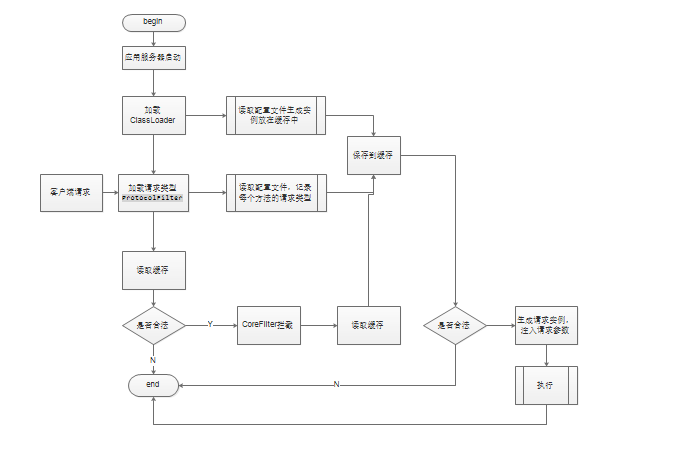
# 前言

simpleMVC框架是基于servlet3.0和jdbc组建的一个快速开发框架。基于少一些配置文件，少一些无用的类。如果你熟悉或了解springMVC和Hibernate您将很快上手。本文将介绍本框架的基本结构和如果使用本框架进行开发。

# 系统流程图



启动服务器时，自动读取配置文件，加载对应的Action层，并记录每个方法的请求类型，并未每个请求方法记录对应的参数，把结果根据对应的key，保存到缓存中。当请求来临的时候，首先判断此请求是GET还是POST，如果合法则被CoreFilter拦截，从缓存中查找要执行的单元，生成对应的实例，并注入相应的参数，执行相关逻辑。

# 注解介绍

1. @Action

Action注解用着编写的action类头部，属性namespace=”XX”用来区分命名空间。

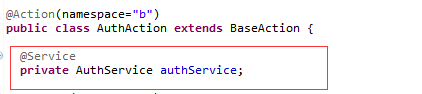
如：



在http请求的时候，如下格式：http://hostname/b/xxx.do

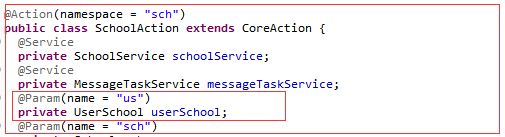
1. @Service

此注解用在Action引用的Service类，如：



1. @Param

此注解表示Action处理在从页面请求时带过来的参数，属性name=”xx”表示参数的别名。

如下图：  
 

表示从页面请求过来的参数如下，如：

<http://hostname/sch/xxx.do?us.schName=xxx&us.userName=xxxx>

注意：这里us表示的时UserSchool的一个别名，框架收到这个参数后会自动转换为UserSchool这个类，并且schName字段是类UserSchool的属性。

1. @Table

此注解用在实体类上，表示这个实体类和哪一个数据库对应，属性name表示数据库名。如下图：

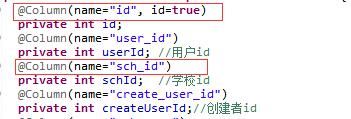


1. @Column

此注解用在实体类的属性上，表示和数据库表字段的对应。

属性name=”XX”表示表字段，属性id表示是不是主键，默认值false

如下图：



1. @Redirect

此注解表示重定向，属性name表示重定向的key，属性url表示重定向的地址。

此注解配合@Method注解使用。

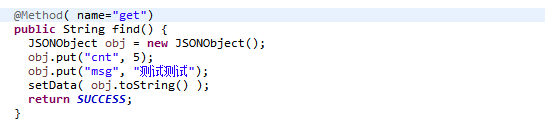
1. @Forward

此注解表示跳转，属性name表示重定向的key，属性url表示重定向的地址。

此注解配合@Method注解使用

1. @Method

此注解用在Action类的方法上，用来标记URL中的请求方法。属性name=”XX”表示URL中请求的\*.do的名字。如下图：

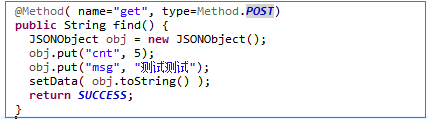


在URL请求中如：http://hostname/namespace/get.do

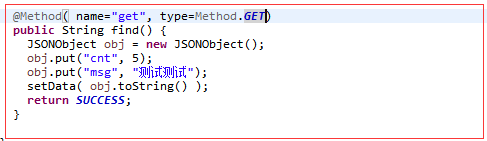
属性type表示此方法允许使用的请求方式，type=Method.POST表示只允许POST请求，type=Method.GET表示只允许get请求，type=Method.ALL表示都允许，不填默认为ALL。

如下图：

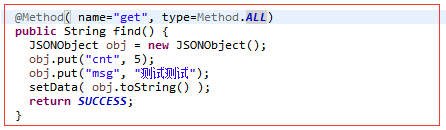
POST:



GET:



ALL:



属性redirects表示要重定向的集合，forwards表示跳转的集合，如下图：

# 基础类介绍

1. **BaseAction**

编写Action类的时候需要继承，BaseAction提供了

setResult(int status, String errMsg)

setResult(int status, String data, String errMsg)

setResult(int status, JSONObject data, String errMsg)

setData( String data )

以上方法用于在ajax调用时使用。

setForword( String key )

setRedirect( String key )

以上2个方法用于跳转到其他页面使用

1. **BaseService**

编写service时需要继承此类，此类引入了CoreDao类。

使用时可以直接用this.dao即可。

1. **ORM类**

ORM类时把对应的Table类转换成对应的基本sql，Table类就是带有@Table注解的类

如：User类带有注解@Table(name=”t\_user”), 使用时ORM orm = new ORM( new User());

可以获取基本的sql语句，省去手工写sql。

a) **select** 语句

orm.getSelectSQL()

b)**insert**语句

orm.getInsertSQL()

c) **delete**语句

orm.getDeleteSQL()

d) **update**语句

orm.getUpdateSQL()

注：需要的条件需要在自己拼写

1. **CoreDao**

此类用作操作数据库的基本类，提供了如下几个方法。

*a1)* **save**(Connection con, String tableName, String[] columns, Object... params)

tableName表示表明，columns表示插入的字段，params表示字段的值。

如：save(con,”t\_users”, new String[]{“name”,age}, new Object[]{“Tom”,10});

*a2)* **saveBean**(Connection con, Object obj)

表示直接插入到数据库中，obj必须为带有@Table的类

说明:如果插入的表的主键是自增类型，则返回主键值

如果不支持自增类型，如果成功返回1，否则返回0.

*a3)* **saveBeans** ( Connection con, List<Object> objs, int loop )

表示批量插入数据objs存放的为@Table类，loop表示多少数据提交一次事物

*a4)***saveBeans**(Connection con, String tableName,

String[] columns, List<Object[]> params, int loop)

*b1)***deleteBean**(Connection con, String sql, Object... params)

表示删除一条数据，sql，parmas表示sql里的参数

*c1)***int** **update**(Connection con,

String tableName,

String[] columns,

String[] whereColumns,

Object... params

)

*c2)***int** **updateBean**(

Connection con,

Object obj,

String whereSql,

Object... whereParams)

表示更新数据，obj必须为@Table类， whereSql表示条件语句，

whereParams表示条件语句的值

*c3)***int** **updateBean**(Connection con, String sql, Object... params)

表示根据sql来执行更新操作

*c4)***updateBeans**(

Connection con,

String tableName,

String[] columns,

String[] whereColums,

List<Object[]> params, **int** loop)

表示批量更新数据，columns表示更新的字段，whereColums表示条件字段，

params存放的是前2者对应的值。

*d1)*<T> List<T> **selectBeans**(Connection con, String sql, Class<T> clazz, Object... params)

表示通过一个sql查询封装成一个类，注，sql里的字段别名必须和类T里的属性一直。例如：

sql = select user\_id as userId, user\_name as userName from t\_user,

这里的userId和userName对应类User的属性，params表示参数。

*d2)*<T> **T** **selectBean**(Connection con, String sql, Class<T> clazz, Object... params)

查询一个，并且封装成对象，sql和d1）一样情况。

*d3)List***<JSONObject> selectColumns**( Connection con, String sql, Object... param )

表示查询出的结果是一个json列表。如

sql = select user\_id as userId, user\_name as userName from t\_user

查询出的数据就是

[

{“userId”:1,”userName”:”A”},

{“userId”:2,”userName”:”B”}

]

*d4)* <T> T **getBean**(Connection con, String sql, Class<T> clazz, Object... params)

同d2

*d5)* <T> List<T> **getBeans**(Connection con, String sql, Class<T> clazz, Object... params)

同d1

*d6)*<T> T **queryBean**(Connection con, String sql, Class<T> clazz, Object... params)

获得一个对象，clazz必须是一个@Table对象。

*d7)*<T> List<T> **queryBeans**(Connection con, String sql, Class<T> clazz, Object... params)

获得一个对象列表，clazz必须是一个@Table对象。

1. **DBConnectionUtil**

用来处理数据库连接和释放连接，此方法用在action层。

1. Connection getDBConnection()， 用来连接默认数据，采用连接池
2. Connection getDBConnection( String databasekey)用来连接数据库为databasekey，采用连接池
3. Connection getDBConnection(String url, String user, String passwrod)

默认用来连接Mysql数据库，需要制定数据的url,user,password，一般测试使用，无连接池

1. Connection getDBConnection(String driver, String url, String user, String passwrod)

用来连接制定的数据库，需要制定驱动，url，user，password，一般测试使用，无连接池

1. rollBack(Connection conn) 数据库回滚。
2. 数据库关闭close(Connection con)一般用这个；

close(ResultSet rs, Statement stmt, Connection con)

**工具类介绍**

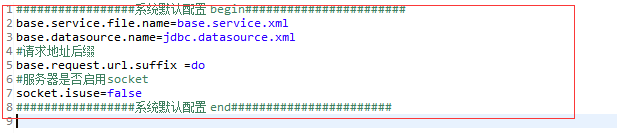
1. pager类，提供给分页使用
2. CommUtils工具类
3. DateUtil类，处理时间之类的数据
4. MD5Util类，处理MD5加密
5. ExportExcel和ExportExcelStream处理导出文件为Excel

注意：使用这两个进行导出文件时，调用的Action方法 不要使用setResult方法。

1. XssFilter 用来防御Xss攻击的过滤器

**配置文件介绍**

1. conf.properties是框架的核心配置文件，放到classes目录下，文件名不能修改。如下图：



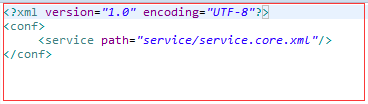
base.service.file.name用来记录所有Action的配置文件是哪一个，如图是base.service.xml;

base.datesource.name用来记录数据源配置文件是哪一个；

base.request.url.suffix 表示的是过滤器拦截的后缀。

1. 承接上图，base.service.xml文件，此文件名可以修改，需要放到classes目录下。

如图：

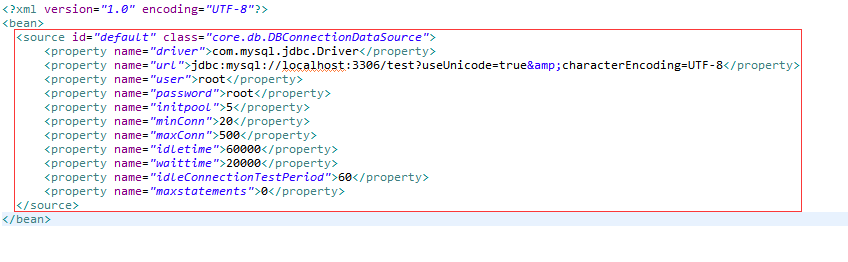


里面配置了所有的Action类的配置，如service.core.xml文件里的内容



每一个Action类的路径都在此配置。

1. 承接1，jdbc.datasource.xml文件内容如下：



此文件名称可以修改。

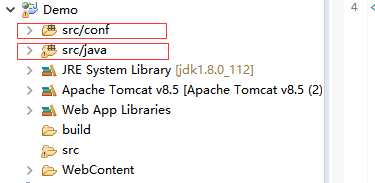
如果配置多个数据源，如下进行：



通过同的id标记不同的数据库。

# 搭建一个开发框架

1. 通过eclipse新建一个Web工程。例如叫Demo



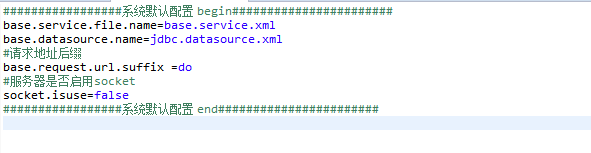
新建2个源文件包，conf和java，conf用来存放配置文件，java用来存放java类。

1. 引入web.xml, 如下图：



默认拦截所有的路径。

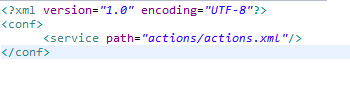
1. 添加conf.properties到src/conf目录下,如下图：



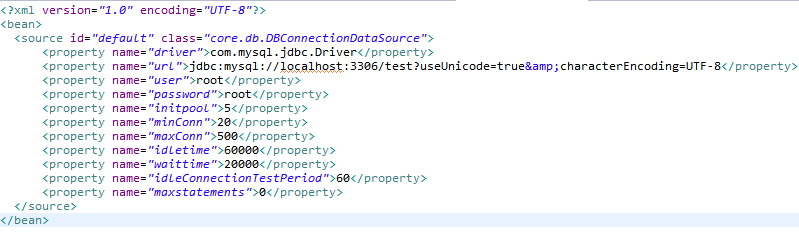
如果想改后缀就把url.suffix修改成别的，并且把web.xml中的\*.do也修改

1. 添加如上图内容，增加一个base.actions.xml和jdbc.datasource.xml

内容如下：base.actions.xml

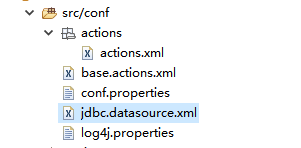


jdbc.datasource.xml如下：

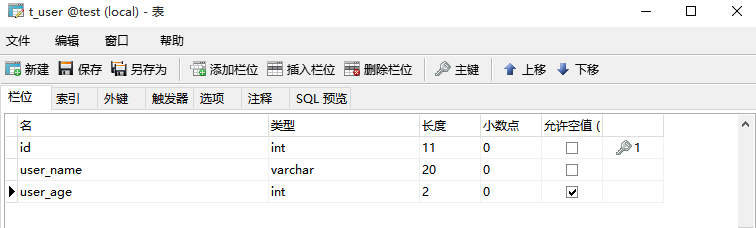


1. 根据base.actions.xml文件配置，创建目录actions,并创建一个actions.xml

最后的配置结果如下：



1. 创建一个表,例如叫t\_user,如下图

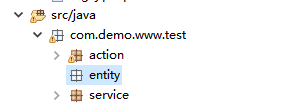


1. 在src/java中创建包com.demo.www.test.action,

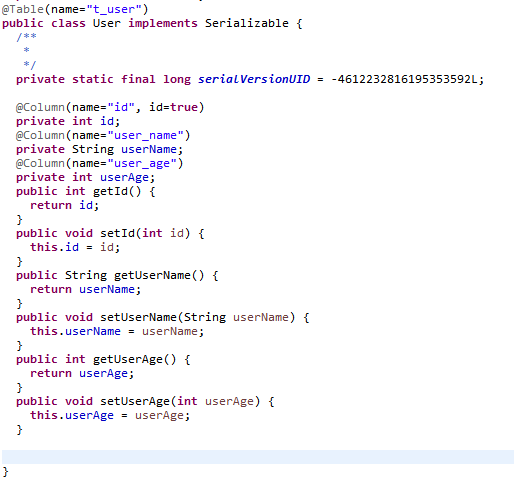
com.demo.www.test.entity,

com.demo.www.test.service

如下图：

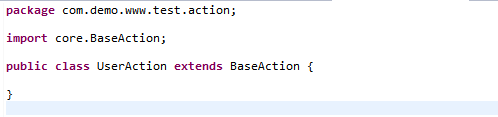


1. 编写User类对应t\_user表

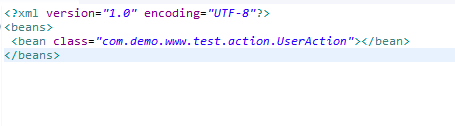


1. 编写Acton，UserAction

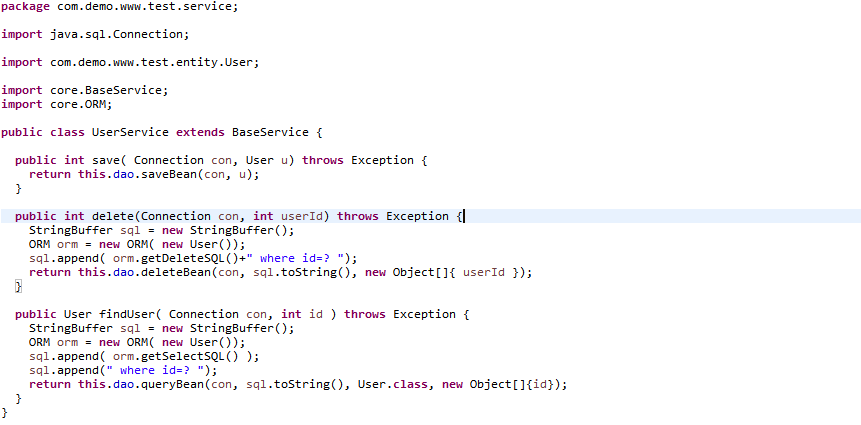
如图：



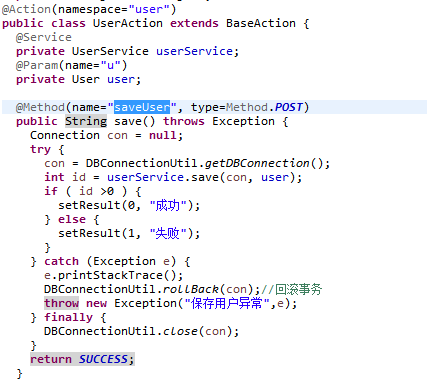
并且在actions.xml中配置此action路径，如下图：

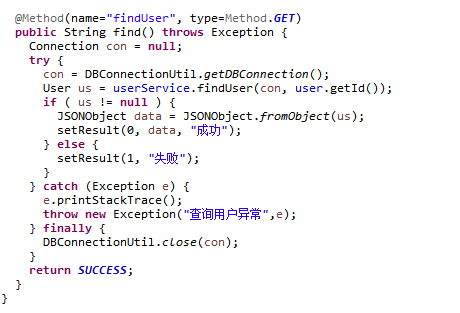


1. 编写Service, UserService



1. Action引用Service、User如下图：





1. 在浏览器中输入:http://localhost

开始测试

注：本框架引用的包可能不是最新包，请自行进行更新最新包，如果需要加新功能，如Redis或ActiveMQ可自行添加。

**数据库类型与java类型的对照表**

|  |  |
| --- | --- |
| 数据库类型 | java类型 |
| char | String |
| varchar |
| varchar2 |
| longvarchar |
| longtext |
| text |
| NUMERIC | BigDecimal |
| DECIMAL |
| BIT | Boolean |
| TINYINT | Integer |
| SMALLINT |
| INTEGER |
| INT |
| BIGINT | Long |
| LONG |
| FLOAT | Double |
| DOUBLE |
| BINARY | byte[] |
| VARBINARY |
| LONGVARBINARY |
| DATE | date |
| TIME | java.sql.Time |
| TIMESTAMP | java.sql.timestamp |
| DATETIME |
| REAL | Float |
|  |  |
|  |  |