G4学习笔记

ZhangJie

# 1．数据库

## 1.1．G4数据库表说明

1）Code

这个是代码对照表，将其他表中字段的可能取值以及取值表示的意义填写在这里，这样也算是一层分层设计吧，数据库开发人员可以将数据库中的信息提前设计好，程序开发人员则无需关心数据库的设计，如数据库中表中字段sex的值取0时表示什么意思，取1时表示什么意思，程序开发人员只需在查询数据库时，将sex和0在code表中进行一次查询就可以了。

这样做还有一个好处，就是，在不同的表中都有sex字段，不同的数据库设计人员可能设定的取值与取值的意义不同，如果将所有的sex提取到码表中进行定义，则使得数据库的设置更加像一个整体、有条理。

2）dept

这个是部门信息表，其中定义了部门的id、名称、上级部门等信息，上级部门为0的部门表示是公司的最上级部门，也就是公司的总部，上下级部门的id生成遵循这样的关系，如果上级部门的id为001，则直接下级部门的id为001001~001999,也就是每次在后面追加3位数字。

如果部门有下级部门则该部门不是叶节点，如果没有下级部门，则是叶节点。为什么要定义叶子节点leaf这个字段，这主要是考虑到，ext中树控件自动装载数据，生成所有的部门节点。

3）event

这个是事件表，用来记录每一个登录用户的操作事件，这个表真的是非常的重要。

它可以记录每个员工在什么时间请求了什么路径，请求的action操作是什么等。

4）exception

这个是系统异常监控表，可以记录异常类型、方法名、异常信息，它这里说监控系统异常，应该指的是我们这个web系统的异常信息，而不是对容器的监控，因为它可以记录方法名，应该是我们写的java类中的方法名。

没错确实是这样。

5）help

这个是帮助信息表，因为g4系统里面也没有在这个表里面存储什么信息，我也不是很清楚它现在到底存储的内容是用在什么地方，不过它的字段里写着，menuid、content，就是菜单的id以及具体的帮助信息，难道是在不同的页面上显示的帮助菜单吗？**那么帮助内容又是什么呢？**

6）httpsession

这个是http会话资源管理， 当用户登录系统的时候，系统创建一个session与之关联，并将其信息记录在这个表里面，包括登录用户的ip地址、客户端浏览器类型、session创建时间、登录用户的id、用户名等信息。

可能是g4没有实现这个功能，对系统进行了一系列操作之后，在这个表里面并没有生成相应的记录。

**不对，g4确实实现了这个功能，它是这样实现的，当session创建的时候，它会生成记录；当session被销毁的时候，它就将对应的session从这个表中删除。**

**不晓得它为什么要删除这条记录，记录这条记录不是更好吗？将来系统出了什么问题的时候，不仅可以通过event追踪到是谁登录了系统、进行了什么操作，还可以通过httpsession这个表追踪到是谁使用哪个ip地址、在什么时间进行了登录。**

7）icon

这个是图标表，记录了图标的id、图标对应的文件名以及当前这个图标使用的css名称。

8）jdbcmonitor

这个表用来对jdbc进行监控，监控的内容包括在什么时间执行了哪些sql操作，以及sql操作消耗的时间，影响的行的数量，以及数据操作类型（insert、update等）。

**这个表也是很有用的，将来我们可以拿来对sql进行调优。**

9）menu

这个表示系统菜单表，主要字段是菜单id、菜单名称、上级菜单名称、是否是叶子节点（添加一个新的菜单项的时候，该新添加菜单项默认是叶子节点）、是否展开、节点图标icon以及图标的css类型iconcls。

**这个表最后有几个字段width,height,scrollbar，它的注释写的是表示桌面窗口的高度、宽度、有无滚动条，不明白有什么用途？**

10）menupart

这个表是托管ui组件表，托管ui组件的目的是先让系统知道有哪些ui组件需要管理，以方便后期为用户分配不同的权限，让某些用户可以访问该组件、某些用户则不能访问该组件。

要托管的ui组件可以具体到某个页面上的某个元素，这个元素我们用这个元素的id进行标定，这个页面我们用menuid进行标定。每一个菜单项都有一个menuid和一个请求路径，而该请求路径在struts里面根据返回的视图访问到不同的jsp页面，这个jsp页面最终由容器执行并返回的html文件，就是我们要托管的页面，其中的页面元素，就是我们要托管的ui组件。

11）param

这个是全局参数表，主要包括了参数的键名，以及参数的值，我在想，系统在初始加载的时候，应该是读取了这个表，然后将表中的所有内容构成一个hashmap结构，然后以后希望访问全局参数的时候，就不需要再次访问这个表，而是直接读取这个hashmap结构体就可以。应该是这样的，不然的话，数据库中表字段的命名就不会是paramkey\paramvalue了。

12）role

这个是角色表，主要包括了角色的id、角色的名称、角色所属部门的id。这个表应该是与权限有关系的，比如给不同的角色授予不同的访问权限。

13）roleauthorize

这个表示给角色进行授权的表，包含的字段包括角色id、可访问菜单的menuid，以及对该菜单的权限级别authorizelevel，权限级别分为两种：1访问权限和2管理权限。访问权限表示可以访问该菜单项对应的页面，管理权限表示可以对当前菜单项进行删除、修改操作。

14）rolemenupart

这个是ui元素授权表，包含的字段有roleid、menuid、partid、partauthorize，menuid与一个请求路径对应，请求路径与一个视图对应，视图与一个jsp页面对应，也就是说menuid与该对应的jsp页面返回的html页面对应，最终角色与某个html页面中的id属性为partid的某个元素对应，达到元素级权限控制。

**Ui元素授权类型partauthorize，暂时不清楚到底是什么意思，可能与前面一张表中的authorizlevel类似，暂时不确定。**

15）sequence

这个表示序列号生成器表，序列sequence是oracle里使用的一种方式，oracle里面不支持主键自增，当我们希望插入一条记录的时候，如果希望达到主键自增的目的，可以使用序列来实现，例如create sequence sss minvalue 1 nomaxvalue start with 1 increnment by 1;然后插入表insert into tablename values(sss.nextval,….)这样就实现了主键的自增。

至于g4里面为什么使用这种方式来实现序列号呢，我想它肯定是考虑到了在不同的数据库产品之间的差异，例如mysql可以将主键设置成int类型，并使之成为auto increment类型就可以了，但是oracle就不支持这种功能，前面我们也提到了需要使用序列来辅助实现，至于其他的数据库产品，可能也存在类似的问题。

其实g4这样设计挺好的，我们可以根据自己的需要来生成序列号，而不仅仅是用作主键自增的目的，例如dept表中，有一个部门id为001，其子部门的id就可以这样生成001001~001999，这样，如果我们希望查询001的下级部门时就可以使用001…来查询出直接下属部门，而对001999来说，如果希望查出其上级部门则也只需查询部门id等于001的即可。

这种设计sequence的方式，非常值得借鉴。

这个表中主要的字段包括，fieldname，这个字段其实就是我们希望生成的某个表的主键的字段名称，maxid，表示该字段对应的主键的当前最大取值，下一次再次生成该主键的值时应该将maxid+1了，pattern表示要生成的主键的长度，因为在生成主键的时候，还是在maxid的基础上进行了某些数学上的操作，最终要格式化成与pattern一样长度的形式。Idtype表示id类型，分为1系统内置和2用户自定义两种类型。**我理解的，这里的系统内置类型就是mysql、oracle等数据库产品里面的内置类型吧。**

16）user

这个是用户信息表，包含了用户的用户名、**登录账户名**、密码、性别、所属部门id等信息。

17）userauthorize

用户授权表，包含的主要字段是userid、roleid，表示在某个用户身上添加某个角色所具有的权限，某个角色可以访问的菜单、托管的ui元素，具有该角色的用户也可以访问。

18）usermenumap

用户菜单映射表，包含的主要字段是userid、menuid、authorizelevel，表示某个用户可以访问的所有菜单项，以及对应的权限级别。

前面一个表，表示用户可以通过角色信息获取某些访问权限，但是仅靠前面这张表的话，可能访问还是不能达到精细的程度，例如用户a获得了b角色，那么a可以访问b角色可以访问的所有菜单项以及托管的ui元素。如果此时我想给a添加一个b角色所不具有的访问权限时，例如希望a可以访问另一个b不可以访问的菜单项，并且我不希望新建角色c来实现这种控制，那么我就可以再直接给a授权访问菜单项。

**我的理解就是，用户的全部访问权限，是用户所具有的角色的权限，与系统直接授予用户的权限的并集。**

19）usermenupart

托管的ui元素的人员授权表，这里这个表示将对菜单项menuid对一个的页面中的id属性为partid的托管元素的访问权限，授予用户userid，并且权限级别为partauthorize。

至于为什么需要直接给用户授予对托管元素的访问权限，介绍前面一张表的时候，已经介绍过了。

20）usersubinfo

用户附加信息表，我在想为什么使用user-sub-info这个名字，sub通常具有子……的意思，难道作者是想用“用户信息的子信息”来表示用户附加信息，不过这样想的话，也挺好的。

## 1.2. 数据库存在的问题

1）数据库表中，本来很多可以使用int类型的字段，使用了很多的varchar类型，例如code.enabled，就是使用了varchar(2)类型，使用varchar还好了，但是如果使用char的话就坏了，数据库会将0自动填充为“ 0”，将1自动填充为“ 1”，总之如果长度达不到char(n)，那么值的前面就会自动填充空格，这就很糟糕。

其实有些地方可以使用int的地方，为什么不使用呢？我现在觉得有两个方面需要考虑：

这个字段是否会频繁地参与运算？如果经常参与算术运算，那么能用int还是int，就不要手动转类型，例如jdbc中getInteger就取出int字段，不需要Integer.parseInt(String…)将varchar字段转成Integer类型；

这个字段是否是常常用作比较，不常用作算术运算？如果是的话，就比如g4里面code.enabled只是常常用于比较，那么就可以使用varchar，因为不涉及到频繁的操作，而且这样还节省对空间的占用。我想的是，内存中使用4字节来保存一个int数，在实际物理存储的时候也是这样的吧，不然读出来读到内存中的时候，不就错了嘛。

2）没有外键约束，有些字段很明显，都存在外键约束，但是作者好像有意回避了这一点一样，例如dept表中的parentid。加不加外键，看自己吧，其实我觉得加上要好一点。

3）……

# 2. g4引用的jar文件

## 2.1. JRE System Library

1）resources.jar：资源包（图片、properties文件）；

2）rt.jar：java运行时包；

3）jsse.jar：java secure socket extension；

4）jce.jar：java cryptography extension；

5）charsets.jar：字符集

6）dnsns.jar：dns相关；

7）localedata.jar：本地化数据；

8）sunjce\_provider.jar：

## 2.2. Java EE Libraries

Javaee.jar：java enterprise edition；

Jsf-impl.jar：

Jsf-api.jar：

Jstl-1.2.jar：Java Standard Tag Libraries；

注：

1. JavaServer Faces，新一代的Java Web应用技术标准，吸收了很多Java Servlet、JavaServer Pages（JSP）以及其他的Web应用框架的特性。JSF为Web应用开发定义了一个事件驱动的、基于组件的模型；
2. JSF到底是什么？

## 2.3. Referenced Libraries

abdera-core-1.0.jar

abdera-extensions-json-1.0.jar

abdera-extensions-main-1.0.jar

abdera-i18n-1.0.jar：国际化相关；

abdera-parser-1.0.jar

ant-1.6.5.jar：ant编译工具的jar包；

antlr-2.7.2.jar：语言转换工具，hibernate利用它来实现**hql**到sql的转换，其中hql是hibernate query language的意思，g4中包含了这个包，**难道是ibatis也是利用了这个工具进行转换的吗**；

aopalliance-1.0.jar

asm-2.2.2.jar：asm字节转换库；

asm-commons-2.2.2.jar：apache提供的字节码转换库？

asm-util-2.2.2.jar：字节码转换工具？

aspectjrt.jar：

aspectjweaver.jar：

avalon-logkit-2.1.jar：

axiom-api-1.2.7.jar：

axiom-impl-1.2.7.jar：

bcprov-jdk15-1.43.jar：

bsf-2.3.0.jar：

bsh-2.0b4.jar：

cglib-nodep-2.1\_3.jar：字节码解释工具库？

classes12.jar：操纵oracle数据库；

commons-beanutils-1.7.0.jar：apache commons包中的一个，包含了一些bean工具类；

commons-chain-1.1.jar：

commons-codec-1.3.jar

commons-collections.jar：apache commons包中的一个，包含了一些apache开发的一些集合类，功能比java.util.\*强大，必须使用的jar包；

commons-dbcp.jar：apache Jakarta组织开发的数据库连接池；

commons-digester-1.6.jar

commons-fileupload-1.1.1.jar：文件上传组件？

commons-io-1.1.jar：io组件？

commons-lang-2.3.jar：apache包中的一个，包含了一些数据类型工具类，是java.lang.\*的扩展；

commons-lang3-3.1.jar：同上；

commons-logging-1.0.4.jar：apache包中的一个，提供了日志功能。Apache提供的这个包中的日志功能很弱。对日志记录来说，能用log4j就用log4j，不能用就用jdk自带的logging，再不行，就用apache提供的这个simple logger。

commons-pool.jar：apache jakatar提供的数据库连接池，是不是与commons-dbgp配合使用？

commons-validator-1.3.0.jar：验证器？

cxf-2.2.10.jar

cxf-manifest.jar

cxf-xjc-boolean-2.2.10.jar

cxf-xjc-bug671-2.2.10.jar

cxf-xjc-dv-2.2.10.jar

cxf-xjc-ts-2.2.10.jar

dom4j-1.6.1.jar：dom for java，提供读写xml文件的api；

ehcache-1.3.0.jar

ezmorph-1.0.6.jar

FastInfoset-1.2.7.jar

fdsapi-1.2.jar

freemarker-2.3.19.jar

groovy-all-1.5.5.jar

iText-2.1.0.jar

iTextAsian.jar

jakarta-oro-2.0.8.jar

jasperreports-3.7.0.jar

jasperreports-fonts-3.7.0.jar

javassist-3.3.jar

jaxb-api-2.1.jar

jaxb-impl-2.1.13.jar

jaxb-xjc-2.1.13.jar

jaxen-1.1.1.jar

jdt-compiler-3.1.1.jar

jra-1.0-alpha-4.jar

jruby.jar

js-1.7R1.jar

jsf-api.jar

jsf-impl.jar

json-lib-2.3-jdk15.jar

jsr311-api-1.0.jar

jstl-1.2.jar

jta.jar

junit-4.4.jar

jxl.jar

log4j-1.2.14.jar

mybatis-2.3.5.jar

mysql-connector-java-5.0.5.jar

neethi-2.0.4.jar

ognl-3.0.5.jar：

oro-2.0.8.jar

oscache-2.0.1.jar

png-encoder-1.5.jar

poi-3.0.1.jar

saaj-api-1.3.jar

saaj-impl-1.3.2.jar

serializer-2.7.1.jar

slf4j-api-1.5.8.jar

slf4j-jdk14-1.5.8.jar

spring-webmvc.jar：spring mvc相关；

spring.jar：spring相关；

struts2-core-2.3.4.1.jar：struts核心；

velocity-1.6.2-dep.jar：

velocity-1.6.2.jar

wsdl4j-1.6.2.jar

wss4j-1.5.8.jar

wstx-asl-3.2.9.jar

xalan-2.7.1.jar

xml-resolver-1.2.jar

xmlbeans-2.4.0.jar

XmlSchema-1.4.5.jar

xmlsec-1.4.3.jar

xwork-core-2.3.4.1.jar：xwork核心，struts2是在xwork基础上建立起来，g4以前使用的是struts1，但是不晓得它是不是使用了当时的xwork，我们现在使用的是struts2，**不知道这个xwork还有没有用处；**

其中有些包的功能以及用途不是很清楚，需要了解一下，这样才能明白系统的各个部分使用了些什么，如何配合起来工作的。

# 3. SSH/SSI框架

## 3.1. Struts框架

Struts是一个技术框架，是由Craig R. Mclanahan编写，并在2000年捐献给了ASF，由很多组织和个人参与struts的开发，利用它开发的应用也越来越多，使其成为MVC开发中VC上的事实上的标准。

Jsp开发过程中，有人提出了model1和model2两种开发方式，model2开始显现出mvc的思想。Struts就是一种实现mvc的框架，并且侧重点主要在vc上。

### 3.1.1. Struts 1.0

### 3.1.2 Xwork

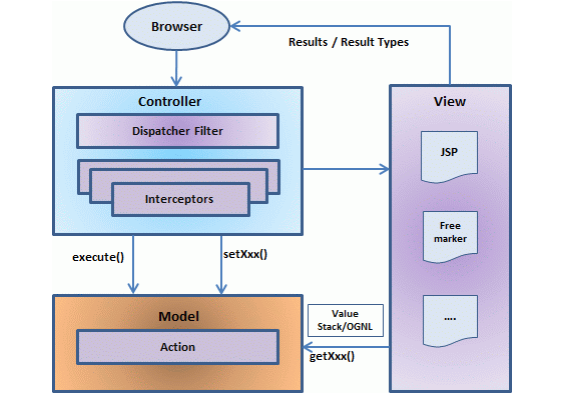
### 3.1.3 Struts 2.0

Struts2，与其说是struts1的新版本，不如说是xwork的新版本。

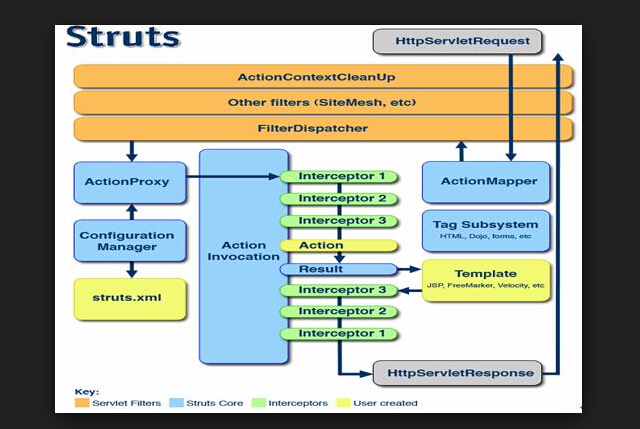
**与传统的mvc框架不同的是，struts2的模型层是在action中实现的，而它的控制器则是通过一系列的过滤器和拦截器实现的。这一点要明确，不然要闹笑话。**

1. struts2架构

actions、interceptors、value stack和ognl、results和result types、view technologies。



图x struts2架构



图xx 此图更经典

1. 请求生命周期

* 用户通过浏览器发送请求到web服务器，请求某种资源，例如页面或者某个动作等。

详细过程：

浏览器发送一个url请求到web服务器，假定这里的web服务器是tomcat容器。请求到达tomcat容器之后，tomcat检查web.xml，发现其中通过过滤器filter定义了struts2的入口，此时，如果请求的url与指定的url-pattern匹配，则将请求转发给struts2处理；

* Struts2根据struts.xml找到与当前请求对应的action类。

详细过程：

Struts2里面通过请求分发过滤器FilterDispatcher，根据struts.xml中的配置的action与目标action类的映射关系，找到处理当前请求的action类，以及与这个action类相关联的一系列的拦截器Interceptor（实现接口Interceptor而非Intercepter，注意），将action类对象与一些列拦截器封装到ActionInvacation对象中；

* 在调用action方法前先执行一系列的拦截器进行前期处理。

详细过程：

应用一系列的拦截器对请求进行处理。拦截器包含需要包含3个方法init\destroy\Interceptor。Interceptor方法接收ActionInvation arg对象，其中包含了拦截器栈以及目标action对象，通过调用arg.invoke()调用下一个拦截器，如果再没有拦截器，则调用目标action方法，默认为execute；initialize方法在容器初始化的时候执行一次，通常用于申请某些资源；destroy在容器关闭的时候执行一次，用于释放资源。

在一个拦截器里面，可以将代码分成3段，

Codeblock1

Arg.invoke();

Codeblock2

假定现在有3个拦截器，那么执行顺序大致如下：

拦截器1 拦截器2 拦截器3 Action类

Codeblock1

**Res1=Arg.invoke()**

🡪 codeblock1

**Res2=Arg.invoke()**

🡪 codeblock1

**Res3=Arg.invoke()**

🡪 **execute() {**

**return RES4；**

**}**

🡨

Codeblock2

return **RES3**

🡨

Codeblock2

return **RES2**

🡨

Codeblock2

return **RES1**

* 执行action类中的特定方法。

详细过程：

Action类的默认方法是public String execute()；

action方法在执行之前，通常先执行一些列的拦截器进行处理，假如某个拦截器在执行过程中出错了，那么应该返回一个结果字符串，返回对应的处理视图。如果是中间某个拦截器出错返回的话，拦截器栈某些拦截器的处理操作就不会被执行了、action方法可能也不会被执行了（要看拦截器是在invoke前返回还是invoke之后返回）。

1. Workflow拦截器：

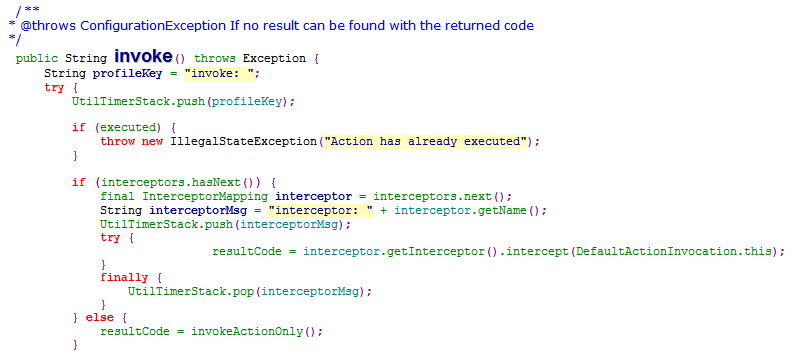
假如action类中实现了Validatable接口，需要override函数validate()，这个函数可以对用户提交的表单数据进行校验。需要注意的是，这个函数并不是在execute()或者action类的某个其他方法中被调用，它是在拦截器workflow中被调用的。

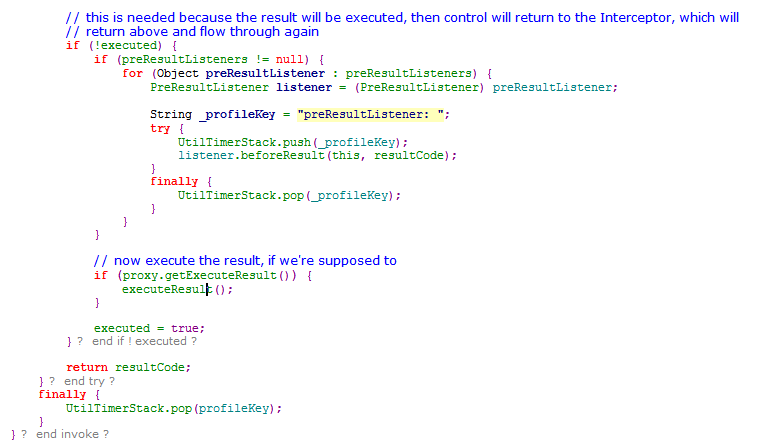
拦截器workflow会首先检查目标action有没有实现Validatable接口，如果实现了的话，就会调用这个类的validate()方法，这个方法中对用户输入进行校验，如果存在某些错误，则会通过方法addFieldError(表单元素名称，错误提示msg)记录出现的错误，这些错误都被记录在ValidationAware接口中。拦截器workflow执行完validate方法后，会检查是否有错误被记录了下来，如果有，则返回”input”，表示需要用户继续进行输入，此时，目标action中的execute方法就不会被执行了。只有validate验证没有错误的时候，才会继续执行execute方法。

另外需要注意，在拦截器栈中的拦截器，其返回值不是一致的。

**当目标action类返回时，立即转向对应的视图，并不是拦截器栈中的所有拦截器都执行完成之后，才转向对应的视图。而且，转向指定的视图至于action的返回结果有关，与其他拦截器的返回结果没有半点关系；**

**查看DefaultActionInvocation的源代码，主要看invoke方法：**





**图xxx invoke方法**

**另外，如果是中间的某个拦截器没有调用invoke就返回了，此时由于action类的方法没有执行，所以不会转向对应的视图，此时就要求在拦截器中进行页面重定向控制。**

**Action中的方法返回后，即开始调用executeResult方法，寻找对应的视图，例如jsp、模板freemarker文件等。然后再回到拦截器，拦截器中可能会对这些文件进行某些修改或者渲染，最后拦截器全部结束，视图层形成，返回给用户。**

1. Params拦截器

这个拦截器用于完成对目标action的属性注入，注意，是将请求表单中的数据填充到目标action的成员中去，是属性注入，还称不上依赖注入DI。

1. Double-Commit Guard拦截器

Struts2里面是通过令牌实现的，可以防止同一个表单多次提交。

但是有这种需求：

**同一个表单会被使用多次，每次可能提交一个或几个文档，用户可能使用这个表单多次，这个时候，struts2中的TokenInterceptor就不好用，因为表单中tokenid有效的话，它就会将它从session中删除，使其不再有效。表单使用一次的话没有问题，但是要使用多次的话，就有问题了。**

**再重点看下TokenInterceptor与TokenSessionStoreInterceptor这两个实现，可能理解有误。**

* 应用一些列的拦截器进行后续处理，并将action返回的结果向上层层层递送。
* 根据action返回的结果，转向对应的视图层，渲染完毕后，将结果页面发送给用户。

详细过程：

现在常用的视图层技术包括JSP、Velocity、FreeMarker，以JSP为例，在渲染的时候，可能需要通过访问值栈ValueStack完成JSTL的处理。

* 配置文件

Struts中比较重要的配置文件有struts.xml、struts-config.xml、struts.properties、default.properties，此外web.xml也息息相关。

我们在web.xml中定义好一个struts的过滤器，将某些或全部的url请求交给struts处理，因此也说，我们在web.xml中定义struts的入口；

Struts.xml中一般定义一些package（package的引入使得复用非常方便），package里面可以定义interceptors，interceptors里面定义interceptor，action可以引用前面定义的interceptor（注意，拦截器要在引用它的action前面定义）、action中定义result以及对应的view等。这里的view可以是jsp页面，也可以是freemarkder模板等。

1. tomcat+struts2后对url的处理方式

假如一个站点是app当访问http://localhost:8080/app的时候，tomcat就先扔出welcomefilelist中找到的页面作为欢迎页面。假如是index.html当这样访问的时候http://localhost:8080/app/index.html，tomcat先检查app/后面的这个字符串是否是welcomelistfile中的某个文件，如果是则扔出这个页面，如果不是，再丢给struts过滤器处理。

其实还有这种情况，当我这样访问http://localhost:8080/pagename.jsp，如果pagename.jsp这个页面存在的话，则直接扔出这个页面，不存在的话，才会丢给struts。

刚开始访问站点的时候，如果欢迎文件找不到也会报错。

4）请求一个action类中的多个方法

对于struts.xml中对请求url以及action的映射，action类中默认将public String execute()当作要调用的方法，因此可以在action类里面写好这个方法，然后默认调用这个方法。我的意思是，如果请求action，并且只能映射到action类，没有指名方法的话，那么就调用execute方法。

一个action类里面可以定义多个方法，也可以分别将这多个方法，作为多个action来请求，在前台可以通过这种方式来访问action类中的方法，“actionname!methodname”，或者通过“actionname.methodname”，想这么使用的一个前提是，必须先对struts2进行必要的设置，在struts.xml中定义常量，<constant name=*"struts.enable.DynamicMethodInvocation"* value=*"true"* />，这样就激活了对action的动态方法调用，而不是只能默认访问execute。

但是需要注意的是，不管是使用“actionname!methodname”还是“actionname.methodname”，都会将其当做“actionname!methodname”进行处理。

5）派遣过滤器指的就是struts根据struts.xml中的配置，决定由哪一个action进行处理。

6）struts2属性注入，要求某个页面中包含一个表单，表单中各个元素的名字，与待提交到的action类中的成员名字必须一致，且待注入的元素，必须提供对应的getter、setter方法。

7）validate

action继承自ActionSupport，重写valiate（）方法用于校验，addFieldError记录校验错误，如果有字段错误，则执行完validate()后，struts自动调用return INPUT,反之调用execute()方法。如果出错返回input之后，应该提示出错信息，struts2会自动在原有表单对应元素的前面添加错误字符串。

其实这个原理比较简单，就是在不合法的表单输入元素的前面插入一个新的元素，在这个元素中显示出错信息，定位输入不合法元素的时候是通过元素的名称定位的。这个可以在浏览器端查看源代码加以了解。

8）\*.properties文件

资源绑定ResourceBundle，这是非常有用的。

对于一些静态文件，例如用户没有输入密码，需要在validate方法里面指定一些错误消息，通过方法addFieldError完成，这个时候，如果是将错误提示信息写在源代码里面的话，就显得有些臃肿，而且如果需要多次提醒的话，不仅充斥大量字符串，还可能引起不一致的提示信息。我们将这些文本字符串放进一个属性文件中，\*\*\*.properties，注意\*\*\*部分必须

与action类的名字一致，然后在action类里面就可以调用getText来获取属性对应的值，属性、值对在属性文件中按照“属性名=值”这种形式进行定义，一行定义一个属性、值对。

9）拦截器实现的接口是Interceptor（拦截器）而不是Intercpter（截击机）。

10）在struts.xml中定义拦截器的时候，因为action里面要通过interceptor-ref引用定义的拦截器，xml在解析的时候从前往后解析，因此拦截器必须要定义在action定义的前面，否则会报错。

12）spring和struts2都可以实现注入，异同点

（1）相同点：

目标待装配对象，必须将待赋值的成员设置为private类型，并设置public类型的getter、setter方法。

（2）异同点：

spring利用applicationContext中定义的bean实例化目标对象，并利用设置的property，将待装配对象的成员完成赋值；

struts2，假如没有spring来实例化目标对象的话，struts2会自己完成实例化，并调用getter、setter方法设置目标对象的成员的值。

需要注意的是，struts2里面的这种设置，只是属性注入而已，还称不上是依赖注入。

struts2里面的这种注入是在params拦截器里面实现的，只是为了将表单中元素的值当做action对象的成员的值传入到action对象里面方便action后续处理，而spring是为了依赖注入，与struts2的这种表单属性注入action对象的成员，在性质上完全不相同。

spring的依赖注入是非常强大的，假如a依赖b，b依赖c，c依赖d，现在我希望获得一个对象a的实例，struts2是应付表单的，对这种工作完全不能处理，这里a的实例化还是要spring来做，spring根据配置文件中写明的bean abean、bbean……之间的依赖关系，调用其getter、setter方法完成目标对象a实例的装配。

getter、setter方法，其中的setter方法在spring完成注入、struts完成表单属性赋值的时候使用，在渲染的时候，或者在其他类中可能会用到setter，例如，在jsp页面里面通过jsp标准标签库引用valuestack中存储的action对象的成员的值的时候，就会调用getter了，如果没有定义getter，不会报错，但是渲染出的文本会显示为“空”。

视图层的渲染，一般是action返回的时候，就执行了，可以参见上面的invoke的源代码以及executeResult源代码（该文中没有给出，请自行查阅struts2源代码）。拦截器一般不会对视图层进行处理。

## 3.2. Spring框架

### 3.2.1. Spring框架版本命名方式

x.y.z.o：x表示project version，这个可能好几年都不变；y表示major version，每隔一到两年会出一个新的major version，z表示修订版本，一般是在major version上改正了一些bug，并没有特别巨大的改变；o这个地方，通常使用的是Beta、RC、RELEASE等，建议查看一下软件版本的约定方式。

一般软件版本的命名方式有：alpha，内部测试版本；Beta，公开测试版本；Gamma，候选发行版本，现在也称为RC，Release candidate；RELEASE，发行版本。

### 3.2.2. Spring版本变化

Spring在3.0版本之前，是所有的东西都在一个jar包里面，自3.0之后，实现了模块化，各个模块独立地用一个jar包，各个jar包之间可能存在依赖关系。用户需要根据自己的需要选择jar包以及依赖。

### 3.2.3. Spring jar包依赖

1）以Spring框架4.0.5版本为例，Spring-core核心包，必须包含，spring-context依赖spring-expression、spring-bean和spring-core，同时还依赖日志包commons-logging。

其中当调用FileSystemXmlApplicationContext的时候依赖的是spring-context，调用ctx.getBean的时候依赖的是spring-bean，在解析xml文件的时候，还需要依赖spring-expression包，其中的日志输出依赖的是commons-logging。

2）

### 3.2.4. Spring知识点

Spring可以用在任何类型的工程上，以一个普通的java工程为例，我们新建一个java工程，然后希望给它添加spring框架支持，这个时候我们需要做的是：

1. 设置build path，添加所需要的jar包到build path中去；
2. 编写bean.xml配置文件，在其中定义我们所需要的bean，需要注意的是，由于spring不同版本的潜在问题，在对xml配置文件进行解析的时候，对xml文件内容格式的要求可能不同，例如在3.0之前的版本中可能按如下方式下就不会存在问题，但是在4.0的版本中可能就会出现问题，比如说找不到beans的定义。

<beans>

<bean id=”…” class=”…>

<property name=”…”>

<value>….</value>

</property>

</bean>

</beans>

对于这种问题，要尽可能规范，网上有网友提供了一种方案，测试后发现是可用的：

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

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" “http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

…

</beans>

1. 然后我们可以在程序里面通过ApplicationContext ctx = new FileSystemXmlApplicationContext(configFilePath)来获取一个上下文，然后ctx.getBean(id)来获取所需要的类的实例。

这一步需要注意的是，必须在待实例化的类里面设置各个属性的正确的getter、setter方法。Spring利用Java的反射机制检测getter的返回值，然后将xml文件中对应bean的property的value转成这种类型，然后调用setter方法，将值传递给实例化出来的对象，也就是getter的返回值与setter的参数必须一致。

另外需要注意java的访问修饰符，public,default,protected,private，其可见范围越来越小，public修饰的可以跨类访问、跨包访问，default修饰的可以在同一个包中进行访问，protected修饰的只允许在当前类及其子类中进行访问，即使子类在不同的包中也是可以访问的，private只能在当前类中进行访问。

由于我们希望spring希望调用类的getter、setter方法完成对类的装配，而我们调用的spring的api很明显与我们自己写的包含getter、setter的类不在同一个包、同一个类中，也不是子类关系，所以我们必须将getter、setter方法设置成为public类型，否则会报getter、setter类型错误。

1. spring自动对类进行装配的优势：

客户代码可以只面向接口编程，无需关心底层实现类的名称以及实现细节；底层实现类的开发人员，根据约定的接口进行实现，实现的版本可能有多个，实现方式可能由多种，可以通过修改spring配置文件，轻松的在各个实现类之间进行切换。

1. spring使组件之间的依赖关系减少，极大地改善了代码的可重用性。
2. spring使得面向接口的编程更加自然。
3. **spring是一个从实际项目开发经验中抽取的、可以高度重用的应用框架。**
4. IOC，inverse of control，控制反转；DI，dependence injection，依赖注入。

准确的说，IoC和DI并不相同，这一点从字面上就可以看出，否则，它们可以叫一个名字。^\_^

理解IoC，我们需要知道Control是什么，它又是怎样被Inversion的。其实，IoC是用来说明“程序库”和“框架”区别的最好证据。在使用程序库的时候，控制权是掌握在我们手中的，我们编写的代码调用程序库的实现，完成相应的功能，想想我们使用JDK的情况。使用框架的时候，控制权则掌握在框架手中，我们的代码最终是由框架调用，一个常见的例子是Servlet，我们编写的Servlet代码是放在整个Servlet的框架中，由Web容器进行调用。这就是差异所在。我们更习惯于自己掌控一切，因此，对框架掌握控制权的这种情况，我们用“Inversion”来形容，这也是[Martin Fowler](http://www.martinfowler.com/)在那篇[给DI正名的文章](http://www.martinfowler.com/articles/injection.html)中提到，所有框架都是IoC的原因。

Spring的核心容器是一个框架，所以，我们可以说它是IoC，但是就如前面所说，每个框架都有IoC，所以，仅仅用IoC是不足以说明一切的。Spring核心容器完成的是组件组装的过程，这是它和其它普通框架区别最为显著的地方。如果说用IoC描述这个框架，那么，这里所指的Control实际上是组件的组装过程。

站在Spring核心容器的层面上看，它完成组装过程是把组件所依赖的零部件给组件安装上去。站在单个组件层面上看，它所需要的零部件是由外部给它安装的，这个过程就像是把“Dependency”这管药水用注射器“Injection”到组件的身体中去，所以，我们称之为“Dependency Injection”。

完成组件组装的容器也不只是注入一种形式，还有一种常见的方式是“Dependency Lookup”，即每个组件自己去查找自己所需要的内容。至于到哪去找，也有不同的实现方式，有固定到某个地方（比如使用静态方法），有把查找点通过DI的方式注入进来等等。

1. 依赖注入，何用之有？依赖注入并不是给软件系统带了更多的功能，而是提高了组件的可重用的概率，并未系统搭建了一个灵活、可扩展的平台。

例如一个对象a，它依赖一个对象b，如果没有依赖注入，那么我们需要new b，然后new a，再将b设置到a中的某个属性上去。如果有依赖注入，我们通过在bean.xml中进行设置，描述出a依赖b的信息，如下所示：

<bean id=”a” class=”AClass”>

<property name=”b”>

<rel local=”b”>

</property>

</bean>

<bean id=”b” …>

…

</bean>

这样当我们对通过ctx.getBean(“a”)获取a的实例时，spring会通过依赖注入，自动的将b装配到a中返回给我们，看，这样是不是特别方便！

10）DI（依赖注入）的实现方式包括：接口注入、构造注入、设置注入，下面分别进行阐述。

* 接口注入

比如一个类a依赖一个接口b，b有多个实现b1，b2……，在类a的初始化方法里面完成对接口b的赋值，看如下代码：

Class a {

Interface b;

Public void init() {

Class bimpl = class.forName(Config.b).newInstance();

This.b = bimpl;

}

}

我们仍然是在配置文件里面写入InterfaceName=ImplClassName，然后通过Properties Config = new Properties()；然后Config.load(new FileInputStream(“配置文件”))来装载配置数据，然后通过Config.b来获取接口b的实现类的名称。

* 构造注入

构造注入很容易理解，就是给一个类的构造函数传递参数，在构造函数里面完成对依赖的设置。看如下代码：

Class a {

Class b;

a(class b) {

this.b = b;

}

}

* 设置注入

设置注入就是我们的spring的这种方式，通过调用类的getter、setter方法来完成对依赖的设置。相比于前两者，设置注入更加灵活，便于实现组件的自动装配。

其实spring对构造注入、设置注入都支持，但是设置注入使用更加广泛。

举个例子有个接口Action，它有两个实现类LoginAction，LogoutAction，这个时候如果调用ctx.getBean(Action.class)获取bean实例的时候，会报错，因为有两个实现类，不确定使用哪一个，在bean.xml中进行配置可以解决这个问题。所以我们看到，spring会自动去寻找接口的实现类，如果只有一个实现类的话，甚至可以不用在bean.xml中写明，但是我们希望更加清晰的管理bean的生成，所以应该在bean.xml中写清楚，尽量使用ctx.getBean(String classname)来调用而不是ctx.getBean(Class classname)来调用。

Spring实现DI的两个核心组件是bean wrapper和bean factory。

Bean wrapper实现了针对单个bean实例的装配，beanfactory则是针对多个bean进行管理的一个容器。

ApplicationContext则是覆盖了beanfactory的所有功能，beanfactory更像是一个api调用，applicationContext才更像是一个框架。

在配置文件中对bean进行配置时，可以配置的选项有：bean属性值及其依赖关系、bean创建模式（是否为singleton模式）、bean初始化方法和销毁方法、bean的依赖关系。

例如：

<bean

Id=””,

Class=””,

Singleton=””,

Init-method=””,

Destroy-method=””,

Depends-on=””

>

<property name=””>

<value>…</value>

</property>

<property name=””>

<rel local=””/>

</property>

<property name=””>

<null/>

</property>

</bean>

1. ApplicationContext

它可以对如下信息进行配置：国际化支持、资源访问、事件传播、多实例加载。

Spring事件处理相关依赖包spring-aop。

1. 获取applicationcontext对象

前面我们已经提到过applicationContext对象的强大，如何获取这个对象呢，我们可以通过在程序中手动加载配置文件来获取这样一个对象，例如new FileSystemXMLApplicationContext(path)，但是这样的话就会生成一个冗余的applicationcontext对象，为什么说是冗余的呢？因为在spring初始化的时候，就已经实例化了一个applicationContext对象，我们可以直接使用这个对象，而无需再次生成该对象。首先需要了解的是，我们可以使一个类实现接口ApplicationContextAware，如果实现了这个接口，那么spring在初始化的时候，就会通过DI自动将spring生成的这个对象注入到我们的类中，所以呢，我们可以写一个Helper类，例如AppContextHelper，然后在这个类里面，我们将ApplicationContext对象定义为一个静态变量，并添加getter、setter方法，这样以后需要获取applicationcontext对象的时候，就可以从这个帮助类中直接获取到。

1. Web Context

对于web content来说，对于applicationContext对象的获取，spring提供了两种可配置的加载机制，ContextLoaderListener和ContextLoaderServlet。

13）事件传播

Spring使用的事件传播机制是观察者模式。观察者模式，可以简单地理解为这样一种模式，有多个类同时依赖于同一个类a，当a的状态改变时，需要通知依赖于它的所有类，这些类收到来自a的通知后，就可以更新自己的状态信息。

Spring中的事件传播，由applicationContext来负责完成，调用publishEvent方法将事件传播出去，然后这些事件会被系统中所有的监听器捕获，并进行适当的处理。实际上我们可以为一个特定的事件写一个特定的监听器，这个特定的监听器可能会接收到很多的事件，但是只处理其中的某一个特定的事件。

监听器的bean信息也需要在bean.xml中写明，然后spring在初始化的时候，会实例化所有的对象，然后监听器就被启动了，我们无需通过ctx.getBean(监听器)获取监听器对象并启动它，spring会负责完成这些操作，我们只需要实现监听器的所需功能，并在bean.xml这个配置文件中写明就可了。

1. 表现层技术

* JSP

Jsp代码中往往充斥着视图数据、业务逻辑、数据处理，代码比较混乱，不利于工程的维护。

* XSLT模板

这个模板较之以前得到了大力的发展，其可移植性是现有模板技术中最高的，比freemarker和velocity强多了，但是它的性能表现却不尽人意。Freemarker和velocity对于同等复杂度的表现层逻辑，平均处理速度是xslt的10倍，另外xslt对内存的占用率也是freemarker和velocity的数倍，这是一个不可忽略的巨大差异。另外，由于xslt中将每一个节点都用一个对象来表示，对象的频繁创建、销毁，也会对jvm的垃圾回收产生了很大的负面影响。

在项目中引入大量xslt技术，需要认真的考量，不建议使用。

* Freemarker模板

Freemarker是除velocity之外另一个成熟的模板技术，与velocity相比，它对表现层、业务逻辑层的分离更加彻底、更加严格。例如在freemarker中不能够直接调用servlet api，这样就强制将业务逻辑转移到业务逻辑层，而不是混杂在表现层。此外，它还支持jsp tag，这是velocity所不能的。

有些人建议使用freemarker，可以达到最大收益。

* Velocity模板

该模板技术由apache Jakarta小组开发，是比较成熟的一个模板技术，并且得到了非常广泛的应用。而且该模板技术不仅局限于web开发中的表现层，它可以被用于其它的代码生成系统中。例如，hibernate中的代码生成工具。

模板技术对于表现层表现效果的帮助其实未必突出，它的主要目的是强制程序开发人员将表型层、逻辑层代码相分离。

有些人建议，还是采用以JSTL Core为基础的，以及有限的、与业务逻辑代码无关的自定义Tag（或第三方Taglib）的jsp页面作为表现层解决方案，这样的话，可能会以最低的学习、开发成本，达到最高的生产效率。

1. spring对持久层的贡献

spring-dao包中对jdbc进行了封装，利用JdbcTemplate可以大大简化数据库操作代码的数量。即使是没有spring框架的支持下，spring-dao包也是可以单独使用的。

15）

## 3.3. Hibernate框架

Hibernate不需要自己写sql，对数据库表看做是一个对象，对数据库的CRUD操作，都转化成对对象的操作，这使程序员更加专注于面向对象程序设计。

## 3.4. IBatis框架

Ibatis是一个半自动的框架，需要自己书写sql。

1. 一个非常显著的区别是，ibatis强调sql的使用，而其他的持久化框架，则通常使用一种自定义的查询语言，通过自定义查询语言与sql的转换，来实现对数据库的操作，例如hibernate就是使用HQL查询语言来完成数据库相关的操作。
2. Ibatis的设计哲学

简单、快速开发、兼容多种编程语言、独立的接口。

1. Ibatis的优势

支持存储过程，ibatis可以以存储过程的方式封装sql语句，使得业务逻辑可以“脱离”数据库，应用程序也更容易测试和部署，兼容性也更好；

支持内联sql；

支持动态sql，这一点是ibatis的一个巨大的优势；

支持O/RM；

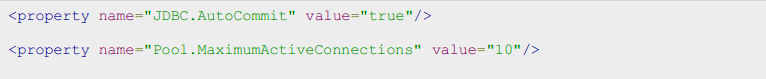
1. Ibatis的安装配置

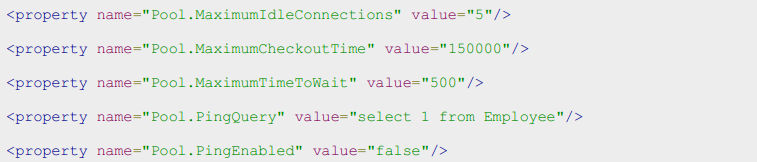
注意对sqlMapConfig.xml的配置，下面是一个配置的实例：



图x sqlMapConfig.xml

还有其他的可选配置参数，例如：





图x sqlMapConfig.xml的其他配置参数

1. Ibatis的CRUD操作

C：Create，对应于sql的insert操作；

R：Read，对应于sql的select操作；

U：Update，对应于sql的update操作；

D：Delete，对应于sql的delete操作；

Ibatis，感觉上可以覆盖hibernate的功能，hibernate在关系数据库表和对象之间进行映射，ibatis通常则在关系数据库表和pojo之间进行映射，例如一个map，其实ibatis也可以实现hibernate那样的映射关系，例如select操作的时候，可以借助于resultMap。

## 3.5. MyBatis框架

MyBatis框架是在IBatis框架基础上进行的升级。

**apache ibatis is retired at the apache software foundation (2010/06/16), the original project team has moved to mybatis hosted at google code. see http://www.mybatis.org/ for more.**

## 3.6. 框架整合

这里的框架整合，以Struts2、Spring、MyBatis为例。

整合过程：

1）添加struts2

首先将struts2所需要的jar文件加入，设置buildpath；

在web.xml中指定struts2入口点，配置struts.xml；

2) 添加spring3

将spring所需要的jar文件加入（最好加struts2解压后里面自带的，兼容性应该要好一点），配置bean.xml（默认使用applicationContext.xml）（别忘了将struts-spring-plugins.jar加入进去，整合用的）

注意，添加struts2、spring的lib文件到webcontent->lib目录下，struts.xml放置在java resource->src目录下，applicationContext.xml与web.xml都放置在webcontent->WEB-INF目录下面。

1. 整合mybatis

拷贝对应的库，设定sqlmap文件。

# 4. G4Studio可圈可点

## 4.1. DTO

DTO是数据传输对象的意思，将浏览器端发送来的请求数据，在拦截器中封装成一个对象，然后床给目标action类进行处理。

1. struts1

在struts1里面，每一个表单都需要一个ActionForm进行处理，非常繁琐；

1. struts2

在struts2里面使用一种称为模型驱动或者叫作领域对象的方式实现数据的封装，例如action类中有一个对象user，user中包含成员user.username\user.password，需要单独实现一个user类，user类包括username、password的getter、setter方法，action类包含user的getter、setter方法，然后在前台页面中将表单loginForm中的元素的名称分别设置为user.username\user.password，然后将表单的action属性设置为对应的目标action类，看吧，其实这样做想法是非常好的，但是确实比较繁琐。

1. G4Studio

G4通过拦截器将所有的请求参数封装成一个DTOImple，这个DTOImpl是用hashmap实现的。当我们希望获取前台传过来的某个参数时，只需要调用这个DTOImpl的对应的get方法即可。

对于后台编码人员来说，确实省时省力。

## 4.2. IDHelper

IDHelper主要是用于辅助生成主键，兼容多种类型的数据库产品。

# 5. 经验教训

## 5.1. 技术相关

1）synchronized(session)

struts2里面通过令牌机制防止表单重复提交的时候，在访问session的时候对session加了锁synchonized，这么做的原因就是为了避免多个线程同时对session进行操作。

对于同一个session，每当接收到一个浏览器请求，容器就会建一个线程（或者用空闲的线程）来处理请求。

2）

## 5.2. 环境配置

1） Jdk/eclipse/tomcat

Eclipse里面提示sun.management.ManagementFactory not visible，不要安装高版本1.8的jdk，因为它确实太高了，即便你设置deprecated api为warning、ignore也不行，直接安装jdk1.6得了，稳定省事。

安装了jdk后在eclipse里面可以设置jdk的兼容级别，例如兼容jdk1.7或者1.6等，但是这不能保证完全解决问题，这里我安装的1.8，设置成兼容1.6，也不行，还是not visible，最后直接装1.6了。

jdk6.0支持tomcat6 和tomcat7，但是tomcat7.0.50中包含websocket，websocket要求使用java7以及以上版本，所以我把tomcat换成了6.

2） Struts2常用jar文件

commons-fileupload-1.3.1.jar

commons-io-2.2.jar

commons-lang-2.4.jar

commons-lang3-3.1.jar

commons-logging-1.1.3.jar

commons-logging-api-1.1.jar

freemarker-2.3.19.jar

javassist-3.11.0.GA.jar

ognl-3.0.6.jar

struts2-core-2.3.16.3.jar

struts2-dojo-plugin-2.3.16.3.jar

xwork-core-2.3.16.3.jar

3） Spring常用jar文件

spring-asm-3.0.5.RELEASE.jar

spring-beans-3.0.5.RELEASE.jar

spring-context-3.0.5.RELEASE.jar

spring-core-3.0.5.RELEASE.jar

spring-expression-3.0.5.RELEASE.jar

spring-test-3.0.5.RELEASE.jar

spring-web-3.0.5.RELEASE.jar

spring-plugin-2.3.16.3.jar

4） Eclipse

eclipse里面如果改了一个已经存在的类方法名称的话，java代码，会自动装载，我们可以从eclipse console输出里面看到这些内容。因此不用手动重启tomcat；改变了文件名的话，也不需要，都会自动装载。改变了jsp中的代码的话，也会自动装载，真是太方便了

myeclipse要收费，我的已经过期了，而且不想破解，虽然网上有方法。

最重要的原因是，myeclipse经常性的崩溃，这一点很要命。

5）tomcat work目录

用tomcat作web服务器的时候，部署的程序在webApps下，这些程序都是编译后的程序（发布到tomcat的项目里含的类，会被编译成.class后才发布过来，源文件没有发布过来，但这里的jsp没有经编译的）。tomcat有一个work目录，里面存放了页面的缓存，访问的jsp都会编译（从 work里进入Catalina后的如localhost站点文件夹下的项目，我们可以看到那些jsp页面会被编译成应该是servlet文件，下次再来访问时，就直接运行servlet类就可以向客户端反应响应页面了，所以有的博客说第一次访问时会比较慢，是因为新发布上去的页面在第一个人访问时，会先 编译成servlet文件，所以慢了，一旦编译好，那么除非jsp页面修改，不然下次访问直接运行servlet就可以响应用户，所以快），编译后的文件都会存储在work目录下。而tomcat显示的目录，都会从这个缓存里找编译后的jsp对应的class文件。所以当清空了work目录后，该过程将会从新来过。

有的时候会遇到一个问题，就是修改后的页面在tomcat运行的时候显示不了修改后的痕迹。这个时候删除work目录下对应的项目文件夹，重新启动tomcat就可以了。

删除work目录下内容的时候，可以从eclipse里面选中server，右键选择clean tomcat work directory。

1. 模板引擎问题

模板错误提示信息：Template processing error: "Error reading included file template/~~~ajax/controlheader-core.ftl"，这是Struts2.3.16.1本身的问题，会在以后的版本中修复：https://issues.apache.org/jira/browse/WW-4254 ，在Struts2.3.15中没有这个问题。