去哪儿日常学习

1. springMvc框架的路由  
    controller开始之前出现注解@RequestMapping("/XXX"),例如 @RequestMapping("/agent");  
      action开始之前出现注解@RequestMapping(value = "/XXX", method = RequestMethod.XXX);  
             例如 @RequestMapping(value = "/query/list", method = RequestMethod.POST)，这就代表是用post方式请求  
      那么以上的网址后面的路由是  .com/agent/query/list  
    action开始之前出现注解@RequestMapping(value = "/XXX/{id}", method = RequestMethod.XXX);  
             例如 @RequestMapping(value = "/query/list/{id}", method = RequestMethod.POST)，这就代表是用post方式请求  
      那么以上的网址后面的路由是  .com/agent/query/list/123 就可以在方法中  
    public ModelAndView getDetail(@PathVariable(value="id") Integer id){ } 就可以在方法中使用id这个
2. Qconfig配置中心  
    配置与代码分离，集中管理配置，结合不同的环境下发配不同的配置  
    监控配置文件的发布流程  
    原理：qconfig通过调用ScheduledThreadPoolExecutor的scheduleWithFixedDelay方法定时拉去配置文件内容，并比较配置文件版本，如需更新本地配置文件，则执行拉取配置文件操作，延迟启动periodInSecond秒，上一个任务执行完成后，延迟periodInSeconds秒启动下一个任务。  
    如果在qconfig server端修改了配置后，由服务端发出通知，客户端订阅通知，进行配置拉取，这样客户端就不需要频繁的调用远程server了。（有更新，服务器端远程自动提醒，客户端自动加载新的配置文件）  
    <http://qconfig.corp.qunar.com/home.do> TCDEV 配置中心 后台配置管理
3. 前端文件调试用 alert(某某变量) console.log('data',data);  
    后端文件调试用 断点调试  
    jsp 里面的调试 断点调试，out.print(“ ”) out.println(“ ”), 最好不要用system.out.print之类的，

Jsp里面调试可以用 ${username} EL表达式，username就是变量  
因为这个是输出到控制台上，而jsp文件是要解析成页面的

1. ajax的格式

ajax格式  
 **function** renderUpdateAgentMsg() {  
 **var** BUS\_DIS\_URL = '/config/queryAllFieldFromConfig'; //需要传入url  
 $.ajax({  
 url: BUS\_DIS\_URL,  
 type: "GET", //传送的格式是get还是post还是request  
 success: **function**(data) {  
 **var** type = data.ret ? data.data : **null**; //这是获得的数据  
 render(type);  
 }  
 });  
}

1. git返回以前的版本   
    git reset –hard commit\_id  
    git revert commit\_id
2. 注意每个项目，前端，后端，api项目的分支尽量都要一致，还有远程分支尽量只留一个

删除远程分支

Git branch –a 查看远程分支

git push origin --delete <branchName> 删除远程分支

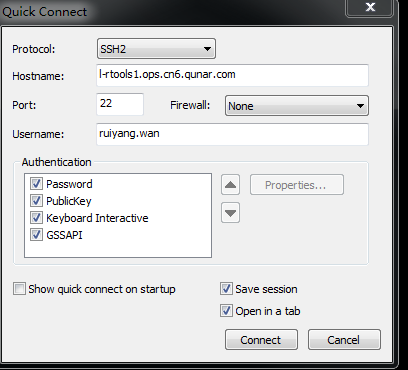
Git branch –D branch\_name 删除本地分支

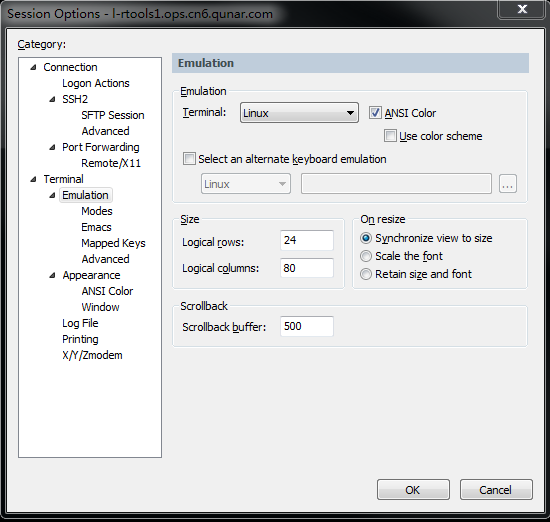
1. Git branch –vv 查看本地分支与远程分支的关联
2. beta 的端口是8080 ，线上端口是 9000  
    <http://l-hagent1.h.beta.cn6.qunar.com:8080/agent/save> 例如这个url
3. 后台beta版服务器的权限申请，<http://ops.corp.qunar.com/ops/account/> 注意这里有一个passphrase  
    在secureCRT中登录到这个服务器终端，可以查看日志  
    绿色是beta版，红色是线上服务器  
    进去之后，会出现 linux中命令行界面  
    ssh 服务器地址 是进入到这个服务器当中  
    cd /home/q/www 进入到想要的beta版的发布文件夹里面  
    查看日志 是在log文件夹下面 tail –f catalina.out 即可进入打印日志模式

tail -f catalina.out|grep 'Exception'

cat catalina.out|grep 'Exception'

线上查看抛出异常的日志

secureCRT 连接跳板机的作用就是做了中间的安全机制  
  **secureCRT的连接,注意着下面得端口是22,hostname是l-rtools1.ops.cn6.qunar.com**  
   
 注意这里每个服务器的hostname最后都要加上 qunar.com  
 linux风格设置： Option->Session Options->Terminal



快速登录跳板机 <http://wiki.corp.qunar.com/pages/viewpage.action?pageId=128726623>

1. IDEA快捷键  
    ctrl+shift+n 寻找文件名  
    ctrl+n 寻找类名  
   　　 ctrl+shift+u 大小写之间的转化
2. IDEA调试  
    <http://blog.csdn.net/idiot_xue/article/details/51054439>   
    本地断点调试
3. linux yum文件
4. 数组的声明与定义  
    数据类型[][] 数组名 = new 数据类型[行的个数][列的个数] 这恶鬼声明一个规则的二维数组，并且分配了空间,数组元素赋值的时候一个元素一个元素的赋值  
    int[][] num = new int[3][]; num[0] = new int[2]; 如果是这种方式在定义的时候只指定行的个数，可以创建不规则的二维数组

Array.sort(arr[]) 对数组升序排序  
 Array.toString(arr[]) 将数组转化成字符串，元素之间用 ，隔开

方法调用时，不能传进去一个在参数里面定义的 hello.getNameById(int a = 12); 这是不对的

注意使用Array.toString(arr) 这种方法时候，要导入 java.util.Arrays

变量成员修饰符Public与static之间的区别：

Public是修饰权限的，表示共有的  
static是修饰状态的，表示静态的

静态初始化块，要在前面加上 static: {body}

成员内部类：

定义了成员内部类之后，必须使用外部类对象来创造内部类对象，不能直接new 一个内部类的对象， 内部类 对象名 = 外部类对象.new 内部类()

Outer outer\_name = new outer(); outer inner\_name = outer\_name.new inner();

编译含有内部类的外部类，会产生两个class文件

一个是 Outer$Inner.class 一个是 Outer.class

内部类想访问被重载的外部类方法，可以用Outer.this.function\_name();即可  
   
 静态内部类 ：

静态内部类不能直接访问外部类的非静态属性，但是可以通过 new 外部类().成员 去访 即 new Outer().xxx

如果外部类的静态成员与内部类的成员名称相同，可以通过 class\_name.静态成员

如果外部类的静态成员与内部类的成员名称不相同，则直接 使用 静态成员

创建静态内部类的对象时，可以直接创建 Inner Inner\_name = new Inner()

方法内部类：

方法内部类是定义在外部类的方法当中，只在方法中可见，只在方法中使用

方法内部类由于不能在方法以外的地方使用，因此不能使用访问控制符和static修饰符

如果子类显示的调用构造方法，必须只在子类的构造方法的第一行，如果自雷没有显示的调用构造方法，则默认调用父类无参的构造方法，默认通过super();来调用，这个super被隐藏了。

如果类中定义了有参的构造方法，系统就不会自动生成无参的构造方法

Object的 toString()方法 equals()方法 两个对象的引用是否指向同一块地址

多态：

引用多态：父类引用指向子类对象 Animal a = new Dog() 然后这里animal是接口，更简单

引用类型转换：

向上类型转换（隐式/自动类型转换）,是小类型向大类型的转换  
 向下类型转换（强制类型转换），是大类型向小类型的转换  
 instanceof运算符，来解决引用对象的类型，避免类型转换的安全性问题

1. 二进制基础  
    & 与运算 0&0=0 0&1=0 1&0=0 1&1=1  
    清零：想将一个单元清零，全部二进制位为0 与一个二进制位都为0的数值相与  
    取一个数中指定位：找一个数，对应X要取的位，该数的对应位为1，其余位为0进行   
    与运算  
    | 或运算 0|0=1 0|1=1 1|0=1 1|1=1 只要有一个为1，结果就为1  
    常用来对一个数据的某些位 置为1：找到一个数，对应X要置为1的位，则与或运算的  
    数对应的位 置去，其余置为0即可  
    比如讲x=1010 0000的低四位 置1 x|0000 1111即可得到 1010 1111  
    ^ 异或运算 0^0=0 0^1=1 1^0=1 1^1=1 有1就为1，无1则为0

<< 左移运算  
 >> 右移运算

1. 线上上线某项目，一台服务器一台服务器的上线，在上线的时候，线上访问的请求都不会打到正在发布的服务器上面，只有拥有这台服务器的权限，才可以访问正在发布的页面，所以在线上发布之前，要申请线上服务器的权限，这样就可以查看发布线上代码时候的日志了
2. 代理商和wrapper id是一一对应的

**Java Spring框架**

1. 目录结构  
   .idea   
    artifacts copyright版权 libraries存放web项目需要的额外的库文件，jar包  
   .settings IDEA的一些配置  
   catalina.base\_IS\_UNDEFINED   
    logs  
   src 存放源码和配置文件

main test测试用得到自己写的文件，测试文件

service 对服务的包装，bean是model层对数据库的操作

dao 数据访问对象，持久层，主要与数据库打交道，dao中不应该有业务逻辑

service 服务层，防止业务逻辑，涉及到的数据库操作就调用dao

webapp必须有一个命名为 WEB-IBF的，该目录之外的资源，用户可以通过浏览器直接访问，位于该目录下的所有资源，用户通过浏览器无法直接访问，但是内部servlet是可以进行条用的。很多mvc框架，将jsp文件放在WEB-INF目录下进行保护，所有的jsp文件访问通过servlet进行跳转

temp 通常都是临时产生的文件，比如一些缓存

lib 库文件  
 <https://my.oschina.net/zh119893/blog/192187>

<https://my.oschina.net/kzhou/blog/108971>

1. Xml

Xmlns 是xml namespace的缩写，是xml命名空间，类似java文件里面命名机制

<?xml version="1.0" encoding="UTF-8"?>

<!--xml版本信息和编码方式-->

<!—根节点为beans-->  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://code.alibabatech.com/schema/dubbo  
 http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

<!--xmlns xml xml命名空间,默认的命名空间，如果默认的没有加，则用的是这个路径下的-->  
 <!--xmls:xsi xml schema instance web xml遵守的xml规范，namespaces 在beans中定义xsi的命名空间，这个命名空间里面的元素或者属性就必须要以 xsi: 这种方式来写-->  
 <!--xmlns:dubbo 定义名为dubbo的命名空间，比如下面用到的这个命名空间的节点和属性，前面就要加上 dubbo: -->  
 <!--xml:schemaLocation 具体用到的xml资源-->  
 <!-- 提供方应用信息，用于计算依赖关系 -->   
 <dubbo:application name="q\_hotel\_agent" owner="xunxin.wan" organization="q\_hotel\_agent"/>  
  
 <!-- 使用multicast广播注册中心暴露服务地址 -->  
 <dubbo:registry protocol="zookeeper" address="l-zk1.beta.cn6.qunar.com:2181" id="qHotelAgent" check="false" group="beta"/>  
 <!-- 用dubbo协议在20990端口暴露服务 -->  
 <dubbo:protocol name="dubbo" port="20990"/>  
 <dubbo:provider filter="qmonitor" retries="1" executes="400"/>  
 <import resource="service-pub.xml"/>  
</beans>

public static String hotelVipToSql\_insert(String fileName) throws IOException, ParserConfigurationException, SAXException {  
 File f = new File(fileName);  
 FileInputStream fis = new FileInputStream(f);  
 //使用DOM方式解析xml  
 //加载xml文件，加载文件的一系列方法  
 DocumentBuilderFactory domfac = DocumentBuilderFactory.*newInstance*();

//创建一个DocumentBuilderFactory的对象  
 DocumentBuilder dombuilder = domfac.newDocumentBuilder();   
 //创建一个DocumentBuilder的对象  
 Document doc = dombuilder.parse(fis);

//调用parse() 方法加载xml文件到当前项目  
  
 //解析xml文件的属性和属性值  
 NodeList hotels = doc.getElementsByTagName("hotels");

//得到所有标签名为hotels的节点，得到一个hotels的集合  
 NodeList adconf = ((Element) (hotels.item(0))).getElementsByTagName("adconf");  
 NodeList adinfo = ((Element) (adconf.item(0))).getElementsByTagName("adinfo");  
 StringBuilder s = new StringBuilder();

for (int i = 0; i < adinfo.getLength(); i++) {

//获取集合长度，遍历集合hotels  
 Element e = (Element) (adinfo.item(i));

//集合hotels调用item()方法 获取集合hotels里面的节点e  
 String wrapperId = e.getAttribute("wrapper");

//节点e调用attribute()方法 获取所有的属性集合  
 String wrapperName = *getTextContent*(e, "wrapperName");

//节点e直接调用getTextContent()方法直接获取节点值

DOM使用步骤：   
准备工作   
1、创建一个DocumentBuilderFactory的对象   
DocumentBuilderFactory dbf = DocumnetBuilderFactory.instance();   
2、创建一个DocumentBuilder的对象   
DocumentBuilder db = dbf.newDocumentBuilder();   
3、通过DocumentBuilder对象的parse方法加载xml文件到当前项目   
Document document = db.parse(“\*.xml”);   
2-2 使用 DOM 解析 XML 文件的属性名和属性值   
使用Dom解析xml文件的属性节点。   
在不知道节点属性的个数和属性名时：   
1、通过document.getElementByTagName(“标签名”)获得所有标签名的节点，得到一个NodeList集合   
2、通过NodeList.getLength()获得集合长度，遍历集合   
3、Node node = NodeList.item（index）获得里面的节点   
4、通过NamedNodeMap attrs = node.getAttributes()获取所有属性集合   
5、通过attrs.getLength()遍历集合,Node attr = atrrs.item(index)   
6、attr.getNodeName()获得属性名，attr.getNodeValue()获取属性值   
前提已经知道book节点有且只有1个id属性，将book节点进行强制类型转换，转换成element类型。   
1、通过document.getElementByTagName(“标签名”)获得所有标签名的节点，得到一个NodeList集合   
2、通过NodeList.getLength()获得集合长度，遍历集合   
3、element book=（element）bookList.item(i);//强制转换为element类型   
4、string attrValue = book.getAttribute(“id”);

Xml文件的四种读取<http://blog.csdn.net/sinat_34443859/article/details/51692842>

Xml文件的四种写入<http://blog.csdn.net/sinat_34443859/article/details/51693936>

1. Java socket通信

端口号：

用于区分不同的应用程序

端口号范围为0-65535，其中0-1023为系统保留

Ip地址和端口号组成了所谓的socket，socket是网络上运行的程序之间双向通信链路的终结点，是TCP和UDP的基础

常用的端口号：http：80， ftp：21， telnet：23

针对网路通信的不同层次，java提供的网络功能的四大类

1、InetAddress：用于标识网络上的硬件资源（IP地址）

2、URL:统一资源定位符–通过URL可以直接读取或写入网络上的数据

3、Sockets：使用TCP协议实现网络通信的Socket相关的类

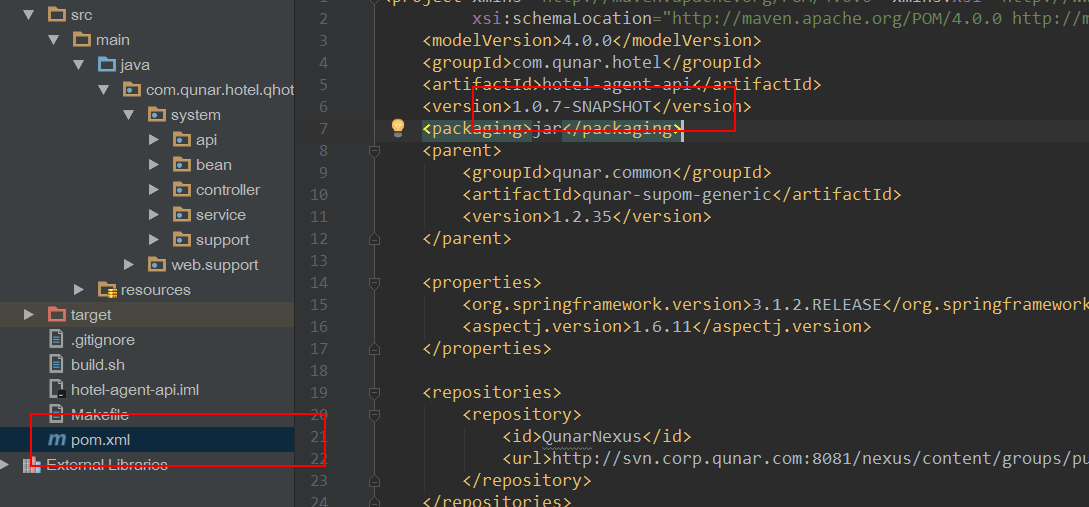
4、Datagram：使用UDP协议。将数据保存在数据报中，通过网络进行通信

<http://blog.csdn.net/sinat_34443859/article/details/51775831>

1. 每一次线上发布之前，都要申请其beta测试的服务器权限，还有线上机子的权限，
2. 每一次提交代码的时候，都要merge一下master分支的代码
3. 每一次线上发布，改完代码前，都需要将代码CodeReview，这个就在pmo信息里面



注意一个需求有多个CR，都要CodeReview

1. 每一次beta版本发布，Pom.xml文件里面引用的version是有SANPSHOT（快照 ），beta版本引用的是快照版本
2. 
3. 但是线上发布的时候，是生成没有SANPSHOT的maven版本，因此线上发布的时候必须要将SANPSHOT去掉
4. Java注解 annotation @AspectJ注解技术

注解原理：

自定义的一些标签，并通过java语言的反射机制读取注解的信息  
注解不能直接干扰程序代码的进行，无论增加或者删除注解，代码都能够正常进行，java语言解释器会忽略这些注解，由第三方工具负责对注解进行处理。  
第三方工具通过java反射机制读取注解的信息，并根据这些信息更改程序的逻辑，通过注解渐渐控制代码的运行  
@override 只能用在方法上，这个方法是改写父类的  
@Deprecated 建议不要使用旧的API，

@SuppressWarning 关闭一些不当的编译器警告信息

定义注解：

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Inherited //元注解

public @interface Description {

String value(); //注解元素

}

注解定义看起来很像接口的定义，注解会被编译成class文件  
设计注解的时候必须把一个类型定义为@interface，而不能用class或interface关键字，所有的注解类都隐式继承java.lang.annotation.Annotation,注解不允许显示继承于其他的接口

元注解 meta-annotation，自定义注解会有四个元注解：

@Target：用来定义注解将用于什么地方，可能的ElementType参数包括  
 CONSTRUCTOR：构造器的声明

FIELD：域声明（包括enum实例）

LOCAL\_VARIABLE：局部变量声明

METHOD：方法声明

PACKAGE：包声明

PARAMETER：参数声明

TYPE：类、接口（包括注解类型）或enum声明

ANNOTATION\_TYPE：注解声明（应用于另一个注解上）

TYPE\_PARAMETER：类型参数声明（1.8新加入）

TYPE\_USE：类型使用声明（1.8新加入）

PS：当注解未指定Target值时，此注解可以使用任何元素之上，就是上面的类型

@Retention 定义注解在哪个级别可用，在什么级别保存该注解信息，由RetentionPolicy枚举定义

SOURCE：注解将被编译器丢弃（该类型的注解信息只会保留在源码里，源码经过编译后，注解信息会被丢弃，不会保留在编译好的class文件里）  
CLASS：注解在class文件中可用，但会被VM丢弃（该类型的注解信息会保留在源码里和class文件里，在执行的时候，不会加载到虚拟机（JVM）中）  
RUNTIME：VM将在运行期也保留注解信息，因此可以通过反射机制读取注解的信息（源码、class文件和执行的时候都有注解的信息）

当注解未定义Retention值时，默认值是CLASS

举一个例子，如@Override里面的Retention设为SOURCE，编译成功了就不要这一些检查的信息，相反，@Deprecated里面的Retention设为RUNTIME，表示除了在编译时会警告我们使用了哪个被Deprecated的方法，在执行的时候也可以查出该方法是否被Deprecated。

@Documented 表示注解会被包含在java api（javadoc ）文档中

@Inherited 允许子类继承父类的注解

注解元素：

在注解中，一般都会包含一些元素以表示某些值，当分析处理注解时，程序可以利用这些值

没有元素的注解称为标记注解 marker annotation

注解元素的类型：  
所有基本类型 ：int,float,boolean,byte,double,char,long,short

String Class enum Annotation 注解 以及以上类型的数组，注解也可以作为元素

如果使用了其他类型，编译器就会报错，也不允许使用任何包装类型

成员以无入参无抛出异常的方式声明，如boolean value(String str)、boolean value() throws Exception等方式是非法的；

可以通过default为成员指定一个默认值，如String level() default "LOW\_LEVEL"、int high() default 2是合法的，当然也可以不指定默认值；

如果注解只有一个成员，则成员名必须取名为value()，在使用时可以忽略成员名和赋值号（=），如@Description("使用注解的实例")。注解类拥有多个成员时，如果仅对value成员进行赋值则也可不使用赋值号，如果同时对多个成员进行赋值，则必须使用赋值号，如@DeclareParents (value = "NaiveWaiter", defaultImpl = SmartSeller.class)。

注解类可以没有成员，没有成员的注解称为标识注解，解释程序以标识注解存在与否进行相应的处理；

对于非基本类型的元素，无论是在源代码中声明，还是在注解接口中定义默认值，都不能以null作为值。这就是限制，这就造成处理器很难表现一个元素的存在或缺失状态，因为每个注解的声明中，所有的元素都存在，并且都具有相应的值。为了绕开这个限制，只能定义一些特殊的值，例如空字符串或负数，表示某个元素不存在。

使用注解

@Description(value="使用注解的实例")

public class TestAnnotation { }

使用注解的语法：使用注解的语法：@<注解名>(<成员名1>=<成员值1>,<成员名1>=<成员值1>,...)

如果成员是数组类型，可以通过**{}**进行赋值，如boolean数组的成员可以设置为{true,false,true}。

1)   多成员的注解

@AnnoExample(id= 2868724, synopsis = "Enable time-travel", engineer = "Mr. Peabody")

2)   一个成员的注解，成员名为value

可以省略成员名和赋值符号：@Copyright("Right Reserved")

@Copyright("2011 bookegou.com All Right Reserved")

3)   无成员的注解

@Override

@Override

4)   成员为字符串数组的注解

@SuppressWarnings(value={"unchecked","fallthrough"})

@SuppressWarnings(value={"unchecked","fallthrough"})

5)   成员为注解数组类型的注解

@Reviews({@Review(grade=Review.Grade.EXCELLENT,reviewer="df"),

@Review(grade=Review.Grade.UNSATISFACTORY,reviewer="eg",

comment="This method needs an @Override annotation")})

@Reviews({@Review(grade=Review.Grade.EXCELLENT,reviewer="df"),

           @Review(grade=Review.Grade.UNSATISFACTORY,reviewer="eg",

                    comment="This method needs an @Override annotation")})

@Reviews注解拥有一个@Review注解数组类型的成员，@Review注解类型有三个成员，其中reviewer、comment都是String类型，但comment有默认值，grade是枚举类型的成员。

<http://www.cnblogs.com/longshiyVip/p/5189525.html>

<http://blog.csdn.net/chenxiang0207/article/details/8193980>

1. **JSP Java Server Pages是一种动态网页开发技术**

Jsp技术是以java语言作为脚本语言，

jsp文件后缀名为 \*.jsp

JSP开发的web应用可以跨平台使用，linux与windows都可以运行

使用jsp标签在html网页中插入java代码，标签通常是 <% demo %>

<%@ **page** import="com.qunar.hotel.qhotelagent.system.bean.Account" %>  
<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>

<%@**include** file="/WEB-INF/views/common/header.jsp"%>

Jsp是一种 java servlet，用来实现java web程序的用户界面部分

<http://www.runoob.com/jsp/eclipse-jsp.html>

1. **JSP页面处理流程**

就像其他普通的网页一样，您的浏览器发送一个HTTP请求给服务器。

Web服务器识别出这是一个对JSP网页的请求，并且将该请求传递给JSP引擎。通过使用URL或者.jsp文件来完成。

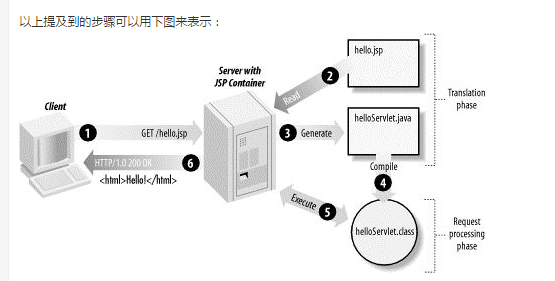
JSP引擎从磁盘中载入JSP文件，然后将它们转化为servlet。这种转化只是简单地将所有模板文本改用println()语句，并且将所有的JSP元素转化成Java代码。

JSP引擎将servlet编译成可执行类，并且将原始请求传递给servlet引擎。

Web服务器的某组件将会调用servlet引擎，然后载入并执行servlet类。在执行过程中，servlet产生HTML格式的输出并将其内嵌于HTTP response中上交给Web服务器。

Web服务器以静态HTML网页的形式将HTTP response返回到您的浏览器中。

最终，Web浏览器处理HTTP response中动态产生的HTML网页，就好像在处理静态网页一样。



JSP引擎会检查JSP文件 对应的servlet是否已经存在，并且检查JSP文件的修改日期是否早于servlet。若早于，则说明servlet有效，则就直接可以使用该servlet，这就很快捷有效

JSP网页几乎可以被当成一个普通的servlet来对待。

**JSP生命周期：**

JSP生命周期就是从创建到销毁的整个过程，类似于servlet生命周期，区别在于JSP生命周期还包括将JSP文件编译成servlet。

* **编译阶段：**

servlet容器编译servlet源文件，生成servlet类

JSP引擎首先去 检查这个文件是否被编译过，没有被编译则编译，编译过就不用编译了

编译过程： 解析JSP文件，将JSP文件转化为servlet，编译成servlet

* **初始化阶段：**

加载与JSP对应的servlet类，创建其实例，并调用它的初始化方法

程序只初始化一次，在为请求提供任何服务之前调用JSPInit()方法，也可以复写jspInit()方法

通常情况下您可以在jspInit()方法中初始化数据库连接、打开文件和创建查询表

* **执行阶段：**

调用与JSP对应的servlet实例的服务方法

描述一切与请求相关的交互行为，知道被销毁

初始化完成后，JSP引擎调用\_jspService()方法，

\_jspService()方法需要一个HttpServletRequest对象和一个HttpServletResponse对象作为它的参数

void \_jspService(HttpServletRequest request,

HttpServletResponse response)

{

// 服务端处理代码

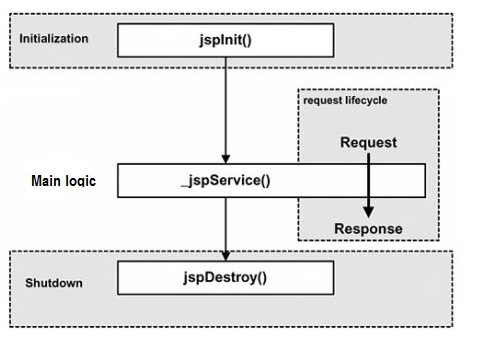
}

\_jspService()方法在每个request中被调用一次并且负责产生与之相对应的response，并且它还负责产生所有7个HTTP方法的回应，比如GET、POST、DELETE等等。

* **销毁阶段：**

调用与JSP对应的servlet实例的销毁方法，然后销毁servlet实例，也就是一个JSP网页从容器中被移除时发生的一切

销毁方法 jspDestroy(),也可以复写



**JSP与Servlet之间的关系与差异**

Servlet是一种运行在Tomcat服务器中，可以自动产生HTML网页的java技术。Servlet是对支持Java的服务器的一般补充，常用语扩展Web服务器，是一种动态加载的模块，为来自Web服务器提供服务

Servlet完全运行在Java虚拟机上，由于在服务器端运行，因此不依赖于浏览器的兼容性

Servlet 是平台独 立的 Java 类，编写一个 Servlet，实际上就是按照 Servlet 规范编写一个 Java 类。

Servlet 被编译为平台中立的字节码，可以被动态的加载到 Java 技术的 Web 服务器中运行。

Servlet 可以被看作一个基于 Java 技术的 Web 组件，该组件由 Servlet 容器 （Tomcat、Resin 等服务器）管理，用于生成动态的 Web 信息

JSP 提供了一套简单的标签，和 HTML 融合得比较好，JSP 页面修改后可以立即看到效果，不需要手工编译，JSP 引擎会来做这些工作；而 Servlet需要编译，重新启动 Servlet 引擎等一系列动作。

。JSP 编写简单，可以方便地嵌入

HTML 中，很容易加入动态内容，方便地输出 HTML。在 Servlet 中输出 HTML则需要调用特定的方法，相对于 JSP 比较复杂。

**JSP页面元素**

**静态部分**（如HTML，CSS标记等，用来完成数据显示和样式）

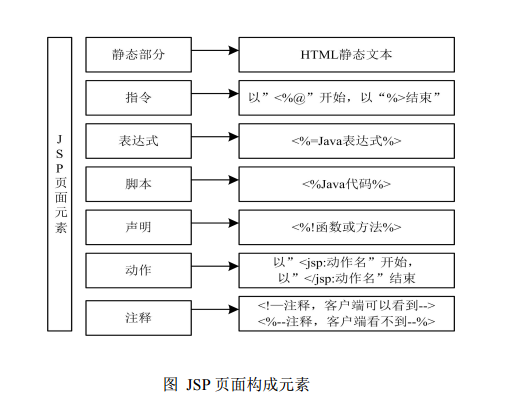
**动态部分**（如脚本程序，JSP标签等，用来完成数据处理）

动态部分包含四种：脚本元素+指令+动作+注释

**脚本元素**用来嵌入 Java 代码，这些 Java 代码将成为转换得到的 Servlet 的一部分，脚本元素又可以划分为**声明、表达式** 和**脚本执行程序**；

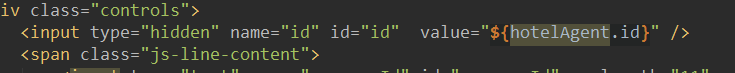
**JSP 指令**用来从整体上控制 Servlet 的结构；

**动作**用来引入现 有的组件或者控制 JSP 引擎的行为。



<http://wiki.corp.qunar.com/download/attachments/88015736/JSP%E9%A1%B5%E9%9D%A2%E5%85%83%E7%B4%A0.pdf?version=1&modificationDate=1438164894000>

1. **EL表达式**：Expression Language 目的是为了使JSP写起来更加简单

${expression} Expression:指定要输出的内容，可以是字符串，也可以是由EL运算符组成的表达式。

如以上，在jsp文件中使用 变量给一个标签的value赋值

<%=request.getParameter(“name”)%>,这样和html混编的方式也可以获取到值

${“expression”}显示的是 expression

${param.name} 与 ${param[“name”]} 是等价的 EL的两个运算符 [] 和 .

<http://www.cnblogs.com/lihuiyy/archive/2012/02/15/2352643.html> EL既可以获取当前页面信息，也可以获取session，cookie，header，以及后端代码中对象的数据

1. Spring框架<http://www.jikexueyuan.com/course/1405.html>
2. **Maven 软件项目管理工具，基于项目对象模型POM的概念**

Maven教程<https://yq.aliyun.com/articles/28591>

Maven安装，安装到的路径作为maven环境变量配置中的路径，要配置一个MAVEN\_HOME（maven根目录），和path里面的%MAVEN\_HOME%\bin;  
maven使用标准的目录结构和默认构建生命周期，能够自动下载项目依赖库

Maven项目的结构和内容在一个xml文件中声明， pom.xml 项目对象模型 POM，是整个maven系统的基本单元

<http://www.cnblogs.com/yakov/archive/2011/11/26/maven2_settings.html> setting.xml文件配置说明

<http://blog.csdn.net/u012152619/article/details/51485152> setting.xml文件配置说明

POM代表项目对象模型，是工作在maven的基本单位，pom包含的项目是用maven来构建的，包含各种配置信息，pom中也包含了目标和插件，在执行任务和目标时，maven会使用当前目录中的pom，读取pom，得到所需要的配置信息，然后执行目标

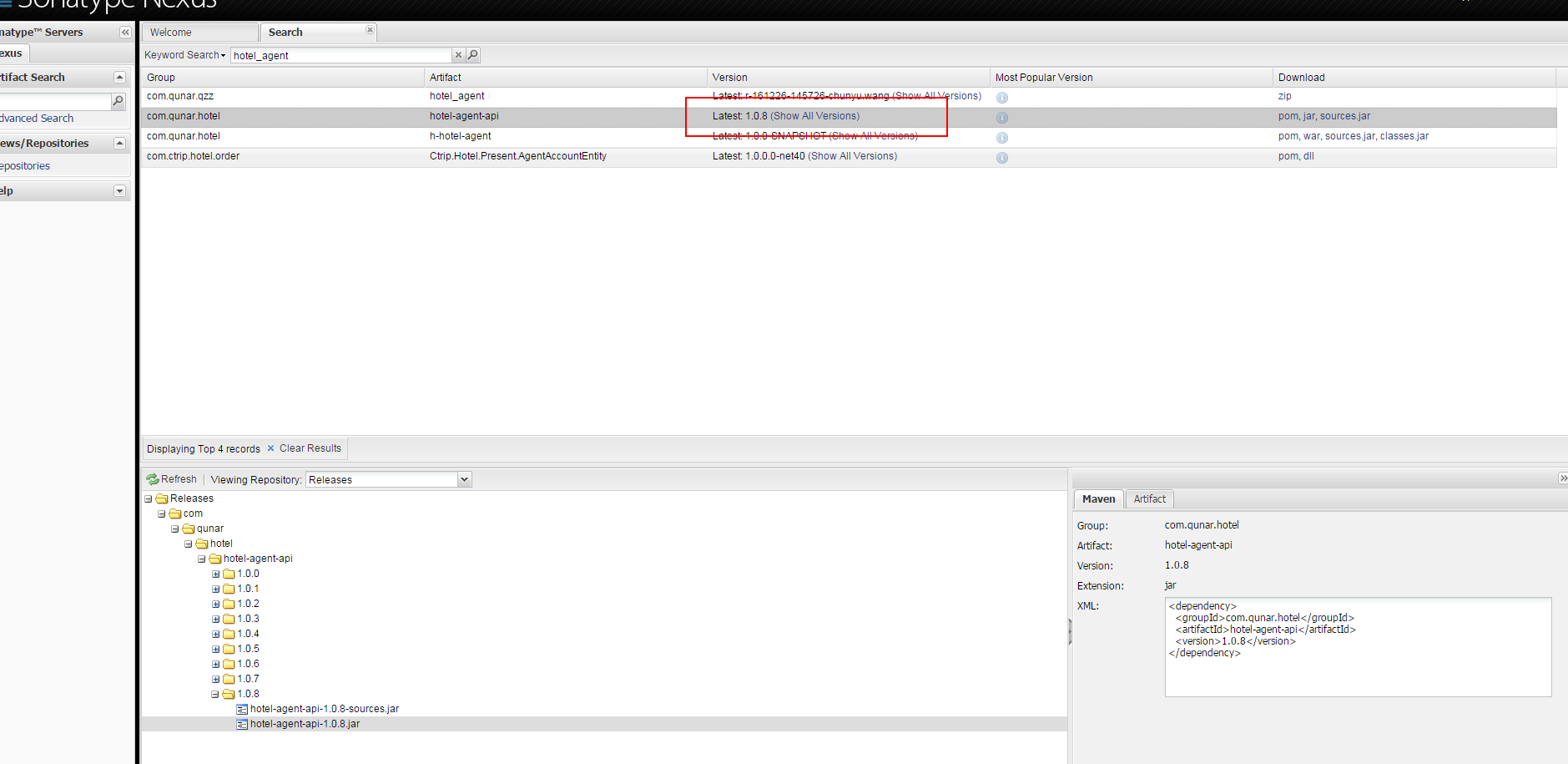
每个项目都有单一的

groupid 项目组织唯一的标识符，实际对应项目的名称，项目根目录的名称  
 一般groupID就是填写 com.qunar.test这种，

groupid一般分为很多段，第一段为域，第二段为公司名称

artifactId是工程名，也就是根文件夹名字，比如要输出helloworld，那就写 hello

举个apache公司的tomcat项目例子：这个项目的groupId是org.apache，它的域是org（因为tomcat是非营利项目），公司名称是apache，artigactId是tomcat。

1. 注意查看maven中jar包的版本的时候
2. 

这里有show all versions

在pom.xml文件中的 这是api jar包的版本 <hotel-agent-api.version>1.0.8-SNAPSHOT</hotel-agent-api.version>

这是 前端的版本 qzz.version <qzz.version>r-161220-154928-chunyu.wang</qzz.version>

1. pom.xml文件的详解

<http://niuzhenxin.iteye.com/blog/2042102>

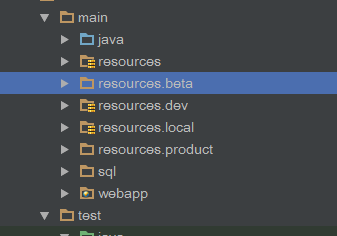
pom.xml的所有标签 <http://www.zuidaima.com/share/1781583829978112.htm>

**maven profile**提供了配置多种环境

<https://zhuanlan.zhihu.com/p/24074624>

<http://www.cnblogs.com/lzxianren/p/maven-profile.html>

hotel\_agent项目的目录



<profiles>  
 <profile>  
 <id>local</id> //唯一表示id  
 <activation>  
 <activeByDefault>true</activeByDefault> //local环境是默认的profile  
 </activation>  
 <properties>  
 <deploy.type>local</deploy.type> //这个表示是local环境  
 </properties>  
 </profile>  
 <profile>  
 <id>dev</id> //dev环境  
 <properties>  
 <deploy.type>dev</deploy.type>  
 </properties>  
 </profile>

</profiles>

<!--build 构建项目需要的信息 -->

<build>

<!-- finalName产生的构件的文件名，默认值是${artifactId}-${version}。 -->  
 <finalName>${artifactId}</finalName>  
 <resources>  
 <resource> <!-- directory描述存放资源的目录，该路径相对POM路径 -->  
 <directory>src/main/resources.${deploy.type}</directory>  
 </resource>  
 <resource>  
 <directory>src/main/resources</directory>  
 </resource>  
 </resources>  
 <plugins> <!-- plugins使用的插件列表-->  
  
 <!--QZZ related Begin-->  
 <plugin> <!-- plugin使用的插件所需要的信息-->  
 <groupId>com.github.goldin</groupId>  
 <artifactId>copy-maven-plugin</artifactId>  
 <version>0.2.5</version>  
 <executions> <!-- executions在构建生命周期中执行一组目标的配置。每个目标可能有不同的配置。-->  
 <execution> <!-- execution元素包含了插件执行需要的信息-->

<!--id 执行目标的标识符，用于标识构建过程中的目标，或者匹配继承过程中需要合并的执行目标 -->  
 <id>unpack-qzz</id>  
 <goals>

<!-- goal 配置的执行目标 -->  
 <goal>copy</goal>  
 </goals>

<!-- phase配置的执行目标 -->  
 <phase>initialize</phase>  
 <configuration>

<!-- configuration作为DOM对象的配置 -->  
 <resources>  
 <resource>

<!-- targetPath描述了资源的目标路径。该路径相对target/classes目录（例如${project.build.outputDirectory}）。举个例子，如果你想资源在特定的包里(org.apache.maven.messages)，你就必须该元素设置为org/apache/maven/messages。然而，如果你只是想把资源放到源码目录结构里，就不需要该配置。 -->  
 <targetPath>${qzz.app\_ver\_path}</targetPath>  
 <dependency>  
 <groupId>${qzz.groupId}</groupId>  
 <artifactId>${qzz.artifactId}</artifactId>  
 <version>${qzz.version}</version>  
 <type>zip</type>  
 </dependency>  
 <unpack>true</unpack>  
 </resource>  
 </resources>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
</build>

在界面最下方termianl终端，可以输入git，maven命令，操作

1. Mvn –v 查看maven是否安装成功， git –version查看git是否安装成功
2. Mvn archrtype:generate
3. Mvn compile 编译源代码（这个就是可以在api里面使用）
4. Mvn test-compile 编译测试代码
5. Mvn test 测试运行
6. Mvn install 在本地respository中安装jar
7. Mvn clean 清除产生的项目
8. Mvn help:effective-pom查看依赖的父pom
9. Mvn dependency:tree
10. Mvn clean package –Pdev –Dmaven.test.skip=true 清除之前生成的jar包，打包到dev环境，这里跳过了test文件夹，因为上传上去，不需要test代码
11. Mvn –U clean package –Pbetamagic –Dmaven.test.skip=true –DcheckDeployRealse\_skip=true –Denforcer.skip=false –DskipTests enforcer:enforcer dependency:tree –Dverbose 对版本进行了限制
12. Mvn有三个标准的**声明周期**，三套生命周期是相互独立的

Clean lifecycle

Pre-clean clean之前的准备工作 clean 移除上次打包生成的文件 Post-clean clean之后的工作

注意执行clean，是会自动先执行Pre-clean的，执行Post-clean，是先会执行Pre-clean和clean的

Default LifeCycle 有23个阶段

pre-resource compile test-compile（只编译test目录下） test package install deploy 等

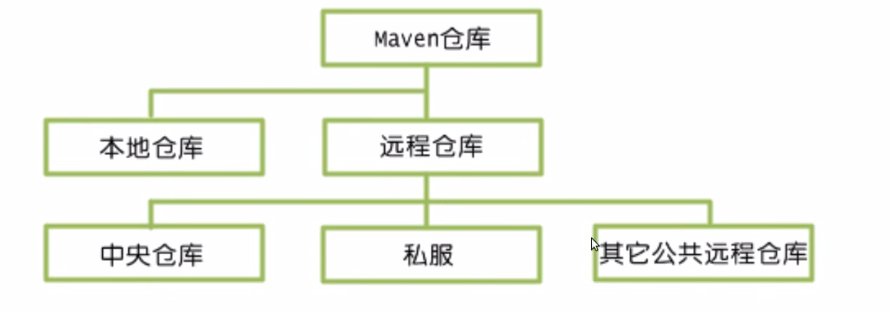
site LifeCycle生命周期 站点信息（项目的描述，项目的使用只能，api描述）

Pre-site Site Post-site Site-deploy

1. 仓库的布局：groupId/artifactId/version/artifactId-version.packaging 根据依赖的坐标进行管理

Snapshot不稳定，尚处于开发中的版本 Release：稳定的版本

1. Maven仓库分类：



1. 本地仓库就是local repository仓库，可以在setting.xml里面修改本地仓库信息
2. 中央仓库就是apache里面的仓库，maven安装的时候就已经有了 <http://repo.maven.apache.org/>
3. Maven依赖

<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.8.2</version>  
 <scope>test</scope>  
</dependency>

Type 依赖类型，对应构建中定义的packaging，不声明默认为jar

Scope 依赖范围

Optional 依赖是否可选

Exclusions 排除传递依赖

1. 依赖范围scope 包括

Compile 编译依赖范围，默认值

Test 测试依赖范围，打包部署的话，就不会用test

Provided 已提供的依赖范围，目标容器或者jdk里面有

Runtime 运行时依赖范围，运行时使用，编译不使用，打包的时候动态载入

System 系统依赖范围

Import 导入依赖范围

1. 传递依赖 mavenDemo1->mavenDemo->junit
2. 依赖冲突
3. 路径优先 A->B->C->X(版本1.0) A->D->X(版本2.0) 优先解析较短路径的X
4. 声明优先 若路径长度相同，则谁先声明，谁被解析
5. 当发生依赖冲突的时候有两种方法
6. 一：在dependencymanagement中指定相应的jar包的版本号
7. 二：用exclusions中排除掉传递依赖的jar包
8. **继承和聚合？**
9. Mvn clean 清理掉之前编译生成的mvn目录
10. IDEA创建Maven Web项目 <http://developer.51cto.com/art/201405/439918.htm>

注意选择maven的时候要选择3.0.5这个版本的

打开Project Structure是采用快捷键 F4 或 用过项目上单击右键即可open module settings

或 File => project structure

[**SSM框架——详细整合教程（Spring+SpringMVC+MyBatis）**](http://blog.csdn.net/zhshulin/article/details/37956105)

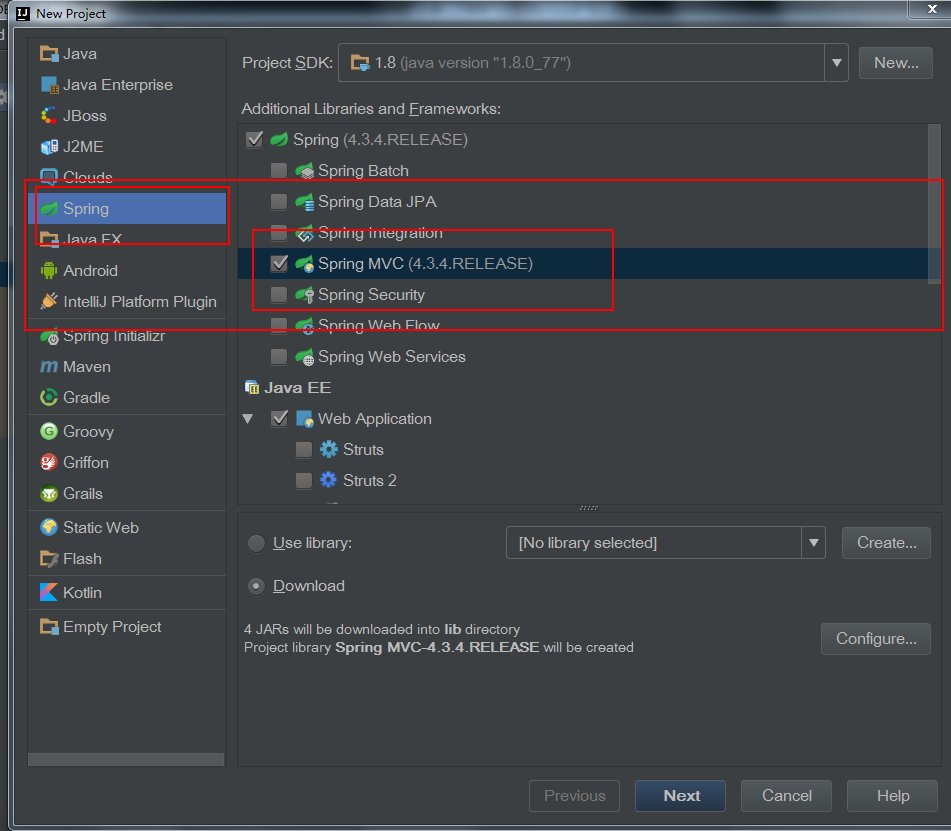
<http://blog.csdn.net/zhshulin/article/details/37956105>

从网上下载下来的文件，要将这个工程变成一个maven的工程，首先要知道这个工程是否有pom.xml文件，其次单击pom.xml文件，然后单击右键，add as maven project即可

IDEA创建Maven Web项目

前置条件， tomcat，maven，jdk本地环境安装好了

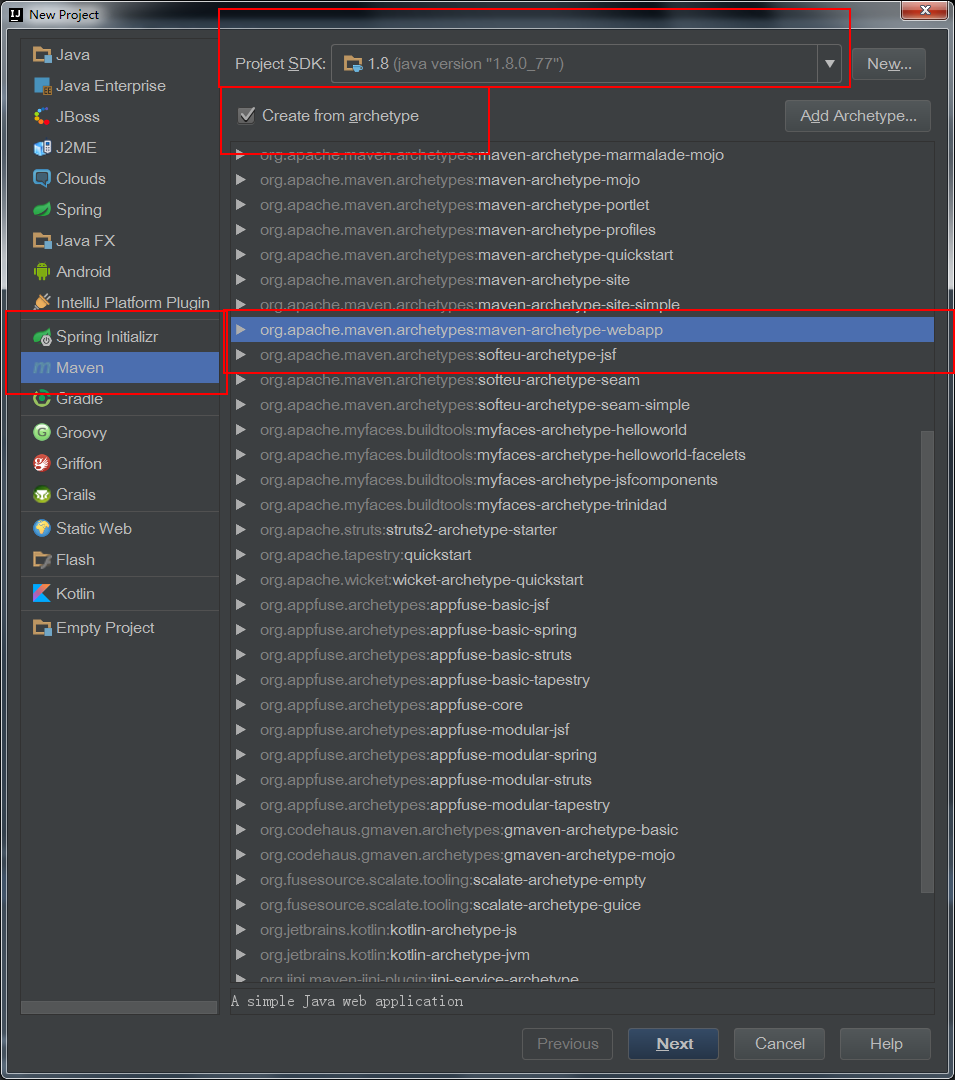
注意idea 16的一个坑



这里生成spring +spring mvc框架的时候，中间有一步是没有template模板可以选择的，这是最坑的

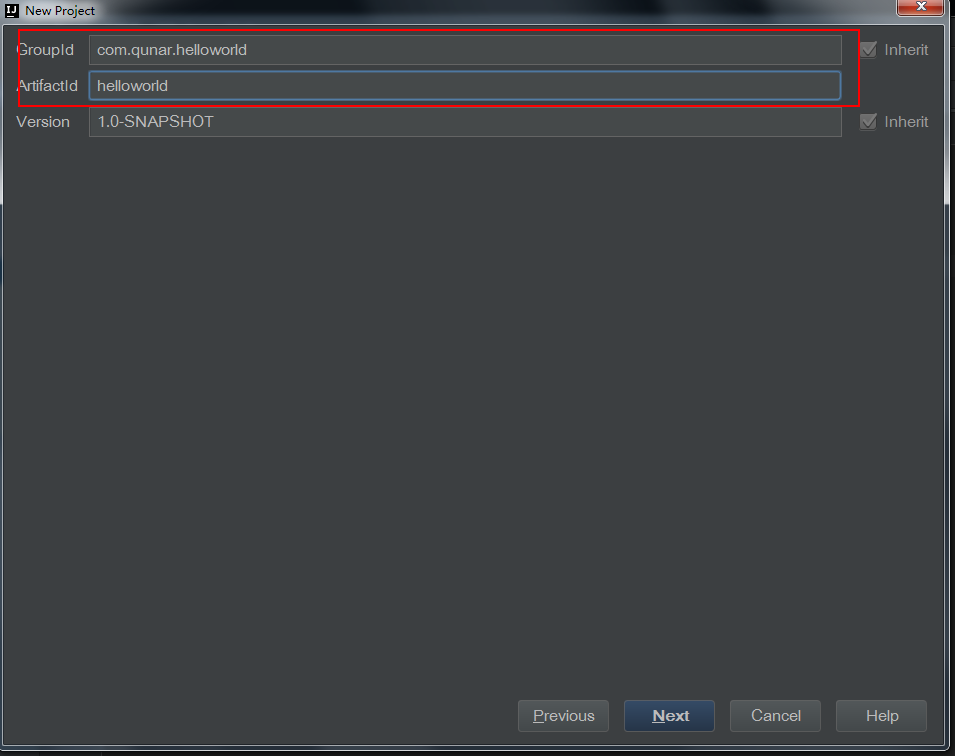
所以采用了另外一种方式去生成这个maven web项目

File->new->project



选择maven项目，create from archetype勾上，选择 webapp那个

Next



Groupid artifactid写上

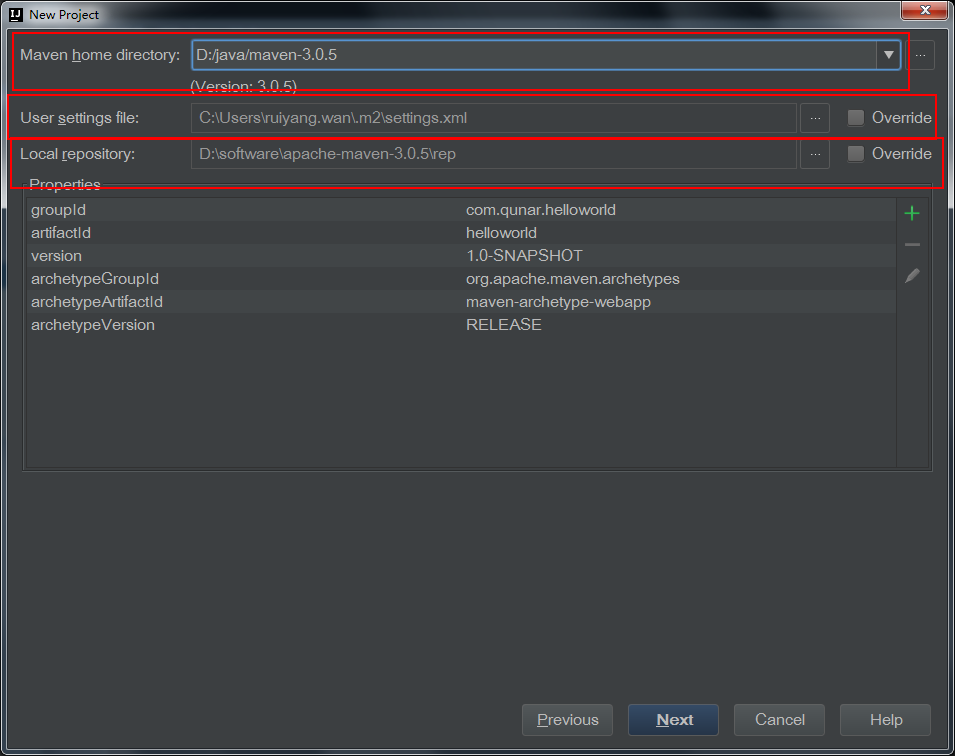
groupid 项目组织唯一的标识符，实际对应项目的名称，项目根目录的名称  
 一般groupID就是填写 com.qunar.test这种，

groupid一般分为很多段，第一段为域，第二段为公司名称

artifactId是工程名，也就是根文件夹名字，比如要输出helloworld，那就写 hello

举个apache公司的tomcat项目例子：这个项目的groupId是org.apache，它的域是org（因为tomcat是非营利项目），公司名称是apache，artigactId是tomcat。

Next



Maven选择你装置的版本，我是3.0.5的版本

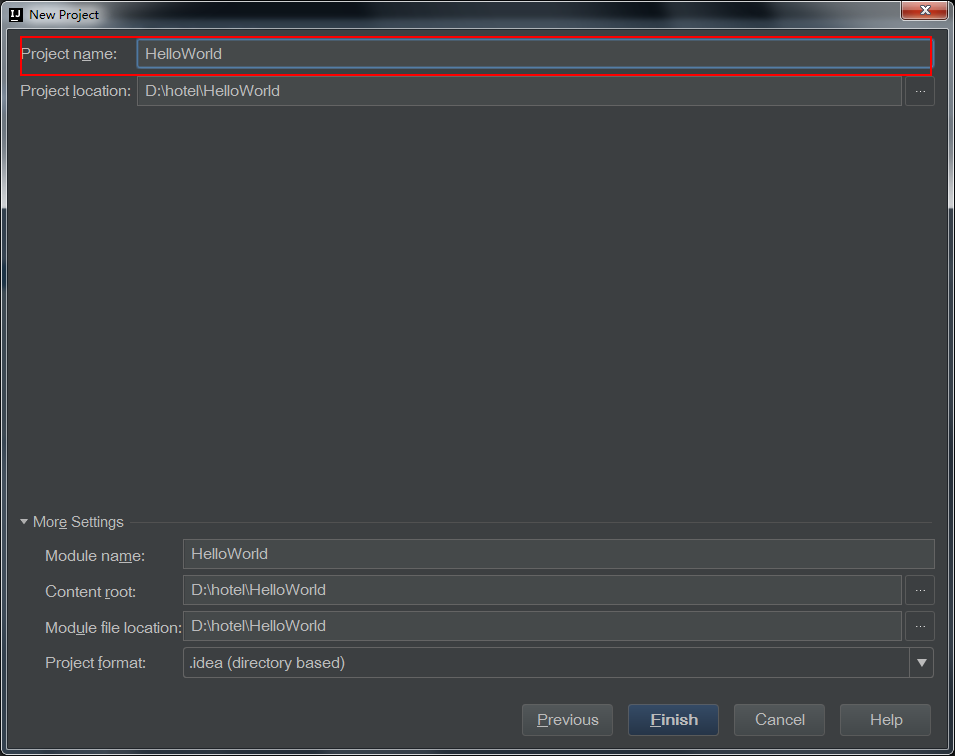
Settings.xml文件 maven的配置文件

[http://www.cnblogs.com/yakov/archive/2011/11/26/maven2\_settings.html setting.xml](http://www.cnblogs.com/yakov/archive/2011/11/26/maven2_settings.html%20setting.xml)文件的说明

local repository就是本地maven仓库

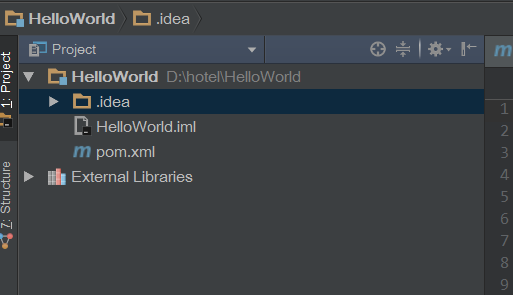
一般是先去本地仓库寻找maven文件，没有的话就去settings.xml中配置的线上maven仓库中下载

Next



填入 project name即可，点finish。

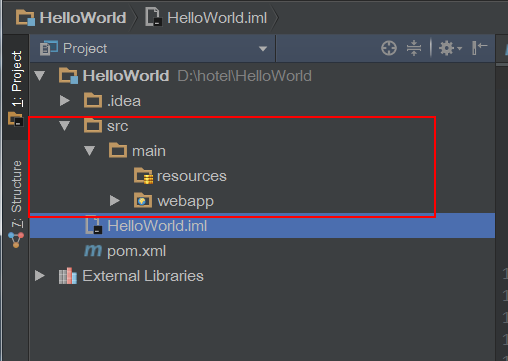
得到的目录（注意这是没有下载完全的目录）



Iml文件是 intellij idea的工程配置文件，里面是当前projec的一些配置信息

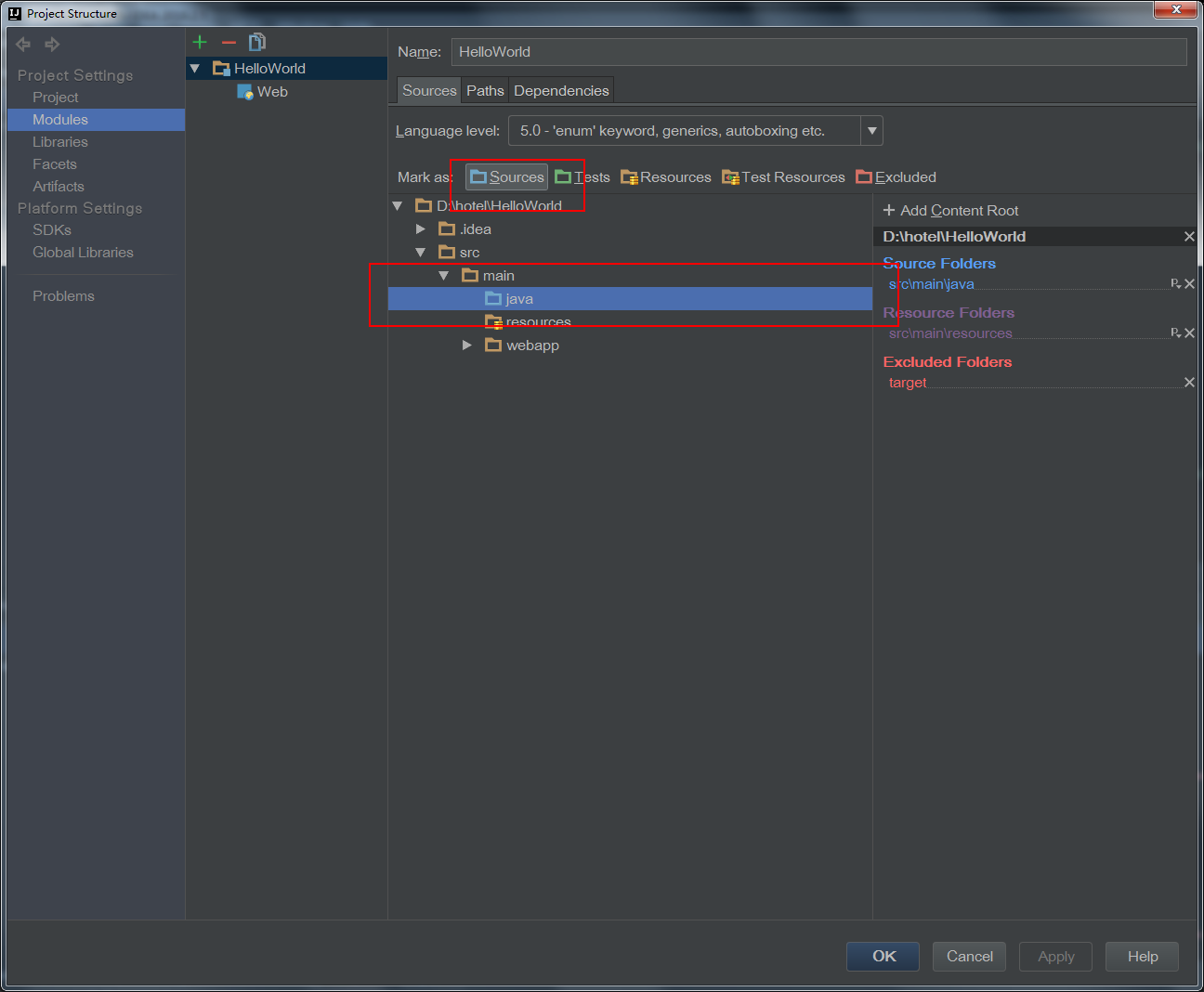
web.xml 是当前这个project是一个web project，里面是web 工程的具体配置信息 初始参数，servlet filter 等等

刚创建完之后，需要稍等一下，加载相应配置文件，最终的目录之后如下图所示



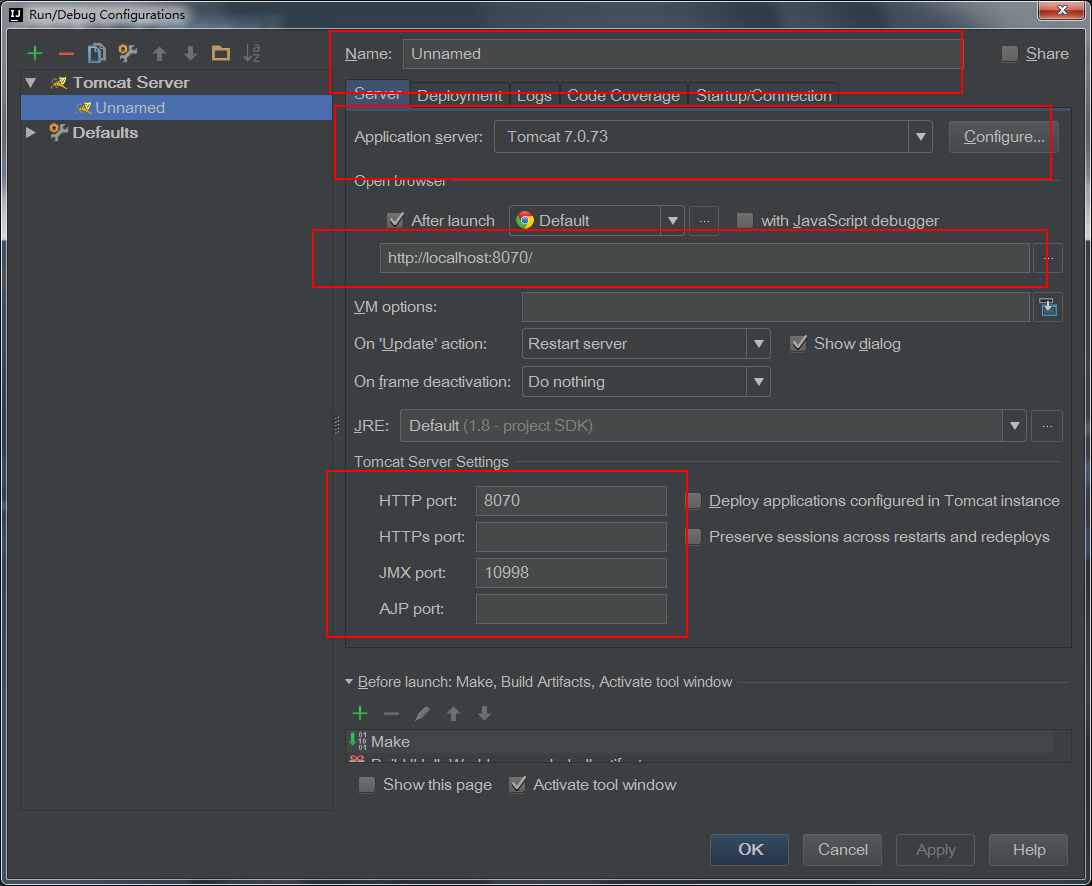
这里我们还缺一个源代码文件，这个java文件夹是自己创建的

单击这个java文件夹，右击，open module setting

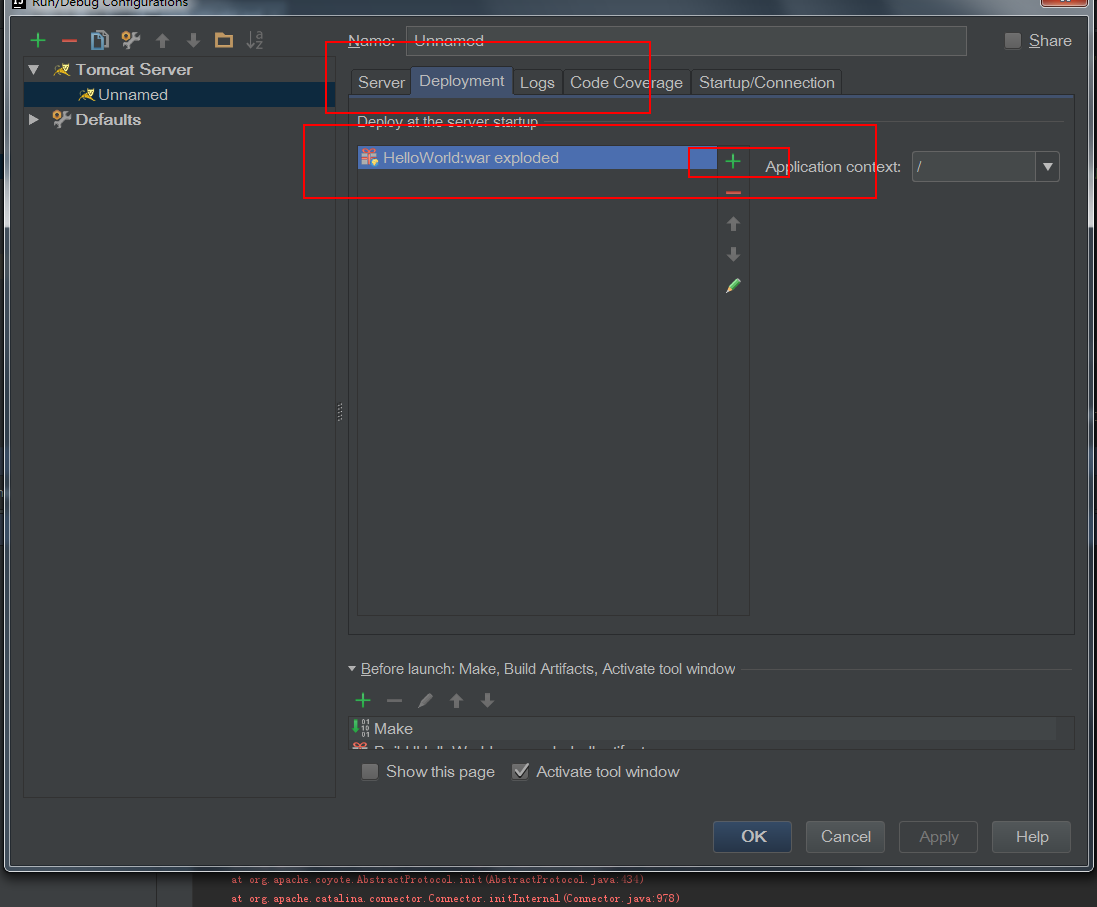


得到蓝色的文件夹之后，表示这个这个文件夹已经变成java项目源代码文件夹了

开始配置tomcat



这些都是自己可以改的，有时候http端口被占用了，需求修改一下端口，记得localhost和http端口需要同时修改



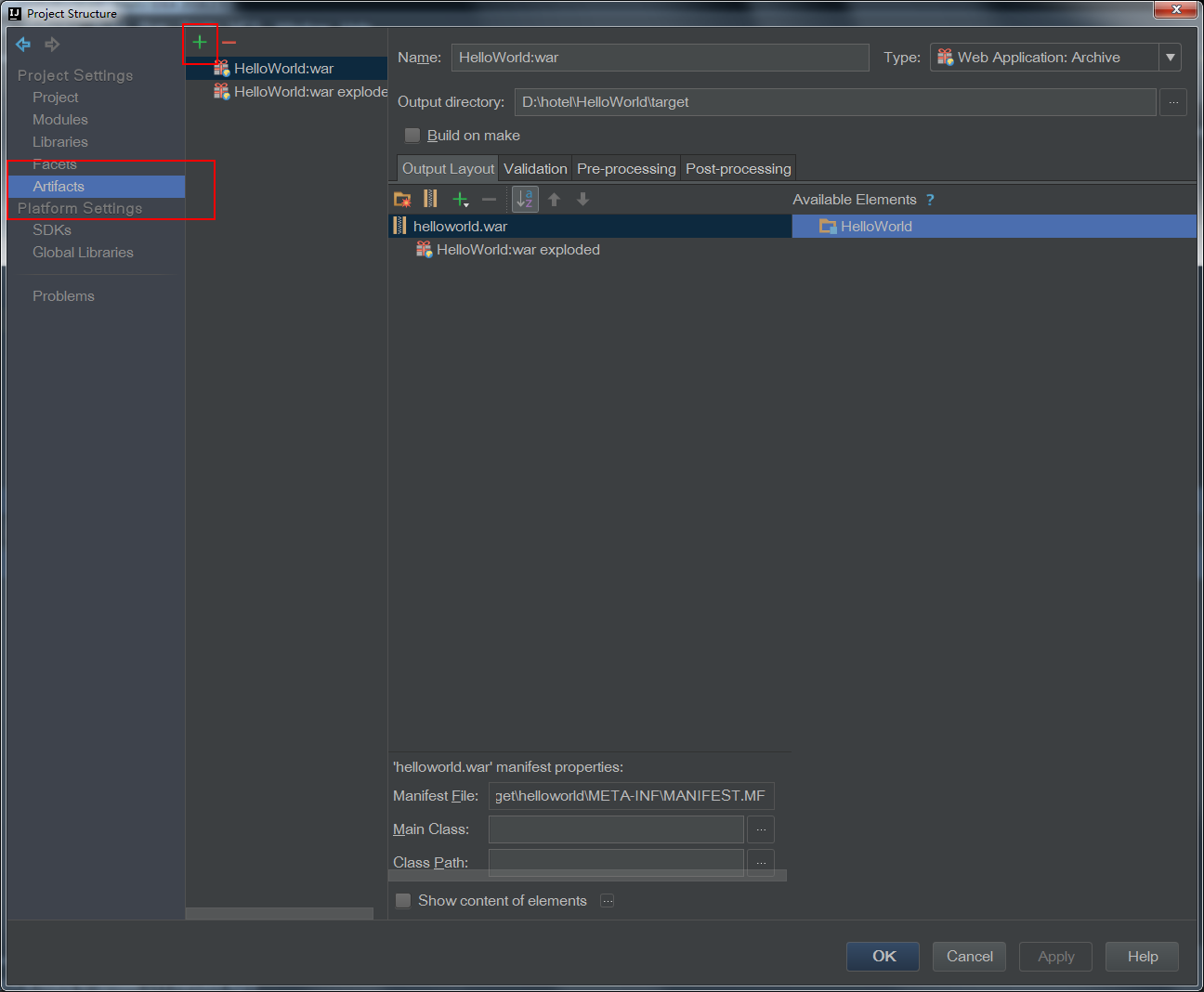
Deployment需要部署一下，绿色加号，选择exploded那个war包

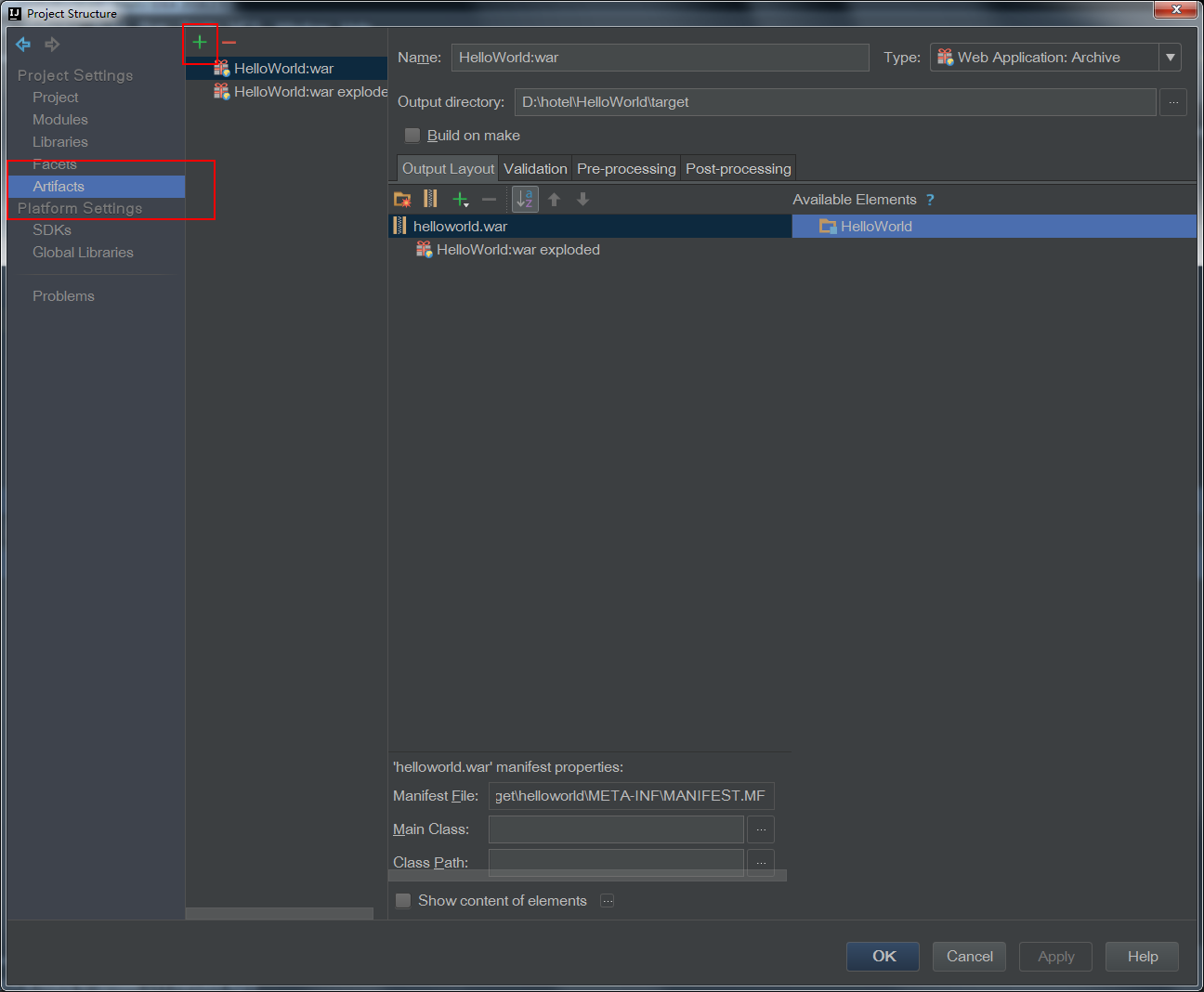
war模式—-将WEB工程以包的形式上传到服务器

war exploded模式—-将WEB工程以当前文件夹的位置关系上传到服务器

在配置服务器时需要选择war exploded

下面是生成war包的方式





生成exploded war包的方式

Tomcat配置好了，就可以运行了

自动在默认浏览器页面输出 hello world

1. Fiddler页面详解<http://www.cnblogs.com/chengchengla1990/p/5681775.html>
2. Mybatis 认识 sql代码和java代码分离的

工作流程：

读取配置文件(基本配置文件，连数据库的信息)

生成SqlSessionFactory对象，sqlsession的工厂，用于建立与数据库之间的会话

建立SqlSession对象，执行sql语句，调用Mybatis提供的API函数

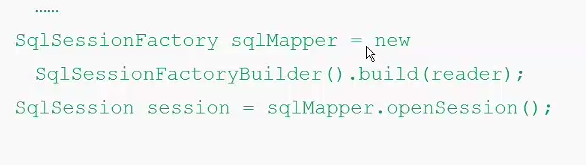
Sql语句放在Map配置文件中

返回结果，关闭SqlSession（这个流程也是hibernate的工作流程）

基本配置文件包含两个方面，链接数据库信息（可能有多个），map配置文件的位置

SqlSessionFactory是一般而言一个程序只有一个，从程序开始到程序结束（生命周期是程序级）

SqlSession 生命周期是过程级的，也许一个方法中要连接数据库，就会建立SqlSession



三种引用map文件的方式

一般mybatisConfig.xml中引用map文件，也有可能分级拆开

1. 相对路径引用

<mappers> <mapper resource=”abc/def/mng.xml/”> </mappers>

1. 绝对路径引用

<mapper url=[file:///asd/adf/wec.xml](file:///C:\asd\adf\wec.xml)/>

1. 包路径引用

<package name=”com.tiantian.mybatis.mapperinterface”/>

Map文件 数据库表的与sql语句的映射

比如

mybatisConfig.xml配置文件,连接数据库的信息<environments>+<mapper>找到相应的map文件

数据库中建立JikeUser表，有字段id，userName，password（数据表）

键入一条记录，id=1的记录

建立JikeUser类,注意有这些成员变量，还有对应的getter，setter方法（数据映射类）

查询代码

JikeUser temp=session.selectOne(“findById”, 1); findById是xml中的id 1是条件id=1

System.out.println(“name=”+ temp.getUserName());

在jikeUser.xml文件中

<mapper namespace=”domain.blog.mappers.AuthorMapper”>

<select id=”findById” parameterType=”int” resultType=”jike.book.pojp.JikeUser”>

Select \* from jikeUser where id=#{id}

</select>

</mapper>

Mybatis大总结

<http://blog.csdn.net/techbirds_bao/article/details/9233599/>

<http://limingnihao.iteye.com/blog/781878>

ResultMap和resultType之间的区别于差异

<http://blog.csdn.net/woshixuye/article/details/27521071>

mybatis 一对一与一对多collection和association的使用

<http://www.cnblogs.com/yansum/p/5819973.html>

1. Mybatis里面foreach collection的用法，主要用于构建 SQL 中的 in 语句

<select id="dynamicForeach2Test" resultType="Blog">

select \* from t\_blog where id in

<foreach collection="array" index="index" item="item" open="(" separator="," close=")">

#{item}

</foreach>

</select>

这里collection 的值就是外面传进来的变量，list，array就是其名字，map就是其中某个键的名字，值是list或者array的对象

<http://www.cnblogs.com/fangyu19900812/p/6046209.html>

<http://blog.csdn.net/qq309414557/article/details/51554281>

<http://www.cnblogs.com/fangyu19900812/p/6046209.html>

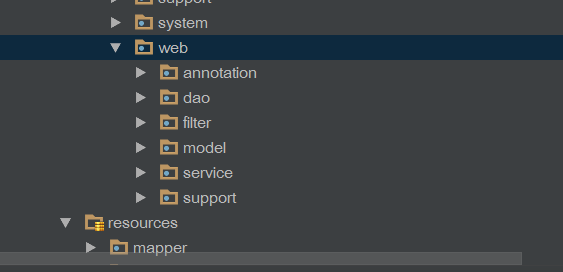
1. Mybatis里面 like模糊查询

Nickname like concat(‘%’, ‘${name}’, ‘%’)

wrapper\_name like concat('%', #{wrapperName}, '%')

username like '%${value}%'

1. #{} 与 ${} 的区别
2. #{} 是经过预编译的，是安全的，可以有效的防止SQL注入，会将传入的数据当成一个字符串，会对数据增加一对双引号
3. ${} 未经过预编译，仅仅是取变量的值，不会修改或转移字符串，是非安全的，存在SQL注入的现象，只能手动过滤处理
4. 目录的理解



Annotation都是自己写的注解

Dao就类似于mapper文件的聚集地

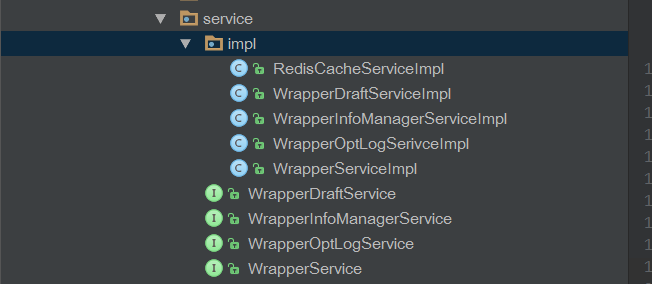
Filter类似于一些拦截访问，获取信息

Model层就是数据对象传来传去

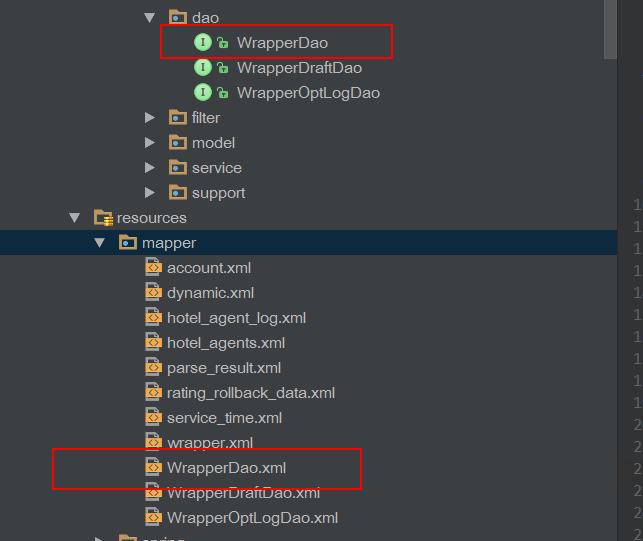
Support相当于支持层，支持所有的功能

Service层、



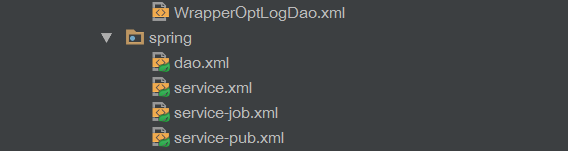


各种service就是各种提供服务的规范，impl里面就是各种服务规范的具体实现



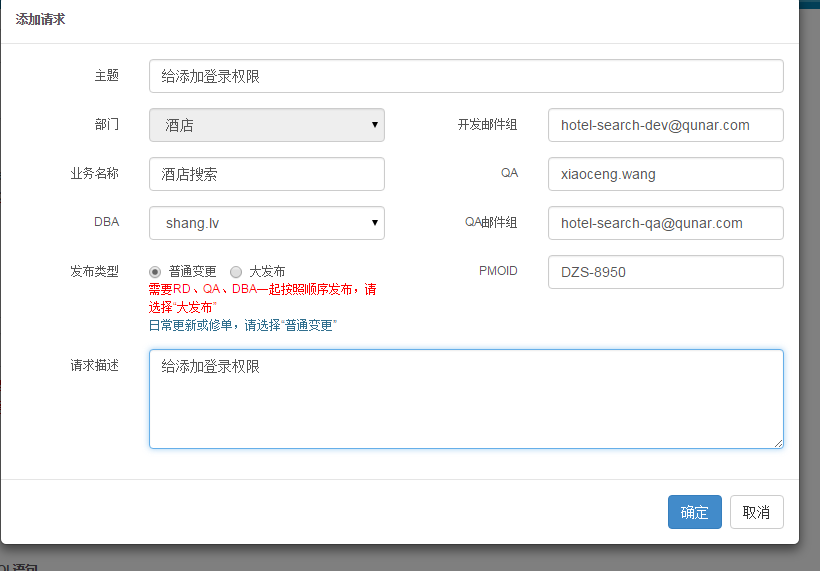
Dao里面的mapper（实际上是接口）和resources里面的xml文件一一对应

实现该接口的所有类，里面相同的方法都调用的是同一个mapper文件对应xml文件里面同一个id对应的方法



Spring里面通常与数据库打交道的mybatis文件会放在这个spring文件夹的dao.xml文件下

1. 线上数据库权限申请<http://putin.corp.qunar.com/qso/dev/index.html#/home>

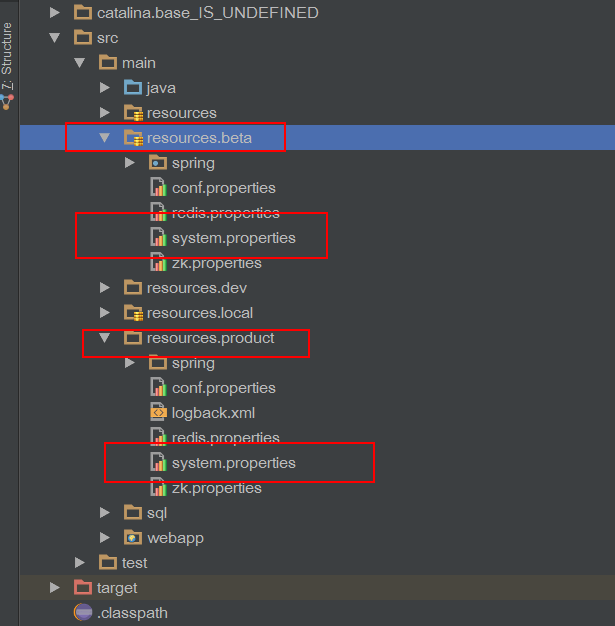


1. 这个做完了还得让对应的qa去审核

开发邮件组 [hotel-search-dev@qunar.com](mailto:hotel-search-dev@qunar.com)

QA邮件组 [hotel-search-qa@qunar.com](mailto:hotel-search-qa@qunar.com)

业务名称：酒店搜索



测试版数据库和线上版数据库的连接信息都是在各自的resource下，注意product是线上

1. 上线代码一般是api这些先上线，接着前端上线，最后后端上线，api是后端或前端的支持，前端项目可能做一下前端脚本处理，最后是后端的处理,其实api有时候没有太大的作用，除非是其他人也要用，做成jar包
2. Beta版的配置文件是在一台服务器上，只有这一台服务器，传上去之后是存在了数据库当中，接着有服务器需要就可以从数据库中定时下载配置文件

第一个项目上传的配置文件hagentBusAndDeliver.xml,定时加载改配置文件

第一个项目hotel\_agent 的beta版url <http://l-hagent1.h.beta.cn6.qunar.com:8080/index>

第一个项目上线的sql语句

set names utf8;

use qhagent;

ALTER TABLE qhagent.hotel\_agents ADD compete\_attach VARCHAR(10) NOT NULL DEFAULT '' COMMENT "竞对归属", ADD compete\_offer\_resource VARCHAR(10) NOT NULL DEFAULT "" COMMENT "竞对报价来源";



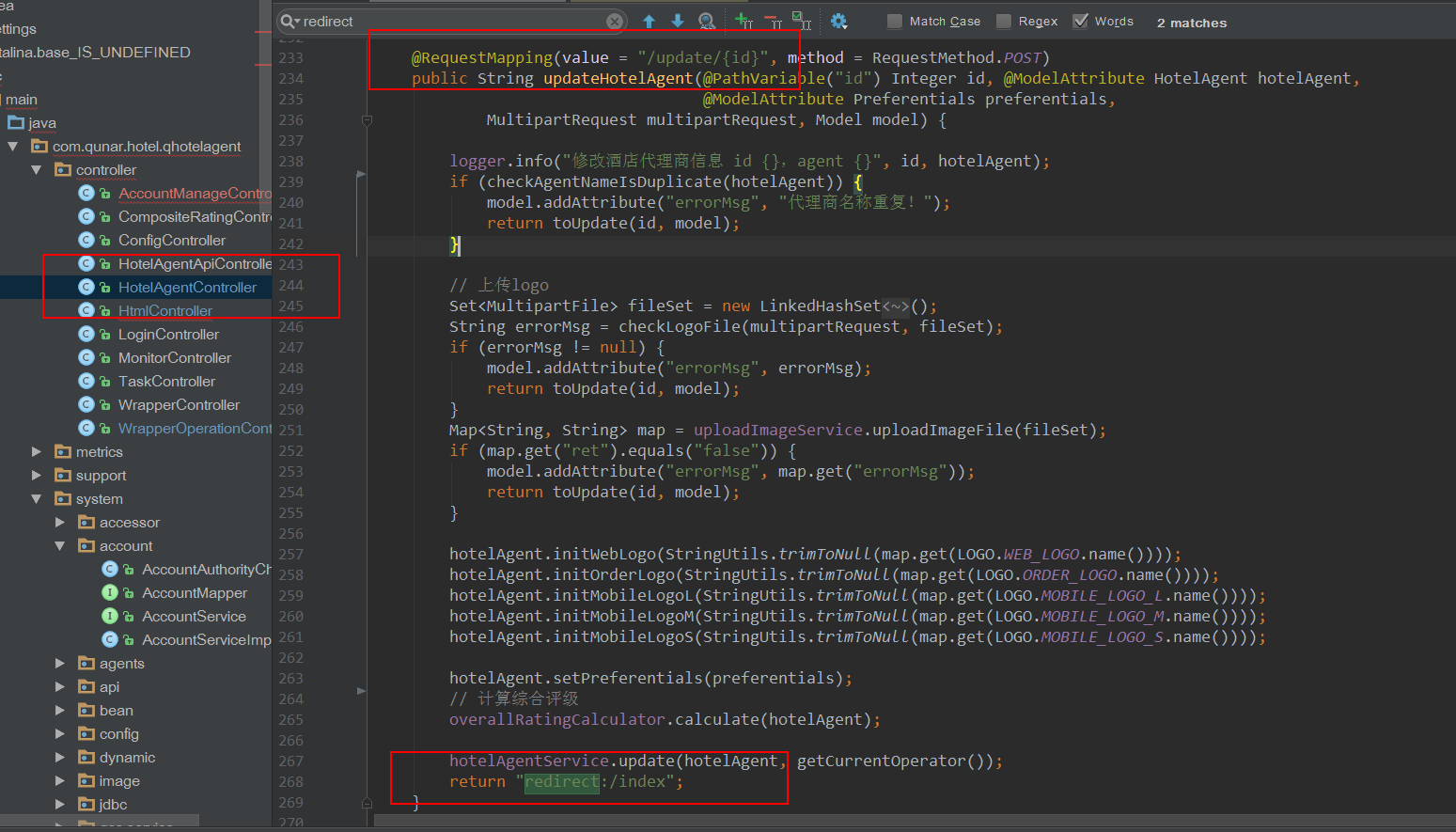
1. <http://wiki.corp.qunar.com/pages/viewpage.action?pageId=128726623> 快速登录跳板机wiki
2. Ops是服务器申请权限的地方，putin是与数据库相关的地方，线上数据库改动之类的

Wiki是wiki it是it工作台

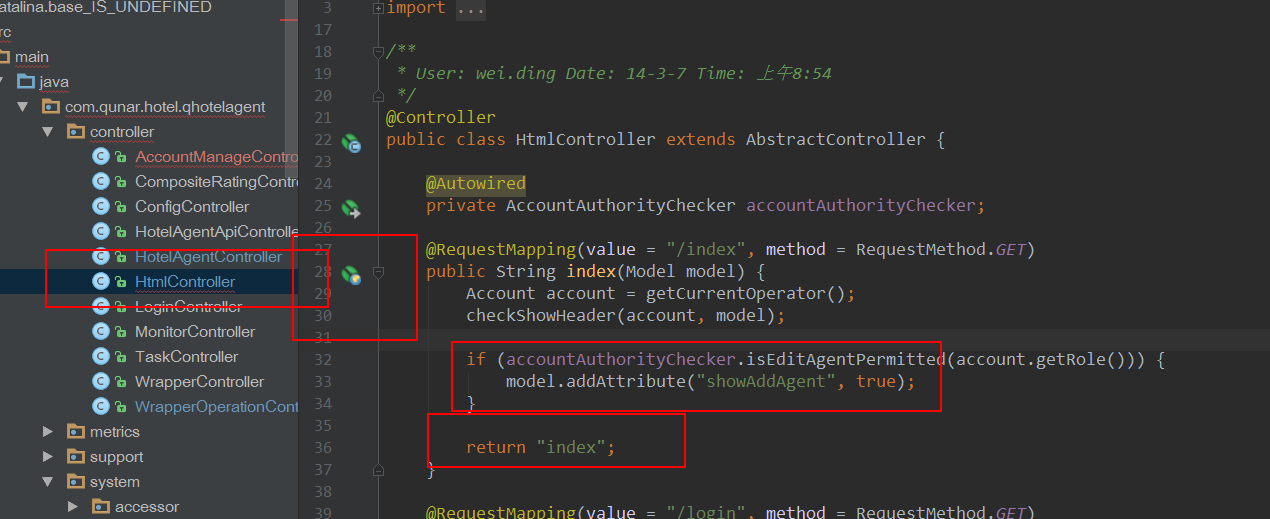
Pmo是JIRA bugfree 是修改bug的地方

1. 每次做新项目的时候，sql语句都要记得记下来，因为线上上线的时候都是需要这些sql语句
2. 一个项目分前端一个仓库，后端一个仓库，前后端发布在不同的服务器上，这个时候要做一个关联，在发布的qtaq上面，前端写上后端的分支，这样就可以关联上了
3. Hotel\_agent项目怎么从一个/agent/update调到 /index 页面仍然可以加载数据分页

首先在controller里面action方法最后跳转到 /index页面

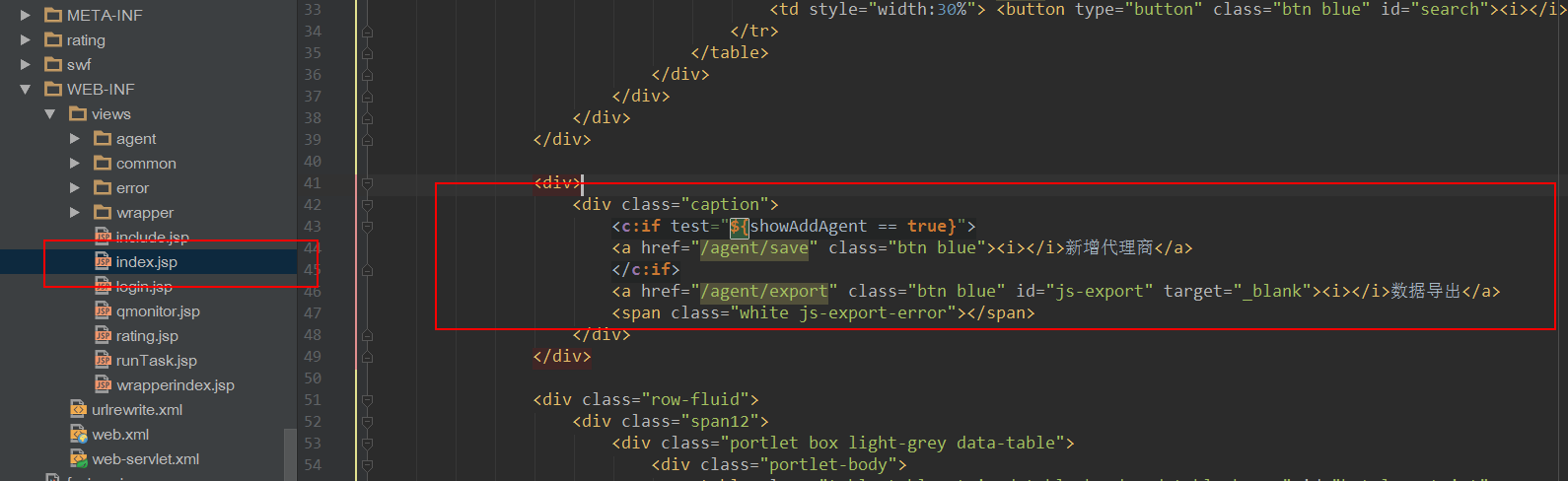


跳转到HtmlController页面，这里只拿到 操作者是否有权限看到分页数据的变量showAddAgent

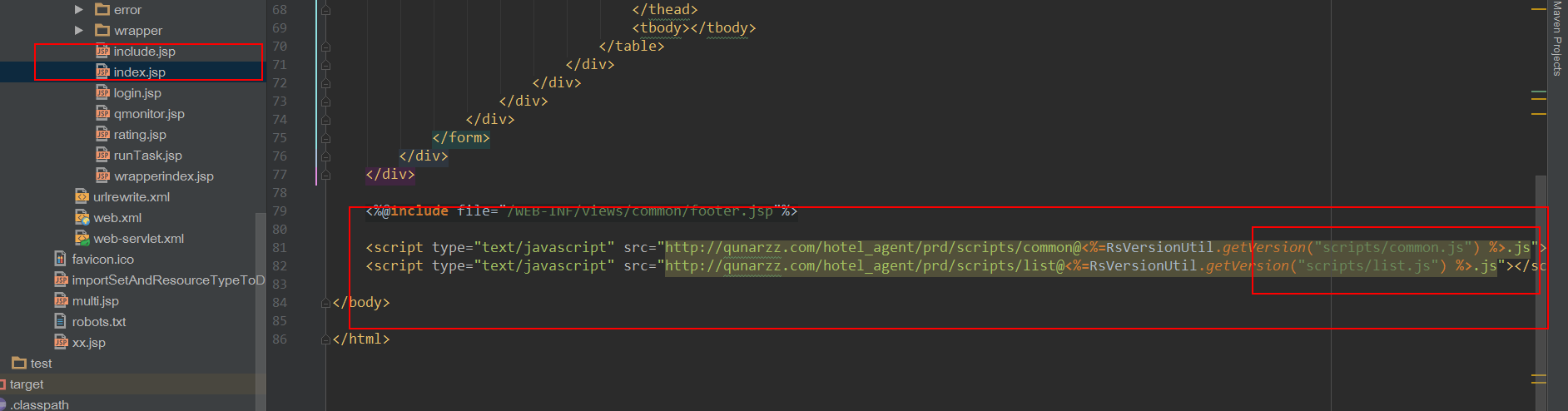


但是最后仍然返回到了/index的页面，这个时候打开index.jsp文件

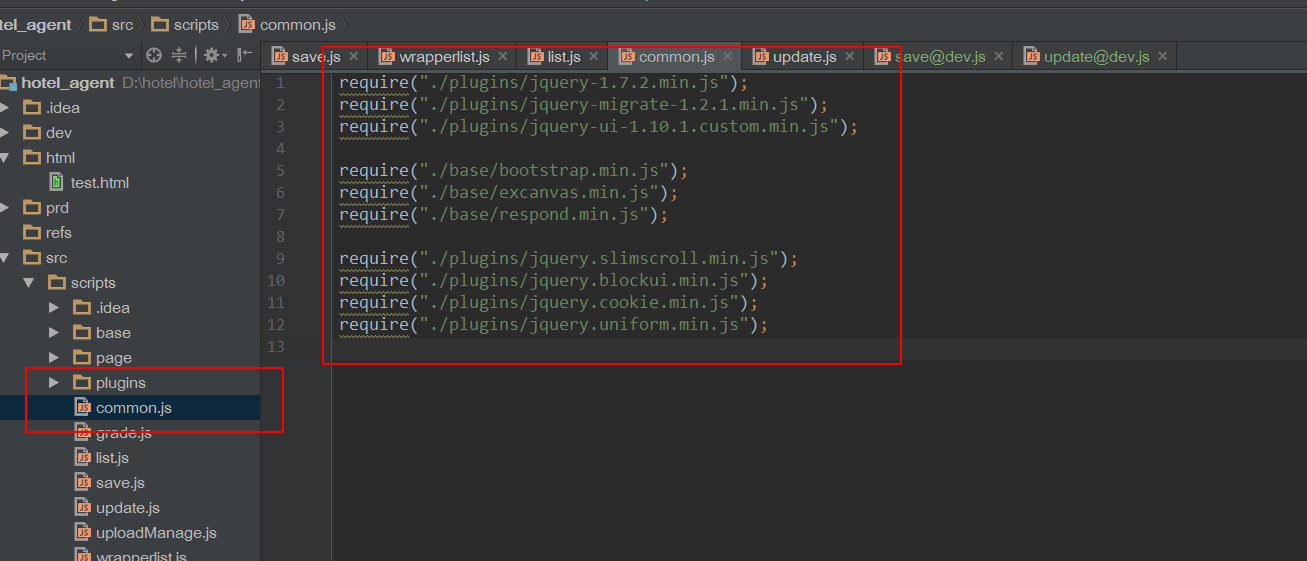
在这里对这个字段showAddAgent进行了判断处理，是否有 新增代理商 这个选项



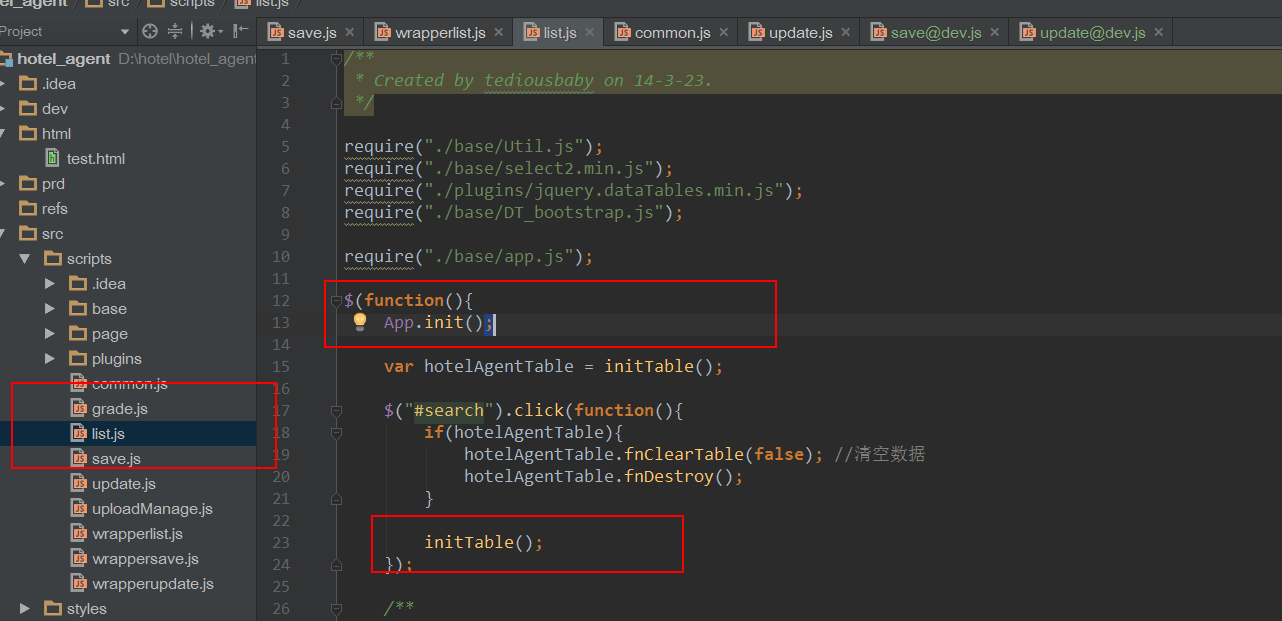
接着加载了两个js文件，包括common.js和list.js



Common.js文件里面包含了多个js文件

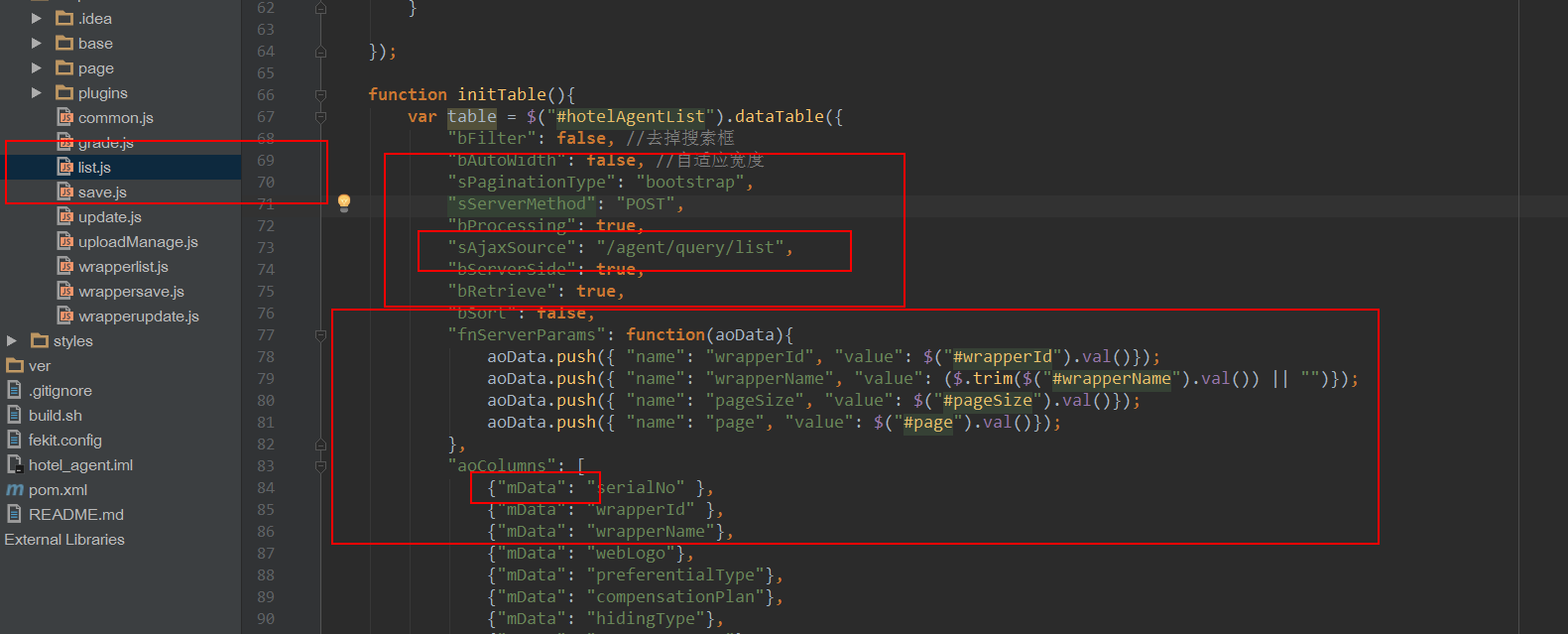


List.js文件里面包含了多个js文件，但是也加载了App.init() 方法和initTable方法



这里进行了一次ajax请求，请求的路径是 /agent/query/list

（这里才是与数据库交互得到的分页数据），下面代码就是对返回来的数据用插件分页处理



1. Spring MVC

之**@RequestParam**

在mvc后台controller层获取前端传来的参数的方式主要有两种，

一种是 request.getParameter(“name”)

一种是 注解@RequestParam 直接获取

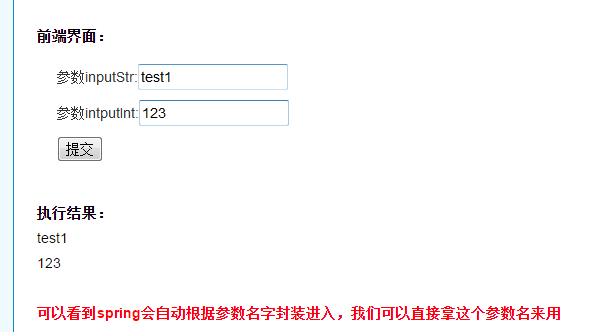
比如

后端代码：注意@RequestMapping里面的路径

1. @RequestMapping("testRequestParam")
2. **public** String filesUpload(@RequestParam String inputStr, HttpServletRequest request) {
3. System.out.println(inputStr);
5. **int** inputInt = Integer.valueOf(request.getParameter("inputInt"));
6. System.out.println(inputInt);
8. // ......省略
9. **return** "index";
10. }

前端html代码，注意action的路径

1. **<form** action="/gadget/testRequestParam" method="post"**>**
2. 参数inputStr:**<input** type="text" name="inputStr"**>**
3. 参数intputInt:**<input** type="text" name="inputInt"**>**
4. **</form>**



作用：

可以对前端传入参数指定参数名

1. // 下面的对传入参数指定为aa，如果前端不传aa参数名，会报错
2. @RequestParam(value="aa") String inputStr

可以通过required=false或者required=true来要求@RequestParam配置的前端参数是否一定要传

1. // required=false表示不传的话，会给参数赋值为null，required=true就是必须要有
2. @RequestMapping("testRequestParam")
3. **public** String filesUpload(@RequestParam(value="aa", required=**true**) String inputStr, HttpServletRequest request)

注意这里不传的话，是给参数复制为null，如果参数是int类型的话，不能不传值了

<http://blog.csdn.net/kobejayandy/article/details/12690161>

<http://825635381.iteye.com/blog/2196911>

1. 一般而言
2. Controller里面的action获取前端传来的一个对象，注意这个对象也要命名一个类，这个类也要
3. public String saveHotelAgent(@ModelAttribute HotelAgent hotelAgent, @ModelAttribute Preferentials preferentials, MultipartRequest multipartRequest, Model model) {  
    logger.info("新增酒店代理商 agent {}", hotelAgent);  
     
    if (!checkQssWrapperId(hotelAgent)) { //前面这些检验方法都是可以在一个controller里面定义  
    model.addAttribute("errorMsg", "根据Wrapper ID无法从QSS查询到相应的Wrapper信息，请核对后重新填写");  
    return "/agent/save";  
    }  
     
    if (checkAgentNameIsDuplicate(hotelAgent)) {  
    model.addAttribute("errorMsg", "代理商名称重复！");  
    return "/agent/save";  
    }  
    try {  
    hotelAgentService.insert(hotelAgent, getCurrentOperator());

//这里调用方法是service接口里面实现的impl实现的方法insert()

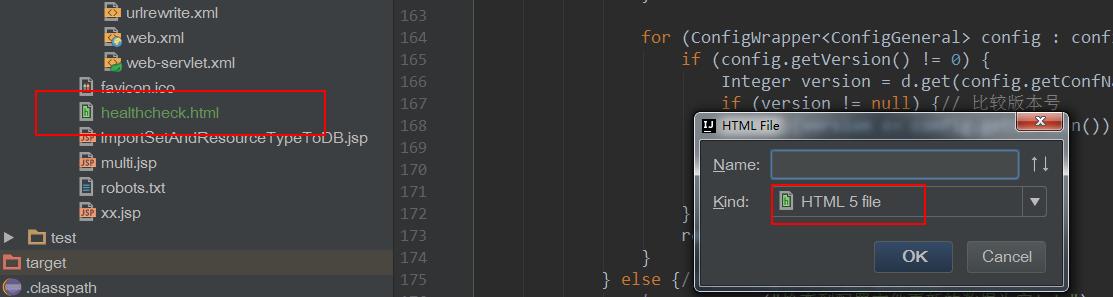
//在insert方法中会调用一个xxxmaper接口实现的对象中的方法，这个xxxmaper对应得就是

xxx.xml文件中的与数据库交互的方法了.这种映射是通过包的方式mybatis  
 } catch (Exception e) {  
 errorMsg = changeExToMsg(e);  
 logger.warn(errorMsg, e);  
 model.addAttribute("errorMsg", errorMsg);  
 return "/agent/save";  
 }  
 return "redirect:/index";  
}

1. 怎样在本地加载QSchedule里面的配置文件

<http://dev.qschedule.corp.qunar.com/jobs.do?jobName=&appCode=h_hotel_agent&lastState=0>

首先你要在本地WEB-INF文件夹下面加上healthcheck.html文件，注意只要新建即可



接着重启tomcat服务器

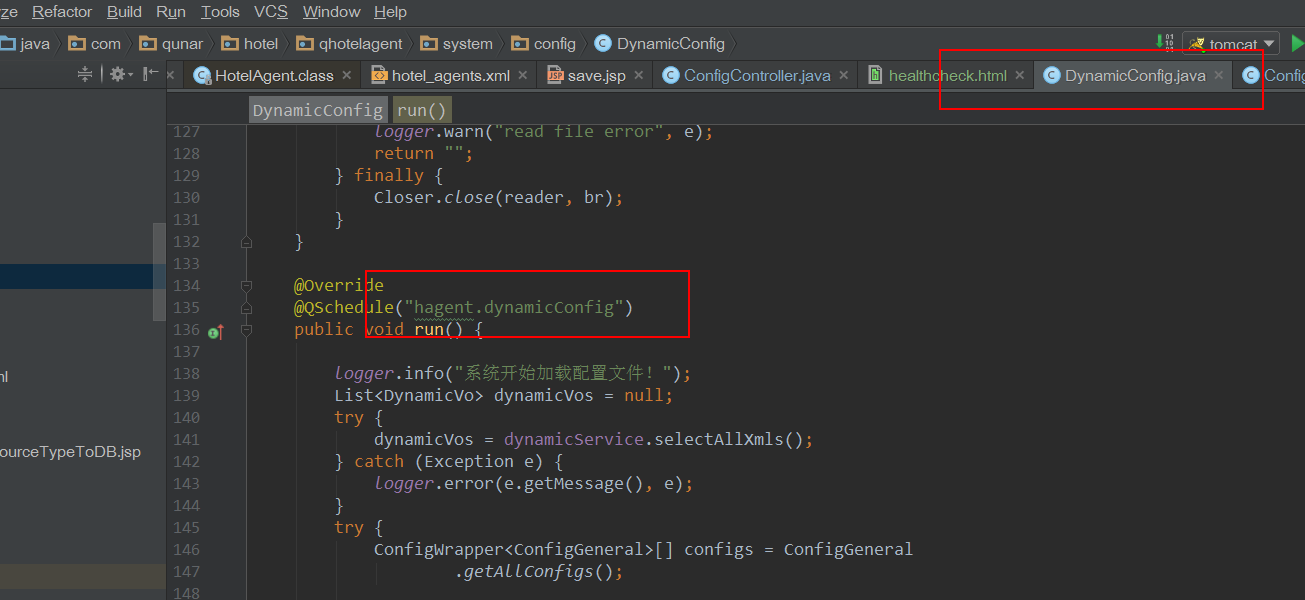
WEB-INF是Java的WEB应用的安全目录。所谓安全就是客户端无法访问，只有服务端可以访问的目录。

如果想在页面中直接访问其中的文件，必须通过web.xml文件对要访问的文件进行相应映射才能访问。

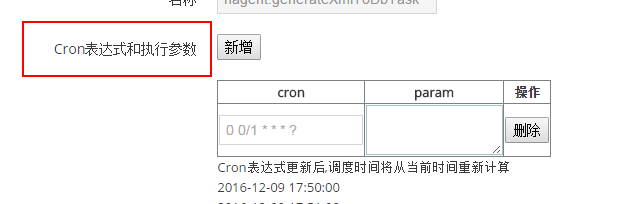
进入线上的QSchedule的网站



这时候就要看你的代码执行的是哪个名称的

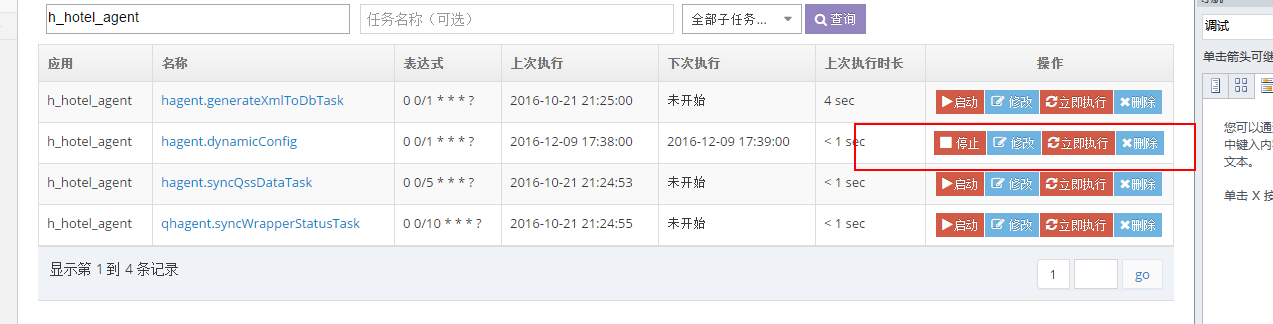


这里注意有一个表达式，这是一个Cron表达式



这个表示一分钟执行一次，也就是运行@QSchedule下面的方法，一分钟就线上推动线下执行一次

然后等待线上定时执行就可以了



这里注意，最后在提交到线上的时候，不要把这个healthcheck文件提交到线上，因为线上在编译的时候，

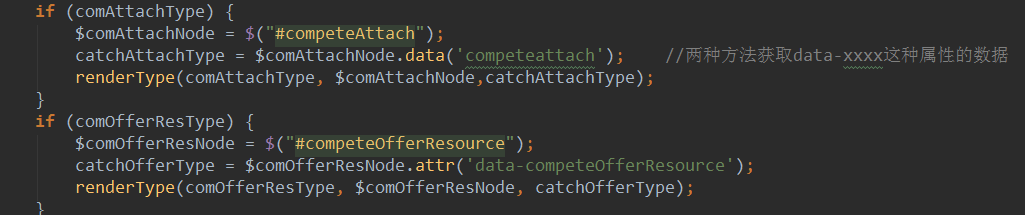
是会自动生成这个文件的

1. Js自定义data-\* 属性和jquery的 data() 方法的使用

往HTML标签上添加任意以 "data-"开头的属性，这些属性页面上是不显示的，它不会影响到你的页面布局和风格，但它却是可读可写的。

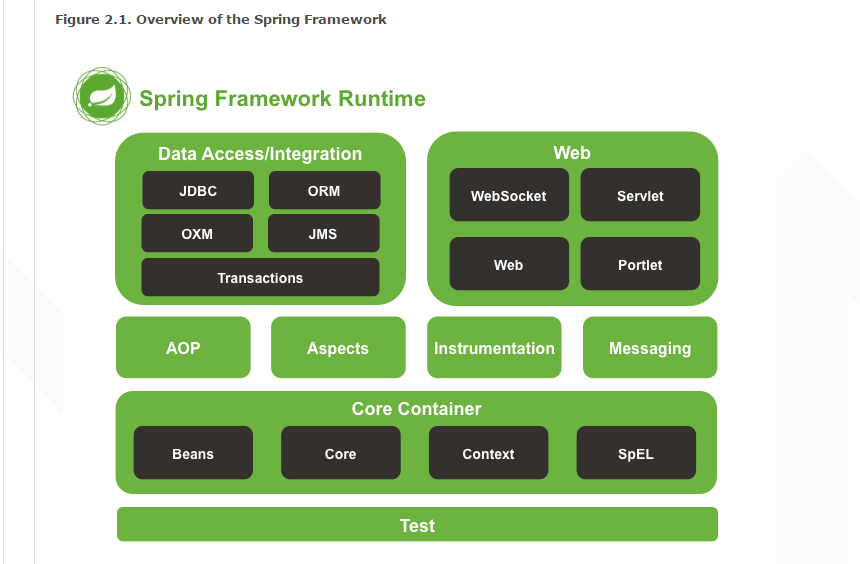
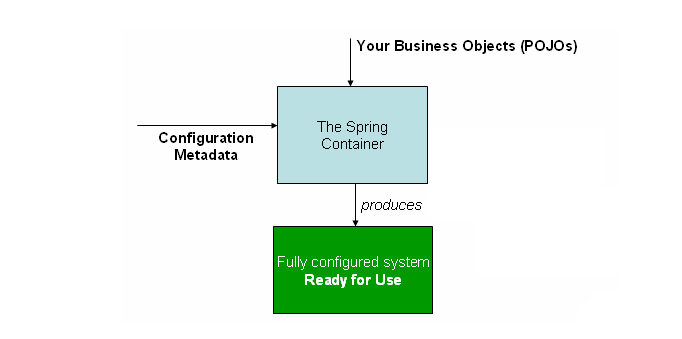


在js文件中获取数据的时候



即可以使用data方法，但是是要去掉data-的属性 data(‘xxx’)

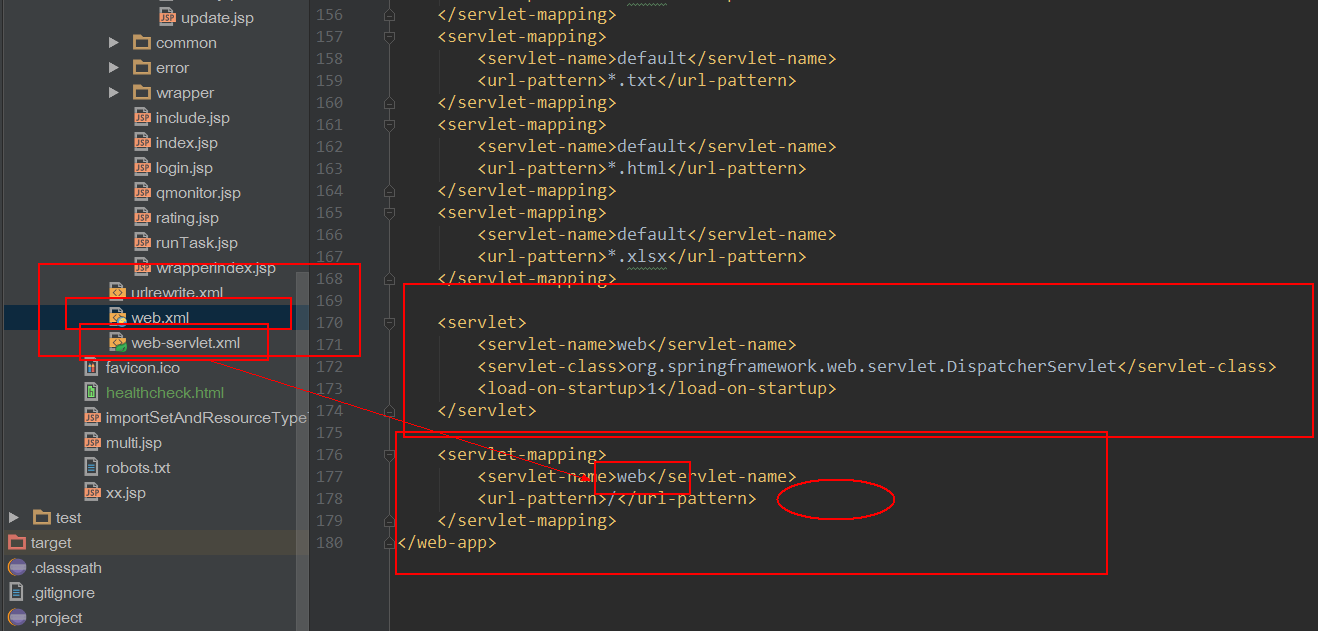
也可以使用attr方法，这个可以把 data-xxx 属性全加上 attr(‘data-xxx’)

1. 内聚和耦合？？？？？？？？？？？
2. Spring FrameWork
3. 
4. 框架就是一套规范或者规则，框架就是别人搭好的舞台，我们来表演。
5. 框架一般是封装了逻辑，高内聚的。类库则是松散的工具组合。框架是楼房架构，类库是钢材
6. IOC 控制反转，应用程序并不自己创建和维护对象，而是由外部同期负责创建和维护
7. DI 依赖注入 创建对象并组装对象之间的关系，当初始化的时候就会创建大量的类
8. The Spring of Container
9. 
10. 在IOc中，把所有的对象叫Bean
11. 对于Bean的使用，一种是基于Xml文件的配置，一种是注解的方式
12. <bean id= ”oneInterface” class=”com.qunar.ioc.interfaces.OneInterfaceImpl”> </bean>
13. 接口名 接口的实现类名字
14. 中介就相当于Bean，我们要租房，联系中介，中介就给了我们租房信息，然后经过一系列逻辑交谈，得到了房子，入住
15. Bean容器初始化
16. 基础：两个包
17. –org.springframework.beans
18. –org.soringframework.context
19. –BeanFactory 提供配置节后和基本功能，加载并初始化Bean
20. –ApplicationContext保存了Bean对象并在Spring中被广泛使用
21. 方式：ApplicationContext
22. –----加载本地文件，指定放在本地某路径下

FileSystemXmlApplicationContext context = new FileSystemXmlApplicationContext(“F:/worksoace/appcontext.xml”);

1. –----Classpath路径下加载文件
2. ClassPathXmlApplicationContext = new ClassPathXmlApplicationContext(“classpath:spring-context.xml”);
3. –----Web应用中依赖Servlet或Listener,这个是在xml文件中完成的

Web.xml文件中会在Servlet中加载bean文件



1. DispatcerServlet 浏览器端的请求的分发，在Dispatcher初始化的过程中，初始化了

上下文（ApplicationContext）

策略（HandlerMapping，HandlerAdapter等）

HandlerAdapter Handler是一个类，在DispatcerServlet内部使用的类，DispatcerServlet调用controller中间一种过渡对象 HandlerAdapter 是一个适配器模式，将每个controller设置为合适的Handler

HandlerInterceptor Interceptor是一个拦截器，

HandlerInterceptor 是一个接口，有三个方法

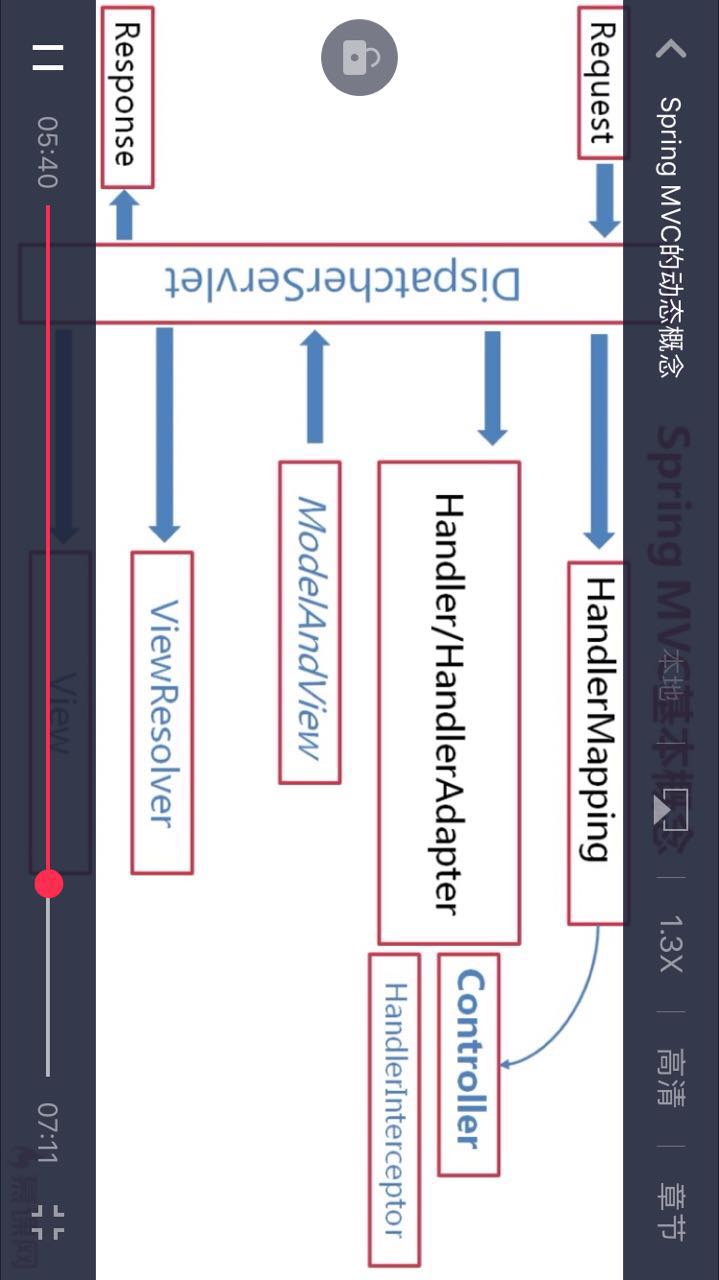
HandlerMapping handler和controller的一个映射，返回给Handler,HandlerAdapter ,HandlerInterceptor

所谓的HandlerMapping和HandlerAdapter完成的正是请求的映射和适配

HandlerExecutionChain 就是上面的三个构成了一个执行链条 ，就是java反射机制实现的

ModelAndView 包含Model和Map，DispatcerServlet最后都会将其转化为model

ViewResolver 根据配置找出，提供给DispatcerServlet相应的view，视图解析器

1. Spring注入：在启动Spring容器加载bean配置的时候，完成对变量的复制行为
2. 常用的注入方式：设值注入，构造注入

启动一个WEB项目的时候，WEB容器会去读取它的配置文件web.xml

以下都是在WEB项目还没有完全启动起来的时候就已经完成了的工作。



创建一个ServletContext（servlet上下文）

<context-param>转换为键值对，并交给servletContext。

<context-param>->

<listener> -> 监听

<filter> -> 拦截，过滤

<servlet> 加载其他文件

<filter-mapping>必须出现在<filter>之后，<servlet>同<filter>类似

<http://mianhuaman.iteye.com/blog/1105522>

<http://blog.csdn.net/believejava/article/details/43229361>

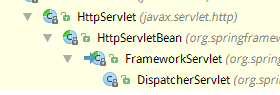
<http://blog.csdn.net/shanliangliuxing/article/details/7458492>

Tomcat服务器

首先会读取web.xml文件的配置

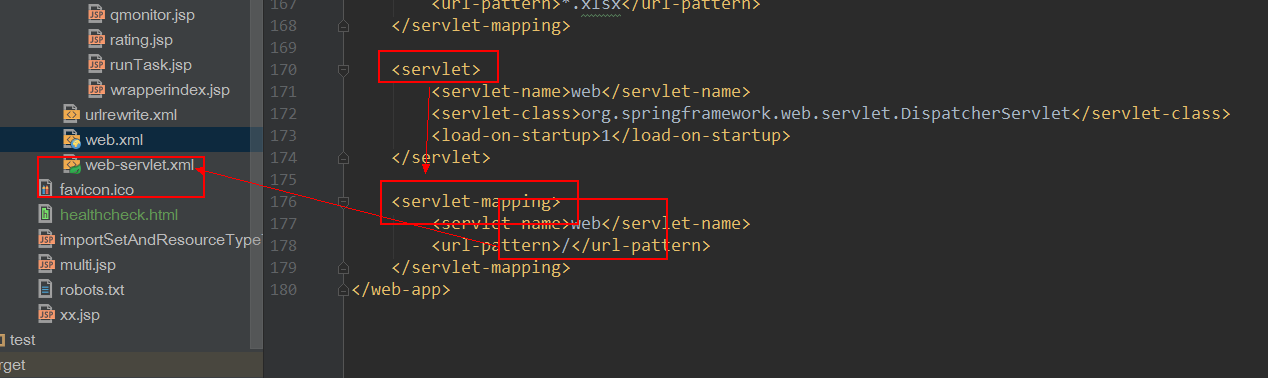
Spring框架支持Servlet容器，并扩展了该容器

标准的Servlet



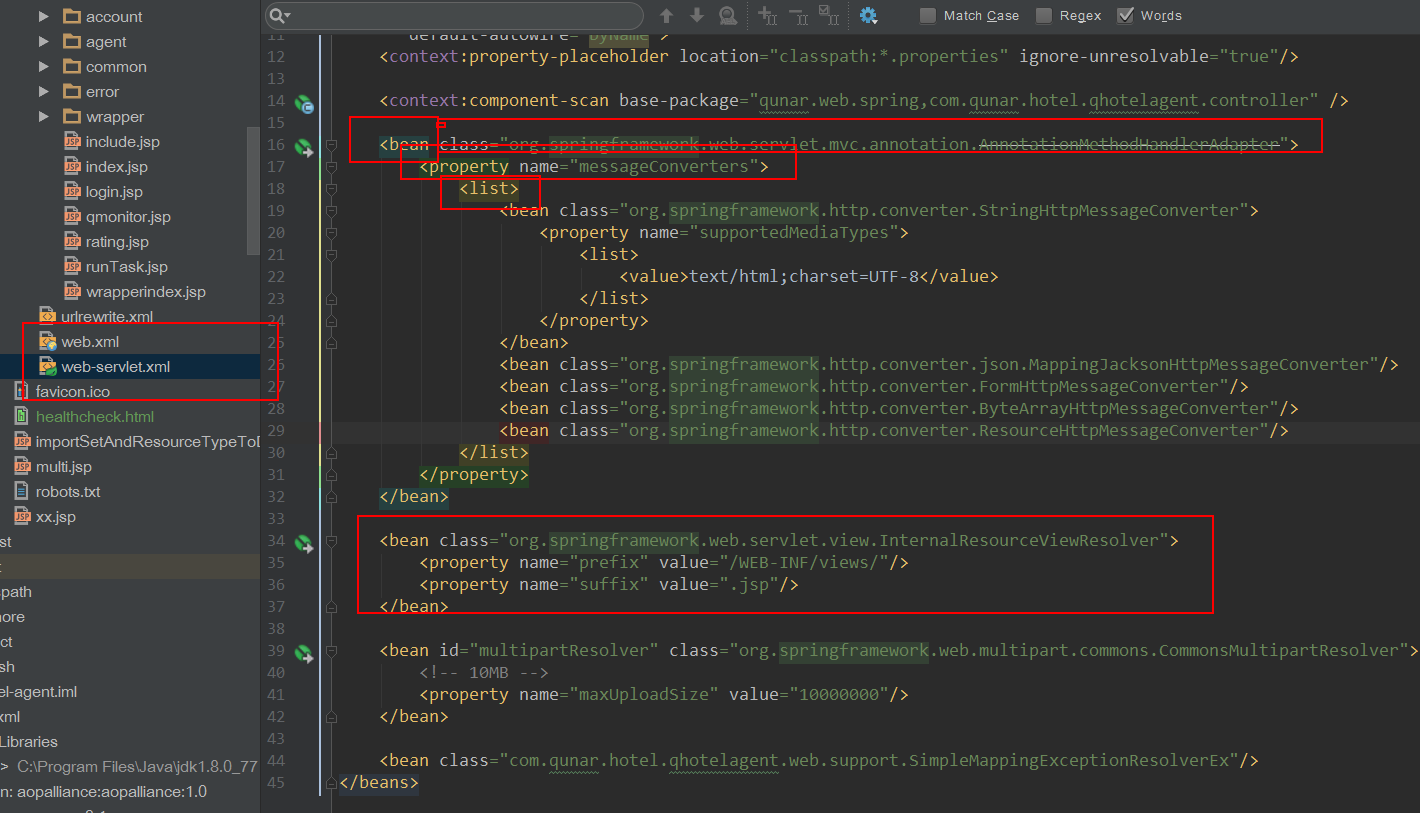
Spring中担任的作用：

接收和转发web请求到内部框架的处理单元



Servlet和servlet-mapping以及 web.xml文件之间的关系

<context-param>-> <listener> -> 监听 <filter> -> 拦截，过滤 <servlet> 这些mapping只是找到了起路径，还要到对应的xml文件中初始化这些bean的对象



<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">  
 <property name="messageConverters"> //某某变量  
 <list> //类型是一个list数组类型，下面是多种元素实现，相当于多态  
 <bean class="org.springframework.http.converter.StringHttpMessageConverter">  
 <property name="supportedMediaTypes">  
 <list>  
 <value>text/html;charset=UTF-8</value>  
 </list>  
 </property>  
 </bean> //这些类都是某种接口的实现，多态  
 <bean class="org.springframework.http.converter.json.MappingConverter"/>  
 <bean class="org.springframework.http.converter.FormHttpMessageConverter"/>  
 <bean class="org.springframework.http.converter.ByteArrayHttpMessageConverter"/>  
 <bean class="org.springframework.http.converter.ResourceHttpMessageConverter"/>  
 </list>  
 </property>  
</bean>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/views/"/> //这个就是这个类初始化，得到的对象里面  
 <property name="suffix" value=".jsp"/> //只有这两个变量有初值  
</bean> //但是你要是new变量的话，就不会有这两个初值，但是是不提倡的，不然要这个做什么

2016/12/12项目完成遇到的问题

1. **Controller的简单介绍**

Controller处理有DispatcherServlet分发的请求，将数据封装成model，将model返回给view展示

**Spring MVC 常用注解**

**@Controller 注解**

该注解就是可以标记是一个类，无需继承特定的类或者特定的接口，当Spring启动，Spring就会自动扫描所有的包（可以设置）

但是没有这个注解，就需要继承CommanController类或者其他很多的类

@Controller 只是定义了一个控制器类，而使用@RequestMapping 注解的方法才是真正处理请求的处理器

（否则只是一个controller，但是不能处理请求）

**@RequestMapping注解**

@RequestMapping(value = "/save", method = RequestMethod.*POST*)

映射URL到控制器类，或者到controller里面的action上

当控制器里面没有@RequestMapping，则action上的@RequestMapping就是绝对路径

**@PathVariable注解 @RequestParam注解**

一个URL/test/hello/showView/2.do?name=hello&age=174

@PathVariable 获取的是？前面的参数，这个是在 @RequestMapping(“/”)后面的路径中定义参数的

@RequestParam获取的是？后面的参数，这个是在前端JSP文件传过来的或者表单中等等

@RequestParam绑定 HttpServletRequest 请求参数到控制器方法参数

@Controller

@RequestMapping ( "/test/{variable1}" )

public class MyController {

@RequestMapping ( "/showView/{variable2}" )

public ModelAndView showView( @PathVariable String variable1, @PathVariable ( "variable2" ) int variable2，@RequestParam(required=false) String name, @RequestParam ( "age" ) int age ) {

//指定名字，默认名字。是否必传参数，以及类型

ModelAndView modelAndView = new ModelAndView();

modelAndView.setViewName( "viewName" );

modelAndView.addObject( " 需要放到 model 中的属性名称 " , " 对应的属性值，它是一个对象 " );

return modelAndView;

}

}

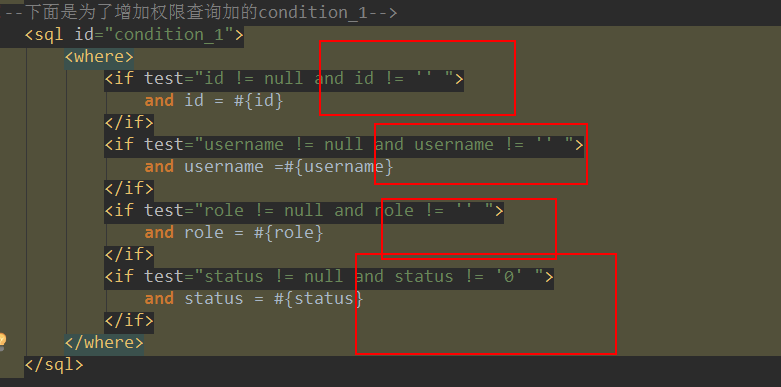
@RequestMapping 中还支持通配符“\* ”

和@PathVariable 一样，当你没有明确指定从request 中取哪个参数时，Spring 在代码是debug 编译的情况下会默认取更方法参数同名的参数，如果不是debug 编译的就会报错。此外，当需要从request 中绑定的参数和方法的参数名不相同的时候，也需要在@RequestParam 中明确指出是要绑定哪个参数。

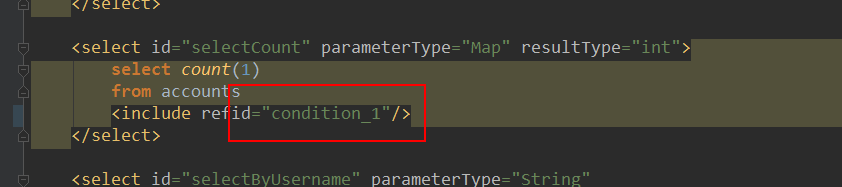
**@CookieValue注解，@RequestHeader注解**和上面两个注解类似

<http://www.cnblogs.com/xiepeixing/p/4243288.html>

1. 

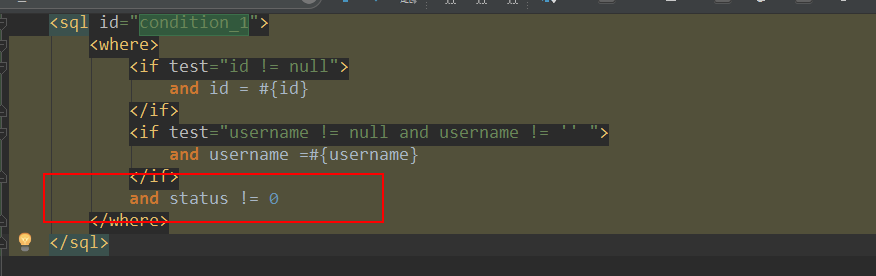


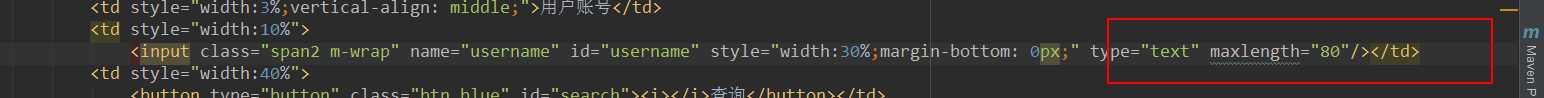
这个原因就是当传进去一个空字符串，这个时候就是



Sql里面where一个条件就是username=””,就查不出来

1. 这样可以在xml里面加上where条件



1. 

<input class="span2 m-wrap" name="username" id="username" style="width:30%;margin-bottom: 0px;" type="text" maxlength="80"/></td>

这里的maxlength指的是输入框内字符最大数目，多了只会保留前面的字符

1. **JavaScript（js,是JQuery）简单函数**
2. Trim()去除字符串左右两端的空格
3. .find(selector) 获得当前元素集合中每个元素的后代，通过选择器、jQuery 对象或元素来筛选。
4. $(selector).html(content) 返回或设置被选元素的内容 (inner HTML)。如果该方法未设置参数，则返回被选元素的当前内容。
5. $(selector).remove()从DOM中删除所有匹配的元素。不会把匹配的元素从jQuery对象中删除，因而可以在将来再使用这些匹配的元素。但除了这个元素本身得以保留之外，其他的比如绑定的事件，附加的数据等都会被移除。
6. $(selector).change()当元素的值发生改变时，会发生 change 事件。
7. 该事件仅适用于文本域（text field），以及 textarea 和 select 元素。当用于 select 元素时，change 事件会在选择某个选项时发生。当用于 text field 或 text area 时，该事件会在元素失去焦点时发生。
8. Location.reload() 页面重新刷新
9. attr(name|properties|key, value|fn) 设置或返回被选元素的属性值。
10. blur([[data],fn]) 当元素失去焦点时触发 blur 事件。

这个函数会调用执行绑定到blur事件的所有函数，包括浏览器的默认行为。可以通过返回false来防止触发浏览器的默认行为。blur事件会在元素失去焦点的时候触发，既可以是鼠标行为，也可以是按tab键离开的

1. validate() 表单提供了强大的验证功能<http://www.runoob.com/jquery/jquery-plugin-validate.html>
2. 首先改api里面的东西->改pom.xml里面的version版本->对应的pmo新建分支提交到线上->给测试分支，让测试法beta版的jar包->IDEA工具里面更新maven jar包
3. 改一个表的字段属性的时候，table这个词要带上
4. ALTER **TABLE** qhagent.accounts MODIFY role VARCHAR(20) NOT NULL DEFAULT 'USER' COMMENT '角色权限' ;
5. **Spring MVC 自动为对象注入枚举类型**
6. 这是新建一个java文件
7. package com.qunar.hotel.qhotelagent.convert;  
   import org.springframework.context.annotation.Role;  
   import org.springframework.core.convert.converter.Converter; //shift+enter就可以导入包  
   import org.springframework.core.convert.converter.ConverterFactory;@SuppressWarnings("unchecked")  
   final class StringToEnumConverterFactory implements ConverterFactory<String, Enum> {  
    public <Role extends Enum> Converter<String, Role> getConverter(  
    Class<Role> Role) { //返回一个<Role extends Enum>类的对象  
    return new StringToEnum(Role);  
    }  
    private class StringToEnum<Role extends Enum> implements Converter<String, Enum> {  
    private final Class<Role> enumType;  
    public StringToEnum(Class<Role> enumType) {  
    this.enumType = enumType;  
    }  
    public Role convert(String source) {  
    if (source.length() == 0) {  
    return null;  
    }  
    return (Role) Enum.*valueOf*(this.enumType, source.trim());  
    }  
    }  
   }

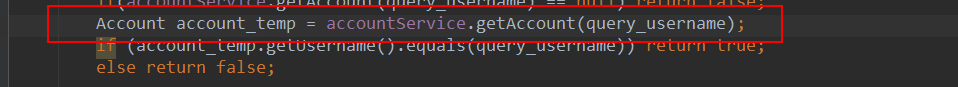
//还要在配置文件中定义这个bean

1. <mvc:annotation-driven conversion-service="conversionService"/>  
   <bean id="conversionService"  
    class="org.springframework.context.support.ConversionServiceFactoryBean">  
    <property name="converters">  
    <set>  
    <bean class="com.qunar.hotel.qhotelagent.convert.StringToEnumConverterFactory" /> //注意这里的路径就可以了，是上一个文件的路径  
    </set>  
    </property>  
   </bean>
2. <http://blog.csdn.net/china_bobo/article/details/43016415>
3. <http://blog.csdn.net/ljhabc1982/article/details/18703459>
4. <http://www.tuicool.com/articles/zQV3Af>
5. <http://blog.csdn.net/renhui999/article/details/9837897>

记得加上   
xmlns:mvc=<http://www.springframework.org/schema/mvc>

http://www.springframework.org/schema/mvc/spring-mvc.xsd

1. Idea的基本使用
2. ALT + 回车 导入包，自动修正
3. Ctrl + N 查找类
4. Ctrl + Shift + N 查找文件
5. Ctrl + Alt + L 格式化代码
6. Alt + Insert 生成代码，如getter，setter方法，构造函数等
7. Ctrl + E 最近修改的代码
8. Alt + Shift + C 对比最近修改的代码
9. 在js中打印表单中元素
10. alert(document.getElementById("txt").value) 红色的是元素的id名字
11. 修改Java文件不用重启Tomcat服务----热部署、热加载
12. <http://blog.csdn.net/feilongjian001/article/details/8576392>

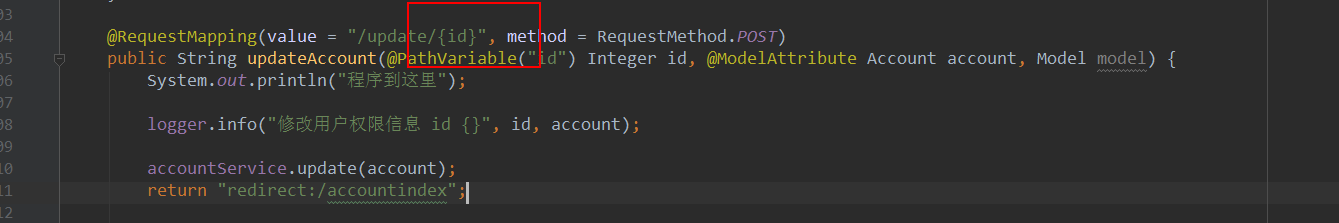


Account account\_temp = accountService.getAccount(query\_username);  
if (account\_temp.getUsername().equals(query\_username)) return true;

1. 如果给对象赋值null，那么在下面使用对象的时候，就会抛出异常，注意！
2. <textarea> 标签的 readonly 属性 <textarea readonly=”readonly”>规定文本区为只读
3. Js中遍历输入一个对象中的所有成员变量

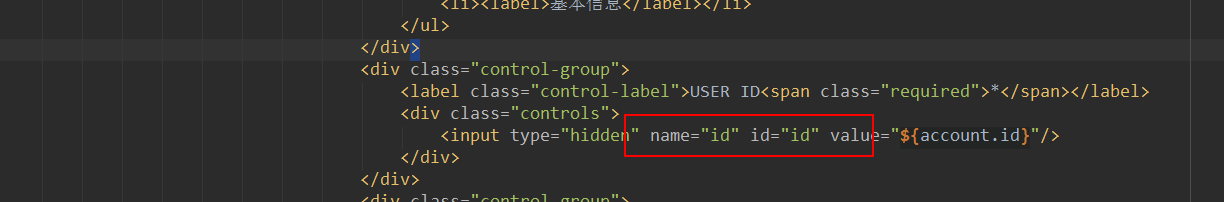
**for** (**var** sProp **in** defaultDatas) { // defaultDatas是对象  
 alert(sProp);  
}

1. 注意这个id不是凭空获得的



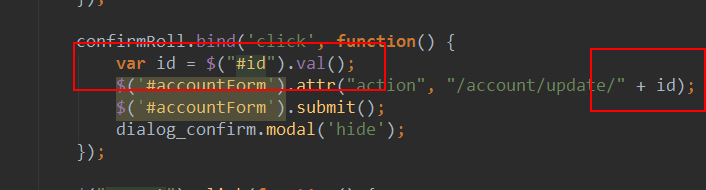
@RequestMapping(value = "/update/{id}", method = RequestMethod.*POST*)

是从上一次通过id请求获得的对象account里面放在jsp文件里面



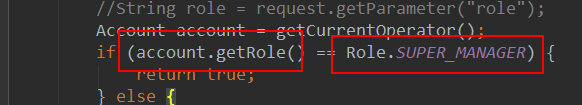
<div class="control-group">  
 <label class="control-label">USER ID<span class="required">\*</span></label>  
 <div class="controls">  
 <input type="hidden" name="id" id="id" value="**${**account.id**}**"/>  
 </div>  
</div>

再将其取出来，通过js文件请求



confirmRoll.bind('click', **function**() {  
 **var** id = $("#id").val();  
 $('#accountForm').attr("action", "/account/update/" + id);  
 $('#accountForm').submit();  
 dialog\_confirm.modal('hide');  
});

1. Web.xml是tomcat里面配置文件，spring文件夹下里面有bean的配置文件才是spring的配置文件
2. 枚举类中比较大小或者相同



Account account = getCurrentOperator(); Role是一个枚举类  
if (account.getRole() == Role.*SUPER\_MANAGER*) { getRole()返回的是一个枚举类的对象  
 return true;

1. Js 页面返回上一步<http://blog.csdn.net/fgdfgasd/article/details/6960892>
2. js前端总结<http://lib.csdn.net/u010323023/structure/JavaScript>
3. **Spring Interceptor spring配置文件+Interceptor类**

<http://www.jb51.net/article/97641.htm>

<http://elim.iteye.com/blog/1750680>

Spring mvc中的 Interceptor 类似于 servlet 中的filter。

<http://www.jb51.net/article/68075.htm>

主要作用是拦截用户的请求并进行相应的处理，

**Interceptor类**

1. import javax.servlet.http.HttpServletRequest;  
    import javax.servlet.http.HttpServletResponse;  
    import org.springframework.web.servlet.HandlerInterceptor;  
    import org.springframework.web.servlet.ModelAndView;  
     
   public class SpringMVCInterceptor implements HandlerInterceptor //注意这里继承了这个接口  
    */\*\*  
    \* preHandle方法是进行处理器拦截用的，顾名思义，该方法将在Controller处理之前进行调用，SpringMVC中的Interceptor拦截器是链式的，可以同时存在  
    \* 多个Interceptor，然后SpringMVC会根据声明的前后顺序一个接一个的执行，而且所有的Interceptor中的preHandle方法都会在  
    \* Controller方法调用之前调用。SpringMVC的这种Interceptor链式结构也是可以进行中断的，这种中断方式是令preHandle的返  
    \* 回值为false，当preHandle的返回值为false的时候整个请求就结束了。  
    \*/* @Override  
    public boolean preHandle(HttpServletRequest request,  
    HttpServletResponse response, Object handler) throws Exception {  
    // *TODO Auto-generated method stub* return false;  
    }  
    */\*\*  
    \* 前提：当前这个Interceptor的preHandle方法返回值为true的时候才会执行。  
    \* 执行时间是在处理器进行处理之后，也就是在Controller的方法调用之后，在DispatcherServlet进行视图的渲染之前执行，  
    在这个方法中你可以对ModelAndView进行操作。  
    这个方法的链式结构跟正常访问的方向是相反的，也就是说先声明的Interceptor拦截器该方法反而会后调用，这跟Struts2里面的拦截器的执行过程有点像，（与prehandle()前后对应）  
    \* 只是Struts2里面的intercept方法中要手动的调用ActionInvocation的invoke方法，Struts2中调用ActionInvocation的invoke方法就是调用下一个Interceptor  
    \* 或者是调用action，然后要在Interceptor之前调用的内容都写在调用invoke之前，要在Interceptor之后调用的内容都写在调用invoke方法之后。  
    \*/* @Override  
    public void postHandle(HttpServletRequest request,  
    HttpServletResponse response, Object handler,  
    ModelAndView modelAndView) throws Exception {  
    // *TODO Auto-generated method stub* }  
     
    */\*\*  
    \* 前提：当前对应的Interceptor的preHandle方法的返回值为true时才会执行。该方法将在整个请求完成之后，也就是DispatcherServlet渲染了视图执行，  
    \* 这个方法的主要作用是用于清理资源的，当然这个方法也只能在当前这个Interceptor的preHandle方法的返回值为true时才会执行。  
    \*/* @Override  
    public void afterCompletion(HttpServletRequest request,  
    HttpServletResponse response, Object handler, Exception ex)  
    throws Exception {  
    // *TODO Auto-generated method stub* }  
   }

**Spring配置文件中的配置 xml文件**

加入Spring Schema以及

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd   
 http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd   
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd   
 http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

<!-- 在实际开发中通常都需配置 mvc:annotation-driven标签,这个标签是开启注解 -->  
<mvc:annotation-driven></mvc:annotation-driven>  
<!-- 拦截器 -->  
<mvc:interceptors>  
 <!-- 多个拦截器，顺序执行 -->  
 <mvc:interceptor>  
 <mvc:mapping path="/\*\*"/>  
 <bean class="com.mvc.interceptor.LoginInterceptor"></bean>  
 </mvc:interceptor>  
 <mvc:interceptor>  
 <mvc:mapping path="/asedf/\*"/>  
 <bean class="com.mvc.interceptor.LoginInterceptor"></bean>  
 </mvc:interceptor>  
 <mvc:interceptor>  
 <mvc:mapping path="/sa/\* "/>  
 <bean class="com.mvc.interceptor.LoginInterceptor"></bean>  
 </mvc:interceptor>  
</mvc:interceptors>

举例

这个是在prehandle() 方法里面用，Request.getParameter() 就可以获得前端传来的参数

String role = request.getParameter("role")

这可以通过request获取前端传来的参数，

request.getRequestDispatcher("/WEB-INF/error/403.jsp").forward(request, response);

这个可以重定向，跳转页面，这个request对象是前端传过来的，里面的参数保持不变

Web是请求/响应架构的使用，而request和response就是在服务器端生成的相应的两个对象，request能够获取客户端传递的参数及相关的一些信息，而response就是给客户端响应相关的页面及信息。

request.getRequestDispatcher().forward(request.response)是转发

将提交这个请求的页面的信息带过去传给被转发页面；

request 和response是servlet内置的8个对象之二

当你要跳转页面但是你又要用到前一个页面的某些信息的时候可以用这个方法

当你要跳转页面并且不需要用到前一个页面的信息时你可以选择用redirect（重定向）

1. **servlet中请求转发（forword）与重定向（sendredirect）的区别**

<http://www.cnblogs.com/CodeGuy/archive/2012/02/13/2349970.html>

1. **request.getParameter() 和request.getAttribute() 区别**

<http://www.cnblogs.com/wengfumin/archive/2011/12/22/2297494.html>

request.getParameter() 方法传递的数据，会从Web客户端传到Web服务器端，代表HTTP请求数据。request.getParameter()方法返回String类型的数据。

**request.setAttribute()**和 **getAttribute()**方法传递的数据只会存在于Web容器内部

1. **Spring拦截器中通过request获取到该请求对应Controller中的method对象**

<http://chenzhou123520.iteye.com/blog/1702563>

1. **使用Spring Interceptor实现URL访问校验**

<https://my.oschina.net/vzr/blog/760746>

1. **Servlet的过滤器**

<http://www.cnblogs.com/wangyuyu/archive/2013/07/02/3167354.html>

**拦截器与过滤器的区别** ：

拦截器是基于java的反射机制的，而过滤器是基于函数回调。

拦截器不依赖与servlet容器，过滤器依赖与servlet容器。

拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。

拦截器可以访问action上下文、值栈里的对象，而过滤器不能访问。

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

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

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

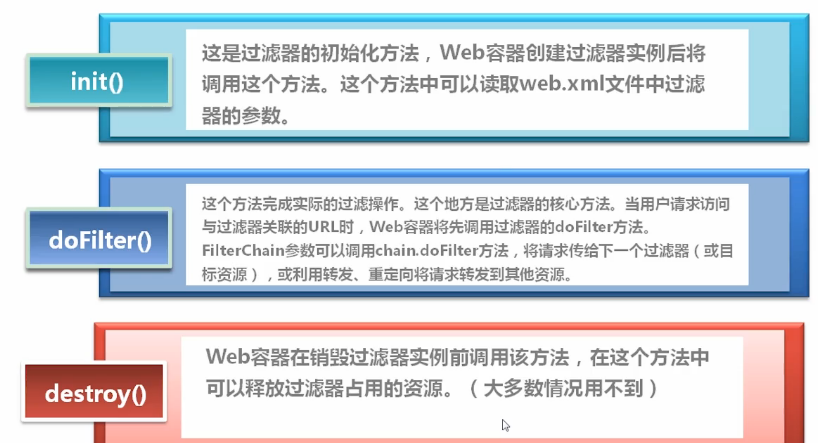
**拦截器，**是在面向切面编程的就是在你的service或者一个方法，前调用一个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，在你调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出异常的时候做业务逻辑的操作。

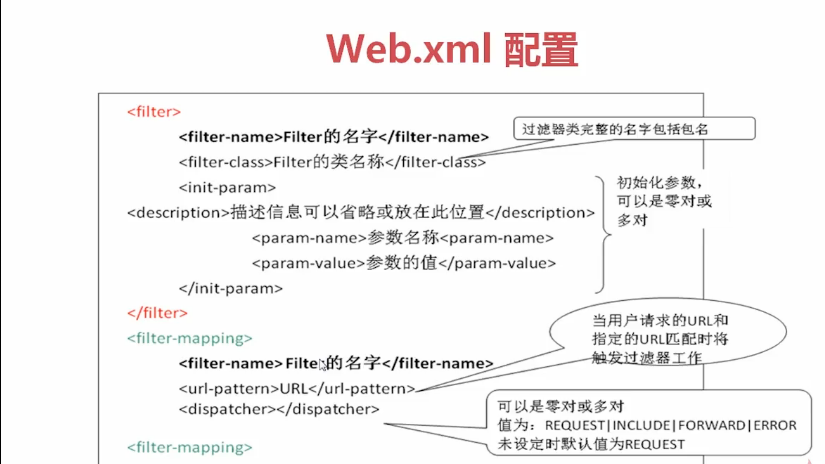
1. **过滤器**<http://www.cnblogs.com/coderland/p/5902878.html>

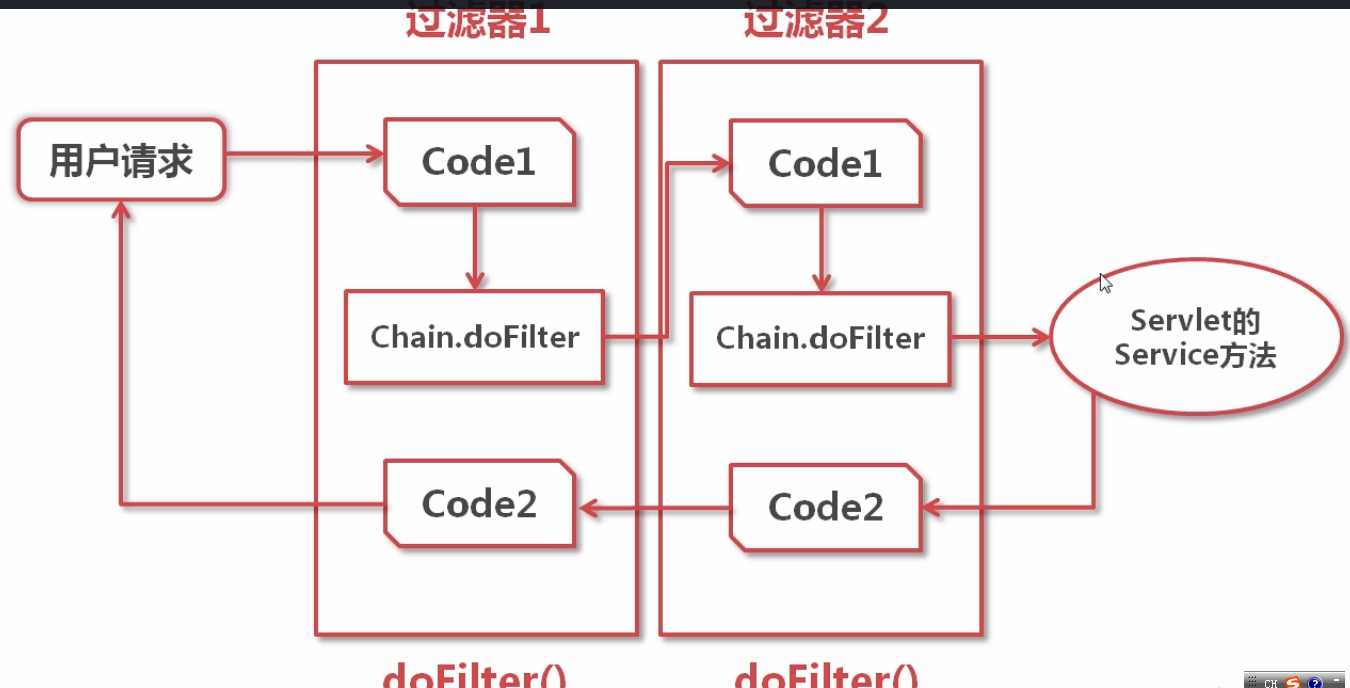
<http://blog.csdn.net/sd0902/article/details/8395641>

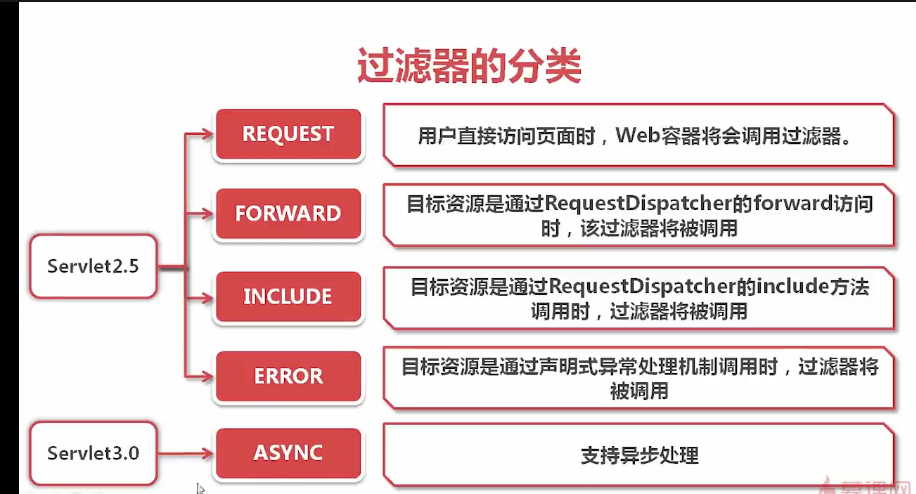
过滤器并不会处理请求

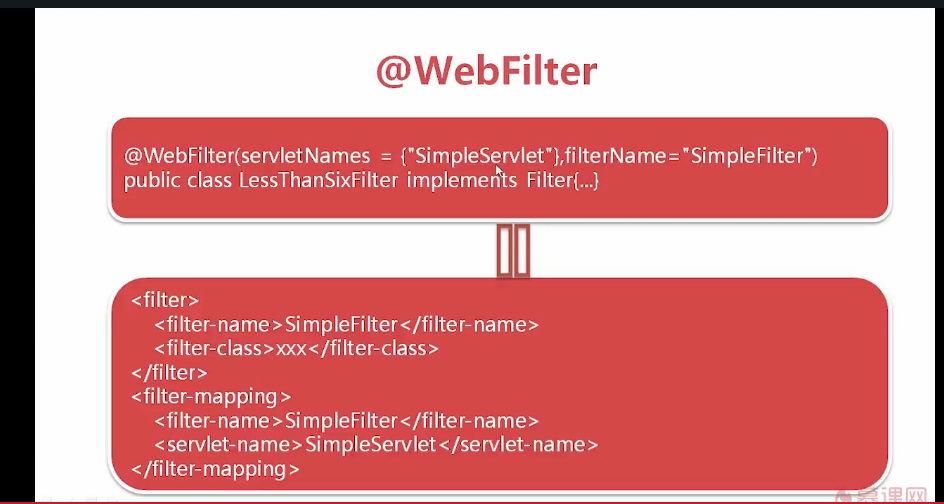






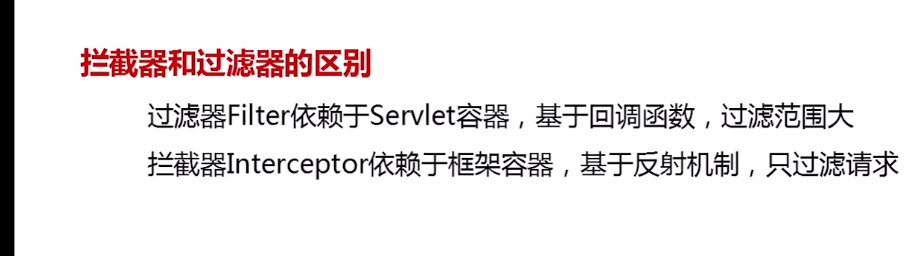






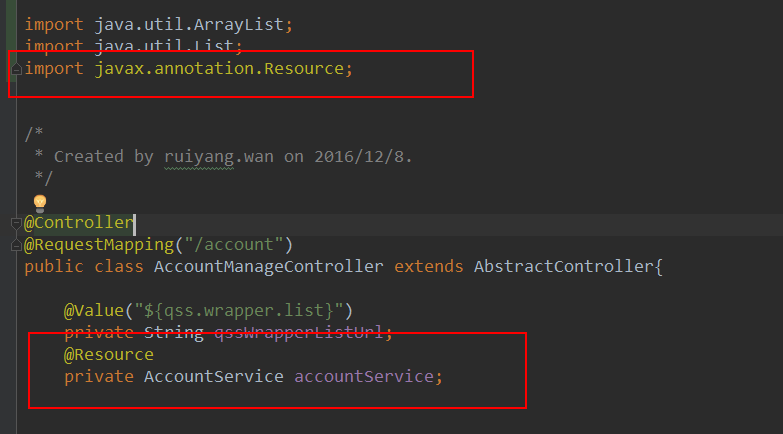
Web3.0 可以通过注解，直接在方法上面加上注解直接搞定，不用再web.xml文件里面配置





过滤器可以改变用户的请求路径，但是不能直接返回数据（处理请求）

1. 判断两个字符串是否相等 A.equals(B) ，不能用 == ，由于字符串是对象类型，，==比较的是两个对象的内存地址，而equals() 比较的是对象的内容，这里是区分大小写的，在枚举类中要注意
2. 注意继承类是在继承接口前面 先extends才能implements，因为接口的范围要比类的范围要大
3. Beta数据库以后修改的时候需要之前给备份一下，
4. **注解 @Resource** 代码注入的意思，不需要任何配置文件的配置



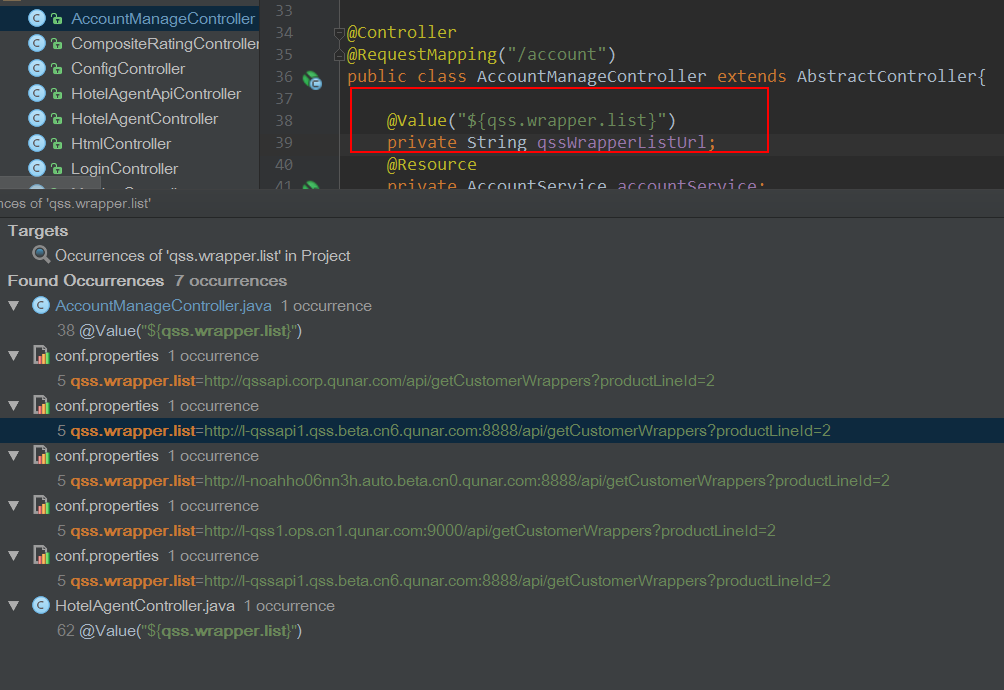
这里用@resource就不用再xml文件里面配置bean了

注意@Resource是java的api，不是spring的api，但是需要结合spring来去做

既没有@Resource也没有<bean>去控制反转，这个时候就需要采用new一个对象来用

1. **注解 @Value** Spring 通过注解获取\*.porperties文件的内容，

除了xml配置外，还可以通过@value方式来获取。



1. 下面是各种properties里面的配置，各种环境下的配置

<http://www.cnblogs.com/hafiz/p/5876243.html> 有五种方式读取properties文件内容

<context:property-placeholder location="classpath:\*.properties" ignore-unresolvable="true"/>

这个一般是放在resource路径下面的某个xml文件里面

1. **XML Schema 中 import 和 include 的区别**

import：只能导入不同命名空间的XSD

include：只能导入相同命名空间的XSD，或被导入的XSD未声明命名空间

1. **org.slf4j.Logger 日志打印**

获取当前类的logger对象 指定类初始化日志对象，只要将getLogger() 里面的参数换成其他类即可

protected final Logger logger = LoggerFactory.getLogger(this.getClass());

注意这里要导入两个类，这都是spring jar包里面自带的

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;

This.getClass() 获取当前运行实力的类

然后logger里面方法的类型有

Logger.error() 这个里面也可以把异常加进去

catch (Exception e) {  
 *logger*.error("解析代理商历史名称失败", e);

logger.info() 这是级别最低的日志打印，没有比这级别更低了，里面可以插入参数，参数各种各样类型

logger.info("[info message]{},{},{},{}", "abc", **false**, 123,**new** Slf4jTest());

logger.warn()

logger.debug()

logger.debug()

logger.isDebugEnabled()

logger.trace()

<http://hanhongke123.blog.163.com/blog/static/62223494201241631644433/>

<http://www.tuicool.com/articles/IfeUfq>

<http://lavasoft.blog.51cto.com/62575/184492/>

<http://www.cjsdn.net/Doc/JDK50/java/util/logging/class-use/Level.html>

logger.info("[info message]{},{},{},{}", "abc", false, 123,new Slf4jTest());  
java日志系统

<http://www.tuicool.com/articles/IfeUfq>

<http://wiki.corp.qunar.com/pages/viewpage.action?pageId=146883101>

<https://www.oschina.net/translate/logging-in-java-switching-to-logback> Java 日志：迁移到 Logback 和 SLF4J部门术语，专业名词

<http://mushiqianmeng.blog.51cto.com/3970029/720415> slf4j+logback配置方式，logback.groovy使用备忘

<http://blog.csdn.net/chszs/article/details/8653460> SLF4J和Logback日志框架详解

<http://blog.csdn.net/longaohun/article/details/6782046> [slf4j+logback使用](http://blog.csdn.net/longaohun/article/details/6782046)

<http://zy19982004.iteye.com/blog/1993388>

<http://blog.csdn.net/zhuyucheng123/article/details/38168291>

<http://www.importnew.com/21486.html>

<https://my.oschina.net/pingpangkuangmo/blog/410224>

<https://my.oschina.net/looly/blog/298675> logback.xml文件详细说明

<http://www.cnblogs.com/warking/p/5710303.html> logback配置的详细说明

Qhotel : 酒店的基础信息管理系统，包括酒店Info，城市，POI，商圈，品牌，城市等信息的维护功能和酒店Tree聚合，城市Tree聚合等功能。、

RT (Room Type) : 房型管理系统，维护酒店的房型相关的信息。

QImage: 酒店图片管理系统，维护酒店的图片信息。

QHotelSearch 提供酒店信息检索的功能，根据query查询出符合要求的酒店的HotelSeq列表和命中情况，hslist会根据这个结果补充上酒店的报价，基础信息等构成最终的结果。

CE 酒店抓取调度系统。根据配置的时间和参数对代理商酒店进行抓取，并推送到qhotel

DC酒店数据中心，该系统已经被Qhotel替换了，现在还能在Qhotel的一些功能中发现它的一些影子，比如DC Info预警

Hslist酒店搜索系统，提供面向前端的搜索，推荐等功能。

Hprice酒店报价服务。

Travelco提供报价酒店信息的抓取和解析功能

Qta酒店交易系统

otatts （OTA total solution）代理商交易系统

直销tts/hms 老的酒店直销系统，现已被qta替换

Hotel Info Qhotel系统中收录的酒店基础信息。对于现实中存在的一家酒店，Qhotel会建立一条对应的Info记录，记录酒店相关的信息，包括酒店的名称、地址、电话和设施等各种信息。每家酒店会有一个HotelSeq，用来唯一标识这家酒店。

CityCode 非地级市的citycode，用来给酒店打标签而得名。

HotelSeq 酒店唯一标识，由CityCode加上一个在城市内唯一的数字构成。HotelSeq在建立酒店Info信息的时候生成。

注：CityCode和HotelSeq的缺点：因为CityCode是有含义的，造成变更CityCode和HotelSeq非常困难，给城市的维护带来很大的麻烦，不如直接用数字ID来编码

hotel tree 这是从酒店 wrapper 的角度来看的。对于酒店 wrapper，在爬取数据的时候，每个酒店会对应多个供应商，那么为了表现这个酒店和供应商的关系，就用一个树状结构来表示这层相互关系。然后，wrapper就可以通过这样的tree，逐个酒店获取不同的供应商，进儿获取爬取所需要的信息，抓到相关的报价等。

酒店Merge 当一家酒店建了两条Info信息的时候，用户搜索的时候会看到两家一模一样的酒店，影响用户体验，所以需要合并为一条记录。酒店Merge完成的功能就是把其中一家酒店A下关联的Tree & 图片 & 点评移到另外一家酒店B下，并把A酒店置为删除状态，然后把合并信息告知提供外网搜索详情页服务的程序（hslist），hslist会根据合并信息做跳转，当访问A酒店详情页URL的时候，会跳转到B酒店的详情页。

物理房型 Qunar定义的标准房型

产品房型 代理商的房型

房型聚合 把代理商的房型关联到Qunar定义的物理房型上的过程

酒店头图 自动或者人工挑选出来显示在列表页上的酒店代表图

预警 通过抓取的其他代理商的数据，根据一定规则判断数据是否变更，提醒人工进行修改。

Wrapper 为了抓取某个网站信息编写的一个包装器，目的是把非结构化的信息转化为结构化信息，完成定向抓取和抽取的功能。

Siteparam 代理商酒店详情页的唯一标识，一般采用代理商详情页URL中的一个唯一标识构成。

Bnb Bed aNd Breakfast 的缩写，意思是提供床位和早餐的酒店类型，泛指民宿，包含客栈、家庭旅馆、日租房等类型。

Hotel Express 酒店直通车。酒店在qunar平台的直销产品（含PPC/PPB）。

Hotel Seq Hotel Sequence：酒店序列号, 酒店在外网的编号，由城市全拼和数字组成,

1. **<mvc:annotation-driven/>** [**https://my.oschina.net/HeliosFly/blog/205343**](https://my.oschina.net/HeliosFly/blog/205343)

会**自动注册DefaultAnnotationHandlerMapping 与AnnotationMethodHandlerAdapter** 这两个bean,所以就没有机会再给它注入**interceptors**属性，就**无法指定拦截器**。

当然我们可以通过人工配置上面的两个Bean，不使用 <mvc:annotation-driven />，就可以 给interceptors属性 注入拦截器了。

1. **@requestBody @responseBody注解配置要点**

<http://www.cnblogs.com/qq78292959/p/3760651.html>

@RequestBody

作用：

i) 该注解用于读取Request请求的body部分数据，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的数据绑定到要返回的对象上；

ii) 再把HttpMessageConverter返回的对象数据绑定到 controller中方法的参数上。

@ResponseBody

作用：

该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

使用时机：

返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；

两个注解都是Spring框架自带的，是Spring里面的jar包里面的

*org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter*

所以这个要记得引入这个jar包，还有加上相应的bean或者加上<mvc:annotation-driven/>

<https://my.oschina.net/shyloveliyi/blog/525309>

1. JS 返回上一步（退回上一步上一个网页）
2. <http://blog.csdn.net/fgdfgasd/article/details/6960892?_t_t_t=0.9243061118380878>
3. <http://www.cnblogs.com/wlqh/p/5948879.html>
4. Posthandle ajax请求 login的时候 怎么获取cookie
5. interceptor拦截所有路径/\*\*
6. 超链接通常用get请求
7. Login的时候出问题，也不一定是login
8. js模态框要学习
9. 给超链接添加 confirm确认框
10. 这个包里面的 **StringUtils.equals(A, B)**方法比较两个字符串是否相等

import org.apache.commons.lang3.StringUtils;

StringUtils.equals("", ""); //结果是true

StringUtils.equals(null, null); //结果是true

StringUtils.equals(null, ""); //结果是false

StringUtils.equals("",null); //结果是false

StringUtils.equals(null,""); //结果是false

StringUtils.equalsIgnoreCase("ss", "Ss"); //不区分大小写--结果是true

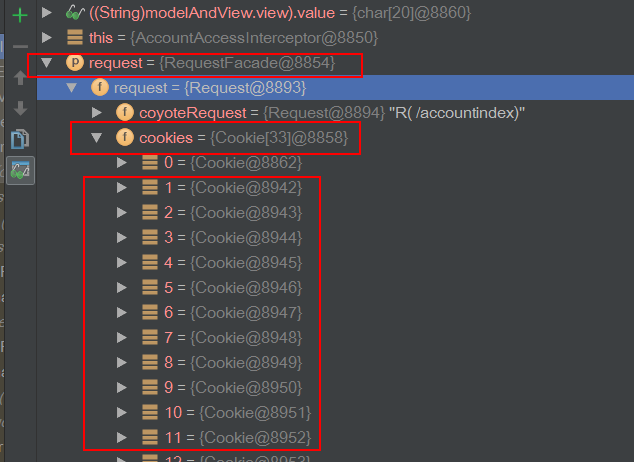
1. StringUtils 里面方法总结

<http://blog.csdn.net/ylchou/article/details/7479861>

1. 在拦截器Interceptor里面获取cookie

Cookie[] cookie = request.getCookies();  
for (int i = 0; i < cookie.length; i++) {  
 Cookie cook = cookie[i];  
 if(cook.getName().equalsIgnoreCase("qha\_un")){ //获取键  
 username = cook.getValue().toString(); //获取值  
 }  
}

1. 可以采用debug的方式，获取当前request的内容，采用Request.getCookies() 获取当前的cookie，
2. 注意login请求的那个页面是没有cookie的。
3. Cookie是一个数组，里面有很多的键值对，要debug调试一下，看需要获取什么键值对

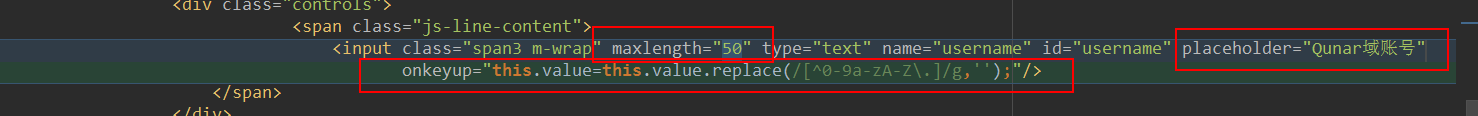


在postHandle()方法中，可以通过

modelAndView.getModel().put("judgeWhetherShow", judgeWhetherShow);

前端传参数

1. Js中的input可以在jsp页面里面实现正则表达式的参数验证



Onkeyup就是验证函数 maxlength表示输入框中最大字节 placeholder是默认提示

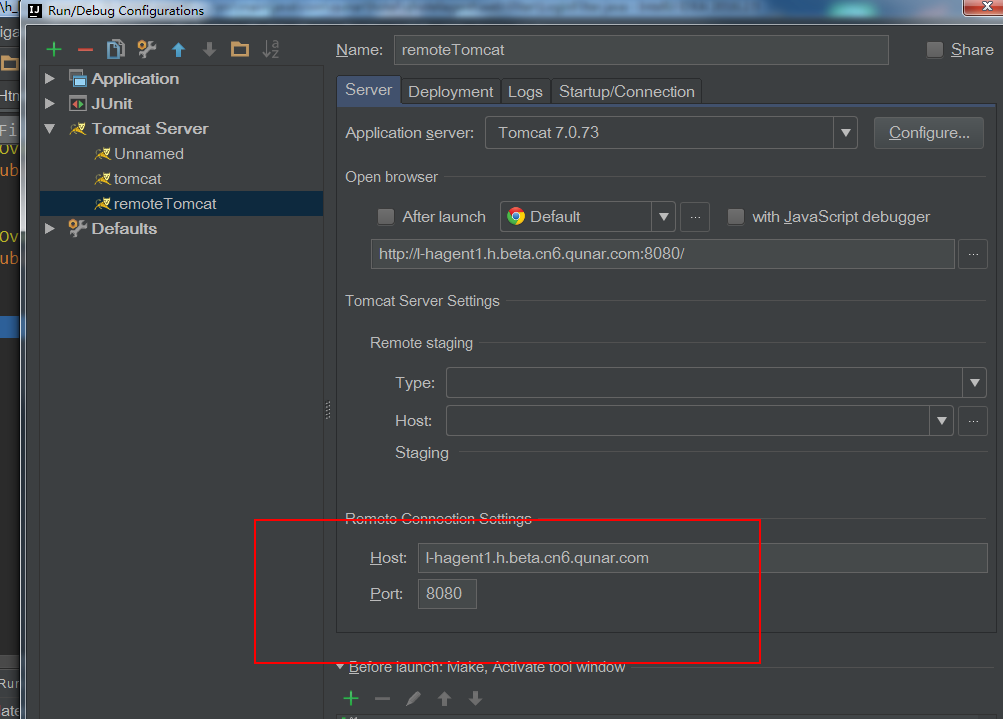
1. 在程序中尽量不要用sout，而是采用日志的方式输出
2. Confirm确认框可以在jsp页面中直接使用

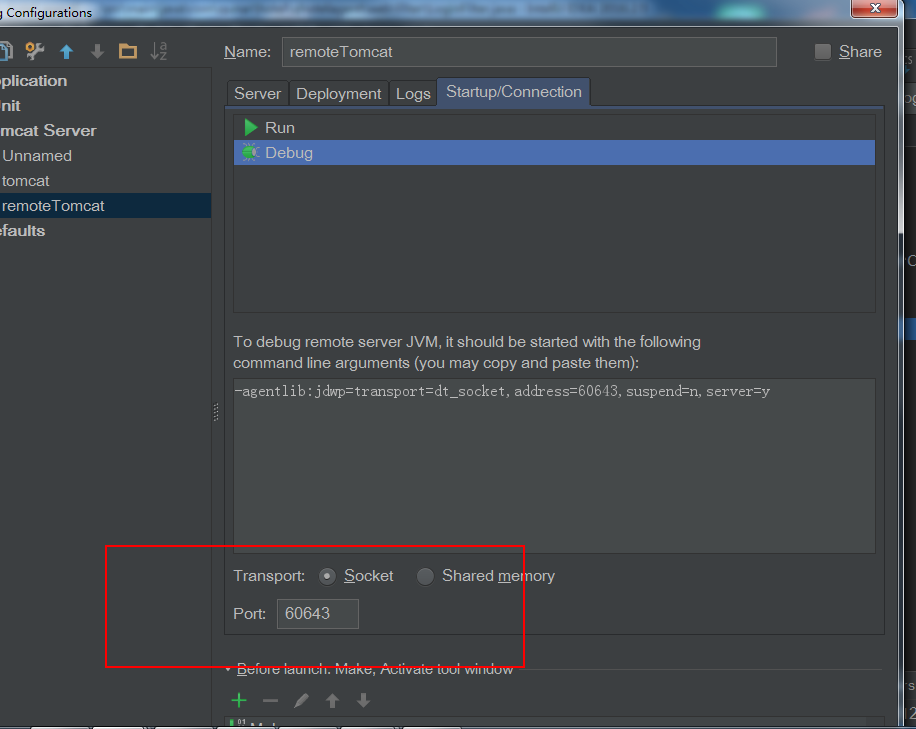
<http://www.jb51.net/article/77673.htm>

1. 注意jsp页面中 js的代码应该在<head> 头中加
2. 页面中做跳转
3. <script language="javascript">  
    alert("抱歉！没有当前功能的权限！");  
    window.onload=**function**()  
    {  
    setInterval("redirect();", 0); //跳转时间  
    }  
    **function** redirect()  
    {  
    window.location.href="/index"; //跳转路径  
    }  
   </script>
4. 前端要求写的字节长度有限制,那写到后端的数据库一定要注意一点!!防止后端数据库的长度设置不正确,
5. 打印日志错误，logger.info(“{}”, mou\_bian\_liang) 这里注意一定要有 {} ,这样才可以把日志打印出来
6. Ctrl+SHIFT+L IDEA里面代码格式化
7. JSP页面里面快速跳转，注意这是js代码，实在script中的，要在header里面加

setTimeout(window.location.href='/index',0);

1. 远程调试





然后debug就可以了，还可以登录这个跳板机到服务器，查看日志以及查看

socat TCP4-LISTEN:**9970**,fork,range=**192.168.110.221**/32 TCP4:127.0.0.1:**9903 这个要在服务器中运行**

9970是在 **192.168.110.221是本地的ip， 9903是在服务器里面 startenv.sh 文件中的端口号**

socat TCP4-LISTEN:<listen\_port>,fork,range=<source\_ip>/32 TCP4:127.0.0.1:<dest\_port>

注：listen\_port 是 socat 映射后的端口，dest\_port 是java远程调试服务监听的端口(对应第一步中的 port)，source\_ip 是运行 idea 或 eclipse 进行调试的机器的 ip 。

若机器上没有socat，联系机器负责人运行以下命令安装socat。

1. String []choosedelete = request.getParameterValues("choosedelete"); 如果获取这种数组的时候，一定要判断数组是否为空，一定要注意，这里的数组是从request里获取来得，如果为null，就经常报错

if(choosedelete==null || (choosedelete!=null&&choosedelete.length==0)) { } 要做一下判断

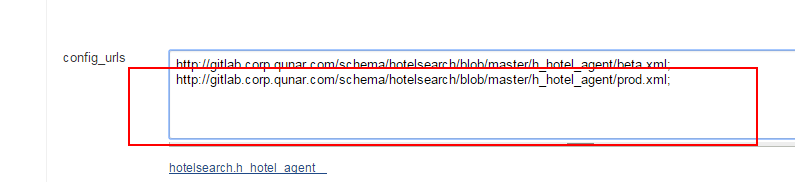
1. **上线的时候，一定要注意之前的数据库修改，一定要注意！！！！！！**！！！

怎么查看发布记录，进去qdr

搜索你的工程



进入parameters



接着访问这个路径就可以了

1. Jsp页面中引入jsp文件，js文件，css文件的方法

<%@**include** file="/WEB-INF/views/include.jsp"%>  
  
<link rel="stylesheet" href="http://qunarzz.com/hotel\_agent/prd/styles/common@**<%=**RsVersionUtil.*getVersion*("styles/common.css") **%>**.css"/>  
<link rel="stylesheet" href="http://qunarzz.com/hotel\_agent/prd/styles/save@**<%=**RsVersionUtil.*getVersion*("styles/save.css") **%>**.css"/>  
<link rel="stylesheet" href="/css/agentSavePop.css"/>  
<script type="text/javascript" src="/javascript/agentSavePop.js"></script>

1. 如果发现前端某个文件没有引入，可以用过F12，访问页面，查看network里面是否出现该文件
2. Hotel\_agent项目

线上路径 [http://l-hagent1.h.cn6.qunar.com:8080/](javascript:openUrl(%22http://l-hagent1.h.cn6.qunar.com:8080/%22))

Beta路径 <http://l-hagent1.h.beta.cn6.qunar.com:8080/wrapperindex>

本地路径 <http://l-hagent1.qunar.com:8090/agent/save>

1. 线上登录服务器之后，才可以登录数据库

mysql -uqhagent\_rw -p4b190d0a64c261fc -h192.168.224.186 -P3307 qhagent

1. –u 用户名 -p 密码 -h 数据库地址 -P 端口号 最后可以加一个数据库的名字
2. <http://wiki.corp.qunar.com/pages/viewpage.action?pageId=55509483> 去哪儿日常
3. <http://wiki.corp.qunar.com/pages/viewpage.action?pageId=55509483>
4. Varchar里面三个字节一个汉字
5. ALTER TABLE qhagent.hotel\_agents ADD sign\_subject VARCHAR(60) NOT NULL DEFAULT "" COMMENT "签约主题",

ADD default\_language VARCHAR(60) NOT NULL DEFAULT "" COMMENT "EB默认语言",

ADD notice\_language VARCHAR(60) NOT NULL DEFAULT "" COMMENT "通知语言",

ADD settlement\_unit VARCHAR(60) NOT NULL DEFAULT "" COMMENT "货币结算单位";

1. EL表达式的判断 <http://blog.csdn.net/kaishuaige/article/details/8504857>
2. JSP页面中调用函数 ${fn: } 内嵌函数

fn:escapeXml 把一些字符转成XML表示，例如 <字符应该转为&lt; ${fn:escapeXml(param:info)}

<http://blog.csdn.net/bbg_0622/article/details/5706014/>

1. Excel行去重 <http://www.orsoon.com/news/129486.html>

 public static void readTxtFile(String filePath, Set<Character> set, List<String> wrappers){  
        try {  
            String encoding="utf-8";  
            File file=new File(filePath);  
            if(file.isFile() && file.exists()){ //判断文件是否存在  
                InputStreamReader read = new InputStreamReader(  
                        new FileInputStream(file),encoding);//考虑到编码格式  
                BufferedReader bufferedReader = new BufferedReader(read);  
                String lineTxt = null;  
                while((lineTxt = bufferedReader.readLine()) != null){  
                    String content = lineTxt;  
                    int s = lineTxt.indexOf("roomType:");  
                    int e = lineTxt.indexOf("roomId");  
                    String roomType = lineTxt.substring(s + 9, e);  
//                    System.out.println(roomType);  
                    int cs = content.indexOf("cb=");  
                    int ce = content.indexOf("&cn=");  
                    String codebase = lineTxt.substring(cs+3, ce);  
                    test(roomType, roomType, codebase, wrappers, set);  
                }  
                read.close();  
                System.out.println("end");  
            }else{  
                System.out.println("找不到指定的文件");  
            }  
        } catch (Exception e) {  
            System.out.println("读取文件内容出错");  
            e.printStackTrace();  
        }  
    }

# [Java]读取文件方法大全](http://www.cnblogs.com/lovebread/archive/2009/11/23/1609122.html)

1、按字节读取文件内容  
2、按字符读取文件内容  
3、按行读取文件内容

4、随机读取文件内容

1. <http://www.cnblogs.com/lovebread/archive/2009/11/23/1609122.html>
2. public class ReadFromFile {  
       /\*\*  
        \* 以字节为单位读取文件，常用于读二进制文件，如图片、声音、影像等文件。  
        \*/  
       public static void readFileByBytes(String fileName) {  
           File file = new File(fileName);  
           InputStream in = null;  
           try {  
               System.out.println("以字节为单位读取文件内容，一次读一个字节：");  
               // 一次读一个字节  
               in = new FileInputStream(file);  
               int tempbyte;  
               while ((tempbyte = in.read()) != -1) {  
                   System.out.write(tempbyte);  
               }  
               in.close();  
           } catch (IOException e) {  
               e.printStackTrace();  
               return;  
           }  
           try {  
               System.out.println("以字节为单位读取文件内容，一次读多个字节：");  
               // 一次读多个字节  
               byte[] tempbytes = new byte[100];  
               int byteread = 0;  
               in = new FileInputStream(fileName);  
               ReadFromFile.showAvailableBytes(in);  
               // 读入多个字节到字节数组中，byteread为一次读入的字节数  
               while ((byteread = in.read(tempbytes)) != -1) {  
                   System.out.write(tempbytes, 0, byteread);  
               }  
           } catch (Exception e1) {  
               e1.printStackTrace();  
           } finally {  
               if (in != null) {  
                   try {  
                       in.close();  
                   } catch (IOException e1) {  
                   }  
               }  
           }  
       }  
     
       /\*\*  
        \* 以字符为单位读取文件，常用于读文本，数字等类型的文件  
        \*/  
       public static void readFileByChars(String fileName) {  
           File file = new File(fileName);  
           Reader reader = null;  
           try {  
               System.out.println("以字符为单位读取文件内容，一次读一个字节：");  
               // 一次读一个字符  
               reader = new InputStreamReader(new FileInputStream(file));  
               int tempchar;  
               while ((tempchar = reader.read()) != -1) {  
                   // 对于windows下，\r\n这两个字符在一起时，表示一个换行。  
                   // 但如果这两个字符分开显示时，会换两次行。  
                   // 因此，屏蔽掉\r，或者屏蔽\n。否则，将会多出很多空行。  
                   if (((char) tempchar) != '\r') {  
                       System.out.print((char) tempchar);  
                   }  
               }  
               reader.close();  
           } catch (Exception e) {  
               e.printStackTrace();  
           }  
           try {  
               System.out.println("以字符为单位读取文件内容，一次读多个字节：");  
               // 一次读多个字符  
               char[] tempchars = new char[30];  
               int charread = 0;  
               reader = new InputStreamReader(new FileInputStream(fileName));  
               // 读入多个字符到字符数组中，charread为一次读取字符数  
               while ((charread = reader.read(tempchars)) != -1) {  
                   // 同样屏蔽掉\r不显示  
                   if ((charread == tempchars.length)  
                           && (tempchars[tempchars.length - 1] != '\r')) {  
                       System.out.print(tempchars);  
                   } else {  
                       for (int i = 0; i < charread; i++) {  
                           if (tempchars[i] == '\r') {  
                               continue;  
                           } else {  
                               System.out.print(tempchars[i]);  
                           }  
                       }  
                   }  
               }  
     
           } catch (Exception e1) {  
               e1.printStackTrace();  
           } finally {  
               if (reader != null) {  
                   try {  
                       reader.close();  
                   } catch (IOException e1) {  
                   }  
               }  
           }  
       }  
     
       /\*\*  
        \* 以行为单位读取文件，常用于读面向行的格式化文件  
        \*/  
       public static void readFileByLines(String fileName) {  
           File file = new File(fileName);  
           BufferedReader reader = null;  
           try {  
               System.out.println("以行为单位读取文件内容，一次读一整行：");  
               reader = new BufferedReader(new FileReader(file));  
               String tempString = null;  
               int line = 1;  
               // 一次读入一行，直到读入null为文件结束  
               while ((tempString = reader.readLine()) != null) {  
                   // 显示行号  
                   System.out.println("line " + line + ": " + tempString);  
                   line++;  
               }  
               reader.close();  
           } catch (IOException e) {  
               e.printStackTrace();  
           } finally {  
               if (reader != null) {  
                   try {  
                       reader.close();  
                   } catch (IOException e1) {  
                   }  
               }  
           }  
       }  
     
       /\*\*  
        \* 随机读取文件内容  
        \*/  
       public static void readFileByRandomAccess(String fileName) {  
           RandomAccessFile randomFile = null;  
           try {  
               System.out.println("随机读取一段文件内容：");  
               // 打开一个随机访问文件流，按只读方式  
               randomFile = new RandomAccessFile(fileName, "r");  
               // 文件长度，字节数  
               long fileLength = randomFile.length();  
               // 读文件的起始位置  
               int beginIndex = (fileLength > 4) ? 4 : 0;  
               // 将读文件的开始位置移到beginIndex位置。  
               randomFile.seek(beginIndex);  
               byte[] bytes = new byte[10];  
               int byteread = 0;  
               // 一次读10个字节，如果文件内容不足10个字节，则读剩下的字节。  
               // 将一次读取的字节数赋给byteread  
               while ((byteread = randomFile.read(bytes)) != -1) {  
                   System.out.write(bytes, 0, byteread);  
               }  
           } catch (IOException e) {  
               e.printStackTrace();  
           } finally {  
               if (randomFile != null) {  
                   try {  
                       randomFile.close();  
                   } catch (IOException e1) {  
                   }  
               }  
           }  
       }  
     
       /\*\*  
        \* 显示输入流中还剩的字节数  
        \*/  
       private static void showAvailableBytes(InputStream in) {  
           try {  
               System.out.println("当前字节输入流中的字节数为:" + in.available());  
           } catch (IOException e) {  
               e.printStackTrace();  
           }  
       }  
     
       public static void main(String[] args) {  
           String fileName = "C:/temp/newTemp.txt";  
           ReadFromFile.readFileByBytes(fileName);  
           ReadFromFile.readFileByChars(fileName);  
           ReadFromFile.readFileByLines(fileName);  
           ReadFromFile.readFileByRandomAccess(fileName);  
       }  
   }

public class AppendToFile {  
    /\*\*  
     \* A方法追加文件：使用RandomAccessFile  
     \*/  
    public static void appendMethodA(String fileName, String content) {  
        try {  
            // 打开一个随机访问文件流，按读写方式  
            RandomAccessFile randomFile = new RandomAccessFile(fileName, "rw");  
            // 文件长度，字节数  
            long fileLength = randomFile.length();  
            //将写文件指针移到文件尾。  
            randomFile.seek(fileLength);  
            randomFile.writeBytes(content);  
            randomFile.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
  
    /\*\*  
     \* B方法追加文件：使用FileWriter  
     \*/  
    public static void appendMethodB(String fileName, String content) {  
        try {  
            //打开一个写文件器，构造函数中的第二个参数true表示以追加形式写文件  
            FileWriter writer = new FileWriter(fileName, true);  
            writer.write(content);  
            writer.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
  
    public static void main(String[] args) {  
        String fileName = "C:/temp/newTemp.txt";  
        String content = "new append!";  
        //按方法A追加文件  
        AppendToFile.appendMethodA(fileName, content);  
        AppendToFile.appendMethodA(fileName, "append end. \n");  
        //显示文件内容  
        ReadFromFile.readFileByLines(fileName);  
        //按方法B追加文件  
        AppendToFile.appendMethodB(fileName, content);  
        AppendToFile.appendMethodB(fileName, "append end. \n");  
        //显示文件内容  
        ReadFromFile.readFileByLines(fileName);  
    }  
}

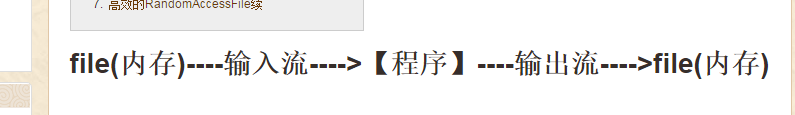
1. java中抛出异常的两种方式

<https://my.oschina.net/kaixuan1995/blog/366336>

1. 读excel文件，写word文件 代码

package com.qunar.hotel.qhotelagent.system.agents.service;  
  
import com.sun.org.apache.xpath.internal.SourceTree;  
import org.apache.poi.xssf.usermodel.XSSFCell;  
import org.apache.poi.xssf.usermodel.XSSFRow;  
import org.apache.poi.xssf.usermodel.XSSFSheet;  
import org.apache.poi.xssf.usermodel.XSSFWorkbook;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Created by ruiyang.wan on 2016/12/29.  
 \*/*public class ReadAndWrite {  
 private String sql;  
 private String wrapperId;  
 private String signSubject;  
 private String defaultLanguage;  
 private String noticeLanguage;  
 private String settlementUnit;  
 public static void main(String[] args) throws Exception {  
  
 ReadAndWrite readAndWrite = new ReadAndWrite();  
 List<List<String>> list = readAndWrite.readXlsx();  
/\* File file\_read = new File("D:/hotel/demo\_script/demo.txt"); //用反斜杠  
 File file\_write = new File("D:/hotel/demo\_script/sql.tx

t");  
 BufferedReader reader = null;  
 BufferedWriter writer = null;  
 try {  
 reader = new BufferedReader(new FileReader(file\_read));  
 writer = new BufferedWriter(new FileWriter(file\_write));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
 if(file\_read.isFile() && file\_read.exists()) {  
  
 }\*/  
  
 }  
  
 private List<List<String>> readXlsx() throws Exception {  
 InputStream is = new FileInputStream("D:/hotel/demo\_script/read.xlsx");  
 File file\_write = new File("D:/hotel/demo\_script/sql.txt");  
 PrintStream ps = new PrintStream(new FileOutputStream(file\_write));  
 ps.println("start");  
 XSSFWorkbook xssfWorkbook = new XSSFWorkbook(is);  
 List<List<String>> result = new ArrayList<List<String>>();  
 for (XSSFSheet xssfSheet : xssfWorkbook) {  
 if(xssfSheet == null)  
 continue;  
 for (int rowNum = 1; rowNum <= xssfSheet.getLastRowNum(); rowNum++) {  
 sql = "update qhagent.hotel\_agents set ";  
 wrapperId = " where wrapper\_id = ";  
 signSubject = " sign\_subject = ";  
 defaultLanguage = " default\_language = ";  
 noticeLanguage = " notice\_language = ";  
 settlementUnit = " settlement\_unit = ";  
  
 XSSFRow xssfRow = xssfSheet.getRow(rowNum);  
 int minColIx = xssfRow.getFirstCellNum();  
 int maxColIx = xssfRow.getLastCellNum();  
 System.*out*.println("minColIx: " + minColIx);  
 System.*out*.println("maxColIx: " + maxColIx);  
 List<String> rowList = new ArrayList<String>();  
 for (int colIx = minColIx; colIx < maxColIx; colIx++) {  
 XSSFCell cell = xssfRow.getCell(colIx);  
 if (cell == null) {  
 continue;  
 }  
 System.*out*.println(colIx);  
  
 switch (colIx)  
 {  
 case 0:  
 wrapperId = wrapperId + '"' + cell.getStringCellValue() + '"';  
 break;  
 case 1:  
 signSubject = signSubject + '"' + cell.getStringCellValue() + '"';  
 break;  
 case 2:  
 defaultLanguage = defaultLanguage + '"' + cell.getStringCellValue() + '"';  
 break;  
 case 3:  
 noticeLanguage = noticeLanguage + '"' + cell.getStringCellValue() + '"';  
 break;  
 case 4:  
 settlementUnit = settlementUnit + '"' + cell.getStringCellValue() + '"';  
 break;  
 default:  
 throw new Exception("数据文件解析错误");  
 }  
 rowList.add(cell.toString());  
 }  
 sql = sql + signSubject + ',' + defaultLanguage + ',' + noticeLanguage + ',' + settlementUnit + wrapperId;  
 ps.append(sql);  
 ps.append("\n");  
  
 result.add(rowList);  
 }  
 }  
 return result;  
 }  
  
}

1. ajax接受Date类型的数据是会把date类型转化成时间戳，后端传给前端的 ajax数据，通常就集中类型，就是String，boolean，int等类型
2. **java的文件读写操作**
3. 
4. **File类：**
5. 在java中，所有东西都是文件，可以用File类来表示一个文件，既可以是文件，也可以是文件夹
6. File类中的 isDirectory(file) 方法，判断文件是一个文件夹（目录）还是一个文件
7. file.getPath() 打印文件的路径，即转换成一个路径名字符串
8. file.seperator 判断在不同的系统中斜杠的方向，windows中是 \\右斜杠，linux中是左斜杠 //
9. file. createNewFile(); 不存在该文件，创建该文件，返回值是boolean类型
10. file.delete() 删除一个文件或者整个文件夹（目录），返回结果为boolean类型
11. file.exists() 测试此抽象路径名表示的文件或目录是否存在
12. file.getParent() 返回此抽象路径名父目录的路径名字符串
13. file.list(); 返回一个字符串数组，字符串指定此抽象路径表示的目录中的文件和目录

        String[] str = f.list();  
        for(int i=0;i<str.length;i++){  
            System.out.println(str[i]);  
        }

1. 扫描一个磁盘中的所有文件，并且打印到控制台上面
2. import java.io.File;  
   public class Demo81 {  
       public static void main(String[] args) {  
           String str = "C:\\";  
           File f = new File(str);  
           fun(f);  
       }  
       //要不断列出，因为给出的File类对象可能是一个目录  
       public static void fun(File f){  
           //判断给定的路径是否是目录，如果是目录在列出  
           if(f.isDirectory()){  
               File[] file = f.listFiles();  
               //再依次循环进行判断  
               try{  
                   for(int i = 0;i < file.length;i++){  
                       //继续把内容传入到fun方法之中进行验证  
                       fun(file[i]);                }  
               }catch(Exception e){}  
           }  
           else{  
               System.out.println(f);  
           }  
       }  
     
   }
3. File类关注的是文件在磁盘上的存储，流类关注的是文件内容
4. File类不是文件流类，只能代表一个文件或是目录的路径名
5. FileInputStream类是流类，关注的是文件内容，而File类关注的是文件在磁盘上的存储
6. FileInputStream类或者FileReader类的构造函数有多个，典型的有两个：
7. 一：使用File对象作为参数（这个还可以处理文件或者目录名）
8. 二：使用表示路径的String对象作为参数（这个固定路径下的文件）

**InputStream与Reader之间的区别**

InputStream与Reader都是抽象类，并不直接地从文件或者套接字（socket）中读取数据

主要的区别是：

InputStream用于读取二进制数据（字节流方式）

Reader 用于读取文本数据（字符流方式），即是Unicode

读取的东西本质上是字节，只是会有一套字符编码方案，把字节转换成文本

Reader类使用字符编码来解码字节，并返回字符给调用者

Reader类要么使用平台的默认字符编码，要么用Charset对象或者String类型的字符编码（UTF-8）

类之间的继承关系

**FileReader** extends InputStreamReader extends **Reader** implements Readable,Closeable

**FileInputStream** extends **InputStream** implements Closeable

**BufferedReader** extends **Reader**



<http://www.jb51.net/article/44684.htm> IO流详解

<http://blog.csdn.net/readiay/article/details/45540067> 读取excel并解析文本，格式化输出

1、根据流的数据对象来分：  
高端流：所有的内存中的流都是高端流，比如：InputStreamReader    
低端流：所有的外界设备中的流都是低端流，比如InputStream，OutputStream   
如何区分：所有的流对象的后缀中包含Reader或者Writer的都是高端流，反之，则基本上为低端流，不过也有例外，比如PrintStream就是高端流

2、根据数据的流向来分：  
输出流：是用来写数据的，是由程序（内存）--->外界设备  
输入流：是用来读数据的，是由外界设备--->程序（内存）  
如何区分：一般来说输入流带有Input，输出流带有Output

3、根据流数据的格式来分：  
字节流：处理声音或者图片等二进制的数据的流，比如InputStream   
字符流：处理文本数据（如txt文件）的流，比如InputStreamReader    
如何区分：可用高低端流来区分，所有的低端流都是字节流，所有的高端流都是字符流

4、根据流数据的包装过程来分：  
原始流：在实例化流的对象的过程中，不需要传入另外一个流作为自己构造方法的参数的流，称之为原始流。  
包装流：在实例化流的对象的过程中，需要传入另外一个流作为自己构造方法发参数的流，称之为包装流。  
如何区分：所以的低端流都是原始流，所以的高端流都是包装流

1. 日期格式必须是这样（大小写），为了区分mont，h和minute之间的区别

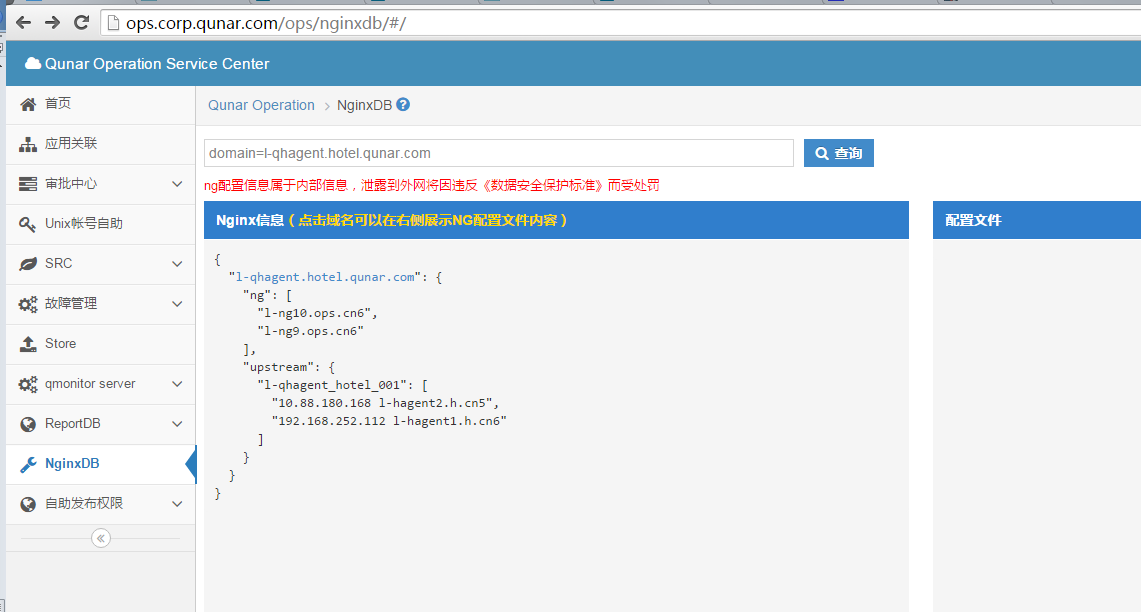
SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

1. mysql大全 <http://www.cnblogs.com/shockerli/p/1000-plus-line-mysql-notes.html>
2. 注意数组是从0开始的,hotel\_agent有一次从excel里面导出来的数组，从1开始遍历，导致少了一个数据
3. Switch(xxx) { case 1: break; case 2: break; default: ; }
4. <http://l-qhagent.hotel.qunar.com/> 这个是ng里面配置的域名

<http://l-hagent1.h.cn6.qunar.com:8080/> 这个是具体的ip和端口访问

在nginx里面 可能会配置一个ip多个域名，ng会按照某规则将请求打到某服务器上，因此后端的发布一定要每台机器都要发布，前端是关联版本号的，因此前端只需要发布到特定的服务器上即可

查询去哪儿网的ng配置 在ops里面



可以发现l-qhagent.hotel.qunar.com 这个域名配置了两台服务器

1. 在interfere 的service前面，不要加上，这是在implement类里面加上去的！！

（注意不要引错了jar包）

@Service(注意这个是SpringframeWork的jar包)

@Component(注意这个也是springframeWork的jar包)

然后直接使用@Resource才可以使用

@Service(“xxx”) 这个xxx的作用是防止service接口有多个实现类，这个时候xxx是为了标示某个实现类，在

在@Resource(“xxx”)中就可以指定对象是这个实现类的对象

1. Select下拉框中不要用disable=”disable” 这个属性来置灰这个字段，因为这样表单中也不能获取到select这个值，要用readonly=”readonly” 这个属性来置灰select下拉框，但是select是不支持readonly属性的，

我采取的方法就是前端要是传过来的值是null，后端mapper.xml文件中就不修改这个字段

Select的只读显示？？？？？？？？？？？？？？？？？？？？

<http://www.cnblogs.com/love540376/p/5822926.html>

1. 空格字符串和空字符串是不一样的，要判断一个字符串是都是空格字符串，首先要去掉前后的空格，采用的是 trim() 方法 ，再判断是都是 isEmpty()

Str.trim().isEmpty()

<http://www.cnblogs.com/ayan/p/3524816.html>

1. 将字符串变成数组 <http://blog.csdn.net/cathylou/article/details/52449433>

String[] relateManagerList = accountRelate.getRelateManager().split(",");

1. 将数组变成字符串 <http://www.cnblogs.com/SharkBin/p/4512561.html>

如果是 “字符串数组” 转 “字符串”，只能通过循环，没有其它方法

String[] str = {"abc", "bcd", "def"};

StringBuffer sb = new StringBuffer();

for(int i = 0; i < str.length; i++){

sb. append(str[i]);

}

String s = sb.toString();

如果是 “字符数组” 转 “字符串” 可以通过下边的方法

char[] data={'a','b','c'};

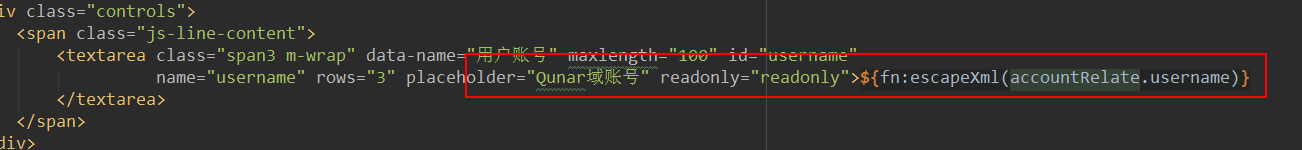
String s=new String(data);

StringUtils.join(Object array[],String separator)

将数组以符号或其他字符串为间隔组成新的字符串

Object array[] 需要转换的数组。separator组成新串的间隔符号，如 "," "|"

1. for(type element : array) { }
2. <textarea> 文本框里面的内容，注意标签换行的话，会在文本框内容中出现几个空格的



1. @PathVariable出现点号&quot;.&quot;时导致路径参数截断获取不全的解决办法，路径参数中出现小数点

<https://yq.aliyun.com/articles/28729>

1. <sql id="columns">  
    id, username, role, relate\_manager relateManager, relate\_agent relateAgent  
   </sql>
2. Mapper.xml文件里面要取别名
3. public boolean deleteByUsername(String username) {  
    return accountRelateMapper.deleteByUsername(username)>0;  
   }

这个是impl里面的方法，注意这个方法的返回值

1. EL表达式里面的if else逻辑判断 <c:choose> </c:when > </c:otherwise> </c:choose>
2. Interface xxxService =》class xxxServiceImpl implements xxxService （其中方法调用接口xxxMapper中的方法）
3. Js里面判断两个字符串是否相同

因为js是弱类型的语言，因此可以直接使用 == 进行判断 str == “abc” 即可，这时会类型的转化（找值）

=== 进行判断的话就是要求两边的类型要相同，值相同，才会返回true（找值和类型）

<http://www.jb51.net/article/69039.htm>

1. Js 直接 alert(typeof str), 即可判断变量str的类型，注意没有typeof没有方法括号啥的
2. Window.location.href=”xxxurl” 直接进行路径跳转 js页面跳转
3. a标签加入单击事件 屏蔽href跳转页面

<http://blog.163.com/huang_qy/blog/static/615601452012101411625600/>

a标签的href和onclick属性

http://www.cnblogs.com/happykakeru/archive/2011/10/24/2222702.html

1. Js中 onclick=”save();return false;” return false

也可以用 onclick=”return false”

表示 不会执行默认事件（取消浏览器默认行为），如果是表单，则不会被提交，可以在save()函数里面控制？？？？？？？？？？？？？？？？？？

上面是错误的想法，上面就是执行完save()函数，并不会因为save里面的返回值改变不执行浏览器的默认行为

你可以这样写 onclick=”return method\_onclick();” 这样的话，就可以直接在method\_onclick() 函数里面控制返回值，这个函数也可以传参数

<http://blog.csdn.net/ydm19891101/article/details/49683107>

1. Java里面判断两个字符串是否相同

Java里面字符串是引用类型，因此判断不能用 == (基本类型可以用 ==)

用equals A.equals(B) 判断A，B的值是否相同

要是既判断值有判断引用地址是否相同，就可以用 ==（判断对象）

StringUtils.equals(A, B) 也可以用来判断两个字符串值是否相等，当然也可以用来比较其他类型的值是否相等 ，这是封装了一层的equals方法

<http://blog.csdn.net/aiqinhai1016_/article/details/9929447>

1. Ajax请求从后端拿回来一个值，不管是什么是什么格式，都会转化为json格式（也就是String类型）切记 <http://blog.csdn.net/qq_18297675/article/details/51862827>
2. String a = new String(); a是分配了内存空间，但值为空，是绝对的空，是一种有值（值存在为空而已）

a.isEmpty() 为true

String b = ""; b是分配了内存空间，值为空字符串，是相对的空，是一种有值（值存在为空字串）  
 b.isEmpty() 为true

String c = null; c未分配存储空间 c==null 为true

1. 常用的jquery获取表单对象的属性与值

<http://www.cnblogs.com/thingk/archive/2013/03/01/2938957.html>

1. Confirm框

1、confirm会阻塞当前进程。也就是说，在confirm语句被确定或者被取消之前，位于confirm后面的代码不会得到执行。  
2、confirm的返回的结果是一个Boolean类型的，即点击确定时返回true，点击取消返回false。  
3、用变量var trueOrFalse = confirm（“你要提醒用户的消息！”）；再用一个if（！trueOrFalse）{return；}else{删除数据等等操作}。这里的return ；是一个程序中断语句。程序执行过程中遇到该语句，即停止运行下面的代码。  
4、结合问题，我判断你应该是没有添加return

1. 如果ajax没有任何返回值，就会弹出alert框，提示ajax没有返回值
2. 建表CREATE TABLE `accounts\_relation` (  
      `id` int(10) unsigned NOT NULL AUTO\_INCREMENT,  
      `username` varchar(24) NOT NULL COMMENT 'QSSO登录用户名',  
      `role` varchar(20) NOT NULL DEFAULT 'USER' COMMENT '角色权限',  
      `relate\_manager` text NOT NULL DEFAULT '' COMMENT '关联管理员',  
      `relate\_agent` text NOT NULL DEFAULT '' COMMENT '关联代理商',  
      PRIMARY KEY (`id`)  
    ) ENGINE=InnoDB AUTO\_INCREMENT=22 DEFAULT CHARSET=utf8 COMMENT='权限关联表'

INSERT INTO qhagent.accounts\_relation(username,role) SELECT username,role FROM qhagent.accounts

1. 把一个表中的数据查到另外一个表中（已有表和字段）

INSERT INTO TestTable (FirstName, LastName)   
SELECT FirstName, LastName   
FROM Person.Contact   
WHERE EmailPromotion = **2**

1. 把一个表中的数据查到另外一个表中（没有表和字段，执行语句的时候就新建了，字段和老表一样）

SELECT FirstName, LastName   
INTO TestTable   
FROM Person.Contact   
WHERE EmailPromotion = **2**

1. 将汉字转为URL编码

java.net.URLEncoder.encode("中文","utf-8") 表示汉字是有utf-8编码的

将URL编码转化为汉字 java.net.URLDecoder.decode（”asdfasfe”,”utf-8”）

1. String keyWord = URLDecoder.decode("%C4%E3%BA%C3", "GBK");
2. System.out.println(keyWord);  //输出你好
3. // 将普通字符创转换成application/x-www-from-urlencoded字符串
4. String urlString = URLEncoder.encode("你好", "GBK");  //输出%C4%E3%BA%C3
6. System.out.println(urlString);
7. 字符串的截取函数 substring(start, length)

str＝str.substring(int beginIndex);截取掉str从首字母起长度为beginIndex的字符串，将剩余字符串赋值给str；

str＝str.substring(int beginIndex，int endIndex);截取str中从beginIndex开始至endIndex结束时的字符串，并将其赋值给str;

1. 去掉最后一个字符 str=str.substring(0, str.length()-1)
2. Java 将String转化成 List

  String abc = "aaa,bbb,ccc";

     String[] arr = abc.split(","); //先转化成数组

    List<String> list = java.util.Arrays.aslist(abc); //由数组转化成list，list里面元素是String类型

1. 将List再转化成Map

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

params.put("idsList", list); //这就将上面的list变成了map里面的一个键值

1. 数组arrays，LinkedList，ArrayList，Vector
2. 对数组进行随机访问和迭代操作的速度是最快的；对LinkedList进行插入和删除操作的速度是最快的；对ArrayList进行随机访问的速度也很快；Vector类在各方面没有突出的性能，且此类已不提倡使用了。
3. List和Array之间的转化

List转Array

ArrayList<String> list = new ArrayList<String>();

String[] array = new String[list.size()]; //这里直接就转化成这样大小的Array，元素是String类型的  
String[] s=list.toArray(array);

最好用new的方式构建，因为toArray默认转换为Object[]对象！

Array转List

String[] string=new String[]{"s1","s2","s3","s4","s5"};  
List<String> li= Arrays.asList(string);

Array是表态连续分配的一片内存区域，与ArrayList相比、不能动态改变大小，通过Arrays进行sort、binarySearch等操作；

ArrayList是继承自List的可动态改变大小的数组，和Array一样要求连续分配，内部封闭了一个Object数组，许多方法直接调用Arrays实现；

Vector和ArrayList功能基本一致，但Vector是线程安全的。

效率由高到低依次为：Array、ArrayList、Vector。

<http://blog.csdn.net/u010525970/article/details/52381730>

1. 判断一个字符串中是否包含另外一个字符或者字符串

str.indexOf("ABC")!=-1 str中是否包含 “ABC”

1. <http://qunar.it/s/?kw=%E4%B8%87%E7%91%9E%E9%98%B3> 这个可以通过中文名字获取qunar的英文名
2. Java的HttpClient 使用

可以发送get，POST请求，上传表单，通过http链接ssh

<http://blog.csdn.net/zhouzhiwengang/article/details/42062457>

1. java中的url 编码与解码
2. <http://www.cnblogs.com/shishm/articles/1614407.html>
3. Java 解析json 字符串成 数组或列表 <http://www.linuxidc.com/Linux/2016-02/128410.htm>

可以运用类，将json字符串里面的东西转化成一个元素是这个类的数组或者元素是这个类的列表

import java.util.ArrayList;  
import java.util.List;

import net.sf.json.JSONArray; //json转变成数组的类

import com.bean.Student; //导入这个类

String jsonStr = "[{\"age\": 25,\"gender\": \"female\",\"grades\": \"三班\",\"name\": \"露西\",\"weight\": 51.3},{\"age\": 26,\"gender\": \"male\",\"grades\": \"三班\",\"name\": \"杰克\",\"weight\": 66.5},{\"age\": 25,\"gender\": \"female\",\"grades\": \"三班\",\"name\": \"莉莉\",\"weight\": 55}]";

JSONArray jsonArray = JSONArray.fromObject(jsonStr);

Student[] stus = new Student[3];  
List<Student> list = new ArrayList<Student>();

stus = (Student[]) JSONArray.toArray(jsonArray, Student.class);//转换成数组  
list = (List<Student>) JSONArray.toCollection(jsonArray, Student.class);//转换成列表

1. **import** net.sf.json.JSONArray;
2. **import** net.sf.json.JSONObject;
3. String joStr = "{name:\"张三\",age:\"20\"}";
4. //将json字符串转化为JSONObject
5. JSONObject jsonObject = JSONObject.fromObject(joStr);
6. //通过getString("")分别取出里面的信息
7. String name = jsonObject.getString("name");
8. String age = jsonObject.getString("age");
10. String jaStr = "[{user:{name:\"张三\",age:\"20\"}},{score:{yuwen:\"80\",shuxue:\"90\"}}]";
11. //将jsonArray字符串转化为JSONArray
12. JSONArray jsonArray = JSONArray.fromObject(jaStr);
13. //取出数组第一个元素
14. JSONObject jUser = jsonArray.getJSONObject(0).getJSONObject("user");  //注意这里是对象
15. //取出第一个元素的信息，并且转化为JSONObject
16. String name2 = jUser.getString("name");
17. String age2 = jUser.getString("age");
18. //取出数组第二个元素，并且转化为JSONObject
19. JSONObject jScore = jsonArray.getJSONObject(1).getJSONObject("score");
20. //取出第二个元素的信息
21. String yuwen = jScore.getString("yuwen");
22. String shuxue = jScore.getString("shuxue");
23. String t= {"initial":"R","name":"reiz","likes":["JavaScript","Skiing","Apple Pie"]}
24. JSONObject jo=new JSONObject(t); //json-lib中是jo=JSONObject.fromObject(t);
25. JSONArray likes=jo.getJSONArray("likes"); //得到likes数组，注意这里是数组
26. System.out.prinlnt(like.getString(2)); //打印Apple Pie
27. 通过数组或列表生成 json字符串

      Student stu1 = new Student();  
        Student stu2 = new Student();  
        Student stu3 = new Student();

        stu1.setAge(25);  
        stu1.setGender("female");  
        stu1.setGrades("三班");  
        stu1.setName("露西");  
        stu1.setWeight(51.3f);

        stu2.setAge(26);  
        stu2.setGender("male");  
        stu2.setGrades("三班");  
        stu2.setName("杰克");  
        stu2.setWeight(66.5f);

        stu3.setAge(25);  
        stu3.setGender("female");  
        stu3.setGrades("三班");  
        stu3.setName("莉莉");  
        stu3.setWeight(55.0f);

        Student[] stus = new Student[3];  
        List<Student> list = new ArrayList<Student>();

        stus[0] = stu1; //数组采用赋值即可  
        stus[1] = stu2;  
        stus[2] = stu3;

        list.add(stu1); //list使用方法add  
        list.add(stu2);  
        list.add(stu3);

        String jsonStr1 = JSONArray.fromObject(stus).toString();  
        String jsonStr2 = JSONArray.fromObject(list).toString();

        System.out.println(jsonStr1);  
        System.out.println(jsonStr2);

1. Json简单介绍 <http://www.jb51.net/article/50009.htm>

Javascript object notation JavaScript对象表示法，轻量级的文本数据交换格式，独立于语言和平台

Json的两种结构：

1.“名称/值”对的集合（A collection of name/value pairs）。不同的语言中，它被理解为对象（object），纪录（record），结构（struct），字典（dictionary），哈希表（hash table），有键列表（keyed list），或者关联数组 （associative array）。  
2. 值的有序列表（An ordered list of values）。在大部分语言中，它被理解为数组（array）。  
这些都是常见的数据结构。

对象是一个无序的“‘名称/值'对”集合。一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值' 对”之间使用“,”（逗号）分隔。

数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）分隔。

json串，在js中会解析成对象。其中对象的名称比如：employees、firstName，必须用""或者''引起来才行。后面的value，如果是int和bool（js是不区分的哦），可以不用""或者''引起来，其他的字符串必须引起来，尤其是记得把时间日期引起来！

里面的是一个对象，如果是多个对象，则用逗号间隔，即{},{}，这样就组成了一个对象序列，为了辨别开始和结束，则需要加上[]才能进行传递，则实际传递的形式应该是[{},{}]，如果只要传递一个对象，则{}的形式就可以了。

对象属性的表现方式：

属性必须在""里面

属性与值之间用:隔开

属性之间用,来分隔

如果属性的值为数组，则用[]包括起来

实际传递的数据格式可能有：

1、{"属性1":值1,"属性2":值2}，如果值为字符串，则也需要用""括起来（下同）。

2、{"属性1":值1,"属性2":[值1，值2]}，其中属性2是一个数组，包含了值1和值2。

3、{"属性1":值1,"属性2":{"属性a":值a,"属性b":[值b，值c]}}，这个比较复杂了，属性2是个对象，这个对象由包含属性a和属性b，属性b又是一个数组包含值b和值c。我想这些应该是最基本了，其余的就是在此基础上的拓展而已。

1. 方法头中出现
2. public String getToQtalkName(String username) throws UnsupportedEncodingException

这个是方法中抛出异常，但是不想处理这个异常，于是抛出这个异常，给调用者处理异常，如果其他方法调用这个方法仍然没有处理这个异常，那么在方法头上面也要throws一个异常

1. **httpclient教程**<http://www.cnblogs.com/loveyakamoz/archive/2011/07/21/2112804.html>

HttpClient使用详解 <http://blog.csdn.net/wangpeng047/article/details/19624529>

使用方法：

创建HttpClient对象->创建HttpGet或者HttpPost对象（请求方式的对象）->执行请求，获取response->释放链接（注意最后都要释放链接）

1. 创建HttpClient对象。

2. 创建请求方法的实例，并指定请求URL。如果需要发送GET请求，创建HttpGet对象；如果需要发送POST请求，创建HttpPost对象。

3. 如果需要发送请求参数，可调用HttpGet、HttpPost共同的setParams(HetpParams params)方法来添加请求参数；对于HttpPost对象而言，也可调用setEntity(HttpEntity entity)方法来设置请求参数。

4. 调用HttpClient对象的execute(HttpUriRequest request)发送请求，该方法返回一个HttpResponse。

5. 调用HttpResponse的getAllHeaders()、getHeaders(String name)等方法可获取服务器的响应头；调用HttpResponse的getEntity()方法可获取HttpEntity对象，该对象包装了服务器的响应内容。程序可通过该对象获取服务器的响应内容。

6. 释放连接。无论执行方法是否成功，都必须释放连接

Get请求

import org.apache.http.HttpEntity;  
import org.apache.http.client.ClientProtocolException;  
import org.apache.http.client.methods.CloseableHttpResponse;  
import org.apache.http.client.methods.HttpGet;  
import org.apache.http.impl.client.CloseableHttpClient;  
import org.apache.http.impl.client.HttpClients;  
import org.apache.http.util.EntityUtils;  
import org.apache.http.ParseException;  
  
import java.io.IOException;  
import java.io.UnsupportedEncodingException;

{  
 CloseableHttpClient httpclient = HttpClients.*createDefault*();  
 String user = java.net.URLEncoder.*encode*(username, "utf-8");  
 String url = "http://qunar.it/s/?kw=" + user;  
 try {  
 // 创建httpget.创建的时候要把url当成参数传进去  
 HttpGet httpget = new HttpGet(url);  
 System.*out*.println("executing request " + httpget.getURI());  
 // 执行get请求.  
 CloseableHttpResponse response = httpclient.execute(httpget);  
 // 获取响应实体  
 HttpEntity entity = response.getEntity();  
 // 打印响应状态  
 System.*out*.println(response.getStatusLine());  
 if (entity != null) {  
 // 打印响应内容长度  
 System.*out*.println("Response content length: " + entity.getContentLength());  
 // 打印响应内容  
 String reponseBody = EntityUtils.*toString*(entity); //注意这里只能使用entity一次  
 System.*out*.println("Response content: " + reponseBody);   
 return reponseBody;  
 }  
 }  
 catch (ClientProtocolException e) {  
 e.printStackTrace();  
 } catch (ParseException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 // 关闭连接,释放资源  
 try {  
 httpclient.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 return "null";  
}

1. Post请求和Get请求有一点不一样，因为Post请求需要创建参数队列

import org.apache.http.message.BasicNameValuePair;  
import org.apache.http.client.entity.UrlEncodedFormEntity;  
public void post() {  
 // 创建默认的httpClient实例.   
 CloseableHttpClient httpclient = HttpClients.*createDefault*();  
 // 创建httppost   
 HttpPost httppost = new HttpPost("http://localhost:8080/myDemo/Ajax/serivceJ.action");

// 创建参数队列，这里和get请求不一样   
 List<namevaluepair> formparams = new ArrayList<namevaluepair>();  
 formparams.add(new BasicNameValuePair("type", "house"));  
 UrlEncodedFormEntity uefEntity;  
 try {  
 uefEntity = new UrlEncodedFormEntity(formparams, "UTF-8");  
 httppost.setEntity(uefEntity);

System.*out*.println("executing request " + httppost.getURI());  
 CloseableHttpResponse response = httpclient.execute(httppost);  
 try {  
 HttpEntity entity = response.getEntity();  
 if (entity != null) {  
 System.*out*.println("Response content: " + EntityUtils.*toString*(entity, "UTF-8"));  
 }  
 } finally {  
 response.close();  
 }  
 } catch (ClientProtocolException e) {e.printStackTrace();}

catch (UnsupportedEncodingException e1) {e1.printStackTrace();}

catch (IOException e) {e.printStackTrace();}

finally {  
 // 关闭连接,释放资源   
 try {  
 httpclient.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

1. @PathVariable与@RequestParam的区别

@PathVariable请求路径，这是在请求路径里面的

@RequestParam请求参数，这是在请求路径url后面那个？之后的参数的

Springmvc/user/page.do?pageSize=3&pageNow=2  请求路径？请求参数

1. 如果在请求url路径中是有带 . 这个需要这样处理

@RequestMapping(value = "/update/{username:.+}", method = RequestMethod.*POST*)

1. IDEA快捷键
2. Ctrl+Shift + Enter，语句完成
3. “！”，否定完成，输入表达式时按 “！”键
4. Ctrl+E，最近的文件
5. Ctrl+Shift+E，最近更改的文件
6. Shift+Click，可以关闭文件
7. Ctrl+[ OR ]，可以跑到大括号的开头与结尾
8. Ctrl+F12，可以显示当前文件的结构
9. Ctrl+F7，可以查询当前元素在当前文件中的引用，然后按 F3 可以选择
10. Ctrl+N，可以快速打开类
11. Ctrl+Shift+N，可以快速打开文件
12. Alt+Q，可以看到当前方法的声明
13. Ctrl+P，可以显示参数信息
14. Ctrl+Shift+Insert，可以选择剪贴板内容并插入
15. Alt+Insert，可以生成构造器/Getter/Setter等
16. Ctrl+Alt+V，可以引入变量。例如：new String();  自动导入变量定义
17. Ctrl+Alt+T，可以把代码包在一个块内，例如：try/catch
18. Ctrl+Enter，导入包，自动修正
19. Ctrl+Alt+L，格式化代码
20. Ctrl+Alt+I，将选中的代码进行自动缩进编排，这个功能在编辑 JSP 文件时也可以工作
21. Ctrl+Alt+O，优化导入的类和包
22. Ctrl+R，替换文本
23. Ctrl+F，查找文本
24. Ctrl+Shift+Space，自动补全代码
25. Ctrl+空格，代码提示（与系统输入法快捷键冲突）
26. Ctrl+Shift+Alt+N，查找类中的方法或变量
27. Alt+Shift+C，最近的更改
28. Alt+Shift+Up/Down，上/下移一行
29. Shift+F6，重构 - 重命名
30. Ctrl+X，删除行
31. Ctrl+D，复制行
32. Ctrl+/或Ctrl+Shift+/，注释（//或者/\*\*/）
33. Ctrl+J，自动代码（例如：serr）
34. Ctrl+Alt+J，用动态模板环绕
35. Ctrl+H，显示类结构图（类的继承层次）
36. Ctrl+Q，显示注释文档
37. Alt+F1，查找代码所在位置
38. Alt+1，快速打开或隐藏工程面板
39. Ctrl+Alt+left/right，返回至上次浏览的位置
40. Alt+left/right，切换代码视图
41. Alt+Up/Down，在方法间快速移动定位
42. Ctrl+Shift+Up/Down，向上/下移动语句
43. F2 或 Shift+F2，高亮错误或警告快速定位
44. Tab，代码标签输入完成后，按 Tab，生成代码
45. Ctrl+Shift+F7，高亮显示所有该文本，按 Esc 高亮消失
46. Alt+F3，逐个往下查找相同文本，并高亮显示
47. Ctrl+Up/Down，光标中转到第一行或最后一行下
48. Ctrl+B/Ctrl+Click，快速打开光标处的类或方法（跳转到定义处）
49. Ctrl+Alt+B，跳转到方法实现处
50. Ctrl+Shift+Backspace，跳转到上次编辑的地方
51. Ctrl+O，重写方法
52. Ctrl+Alt+Space，类名自动完成
53. Ctrl+Alt+Up/Down，快速跳转搜索结果
54. Ctrl+Shift+J，整合两行
55. Alt+F8，计算变量值
56. Ctrl+Shift+V，可以将最近使用的剪贴板内容选择插入到文本
57. Ctrl+Alt+Shift+V，简单粘贴
58. Shift+Esc，不仅可以把焦点移到编辑器上，而且还可以隐藏当前（或最后活动的）工具窗口
59. F12，把焦点从编辑器移到最近使用的工具窗口
60. Shift+F1，要打开编辑器光标字符处使用的类或者方法 Java 文档的浏览器
61. Ctrl+W，可以选择单词继而语句继而行继而函数
62. Ctrl+Shift+W，取消选择光标所在词
63. Alt+F7，查找整个工程中使用地某一个类、方法或者变量的位置
64. Ctrl+I，实现方法
65. Ctrl+Shift+U，大小写转化
66. Ctrl+Y，删除当前行
67. Shift+Enter，向下插入新行
68. psvm/sout，main/System.out.println(); Ctrl+J，查看更多
69. Ctrl+Shift+F，全局查找
70. Ctrl+F，查找/Shift+F3，向上查找/F3，向下查找
71. Ctrl+Shift+S，高级搜索
72. Ctrl+U，转到父类
73. Ctrl+Alt+S，打开设置对话框
74. Alt+Shift+Inert，开启/关闭列选择模式
75. Ctrl+Alt+Shift+S，打开当前项目/模块属性
76. Ctrl+G，定位行
77. Alt+Home，跳转到导航栏
78. Ctrl+Enter，上插一行
79. Ctrl+Backspace，按单词删除
80. Ctrl+"+/-"，当前方法展开、折叠
81. Ctrl+Shift+"+/-"，全部展开、折叠

【调试部分、编译】

1. Ctrl+F2，停止
2. Alt+Shift+F9，选择 Debug
3. Alt+Shift+F10，选择 Run
4. Ctrl+Shift+F9，编译
5. Ctrl+Shift+F10，运行
6. Ctrl+Shift+F8，查看断点
7. F8，步过
8. F7，步入
9. Shift+F7，智能步入
10. Shift+F8，步出
11. Alt+Shift+F8，强制步过
12. Alt+Shift+F7，强制步入
13. Alt+F9，运行至光标处
14. Ctrl+Alt+F9，强制运行至光标处
15. F9，恢复程序
16. Alt+F10，定位到断点
17. Ctrl+F8，切换行断点
18. Ctrl+F9，生成项目
19. Alt+1，项目
20. Alt+2，收藏
21. Alt+6，TODO
22. Alt+7，结构
23. Ctrl+Shift+C，复制路径
24. Ctrl+Alt+Shift+C，复制引用，必须选择类名
25. Ctrl+Alt+Y，同步
26. Ctrl+~，快速切换方案（界面外观、代码风格、快捷键映射等菜单）
27. Shift+F12，还原默认布局
28. Ctrl+Shift+F12，隐藏/恢复所有窗口
29. Ctrl+F4，关闭
30. Ctrl+Shift+F4，关闭活动选项卡
31. Ctrl+Tab，转到下一个拆分器
32. Ctrl+Shift+Tab，转到上一个拆分器

【重构】

1. Ctrl+Alt+Shift+T，弹出重构菜单
2. Shift+F6，重命名
3. F6，移动
4. F5，复制
5. Alt+Delete，安全删除
6. Ctrl+Alt+N，内联

【查找】

1. Ctrl+F，查找
2. Ctrl+R，替换
3. F3，查找下一个
4. Shift+F3，查找上一个
5. Ctrl+Shift+F，在路径中查找
6. Ctrl+Shift+R，在路径中替换
7. Ctrl+Shift+S，搜索结构
8. Ctrl+Shift+M，替换结构
9. Alt+F7，查找用法
10. Ctrl+Alt+F7，显示用法
11. Ctrl+F7，在文件中查找用法
12. Ctrl+Shift+F7，在文件中高亮显示用法

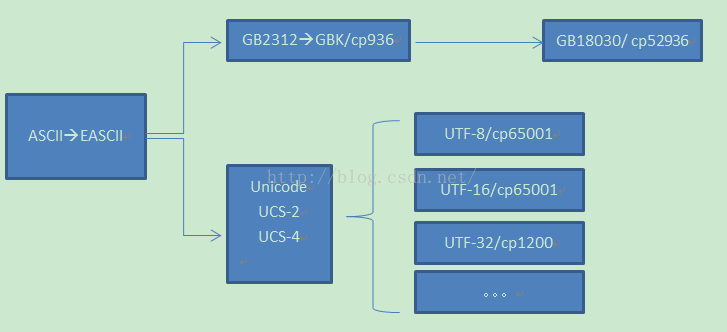
【VCS】

1. Alt+~，VCS 操作菜单
2. Ctrl+K，提交更改
3. Ctrl+T，更新项目
4. Ctrl+Alt+Shift+D，显示变化
5. 在提交上去的代码中,严禁出现system.out.println() 这种
6. 在抛出的异常中，不要值e.printStackTrace() 这种只打印堆栈，要在catch里面做一些处理
7. 注意不要return “null”; 这种返回值，因为会形成误解
8. 判断一个String类型是否为空，用 StringUtils.isEmpty(xxx) 这个里面将这个字符串是null的时候封装起来做了判断，抛出异常
9. Return count>=1 可以有效的代替
10. !(A&&B) -》 !A||!B
11. **字符编码的认识**

字符：一个文字就是一个字符，不同的文字用不同的数字表示

字符集：系统支持的所有抽象字符的集合，字符是文字和符号的总称。

字符编码：一套法则，符号集合和数字系统之间做映射关系，将数据通过一定方式表达或者存储，元件不同状态的组合能代表数字系统的数字，因此字符编码就是讲符号转化为计算机可以接受的数字系统的数，称为数字编码。



**ASCII编码：** 128个字符的编码，只占用了一个单字节（8bit）的后面7位，最前面的1位统一为0。

对于最高位为1的时候，就产生了非ASCII编码，如IOS-8850-1编码

**IOS-8850-1编码：**单字节编码，卒子多表示256个字符，可以完全兼容ASCII编码

**GB2312编码：**双字节编码，高字节表示区，低字节表示位（汉字编码）

**BIG5编码：**双字节编码，两个字节来表示一个字符，与GBK编码没有什么关系（汉字编码）

**GBK编码：**由GB2312扩展而来，完全兼容GB2312编码，但不兼容BIG5编码，（汉字编码）

**Unicode编码与UCS编码：**本来由两个组织开发的，后来兼容了，Unicode是世界上最全面的编码

**UTF-8编码：**变长编码方式，采用1-4个字节进行编码，完全兼容ASCII，是Unicode的一种具体编码实现，是在互联网上使用最广的一种Unicode的编码规则，有利于节约网络流量

**UTF-16编码：**只使用2或4个字节编码

<http://www.cnblogs.com/leesf456/p/5317574.html>

<http://blog.csdn.net/ghking1/article/details/47059587>

<http://www.cnblogs.com/skynet/archive/2011/05/03/2035105.html#_参考文献&进一步阅读>

<http://www.cnblogs.com/wsylly/p/3899581.html>

1. 删除数据库表中某一字段相同的记录

DELETE FROM qhagent.accounts\_relation //这个表不能起别名，起了别名就会报错  
 WHERE  
id NOT IN  
 (  
 SELECT id FROM  
 ((SELECT MIN(id) id FROM qhagent.accounts\_relation GROUP BY username HAVING COUNT(\*)>1) dsfg) //注意这个临时表要起别名  
 )  
 AND username IN  
 (  
 SELECT username FROM  
 ((SELECT username FROM qhagent.accounts\_relation GROUP BY username HAVING COUNT(\*)>1) dsfg)  
 )

1. Varchar最大字符，text的大小比varchar更大
2. Cmd里面可以使用notepad++ file 打开当前下面的文件，这个比vim好用多了
3. Ctrl + shift + - 将所有的都变成一行折叠起来
4. Ctrl + shift + + 将所有折叠都展开
5. OutOfMemoryError 内存不足 StackOverflowError 栈溢出
6. **受检异常和非受检异常**

**受检异常** checked exception 非运行时异常

非运行时异常必须进行处理throw或者try{}catch(){}处理，否则编译器报错。

这种在编译时被强制检查的异常称为"受检查的异常"。即在方法的声明中声明的异常。

通常都是由一些外部的偶然因素所引起的，java编译器要求强制处理，程序必须对着一些异常做处理

**非受检异常**

所有继承自RuntimeException的异常：

在方法的声明中没有声明，但在方法的运行过程中发生的各种异常被称为"不被检查的异常"。这种异常是错误，会被自动捕获

通常是程序的逻辑问题，java编译器不要求强制处理，可以处理也可以不处理

Exception常用的方法

Exception.getMessage() 获取异常描述，可以带上抛出异常时候的状态变量

Exception.getCause() 获取导致该异常的原因，一场可以嵌套，getCause可以得到原始异常

Exception.printStackTrace() 在stderr中输出异常堆栈

1. QMQ 完全可靠的消息队列

Qshedule 任务调度中心

Qconfig 统一配置中心

1. Long a = 5000 \* 365 \* 12 \* 24 \* 60 \* 1000L 这个仍然会溢出，这个L应该放在第一个数那里

Double a = 0.58; 实际上程序里面的a是0.57999之类的，所以一定要注意double的精度是有问题的

BigDecimal b = new BigDecimal(“0.58”); 这个b实际上就是 0.58

A.equals(b); 这个时候就会报错，因为两者精度不同，在程序里面也是有微小差别

Money是公司对bigdecimal的封装

import java.util.Arrays; //jdk自带的包  
import com.google.common.primitives.Ints; //Ints是guava的包

int[] array = new int[]{1, 2, 3};  
Arrays.*asList*(array); //[{324123}] 这个方法里面参数是一个泛型，这个传进去就是一个数组  
Ints.*asList*(array); // [1,2,3] //这个方法里面参数是int类型，这个传进去是数组的各个int元素

1. **Guava里面的joiner接口**
2. <dependency>  
    <groupId>com.google.guava</groupId> //guava的依赖配置 pom.xml文件里面  
    <artifactId>guava</artifactId>  
    <version>16.0.1</version>  
   </dependency>
3. 可以连接一个数组，list或者map，
4. 在数组，list，map里面是不允许出现null的，如果有null元素，就会抛异常
5. 要么用useForNull(“text”),将null最后转化为text (不能连续调两次，会抛出异常)
6. 要么用skipNulls(), （不能与userForNull同时使用）
7. Joiner的构造器是private的，无法直接通过new实例化，可以调用on函数

public static Joiner on(String separator) {  
 return new Joiner(separator);}public static Joiner on(char separator) {  
 return new Joiner(String.*valueOf*(separator));} //这个on函数里面参数都是分隔符  
private final String separator;  
private Joiner(String separator) {  
 this.separator = *checkNotNull*(separator);}  
private Joiner(Joiner prototype) {  
 this.separator = prototype.separator;}

//joiner的使用

String content = Joiner.*on*("").join(list);

1. 连接List列表

Joiner joiner = Joiner.*on*(",");  
List<String> list =new ArrayList<String>();  
list.add("xiaoming");  
list.add("xiaohong");  
list.add("wangsan");  
System.*out*.println(joiner.join(list));

结果：xiaoming,xiaohong,wangsan

1. 连接Iterator<T>列表

Joiner joiner = Joiner.*on*(",");  
List<String> list =new ArrayList<String>();  
list.add("xiaoming");  
list.add("xiaohong");  
list.add("wangsan");  
Iterator<String> it=list.iterator();  
System.*out*.println(joiner.join(it));  
结果：xiaoming,xiaohong,wangsan

1. 连接三个字符串

Joiner joiner = Joiner.*on*(",");  
System.*out*.println(joiner.join("123","345","56","43"));  
结果：123,345,56,43

1. 连接字符串和列表

Joiner joiner = Joiner.*on*(",");  
List<String> list =new ArrayList<String>();  
list.add("xiaoming");  
list.add("xiaohong");  
list.add("wangsan");  
StringBuilder builder=new StringBuilder("标题");  
System.*out*.println(joiner.appendTo(builder,list));//连接两个不同类型的话要用appendTo()方法  
结果：标题xiaoming,xiaohong,wangsan

1. Guava Joiner对空指针有着严格的控制，一旦传入的对象包含空指针，Joiner直接抛出Null Pointer Exception
2. 但是joiner提供了下面两个方法处理待拼接集合中的空指针

Joiner.*on*(' ').skipNulls().join(1, null, 3);//1 3  
Joiner.*on*(' ').useForNull("None").join(1, null, 3);//1 None 3

需要注意的是，Joiner 实例是不可变的，skipNulls 和 useForNull 都不是在原实例上修改某个成员变量，而是生成一个新的 Joiner 实例。

1. 替换null值进行连接（跳过和替换null要用appendTo()方法）

Joiner joiner = Joiner.*on*(",").useForNull("123");//替换null为123   
List<String> list =new ArrayList<String>();  
list.add("xiaoming");  
list.add(null);  
list.add("wangsan");  
StringBuilder builder=new StringBuilder("标题");  
System.*out*.println(joiner.appendTo(builder,list));  
结果：标题xiaoming,123,wangsan

1. 跳过null值连接（useForNull() 与skipNulls()这两个方法不能一起用）

Joiner joiner = Joiner.*on*(",").skipNulls();  
List<String> list =new ArrayList<String>();  
list.add("xiaoming");  
list.add(null);  
list.add("wangsan");  
StringBuilder builder=new StringBuilder("标题");  
System.*out*.println(joiner.appendTo(builder,list));

1. 来看一看useForNull() 与skipNulls()的源码

@CheckReturnValue  
public Joiner useForNull(final String nullText) {  
 *checkNotNull*(nullText); //首先要保证这个nullText不是null  
 return new Joiner(this) { //使用复制构造函数保留先前初始化时候设置的分隔符  
 @Override CharSequence toString(@Nullable Object part) { //覆盖了之前的toString()方法  
 return (part == null) ? nullText : Joiner.this.toString(part);  
 }  
  
 @Override public Joiner useForNull(String nullText) {  
 throw new UnsupportedOperationException("already specified useForNull");  
 } //覆盖了userForNull()和skipNulls() 防止重复调用，直接抛异常  
 @Override public Joiner skipNulls() {  
 throw new UnsupportedOperationException("already specified useForNull");  
 }  
 };  
}

@CheckReturnValue  
public Joiner skipNulls() {  
 return new Joiner(this) { //使用复制构造函数保留先前初始化时候设置的分隔符  
 @Override public <A extends Appendable> A appendTo(A appendable, Iterator<?> parts)  
 throws IOException {  
 *checkNotNull*(appendable, "appendable"); //首先判断是否为null，是否初始化  
 *checkNotNull*(parts, "parts"); //判断迭代器初始化了  
 while (parts.hasNext()) { //这个while是去除null  
 Object part = parts.next();  
 if (part != null) { //当不是null的时候就会从新加进去，是null就跳过  
 appendable.append(Joiner.this.toString(part));  
 break; }  
 }  
 while (parts.hasNext()) { //这个while循环和原来的一样，都是构造新字符串  
 Object part = parts.next();  
 if (part != null) {  
 appendable.append(separator);  
 appendable.append(Joiner.this.toString(part));  
 }  
 }  
 return appendable;  
 }  
 @Override public Joiner useForNull(String nullText) {  
 throw new UnsupportedOperationException("already specified skipNulls");  
 }  
 @Override public MapJoiner withKeyValueSeparator(String kvs) {  
 throw new UnsupportedOperationException("can't use .skipNulls() with maps");  
 }  
 };  
}

1. Java map的两种遍历迭代方法(第一种方法比较通用)

<http://www.cnblogs.com/lwlxqlccc/p/6143887.html>

Map<String, String> resultList = new HashMap<String, String>();

Iterator i = resultList.entrySet().iterator();  
 while (i.hasNext()) {  
 Map.Entry e = (Map.Entry)i.next();  
 if (res.containsKey(e.getValue())) {  
 res.put((String)e.getValue(), res.get(e.getValue()) + e.getKey() + ",");  
 } //注意这里e.getValue() 获取的不是String类型，这里要显示转换  
 } //map里面重新设置键的值，只需要key相同，用put(key, value) 即可

Set set= resultList.keySet();//用接口实例接口  
 Iterator iter = set.iterator();  
while (iter.hasNext()) {//遍历二次,速度慢  
 String k=(String)iter.next();  
 System.out.println(k +"="+ m.get(k));  
//System.out.println(iter.next()+"="+ m.get(iter.next()));  
//因为指针判断下一个有没有值 iter.next是当前对象 但是 m.get(iter.next())是下一个值   
 }

1. Map的键转化为set，map的值转化为set

Map<String, String> map = new HashMap<String, String>();   
 map.put("A", "ABC");   
 map.put("K", "KK");   
 map.put("L", "LV");  
// 将Map 的键转化为Set   
Set<String> mapKeySet = map.keySet();   
// 将Map 的值转化为Set   
Set<String> mapValuesSet = new HashSet<String>(map.values());

1. Map的键转化为list，值转化为list

Map<String, String> map = new HashMap<String, String>();   
 map.put("A", "ABC");  
 map.put("K", "KK");  
 map.put("L", "LV");  
 // 将Map Key 转化为List   
 List<String> mapKeyList = new ArrayList<String>(map.keySet());  
 // 将Map Key 转化为List   
 List<String> mapValuesList = new ArrayList<String>(map.values());

1. Map的entry

Map类提供了一个称为entrySet()的方法，这个方法返回一个Map.Entry实例化后的对象集。接着，Map.Entry类提供了一个getKey()方法和一个getValue()方法

Set entries = map.entrySet();  
if (entries!=null){  
 Iterator iterator=entries.iterator();  
 while (iterator.hasNext()) {  
 Map.Entry entry=iterator.next();  
 Object key=entry.getKey();  
 Object value=entry.getValue();  
 ;....  
 }  
}

1. 数组转化为set

String[] arr = {"AA","BB","DD","CC","BB"};  
//数组-->Set   
Set<String> set = new HashSet<String>(Arrays.*asList*(arr)); //先转化为list，再转化为set

1. Set转化成数组

Set<String> set = new HashSet<String>();   
set.add("AA");  
set.add("BB");  
set.add("CC");  
String[] arr = new String[set.size()];  
 //Set-->数组   
set.toArray(arr);

1. List转化为set

List<String> list = new ArrayList<String>();   
list.add("ABC");  
list.add("EFG");

list.add("LMN");  
list.add("LMN");  
 //List-->Set   
Set<String> listSet = new HashSet<String>(list);

1. Set转化为list

Set<String> set = new HashSet<String>();  
set.add("AA");  
set.add("BB");  
set.add("CC");  
//Set --> List  
List<String> setList = new ArrayList<String>(set);

1. List转变成数组array

//List-->数组  
List<String> list = new ArrayList<String>();  
list.add("AA");  
list.add("BB");  
list.add("CC");  
Object[] objects = list.toArray();//返回Object数组   
System.out.println("objects:"+Arrays.toString(objects));  
String[] arr = new String[list.size()];  
list.toArray(arr);//将转化后的数组放入已经创建好的对象中

1. 数组专为list

//数组-->List   
String[] ss = {"JJ","KK"};  
List<String> list1 = Arrays.*asList*(ss);  
List<String> list2 = Arrays.*asList*("AAA","BBB");