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。比如，001下有子节点001001,、001002，则展开001时，将请求001的直接下属节点，获得001001、001002节点信息将其作为001的子节点。如果001001、001002不是叶节点，则当展开该节点时，将请求对应的直接下属节点；如果001001、001002是叶子节点，则不会展开该节点。

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，以及该操作耗费的时间等。

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G4Studio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。

在后续开发的时候，应尽量保证这个表中的数据为空，以确保系统运行没有异常事件发生。

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信息吗？

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。

**g4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，从而可以获取更多的信息。**

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组件其实是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，系统在初始加载的时候，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。

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。

这个表是与权限有关系的，比如给不同的角色授予不同的菜单访问权限、页面访问权限、ui元素访问权限。

13）roleauthorize

这个表是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。

通过这张表，指明了不同的角色可以访问的菜单项，又因为菜单项对应着请求路径，也间接指明了允许角色可以请求的操作。

14）rolemenupart

这个是针对角色的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partauthtype，指明了对ui元素的访问级别，级别为0~6，分别为挂起、禁用、激活、只读、编辑、显示、隐藏。**

15）sequence

这个表是序列号生成器表，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 increment 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、页面上的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或者调整角色b的权限来实现这种控制，那么我就可以再直接给用户a授权访问菜单项。

**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

针对用户的ui元素的授权表。菜单项menuid对应一个页面，为用户userid指定对页面中id属性为partid的元素的访问权限，并且权限级别为partauthtype。

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。

这个表中存储的附加信息包括界面的布局layout和背景background。布局中1表示传统经典布局，2表示桌面布局。原本我以为是g4作者独创的呢，没想到extjs的示例代码里面就有，可以参考示例：<http://extjs.cachefly.net/ext-3.1.1/examples/desktop/desktop.html>。

## 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

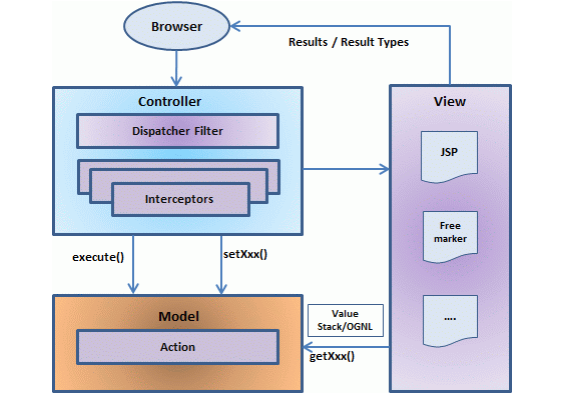
### 3.1.3 Struts 2.0

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

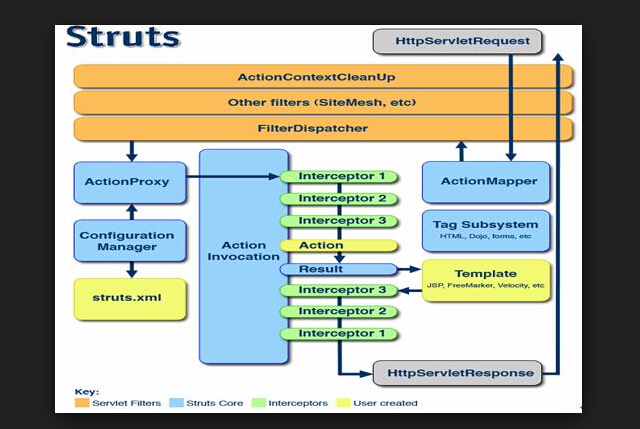
与传统的mvc框架不同的是，struts2的模型层是在action中实现的，而它的控制器则是通过一系列的过滤器和拦截器实现的。这一点要明确，不然要闹笑话。

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ActionInvocation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，可以将代码分成几个部分，

Codeblock1

Res=Arg.invoke();

Codeblock2

Return result；

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 **RES3**

* 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，拦截器栈中后续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：

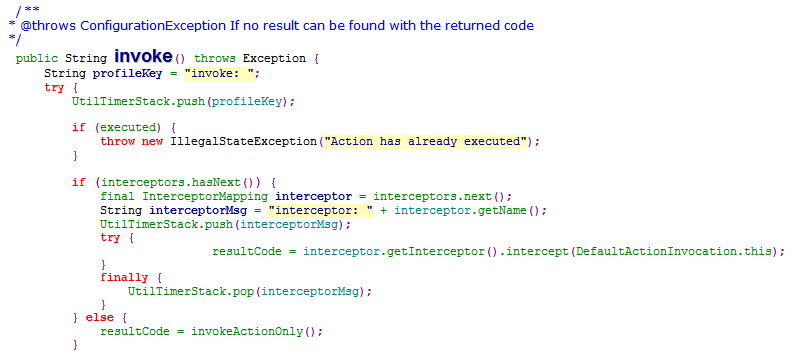
假如action类中实现了Validatable接口（通常通过继承自ActionSupport来间接实现Validatable接口），需要override函数validate()，这个函数可以对用户提交的表单数据进行校验。需要注意的是，这个函数并不是在execute()或者action类的某个其他方法中被调用，它是在workflow拦截器中被调用的。

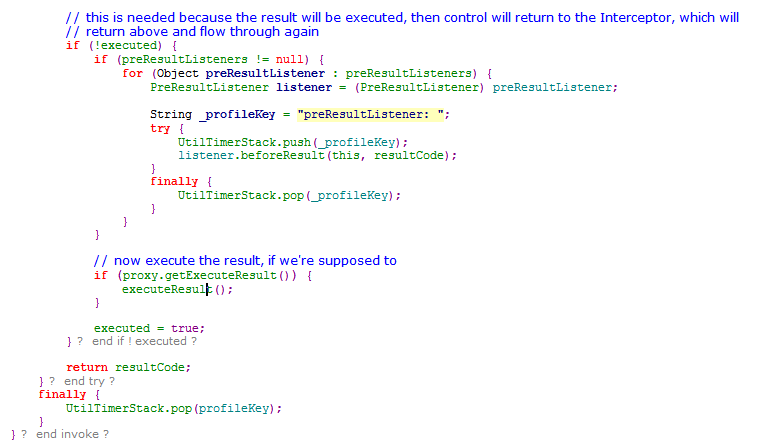
拦截器workflow会首先检查目标action有没有实现Validatable接口，如果实现了的话，就会调用这个类的validate()方法，这个方法中对用户输入进行校验，如果存在某些错误，则会通过方法addFieldError(表单元素名称，错误提示msg)记录出现的错误，这些错误都被记录在ValidationAware接口中。拦截器workflow执行完validate方法后，会检查是否有错误被记录了下来，如果有，则返回”input”，表示需要用户继续进行输入，此时，目标action中的execute方法就不会被执行了。只有validate验证没有错误的时候，才会继续执行execute方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

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，可能会被对应到不同的视图。

**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。或者说拦截器栈中的拦截器一级一级地被调用，第一个调用return的拦截器决定了要转向的视图。**

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





**图xxx invoke方法**

**另外，需要注意，当前次请求最后关联到的视图，是这样确定的。主要还是要考虑拦截器栈中的拦截器以及action的执行情况，从上面的代码executeResult可以看出，它在整个拦截器栈以及action方法执行过程中，只被执行了一次，即这个调用链（拦截器1->拦截器2-…action方法，或拦截器1->拦截器2->…拦截器n）中的最后一个执行结束的时候，返回一个结果字符串，这个字符串在struts.xml里面关联到一个视图，例如jsp页面。以后的各个拦截器的返回值就与视图无关了。**

**Action返回结果，或者第一个调用return的拦截器返回结果后，调用executeResult方法，寻找对应的视图，例如jsp等，加以渲染，然后再回到拦截器层层返回，最后拦截器全部结束，将生成的视图对应的html文件返回给用户。**

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。**

* 应用一系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。另外，如果action中如果添加了自定义的拦截器，如果希望默认的拦截器栈也被包含进来的话，必须显示将默认拦截器栈包含进来。

1. tomcat+struts2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。

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，与action结合生成多个不同的操作请求，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,反之调用目标action的方法，例如execute()。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：

我将一个action类的bean实例写在applicationContext.xml中测试谁可以生成action实例。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。

其实，让struts2负责完成action的实例化，让spring负责service层以及dao层对象的实例化，这样更加实用一点，也容易维护。

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。

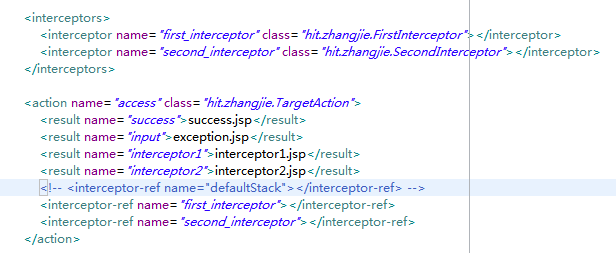
struts2里面的这种注入是在params拦截器里面实现的，只是为了将表单中元素的值当做action对象的成员的值传入到action对象里面方便action后续处理，而spring是为了依赖注入，与struts2的这种表单属性注入action对象的成员，在性质上完全不相同。

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abean、bbean、cbean……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getter，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进行处理。

13）自定义拦截器、默认拦截器栈



图xx 自定义拦截器与默认拦截器栈

**定义了action之后，如果在里面设置了自定义的拦截器，并且没有将默认拦截器栈写在里面的话，则只执行自定义的拦截器栈；**

**当没有在action中设置自定义拦截器的话，则执行默认的拦截器；**

**要想二者都执行的话，必须显示地注明自定义拦截器、默认拦截器栈。**

## 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的返回值类型（变量已经有类型了，为什么不直接用变量的类型呢，为了防止人为构造某个getter方法，但是返回类型与变量类型不一致的情况），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的返回值类型与setter的参数类型必须一致。

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；

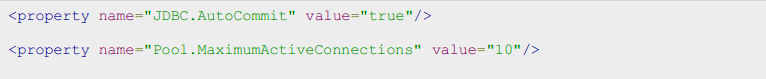
1. Ibatis的安装配置

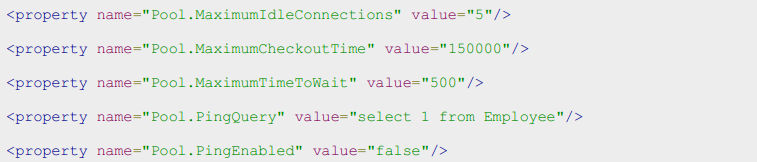
注意对sqlMapConfig.xml的配置，下面是一个配置的实例：



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实现

## 4.1. G4配置

### 4.1.1 web.xml

Web.xml是当前g4平台的主配置文件，其中定义了系统欢迎页面、各种各样的过滤器、监听器，我们一一进行解释。

1）字符编码过滤器

首先需要明白过滤器的实现，过滤器类需要实现Filter接口，由容器在初始化的时候进行实例化，过滤器类中的成员的值，可以在web.xml中对过滤器进行配置时，指定其初始化参数。过滤器类CharacterEncodingFilter实例化的过程是：调用构造函数创建对象实例->调用init方法，init方法接收参数FilterConfig对象，这个FilterConfig对象是容器对web.xml中该CharacterEncodingFilter的配置信息的封装，例如指定的初始化参数信息，然后init方法里面我们手动编写为成员赋值的代码。至此，过滤器的实例化、初始化结束。

以后，当接收到用户请求的时候，根据过滤器的配置信息对请求资源进行拦截处理。

关于此编码过滤器的实现，请查看源代码。

请注意：HttpServletRequest、HttpServletResponse的字符编码的区别。

* Request.setCharacterEncoding？

当浏览器向服务器端发送请求的时候，服务器需要对浏览器提交的请求进行解码，这里的request.setxxx就是告诉服务器端用什么解码方式对浏览器发送来的请求进行解码！

另外通过url访问、超链接访问、表单提交访问，浏览器编码方式稍有区别，通过url访问默认采用utf8，超链接、表单，按照当前显示页面的编码方式进行编码。如果在jsp文件头部设置contentType=”text/html;charset=utf8”，，同时注意在服务器端设置解码方式为utf8，基本上就可以保证万无一失了。

* Response.setCharacterEncoding？

当服务器端要向浏览器端输出html文件时，需要先对文件进行编码，这里的设置就是告诉服务器采用何种编码将html文件编码成网络字节流。

* Response.setContentType?

浏览器接收到服务器发送回来的html文件后，需要对其进行解码，这里的设置是，服务器告诉浏览器采用何种解码方式对服务器发送过去的html文件进行解码。Response.setContentType其实会覆盖response.setCharacterEncoding。这里的设置与在jsp文件头部设置contentType一个效果。

熊春写的CharacterEncodingFilter的doFilter方法有误，将request.setxxxx、response.setxxxx放置到if() {}的花括号内。Encoding为null的时候，哪个也不用设置。

2）请求拦截过滤器RequestFilter，对非法的请求进行拦截，该过滤器所处理的非法请求分成如下几类。

首先明白request.getContextPath\getRequestURI，假如我访问http://localhost:8080/Gnu/login，则getContextPath返回的是/Gnu，而getRequestURI返回的是/Gnu/login。

在g4里面将请求分成两种类型POSTTYPE\_NUDE（裸请求）、POSTYPE\_NORMAL（常规请求），裸请求表示请求不需要经过安全认证，可以直接访问action，常规请求表示需要经过安全认证才可以访问对应的action。

需要进行的安全认证包括如下几类：

* 用户没有登录，却请求与登录无关的操作
  + 用户手动输入与登录操作无关的url，妄图跳过系统引导的工作流程（需要阻止）；
  + 用户伪造页面，伪造js代码，发送ajax请求，请求与登录无关的操作（需要阻止）；
* 用户已登录
  + 处于安全考虑，用户可见页面的展现都必须按照系统规定的流程来，不应该允许用户随便输入url、伪造前台代码来实现对系统的访问，即便用户已经登录也是如此。

熊春在g4里面的实现，还没有达到这个要求，它只是实现了对未登录用户的安全检查，对于已经登录用户则没有进行检查，这是一大不足。可以查看其源代码。

**我们需要改善这点，用户登录后也要验证，必须按照系统流程来访问，不能随便输入个url就访问；另外对于ajax请求，应考虑如何组织用户在前台篡改，例如用户自己写了个ajax请求的代码。**

**Session的本质？如何对session进行管理！！！**

过滤器的执行方式与拦截器有点类似，假定现在有三个过滤器：

过滤器1 过滤器2 过滤器3 拦截器栈 Action

Codeblock1

**fc.doFilter**

🡪 codeblock1

**Fc.doFIlter**

🡪 codeblock1

**Fc.doFilter**

xxx

🡪 **execute() {**

**…；**

**}**

🡨

xxx

Codeblock2 🡨

Codeblock2

🡨

Codeblock2

1. struts2过滤器，这是struts2的入口。
2. 系统启动监听器SystemInitListener

这个监听器实现了接口ContextListener，接收context状态改变事件，并对这些通知事件进行处理。

需要注意监听器的调用顺序，按照在web.xml中指定的顺序，一次调用其init相关方法，当容器关闭的时候，按照相反的顺序调用其destroy相关方法。

ContextListerner接口中定义了contextInitialized方法，接收容器初始化应用程序的事件，并在其中进行某些处理，contextDestroyed方法，接收容器关闭应用程序的事件，并进行适当的处理。

该系统启动监听器主要负责的操作：

* 清理历史会话信息，就是将httpsession中的数据清空。作者在原来设计的时候是将sessionid当做了主键，我暂时不晓得这个sessionid是如何生成的。

**Sessionid是通过session.getId()生成的！！！**

我倒是觉得保留历史会话信息说不定更好，因为这样的话，以后就可以根据httpsession追踪到登录用户的ip地址，如果出现什么安全问题的话，就容易调查。

* 初始化数据信息，当前系统支持的数据库产品包括oracle、mysql、sqlserver，其他的暂不支持。
* 加载数据字典（即code表）
* 加载全局参数表（及params表）
* 打印启动耗时。

1. 垃圾清理监听器IntrospectorCleanupListener
2. 会话监听器SessionListener

该监听器实现了接口HttpSessionListener，需要实现该接口中的两个方法sessionCreated和sessionDestroyed，前者接收session被创建的事件，进行相应的处理，后者接收session被销毁的事件，并进行适当的处理。

熊春在实现的时候，将sessionCreated中没有多做处理，虽然这个时候创建了一个会话，但是还不是一个有效的会话，有效回话的创建在addSession里面进行处理，这个addSession在LoginAction的login方法被调用。

**关于session的生命周期以及与浏览器的关系的详细说明：**

Session存储在服务器端，一般放置在服务器的内存中（为了高速存取），Sessinon在用户访问第一次访问服务器时创建，需要注意只有访问**JSP、Servlet**等程序时才会创建Session，只访问HTML、IMAGE等静态资源并不会创建Session，可调用request.getSession(**true**)强制生成Session。

**Session什么时候失效？**

服务器会把长时间没有活动的Session从服务器内存中清除，此时Session便失效。Tomcat中Session的默认失效时间为20分钟；或者调用方法session.invalidate方法。

**Session对浏览器的要求？**

虽然Session保存在服务器，对客户端是透明的，它的正常运行仍然需要客户端浏览器的支持。这是因为Session需要使用Cookie作为识别标志。HTTP协议是无状态的，Session不能依据HTTP连接来判断是否为同一客户，因此服务器向客户端浏览器发送一个名为JSESSIONID的Cookie，它的值为该Session的id（也就是HttpSession.getId()的返回值）。Session依据该Cookie来识别是否为同一用户。

该Cookie为服务器自动生成的，它的maxAge属性一般为-1，表示仅当前浏览器内有效，并且各浏览器窗口间不共享，关闭浏览器就会失效。因此同一机器的两个浏览器窗口访问服务器时，会生成两个不同的Session。但是由浏览器窗口内的链接、脚本等打开的新窗口（也就是说不是双击桌面浏览器图标等打开的窗口）除外。这类子窗口会共享父窗口的Cookie，因此会共享一个Session。

注意：新开的浏览器窗口会生成新的Session，但子窗口除外。子窗口会共用父窗口的Session。例如，在链接上右击，在弹出的快捷菜单中选择"在新窗口中打开"时，子窗口便可以访问父窗口的Session。

如果客户端浏览器将Cookie功能禁用，或者不支持Cookie怎么办？例如，绝大多数的手机浏览器都不支持Cookie。Java Web提供了另一种解决方案：URL地址重写。

URL地址重写是对客户端不支持Cookie的解决方案。URL地址重写的原理是将该用户Session的id信息重写到URL地址中。服务器能够解析重写后的URL获取Session的id。这样即使客户端不支持Cookie，也可以使用Session来记录用户状态。HttpServletResponse类提供了encodeURL(String url)实现URL地址重写，该方法会自动判断客户端是否支持Cookie。如果客户端支持Cookie，会将URL原封不动地输出来。如果客户端不支持Cookie，则会将用户Session的id重写到URL中。

注意：TOMCAT判断客户端浏览器是否支持Cookie的依据是请求中是否含有Cookie。尽管客户端可能会支持Cookie，但是由于第一次请求时不会携带任何Cookie（因为并无任何Cookie可以携带），URL地址重写后的地址中仍然会带有jsessionid。当第二次访问时服务器已经在浏览器中写入Cookie了，因此URL地址重写后的地址中就不会带有jsessionid了。

监听器的执行顺序：

每个监听器都有自己特有的initialized方法以及destroyed方法，在容器启动应用的时候，会按照在web.xml中声明的顺序从上往下依次执行各个监听器的initialized方法，当容器关闭应用程序的时候，例如容器关闭的时候，会按照相反的顺序依次从下往上调用各个监听器的destroyed方法。

1. 欢迎页面（缺省页面）welcome-file-list
2. 404错误响应页面

请求资源找不到的时候返回404错误代码，对这些错误进行统一处理。

1. 500服务器内部错误响应页面

服务器发生内部错误的时候，发生500错误代码，对这些错误进行统一处理。

### 4.1.2. applicationContext.xml

配置spring，指定需要spring帮我们生成的bean。

### 4.1.3. jdbc.properties

jdbc属性配置，例如连接数据库的驱动程序名称、连接字符串url、用户名、密码。

### 4.1.4. log4j.properties

Log4j非常有用，根据指定的级别打印日志信息。根据指定级别打印相应的日志信息，这种思想，在很多方面都有应用，例如linux内核可以指定printk的日志级别，windows还可以将不同级别的日志信息以不同颜色显示，对系统用户、开发人员来说，不同的人关心的日志信息不一样，所以能根据指定的级别打印日志信息非常有帮助。

在我们程序开发中，可能有时关心比较重要的信息，有时希望看到比较丰富的提示信息，从而对系统运行状态进行判断……总之，log4j可以帮助我们完成这个任务。现在看下log4j的配置方法

需要在log4j.properties中设置对应的需要打印的日志信息级别，其中很重要的一个是log4j.rootCategory，这里的rootCategory级别如果为error，那么所有的debug、info就打印不出来，意思就是只有日志级别高于error的才会被打印出来。

Log4j的级别包括：off、fatal、error、warn、info、debug、all，从左到右，级别由高到低。

当指定rootCategory级别为其中的一个时，例如info时，那么只有级别高于info的才会被打印出来。

另外需要注意的是，log4j里面日志级别存在继承关系，它的继承关系是通过获取到的logger对象的名字来继承的，而logger的名字通常情况下就是当前类的“包名.类名”，当然也可以手动指定，例如a，a.b，a.b.c，那么a.b.c这个logger，在获取这个对象的时候，如果没有手动指定日志级别，它就会使用a.b的日志级别，如果a.b定义的时候也没有指定日志级别，那么就会找到a，使用a的日志级别，如果a也没有指定的话，就使用log4j.rootCategory中定义的日志级别。注意，在创建a.b.c这个logger的时候，a、a.b对应的logger可能没有创建。

总之，记住，log4j中的logger是通过名字来继承日志级别的。

### 4.1.5. struts.xml

主要还是action配置，某些拦截器等。

### 4.1.6. 涉及到g4本身的几个。

g4resource.properties，g4资源文件配置；

global.app.properties，g4 app配置；

global.g4.properties，可以打开关闭某些选项，例如jdbc监视、系统运行模式等等。

## 4.2. G4源码分析

### 4.2.1. action

1）BaseAction

* BaseAction中定义了很多有用的方法，我们后面在写action类的时候，最好继承这个类。
* Spring提供了很多的aware接口，我们在希望通过spring获得某些bean实例的时候，可以在类定义时实现这个接口，并添加相应的getter、setter方法，spring会自动为我们注入这些对象。很是方便，例如常见的ApplicationContextAware可以获取ApplicationContext bean实例，还有其他的，例如ServletRequestAware\ServletResponseAware\SessionAware等等。
* 几个常用的、有效的工具函数：
  + getBean(String)，通过指定bean的名字通过spring获取对应的bean实例，其实现是通过SpringBeanLoader.getSpringBean(String)来实现的。
  + getService(String)，其实这个函数与getBean实现方式是一样的，参数所表达的意思都是applicationContext.xml里面bean的id，只不过这样写更加直观。因为g4将开发分成了如下几层模型，在获取Service类对应的bean时通过getService方法就更加直观，并且能起到强化程序编码人员“分层设计实现”的思想。

前台页面展示

控制器

模型层（业务层）

过滤器

拦截器

Action

Sevice

服务层

Dao

数据持久化层

* + getParamDto()方法将请求参数全部封装成一个Dto类型，Dto就是一个实现了某些其他接口的hashmap，其实现方式就是遍历所有的请求参数，并将他们依次存放到这个Dto对象里面，Dto里面存放的key\value对，其中的value都是String类型或者String[]类型，因此从Dto对象里面取出某个key对应的值时，应该调用getXXAsXXX指明类型。
  + jsonPrint(JsonObject/JsonArray/String)将json对象或者json数组或者String对象打印出来，这个是利用了:

out.println(**Obj.toString()**),

* + jsonPrint(list,formatString)

这个函数跟上面一个函数相比，就稍微高级一点，它在将list中对象生成json对象之前，首先注册了一些数据格式化处理机JsonValueProcessorImpl，这些处理器在碰到java.util.Date\java.sql.Date\java.sql.Timestamp对象类型时，会自动按照指定的格式化字符串formatString对数据进行格式化，这个还是非常有用的。

这个JsonValueProcessorImpl类实现了JsonValueProcessor接口，该接口定义了几个必须实现的方法processArrayValue、processObjectValue。

* + getSessionAttribute(String key)，从session里面获取key对应的取值，返回值为Object类型，根据自己需要自己转换类型。
  + setSessionAttibute(request,key,value)，设置一个session属性对象。
  + getCodeDesc(field,code,request)，获取码表里面与filed、code相关联的描述性信息，注意这里的获取并非访问数据库读取码表code，而是在系统启动的时候就已经将码表中的信息加载到ServletContext里面，这是在系统启动监听器里面完成的。

另外，请注意ServletContext与application对象的异同点。

相同点：

其实servletContext和application 是一样的，就相当于一个类创建了两个不同名称的变量。在servlet中ServletContext就是application对象。大家只要打开jsp编译过后生成的Servlet中的\_jspService()方法就可以看到如下的声明：

ServletContext application = null;

 application = pageContext.getServletContext();

不同点：

两者的区别就是application用在jsp中，servletContext用在servlet中。application和page request session 都是JSP中的内置对象，在后台用ServletContext存储的属性数据可以用application对象获得。

* getParamValue，获取全局参数值，这些值也是在系统启动监听器里面完成的，对应的信息存储在param表里面。
* setOkTipMsg、setErrTipMsg
* getGridDirtyData，获取gridpanel新增和编辑过的脏数据，这个还有待研究。
* encodexxxx，这个是非常有用的。

2）BeanMonitorAction

* 查询异常信息记录，从表exception里面查询

这个类对应的js文件里面，gridpanel里面会将异常信息分页列表显示，同时如果我们双击某一行的话，会将与之相关的详细信息也一并列出来，相当于双击记录展开、再双击又折叠起来，**这种效果是通过控件Ext.grid.RowExpander实现的，如果改一下配置的话，应该能够实现双击展开一个子gridpanel，这是我们要探索的，因为我们的同一个协议的不同版本需要展开列表显示，我们需要这个功能。**

* 删除异常信息记录

3）EventTrackAction

* 这个action对应的操作页面即g4中的Request请求跟踪菜单项。
* 查询请求事件，从表event里进行查询
* 删除请求事件

4）HttpSessionMonitorAction

这个是用来监视http session的。

* 查询会话列表
* **杀死某个会话，到底能不能杀死，还有待验证。我得建个adhoc网络让舍友帮着测试一下。**

**它这里的杀死会话，可能与登录时安全检查时的最后一种有关系，等待检查。**

5）IndexAction

* 方法IndexInit主要是根据用户的配置信息，决定界面的显示是以经典布局来呈现，还是以桌面布局的方式来呈现。在界面初始化的时候会返回经典布局视图或者桌面布局视图。
* 用户可以定制自己喜欢的主题颜色、界面布局、锁定系统、解锁系统、修改密码等。

6）JdbcMonitorAction

* 查询监控记录，从jdbcmonitor表里面进行查询
* 删除监控记录

7）LoginAction

* Init方法，使用户返回到登录界面
* Login方法，检查用户输入账户是否合法、输入密码是否合法（用户密码des加密存储）、是否允许同一个用户创建多个会话连接（根据配置信息中是否允许建立多个会话连接加以决定，关于多个会话请参考前面介绍的session与浏览器的关系）。

此外，该方法不允许在同一个pc端以不同用户身份登录系统。

* Logout方法，退出登录，使session失效
* 保存用户登录事件
* 保存用户退出事件

8）OrganizationAction

* departmentInit方法，主要是获取用户所在的部门id以及部门名称，并将其设置设置到request里面传递给对应的jsp页面，然后jsp页面中包含的js文件中就可以根据这个部门id、部门名称进行后续处理，例如根据部门id生成部门树。
* 部门树初始化departmentTreeInit方法，初始化的过程是这样的，首先在上一步的

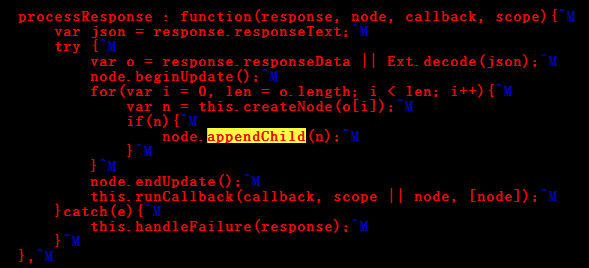
基础上，返回到浏览器的html文件中包含了用户所属部门的deptid以及js文件，js文件中有一段代码会请求parentid为deptid的所有直接下属部门信息，这个在deptTree.on(‘click’,function(node)) {}中是实现，服务器返回的这些直接下属部门信息里面可能包含叶子节点也可能包含非叶子节点。每个树节点都有两个非常重要的属性，一个是node.attributes.id表示其deptid，一个是node.attributes.text表示其deptname。这里的attributes里面可以防止用户自定义的属性。Leaf属性是node.leaf，当然也可以根据自己需要定义到attributes里面。

注意，异步加载树，假如其中有一个非叶子节点a，在将这个节点展开之前，是无法获取到这个对象的？？？熊春说的，有待验证！！！

G4里面将部门id长度为6的自动展开，也就是说总公司（id长度为3）是一级部门的话，g4会自动将二级部门展开。如果其中的三级部门是非叶子节点，不会自动展开，当我们去点击展开的时候，才会向服务器端发送departmentTreeInit方法，请求该节点对应的子树节点。通过浏览器发送的请求数据可以验证这点。这是extjs的对异步加载的实现方式。

如果采用同步装载的话，就不是这样了，所有的部门信息一次性请求到位，但是这样数据量大，延时大，不明智。

现在解释一下如何实现树的自动装载，以g4为例，当我们展开了某个节点之后，向服务器发送departmentTreeInit.do请求，服务器返回该节点下的直接下属部门的json数据，浏览器端接收到相应后，准确地说是树形控件发送完上述请求收到相应后，执行当前节点对应的子树的装载操作，装载就是创建当前节点的子节点，其实现代码在TreeLoader.js里面，如下图所示：



图xx 树请求\接收子节点数据后的装载操作

此外，需要区分的是，树形控件中的节点是可以折叠的，当我们点击树节点时是属于click事件，当我们点击树节点前面的“+”号时，是expand事件，当我们点击了“+”后，会自动向之前在树中指定的url发送请求，获取响应数据后，执行上图中的处理代码。

* 查询部门用于分页显示

这里用的是后台分页，实际上是由ibatis提供的支持，以queryForList为例，我们如下调用getSqlMapClientTemplate().queryForList(statementName, qDto, start.intValue(), end.intValue())，其中statementName表示我们要执行的查询sql的id，qDto是提供的查询参数，start.intValue()表示现有记录的起始行编号，end.intValue()表示记录的结束行编号（这里的行编号，不是rowid（针对oracle））。

* 保存部门信息
* 更新部门信息
* 删除部门
* 通过部门id查询部门信息

9）ParamAction

* Init方法，初始化参数管理界面，查询的时候从param表里面查询
* 查询全局参数分页显示（分页本质上是由ibatis实现的，g4里面只是考虑到了多种数据库产品之间的兼容性，例如mysql、sqlserver的起始行编号从0开始算起，oracle从1开始算起）
* 新增、删除、更新全局参数
* 内存全局参数的同步，因为在系统启动的时候，系统启动监听器将param表的信息加载到了内存中，当我们现在对表param进行了增、删、改之后，需要使得内存中的参数与数据库表param中的内容一致，因此需要从内存中移除旧的paramlist，并重新读取、加载到内存。

10）PartAction

Ui元素托管，菜单类型分为系统菜单和业务菜单。初始化菜单树的时候，自动展开二级菜单，一级菜单为01，二级菜单menuid长度为4，例如0101。

* Init方法，返回initView，返回managePart.jsp页面，其中引用了managePart.js文件。
* queryMenuItems，managePart.js文件中的菜单树控件，其根节点是异步节点，并且设置属性expanded为true，当菜单树被创建的之后，会自动请求queryMenuItems.do操作，该操作返回该菜单项下的直接下级子菜单，并且只返回菜单类型为SystemConstants.MENUTYPE\_BUSINESS的子菜单，我们前面提过菜单类型分为两类，一类是业务菜单，也就是SystemConstants.MENUTYPE\_BUSINESS，一类是系统菜单SystemConstants.MENUTYPE\_SYSTEM。

注意，系统菜单我们不允许对其进行授权管理，只允许对业务菜单进行授权管理。

* queryParts，查询当前菜单项menuid对应的页面request中已经托管的元素的id，即表menuparts中的cmpid。
* **saveDirtyData，保存gridpanel的脏数据（修改后但未保存的数据），该方法以及对应的getDirtyData有待检查**。
* 删除已经托管的元素id。

11）ResourceAction

菜单资源管理页面。

* 菜单资源初始化
* 查询菜单项
* 查询菜单项并用于分页显示
* 新增菜单项
* 修改菜单项
* 删除菜单项

删除菜单项的时候，包括三步操作，首先删除菜单本身，另外菜单可能参与了角色授权、用户授权，因此需要从roleauthorize里面删除对应的授权，从userauthorize里面删除对应的用户授权。

* 码表初始化codeTableInit，这个方法不是在菜单资源管理里面调用，而是在“基础数据维护”->“字典维护”中调用。
* 新增、修改、删除、查询码表记录。
* IconInit，图标管理界面初始化
* 图标的查询
* **当前没有实现图标的修改操作，如果不能删除的话，也应该可以实现图标的修改的，这个以后再考虑吧，说不定还要一套图标都换呢。**

**暂时先用系统默认的图标。**

12）RoleAction

与角色管理与授权页面进行交互，主要涉及到对角色的添删改查以及针对角色授予对应的权限，包括经办权限、授权权限、选择人员（为某些user赋予当前角色）。

* RoleInit，角色管理初始化

G4中的角色，一共只有两种类型，管理角色、业务角色。这样分是合理的，管理角色负责系统的维护，业务角色负责在系统中执行相应的业务操作。

**这样的话，需要将业务角色进行细分，以便不同的业务角色能够执行与角色相符的业务操作。**

**如果在g4的基础上实现基金会的话，该如何修改呢？**

* **基金会包括的角色类型，我们也可以首先分成两种，一种是管理角色，负责系统的维护，另一种是业务角色，负责基金会的业务操作。**
* **基金会在业务上涉及到的角色包括，捐赠方、项目负责人、学院（受赠单位）、项目管理部、法务部、财务部、秘书长。我们可以将业务角色分成这几种。**
* **然后我们绘制一个权限矩阵，表示出管理角色所具有的对系统维护相关的操作；业务角色中的各个角色又涉及到哪些操作。**

**权限矩阵如下：**

* **角色及其权限构思好了之后，就可以在角色授权管理界面，对其涉及到的权限进行分配，包括：**

**角色可以访问的菜单项menuid，设定了可访问的menuid，也即表示设定了角色可以访问到的页面；**

**Menuid对应着可以访问到的页面，页面中包含很多ui元素，通过ui元素托管，对角色进行授权，决定不同的角色可以看到页面中的特定元素；**

**学长当时说，有可能会涉及到数据级的权限，现在想想确实是。 具体的说，比如项目立项的时候，项目负责人只能修改立项信息中的某几个字段的信息，这个时候，我们就需要对其进行授权。以决定该项目负责人角色可以修改那些表的那些字段。**

**当将信息呈现给用户时，比如以一个可编辑表格呈现给用户，用户可能会修改js使得被禁用的字段被打开，然后修改后提交。这个时候，如果将数据提交到后台的时候，肯定不能将所有的字段信息进行更新。**

**G4里面提供了一个获取可编辑表格脏数据的函数getDirtyData，个人以为应该在请求拦截过滤器里面加一些判断，或者再加一层拦截器，专门负责用户数据级的校验。假如我们将每个角色可操作的表以及可操作的字段存储在一张表中，可以就从表中获知当前用户可以修改哪些字段，假如在脏数据中发现了表中没有的字段，则可以认为，用户对前台的代码进行了篡改，拒绝后续处理。**

**现在g4中的实现方式是，首先创建角色，角色分为两种管理角色、业务角色；**

**Ibatis的sqlmap中的namespace、id是区分大小写的，在通过appDao执行对应操作的时候要注意。**

**Sqlmap中namespace首字母大写、id严格按照驼峰命令方式；**

**变量名采用驼峰方式，名字应该简短的同时透漏出其表达的意义，想不出好的名字可以google，严禁用拼音缩写；**

**Java代码中设计到数据库操作、类型转换等的可能出错的代码，用try-catch-finally括起来，解决可能抛出的一切异常；**

* 角色的添加、删除、修改、查询操作

13）RolePartAction

这个action负责为角色进行ui组件的授权操作，是针对某个具体页面中的ui组件的可见性、可操作级别进行授权。注意与RoleAction的区别。

超级管理员super可以对“管理角色”进行授权操作，具体包括经办权限授权、授权权限授权、选择人员。系统开发人员developer的权限跟super一样；

超级管理员super和developer不能对“业务角色”进行授权操作，“业务角色”的授权操作必须由同部门下的“管理角色”进行授权。这样做是合理的！！！

注意用户的用户类型分两种，“管理员”和“经办员”。

* Init方法，进行角色授权操作的初始化操作。
* 查询角色授权条目
* 保存角色授权信息

14）ServerInfoAction

获取服务器信息以及内存实时监控数据。

* Init方法，服务器监控信息初始化。
* UpdateHostMemChart，更新监控到的内存信息。
* UpdateCpuChart，更新监控到的cpu信息。
* UpdateJvmChart，更新监控到的jvm信息。
* GetCpuRatioForWindows，获取cpu的使用率（适用于windows操作系统）。

这个暂时不需要太多关注，先重点关注业务模块。

15）UserAction

用户管理与授权，主要涉及到用户的添删改查以及为用户的授权操作，这里的授权包括，首先为用户赋予角色信息，在此基础上赋予用户可以访问到的菜单信息。

同一个用户可以具有多个角色，其权限是多个角色的并集，另外，在指定可以访问到的菜单时，可以根据角色可以访问到的菜单，选择其中的某个子集或者全部，这一点g4做的不好，当我为某个用户赋予某个角色的时候，对应的该用户可以访问到的菜单项应该是用户具有的角色可以访问到的菜单项，然后在此基础上，可以选择众多菜单项中的某个子集。这样比较合理，g4没有将可以默认访问到的菜单项勾选上，这样用户就不知道它的权限应该在哪些子菜单项里面选择，不方便，还容易出错。

16）UserPartAction

对用户可以访问到的页面中的ui组件进行授权。

现在需要补充一下：

* Roleauthorize表，为角色roleid指定其可以访问到的菜单menuid（相当于知道可以访问哪些页面了）；

Userauthorize表，为用户userid指定其具有的角色roleid；

* rolemenupart表，为角色roleid指定其可以访问到的菜单menuid对应页面中ui组件partid的访问级别；

Usermenupart表，为用户userid指定其可以访问到的菜单menuid对应页面中ui组件

partid的访问级别；

**对用户授权时，可以不为用户授予任何角色，可以单独为其指定可以访问到的菜单资源、菜单对应页面中的ui组件的访问级别。**

**之所以设置角色、角色授权以及为用户指派角色功能，是为了给用户授权时提供方便，设想如果公司有1000个员工，每个员工可以访问10个菜单项，如果没有角色的话，直接为用户授予可以访问到的菜单项这种方式工作量非常大；如果有了角色，我给角色指派好可以访问到的10个菜单项，然后为每个员工赋予这种角色就可以了，工作量大大减少。**

**如果能够将两者结合，则对权限的管理更加灵活方便。**

* usermenumap表，为用户userid指定其可以访问到的菜单menuid对应页面的访问级别；

### 4.2.2. base

1）basebean.BaseBean

这是一个抽象类，其中toDto方法，可以将当前类的成员属性封装到到Dto中返回。

ToXml方法，能够将当前对象转换成一个描述当前对象的xml字符串并返回。

ToJson方法，能够将当前对象转换成一个描述当前对象的Json字符串并返回。

其中toXml和toJson都是首先调用toDto方法，将对象转换成Dto，然后再调用Dto中封装的方法toXml、toJson来完成最后的转换。

2）common.Dto

该接口extends Map，定义了一些非常方便使用的方法。

在g4里面，我们使用最为频繁的，莫过于getParamDto来讲所有的请求参数封装成一个Dto来使用了，还有将Dto作为ibatis的数据库查询时的paramClass，还有服务器将结果返回给浏览器时的write(dto.toJson)等等吧。

3）common.DtoImpl

是对Dto的实现，建议看看其实现代码。

补充点java方面的知识：

**关于类和接口的继承，类只能extends一个父类,但可以implements多个接口；接口不能extends类,但可以extends多个接口,接口不用implements关键字。**

**接口中不允许对方法进行实现，抽象类如果通过implements添加了对某个接口的实现，因为它是抽象类，不能够被实例化，所以可以不用全部实现接口中定义的方法。**

### 4.2.3. bean

1）DeptVo

2）MenuVo

3）RoleVo

4）UserInfoVo

5）UserVo

与对应的表dept、menu、role、userinfo、user对应，添加了getter、setter方法。

在从数据库获取对应信息的时候，通过ibatis vo映射的方式。

建议全面了解下ibatis。

### 4.2.4. dao

1）Dao

这是一个接口，在其中定义了一些数据库操作常用的方法。

2）DaoImpl

对Dao的实现，建议查看下其源代码，了解下，如果不采用g4的话，程序中如何写才能与ibatis结合。简述一下吧，假如自己写一个数据库操作的类的话，让这个类继承自抽象类SqlMapClientDaoSupport，这个抽象类中提供了getSqlMapClientTemplate()这个方法非常重要，通过这个方法获得了SqlMapClientTemplate对象之后就可以调用execute、insert、update、queryForObject、queryForList方法了，这个时候就会去查我们事先准备好的sql映射xml文件。

### 4.2.5. exception

1）Exception

继承自RuntimeException，添加了自定义构造函数Exception(String msg)，g4中我们常看到某些方法定义的时候使用了throw Exception(….)，涉及到的有可能抛出异常的地方可能就会使用这里自定义的构造函数Exception(String msg)，其实jdk里面有这个构造函数Exception(String msg)，这里这个是override，添加的实际异常信息是Constants.*Exception\_Head* + msg，而不是msg，Contants.Exception\_Head是我们定义好的，这样就能将我们自己抛出的异常与jre抛出的异常区分开来。

也算是容易对异常信息进行区分，方便定位出错的原因。

2）NullAbleException

非空异常校验类，有多个构造函数，分别用于生成对象为空、对象属性为空的异常。

3）PrcException

调用存储过程异常校验类，有多个构造函数，可以传递的参数包括调用的存储过程名、错误信息、错误代码。

一般数据库错误的话，如果能够将错误代码通过打印异常信息显示出来的话，对程序设计人员定位错误原因是非常方便的。

### 4.2.6. filter

1）CharacterEncodingFilter

字符编码过滤器

前面已经提过了，结构优点混乱，以后会慢慢整理。

2）RequestFilter

请求过滤器

前面已经提过了，结构优点混乱，以后会慢慢整理。

### 4.2.7. lisenter

1）SessionContainer

会话容器，其中存放userinfo以及reportdata，会话容器被存放到session中。

前面已经提过了，结构优点混乱，以后会慢慢整理。

2）SessionListener

http会话监听器。

前面已经提过了，结构优点混乱，以后会慢慢整理。

3）SystemInitListener

系统初始化监听器，或系统启动监听器。

前面已经提过了，结构优点混乱，以后会慢慢整理。

### 4.2.8. service

1）base.BaseService

接口定义

2）base.BaseServiceImpl

实现了1）中的接口，其中定义了成员appDao，并定义了setter方法，该成员appDao最终会由spring注入，我们再写自己的Service类的时候，别忘了让接口xxxService继承自BaseService，写xxxServiceImpl的时候，让其也继承自BaseServiceImpl，这样在我们自己的xxxServiceImpl中直接使用appDao就可以了。

其实接口BaseService里面什么也没有定义，BaseServiceImpl里面增加了appDao，唯一的好处就是继承它就不用在自己的类里面定义成员appDao了。

3）xxxService+impl.xxxServiceImpl

在这里说说为什么要定义一个xxxService，再定义一个xxxServiceImpl，这体现了面向接口编程的思想。在我们实现接口的时候，可能会有多个版本，多种不同的实现，我们在applicationContext.xml中指名xxxService有哪一个xxxServiceImpl实现，可以很方便地进行切换，也体现了一种解耦和的思想。

1. 提供的其他的Service都是为g4正常运行提供的，不细细描述了。

### 4.2.9. util

#### 4.2.9.1 id

根据sequence表中的说明，生成id，并进行适当的格式化，要想生成某个表的id，必须在这个表里面进行注册，并在IDHelper里面添加适当的代码。

这里必须要提出的是，维护sequence表以前让我们付出了不小代价，对于mysql这种支持数据库主键自增的数据库产品，我不建议使用这里的sequence+IDHelper的方式来获取主键。

如果想兼容多种数据库产品的话，则需考虑。但是我们没有这种需求，我们的数据库产品使用mysql就可以了。

#### 4.2.9.2 json

这个包下面的类使用还是比较频繁的，在服务器端向浏览器端返回结果数据的时候，我们常常返回的是json格式的数据。JsonHelper这个工具类提供了一些方法，可以把对象、列表对象等编码成json数据。

#### 4.2.9.3 properties

用于装载属性配置文件的，属性配置文件中每行都是键值对的形式存储，例如：

Key1=value1

Key2=value2

#### 4.2.9.4 report

与导出报表相关的工具类，这里的导出报表基于模板的，其实我们也可以不基于模板来实现。

利用模板的好处是，以excel为例，对于某些复杂的格式，例如三个单元格合并成一个的情况，如果目标导出格式已知，我们提前建好模板的话，那么我们以后只关心写入数据即可，不用关心格式相关的操作，这可能是大家优先考虑模板的原因。

如果我们不用模板的话，需要了解**jxl**操作excel文件格式的方法，这样的话就比较麻烦，工作量可能会很大。但是不基于模板也有自己的优势，就是它不受模板格式限制，我们而可以通过程序自由控制输出文件的格式，比如列、行、单元格合并、分裂的组织。

对于其他格式的导出，例如**pdf**等等。

#### 4.2.9.5 taglib

G4提供的标签库，其中常用的应该理解其工作原理，以便我们需要修改时对其进行修改。

#### 4.2.9.6 tplengine

模板解析引擎，当前版本的g4只支持velocity模板，不过用好这一个也够用了。前面有对几种velocity、freemarker、jsp的对比。

结构有些混乱，以后会整理的。

#### 4.2.9.7 typecast

涉及到类型转换的工具类。

#### 4.2.9.8 xml

Xm解析工具类，基于dom4j实现。

#### 4.2.9.9 Constants

定义了一些常量。

#### 4.2.9.10 SpringBeanLoader

SpringBeanLoader该类的静态方法被第一次调用的时候，即该类在首次被类装载器装载的时候，其中static {xxxxx}中的代码被执行，其中调用了initApplicationContext方法，该方法实例化一个applicationContext对象，其方法getSpringBean方便我们获取指定的bean，其实现方式是applicationContext.getBean。

某些类中可能需要获取applicationContext的地方，就不要实现ApplicationContextAware接口了，直接通过SpringBeanLoader.getApplicationContext获取就可以了。

一般我们只是获取bean，几乎用不到获取applicationContext。

#### 4.2.9.11 SystemConstants

定义了一些系统常量。

#### 4.2.9.12 ThemeUtils

获取与选定的主题相匹配的颜色，从而改变系统的整体色调。

#### 4.2.9.13 Utils

这也是一个非常有用的工具类，建议先查看下期中提供的方法，避免重复造轮子。

#### 4.2.9.14 WebUtils

涉及到web 请求、相应、会话、cookie的一些工具类。

2）界面以及布局

（1）首先整个界面的布局分为两种形式，一种是经典布局、一种是桌面布局。经典布局就是一种普通的布局，而桌面布局就是我们见过的类似于windows桌面的一种布局方式，主要的我们还是使用经典布局，桌面布局后面可以继续探索，其实也不算复杂。

* 经典布局的生成

经典布局的生成得益于g4标签库HRMS.tld，其中定义了标签hrms:viewport标签，对应的解析类是ArmViewportTag，该解析类调用velocity引擎将数据与模板ArmViewportTag.tpl进行合并。合并的过程不算复杂，关键是要看清这个tpl文件的内容，这样后面的开发有帮助。

捡重点的说，首先在这个tpl文件里面创建了一个viewport，viewport控件在渲染的时候是渲染到整个窗口中去的，这个比较特殊，然后在这个viewport里面创建了north、south、center、west子区域，然后一个比较重要的组件是mainTabs，它被添加到center区域中，负责维护所有的标签页，注意，我们所写的jsp文件（其中引用了js文件），在显示的时候并不是打开一个新的浏览器标签页进行显示，而是在mainTabs这个TabPanel里面创建一个新的标签页进行显示。

其实现方式就是，当我们点击west区域中树控件中的某个节点的时候，这个节点，暂时就称之为功能菜单项吧，当我们点击这个菜单项的时候，会调用addTabs函数，因为每个菜单项都对应着一个待请求的action操作，这个action操作有一个url，addTabs会检查这个通过这个url生成的标签选项卡是否已经存在，如果已经存在的话，那么直接将对应的tab设为活动标签页即可；如果不存在的话，那就创建一个新的标签页，并在标签页里面增加一个iframe元素，iframe元素相当于嵌套了一个html网页，并且制定iframe的src属性为菜单项待请求的url，这样当iframe元素被创建的时候，就会自动请求对应的url，将我们事先定义好的jsp页面显示出来。

* 桌面布局的生成

桌面布局是用的velocity模板、js文件与经典布局不一样，可以自己查看下源代码，了解下。

（2）G4当前支持的模板引擎只有velocity一种。Velocity支持合并模板、合并字符串模板、合并文件模板三种类型。这在其源代码中可以查看到。

（3）

# 5. G4Studio优势/不足

## 5.1. 优势

### 5.1.1. 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。

### 5.1.2. 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。

## 5.2. 不足

5.2.1. 权限相关

权限管理这块，g4的代码中存在一些问题，有的是bug，例如授权页面显示不出来！有的是不合理的地方，例如给用户授权后，假如先给用户授予了角色的权限，那么在授予用户菜单访问权限的时候，应该将角色已经覆盖的菜单项明确地标识出来。

G4里面的权限管理基本上可以满足基金会的需要了，但是对于数据级权限问题等，最好是在struts里面加一层或几层拦截器来负责。

建张表负责维护角色、用户对表、表字段的操作级别，在添加的拦截器里面对用户请求的操作进行检查，如果试图越权操作，则返回错误警告；正常请求操作，则允许。

首先前台我们肯定会对用户的可访问性操作进行限制，主要是防止用户修改前台代码来非法访问。

# 6. 经验教训

## 6.1. 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）Properties getProperty(key, defaultValue)

如果key没有找到，则返回defaultValue。

3）x-requested-with

服务端判断客户端请求是传统的同步请求方式，还是ajax异步请求方式，如何判断呢？ajax异步请求方式，在请求头里面多了一个字段x-request-with，通过判断这个参数的有无可以判断出是同步请求，还是异步请求。

4）如果希望修改系统的全局颜色风格，可以参考g4的主题切换的实现，查看代码发现，这主要是通过设置themecolor（实际上就是选择的theme），并将themeColor的值设置到ArmViewportTag.tpl中的css类.banner中。

**5）Grouping** [**http://extjs.cachefly.net/ext-3.1.1/examples/#sample-3**](http://extjs.cachefly.net/ext-3.1.1/examples/#sample-3)**，这里的这个例子我们可以借鉴，用于实现将同一个协议的不同版本进行合并。**

**[G4中：我们当时想同一协议的不同版本可以折叠成一行，双击后也可以展开，可以考虑使用这个控件加以实现Ext.grid.RowExpander。]**

6）Extjs的包含文件在hrms:html这个tag标签的解析类中实现了，具体的是在HtmlTag.tpl中。

7）发现一个bug，当我修改了某个菜单的请求路径之后，对应的树菜单中的对应子菜单中的节点的属性并没有改变，应该检查一下，如果当前被修改的菜单项已经显示出来的话，应该获取其子节点刷新其请求路径属性，这样就不用关闭页面重新打开、或者刷新。即使像现在的实现需要刷新，也应该提醒用户需要刷新。

8）发现一个问题（算了还是不叫它bug了），hrms:import在引入的时候，假如说我写的src属性为“path”,当前web路径为<http://localhost:8080/context>，该标签的解析类在解析的时候，不会检查path前面有没有符号/，造成拼接出来的src属性值为contextpath，这样就会出现找不到的情况，所以要注意，在js资源名称前面加上符号/。

9）Extjs的gridpanel的哪些列可以显示哪些列可以不显示，是内置支持的。

10）发现一个bug，人员管理与授权里面，用户的类型信息无法进行修改。角色管理与授权、人员管理与授权，都存在这个问题，有时候添加的时候也不能选择类型。

11）如何获取request对象？

* 在action里面可以通过实现HttpRequestAware接口，spring依赖注入的方式来获取request对象，使用g4的时候，我们只要继承自BaseAction就可以了，里面包含了成员request对象；
* 在过滤器里面，doFilter的参数中包含了request对象，也包含了response对象；

在拦截器里面，也可以实现HttpRequestAware这个接口的方式，我们也可以让它继承自BaseAction这个类，这个类实现了HttpRequestAware接口；

* 在TagSupport的子类里面，可以通过pageContext.getRequest()获取；

12）查看UiGrantTag

## 6.2. 环境配置

1） Jdk/eclipse/tomcat

Jdk1.8，Eclipse里面提示sun.management.ManagementFactory not visible，即便设置deprecated api为warning、ignore也不行。

注意jdk/jre与tomcat的兼容问题。

**注：现在我想用jdk1.7，tomcat7.0来运行g4，编译器编译兼容级别是1.7，这种情况下，将sun.management.ManagementFactory替换成java.lang.management.ManagementFactory，引入的包也进行相应的替换，ok。**

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。

1. G4studio用tomcat6.0不行，需要用tomcat7.0，用tomcat7的话，其中有websocket模块，jdk1.6不支持，可以暂时将这个模块移除。
2. Eclipse、myeclipse

现在我的myeclipse过期了，也不想想办法进行破解，所以我决定直接将eclipse进行开发。

大家常说myeclipse可以实现热装载、热部署。热部署指的是，myeclipse自动将修改后的代码编译了一遍，然后将编译后的代码以及jsp文件等部署到web容器中；热装载指的是，myeclipse将新的字节码替换原来的字节码文件，根本上热装载是tomcat提供的支持。要修改其server.xml加入参数reloadable=”true”就可以了。其实myeclipse也是通过修改web容器的这个参数实现达到热装载的目的。

不只是myeclipse，eclipse现在也具有了上述热装载、热部署的功能，可能某些方面不如myeclipse完善，但是，至少现在，我发现eclipse工作还是很出色的。

现在我把g4的工程文件从myeclipse下面转移到了eclipse下面，正确配置之后，可以启动了。

10）Eclipse界面卡死

修改web.xml并保存时，eclipse整个界面会卡死，原因是因为工程的dynamic web project的版本与web.xml中的版本不一致。通常如果是新建了一个工程，会自动生成web.xml，如果web.xml中的内容是从另一个工程中的web.xml中复制进去的话，则可能引起版本不一致。

做如下修改就好了（假如新建的时候采用的是3.0），现在的web.xml中是2.5，做如下修改。



图xxx 修改前的web.xml



图xxx 修改后的web.xml

其实卡的原因，主要是因为版本不一致，这个时候eclipse会进行联网校验。如果断网的情况下也不会卡。

11）mysql编码

为了保持编码一致，做如下修改，修改mysql配置文件my.ini(linux下位my.cnf)，将其中的client部分修改为gbk，这主要是考虑到在windows命令行里面能够正常显示（linux下可以设置成utf8，终端默认是utf8编码）；将服务端部分设置成utf8，以后创建数据库的时候默认采用utf8编码，我们要source执行的sql文件中创建表的时候也应采用utf8编码。

12）Servlet-api.jar以及javax.servlet.jar不应该部署到WEB-INF/lib目录下，这相关的jar包的功能由容器来提供，可以在包含路径中包含它，但是不应该将其拷贝到目录WEB-INF/lib下，否则容器启动程序的时候，会发出警告信息，“javax.servlet-api-3.0.1.jar - jar not loaded”。