程序共享.生命周期为：应用程序启动到停止。
* session：会话作用域，当用户首次访问时，产生一个新的会话，以后服务器就可以记住这个会话状态。
* request：请求作用域，就是客户端的一次请求。
* page：一个JSP页面。

以上作用范围使越来越小， request和page的生命周期都是短暂的，他们之间的区别就是：一个request可以包含多个page页(include，forward)。

# session和cookie的区别

session是存储在服务器端，cookie是存储在客户端的，所以从安全来讲session的安全性要比cookie高，然后我们获取session里的信息是通过存放在会话cookie里的sessionid获取的。又由于session是存放在服务器的内存中，所以session里的东西不断增加会造成服务器的负担，所以需要把很重要的信息才存储在session中，而把一些次要东西存储在客户端的cookie里，然后cookie确切的说分为两大类分为会话cookie和持久化cookie，会话cookie确切的说是，存放在客户端浏览器的内存中,所以说他的生命周期和浏览器是一致的，浏览器关了会话cookie也就消失了，然而持久化cookie是存放在客户端硬盘中，而持久化cookie的生命周期就是我们在设置cookie时候设置的那个保存时间，然后我们考虑一问题当浏览器关闭时session会不会丢失，从上面叙述分析session的信息是通过会话cookie的sessionid获取的，当浏览器关闭的时候会话cookie消失所以我们的sessionid也就消失了，但是session的信息还存在服务器端，这时我们只是查不到所谓的session但它并不是不存在。那么，session在什么情况下丢失，就是在服务器关闭的时候，或者是session过期(默认时间是30分钟)，再或者调用了invalidate()的或者是我们想要session中的某一条数据消失调用session.removeAttribute()方法，然后session在什么时候被创建呢，确切的说是通过调用getsession()来创建，这就是session与cookie的区别。

访问HTML页面是不会创建session，但是访问index.JSP时会创建session(JSP实际上是一个Servlet，Servlet中有getSession方法)。

# request和session的区别

1. 他们的生命周期不同，request对应的是一次请求，session对应的是一次会话。
2. request占用资源比较少，相对来说缺乏持续性，而session资源消耗比较大，所以通常使用request来保存信息。

# 报错状态码

301 永久重定向

302 临时重定向

304 服务端 未改变

403 访问无权限\*\*\*

200 正常

404 找不到访问的资源

* 405（方法禁用）：禁用请求中指定的方法
* 415（不支持的媒体类型）：请求的格式不受请求页面的支持



1. 内部错误

503 服务器承受不了压力报错

# 常见协议及默认的端口号

ftp 21 文件传输协议

ssh 22 SSH服务协议，可以通过ssh来访问Linux系统

Pop3 110 它是因特网电子邮件的第一个离线协议标准

Smtp 25 简单邮件传输协议

http 80 超文本传输协议

https 443 即HTTP下加入SSL层，简单讲是HTTP的安全版。

Svn 3690

tomcat 8080

Oracle 默认端口号1521

Zookeeper：2181

Mysql 默认端口号 3306

Sqlserver 默认端口1433

Postgresql 默认端口 3690

# 对过滤器的理解

filter的概述:

filter是一个过滤器，用来在请求前和响应后进行数据的处理。

filter的生命周期是:

实例化--->初始化(init)-->进行过滤(doFilter)--->销毁(destroy)-->释放资源

一个Filter必须实现javax.servlet.Filter接口

在项目中我们通常通过filter进行编码转换，进行安全验证，进行重复提交的判断。

了解(不需要主动说)：

filter 相当于 拦截器 相当于Spring AO

# 脚本篇

# onready和onload的区别

1.onready比onload先执行。

2.onready是在页面解析完成之后执行，而onload是在页面所有元素加载后执行。

3.onload只执行最后一个而onready可以执行多个。

# 对jQuery简单介绍

jquery是一个轻量级的js框架，具有跨浏览器的特性，兼容性好，并且封装了很多工具，方便使用。常用的有：选择器，om操作，ajax(ajax不能跨域)，特效，工具类，验证、UI引用。

# XML和Json的特点

Xml特点：

　　1、有且只有一个根节点；

　　2、数据传输的载体

　　3、所有的标签都需要自定义

　　4、是纯文本文件

Json（JavaScript Object Notation）特点：

json分为两种格式:

1. json对象(就是在{}中存储键值对，键和值之间用冒号分隔，键 值 对之间用逗号分隔)。
2. json数组(就是[]中存储多个json对象，json对象之间用逗号分隔)，（两者间可以进行相互嵌套）数据传输的载体之一。

区别：

　　传输同样格式的数据，xml需要使用更多的字符进行描述，

流行的是基于json的数据传输。

xml的层次结构比json更清晰。

共同点：

xml和json都是数据传输的载体，并且具有跨平台跨语言的特性。

# 对JQuery中Ajax介绍

JQuery中对Ajax进行了三层封装，第一层封装的方法是$.ajax()，第二层封装的方法有$.post()、$.get()、$(“对象”).load()，第三层封装的方法有$.getJson()、$.getScript()。

# 对Ajax介绍

AJAX的全称是Asynchronous JavaScript And XML（异步JavaScript及 XML），Ajax的核心是JavaScript对象XmlHttpRequest(XHR)。

Ajax的优点：

1. 提高用户体验度（UE）
2. 提高应用程序的性能
3. 进行局部刷新

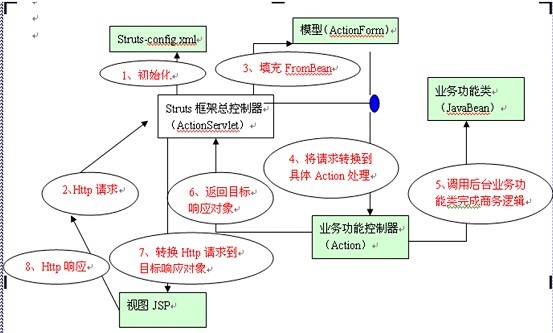
Ajax详细介绍：

1. AJAX不是一种新的编程语言，而是一种用于创建更好更快以及交互性更强的 Web 应用程序的技术。
2. 通过 AJAX，我们的 JavaScript 可使用JavaScript的XMLHttpRequest对象来直接与服务器进行通信。通过这个对象，我们的 JavaScript 可在不重载页面的情况与Web服务器交换数据，即可局部刷新。
3. AJAX 在浏览器与 Web 服务器之间使用异步数据传输（HTTP 请求），这样就可使网页从服务器请求少量的信息，而不是整个页面，减轻服务器的负担，提升站点的性能。
4. AJAX 可使因特网应用程序更小、更快，更友好，用户体验（UE）好。
5. Ajax是基于标准化并被广泛支持的技术，并且不需要插件和下载小程序。

# 框架篇

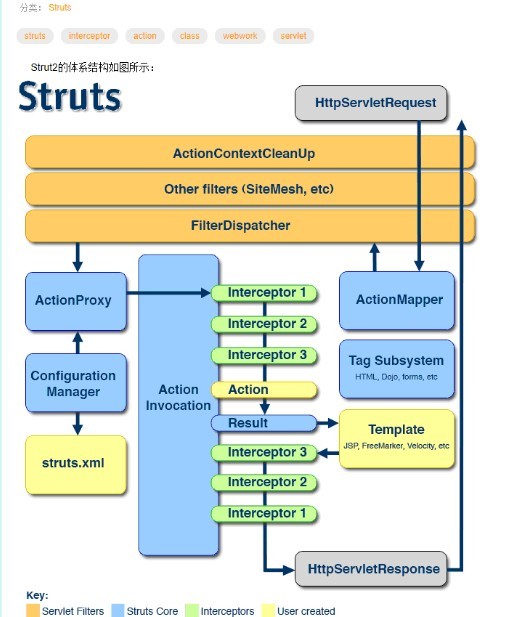
# Struts1运行原理

1. 在servlet容器启动时加载web.xml，通过前端控制器ActionServlet加载struts-config.xml并进行解析。
2. 客户端发送请求，请求被ActionServlet控制器接收。
3. ActionServlet在处理请求时，根据struts-config.xml配置关系通过访问路径找到相应的ActionForm和Action。Struts将数据封装在ActionForm中，将封装后的ActionForm传递给Action相应方法进行使用。
4. 在Action处理完请求后，通过ActionMapping中的findForward()方法返回一个ActionForward对象，通过struts-config.xml配置的forward名称找到响应的请求路径。



# Struts2运行原理

1. 在servlet容器启动时加载web.xml，通过核心控制器FilterDispatcher加载struts.xml文件并进行解析，将解析的信息封装在ActionMapper类中。
2. 客户端发送请求，请求被核心控制器FilterDispatcher进行拦截。
3. FilterDispatcher通过ActionMapper找到对应的ActionForm和Action中的响应方法，在访问方法前先经过Struts2设置的17个默认拦截器，其中一个拦截器的功能是将请 求时的数据封装在ActionForm中，然后方法进行响应。
4. 方法处理完后返回一个字符串，通过sturts.xml配置文件中的<result />找到响应的请求路径。



需要注意的两点：

1. struts 2.1.3以后FilterDispatcher核心已标注为过时，建议使用StrutsPrepareAndExecuteFilter过滤器。
2. struts2是基于拦截器实现的，在理解struts2运行原理时，一定要搞清楚拦截器是什么。

# Hibernate运行原理

首先通过configuration去加载hibernate.cfg.xml这个配置文件，根据配置文件的信息去创建sessionFactory，sessionFactory是线程安全的，是一个session工厂，用来创建session，session是线程不安全的，相当于jdbc的connection，最后通过session去进行数据库的各种操作，在进行操作的时候通过transaction进行事务的控制。

# Spring运行原理

内部最核心的就是IOC了， 动态注入，让一个对象的创建不用new了，可以自动的生产，这其实就是利用java里的反射。反射其实就是在运行时动态的去创建、调用对象，Spring就是在运行时，跟xml Spring的配置文件来动态的创建对象，和调用对象里的方法的。

Spring还有一个核心就是AOP这个就是面向切面编程，可以为某一类对象 进行监督和控制（也就是在调用这类对象的具体方法的前后去调用你指定的 模块）从而达到对一个模块扩充的功能。这些都是通过配置类达到的。

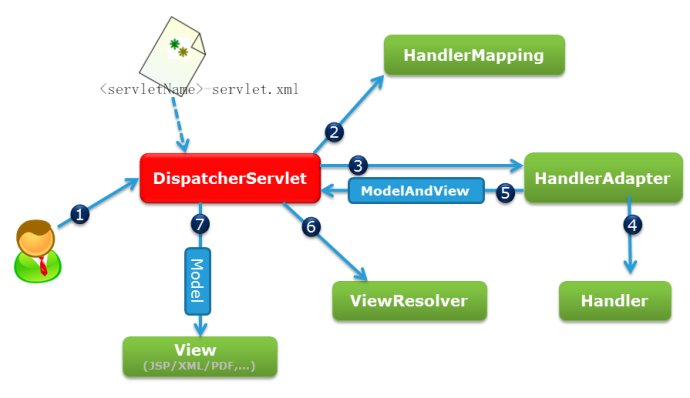
Spring目的：就是让对象与对象（模块与模块）之间的关系没有通过代码来关联，都是通过配置类说明 管理的（Spring根据这些配置 内部通过反射去动态的组装对象）要记住：Spring是一个容器，凡是在容器里的对象才会有Spring所提供的这些服务和功能。

Spring里用的最经典的一个设计模式就是：模板方法模式。（这里我都不介绍了，是一个很常用的设计模式）

Spring里的配置是很多的，很难都记住，但是Spring里的精华也无非就是以上的两点，把以上两点跟理解了 也就基本上掌握了Spring。

# SpringMVC运行原理

1. 客户端发出一个http请求给web服务器，web服务器对http请求进行解析，如果匹配DispatcherServlet的请求映射路径（在web.xml中指定），web容器将请求转交给DispatcherServlet.
2. DipatcherServlet接收到这个请求之后将根据请求的信息（包括URL、Http方法、请求报文头和请求参数Cookie等）以及HandlerMapping的配置找到处理请求的处理器（Handler）。
3. DispatcherServlet根据HandlerMapping找到对应的Handle，将处理权交给Handler（Handler将具体的处理进行封装），再由具体的HandlerAdapter对Handler进行具体的调用。
4. Handler对数据处理完成以后将返回一个ModelAndView()对象给DispatcherServlet。
5. Handler返回的ModelAndView()只是一个逻辑视图并不是一个正式的视图，DispatcherSevlet通过ViewResolver将逻辑视图转化为真正的视图View。
6. Dispatcher通过model解析出ModelAndView()中的参数进行解析最终展现出完整的view并返回给客户端。



# Struts1和Struts2区别

**依赖关系：**

Struts2以WebWork（也就是MVC框架）为核心。采用拦截器的机制来处理用户请求，struts1严重依赖于servletAPI，属于侵入性框架，struts2不严重依赖于servletAPI，属于非侵入性框架。

**线程模型方面：**

struts1的Action是单实例的，一个Action的实例处理所有的请求。

struts2的Action是一个请求对应一个实例（每次请求都新new出一个对象），没有线程安全方面的问题。

**封装请求参数：**

struts1中强制使用ActionForm对象 封装请求的参数。

Struts2可以选择使用POJO类（也就是javaBean）来封装请求的参数，或者直接使用Action的属性。

**核心控制器：**

struts1的前端总控制器（核心总控制器）为ActionServlet，是一个Servlet类。

struts2的前端总控制器（核心总控制器）为FileterDispatcher，是一个过滤器。

**说明：**

struts2.1.3以前，核心总控制器：

org.apache.struts2.dispatcher.FilterDispatcher

struts2.1.3以后，核心总控制器：

org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter

# SpringMVC、Struts1和Struts2区别

1. springMVC单例非线程安全，struts1单例非线程安全，struts2线程安全对每个请求都产生一个实例。
2. springMVC和String1的入口都是servlet，而struts2是Filter。springMVC 的前端总控制器为 DispatcherServlet，struts1 的前端总控制器为 ActionServlet，struts2 的前端总控制器为 FilterDispatcher。
3. 参数传递：struts是在接受参数的时候，可以用属性来接受参数，这就说明参数是让多个方法共享的。springMVC用方法来接受参数。
4. springMVC是基于方法的设计，而sturts是基于类。

# Struts2在的type类型

1. dispatcher：它是默认的，用来转向页面，通常处理JSP。
2. redirect：将用户重定向到一个已配置好的URL。
3. redirectAction：将用户重定向到一个已定义好的action。
4. chain：将action和另外一个action链接起来。
5. freemarker：呈现Freemarker模板。
6. httpheader：返回一个已配置好的HTTP头信息响应。
7. stream：向浏览器发送InputSream对象对下载的内容和图片非常有用。
8. velocity：呈现Velocity模板。
9. xslt ：该XML可以通过XSL模板进行转换。
10. plaintext：显示原始文件内容，例如文件源代码。

# Struts2的常用标签

首先需要引用 <%@taglib prefix="s" uri="/struts-tags"%>

1. <s:if></s:if> 判断标签 后面可跟 <s:else>
2. <s:iterator> </s:iterator> 迭代标签
3. <s:include></s:include> 引入标签 可以把一个JSP页面或者servlet引入一个页面中
4. <s:property></s:property> 输出标签
5. <s:set></s:set> 标签赋予变量一个特定范围内的值
6. <s:form></s:form> 表单标签
7. <s:testarea></s:textarea> 文本域标签
8. <s:select></s:select> 下拉标签
9. <s:url></s:url> 声明一个url的路径

最常用的是：

判断<s:if></s:if>

循环<s:iterator></s:terator>

输出<s:property></s:property>

# Hibernate可get和load的区别

加载方式：

　　　　load为延迟加载（返回的是一个只有id属性的代理，只有使用该对象属性时，才 发出sql语句）。

　　　　get为立即加载（执行时，会立即向数据库发出sql语句）。

返回结果：

　　　　load检索不到记录时,会抛ObjectNotFoundException异常。

　　　　get检索不到记录时,会返回null。

# Hibernate、Ibatis、JDBC区别

Hibernate属于全自动， Ibatis属于半自动，Jdbc属于手动，从开发效率上讲hibernate较高，ibatis居中，jdbc较低，从执行效率上讲hibernate较低，ibatis居中，jdbc较高，因为jdbc是手工写sql语句，程序员对sql的控制能力更大，可以根据业务需要进行优化，而ibatis虽然也可以对sql进行优化，但是他里面将resultset封装为实体的过程中采用了反射机制所以一定程度上影响了性能，而hibernate因为高度封装所以开发效率相对较高，但正因为这个原因，所以程序员在对sql语句的控制和优化方面相对比较弱，而且在将resultset封装成实体的过程中也采用了反射机制，所以在性能方面较低。

# Hibernate与JDBC区别

1. hibernate和jdbc主要区别就是，hibernate先检索缓存中的映射对象( 即hibernate操作的是对象)，而jdbc则是直接操作数据库。
2. Hibernate是JDBC的轻量级的对象封装，它是一个独立的对象持久层框架。Hibernate可以用在任何JDBC可以使用的场合。
3. Hibernate是一个和JDBC密切关联的框架，所以Hibernate的兼容性和JDBC驱动，和数据库都有一定的关系，但是和使用它的Java程序，和App Server没有任何关系，也不存在兼容性问题。
4. 如果正确的使用JDBC技术,它的执行效率一定比hibernate要好,因为hibernate是基于jdbc的技术。
5. JDBC使用的是SQL语句，Hibernate使用的是HQL语句，但是HQL语句最终还会隐式转换成SQL语句执行。

# Hibernate两大配置文件

\*.hbm.xml：主键生成策略，映射关系，一对多，一对一的关系。

Hibernate.cfg.xml：方言(用哪个数据库)，数据库连接信息，包含\*.hbm.xml内容，映射文件，也可以配事务。

# Hibernate中的事务处理

开启事务 session.beginTransaction();

执行相关的操作，如果成功则session.getTransaction().commit();

执行操作失败则 session.getTransaction.rollback();

# Hibernate三种状态及如何转换

**Transient(临时)**

new 一个初始化对象后，并没有在数据库里保存数据，处于临时状态；

**Persistent(持久化)**

当执行save()方法，调用session.close()方法之前，内存中的对象与数据库有对应关系处于持久化状态；

**Detached(托管/游离)**

　　　当执行session.close()之后，处于托管状态；

**状态的转换**

处于托管状态下，调用update()方法后，转换为持久化状态；

在持久化状态下，执行delete()方法后，转换为临时状态；

在未初始化对象之前，调用get(),load(),find(),iterate()之后，直接进入持久化状态。

# Hibernate中的五大核心接口

1. Configuration接口的作用是对Hibernate进行配置，以及对它进行启动。（加载hibernate.cfg.xml）并创建一个SessionFactory对象。
2. SessionFactory接口负责初始化Hibernate。它充当数据存储源的代理，并负责创建Session对象。SessionFactory是线程安全的。
3. Session（会话）接口是Hibernate应用使用的主要接口。Session接口负责执行被持久化对象的CRUD操作(增删改查)。Session对象是非线程安全的。Session 相当于jdbc的connection
4. Query和Criteria接口负责执行各种数据库查询。
5. Transaction接口负责操作相关的事务。

# Hibernate缓存概述

hibernate分为一级缓存即session缓存也叫事务级别的缓存以及二级缓存sessionFactory即应用级别的缓存，还有查询缓存即三级缓存。

一级缓存的生命周期和session的生命周期保持一致，hibernate默认就启用了一级缓存，不能将其关闭，可以通过session.clear()和session.evict(object)来管理一级缓存。其中get,load,iterate都会使用一级缓存，一级缓存缓存的是对象。

二级缓存的生命周期和sessionFactory的生命周期保持一致，可以跨session,被多个session共享，hibernate3默认开启二级缓存,也可以手动开启并指定缓存插件如ehcache,oscache等。二级缓存也只能缓存对象。

三级缓存也叫查询缓存，查询缓存是针对普通属性结果集的缓存，对实体对象的结果集只缓存id。对query.list()起作用，query.iterate不起作用，也就是query.iterate不使用查询缓存。

# SSH概述

ssh是web开发中常见的一种框架

s-struts

s-spring

h-hibernate

其中struts在框架中充当控制器，实现MVC，主要用来处理用户的请求，和跳转页面。使项目结构清晰，开发者只需要关注业务逻辑的实现即可。

spring在ssh充当粘合剂，粘合struts-sping-hibernate，主要用来进行事物的控制，hibernate-充当数据库持久层，主要用它来与数据库交互，提高开发效率，减轻程序员sql控制要求，而且hibernate通过反射机制，有灵活的映射性，还支持各种关系，一对一，一对多，多对多。

在进行ssh整合的时候，我们应该注意：

1. Action继承于ActionSupport；
2. 引入struts-spring-plugin.jar包，从而完成struts和spring的整合；
3. 在struts2的action中注入service，保证service的名字和配置文件中的一致，并生成get,set方法；
4. Dao层继承于hibernateDaoSupport；
5. 在dao层的配置文件中注入sessionFactory。

# 分页步骤

①前台封装一个显示分页的组件

②查询总条数

③后台封装分页工具类，计算开始位置、结束位置、总页数

④后台写支持分页的sql语句

⑤前台包含分页组件，实现分页效果

注意：

查询总条数的where和查询列表信息的where条件要保证一致。

# 如何防止表单重复提交

针对于重复提交的整体解决方案：

1.用redirect来解决重复提交的问题

2.点击一次之后，按钮失效

3.通过loading

4.自定义重复提交过滤器

# 通过Struts2来解决表单重复提交

解决struts2重复提交，可以结合s:token标签来解决重复提交问题。

1.在前端的jsp页面中加入s:token标签，在访问该页面时就会生成隐藏域，该隐藏域中包含一个随机生成的字符串，并把该字符串存入session中。

2.在struts2的配置文件中加入token拦截器后，当正常访问action的时候，会从session中取出该字符串，然后和页面隐藏域中提交字符串做对比，如果一致则正常执行并删除session中存储的字符串。

# 拦截器

什么是拦截器：

拦截器是AOP中的概念，它本身是一段代码，可以通过定义“织入点”，来指定拦截器的代码在“织入点”的前后执行，从而起到拦截的作用

正如上面 Struts2的Reference中讲述的，Struts2的Interceptor，其拦截的对象是Action代码，可以定义在Action代码之前或者之后执行拦截器的代码。

在项目中，我们经常用来拦截通过非正常程序而进行的访问

Struts2的拦截器和Servlet过滤器类似。在执行Action的execute方法之前，Struts2会首先执行在struts.xml中引用的拦截器，在执行完所有引用的拦截器的intercept方法后，会执行Action的execute方法。

其中intercept方法是拦截器的核心方法，所有安装的拦截器都会调用之个方法。在Struts2中已经在struts-default.xml中预定义了一些自带的拦截器，如timer、params等。如果在<package>标签中继承struts-default，则当前package就会自动拥有struts-default.xml中的所有配置。代码如下：

<package name="demo" extends="struts-default" > ... </package>

拦截器是Struts2框架的核心，它主要完成解析请求参数、将请求参数赋值给Action属性、执行数据校验、文件上传等工作

在struts-default.xml中有一个默认的引用，在默认情况下（也就是<action>中未引用拦截器时）会自动引用一些拦截器。struts2中默认的拦截器是defaultStack.

自定义拦截器需要特别注意的是不要忘记引入struts2默认的拦截器。为了实现某些操作，我们可以自定义拦截器，

自定义拦截器有三种方式定义。分别为实现Interceptor接口，继承抽象类AbstractInterceptor，继承MethodFilterInteceptor类。

拦截器在项目中的运用：

同时可以减轻代码冗余，提高重用率。

如果要求用户密码、权限等的验证，就可以用自定义的拦截器进行密码验证和权限限制。对符合的登入者才跳转到正确页面。

# 数据库篇

# JDBC连接数据库步骤

1. 加载驱动。
2. 提供URL，数据库用户名密码，创建Connection连接对象。
3. 创建Statement，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。
4. 执行SQL语句，并处理结果。
5. 关闭JDBC连接。

# 数据库连接池

数据库连接池的优点运行原理：

在我们不使用数据库连接池的时候，每次访问数据库都需要创建连接，使用完成之后需要释放关闭连接，而这样是很耗费资源的。当我们使用数据库连接池的时候，在容器启动的时候就创建了指定数量的连接，之后当我们程序使用的时候就直接从连接池里面取，而不需要创建，同理，当我们使用完的时候也不需要关闭连接，而是将连接返回到连接池中，供其他请求继续使用。

现在常用的开源数据连接池主要有c3p0、dbcp和proxool三种，其中：

hibernate开发组推荐使用c3p0;

spring开发组推荐使用dbcp(dbcp连接池有weblogic连接池同样的问题，就是强行关闭连接或数据库重启后，无法reconnect，告诉连接被重置，这个设置可以解决);

hibernate in action推荐使用c3p0和proxool;

# 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。

# JDBC批量处理数据

批量处理数据:(代码优化:提高程序执行性能)

降低了java程序代码(客户端)和数据库之间的 网络通信的次数。

在jdbc中进行批量插入的核心API为 addBatch,executeBatch

大数据量的插入问题:（jdbc,hibernate,ibatis）

1.每次只插入一条和数据库交互多次(很耗时间)

2.批量插入和数据库只交互一次(内存溢出)

3.分段批量插入(推荐)

jdbc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();

}

# Oracle分页语句

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

# MySQL分页语句

select \* from music where id limit 5,5

# Oracle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.

# id、rowid、rownum的区别

rowid物理位置的唯一标识。而id是逻辑上的唯一标识，所以rowid查找速度要快于id,是目前最快的定位一条记录的方式。

rowid和rownum都是"伪数列"，所谓“伪数列”也就是默认隐藏的一个数列。rownum用于标记结果集中结果顺序的一个字段，它的特点是按顺序标记，而且是连续的，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<或者<=直接关联，如果是>或者=则需要给他起个别名。

# 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。

# Preparedstatement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可以大大提高程序的安全性，因为Prepared statement是用‘？’传参,可以防止sql注入，具有安全性，而statement用的是‘+’字符串拼接，安全性较低。

# 数据库三范氏

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。

# 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

# 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。

# 索引概述

1. 索引的概念

索引就是为了提高数据的检索速度。

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。

1. 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）

1. 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.索引需要占用额外的物理空间

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

4、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：对单个字段建立索引

　　 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 userInfo(firstname,lastname)

3.顺序索引，散列索引,位图索引

# Ibatis批量

this.getSqlMapClientTemplate().execute(

new SqlMapClientCallback() {

public Object doInSqlMapClient(

SqlMapExecutor executor)

throws SQLException {

executor.startBatch();

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

executor.insert(

"productAttach.insertProductAttach",

list.get(i));

}

executor.executeBatch();

return null;

}

});

ibatis,jdbc,hibernate的分段的实现:

都应该在组装list的时候进行拆分（如：action层加入）

if(list.size() % 1000 == 0)

{

productAttachService.addBatch(list);

list.clear();

}

if (list.size() > 0)

productAttachService.addBatch(list);

# 业务场景篇

# Spring的概述

Spring 是完全面向接口的设计，降低程序耦合性，主要是事务控制并创建bean实例对象。在ssh整合时，充当黏合剂的作用。IOC(Inversion of Control) 控制反转/依赖注入，又称DI(Dependency Injection) (依赖注入)

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。

Spring AOP事务的描述：

在spring-common.xml里通过<aop:config>里面先设定一个表达式，设定对service里那些方法 如：对add\* ,delete\*,update\*等开头的方法进行事务拦截。我们需要配置事务的传播（propagation="REQUIRED"）特性,通常把增,删,改以外的操作需要配置成只读事务（read-only="true"）.只读事务可以提高性能。之后引入tx:advice,在tx:advice引用 transactionManager（事务管理）,在事务管理里再引入sessionFactory,sessionFactory注入 dataSource，最后通过<aop:config> 引入txAdvice。

Spring实现ioc控制反转描述：

原来需要我们自己进行bean的创建以及注入，而现在交给spring容器去完成bean的创建以及注入。所谓的“控制反转”就是 对象控制权的转移，从程序代码本身转移到了外部容器。

官方解释:

控制反转即IoC (Inversion of Control)，它把传统上由程序代码直接操控的对象的调用权交给容器，通过容器来实现对象组件的装配和管理。所谓的“控制反转”概念就是对组件对象控制权的转移，从程序代码本身转移到了外部容器。

# 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。

# 权限概述

权限涉及到5张表：

用户表，角色表，权限表(菜单表)，用户角色关联表，角色权限关联表。

当用户登录时，根据用户名和密码到用户表验证信息是否合法，如果合法则获取用户信息，之后根据用户id再到用户角色关联表中得到相关连的角色id集合,之后根据角色id再到角色权限关联表中获取该角色所拥有的权限id集合，然后再根据权限id集合到权限表（菜单表）中获取具体的菜单，展现给当前登录用户，从而达到不同用用户看到不同的菜单权限。

我们通过ZTree来给角色赋权并且通过ZTree来展示菜单，以及通过ZTree来管 理菜单即增加和编辑菜单。

我们做的权限控制到url级别,为了防止用户不登录直接输入url访问的这个弊端，通过拦截器进行拦截验证。

# OSCache业务场景

在我以前的项目中，我们考虑了系统性能问题，这个时候我们采用了Oscache缓存，刚开始把这个功能交给了项目组中的另外一个同事来做的，但是他做完的时候他发现缓存中明明已经缓存了数据，但是在取得时候发现没有数据，我们项目经理让我去帮忙看看这个问题，我阅读完他的代码之后，我发现了他每次缓存的时候都是调用一个新的缓存对象的方法，结果出现了明明已经走了缓存的方法而取不到数据的问题，通过我多年的工作经验，我就想到了应该用单例模式去封装一个单例工具类来调用oscache。但是，在后来的测试过程中，发现当并发访问的时候也会出现上述的问题，这个时候我直接采取的DCL（双重判定锁）单例模式封装了工具类，既解决了线程安全问题，相对的性能问题也考虑到了，这个问题才得到了完善的解决。

# 线程概述

线程的状态以及状态之间的相互转换：

　 1、新建状态(New)：新创建了一个线程对象。

2、就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态(Running)：就绪状态的线程获取了CPU，执行程序代码。

4、阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

(一)、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

(二)、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

(三)、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

5、死亡状态(Dead)：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

实现线程的两种方式：

是继承Thread类或实现Runnable接口，但不管怎样，当new了这个对象后，线程就已经进入了初始状态

wait和sleep的区别：

线程访问：

锁池状态，之后等待锁释放，然后访问代码

wait

等待队列(释放资源)--->调用notify或者notifyall之后锁池状态--->( 等待锁释放)--->可运行状态--->运行状态---->访问代码

sleep,join

不释放资源-->结束后直接进入可运行状态--->运行状态---->访问代码

一个java控制台程序，默认运行两个线程，一个主线程，一个垃圾回收线程。

线程与进程的区别：

1.线程(Thread)与进程(Process)

进程定义的是应用程序与应用程序之间的边界，通常来说一个进程就代表一个与之对应 的应用程序。不同的进程之间不能共享代码和数据空间，而同一进程的不同线程可以共 享代码和数据空间。

2.一个进程可以包括若干个线程，同时创建多个线程来完成某项任务，便是多线程。

# Ajax请求Session超时问题

我在做项目时有时会遇到session超时问题，如果session超时，平常请求没有什么问题，通过拦截器可以正确跳到登陆页面，可是你如果用ajax请求的话这就出现问题了，因为ajax是异步的，局部刷新，所以登陆界面不会再全页面中显示，他只会显示到页面的一部分当中。所以根据我这几年的经验找到了我认为比较好的一种方法。因为那我用的框架是和struts2集成的，所以就在拦截器中进行设置：

首先判断session是否为空就是判断session是否超时，如果超时就取出请求的head头信息request.getHeader("x-requested-with")，如果不为空就和XMLHttpRequest(Ajax标识)进行比较

(request.getHeader("x-requested-with").equalsIgnoreCase("XMLHttpRequest"))) 如果相等说明此请求是ajax请求。

如果是ajax请求就可以用response.setHeader("键","值")来设置一个标识来告诉用户这是ajax请求并且session超时时发出的，这样我就可以在回调函数中取出自己设置的那个唯一标识：XMLHttpRequest.getResponseHeader("");如果取出的值是和自己在后台中设置的值一样的话，就证明session已经超时，这样就可以设置window.location.replace("登陆界面")，来跳转到登陆界面了。

这样做虽然解决了问题，但是，会在每个回调函数中写入那些代码，这样的话代码就会显得特别零散，所以就想能不能定义一个全局的设置所以就找到了jqery的ajaxSetUp方法，通过ajaxSetUp对jqery的ajax进行全局的判断(ajaxSetUp就相当于ajax的拦截器)，通过设置ajaxSetUp里的complete，它就相当于回调函数，这样那就弥补了上一方法的不足。

我做项目时还用到$(document).ajaxStart()，这是ajax请求时的事件；$(document).ajaxSuccess()，这是AJAX请求成功后的事件。我一般用他们来显示遮罩层和隐藏遮罩层用的加遮罩层是为了不让用户重复提交，更提高了用户体验度，让用户知道已经提交了。

# 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。

# OSCache概述

oscache是一个高性能的j2ee框架，可以和任何java代码进行集成，并且还可以通过标签对页面内容进行缓存，还以缓存请求。

我们通常将那些频繁访问但是又不是经常改变的数据进行缓存。为了保证缓存数据的有效性，在数据发生改变的时候，我们要刷新缓存，避免脏数据的出现。刷新缓存的策略有两种，一种是定时刷新，一种手动刷新。

缓存数据的时机通常也分为两种，即在tomcat(web容器)启动时候加载数据进行缓存，另外也可以在用户第一次访问数据的时候进行缓存，这个相当于缓存的立即加载和按需加载。

缓存的层次如下:jsp-->action-->service-->dao,缓存越靠前对性能的提升越大一个action里面可以有多个service,一个service中可以有多个dao或者多个service任何类之间都可以进行相互调用，可以通过构造函数传参，set,get传参或者是方法传参将相关的类连接起来。

# OSCache+autocomplete+单例业务场景

在我以前做某项目的过程中，其中我们在做产品列表的查询的时候为了提高用户的体验度，我们使用了autocomplete插件来代替select进行品牌的选择，才开始的时候每次都要根据用户输入的信息去查询数据库进行模糊匹配返回结果，后来我们考虑到系统的性能，因此我们采用了oscache缓存，才开始这个功能是交给我们项目组中的另外一个同事来做的，但是他做完后，我们在使用这个工具类的时候，发现有时缓存中明明已经有时我们需要的数据，但是从缓存里面取的时候，发现没有，之后项目经理让我去帮这个同事看看这个问题，我经过阅读他的代码发现，它里面在使用缓存的时候，针对于每次方法的调用都产生一个新的实例，结果导致了上面的问题，这个时候我想起了可以使用设计模式中的单例模式来解决这个问题，才开始我直接采用了普通的单列模式，但是后来在测试的过程中，发现当用户并发量大的时候还是会出现上面的问题，之后我再次考虑了代码，最后发现是因为没有给单列模式加锁的原因，从而导致了大用户并发的时候，线程安全的问题，之后我便在方法上加上了synchronized关键字，解决上述的问题，但是后来测试人员反馈，觉的这段的性能有问题，我考虑之后便采用在方法体内加锁并结合双重判定的方式解决了上面的问题。我们是将数据在tomcat启动的时候加载到缓存中，之后用户进行查询的时候直接从缓存中获取数据，根据前缀匹配进行查询，将结果返回给用户。这样在提高用户体验度的同时也提高性能。

# 缓存概述

应用程序为了提高性能，可以通过使用缓存来达到目的，缓存的存储介质可以内存或者硬盘，通常将数据存储在内存里，确切的说是jvm的内存中，缓存是基于Map这种思想构建的，以键值对的方式进行存取，之所以还可以将缓存的数据存储在硬盘中，是因为内存资源相当有限和宝贵，所以当内存资源不足的时候，就可以将其存储到硬盘中，虽然硬盘的存取速度比内存要慢，但是因为减少了网络通信量，所以还是提高程序的性能。缓存可以分为客户端缓存和服务器端缓存，所谓的客户端缓存通常指的是IE浏览器的缓存，服务器端缓存指的web服务器的缓存，通常可以通过第三方组件实现，如oscache,memcache。

我们通常将那些频繁访问但是又不是经常改变的数据进行缓存。为了保证缓存数据的有效性，在数据发生改变的时候，我们要刷新缓存，避免脏数据的出现。刷新缓存的策略有两种，一种是定时刷新，一种手动刷新。

缓存的层次如下:jsp-->action-->service(通常放置在service)-->dao,缓存越靠前对性能的提升越大

缓存的策略:(缓存空间不足需要进行清理的时候使用)

LRU:最近最少使用原则.(理解:存储书)

FIFO:先进先出的缓存策略.(理解:排队)

你来说说缓存？说说你对缓存的理解（如果遇到重复的，就可以省略）

我们在项目中使用缓存的目的是为了提高应用程序的性能，减少访问数据库的次数，从而提高应用程序的吞吐量。我们通常将权限，菜单,组织机构这些频繁访问但是不经常改变的基础数据进行缓存，其中我在做()某某项目的时候就通过oscache对ZTree的树形菜单进行了缓存，并且在做的时候和单列设计模式进行结合，考虑到多线程下的安全问题，还对单例模式加入了双重判定锁的检查方式。

# 实现页面静态化业务场景

我们在做某项目时，涉及到程序访问的性能问题，这时候我们想到可以通过静态化来提高用户访问时候的性能，所以我们就采用了freemarker模板引擎，考虑到页面也是要有动态的变化的，所以我们采用spring定时器在每天晚上2点钟的时候定时再次生成html静态页面，考虑发布时候的性能问题，我们又采取线程池技术，让多个线程同时发布，从而缩减发布时间。

# servlet线程安全描述

servlet是单列的，对于所有请求都使用一个实例，所以如果有全局变量被多线程使用的时候，就会出现线程安全问题。

解决这个问题有三种方案:

1.实现singleThreadModel接口，这样对于每次请求都会创建一个新的servlet实例，这样就会消耗服务端内存，降低性能，但是这个接口已经过时，不推荐使用。

2.可以通过加锁(synchroniezd关键字)来避免线程安全问题。这个时候虽然还是单列，但是对于多线程的访问，每次只能有一个请求进行方法体内执行，只有执行完毕后，其他线程才允许访问，降低吞吐量。

3.避免使用全局变量，使用局部变量可以避免线程安全问题，强烈推荐使用此方法来解决servlet线程安全的问题。

# (jbpm4)工作流引擎描述:

JPBM是JBOSS旗下的一个开源的基于hibernate的工作流引擎。工作流就是在日常生活中，我们一些常见的如请假流程、采购流程、入职流程，通俗的来讲就是一些在现实生活中的流程以信息化以程序的方式实现。

一个工作流首先需要进行流程定义，流程定义是由节点和跳转组成的，节点又可以称为环节、活动节点、活动环节，并且节点也可以分为两大类型：人工节点和自动节点，人工节点有start开始节点、end结束节点、task任务节点，自动节点有decision判断节点、fork分支节点、join聚合节点和state状态节点，并且一个流程有且只有一个开始节点，但可以有多个结束节点。

流程定义是静止的，它在运行状态时会转换成流程实例，一个流程定义可以对应多个流程实例。流程运行后，会产生两个文件，\*.jdpl.xml文件和\*.png图片文件，也会生成18张数据库表，常用且核心的表有JBPM4\_LOB 存储表，主要存储xml文件和png图片、JBPM4\_TASK 任务表、JBPM4\_EXECUTION 流程实例表、JBPM4\_VARIABLE变量表。

图形化的灵活定制（主动说）

可以根据需求进行流程图的改变的，即定义的流程图是可以根据需要改变的，而不是死的。

可以进行图形化的监控(主动说)

输出图片

获取活动节点的坐标

进行叠加

判断节点:(主动说，也可以了解)

实现implements DecisionHandler接口并重写decide方法,

返回的字符串要和xml中配置的transition的name保持一致。

分支判定节点

JBPM有五大核心类：

ProcessEngine：主要获取各种的Service

RepositoryService：主要发布流程定义

ExecutionService：主要操作流程实例

TaskService：主要操作人工服务

HistoryService：主要操作历史服务。

核心方法：

读取jbpm定义的文件生成zip包存到lob表中：createDeployment()

获取流程定义列表：createProcessDefinitionQuery

根据定义的key或id来启动流程实例：startProcessInstanceByKey(id)

获取待办任务列表：findPersonalTasks(userName)

完成指定任务列表：completeTask(\*.getActivityId())

获取历史任务列表：createHistoryTaskQuery()

获取流程实例的ID：task.getExecutionId()

(了解的表)

JBPM4\_HIST\_ACTINST 流程活动(节点) 实例表

JBPM4\_HIST\_DETAIL 流程历史详细表

JBPM4\_HIST\_PROCINST 流程实例历史表

JBPM4\_HIST\_TASK 流程任务实例历史表

JBPM4\_HIST\_VAR 流程变量( 上下文) 历史表

# JPBM业务场景

首先进行请假的流程定义，我们流程的定义是（员工提交请假单---》经理审批---》总监审批---》总经理审批---》结束），通过repositoryService将其发布部署到jbpm4\_lob表中，之后获取流程定义列表，选中请假的流程定义，员工开始进行请假单的填写，保存并通过executionService开启流程实例，然后用taskService获取经理的待办任务列表，选中待办任务，进行审批，通过调用taskService.completeTask()进入到总监审批环节，然后用总监进行登录，同样获取待办任务列表，然后调用taskService.completeTask()进入总经理审批环节，总经理审批之后，结束流程。在这个过程中我们还可以根据historyService查看当前登录人已办的任务列表。

# Ant描述

Ant是apache旗下的对项目进行自动打包、编译、部署的构建工具，他主要具有轻量级并且跨平台的特性，而且基于jvm，默认文件名为build.xml

Ant主要的标签：

Project 根标签，

target 任务标签，

property 属性标签，自定义键/值 供多次使用，

java 执行编译后的java文件

javac 编译java文件

war 打成war包

其它标签：copy，delete，mkdir，move，echo等。

# 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。

# webService描述

（主动说）

webservice是SOA（面向服务编程）的一种实现，主要是用来实现异构平台通信也就 是不同平台不同项目之间的数据传输，从而避免了信息孤岛的问题，它之所以能够进行异构平台通信是因为它是完全基于xml的，所以说，webService是跨平台，跨语言，跨框架的，在java中通常有三种技术框架分别是xfire,cxf,axis2。我们为了保证 webservice的安全性，采用了基于WS-Security标准的安全验证(使用回调函数)。

（没必要主动说）

webservice的三要素分别是：

wsdl（webservice description language）

用来描述发布的接口（服务）

soap(simple object access protocol)

是xml和http的结合，是webservice数据通信的协议

uddi 用来管理，查询webService的服务

（没必要主动说）

webservice的具体三种实现方式（框架）或者三种实现框架的区别

1. Axis2:可以用多种语言开发，是一个重量级框架，功能非常强大，但是它的性能比较低。

2. Xfire：它相比Axis2来说是一个轻量级框架，它的性能要比Axis2高。

3.cxf：是Xfire的升级版，就好比是，struts2是webwork的升级，然后cxf和spring集成起来非常方便，简易，性能方面也要比Xfire高。

【注】jdk6 自带的webservice jws

（主动说）

业务场景

我在以前做项目的时候，其中遇到一个功能，需要进行两个项目之间的数据的传输，项目经理让我去完成这个任务，我根据以往的项目经验，想到两种解决方案，第一种就是开放另外一个项目的数据库的权限给我，然后我直接通过访问另外一个项目的数据库，来得到需要的信息，但后来我分析了下，觉的这种方式不安全，而且因为当时这个项目是另外一家公司负责在做，所以数据库里面的表结构，以及以后牵涉到的责任问题都很多，所以我就采用了第二种方案，即通过webservices的方式，进行异构系统之间数据信息的传递，webservices的具体实现，有xfire,cxf,axis2,我根据以往的项目经验，了解到cxf是xfire的升级版本，适用于java语言，xfire/cxf 性能比axis2要高，并且和spring整合起来也比较方便，而axis2支持更多的语言，性能相对于cxf要低，通过上面分析，结合我们目前的两个项目都是基于java语言的，所以我采用cxf这种方式实现了两个项目之间数据的传递，我们为了保证webservice的安全性我们采用了基于WS-Security标准的安全验证(使用CXF回调函数)。

（没必要主动说）

webservice服务端配置流程

首先在web.xml中引入cxfServlet核心类，指定对以/cxf开头的url路径提供webservice服务，之后我们在要发布成webservice接口上添加@Webservice 注解，而且还要在实现类上添加同样的webservice注解并且要说明实现了哪个接口，之后在spring-webservice.xml中发布webservice服务，通过jaxws:endpoint这个标签，并且在标签配置implementor和address来表明实现服务的类，以及发布的地址，最后在浏览器中输入相关的webservice地址?wsdl来验证服务是否发布成功。

（没必要主动说）

webservice客户端的配置

首先通过wsdl2java根据发布的webservice服务端地址的wsdl生成客户端调用的中间桥梁java类，将生成的java类拷贝到客户端项目中，配置spring-client.xml文件，通过jaxws:client定义一个bean,并通过address属性指明要访问的webservice的服务地址，通过serviceClass指明充当中间桥梁的服务类，之后获取该bean,就可以通过它来访问发布的webservice接口中的方法。

# oracle索引概述

索引是与表相关的一个可选结构，可以提高sql语句的检索效率，相当于我们的字典目录 ，可以快速进行定位 ，所以可以减少磁盘I/O，但是因为索引在物理与逻辑上都是独立于表的数据它会占用一定的物理空间(额外磁盘空间) 所以并不是索引越多越好而我们应该根据业务需求去创建索引,而且进行增删改操作时 oracle又要自动维护索引 所以在一定程度上也降低了维护速度，而且我们在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，我们一般创建索引呢是这样创建的 create index 索引名 on 表名(字段)，索引又分为普通索引 唯一索引(unique) 单个索引 复合索引(又叫组合索引，在索引建立语句中同时可包含多个字段名)，顺序索引，散列索引,位图索引。

# oracle存储过程

存储过程就是封装一些sql的集合，也就是一条条的sql语句，过程的优点就是简化了sql命令加上它是预编译的，所以它的执行效率和性能较高，再者，如果不调用过程的话就要和数据库发生多次交互，调用过程只需传一个命令所有的那些执行逻辑都在数据库端执行，所以说它降低了网络的通信量，其次，存储过程大大提高了安全性，这就是优点。

缺点呢，就是不同的数据库对过程支持的关键字支持的关键字都是不一样的，所以它的移植性是非常差的，再者，它的维护性难度也比较大，因为它没有专业的调试和维护工具，所以说它维护起来比较麻烦，这就是存储过程的基本概述。

# Junit 业务场景

在我们开发项目的时候为了提高代码的性能和保证逻辑正确性，在我们编写代码后往往都要进行单元测试，来验证代码，当时我们公司开发人员全部使用的main方法来进行验证，但是使用mian的最大缺点就是不能将多个类同时进行验证，验证的结果不直观，测试复杂（每个类都要写main方法，单个运行），一定程度上浪费时间，所有我和项目经理提议使用专业测试工具Junit来进行测试，因为Junit是一个Java语言的单元测试框架 ，测试简单，不仅可以提供工作效率和代码的质量，也提高团队的合作能力，我提议后我们进行了Junit的培训使用Junit4加注解的方式来测试。

# Apache+Tomcat 实现负载均衡及seesion复制

当我们tomcat访问量大,线程连接数不够时,我们考虑到了tomcat的负载均衡来分担过多的访问.性能方面负载均衡也能利用多台tomcat来增大内存量,**流程**,准备工作apache,Jk\_mod,tomcat,在apache的conf/httpd.conf文件中 使用include 标签引入我们自定义的一个mood\_jl.conf,在modules中引入下载的k\_mod-apache-X.X.XX.so文件,在其中引入我们的.so,及work.properties文件,及指定负载分配控制器controller,在work.properties文件中worker.list=controller,tomcat1,tomcat2指定service,worker.tomcat1.port Ajp端口号,type 是ajp,host为指定ip,lbfactor 指定分配权重值越大分担请求越多,worker.controller.type=lbworker.controller.balanced\_workers=tomcat1,tomcat2 指定分担请求的tomcat Session的复制在tomcat中service.xml中Engine标签加入 jvmRoute 值为work,properties中指定的tomcat名称,然后打开<Cluster标签的注释，最后在应用中程序的web.xml文件中增加<distributable/>。

我们在做这个项目时，我们考虑到服务器性能的问题，我们最开始想到使用纵向扩展，来增加硬件的配置提高其性能，但这样做比较耗费资金，而且服务器内存空间也是有限的；所以后来就想到使用横向扩展来达到这一目的。

当时我们的apache是通过jk借助于ajp协议与tomcat进行通信的，在我们不进行负载均衡之前，那所有的请求都由一台tomcat进行处理，这样会使我们的tomcat所承受的压力增大，而我们进行负载均衡之后，同样数量的请求经过apache和jk将其分发到多台tomcat进行处理，从而降低每台tomcat所承受的压力,而且当其中一台机器宕机时，其他机器还可以继续提供服务，保证服务不间断。

在这个过程中，我们遇到了session问题，然后我此昂到用session复制来解决这个问题；在apache的配置文件中增加session粘带特性:

worker.lb.sticky\_session=1

worker.lb.sticky\_session\_force=0

Tomcat的配置

修改server.xml文件：

<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat2">增加jvmRoute=”tomcat2” \*. jvmRoute赋的值为worker.properties中配置的相应的server名一致

<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/> 将此配置的注释去掉

修改应用的web.xml文件

在应用中的web.xml文件中增加<distributable/>。

如果这样做，当第一次访问的时候，会把所以数据全部缓存到第一台服务器上，通过web配置文件，会把第一台缓存的数据全部复制到第二胎服务器上，这样做就加大网路通信量，导致阻塞，所以我们就想到了可以通过memcached分布式缓存来存取session从而解决上述问题。

# Ant业务场景

Ant是基于java语言编写的，因此具有跨平台的特性，此外还具有简洁方便，灵活配置的特性，因此我就在XX项目中使用ant进行项目的编译，打包，部署操作。使用ant之后，如果我们在客户那里修改代码后，就可以直接使用ant进行编译，打包，部署，而不需要为了编译，打包，部署专门在客户那里安装eclipse.此外使用ant也可以直接和svn进行交互，下载源码的同时进行编译，打包，部署。

# maven业务场景

maven业务场景

前段时间在研究maven，知道maven是一个项目管理工具，其核心特点就是通过maven可以进行包的依赖管理，保证jar包版本的一致性，以及可以使多个项目共享jar包，从而能够在开发大型j2ee应用的时候，减小项目的大小，并且和ant比起来，maven根据“约定优于配置”的特性，可以对其项目的编译打包部署进行了更为抽象的封装，使得自己不需要像ant那样进行详细配置文件的编写，直接使用系统预定好的mvn clean,compile,test,package等命令进行项目的操作。于是我就在XX项目中采用了maven,为了保证团队中的成员能够节省下载jar包所需要的时间，于是我就采用nexus搭建了在局域网内的maven私服，然后通过配置settings.xml中建立mirror镜像，将所有下载jar包的请求都转发到maven私服上，之后通过在pom.xml即(project object model)中配置项目所依赖的jar包，从而达到在构建项目的时候，先从本地仓库中查找，如果不存在从内部私服查找，如果不存在最后再从外网central服务器查找的机制，达到了节省下载带宽，提高开发效率，以及jar包重用的目的。

ant业务场景

ant是基于java语言编写的，因此具有跨平台的特性，此外还具有简洁方便，灵活配置的特性，因此我就在XX项目中使用ant进行项目的编译，打包，部署操作。使用ant之后，如果我们在客户那里修改代码后，就可以直接使用ant进行编译，打包，部署，而不需要为了编译，打包，部署专门在客户那里安装eclipse.此外使用ant也可以直接和svn进行交互，下载源码的同时进行编译，打包，部署。

maven的常用命令

mvn eclipse:clean eclipse:eclipse -Dwtpversion=2.0

mvn clean package

maven的生命周期是独立的，但是生命周期下的阶段是相互关联并且延续的。

maven的生命周期

clean(清理):clean;default(默认):compile,test,packageinstall;site(站点)

# bugfree的操作步骤

我们在使用bugfree的时候我们首先登陆的时候是以测试员的身份登陆的，也就是系统管理员用户；测试员在登陆后首先应该给要测试的项目的相关负责人，每人创建一个账号（也就是在登陆后的页面的后台管理中创建用户），用户都新建完成之后就新建组，把要测试的项目的用户添加到组中。最后就新建项目并且新建该项目的模块。新建完项目之后就是开始测试程序，在程序中遇到bug以后就把错误截图，在到bugfree中新建bug填写相关的信息和要指派的人（出错模块的负责人）和把刚才的错误截图作为附件一并传送过去。

开发人员每天早上上班的第一件事就是用自己的用户登录bugfree，然后输入查询条件看看前一天有没有指派给自己的bug需要解决的如果有就进行解决。

开发人员把对应的bug解决之后就去bugfree上把bug对应的状态改成已解决状态，然后进行保存提交，这样bug的状态就变成已解决状态。测试人员上线查看已解决状态的bug并再次进行测试，如果经过测试bug的问题已解决，就可以把bug关闭；如果经过测试，发现仍然存在bug，就把bug激活；这样等开发人员再次登录的时候就可以再次看到这个未解决的bug，再次进行解决，如此反复直到bug全部解决，因为bugfree对bug的修改都有保留，所有我们可以看到bug的一步步的完善，直到最后把bug关闭。

Bug的三种状态：未解决(Active)（测试人员）、已解决(Resolved)(开发人员)、关闭(Closed)（测试人员）

# Axis2 的配置

axis2服务端配置流程

1.引入相关的jar包并且在web.xml中配置axis2的核心控制器 axisServlet

2.在web-inf下建立相关的三层文件夹结构:

services-->自定义文件夹名-->META-INF-->servies.xml

3.在servies.xml中配置service的name以及对应的springBeanName

4.在浏览器中输入webservice的服务端地址并加上?wsdl来进行测试，看是否发布成功

axis2客户端配置流程

通过wsdl2java根据webservice服务端的url生成客户端代码将代码引入项目的文件夹中进行正常访问

# spring定时器

每隔固定的时间执行

1.建立一个triggers触发器集合

2.建立SimpleTriggerBean并且指定每次间隔的时间以及执行的次数以及要执行的目标

3.通过 targetObject以及targetMethod找到要执行的具体类的具体方法

目标对象是一个普通的java类

每到指定的时间执行

1.建立一个triggers触发器集合.

2.建立CronTriggerBean指定cron表达式以及要执行的目标

3.通过 targetObject以及targetMethod找到要执行的具体类的具体方法

目标对象是一个普通的java类

# Ext概述

据我了解Ext是一个用js编写RIA框架，它可以和各种后台语言结合使用。我在项目中用Ext来完成的模块大概情况是这个样子，首先我通过layout等于border的这种方式来进行布局，分为上下左右中，然后在左边用ext tree来进行菜单的展示，之后在中间区域通过tabs来加入选项卡，而在选项卡中就是一个个的grid以及form，其中我在做grid的时候，首先通过store来存取后台返回的符合model格式数据集，store是通过proxy和后台的contoller进行交互，之后把store赋值给grid的store属性并且通过renderTO在指定的位置进行渲染展示。

Grid问题:

当时我在做grid的时候，发现数据没有展示出来，我通过f12进行跟踪，发现压根就没有发送请求，后来我分析了下，发现因为没有调用store的loadPage方法，所以导致了这个问题。除此之外在我们做项目的过程中，我手底下带的一个人同样在负责grid的时候，数据可以正常展示，但分页信息没有展示，通过跟踪他的代码发现是因为他没有把store属性赋值给分页工具条，所以才导致了这个问题。

tabs选项卡：

当我在做tab选项卡这一模块的时候，我首先在加载页面的时候用TabPanel创建了一个tab页面，让它展示在中间位置，然后点击左边Tree菜单调用add方法动态添加一个个的tab选项卡，但是做的过程中出现了相同的选项卡会重复添加的问题，我查了一些相关资料，最后通过tab的id或者一个唯一标识判断tab是否选中，如果选中则调用setActiveTab来激活该选项卡，让它选中，否则就添加一个tab。最后达到了tab不存在就添加，存在就选中的效果。

了解:

Ext4.0也支持前端的MVC开发模式.

为啥没采用mvc的开发模式？

我们当时因为时间方面的原因，项目经理就决定用普通的这种开发模式进行开发，并没有采用Ext4.0这种mvc模式的特性。但我认为他们的核心操作流程是一致的所以对我来说去学习和使用这种方式并没有什么难度。

# lucene的概述

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。

# 线程池作用

1.减少了创建和销毁线程的次数，每个线程都可以被重复利用，可执行多个任务。

1. 可以根据系统的承受能力，调整线程池中线程的数目，防止因为消耗过多的内存，而导致服务器宕机(每个线程需要大约1MB内存，线程开的越多，消耗的内存也就越大，最后宕机)。

通常我们使用的线程池是实现了ExecutorService的ThreadPoolExecutor。

# jbpm是如何和spring进行整合

1. 通过在spring-common.xml配置文件中配置springHelper，通过springHelper创建processEngine，再通过processEngine获取各种工作流的Service，如repositoryService，executionService，historyService，taskService
2. 在src根目录下新建jbpm.cfg.xml文件

# Tomcat优化(记录重点中)

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

ompression="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" />

# memcached的介绍

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）

# 优化篇

# 代码优化

代码结构层次的优化(目的:更加方便代码的维护--可维护性，可读性)

1.代码注释(代码规范)

2.工具类的封装(方便代码的维护，使代码结构更加清晰不臃肿，保证团队里代码 质量一致性)

3.公共部分的提取

代码性能的优化(目的:使程序的性能最优化)

1.使用一些性能比较高的类(bufferInputStream)

2.缓冲区块的大小(4k或者8k)

3.公共部分的提取

4.通常要用stringbuffer替代string加号拼接

# 业务优化

我们做项目的时候业务优化这方面最主要是从用户体验度角度进行考虑,减少用户操作的步骤提高工作效率，通常有以下几种：

1.可以通过tabindex属性来改变tab键盘的操作顺序

2.可以通过回车键来进行搜索或者提交操作

3.对于单选按钮和复选按钮可以通过操作后面的文本来选择前面的单选按钮从及复选按钮

4.添加的信息要按照id倒序进行排列

5.进行搜索操作时加入js loading操作（不仅告诉用户所进行的请求正在被处理，而且防止用户多次点击提交操作）

6.当进行删除操作的时候要弹出提示框，警告用户要进行删除操作，是否确认。

7.根据returnURL在用户登录成功后直接跳到想要访问的资源。

8.进行删除操作时通过confirm提示用户是否确认删除操作，操作完后提示操作是否成功。

9.减少用户操作的步骤

10.使用autocomplete插件快速进行搜索

11.通过ztree，以及kindeiditor来提高用户的体验度

必背，必做:

1.可以通过回车键来进行搜索或者提交操作

2.添加的信息要按照id倒序进行排列

3.进行搜索操作时加入js loading操作（不仅告诉用户所进行的请求正在被处理，而且防止用户多次点击提交操作）

4.当进行删除操作的时候要弹出提示框，警告用户要进行删除操作，是否确认,如果删除成功则弹出提示框告诉用户。

5.减少用户操作的步骤

6.通过ztree，以及kindeiditor来提高用户的体验度。

# sql优化(记录重点中)

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。

# 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特定的权限，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](http://baike.baidu.com/view/94238.htm)（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。

# 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

# 深入java虚拟机以及大数据

**1、jvm的相关概念**

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。

heap(堆)，stack(栈)

jvm的结构细分及其概述?

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。

jvm 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。

GC是什么，为什么要有GC？

GC就是垃圾回收，java这种语言是动态分配内存大小的，并且依靠垃圾回收机制来完成对分配内存空间的回收，从而来避免内存溢出的问题，也在一定程度上降低了程序员工作的复杂度。

jvm中的GC采用了generation（分代回收）算法，因为大多数的对象存活的时间比较短，而少部分的对象才能够长时间存活。因此，jvm将堆内存划分为年轻代（young generation）和年老代（old generation）。年轻代中的对象通常建立时间不久，且大部分生命周期也很短；年老代中的对象则已经创建比较久了，其声明周期也相对年轻代比较长。按照上面的划分，jvm在做GC时也进行了区别对待，对年轻代GC会相对比较频繁，且采用了copying(复制)算法；年老代的GC相对比较少，且采用的是tracing算法的一种，是标记-清除-压缩。

JVM内存限制(最大值)

JVM内存的最大值跟操作系统有很大的关系。

简单的说就32位处理器虽然 可控内存空间有4GB,但是具体的操作系统会给一个限制，这个限制一般是2GB-3GB（一般来说Windows系统下为1.5G-2G，Linux系统 下为2G-3G），而64bit以上的处理器就不会有限制了。

Java 监视和管理控制台：

JConsole 使您能够在运行时监视各种 JVM 资源统计信息。这种特性特别适用于检测死锁、内存泄漏。它可以连接到一个本地或远程 JVM 并可用来进行监视：

线程状态（包括相关的锁）

内存使用情况

垃圾收集

运行时信息

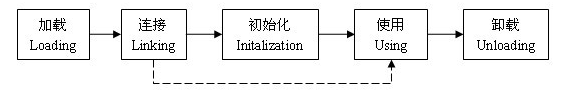
JVM 信息

jvm的调优?

开启-Server模式，增大堆的大小，以及持久代的大小，从而提高程序的运行效率，并且将初始堆大小和最大堆大小设置为一样的值从而避免了堆增长会带来额外压力。持久代大小的设置同理，也设置为初始大小和最大大小一样大。

jvm的类加载机制? jvm中类的生命周期？

生命周期：加载、连接、初始化，使用，卸载



对象基本上都是在jvm的堆区中创建，在创建对象之前，会触发类加载（加载、连接、初始化），当类初始化完成后，根据类信息在堆中实例化类对象，初始化非静态变量、非静态代码以及默认构造方法，当对象使用完之后会在合适的时候被jvm垃圾收集器回收。

要经过三步：加载（Load），链接（Link），初始化（Initializ）。其中链接又可分为校验（Verify），准备（Prepare），解析（Resolve）三步。ClassLoader就是用来装载的。通过指定的className，找到二进制码，生成Class实例，放到JVM中。

ClassLoader从顶向下分为：

Bootstrap ClassLoader:引导类加载器，它负责加载Java的核心类(如rt.jar)

Extension ClassLoader:扩展类加载器, 它负责加载JRE的扩展目录（JAVA\_HOME/jre/lib/ext）中的JAR包

System ClassLoader:系统（也称为应用）类加载器， 它负责在JVM被启动时加载来自在命令java中的-classpath中的JAR包

User-Defined ClassLoader：用户自定义的类加载器

linux中的命令：

ps -ef | grep :查看进程信息

vi:文件编辑命令

more:分页查看命令

top:常用的性能分析工具,能够实时显示系统中各个进程的资源占用状况

ifconfig:显示或配置网络设备的命令

ping:它通常用来测试与目标主机的连通性

rsync、scp：文件同步命令

**2、云计算+大数据的具体技术实现方案：**

Hadoop是一个能够对大量数据进行分布式处理的软件框架。它以并行的方式工作，通过并行处理加快处理速度，维护多个工作数据副本，具有可伸缩性，能够处理 PB 级数据.

hadoop 由许多元素构成。其最底部是HDFS，它存储 Hadoop 集群中所有存储节点上的文件。

HDFS的上一层是MapReduce 引擎。

hadoop下的子项目:

HDFS:Hadoop分布式文件系统

MapReduce：并行计算框架（建立在HDFS上的）

HBase: 类似Google BigTable的分布式NoSQL 列数据库

Hive：数据仓库工具

Zookeeper：分布式锁设施

Pig: 大数据分析平台，为用户提供多种接口

行业知识(了解):

存储容量：是该存储设备上可以存储数据的最大数量，通常使用千字节（kb kilobyte）、兆字节（MB megabyte）、吉字节（GB, gigabyte）、太字节（TB ，terabyte）和PB(Petabyte)、EB(Exabyte)等来衡量。

1KB=2(10)B=1024B； 括号中的数字为2的指数(即多少次方)

1MB=2(10)KB=1024KB=2(20)B；

1GB=2(10)MB=1024MB=2(30)B。

1TB=2(10) GB=1024GB=2(40)B

1PB=2(10) TB=1024TB=2(50)B

1EB=2(10) PB=1024PB=2(60)B

1Byte(相當於一個英文字母，您的名字相當6Bytes(6B)。

Kilobyte(KB)=1024B相當於一則短篇故事的內容。

Megabyte(MB)=l024KB相當於一則短篇小說的文字內容。

Gigabyte(GB)=1024MB相當於貝多芬第五樂章交響曲的樂譜內容。

Terabyte(TB)=1024GB相當於一家大型醫院中所有的X光圖片資訊量。

Petabyte(PB)=l024TB相當於50%的全美學術研究圖書館藏書資訊內容。

Exabyte (EB)=1024PB；5EB相當於至今全世界人類所講過的話語。

Zettabyte(ZB)=1024EB如同全世界海灘上的沙子數量總和。

Yottabyte(YB)=1024ZB相當於7000位人類體內的微細胞總和。