LLSFW

框架详细说明文档

[1 简介 4](#_Toc400988294)

[2 技术体系 4](#_Toc400988295)

[3 平台功能 5](#_Toc400988296)

[4 重要资源 5](#_Toc400988297)

[4.1 快速开始 5](#_Toc400988298)

[4.1.1 准备 5](#_Toc400988299)

[4.1.2 内容 6](#_Toc400988300)

[4.1.3 环境搭建 6](#_Toc400988301)

[4.2 详细介绍 12](#_Toc400988302)

[4.2.1 配置文件总览 13](#_Toc400988303)

[4.2.2 私有配置文件 14](#_Toc400988304)

[4.2.3 公共配置文件 18](#_Toc400988305)

[4.3 开发范例 25](#_Toc400988306)

[4.3.1 表结构设计 25](#_Toc400988307)

[4.3.2 生成model,sqlMap和Mapper 25](#_Toc400988308)

[4.3.3 创建service层 27](#_Toc400988309)

[4.3.4 创建controller层 28](#_Toc400988310)

[4.3.5 创建前端结构 28](#_Toc400988311)

[4.3.6 开发查询功能 29](#_Toc400988312)

[4.3.7 开发新增功能 33](#_Toc400988313)

[4.3.8 开发修改功能 37](#_Toc400988314)

[4.3.9 开发删除功能功能 40](#_Toc400988315)

[4.4 自带功能介绍 41](#_Toc400988316)

[4.4.1 环境变量设定 41](#_Toc400988317)

[4.4.2 定时任务管理 41](#_Toc400988318)

[4.4.3 功能维护 49](#_Toc400988319)

[4.4.4 角色维护 49](#_Toc400988320)

[4.4.5 角色维护 51](#_Toc400988321)

[4.4.6 用户维护 52](#_Toc400988322)

# 简介

LLSFW是以Spring Framework为核心的，务实风格的JavaEE应用集成参考示例，是JavaEE世界中的主流技术选型，最佳实践的总结与演示。

# 技术体系



# 平台功能



# 重要资源

## 快速开始

### 准备

框架使用MAVEN构建,所以说那种以往100多M的傻瓜安装包不会再有了,你首先需要的是下载依赖包时的一点点耐心.

1.安装JDK6.0+ 和 设置JAVA\_HOME

2.安装MAVEN3.0+ 和 设置MAVEN\_HOME(如果机器不能直接上网, 请参照[Maven](https://github.com/springside/springside4/wiki/Maven)相关文档,设置代理或私服)

3.安装TOMCAT 6.0+

4.将JDK,TOMCAT和MAVEN配置到ECLIPSE中

### 内容

框架的安装包结构如下:



llsfw-core:为框架核心代码,里面包含了各个开源框架的集成,封装的API,以及权限和安全管理的服务器端代码

llsfw:为框架的web端,里面包含了各个开源框架的配置文件,前端集成,前端封装的API,以及权限管理,系统登陆等客户端代码

llsfw-generator:为自动生成器,可自动生成model层和dao层的代码,提高开发效率

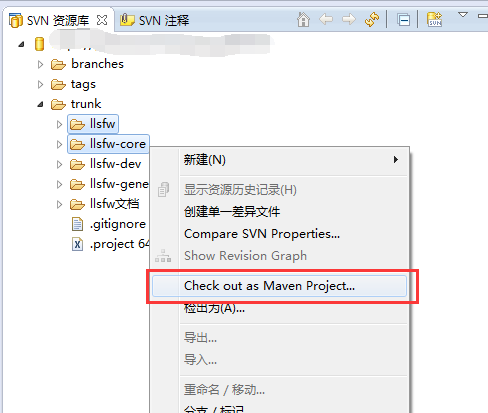
llsfw-dev:为演示项目,包含了集成llsfw框架的方法

llsfw文档:为项目文档项目,包含了框架中所有的文档,DBMODEL等内容

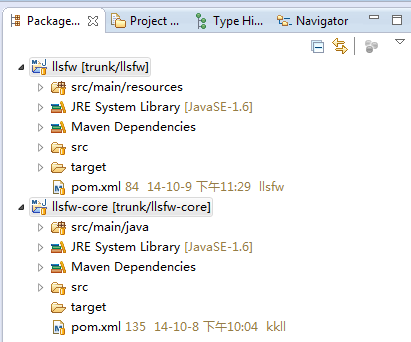
框架集成仅需llsfw-core和llsfw两个项目即可, llsfw文档可作为参考, llsfw-generator可作为具体项目的代码自动生成器

### 环境搭建

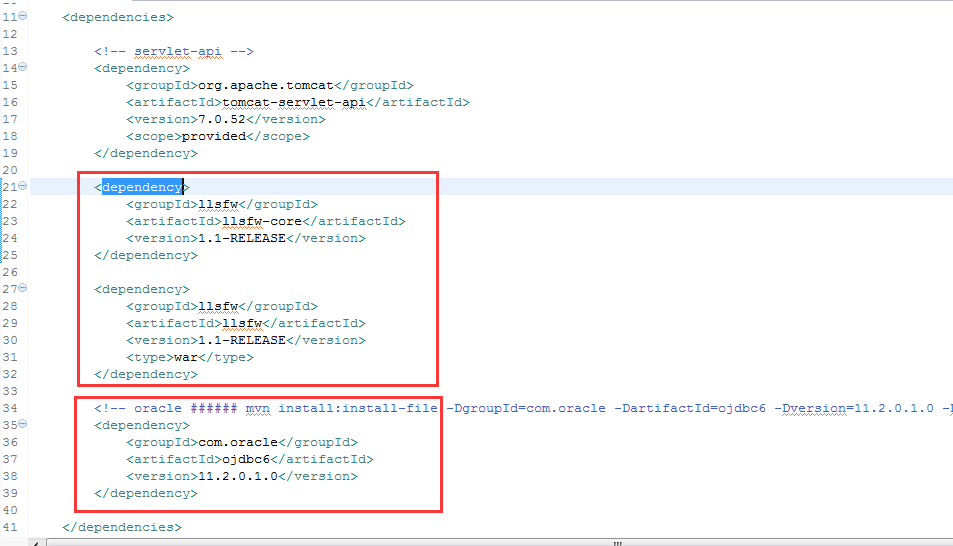
从代码库中签出框架,SVN和GIT的方式不限,这里演示的是SVN的方式,选择好对应的项目,右键选择Check out as Maven Porject,然后一直下一步直到完成即可,如下图:



经过短时间的下载和自动构建后,在工作空间就已经有了两个项目:



截止到目前,框架的构建已经完毕.后续将创建自己的业务项目,这里以llsfw-dev为例,使用MAVEN构建的方式建立好普通的WEB项目后,打开POM.XML,在dependency节点中,加入如下配置:



第一个框为llsfw框架的依赖,第二个框为项目中自定义依赖的添加,这里为oracle驱动.

然后继续在POM.xml中的plugins节点中添加如下信息:



第一个plugin节点设定的是编译级别和编码,第二个plugin中设定的是war包合并插件,用于发布的版本的时候,框架和项目的web内容合并.

到此为止POM.xml编辑完毕,后续打开web.xml,进行配置,首先配置context-param,如下图:



然后配置listener,如下图:



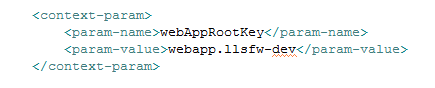
然后配置filter



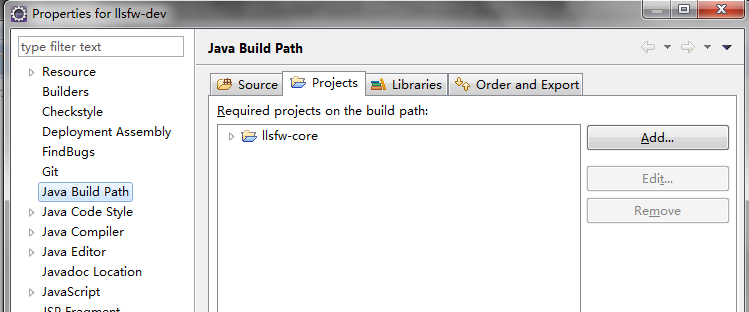
最后配置servlet



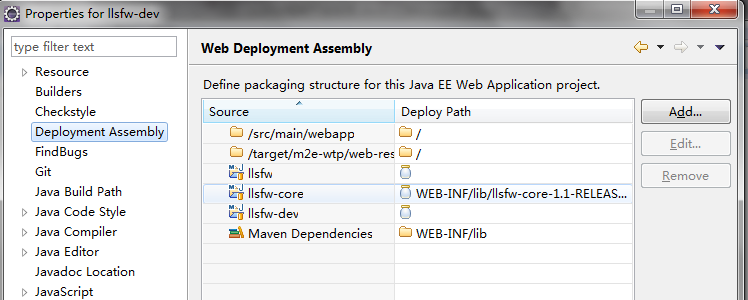
截止到目前,web.xml的内容已经编辑完毕,内容较多,主要还是配置spring和shiro.在高版本J2EE中,web.xml是可以拆分的,但是框架这里考虑到通用性,没有使用高版本的J2EE,所以这里需要配置较多的内容.

这里有一个更为简单的方式,则是拷贝llsfw项目中的web.xml到具体项目中,然后只需更改context-param中的webAppRootKey参数即可,如图:  


到这里,整个框架的集成就已经完毕了,这个时候可以Ctrl+B构建一下项目,看是否还有构建错误,如果还有错误,则检查如下2个地方,是否正确:



Java Build Path中是否有引入llsfw-core项目



Deployment Assembly中选项中确认结构是否如下.

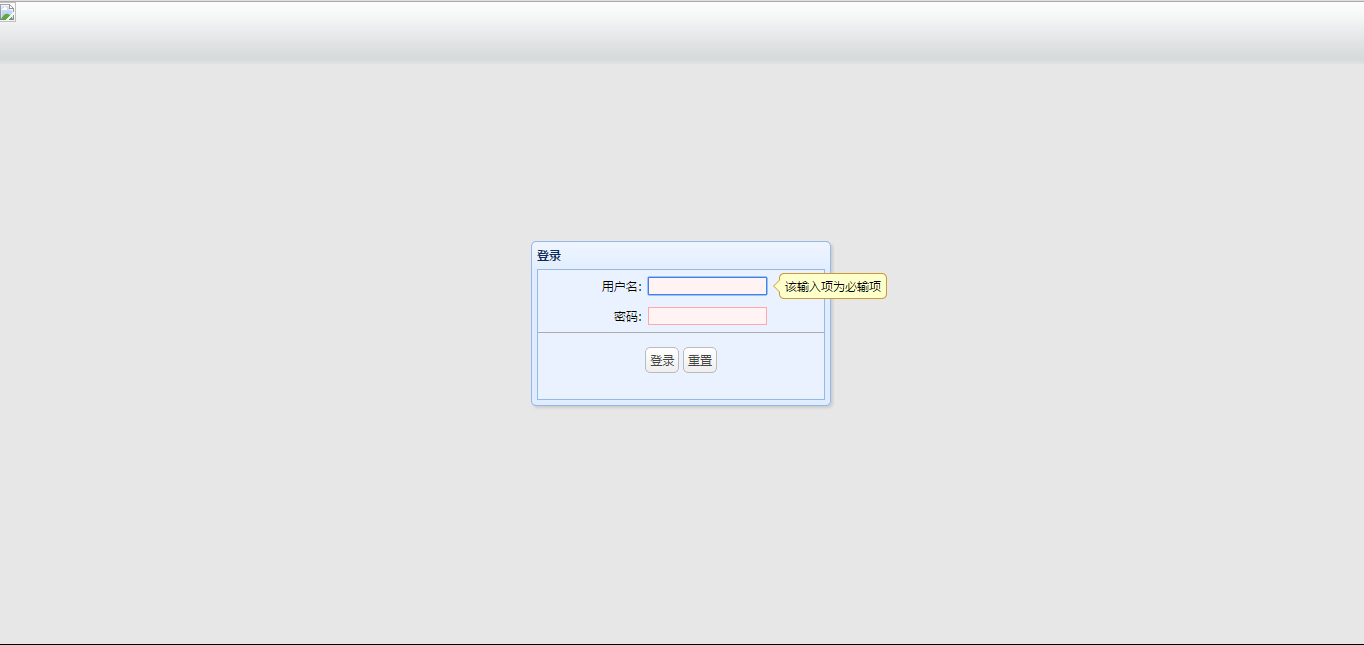
如上两点如果都确认没问题,那应该是能解决大部分问题,如果还是继续有构建错误的话,则联系框架支持人员协助解决.

后续将进行数据库脚本初始化,签出”llsfw文档”项目,打开”llsfw文档\llsfw\_db\_model\ LLSFW-CORE-DB.pdm”,使用pdm将模型导成数据库脚本,然后在数据库中执行.

然后在” llsfw文档\ quartz-2.2.1\dbTables”中找到对应数据库的脚本,然后执行,这里是” tables\_oracle.sql”

最后执行” llsfw文档\初始化数据”中的脚本,进行数据初始化.

整个过程完成后,可将llsfw-dev项目添加到tomcat中,启动tomcat,如果无错误,并且访问<http://127.0.0.1:8080/llsfw-dev>无错误的话,开发环境算搭建完毕.



## 详细介绍

从以上环境搭建的的过程中不难看出,llsfw采用的集成方式与普通框架有所区别,采用的框架与具体项目代码完全分离的方式来集成的,在这中集成方式下,好处在于基础支持与业务系统完全分离,使得业务系统层次更加清晰,也使得框架的管理也更为遍历,但是一些常规的概念将会有稍许区别,后续章节将会详细介绍.

### 配置文件总览

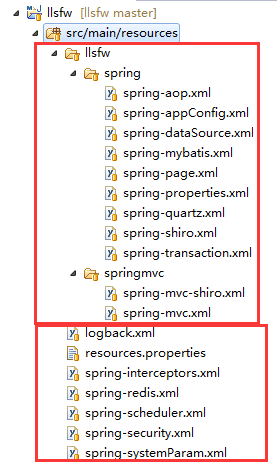
采取分离式的框架集成,也就是说整个项目会有2个(或者以上)WEB类型的工程,同时也将会有2个(或者以上)的web.xml,以及resources区域,这样就会造成一个文件覆盖的问题.

那么,在这种结构下,开发者需要记住的是,**llsfw框架的所有文件永远是被业务系统覆盖的,相同目录并且同名的文件,打包最终抓取的文件永远是业务系统的文件,而框架有的文件,并且业务系统不存在的文件,则抓取的是框架的文件**

正因为这种特性,也赋予了llsfw更加灵活多变的配置方式,相信大家在前面的”环境搭建”章节中也了解到了,整个开发环境搭建所需修改的配置文件只有2个,一个是pom.xml,另外一个就是web.xml,并且web.xml也无需手写,只需从llsfw中拷贝出来,简单修改即可.后续这种**拷贝修改配置文件**的模式会贯穿整个项目.

那么后续就来详细讲解一下各个配置文件的作用与特性.

查看llsfw项目的resources文件夹,如下图:



图中第一个框中llsfw文件夹下面的配置文件是”私有”配置文件,含义是”非本项目无需修改”,也就是说这是llsfw框架的配置文件,业务系统非必要,则无需修改.

图中第二个框中llsfw文件夹外的配置文件是”公共”配置文件,含义是”可按需修改”,也就是说集成的业务系统是可以修改此文件的.

### 私有配置文件

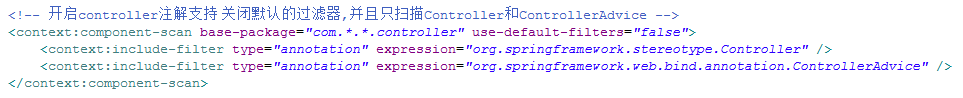
llsfw的私有配置文件对于大多数业务系统来说,是透明的,无需关心的.私有的配置文件定义了所有第三方框架的集成,以及框架默认的行为,下面将摘取对于开发者重要的配置文件进行介绍.

#### spirngmvc/spring-mvc.xml

此文件中定义了spring mvc中的大部分行为,如:



上图定义了引入拦截器配置文件

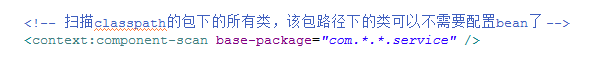


上图定义了controller扫描的的范围,例如:com.svw.part.controller,com.svw.order.controller,这些包名都是符合要求的.

此文件中还有其他的定义,如文件上传大小限制,json转换器,默认路径等,这里不一一说明了,各位开发者可到此文件中查阅.

#### spirng/spring-transaction.xml

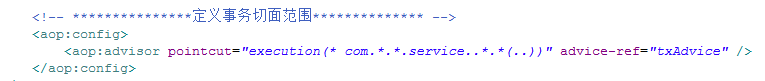
此文件中定义了整个项目的事务处理方式,详细如下:



上图定义了serivce层的路径结构,例如:com.svw.part.service,com.svw.order.service,这些包结构都是符合要求的.



上图定义了事务的传播方式,目前定义了两种模式,如果还有其他的需求,可调整此属性.



此处定义了整个项目数据库事务包裹的范围,就是service层,除service层外,其他所有代码区域都是没有事务的.

#### spring/spring-mybatis.xml

此处定义了mybatis的行为,规定了model和sqlMap和Mapper的路径,如下:



上图中sqlmap和mapper的路径都分为standard和expand,区别在于standard包下存放的全部是由生成器生成的文件,而expand为手写的内容.包结构例如:com.svw.part.sqlmal.expand,com.svw.part.mapper.expand,com.svw.part.model.expand,都是符合要求的.

#### 其他配置文件

springmvc/spring-mvc-shiro.xml

此文件定义了shiro在spring mvc中的入口

spring/spring-shiro.xml

此文件定义了shiro的详细行为

spring/spring-dataSource.xml

定义了数据源,框架默认的数据源方式为JNDI

spring/spring-quartz.xml

定义了quartz的详细行为

spring/spring-properties.xml

定义了properties的注入行为,bean中可直接注入属性文件中的值

spring/spring-page.xml

定义了分页的实现类,目前支持oracle,H2Database,MySQL

spring/spring-appConfig.xml

定义了,如LOGO,系统标题,主题,图片等信息,供在属性文件中修改

spring/spring-aop.xml

定义了一个service的切面,用于在service层出错的时候,记录出错信息

### 公共配置文件

#### logback.xml

此文件中定义了日志的行为,此配置文件中大多数属性是可以在resources.properties中设定的,所以详细的会后续介绍,一般情况下无需拷贝此文件到业务系统中,使用默认的就好.

如有特殊需求,此配置文件中可支持**将错误日志发送至指定信箱**,和**将错误日志记录到数据库中**的功能

#### spring-interceptors.xml

此文件中可定义基于spring mvc的拦截器,具体配置方式无任何差别

#### spring-security.xml和spring-redis.xml

spring-security.xml文件中定义了session的处理方式,具体如下:

##### 内存会话

如果选择为内存会话,则无需更改此配件文件中的任何内容,也无需拷贝此配置文件到业务系统中.

此模式适用于单机服务器或者指定了会话粘滞的服务器群集,优点在于简单,对于会话来说无需额外开销,缺点在于,如果群集中的某一个节点宕机,而在这个节点上的会话将全部失效,造成体验上的减分

##### 集群会话

如果选择集群会话的话,则需要将此文件拷贝到业务系统中,然后更改securityManager节点为如下配置:



同时也许要将spring-redis.xml拷贝到业务系统中,无需改动.

在llsfw中会话集群的方式采用的是持久化会话的方式,采用NoSql数据库Redis,来保存会话,从而做到会话与系统分离,达到会话集群的效果,而Redis的集群能力也让整个架构的稳定性得到了进一步的提升.

此模式的优点在于,在app server和session server都可集群的情况下是的单点故障的几率被无限降低,同时也让系统能够处理超大的访问量,缺点在于,需要额外的服务器开销,成本较高.

#### spring-scheduler.xml

此文件中定义了计划任务的静态配置方式,方式与spring集成quartz方式无差异,配置JobDetail和JobTrigger即可,可配置Simple方式和CRON方式的计划任务

#### spring-systemParam.xml

此文件中定义了系统的环境变量,可自定义添加环境变量,并且也自带默认环境变量,详情如下:

<entry key=*"PAGE\_SIZE"* value=*"50"* />

定义系统分页条数

<entry key=*"SYSTEM\_DEF\_PSWD"* value=*"123456"* />

定义用户默认密码

<entry key=*"SESSION\_NAME"* value=*"USER\_SESSION"* />

定义session名称

<entry key=*"SYSTEM\_DEF\_NAME"* value=*"SYSTEM"* />

默认用户名

<entry key=*"DETAILED\_RECORDS"* value=*"true"* />

计划任务是否详细记录执行过程

<entry key=*"LOG\_SCH\_FLAG"* value=*"true"* />

计划任务是否记录执行历史

<entry key=*"TRIGGER\_SEARCH\_INTERVAL"* value=*"60000"* />

计划任务功能自动轮询时间间隔

<entry key=*"SCHEDULED\_LOG\_KEEP\_TIME"* value=*"7"* />

计划任务变更日志保留天数

<entry key=*"SCHEDULED\_TRIGGER\_LOG\_KEEP\_TIME"* value=*"7"* />

计划任务日志保留天数

<entry key=*"APP\_LOG\_KEEP\_TIME"* value=*"7"* />

系统错误日志保留天数

在很多情况下,环境变量是需要区分环境的,如测试环境和生产环境中的某个变量的值就有可能是不一样的,所以配置文件也是要区分环境的,所以说环境变量可以配合着resources.properties一起使用,来解决此问题.

在数据库中存在TT\_SERVER\_GLOBAL\_PARAMETERS这张表,这张表里存储的就是环境变量的数据,但是数据库中的与配置文件中的是不一致的,数据库中的优先级要高于配置文件中的优先级.

#### (重要) resources.properties

此文件定义了框架的详细参数配置,也可以自行扩展,具体如下:

connection.jndi

数据库JNDI名称,默认使用JNDI链接数据库

pageResult.impl

分页实现类,目前取值为:

[com](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom).[llsfw](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw).[core](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core).[service](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service).[pagequery](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service.pagequery).MySqllPageResultService

[com](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom).[llsfw](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw).[core](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core).[service](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service).[pagequery](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service.pagequery).OracelPageResultService

security.hashAlgorithmName

加密方式,默认为MD5

security.hashIterations

加密迭代次数,默认2

security.cachingEnabled

是否缓存,默认false

security.globalSessionTimeout

session超时时间

security.sessionDaoKeyPrefix

session前缀

security.cacheManagerKeyPrefix

cache前缀

scheduler.startupDelay

计划任务启动延迟时间

scheduler.threadCount=

计划任务线程数

scheduler.driverDelegateClass

计划任务数据操作实现类,请参考quart文档

appConfig.appTitle

系统标题

headPageConfig.easyuiThemesName

系统主题

loginPageConfig.loginPageBackgroundImagePath

系统登陆界面背景图

loginPageConfig.loginPageTileBackgroundImagePath

系统登陆界面标题图

loginPageConfig.loginPageLogoImagePath

系统登陆界面LOGO

topPageConfig.topPageBackgroundImagePath

系统背景图

topPageConfig.topPageLogoImagePath

系统主页标题LOGO

mainPageConfig.mainPagePath

系统主页路径,用于自定义主页

logback.contextName

日志上下文名称

logback.logDir

日志路径

logback.pattern

日志格式

logback.maxFileSize

日志大小

redis.pool.maxTotal

redis最大连接数

redis.pool.maxIdle

redis最大进程数

redis.pool.maxWaitMillis

redis最大等待时间

redis.pool.testOnBorrow

redis测试

redis.hostname

redis主机名称

redis.port

redis端口

redis.password

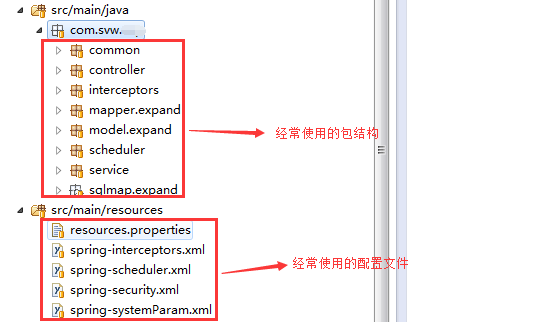
redis密码

对于此配置文件,通常会被用来作为系统的主配置文件,也是需要区分测试与生产区别的.

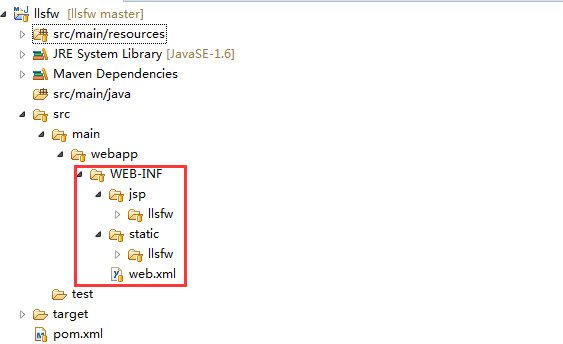
#### llsfw-core包结构介绍



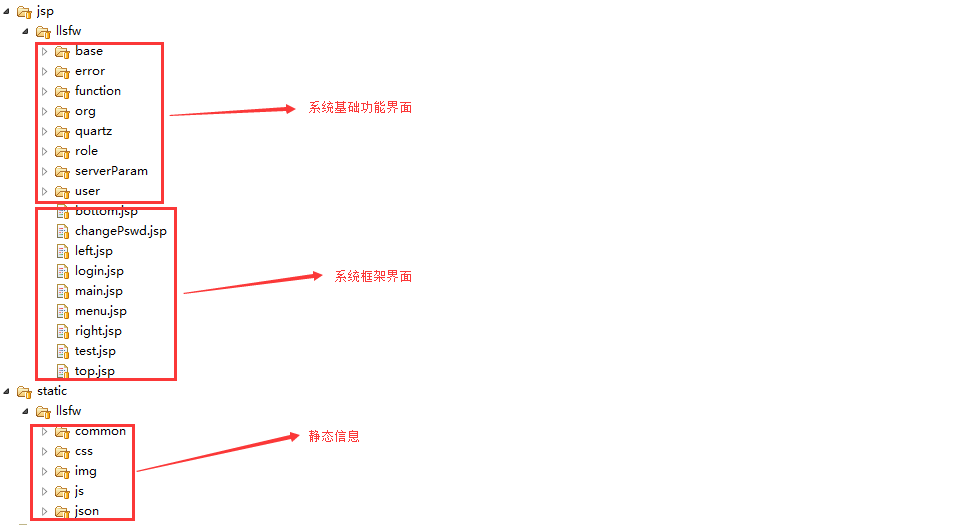
在业务系统中,开发者可创建包结构的范例,如下:



#### llsfw文件夹结构介绍



上图红框内的前端文件全部放在WEB-INF目录下,以保护静态资源,在此之下分为JSP和STATIC两个文件夹,分别为动态页面和静态文件,分别在下级的文件夹是按项目划分,上图的则是llsfw框架自带的功能目录了,如果业务系统的名称叫partOrder的话,那则可以在jsp和static下分别建立叫partOrder的文件夹,然后将项目所有相关的文件全部放入此文件夹内,达到一个内容区分的目的.

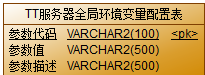


上图为文件夹展开后的分布和功能区域划分.

## 开发范例

本章节将带着各位开发者做做一个简单的功能开发,来更加了解llsfw框架,这里我们用框架中的”环境变量维护”来讲解.

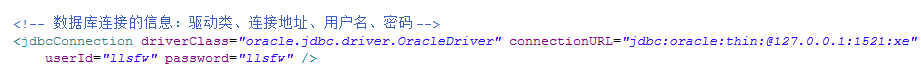
### 表结构设计



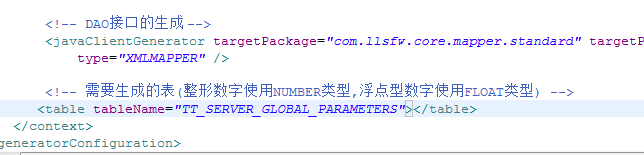
三个字段,参数代码为主键

### 生成model,sqlMap和Mapper

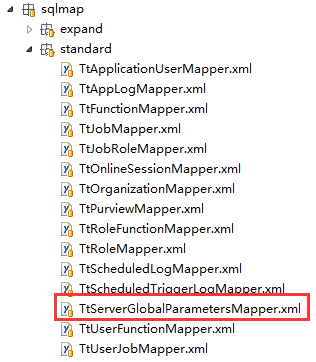
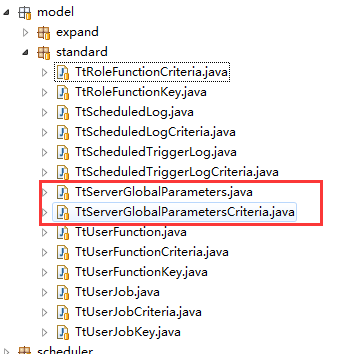
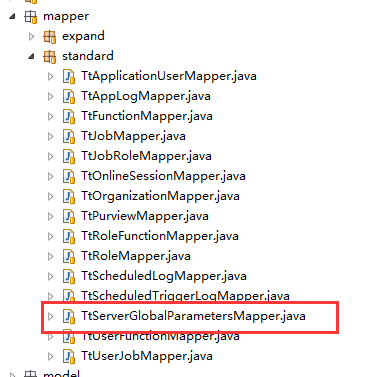
修改llsfw-generator项目中的/llsfw-generator/src/main/resources/llsfw/generatorConfig/generatorConfig.xml 配置文件,将



调整成自己数据库的信息,然后在结尾做如下修改



添加表明,同时需注意javaClientGenerator, sqlMapGenerator, javaModelGenerator三个节点中配置的targetProject和targetPackage路径是否正确,确定完毕后,运行/llsfw-generator/src/main/resources/llsfw/generatorConfig/generatorConfig.bat,完成后,在项目将会出现,如下文件:

如上三张图的信息表明为数据库访问层已经创建成功.

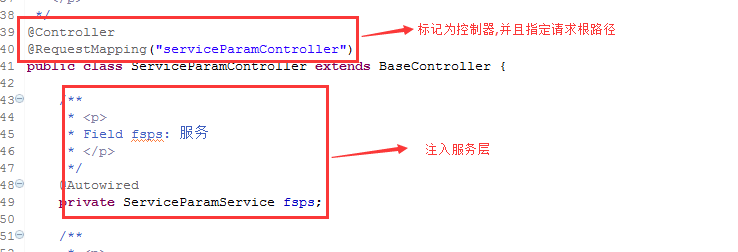
### 创建service层

创建包com.llsfw.core.service.serverparam,并且创建类ServiceParamService,在类名上标记@Service,并且注入刚刚生成的Mapper,如下图:



### 创建controller层

创建包com.llsfw.core.controller.serverparam,并且创建类ServiceParamController,在类名上标记Controller和RequestMapping,并且注入Service层,如下图:



到这里,后台的结构就已经,搭建完成,后续搭建前端结构

### 创建前端结构

在llsfw项目中创建如下两个结构:

/llsfw/src/main/webapp/WEB-INF/jsp/llsfw/serverParam

/llsfw/src/main/webapp/WEB-INF/static/llsfw/js/serverParam

并且分别建立serverParam.jsp和serverParam.js.

**注意,尽量做到一个JSP对应一个JS,做好前端的VIEW和CONTRILLER分离**

到这里,前端的结构已经搭建完毕.

### 开发查询功能

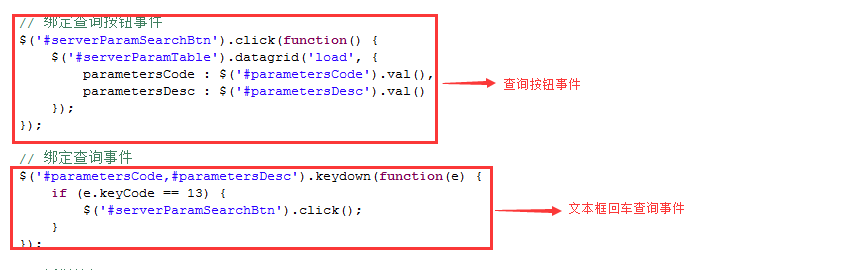
准备工作已经完成,首先我们编辑serverParam.jsp,如下图



以上为jsp的编辑成果,主要的工作是定义组件,后续渲染组件的工作交给JS完成,如下图:



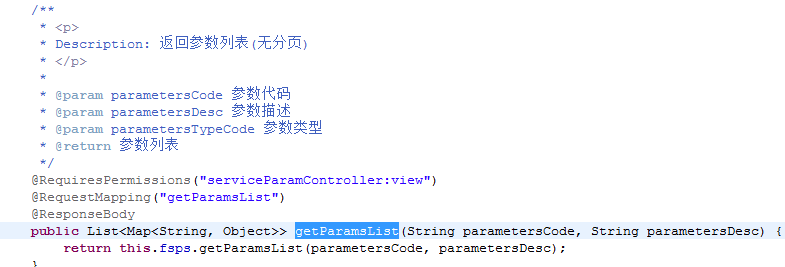
此段js是构造表格的JS



到此,查询相关的的前端代码已经完成,后续,写后端的代码:



在controll中定义功能入口,此入口后续要配置到function中的



在conroller中定义查询方法,调用service中的查询方法,并且制定权限为view

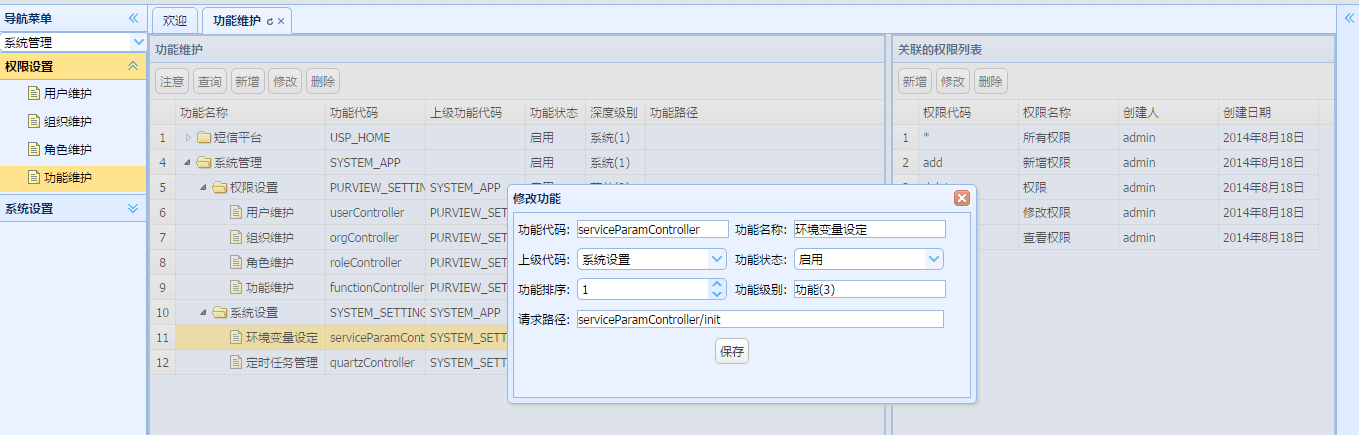


在service中定义查询方法,此处为直接在代码中拼接的SQL语句,正常情况应当将SQL语句配置到SQLMAP中,并且此处是没有分页的,如果需要分页的话可调用

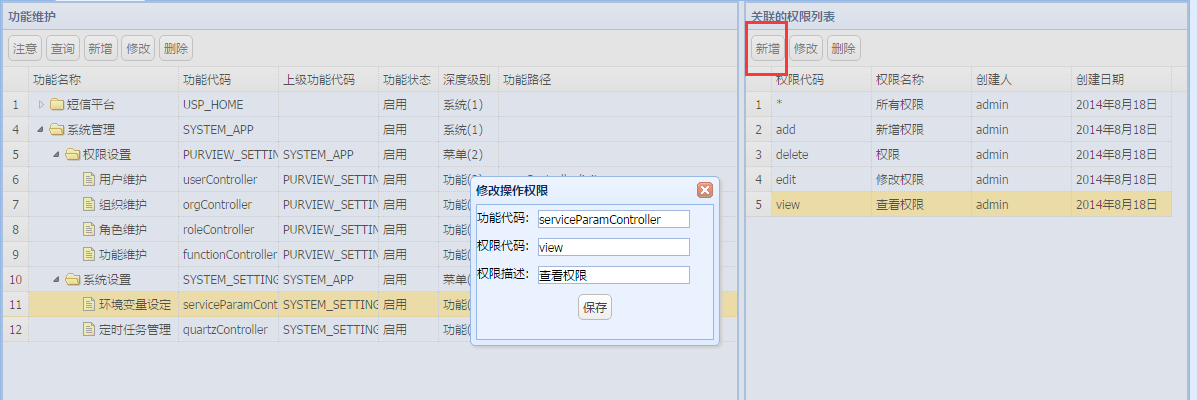
[PageResult](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service.pagequery%7BIPageResult.java%E2%98%83IPageResult~pageQuery~QString;~I~I%E2%98%82PageResult) [com](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom).[llsfw](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw).[core](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core).[service](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service).[pagequery](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service.pagequery).[IPageResult](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service.pagequery%7BIPageResult.java%E2%98%83IPageResult).pageQuery([String](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service.pagequery%7BIPageResult.java%E2%98%83IPageResult~pageQuery~QString;~I~I%E2%98%82String) sql, int pageSize, int curPage) throws [Exception](eclipse-javadoc:%E2%98%82=llsfw-core/src%5C/main%5C/java%3Ccom.llsfw.core.service.pagequery%7BIPageResult.java%E2%98%83IPageResult~pageQuery~QString;~I~I%E2%98%82Exception)

这些代码完成后,即可运行.

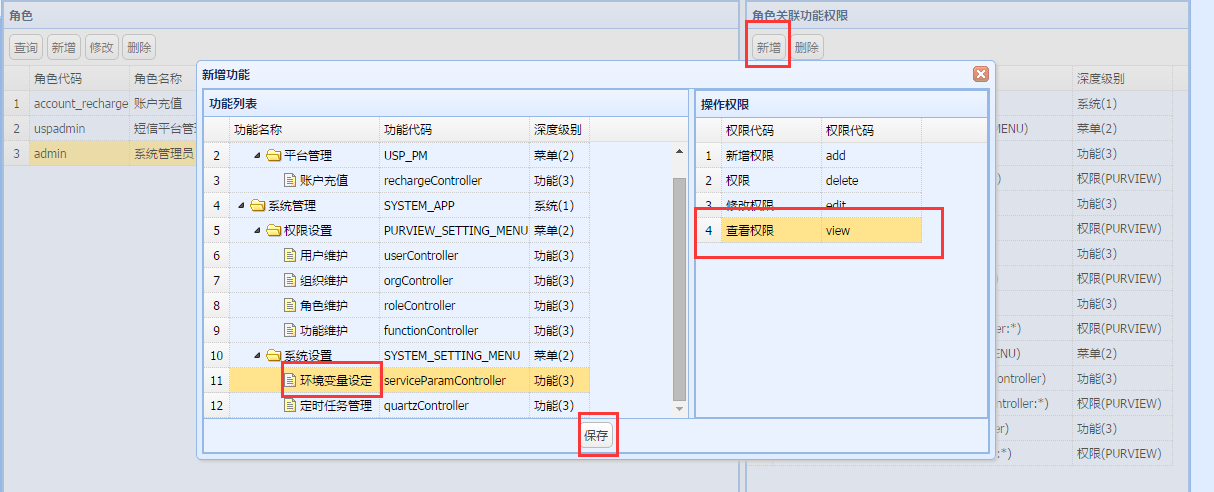
但是现在系统中并没有此功能,所以暂时无法看到,后续我们配置功能菜单



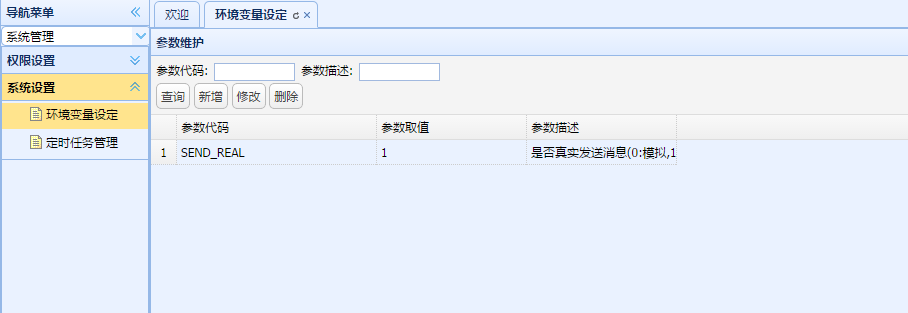
在合适的菜单下添加功能,然后在添加相应的操作权限



这样,我们的功能就定义好了,后续在角色管理中,将此功能加入admin角色中



这样管理员就有此功能的查看权限了,

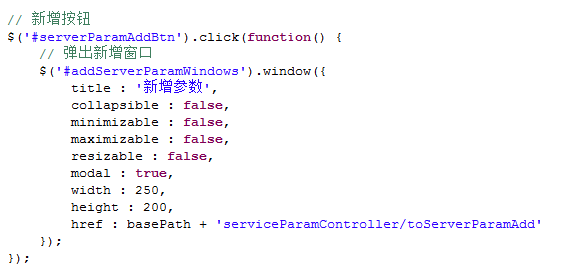


### 开发新增功能

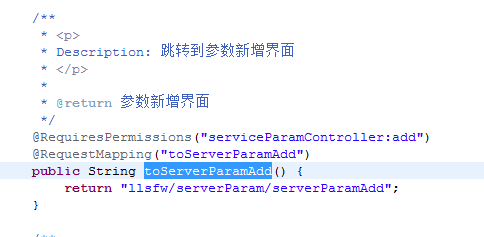
准备工作已经完成,首先我们编辑serverParam.jsp,如下图



设定一个用于新增的框,然后编辑js

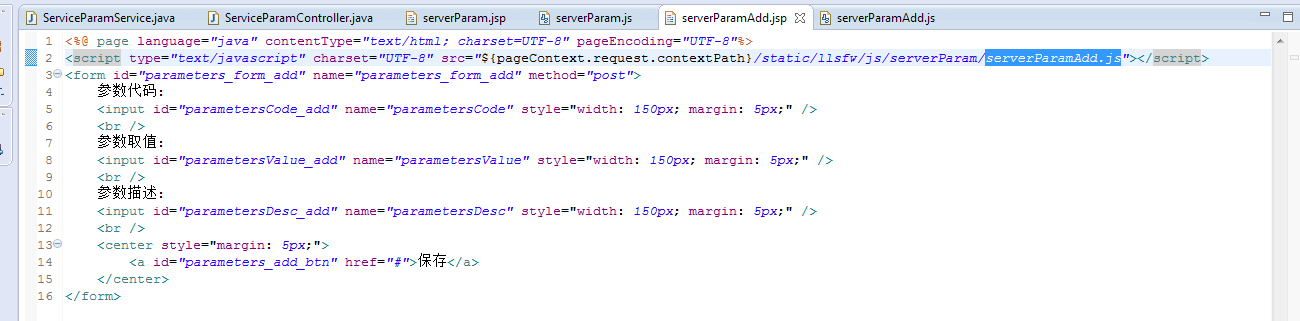


绑定新增按钮事件,后续跳转到controller



设定跳转到新增页面的控制器,设定权限为add

创建/llsfw/src/main/webapp/WEB-INF/jsp/llsfw/serverParam/serverParamAdd.jsp和 /llsfw/src/main/webapp/WEB-INF/static/llsfw/js/serverParam/serverParamAdd.js,并且做如下编辑:

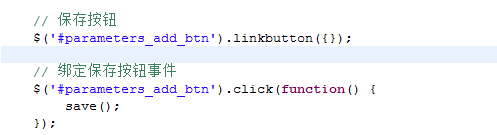


设定新增界面,然后编辑JS



llsfw自带一些基础校验,可从

/llsfw/src/main/webapp/WEB-INF/static/llsfw/js/validatebox.js中查看到



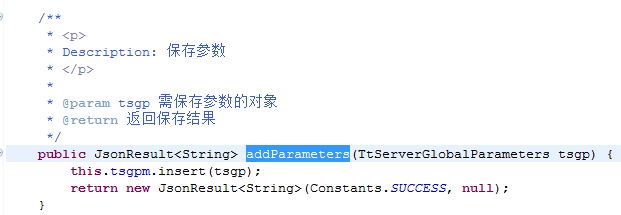
然后设定保存按钮事件



后续编写保存方法:



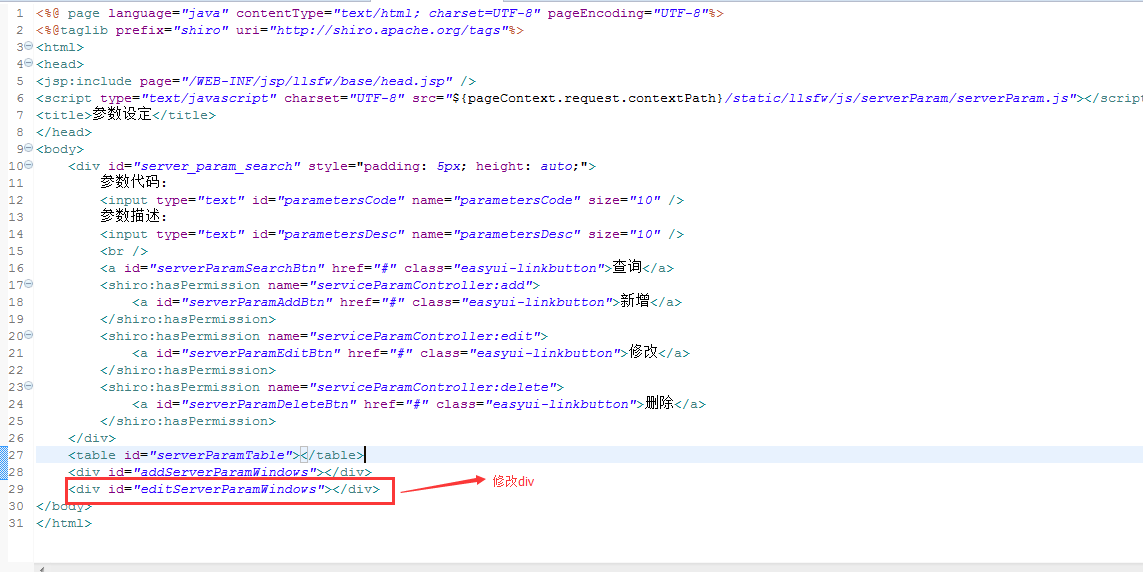
然后编写service方法



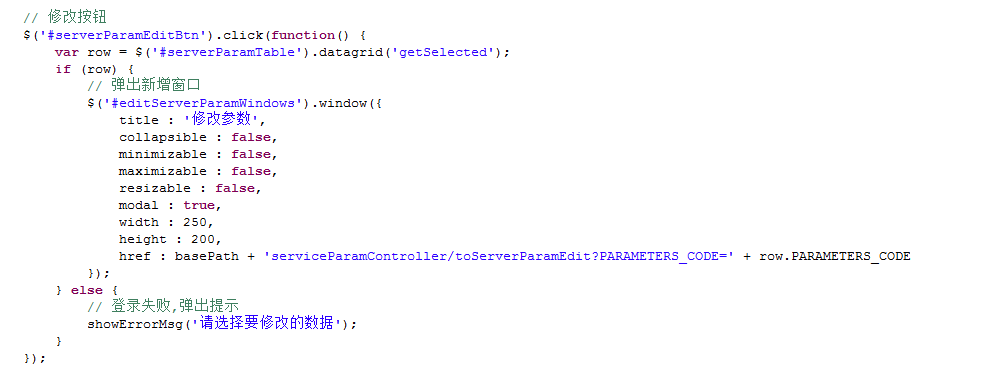
service中调用dao的方法,完成保存的操作.

### 开发修改功能

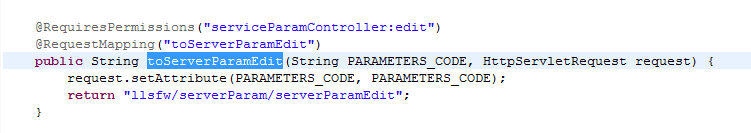
准备工作已经完成,后续我们开发修改功能



然后编辑js文件



添加修改按钮的事件,后续编写controller,



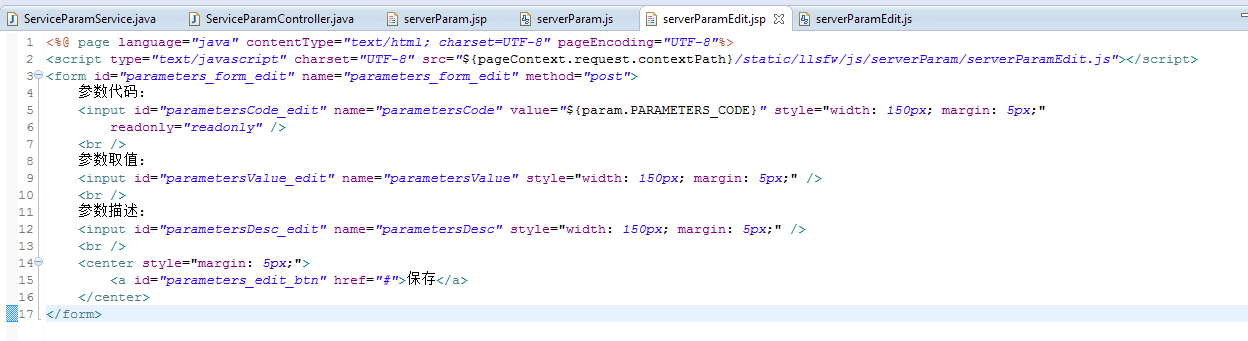
设定跳转到修改界面的控制器,并且设定权限为edit

然后创建如下2个文件

/llsfw/src/main/webapp/WEB-INF/jsp/llsfw/serverParam/serverParamEdit.jsp

/llsfw/src/main/webapp/WEB-INF/static/llsfw/js/serverParam/serverParamEdit.js

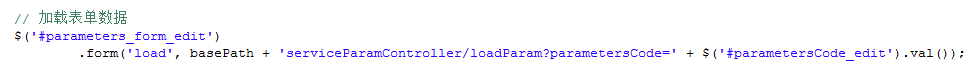
并且编写serverParamEdit.jsp



后续编写serverParamEdit.js

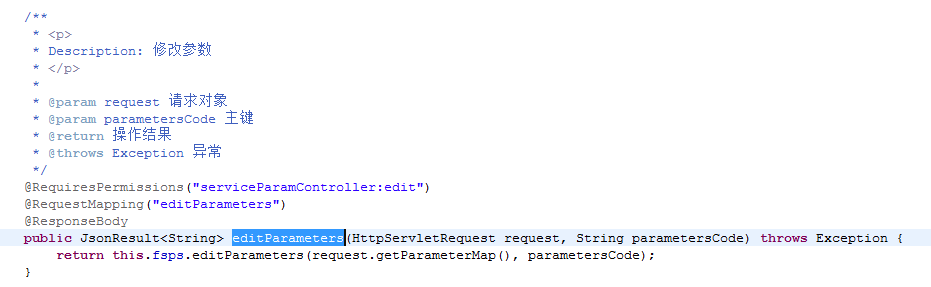


首先是表单初始化

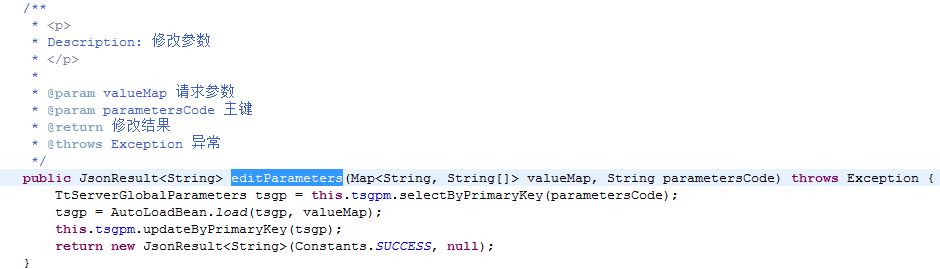
然后设定表单初始化的地址



最后设定保存方法,后续进行服务端开发,



controller设定保存方法



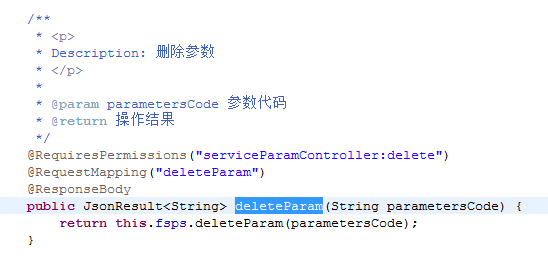
service完成修改动作

### 开发删除功能功能

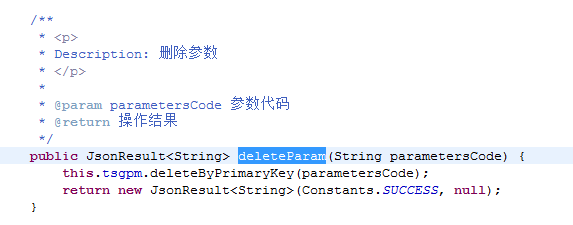
准备工作已经完成,后续我们开发删除功能



设定删除按钮的时间,后续开发服务端功能



Controller定义删除方法



Service完成删除动作

## 自带功能介绍

本章节将带着各位了解一下llsfw框架中自带功能的用途跟用法

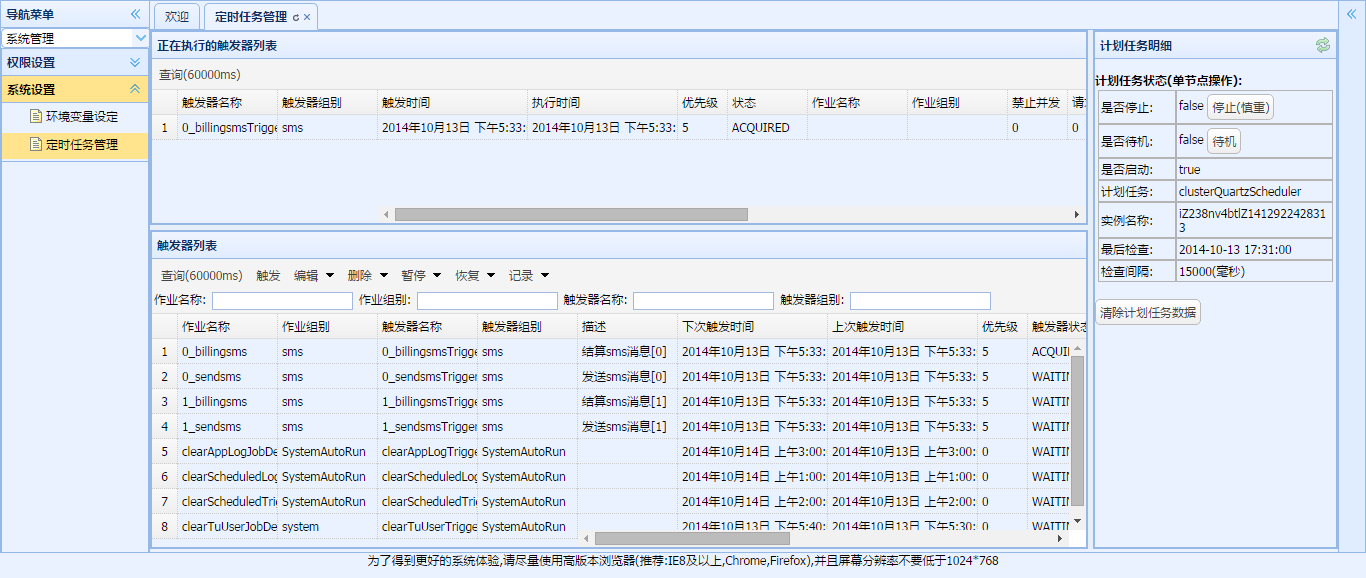
### 环境变量设定

此功能主要是定义环境变量,与systemParam.xml中定义的数据内容一致,区别在于优先级,此功能配置的数据优先于文件中配置的数据.

以做到,在系统运行的过程中,动态调整参数,而无需调整代码的功能.此功能没有默认数据,并且提供了增,删,查,改的功能,需要注意的是,参数的值一定要是设定范围内的,以免引起不必要的错误.

### 定时任务管理

此功能主要是管理系统中的定时任务,系统中提供2种配置方式,分别为”动态”和”静态”,在前面配置文件讲解中,已经了解了”静态”的方式,这里则介绍动态的方式,首先看一下主界面:



主界面分为三块区域,左上的表位为正在执行的任务,左下的表格为触发器清单,右边的信息为计划任务集群和触发器明细.

点击某一个触发器,会刷新右边的明细信息,

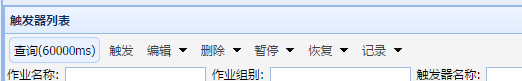
在右边有几个按钮



分别可控制计划任务的”停止”,”待机”,”启动”

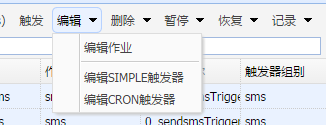
需要注意的是这些操作都是单节点操作,如果在集群环境下的话,切记要分开单个节点依次操作,才能完全停止.

另外一个则是”清楚计划任务数据”,执行此操作需要操作密码.

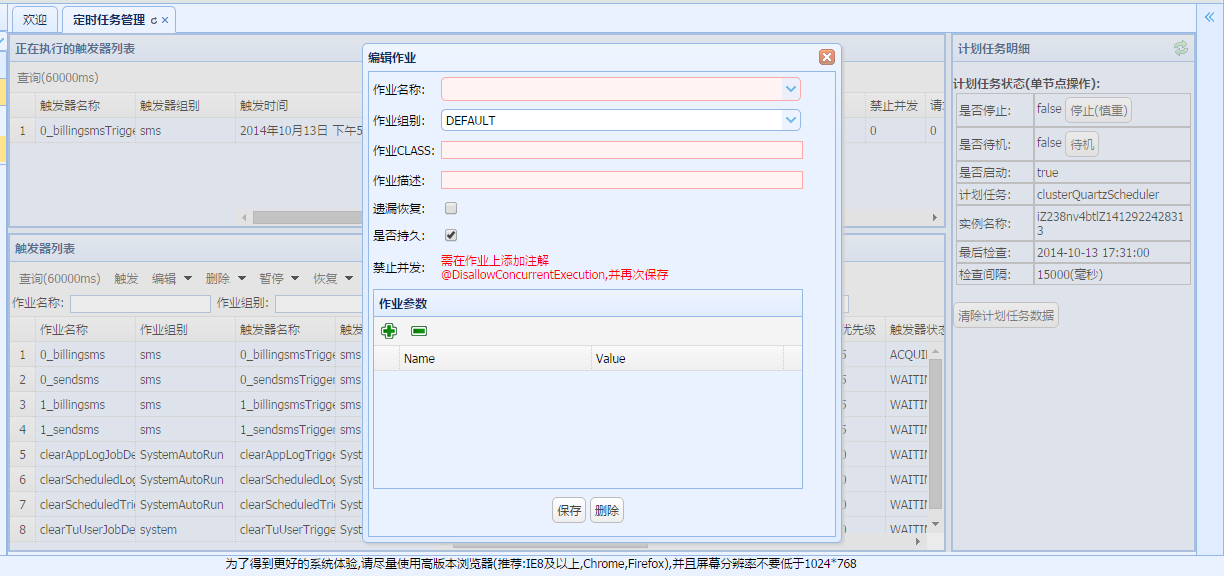


在触发器列表中有一排操作按钮,首先是”触发”,可以手动的让某个触发器立即执行一次

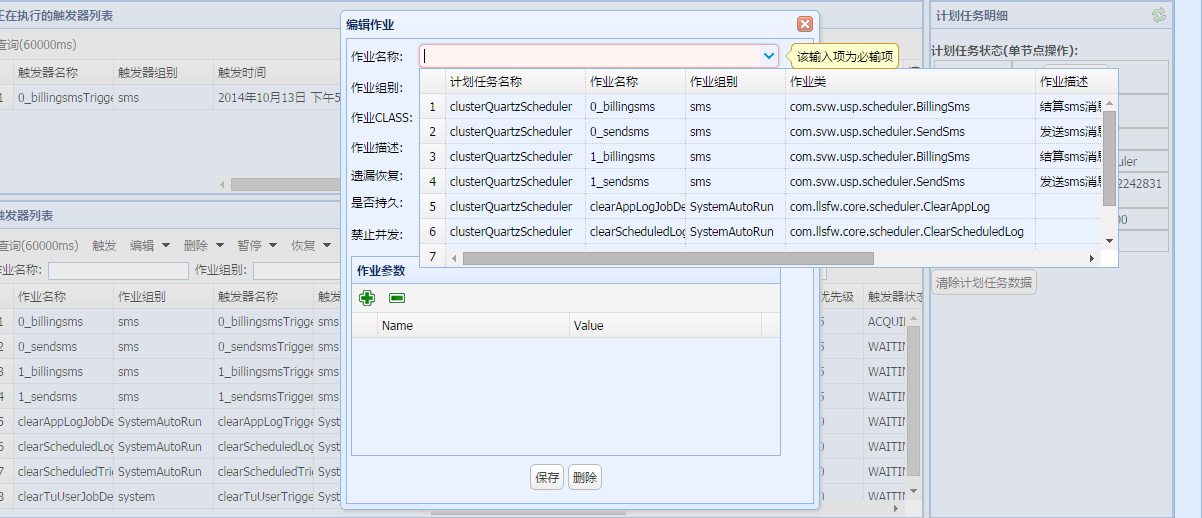
然后是编辑,编辑为一个菜单按钮具体如下:



点击编辑作业

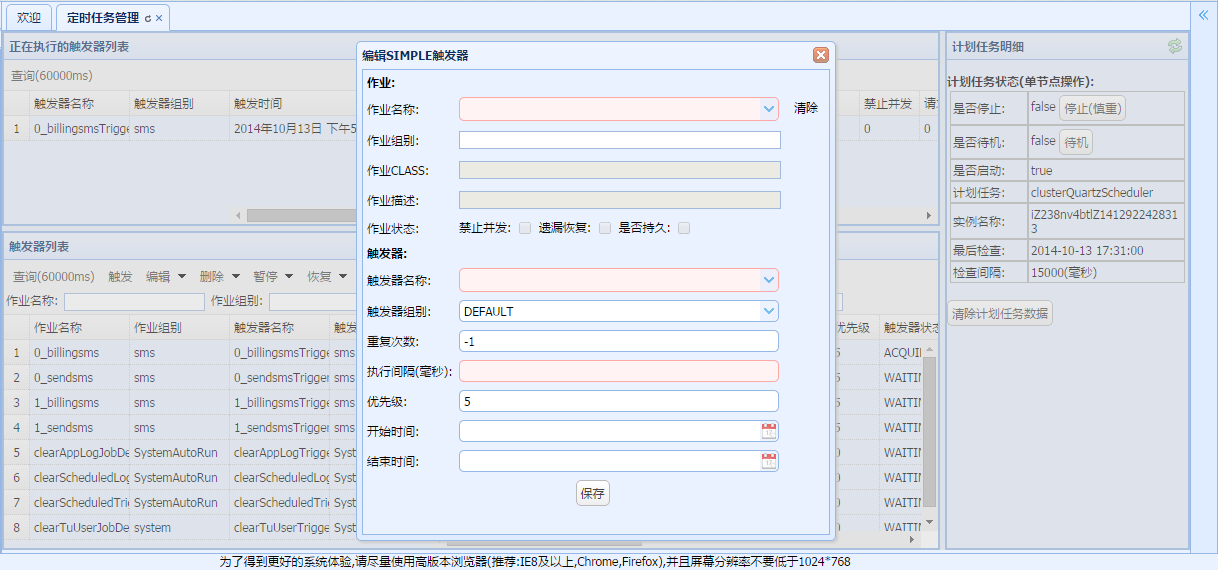


可弹出编辑作业的框,在此可创建一个JOBDETAIL,也可以通过下拉表格

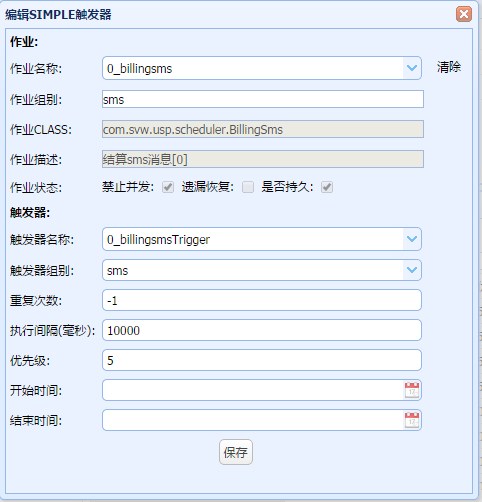


来修改一个jobdetail和删除一个jobdetail

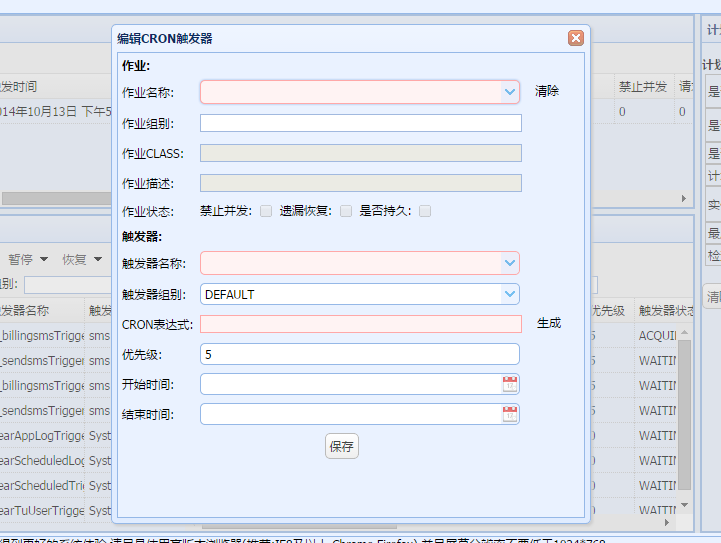
选择编辑SIMPLE触发器,将弹出如下界面:



可以先选择一个要创建触发器的jobdetail,然后在下方的表格中新增或者修改某个trigger

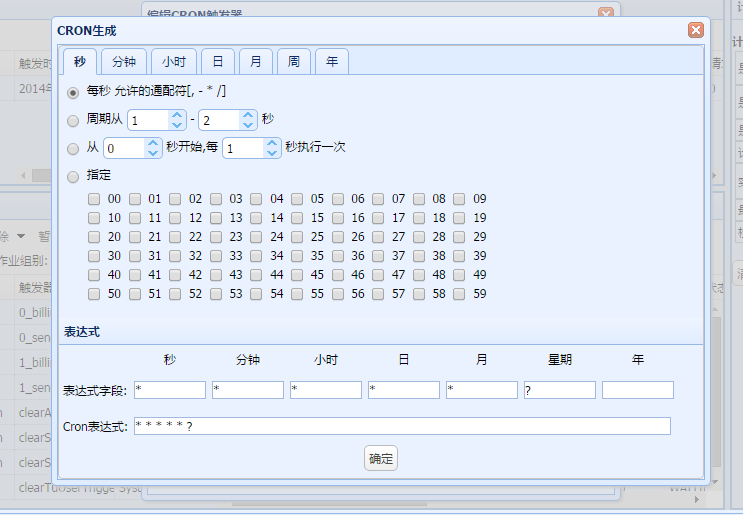


点击”编辑CRON触发器”弹出如下界面:

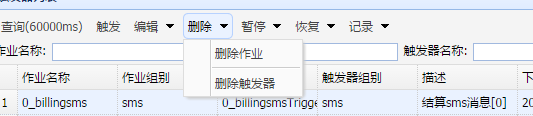


同样可选择一个jobdetail,然后在进行trigger的新增或者修改.

点击又下方的生成,可弹出cron生成界面

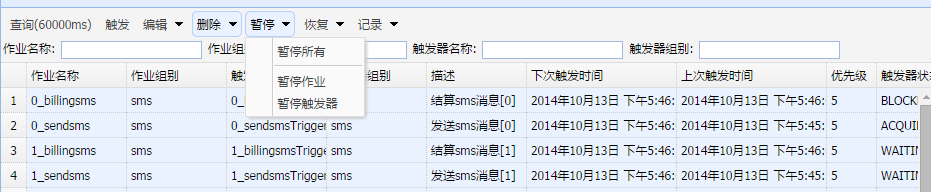


删除菜单中



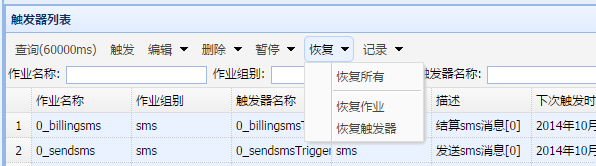
选择删除作业,可彻底删除此jobdetail,选择删除触发器,可删除jobdetail的其中一个trigger

暂停菜单中



选择暂停所有,可暂停所有的jobdetail,选择暂停作业,可暂停指定的作业,选择暂停触发器,可暂停作业的某一个触发器

恢复菜单中

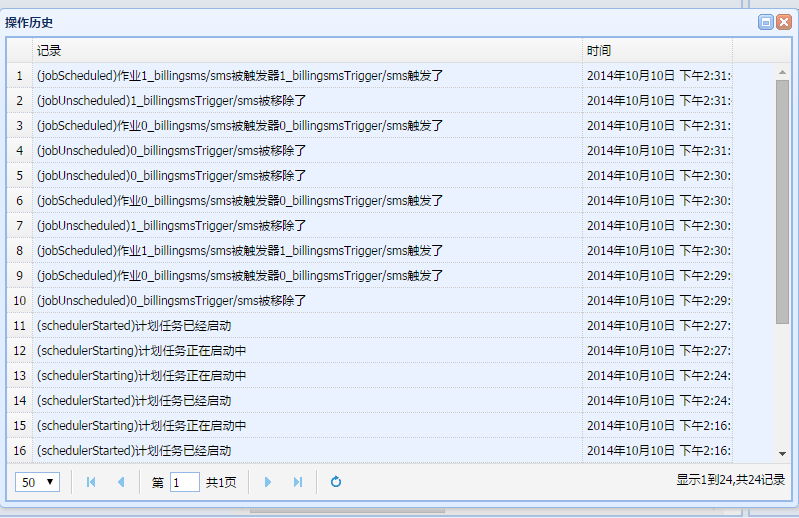


选择恢复所有,可唤醒所有的jobdetail,选择恢复作业,可唤醒指定的作业,选择恢复触发器,可唤醒作业的指定的触发器

记录菜单中

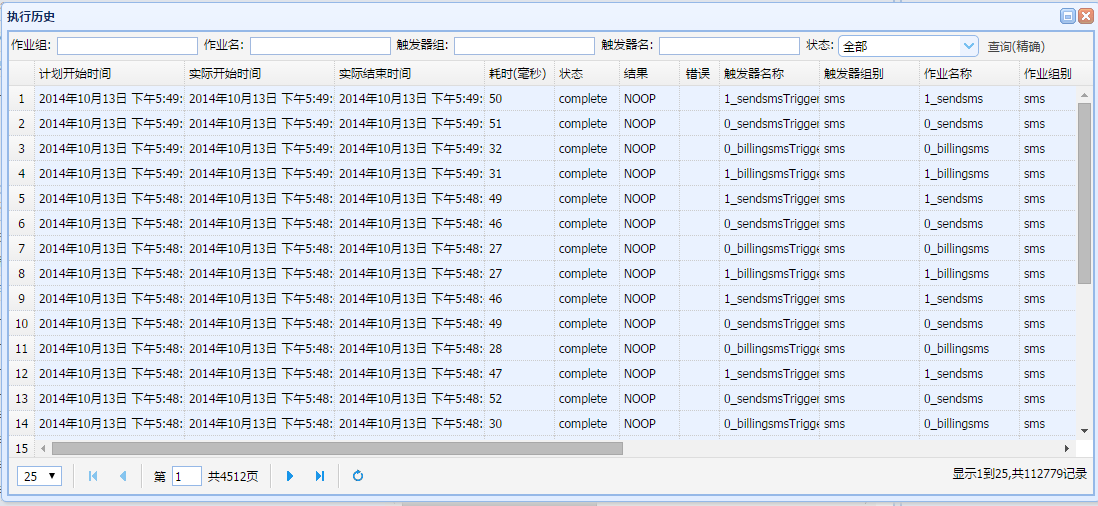


选择操作历史可弹出界面



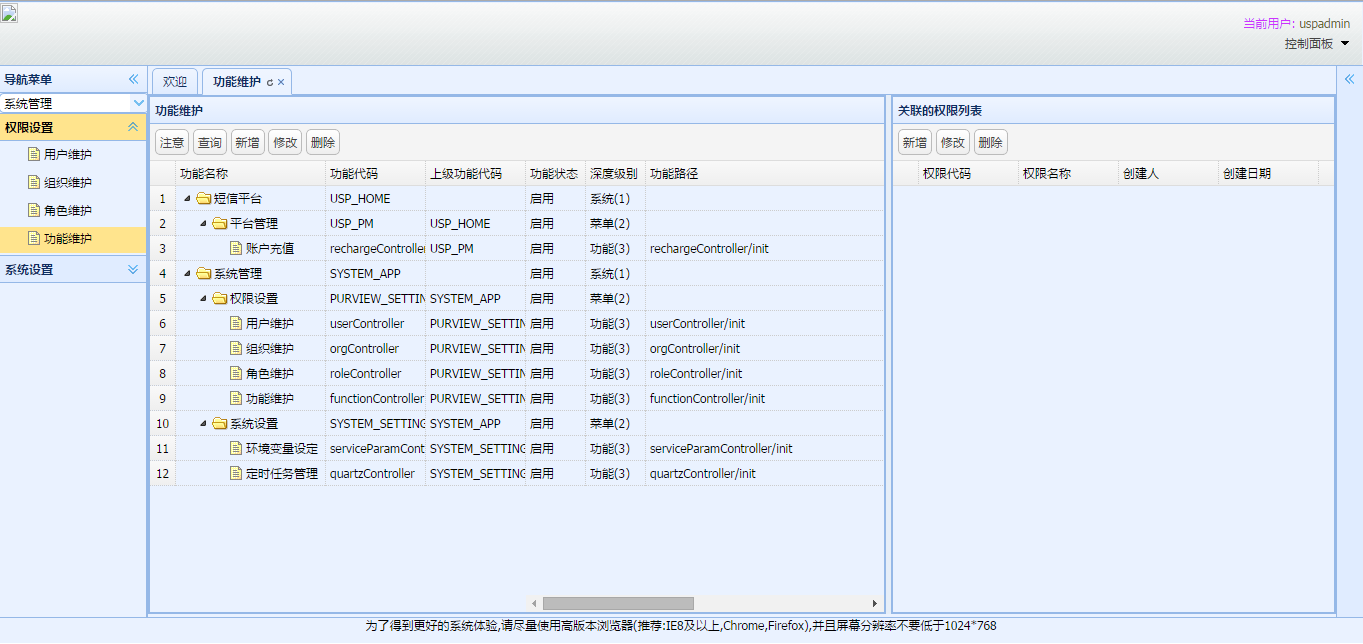
此表格中记录了,所有的计划任务操作历史

选择执行历史可弹出界面



此界面中展示的数据是计划任务执行历史

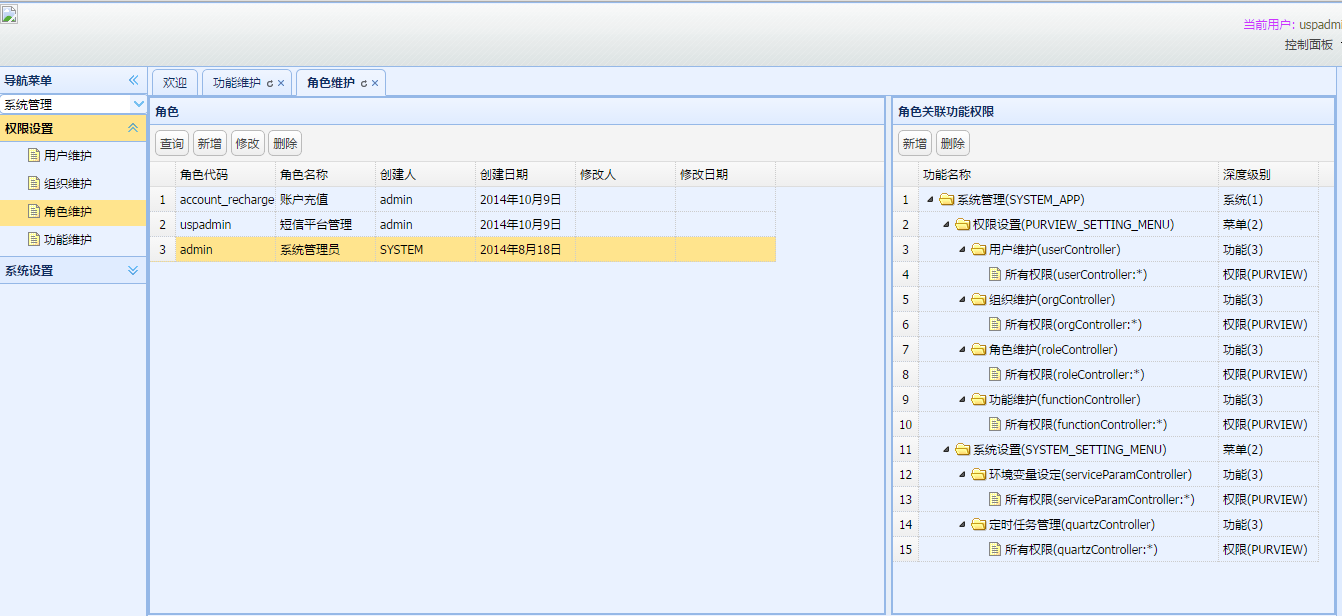
### 功能维护

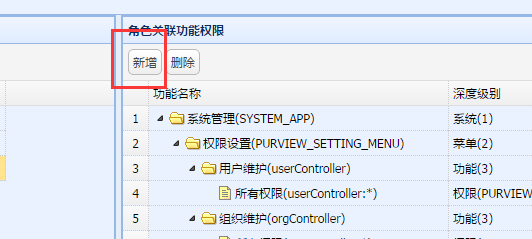
此功能主要是管理系统中的所有菜单,功能以及操作权限,主界面:  


左边的树形表格为功能数据,主要分为3层,第一层为系统,第二层为菜单,第三层为功能,点击具体功能,右边的表格会列出此功能的操作权限.

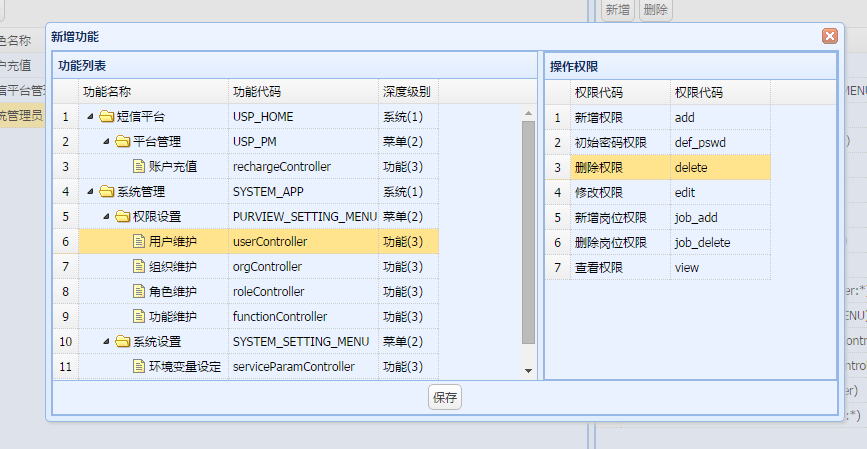
### 角色维护

此功能主要是管理系统中的所有角色,以及角色所拥有的权限,主界面:

左边的表格为角色列表,右边的表格为当前选择角色所拥有的功能,点击



会弹出如下界面:



选择指定的功能和指定的权限,点击保存,则可为角色分配操作权限

### 角色维护

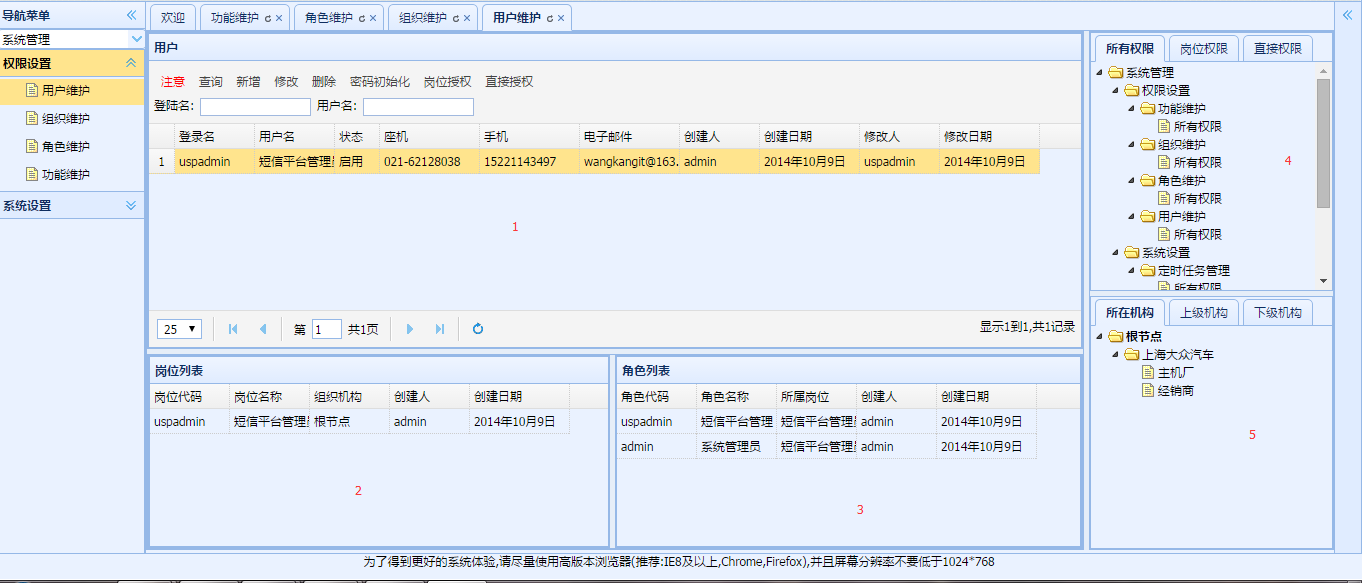
此功能主要是管理系统中的组织机构,岗位等信息,主界面:



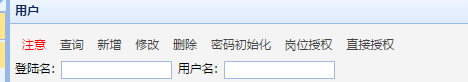
左边的树形菜单为组织机构菜单,可做增,删,差,改等操作,右上方的表格为岗位表格,可在指定的组织机构下创建岗位,并且可为岗位添加相应的角色

### 用户维护

此功能主要是管理系统中的用户信息,主界面:



标记为1的区域为用户列表,标记为2的区域为用户所属的岗位,标记为3的区域为用户所拥有的角色,标记为4的区域为用户所拥有的功能权限,标记为5的区域为用户所在的组织机构



工具栏上面的操作出了增,删,查,改的功能外,还有”密码初始化”

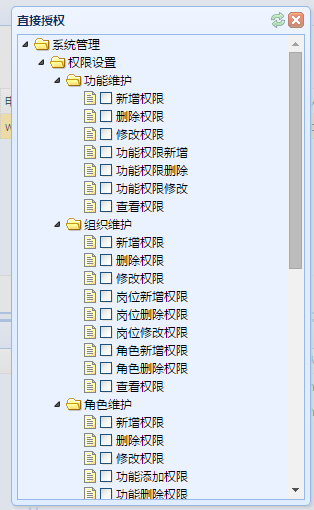
点击后,可将用户的密码恢复初始,

点击”岗位授权,可”弹出如下界面:



可选择指定机构下的岗位,打钩代表授权,不打钩代表取消授权

点击直接授权,可弹出如下界面:



勾选则代表授权,不勾选则代表取消授权.