**中国电信小CEO项目之**

**SimpleMVC框架说明**

|  |  |  |
| --- | --- | --- |
|  | 部 门: | 研发中心 |
|  | 项 目: |  |
|  | 编 写: | 齐文学 |
|  | 审 核: |  |

用友移动通信技术服务有限公司

2017年08月27日

# 引言

## 编写目的

可以要求提供一种简单、快速的开发框架，并且能够易于维护。并且不能采用市面常用的框架，项目脱离本框架将不能运行。基于此目的特编写SimpleMVC轻量级框架。

## 适用范围

本文适用于中国电信小CEO项目后期维护的开发人员。

## 术语定义

Action，Service，Table，Column，Redirect，Method

## 参考资料

# 框架介绍

## 说明

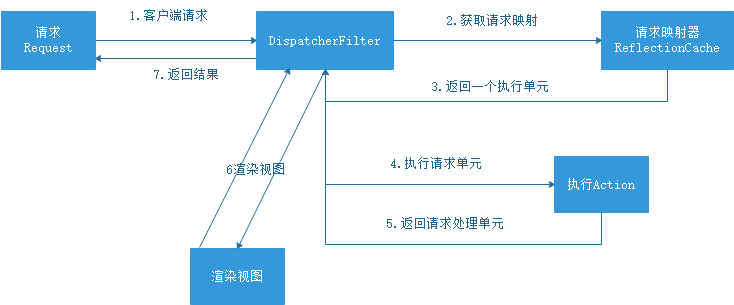
SimpleMVC框架是基于servlet3.0和jdbc组建的一个快速开发框架，并无依赖其他组件。基于少一些配置文件，少一些无用的类，分布式部署。如果你熟悉或了解springMVC和Hibernate您将很快上手。

目前支持的数据库有Mysql、Oracle、PostGreSQL。

本框架中参数采用注入方式，请求不采用单例方式，一个请求会创建一个实例，后续再继续优化。

本文将介绍本框架的基本结构和如何使用本框架进行开发。

## 整体流程图



具体步骤：

* 第一步：发起请求到前端控制器(DispatcherFilter)
* 第二步：前端控制器请求ReflectionCache查找请求执行单元 Unit（可以根据xml配置、注解进行查找）
* 第三步：请求映射器ReflectionCache向前端控制器返回一个执行单元Unit，一个Unit包含了要执行的方法、参数、Session、CoreDao、视图处理器、请求方式等。
* 第四步：前端控制器调用执行单元Unit，去执行对应的Action。
* 第五步：执行单元Unit返回执行结果，并封装到Unit中。
* 第六步：渲染视图。

第七步：返回请求端结果。

## 基础类介绍

1. **BaseAction**

编写Action类的时候需要继承BaseAction，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，注意：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

4）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

5）rollBack(Connection conn) 数据库回滚。

6）close(Connection con)数据库关闭一般用这个；

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。

6）XssFilter 用来防御Xss攻击的过滤器

## 配置文件介绍

1）system.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：



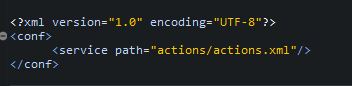
base.service.file.name用来记录所有Action的配置文件是哪一个，如图是base.actions.xml;

base.datesource.name用来记录数据源配置文件是哪一个；

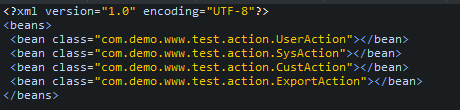
base.request.url.suffix 表示的是过滤器拦截的后缀,可以不填。

2）承接上图，base.actions.xml文件，此文件名可以修改，需要放到classes目录下，此文件再分布式部署的时候可以进行拆分修改。

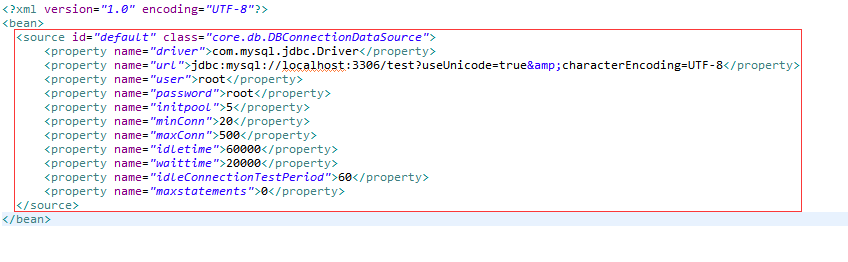
如图：



里面配置了所有的Action类的配置，如actions.xml文件里的内容，此文件再分布式部署的时候可以进行拆分修改。



3）配置文件jdbc.datasource.xml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：



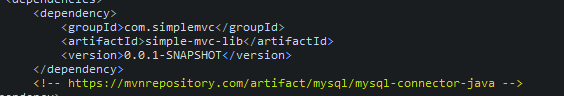
通过同的id标记不同的数据库。

# 快速搭建开发环境

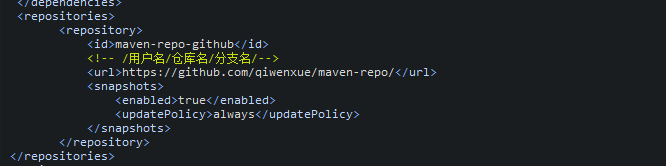
通过上面的介绍，我们可以快速搭建一个开发环境。

通过eclipse新建一个Maven工程。例如叫simple-mvc-demo

在pom.xml中添加下面依赖



在仓库配置中加入下列配置



在web.xml中添加下列配置



这样基本配置就配置好了。

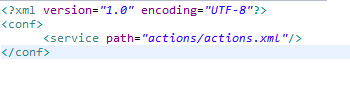
添加system.properties到src/conf目录下,如下图：



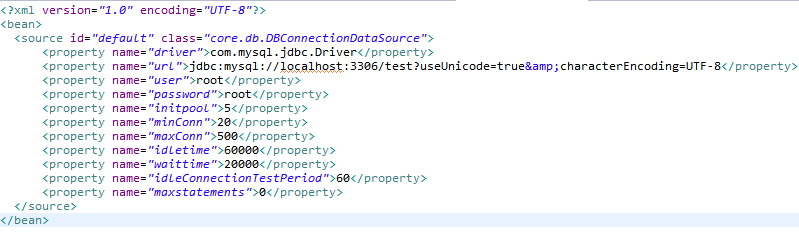
如果想改后缀就把url.suffix修改成别的，并且把web.xml中的\*.do也修改

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

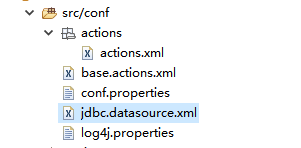


jdbc.datasource.xml如下：

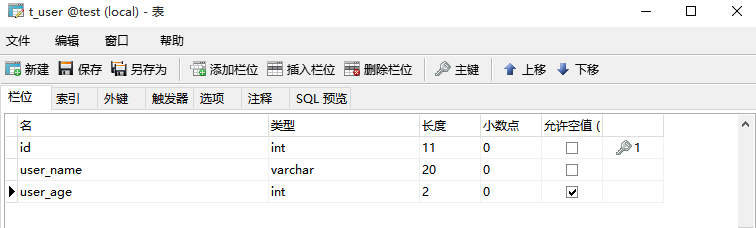


根据base.actions.xml文件配置，创建目录actions,并创建一个actions.xml

最后的配置结果如下：



创建一个表,例如叫t\_user,如下图

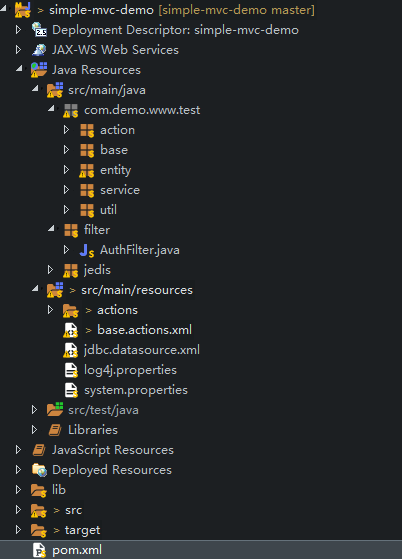


在src/java中创建包com.demo.www.test.action,

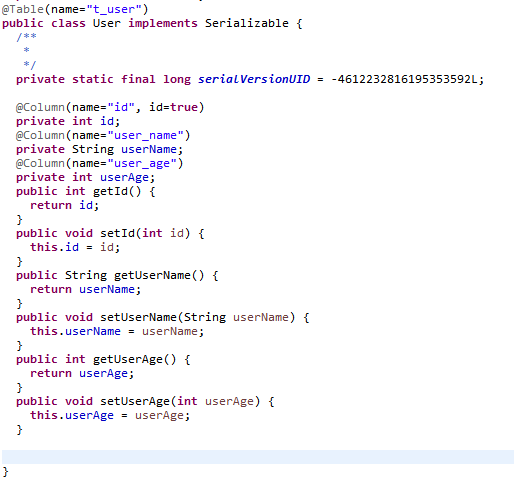
com.demo.www.test.entity,

com.demo.www.test.service

如下图：

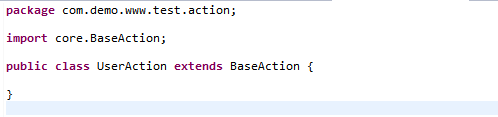


编写User类对应t\_user表

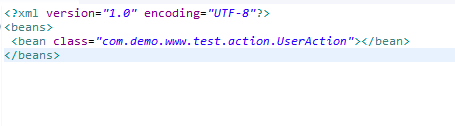


编写Acton，UserAction

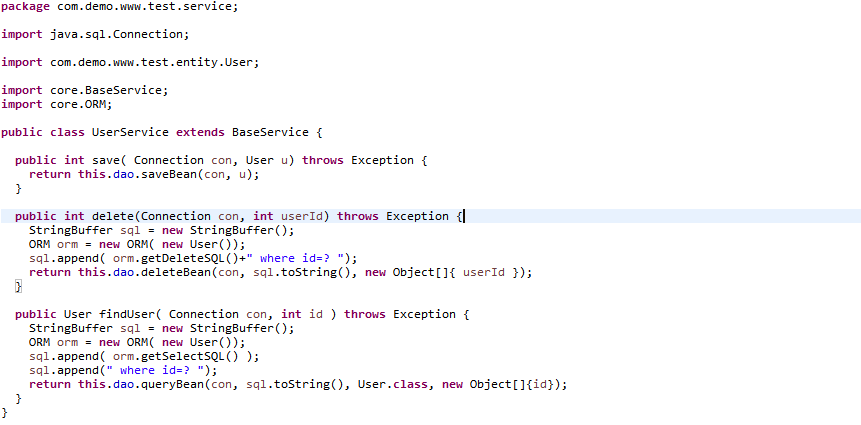
如图：



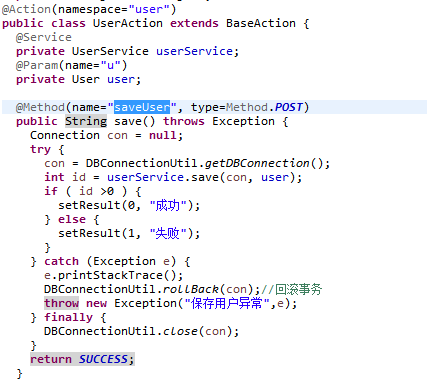
并且在actions.xml中配置此action路径，如下图：

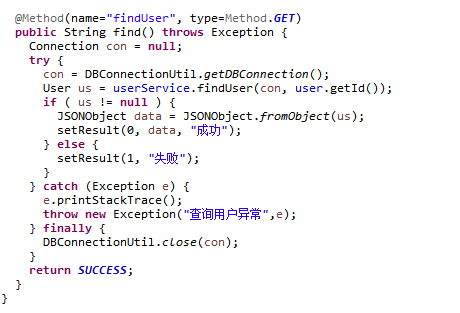


编写Service, UserService



Action引用Service、User如下图：





在浏览器中输入:http://localhost

开始测试

# 附录：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 |
|  |  |
|  |  |