目录

基础篇

一、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

二、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。

三、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。

四、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.

五、Hashtable 与 HashMap 的区别

HashMap 不是线程安全的，HashTable 是线程安全。
HashMap 允许空（null）的键和值（key），HashTable 则不允许。
HashMap 性能优于 Hashtable。

Map
1.Map 是一个以键值对存储的接口。Map 下有两个具体的实现，分别是 HashMap 和 HashTable.
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 允许键或值为空，而 HashTable 不允许键或值为空.

六、九大隐式对象

输入/输出对象： request response out
作用域通信对象： session application pageContext
Servlet 对象： page config
错误对象： exception

七、Forword(请求转发)与 Redirect(重定向)

1、从数据共享上
Forword 是一个请求的延续，可以共享 request 的数据

Redirect 开启一个新的请求，不可以共享 request 的数据

2、从地址栏

Forword 转发地址栏不发生变化

Redirect 转发地址栏发生变化

八、JQurey 总结

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 ，dom 操作 ，ajax(ajax 不能跨域) ，特效，工具类

九、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:

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);

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。

十、request.getSession()、reqeust.getSession(false)和 request.getSession(true)

getSession()/getSession(true)：当 session 存在时返回该 session，否则新建一个 session 并返回该对象
getSession(false)：当 session 存在时返回该 session，否则返回 null

十一、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。

十二、Ajax 总结

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

Ajax 的优点:
提高用户体验度(UE)
提高应用程序的性能
进行局部刷新
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。
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

十三、JSP9 大隐视对象中四个作用域的大小与作用范围

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)。

十四、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。

十五、java 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)
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)

十六、冒泡排序

```
public class Sort {
    public static void sort() {
```

```
        Scanner input = new Scanner(System.in);
        int sort[] = new int[10];
        int temp;
        System.out.println("请输入 10 个排序的数据：");
        for (int i = 0; i < sort.length; i++) {
            sort[i] = input.nextInt();
        }
        for (int i = 0; i < sort.length - 1; i++) {
            for (int j = 0; j < sort.length - i - 1; j++)                {
                if (sort[j] < sort[j + 1]) {
                    temp = sort[j];
                    sort[j] = sort[j + 1];
                    sort[j + 1] = temp;
                }
            }
        }
        System.out.println("排列后的顺序为：");
        for(int i=0;i<sort.length;i++){
            System.out.print(sort[i]+"    ");
        }
    }
    public static void main(String[] args) {
        sort();
    }
}
```

十七、二分查找法

十八、时间类型转换

```
public class DateFormat {
    public static void fun() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy 年 MM 月 dd 日");
        String newDate;
        try {
            newDate = sdf.format(new SimpleDateFormat("yyyyMMdd")
                    .parse("20121115"));
            System.out.println(newDate);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
    public static void main(String args[]) {
```

```
            fun();
        }
}
```

十九、阶乘

```
public class Multiply {
    public static int multiply(int num) {
        if (num < 0) {
            System.out.println("请输入大于 0 的数！");
            return -1;
        } else if (num == 0 || num == 1) {
            return 1;
        } else {
            return multiply(num - 1) * num;
        }
    }
    public static void main(String[] args) {
        System.out.println(multiply(10));
    }
}
```

二十、UE 和 UI 的区别

UE 是用户体验度
UI 界面原型（用户界面）（相当于买房时用的模型）

设计 UI 的作用：
1、帮助程序员工作（界面已由美工设计完成）
2、提前让用户对项目有个宏观的了解，知道效果是什么样子。

二十一、osi 七层模型

第一层：物理层
第二层：数据链路层
第三层：网络层
第四层：传输层
第五层：会话层
第六层：表示层
第七层：应用层

二十二、线程和进程的区别

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。

3.实现线程的两种方式：继承 Thread 类，实现 Runable 接口

二十三、jvm 的内存结构

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。

二十四、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。

二十五、单例

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

二十六、解析 xml 文件的几种技术

1、 解析 xml 的几种技术

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

1.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。

2.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。

3.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。

2、dom4j 与 sax 之间的对比：【注：必须掌握！】

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

二十七、项目的生命周期

1.需求分析
2.概要设计
3.详细设计(用例图，流程图，类图)
4.数据库设计(powerdesigner)
5.代码开发（编写）
6.单元测试（junit 白盒测试）(开发人员)
svn 版本管理工具(提交，更新代码，文档)
7.集成测试 （黑盒测试，loadrunner（编写测试脚本）(高级测试)）
8.上线试运行 （用户自己体验）
9.压力测试（loadrunner）
10.正式上线
11.维护

二十八、OSCache 的判断

```
Object obj = CacheManager.getInstance().getObj("oaBrandList");
//从缓存中取数据
if (null == obj)  {
        obj = brandDao.getBrandList();
        //如果为空再从数据库获取数据
         //获取之后放入缓存中
```

```
            CacheManager.getInstance().putObj("oaBrandList", obj);
        }
        return (List<OaBrand>)obj;
```

二十九、经常访问的技术网站

1.csdn(详细步骤的描述)
2.iteye(详细步骤的描述)
3.oschina（开源中国获取 java 开源方面的信息技术）
4.java 开源大全 www.open-open.com(获取 java 开源方面的信息技术)
5.infoq(对 java,php,.net 等这些语言的一些最新消息的报道)

三十、项目团队中交流的工具

飞秋(局域网)　　qq(局域网，外网)

RTX（局域网，外网）邮箱 （局域网，外网）

三十一、平时浏览的书籍

实战经验：
*** in action(实战)
*** 深入浅出
*** 入门指南
思想基础:
大话设计模式　　重构

三十二、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。

2、运行时异常和非运行时异常

运行时异常: 都是 RuntimeException 类及其子类异常：

IndexOutOfBoundsException 索引越界异常
ArithmeticException：数学计算异常
NullPointerException：空指针异常
ArrayOutOfBoundsException：数组索引越界异常
ClassNotFoundException：类文件未找到异常
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

三十三、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，所以会把很重要的信息存储在 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)

三十四、字节流与字符流的区别

stream 结尾都是字节流，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。
实际使用通常差不多。
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。

三十五、final,finally,finalize 三者区别

Final 是一个修饰符：
当 final 修饰一个变量的时候，变量变成一个常量，它不能被二次赋值
当 final 修饰的变量为静态变量（即由 static 修饰）时，必须在声明这个变量的时候给它赋值
当 final 修饰方法时，该方法不能被重写
当 final 修饰类时，该类不能被继承
Final 不能修饰抽象类，因为抽象类中会有需要子类实现的抽 象方法，（抽 象类中可以有抽象方法，也可以有普通方法，当一个抽象类中没有抽象方 法时，这个抽象类也就没有了它存在的必要）
Final 不能修饰接口，因为接口中有需要其实现类来实现的方法
Finally：
Finally 只能与 try/catch 语句结合使用，finally 语句块中的语句一定会执行， 并 且 会 在 return，continue，break 关键字之前执行
finalize：
Finalize 是一个方法，属于 java.lang.Object 类，finalize()方法是 GC （garbage collector 垃圾回收）运行机制的一部分，finalize()方法是在 GC 清理它所从 属的对象时被调用的

三十六、Io流的层次结构

从流的方向
输入流 输出流

从流的类型上
字符流 字节流

inputstream 和 outputstream 都是抽象类

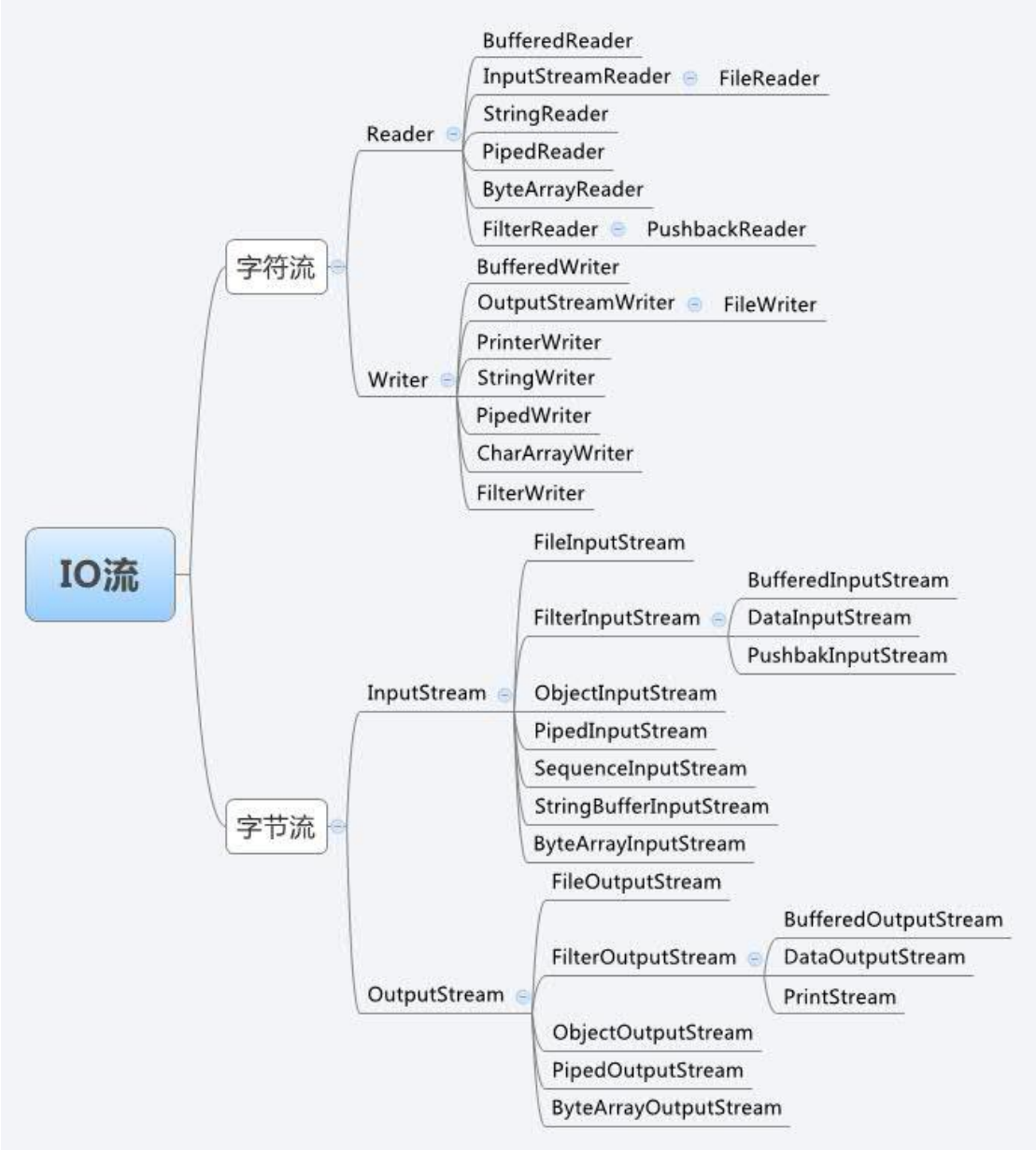
它们下面的实现包括

FileInputStream,BufferedInputStream

FileOutputStream,BufferedOutputStream

reader 和 writer

FileReader,BufferedReader,StringReader
FileWriter,BufferedWriter,StringWriter,PrintWriter



三十七、JAVA:

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

三十八、JavaSE JavaEE JavaME 区别

是什么：
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
特点：
SE 主要用于桌面程序（swing）,控制台开发(main 程序)。

EE 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。

ME 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

三十九、JDK JRE JVM 的区别：

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

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

四十、报错的状态码：

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 路径
500 内部错误

四十一、协议以及默认的端口号

ftp 21 文件传输协议
Pop3 110 它是因特网 <http://baike.baidu.com/view/1706.htm>电子邮件 <http://baike.baidu.com/view/1524.htm> 的 第 一 个 离 线 <http://baike.baidu.com/view/113466.htm>协议标准
Smtp 25 简单邮件传输协议
http 80 超文本传输协议
oracle 默认端口号 1521
mysql 默认端口号 3306

四十二、抽象类与接口的区别

1.一个类只能进行单继承，但可以实现多个接口。

2.有抽象方法的类一定是抽象类，但是抽象类里面不一定有抽象方法；
接口里面所有的方法的默认修饰符为 public abstract，接口里的成员变 量默认的修饰符为 pulbic static final。

关系	
接口和接口	继承
接口和抽象类	抽象类实现接口
类和抽象类	类继承抽象类
类和类	继承

四十三、修饰符的作用

修饰符的作用范围:

	private	default	protected	public
同一个类中	可以	可以	可以	可以
同一个包的类中		可以	可以	可以
不同包的子类中			可以	可以

不同包的类中 可以

四十四、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。

参考：

1.执行时间 window.onload 必须等到页面内包括图片的所有元素加载完毕后才能执行。$(document).ready()是 DOM 结构绘制完毕后就执行，不必等到加载完毕。 2.编写个数不同 window.onload 不能同时编写多个，如果有多个 window.onload 方法，只会执行一个 $(document).ready()可以同时编写多个，并且都可以得到执行 3.简化写法 window.onload 没有简化写法 $(document).ready(function(){}) 可以简写成$(function(){});

以浏览器装载文档为例，在页面加载完毕后，浏览器会通过 Javascript 为 DOM 元素添加事件。在常规的 Javascript 代码中，通常使用 window.onload 方法，而在 Jquery 中，使用的是 $(document).ready() 方法。 $(document).ready() 方法是事件模块中最重要一个函数，可以极大的提高 Web 应用程序的速度。

	window.load	$(document).ready()
执行时机	必须等待网页中所有的内容加载完毕后（包括图片）才能执行	网页中所有 DOM 结构绘制完毕后就执行，可以能 DOM 元素关联的内容并没有加载完

编写个数	不能同时编写多个
	以下代码无法正确执行：
	window.onload = function(){
	alert("text1");
	};
	window.onload = function(){
	alert("text2");
	};
	结果只输出第二个
		能同时编写多个
		以下代码正确执行：
		$(document).ready(function(){
		alert("Hello World");
		});
		$(document).ready(function(){
		alert("Hello again");
		});
		结果两次都输出

简化写法	无	$(function(){
		// do something
		});

另外，需要注意一点，由于在$(document).ready() 方法内注册的事件，只要 DOM 就绪就会被执行，因此可能此时元素的关联文件未下载完。例如与图片有关的 html 下载完毕，并且已经解析为 DOM 树了，但很有可能图片还没有加载完毕，所以例如图片的高度和宽度这样的属性此时不一定有效。要解决这个问题，可以使用 Jquery 中另一个关于页面加载的方法---load() 方法。Load() 方法会在元素的 onload 事件中绑定一个处理函数。如果处理函数绑定给 window 对象，则会在所有内容（包括窗口、框架、对象和图像等）加载完毕后触发，如果处理函数绑定在元素上，则会在元素的内容加载完毕后触发。 Jquery 代码如下：$(window).load(function (){ // 编写代码 });等价于 JavaScript 中的以下代码 Window.onload = function (){ // 编写代码 }

四十五、switch 默认接受的几种数据类型

Short, int, byte, char

四十六、request 跟 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

四十七、找到解决svn冲突方法

对于svn冲突，可以采用手工处理将冲突的部分进行整合，

之后备份最新整合后的文件，采用覆盖更新的方式处理完

冲突之后，再把最新整合后的文件进行提交。

四十八、反射的描述

通过字符串可以动态创建java对象，并且可以动态访问方法，
属性等。
我们在项目中的时候封装过数据库jdbc的持久层，
其中就利用反射这项
技术来达到通用
和灵活的目的。

框架篇

一、Struts1 的运行原理

在启动时通过前端总控制器 ActionServlet 加载 struts-config.xml 并进行解析，当用户在 jsp 页面发送请求被 struts1 的核心控制器 ActionServlet 接收，ActionServlet 在用户请求时将请求参数放到对应的 ActionForm 对象中的成员变量中，然后 ActionServlet 则会根据 struts-config.xml 中的映射关系找到相应的 Action 中的方法，将对应的 ActionForm 一并传给这个 Action 中的方法里，然后执行相应的业务逻辑操作，最后就根据 ActionMapping 的 findforward 方法返回一个 ActionForward,之后在 struts-config.xml 中找到与之对应的 forward 标签，根据它的配置路径找到对应的 jsp 页面。

二、Struts2 的运行原理

1、tomcat 启动的时候会加载 web.xml 、核心控制器 FilterDispatcher 会加载并解析 struts.xml

2、客户端会发送一个请求到 action 、FilterDispatcher 会根据后缀名进行拦截

3、FilterDispatcher 根据 struts.xml 的配置文件信息 找到 某个 action 对应的某个类里的指定方法

4、执行相关的业务逻辑最后返回 一个 String

5、<action/> 里配置 <result/> name 的属性值与返回的 String 进行匹配,跳转到指定的 jsp 页面

三、struts2 的体系结构

1、客户端向 Servlet 容器（例如 Tomcat）发送一个请求；

2、这个请求经过一系列的过滤器（Filter）；

3、接着 FilterDispatcher 被调用，FilterDispatcher 询问 ActionMapper 来决定这个请求是否需要调用某个 Action；

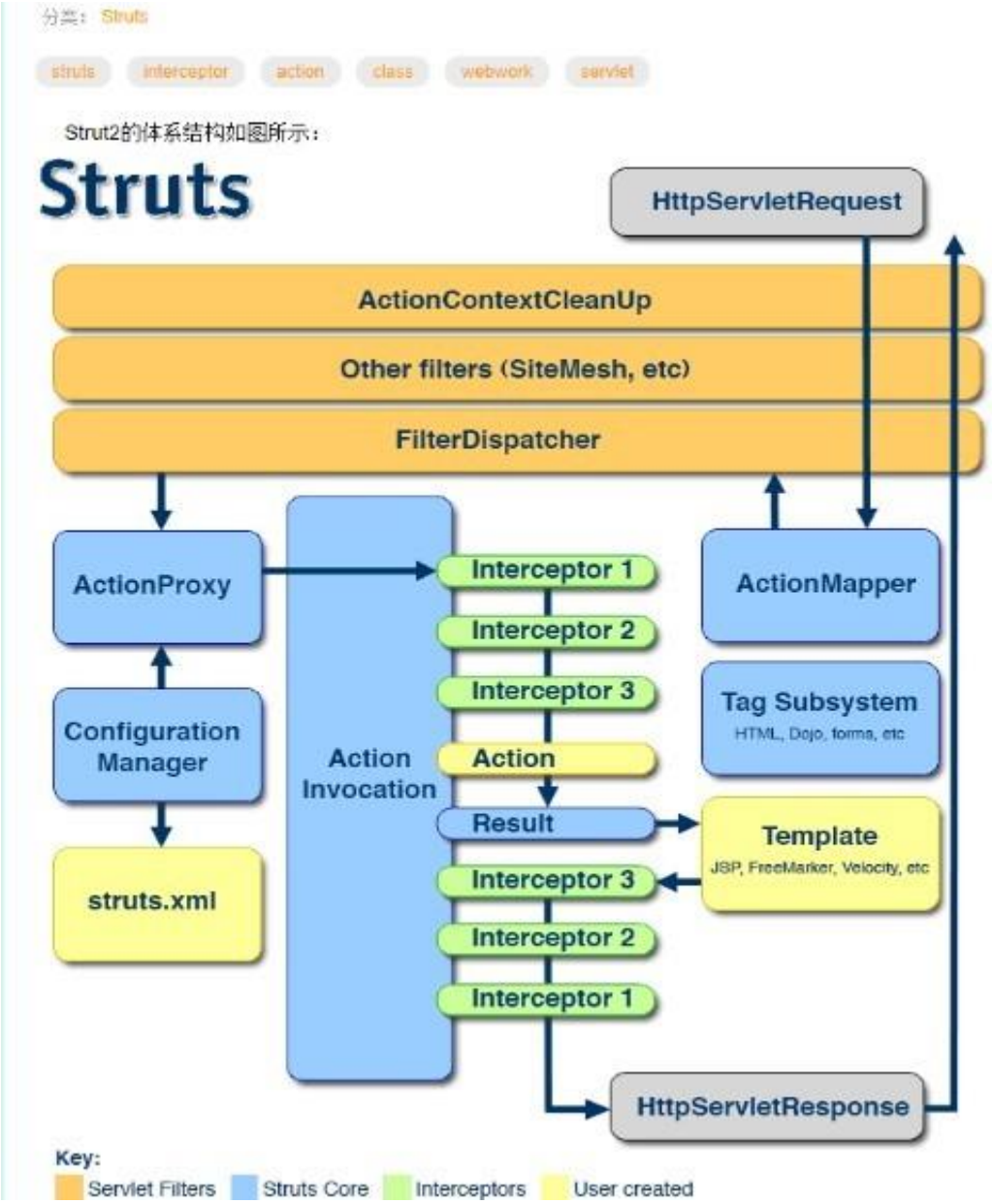
4、如果 ActionMapper 决定需要调用某个 Action，FilterDispatcher 把请求的处理交给 ActionProxy；

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 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 jsp 或者 FreeMarker 的模版。（体系结构图见下一页）



四、Spring MVC 运行原理

整个处理过程从一个 HTTP 请求开始：

1.Tomcat 在启动时加载解析 web.xml,找到 spring mvc 的前端总控制器 DispatcherServlet,并且通过 DispatcherServlet 来加载相关的配置文件信息。

2.DispatcherServlet 接收到客户端请求，找到对应 HandlerMapping，根据映射规则，找到对应的处理器（Handler）。

3.调用相应处理器中的处理方法，处理该请求后，会返回一个 ModelAndView。

4.DispatcherServlet 根据得到的 ModelAndView 中的视图对象，找到一个合适的 ViewResolver（视图解析器），根据视图解析器的配置，DispatcherServlet 将要显示的数据传给对应的视图，最后显示给用户。

五、Struts1.x 与 Struts2.x 的区别

Struts 2 以 WebWork 为核心，

采用拦截器的机制来处理用户的请求，struts1 严重依赖于 servletAPI,
属于侵入性框架,struts2 不严重依赖于 servletAPI,属于非侵入型框架。
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（每次请求时都新 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 类来封装请求的参数，或者直接使用 Action 的属性。
struts1 的前端总控制器(核心总控制器)为 ActionServlet,
struts2 的前端总控制器(核心总控制器)为 FilterDispather

六、Spring MVC、struts1 和 struts2 区别

1.spring mvc 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.spring mvc 的入口是 servlet，而 struts2 是 filter

spring 的前端总控制器为 DispatcherServlet

struts2 的前端总控制器为 FilterDispatcher

struts1 的前端总控制器为 actionServlet

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.spring mvc 是基于方法的设计,而 sturts 是基于类

七、Struts2 中 result 中的 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

八、Struts2 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>

九、SSI 整合

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 SqlMapClientDaoSupport
5、在 dao 层的配置文件中注入 sqlMapClient

十、SSH 整合

1.首先在 web.xml 中通过 ContextLoaderListener 来融入 spring，并加载 spring 的相关配置文件

2.同样配置 sturts2 的前端总控制器 filterDispatcher 来过滤相关的请求并且加载 struts.xml

3.action 继承 ActionSupport，然后通过引入 struts-spring-plugin.jar
包并且根据配置文件中 service 的 id 生成 get,set 方法来注入 service 层。

4.dao 层继承于 HibernateDaoSupport，并且在 dao 的配置文件中注入 sessionFactory.

5.通过 spring 中的配置文件加载 hibernate.cfg.xml 文件从而融入 hibernate.

在 ssh 框架中是怎么整合 spring?
首先在 web.xml 中通过 ContextLoaderListener 来融入 spring，
并加载 spring 的相关配置文件

在 ssh 框架中是怎么整合 hibernate?
通过 spring 中的配置文件加载 hibernate.cfg.xml 文件从而融入 hibernate
dao 层继承于 HibernateDaoSupport，并且在 dao 的配置文件中注入 sessionFactory

在 ssh 框架中是怎么整合 struts2?
配置 sturts2 的前端总控制器 filterDispatcher 来过滤相关的
请求并且加载 struts.xml

十、Spring MVC 整合

1.首先,要在 web.xml 里面配置 SpringMVC 的核心控制器,DispatcherServlet,对指定的后缀请求进行拦截。
2.Controller 层要加 @Controller 注解,表明该类是 MVC 的控制层。
3.创建 Service 接口,给接口加上注解 @Component 或者 @Service 表明这是 Service 业务处理层
4.在 Controller 层声明 Service 变量(属性),给变量（属性） 加上 @Autowired 注解,通过自动绑定机制将 Service 注入到 Controller。 (注：@Autowired 默认是 ByType,如果想根据属性名注入,那么就再加上注解 @Resource(name="属性名"))
5.在 Controller 层的方法上加上注解 @RequestMapping("requestAddress") 表明该方法的请求地址
6.Dao 层要加上注解 @Repository 表明这是数据库持久层
7.同样将 dao 实例注入到 service 层中。
8.配置视图解析器 "InternalResourceViewResolver",对处理后的跳转进行统一配置。

十一、Hibernate 中 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

十二、Hibernate、Ibatis、Jdbc 三者的区别

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

十三、Hibernate 的运行原理

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。

十四、Hibernate 五大核心（类/接口）简述

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

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 接口

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 与 Criteria 接口

总之 Query 和 Criteria 接口负责执行各种数据库查询。

5 .Transaction 接口

Transaction（事务）负责操作相关的事务。

十五、Hibernate 与 JDBC 的区别

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。

十六、Hibernate 中的两大配置文件

*.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。

十七、Hibernate 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();

十八、Hibernate 的三种状态以及状态的转换

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。

十九、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。

二十、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

二十一、Ssh 的概述：

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
引入 struts-spring-plugin.jar 包，从而完成 struts 和 spring 的整合
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

Dao 层继承于 hibernateDaoSupport
在 dao 层的配置文件中注入 sessionFactory

二十二、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
5.解决 struts2 重复提交
可以结合 s:token 标签来解决重复提交问题

利用 token 的原理：
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。

二十三、JSP 标签：

1.JSP include 动作
jsp:include 动作
以“<jsp: 动作名 ” 开始，以“</jsp:动作名> ” 结束
比如：<jsp:include page=" Filename" />

2.JSP 指令：<%@ include%><%@ %>
以“<%@ ” 开始，以“%> ” 结束。比如：
<%@ include file = " Filename" %>
3.JSP 输出表达式：<%= %><%=Java 表达式 %>
输出变量的值，后边不能加<%= ; %>
4.JSP Scriptlet【脚本】:<% ;%> <% Java 代码 %>
例子：
<% Calendar now = Calendar.getInstance(); %>
5.JSP 声明：<%! %> <%! 函数或者方法 %>
例子：

```
<%!
        String getHello(String name) {
          return "Hi," + name + "!";
        }
%>
```

6.迭代标签：<c:foreach>

Jstl 中的核心标签（core）

7.JSP 注释：

<!-- 这是注释,但客户端可以查看到 -->

<%-- 这也是注释,但客户端不能查看到 --%>

8.el 表达式：${}

9.jsp:include 动作是在运行时动态包含。

@include 指令是在编译时包含。

它们两个都只能包含本项目的相关文件，不能包含其他项目的。

如果要包含其他项目的文件可以使用 c:import

二十四、过滤器

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 AOP

servlet+jsp+javabean+jdbc+filter

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>org.leopard.filter.EncodingFilter</filter-class>
    <init-param>
      <param-name>encode</param-name>
      <param-value>utf-8</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>*</url-pattern>
</filter-mapping>
```

二十五、拦截器的理解

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。

二十六、Spring 融入框架

我们通过在 web.xml 中配置 ContextLoaderListener 这个监听器也加载 spring 的配置文件，从而融入到项目框架中。

二十七、项目的部署方式

1、如果项目单独部署到 tomcat 中的时候，应该看 tomcat 中的 server.xml;

2、如果和 eclipse 结合使用进行项目部署的时候，应该看 eclipse 里面的 server.xml.

数据库篇

一、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。

二、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 tomcat 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。

DBCP：比较稳定。

C3P0: 性能比较高。

三、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)

四、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。

五、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();
    }
```

六、Oracle 分页

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

七、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.

八、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

九、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。

十、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。

十一、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

十二、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

十三、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。

十四、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.

2、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）

3、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

十五、必背的 sql 语句

1：oracle 分页

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

2: mysql 分页

select * from music where id limit 5,5

3：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

4：音乐专辑

查询出 special <app:ds:special>表中的 id 专辑名 并下面有多少首歌曲

Select s.id , min(s.sname),count(m.mid) from special s inner join ms m on s.id=m.id group by s.id

5：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

6：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

7：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

8：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

9：oracle 中查询遍历树形结构（start　with）

```
select * from extmenu
start with pid=1
connect by prior id = pid
```

快速删除父节点以及父节点下的所有节点：

```
Delete from extmenu where id in (
elect * from extmenu
start with pid=1
connect by prior id = pid
)
```

10：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

11：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)

12：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
    );
```

13：row_number()，rank() over ，dense_rank() over 按工资排序

```
        select sal,
                    row_number() over(order by sal desc) rank1,
                    rank() over(order by sal desc) rank,
                    dense_rank() over(order by sal desc) drank
            from emp
```

14：

```
select * from (select emp.* from(
dense_rank() over(partition by departNo order by sal desc)
rk from emp )
Where rk=4
```

十六、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

一、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。

二、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、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。

四、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。

五、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。

六、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。

七：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。

八、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。

九、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。

十、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。

十一、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。

十二、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。

十三、(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

十四、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。

十五、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。

十六、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。

十七、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。

十八、oracle 索引概述

索引呢 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。

十九、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.

二十、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。

二十一、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。

二十二、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。

二十三、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)

二十四、Servlet 的概述：

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 方法
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

二十五、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）

二十六、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

1.通过 wsdl2java 根据 webservice 服务端的 url 生成客户端代码

2.将代码引入项目的文件夹中进行正常访问

二十六、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

二十七、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。

二十八、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。

二十九、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。

2.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。

三十、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

三十一、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" />
```

三十二、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

一、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

二、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以及复选按钮

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

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

三、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。

四、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。

五、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)=1024KB 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)=1024TB 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。