新致Manager Cloud框架

开发指南

2017年9月

**更新记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **更新日期** | **说明** | **更新人** |
| v0.1 | 2017-09-26 | 草稿 | 余磊 |
| v0.2 | 2017-10-22 | 增加前端框架相关内容 | 余磊 |
| v0.3 | 2017-11-03 | 增加开发环境相关内容 | 余磊 |
| v0.4 | 2017-12-05 | 增加新增产品/模块相关内容；增加标准JSP页面说明 | 余磊 |
| v0.5 | 2018-01-24 | 增加IDEA配置JVM参数说明；Tomcat插件启动说明；增加打包命令 | 余磊 |
| v0.6 | 2018-03-01 | 增加多环境打包(profile) | 余磊 |
| v.0.7 | 2018-03-30 | 增加项目组件依赖的更新方式2.6 | 余磊 |
|  |  |  |  |

**目录**

[1. 前言 5](#_Toc504612578)

[2. 开发环境 5](#_Toc504612579)

[2.1. 开发环境准备 5](#_Toc504612580)

[2.1.1. JDK 6](#_Toc504612581)

[2.1.2. Maven 6](#_Toc504612582)

[2.1.3. Eclipse 6](#_Toc504612583)

[2.1.4. IDEA 8](#_Toc504612584)

[2.2. 初始化工程项目 9](#_Toc504612585)

[2.2.1. 下载源代码 9](#_Toc504612586)

[2.2.2. 导入工程项目 9](#_Toc504612587)

[2.3. 项目启动 9](#_Toc504612588)

[2.3.1. Jetty插件启动 10](#_Toc504612589)

[2.3.2. Tomcat插件启动 11](#_Toc504612590)

[2.4. 增加java代码目录 12](#_Toc504612591)

[2.5. 项目打包 13](#_Toc504612592)

[2.5.1. 打包命令 13](#_Toc504612593)

[2.5.2. 打包名称 13](#_Toc504612594)

[2.5.3. 多环境打包 13](#_Toc504612595)

[3. 新增产品/模块 13](#_Toc504612596)

[3.1. 后端 14](#_Toc504612597)

[3.2. 前端 15](#_Toc504612598)

[3.3. 资源配置 16](#_Toc504612599)

[4. 后端开发 17](#_Toc504612600)

[4.1. 概述 17](#_Toc504612601)

[4.2. 控制层(controller/action) 17](#_Toc504612602)

[4.3. 业务处理层(business/bp/service) 17](#_Toc504612603)

[4.4. 数据库访问层(dao/data access) 18](#_Toc504612604)

[4.5. 数据库事务 18](#_Toc504612605)

[4.6. 工具类/公用类 18](#_Toc504612606)

[4.7. 最佳实践 21](#_Toc504612607)

[4.7.1. 前后端交互方式 21](#_Toc504612608)

[4.7.2. url资源组织 21](#_Toc504612609)

[5. 前端开发 22](#_Toc504612610)

[5.1. 框架组成 22](#_Toc504612611)

[5.2. UI组件 22](#_Toc504612612)

[5.3. 标准JSP页面 24](#_Toc504612613)

[5.3.1. 页面顶部： 24](#_Toc504612614)

[5.3.2. 页面head 24](#_Toc504612615)

[5.3.3. JSP页面标准变量和Session全局变量 25](#_Toc504612616)

[5.4. JSP页面资源引用体系 26](#_Toc504612617)

[6. 功能开发 27](#_Toc504612618)

[6.1. 增删改查 27](#_Toc504612619)

[6.2. 上传下载 27](#_Toc504612620)

[6.3. 导入导出 27](#_Toc504612621)

[7. 主框架调整与集成 27](#_Toc504612622)

[8. 迁移 27](#_Toc504612623)

[9. 附录 28](#_Toc504612624)

# 前言

本文描述了Manager Cloud框架（简称为MC框架，下同）的开发方式、环境准备、部署等方面的内容。

本文适合项目开发者阅读，以便快速掌握MC框架项目开发方法。

本文不包含开发流程、编码规范、代码风格等内容。这部分由各项目经理或技术负责人自行指定。

# 开发环境

MC框架的开发环境需要安装以下软件：

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **组件** | **版本** | **说明** |
| 1 | JDK | 1.7 |  |
| 2 | Servlet | 3.0或更高 |  |
| 3 | Maven | 3.3.9 |  |
| 4 | Eclipse | 4.x | Eclipse4.x以上才支持JDK1.7 |
| 5 | IDEA |  | 需支持JDK1.7和Maven3.3.9 |

集成开发环境使用Eclipse或IDEA两者之一即可。

注意需要确认集成开发环境（Eclipse或IDEA）中使用的Maven插件的版本。有可能IDE自带的Maven插件版本不是3.3.9（例如Eclipse4.4中自带的Maven是v3.2.1），需要统一调整为Maven v3.3.9。

## 开发环境准备

此处描述开发环境准备过程的要点，以Windows举例（Windows版本不低于Win7）。集成开发环境以Eclipse和IDEA加以说明。

Eclipse以以Eclipse4.4 for J2EE版为例，加以说明。

IDEA以Community 2017版为例，加以说明。

如果要使用其他集成开发环境，请自行参考相关资料，并需要注意是否支持JDK1.7和Maven3。

### JDK

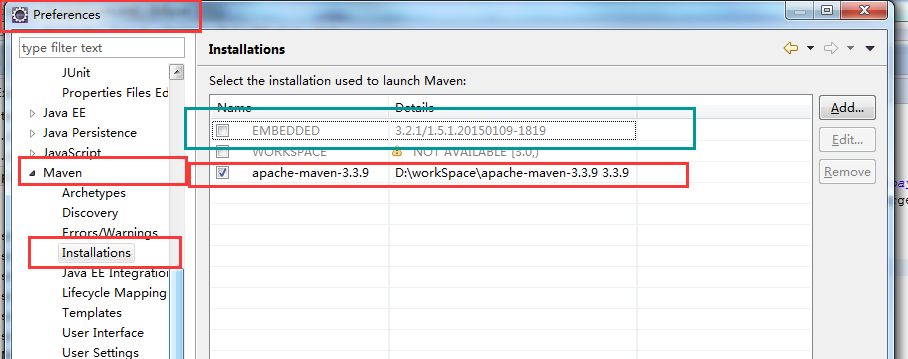
* 安装JDK1.7相应版本。
* 配置JAVA\_HOME环境变量。
* 在命令行中执行命令：java –version，验证配置是否正确。

### Maven

* 解压或安装Maven v3.3.9。
* 配置M2\_HOME环境变量。
* 把M2\_HOME路径配置到Path环境变量中，允许在任意位置都能在命令行中运行maven命令。
* 在命令行中执行命令：mvn –version，验证配置是否正确。

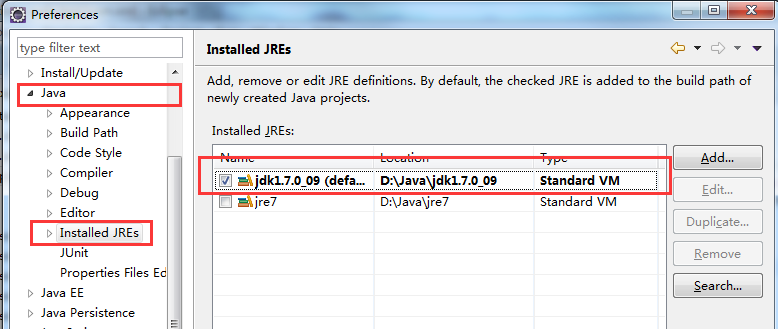
### Eclipse

* 配置Maven，不使用内置Maven插件

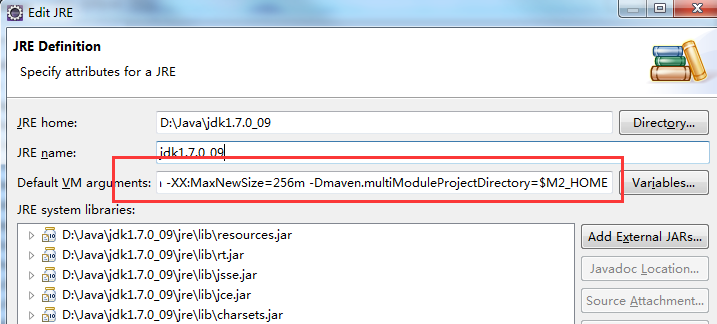


* 修改Eclipse使用系统的JDK

由于Eclipse中默认使用内置的JDK，要改为系统JDK。如下图，jre7为Eclipse内置jdk，jdk1.7.0\_09为系统的JDK。



* 配置JDK内存和Maven路径



为解决大项目启动内存不足，并正常使用maven，需要设置JVM启动参数。在JDK配置界面中点击【Edit】进行设置，在【Default VM arguments】中填写配置参数。

可以先填写以下参数，之后根据项目实际情况调整。

-XX:PermSize=128M -XX:MaxNewSize=256m -XX:MaxPermSize=256m -Xms512m -Xmx512m -XX:MaxNewSize=256m -Dmaven.multiModuleProjectDirectory=$M2\_HOME

其中-Dmaven.multiModuleProjectDirectory=$M2\_HOME为正常使用maven所需。

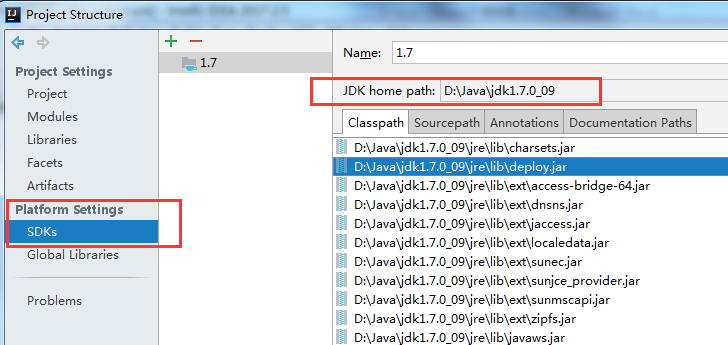
否则在maven项目中使用【Run As】/【Maven build】、【Maven clean】或【Maven install】命令时，可能会报错：

-Dmaven.multiModuleProjectDirectory system property is not set. Check $M2\_HOME environment variable

在【Java/Installed JREs】中，可以设置不同版本的jdk，但不要同时保留jdk和jre，否则maven可能会优先使用jre。

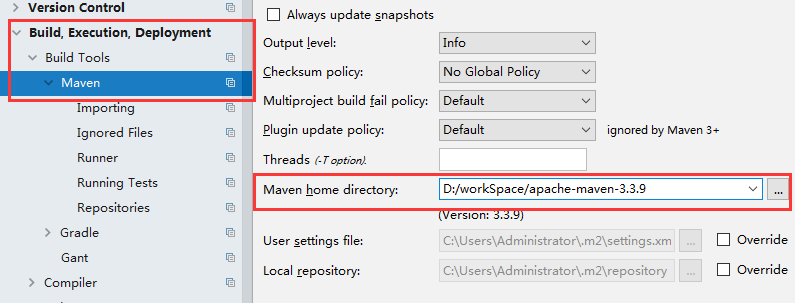
### IDEA

* IDEA会自动检测系统配置的JDK，可以在【File/Project Settings】中的【Platform settings/SDKs】中检查。



* 设置Maven

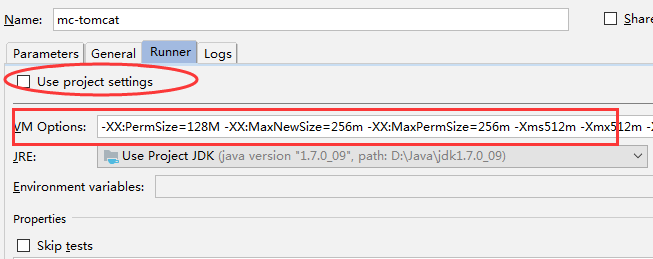
在【File/Settings…】中设置，增加Maven目录指向系统定义的Maven目录，而不是使用IDA内置的Maven。



为解决大项目启动内存不足，需要在项目启动方式下设置JVM启动参数。在每个项目的启动方式界面中，点击【Runner】标签页，取消【Use project settings】的勾，在【VM Options】一栏中填写配置参数。

可以先填写以下参数，之后根据项目实际情况调整。

-XX:PermSize=128M -XX:MaxNewSize=256m -XX:MaxPermSize=256m -Xms512m -Xmx512m -XX:MaxNewSize=256m



## 初始化工程项目

### 下载源代码

* 从svn或git上获得代码。
* 完成后在pom.xml文件所在目录下，执行命令mvn clean package，下载更新依赖项和插件。

### 导入工程项目

* Eclipse

【File/Import…】，选择“Existing Maven Projects”

* IDEA

【File/New/Project from Existing Sources…】

## 项目启动

MC框架目前同时支持Jetty插件和Tomcat插件启动，且在调试过程中支持热更新。

### Jetty插件启动

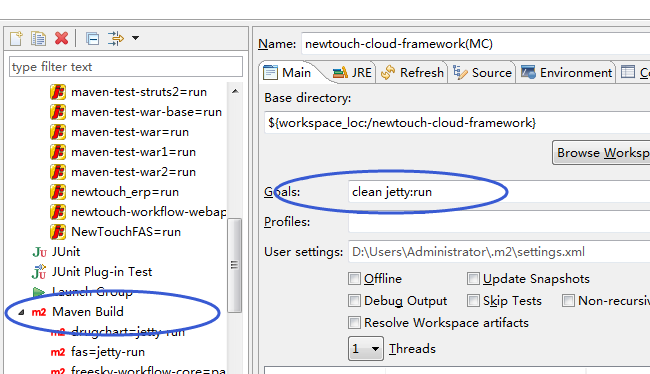
#### 配置启动方式

* 启动命令

clean jetty:run

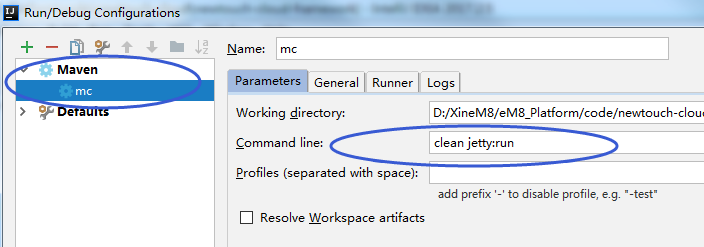
* Eclipse

使用Eclipse自带的M2插件配置启动方式，这样才会使用项目pom中内置jetty插件启动调试。



* IDEA

使用maven方式创建启动方式



#### 修改启动参数

* 应用目录名：在pom.xml文件build下找到jetty插件（artifactId:jetty-maven-plugin），修改plugin\configuration\webApp\contextPath
* 端口号：修改plugin\configuration\httpContector



### Tomcat插件启动

#### 配置启动方式

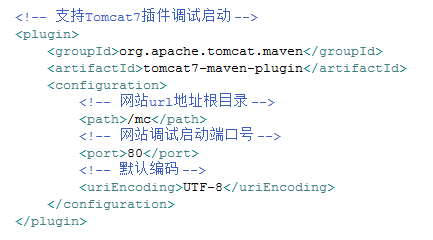
* 启动命令

clean tomcat7:run

* 在Eclipse和IDEA中配置Tomcat插件的启动方式，请参考Jetty插件配置启动方式，只有启动命令不同。

#### 修改启动参数

* 应用目录名：在pom.xml文件build下找到tomcat7插件（artifactId:tomcat7-maven-plugin），修改plugin\configuration\path
* 端口号：修改plugin\configuration\port



## 增加java代码目录

maven默认只支持src/main/java和src/test/java连个java代码目录。

如果开发过程中需要增加了java代码目录，应当在pom.xml中增加设置。否则编辑和打包时会无法访问新增的java代码目录。

在pom.xml中查找插件build-helper-maven-plugin，然后修改configuration\sources\下的设置。



## 项目打包

### 打包命令

请使用Maven命令打包，而不要使用开发环境的导出功能。

在项目根目录下（即pom.xml所在目录），执行以下命令：

mvn clean package

即在\target目录下生成一个war包和一个source包。

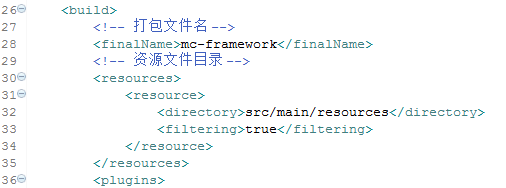
### 打包名称

在pom.xml中修改build\finalName节点。如下图所示，最终将打包为mc-framework.war



### 多环境打包

* 当为不同环境打包时，通常有一些配置参数或配置文件需要修改。我们借助maven的profile机制来执行多环境打包。
* mc框架项目默认的资源文件目录为src/main/resources，与maven标准项目结构一致。在pom.xml配置文件中也有定义。



* pom.xml中profiles小节，分别定义了多个不同的profile，对应不同的打包环境。这里定义了三个：dev开发环境、test集成测试环境、rc生产环境。每个环境指定一个目录，里面存放每个环境额外的配置文件。



* 打包命令：

在项目根目录下（即pom.xml所在目录），执行以下命令，即可按照profile=rc方式打包。

mvn clean package -Prc

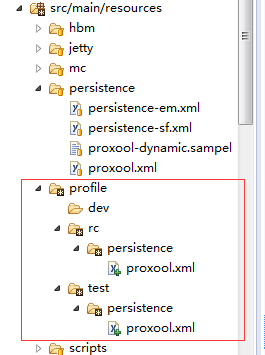
其中“–P”为profile参数，后面紧跟profile名称。

* 资源文件打包过程：

首先抽取标准资源文件目录（src/main/resources）下所有文件

然后根据profile名称，抽取额外的资源文件目录（src/main/resources/profile/[profile]）下的文件。

* 资源文件目录如下所示



## 更新组件依赖

日常开发过程中，一般项目的pom文件会由技术负责人更新。当开发人员从git上获取到最新的pom文件后，需要有一个maven更新过程，才能获取最新版本的组件。

下面针对三种情况说明更新方式。

### 命令行方式

最简洁最彻底的方式，就是使用命令行更新。

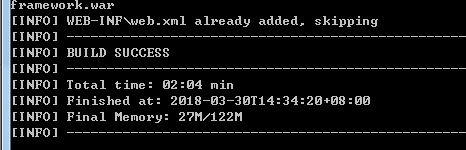
打开命令行工具，进入到项目pom文件所在目录，执行以下命令：

mvn clean package

正常情况下，maven会

1. 自动清除target目录下的文件
2. 按照当前pom文件下载最新的依赖组件。
3. 重新编译并打包。

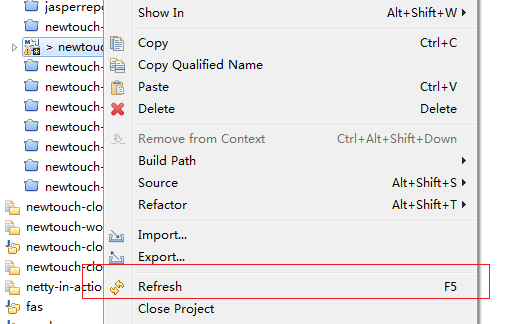
执行完成后，如果出现“BUILD SUCCESS”字样，说明依赖组件更新下载都没有问题。



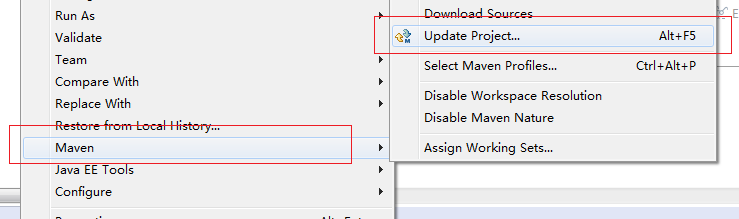
如果组件下载有报错，会出现错误提示。

### Eclipse

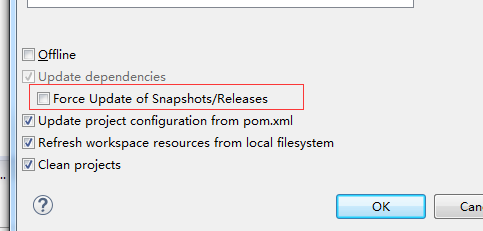
右键选择项目，F5刷新。



如果没有生效，可以右键选择项目后选择【Maven/Update Project…】

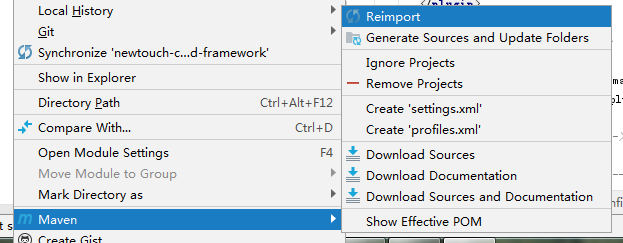


在弹出窗体中选择图中几项，还可以勾选“Forec Update of Snapshots/Releases”，再进行更新。



### IDEA

右键项目，选择【Maven/Reimport】后重新导入依赖项组件。

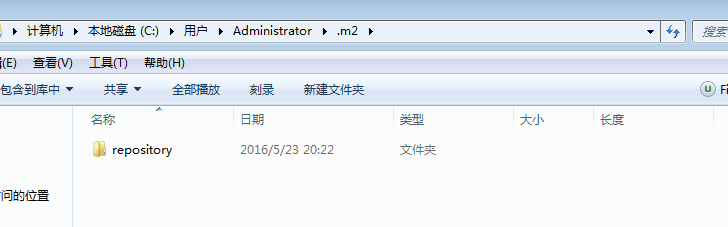


### 后备措施

如果以上更新办法仍然无法更新组件。可能是新老两个版本的组件版本号相同，导致maven认为不需要更新。

这种情况下，需要到maven本地仓库中，手工删除已下载的组件，然后重新执行之前的更新步骤。

maven本地仓库所在目录默认情况下在：${user.home}/.m2/repository/，如下图所示：



# 新增产品/模块

新开一个产品/模块，需要考虑与公用MC框架库、以及其他产品/模块的耦合度。

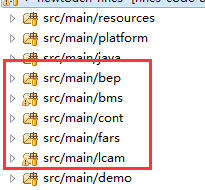
耦合难度主要在前端引用上。

MC框架项目目录结构，请参照《新致MC框架-构建指南》相关章节。

## 后端

* 创建产品/模块的java代码目录
* 如果单模块的项目，可以直接使用src/main/java目录。
* 如果是多模块的项目，可以在src/main/目录下，给每个模块单独建立java目录。

例如下图中红框标出的，即为各模块的java代码目录。



注意，如果增加了java代码目录，需要修改pom文件中build-helper-maven-plugin插件的设置（参见本文档开发环境章节中相关内容）。否则编译打包时将无法读取到这部分java文件。

* java代码目录下典型的分包结构如下：

|  |  |  |
| --- | --- | --- |
| **序号** | **目录** | **说明** |
| **后端** | | |
| 1 | src/main/jara/com/newtouch/$module/ | xxx业务模块java代码 |
| 2 | src/main/jara/com/newtouch/$module/common/ | xxx业务模块公用组件 |
| 3 | src/main/jara/com/newtouch/$module/$submodule1/ | xxx业务模块/子模块1 |
| 4 | src/main/jara/com/newtouch/$module/$submodule1/action/ | xxx业务模块/子模块1/action层 |
| 5 | src/main/jara/com/newtouch/$module/$submodule1/bp/ | xxx业务模块/子模块1/业务处理层 |
| 6 | src/main/jara/com/newtouch/$module/$submodule1/dao/ | xxx业务模块/子模块1/数据访问层 |
| 7 | src/main/jara/com/newtouch/$module/$submodule1/entity/ | xxx业务模块/子模块1/实体类 |
| 8 | src/main/jara/com/newtouch/$module/$submodule2/ | xxx业务模块/子模块2 |
| 9 | src/main/jara/com/newtouch/$module/$submodule2/action/ | xxx业务模块/子模块2/action层 |
| 10 | src/main/jara/com/newtouch/$module/$submodule2/bp/ | xxx业务模块/子模块2/业务处理层 |
| 11 | src/main/jara/com/newtouch/$module/$submodule2/dao/ | xxx业务模块/子模块2/数据访问层 |
| 12 | src/main/jara/com/newtouch/$module/$submodule2/entity/ | xxx业务模块/子模块2/实体类 |

## 前端

* 一般在src/main/webapp/目录下，创建产品/模块的前端代码目录；如果是多模块，则建立多个目录。
* 模块内典型的分包结构如下，其中common目录存放模块内的公用js、component目录存放封装的业务组件。

|  |  |  |
| --- | --- | --- |
| **序号** | **目录** | **说明** |
| **前端** | | |
| 13 | src/main/webapp/$module/ | xxx业务模块web功能代码 |
| 14 | src/main/webapp/$module/common/ | xxx业务模块公用js |
| 15 | src/main/webapp/$module/component/ | xxx业务模块公用业务组件 |
| 16 | src/main/webapp/$module/resources/ | xxx业务模块资源，包括css、图片、国际化资源等 |
| 17 | src/main/webapp/$module/$submodule1/ | xxx业务模块/子模块1 |
| 18 | src/main/webapp/$module/$submodule1/$fun1/ | xxx业务模块/子模块1/功能1 |
| 19 | src/main/webapp/$module/$submodule1/$fun2/ | xxx业务模块/子模块1/功能2 |
| 20 | src/main/webapp/$module/$submodule2/ | xxx业务模块/子模块2 |
| 21 | src/main/webapp/$module/$submodule2/$fun1/ | xxx业务模块/子模块2/功能1 |
| 22 | src/main/webapp/$module/$submodule2/$fun2/ | xxx业务模块/子模块2/功能2 |

* 一般在$module/common/下会编写一个前端公用引用的jspf文件，例如xx\_all.jspf。在这个公用引用文件中，处理模块间耦合的问题。具体功能的JSP页面则只需要引用xx\_all.jspf接口。
* MC前端框架的入口是webapp//common/mc\_all.jspf，其他业务模块应当只引用该文件，而不要直接引用MC前端框架内部的文件。
* 多模块引用关系如下图所示，应当按照以下原则处理。



## 资源配置

当项目规模较大，模块较多时，建议对于各类配置文件、资源也分模块存放。需要注意的包括，但不仅限于：

* spring-content配置文件
* hibernate的orm配置文件(hbm)
* 各类第三方工具组件的配置文件，如log4j、logback等日志组件；quartz定时任务组件等。
* workflow的实体配置文件
* axis等WebService配置文件
* 前端图片、css样式等资源文件

# 后端开发

## 概述

采用三层架构，各层职责：

* action层：处理接受参数和组织返回参数，不包含业务处理
* bp层：业务逻辑处理，不包含具体的的数据库访问，例如sql语句
* dao层：数据访问，不包含复杂的业务处理

## 控制层(controller/action)

* 普通类(POJO)，不继承
* 类注解：

@Controller

@Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE) /\* action采用单例模式 \*/

@RequestMapping("/demo") /\* url映射 \*/

* 自动装配bp，例如：

@Autowired

private DemoBP bp;

* 方法注解：

@RequestMapping("/page") /\* url映射 \*/

@ResponseBody /\* 序列化http请求响应结果 \*/

其中@ResponseBody支持自动把任意对象序列化为json返回，也支持直接返回字符串文本。

如果要直接操作HttpResponse的输出（例如文件下载），则不需要标注@ResponseBody。

## 业务处理层(business/bp/service)

* 普通类(POJO)，不继承
* 类注解，例如：

@Service

@Transactional /\* 事务控制注解 \*/

* 自动装配dao，例如:

@Autowired

private DemoDAO dao;

## 数据库访问层(dao/data access)

* 类注解，例如：

@Repository

* 可以继承MC框架提供的com.newtouch.cloud.common.dao.CommonDAO、com.newtouch.cloud.common.dao.HibernateDAO，或其他自定义的公用DAO类

## 数据库事务

由于数据库事务都标注在业务处理层（BP/Service），为保证数据库事务一致性，一般情况下请遵循以下规则：

* 一个action层方法中只能调用一次bp层方法。
* 一个bp层方法可以多次调用dao层方法，或多次调用其他bp层方法。
* 除非有特别理由，不要在业务代码中手动提交或回退数据库事务，事务统一委托给框架处理。
* 不允许action层方法直接调用dao层方法。

## 工具类/公用类

框架提供了多个后台工具类或公用类，可提供快速开发使用

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **namespace** | **类名** | **说明** |
| 1 | com.newtouch.cloud.common | ActionResultUtil | ActionResultUtil |
| 2 | ApplicationContextUtil | SpringContextBean容器工具类，允许直接访问SpringJavaBean |
| 3 | DateUtil | 时间日期工具类，提供标准化日期格式化功能 |
| 4 | DownloadUtil | 文件下载工具类，负责把文件写入Response中，完成下载 |
| 5 | EntityUtil | 实体转换工具类 提供了一下类型转换： 1)java.sql.ResultSet转为List<Entity>和List<EntityMap> 2)Entity实体类和EntityMap包装互相转换 3)List<Entity>和List<EntityMap>互相转换 4)json字符串反序列化为Entity实体类、EntityMap 5)json字符串反序列化为List<Entity>实体类、list<EntityMap> 6)从普通List<Entity>实体类、list<EntityMap>转换树型数据List<TreeNodeData> |
| 6 | ExcelUtil | Excel工具类，依赖于poi组件 |
| 7 | GenericityUtil | 泛型工具类 |
| 8 | MathUtil | 浮点数高精度计算工具类，基于double |
| 9 | NumberUtil | 数字处理工具类 |
| 10 | ObjectUtil | Object处理工具类，提供反射功能的封装 |
| 11 | RSUtil | ResultSet工具类 |
| 12 | StringUtil | 字符串(String)操作工具类 |
| 13 | XmlUtil | XML文档处理工具类 |
| 14 | com.newtouch.cloud.common.dao | CommonDAO | 数据访问通用类 支持以下一些访问方式： 1.getEntity，可获得实体类对象{@code <T>}。 2.getEntityList，可获得实体类对象列表{@code List<T>}。 3.getEntityPage，可获得实体类对象分页列表{@code PageData<T>}。 4.getMapList，可获得实体Map对象列表{@code List<EntityMap>}。 5.getMapPage，可获得实体Map对象分页列表{@code PageData<EntityMap>}。 6.getTotal，获取查询结果行数 7.querySingleString、querySingleInteger、querySingleDouble、querySingleObject，查询单个数值。 8.execute，执行原生sql。 |
| 15 | CommonJDBCDAO | 数据访问通用类(JDBC) |
| 16 | HibernateDAO | Hibernate Session数据访问类包装 |
| 17 | com.newtouch.cloud.common.entity | ActionResult | Action通用返回对象 |
| 18 | ConditionMap | 过滤条件Map实体 |
| 19 | EntityMap | 通用实体包装类,字段按照put的先后顺序排序 |
| 20 | PageData | 分页数据 |
| 21 | TreeNodeData | 树节点数据实体 |
| 22 | com.newtouch.cloud.common.encrypt | CryptoSP | M8安全加密类，双向加密 |
| 23 | DESUtil | DES加密/解密工具类 |
| 24 | MD5 | MD5算法类，单向加密算法 |
| 25 | SHA1 | SHA1算法类，单向加密算法 |
| 26 | TripleDesUtil | 三重DES加密/解密工具类 |

## 最佳实践

### 前后端交互方式

* 应尽量采取前后端分离方式开发，减少前后端代码耦合。由后端提供数据，由前端负责展现。
* JSP页面尽量不要在服务端拼写html页面，即尽量不要使用标签库，减少使用各类EL表达式。由html、css和js组件负责界面展现。

### url资源组织

* 在action层定义的url，建议同时在类上和方法上做服务请求映射（@RequestMapping）。
* url资源命名可部分采用restful风格，常见的命名方式为：

产品 + 模块 + 数据实体/资源 + 操作

可以在Action类上映射产品 + 模块 + 数据实体/资源；在Action方法上映射操作。

* url应尽量全小写，减少前端调用失误。

基于同样的理由，前端传入后端的参数键值，以及后端返回到前端的数据键值，应尽量采用全小写。

* url资源定义应当简明

例如，wfs产品下有总账模块(gl)，其中有一个数据：会计科目(account)，其相关http api接口定义方式如下：

在AccountAction类上映射wfs/gl/account

增删改查方法映射为list、page、tree、add、update、delete、import、export等

|  |  |  |
| --- | --- | --- |
| **序号** | **url** | **API说明** |
| 1 | [server]/wfs/gl/account/list | 查询列表 |
| 2 | [server]/wfs/gl/account/page | 查询列表(分页) |
| 3 | [server]/wfs/gl/account/tree | 查询树 |
| 4 | [server]/wfs/gl/account/add | 增加(单个) |
| 5 | [server]/wfs/gl/account/update | 修改(单个) |
| 6 | [server]/wfs/gl/account/delete | 删除(单个) |
|  |  |  |

* http API接口，应当控制参数数量，避免出现太多参数。

如果参数数量多过，建议封装成为对象，序列化为json字符串后传递到后端，在服务端的action层反序列化为参数对象Map后使用。

# 前端开发

## MC前端开发框架交付物

前端开发框架包含

* 经过选型的第三方类库，作为基础类库。
* MC框架封装过的UI组件和工具类。
* 开箱即用样例工程，并包含在线开发教程，包括UI组件、界面布局等。
* 提供默认登录、默认首页、系统管理、个人管理功能、工作流管理，作为最佳实践。

## UI组件

**等待更新**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **组件** | **简要说明** | **选定组件** | | | |
| **html标签** | **组件** | **版本** | **选型说明** |
| 1 | 文本 | 显示文本 | <input type='text'> |  |  | 无组件 |
| 2 | 单选框组 | 一组多个单选框，选项互斥 | <input type='radio'> |  |  | 无组件 |
| 3 | 多选框 | 多选框 | <input type='checkbox'> |  |  | 无组件 |
| 4 | 多选框组 | 一组多个多选框组 | <input type='checkbox'> |  |  | 无组件 |
| 5 | 文本输入框 | 普通文本输入框 | <input type='text'> |  |  | 无组件 |
| 6 | 数字输入框 | 只允许输入数字 | <input type='text'> | NumberField |  |  |
| 7 | 金额输入框 | 只允许输入金额，且按照金额格式显示的输入框 | <input type='text'> + 事件 | MoneyField |  | 未完善未选定 |
| 8 | 日期选择/输入框 | 日期的输入框或选择框，允许限制范围，如果支持需要支持日期格式化验证 | <input type='text'> | DateField |  | 拟定使用jQuery-UI的datepicker |
| 9 | 按钮 | 按钮，可以显示文本或图片 | <button> |  | 1.12.1 |  |
| 10 | 下拉框 | 下拉框，支持远程数据 | <select> | ComboBox | 1.12.1 | 单独使用html的select，浏览器兼容性不好 |
| 11 | 列表 |  | <table> + <div> | jQGrid | 5.2.1 | 请求参数与返回参数与EXT风格不一致 |
| 12 | 树 |  |  |  |  | 未选定，拟使用zTree |
| 13 | Tab标签页 |  | <div> + <ul> + <li> | layui.tabs | 2.1.5 |  |
| 14 | 信息提示 | 操作简便，需要考虑弹出顶层 |  | MsgUtil.alert、MsgUtil.confirm | 3.1.0 | 与模态对话框一起使用layer，方便统一风格 |
| 15 | 遮罩 | 表示长时间操作，需要考虑弹出顶层 |  | MaskUtil | 3.1.0 | 与模态对话框一起使用layer，方便统一风格 |
| 16 | 模态对话框 | 需要考虑多种引用方式，需要考虑弹出顶层 |  | layer.open() | 3.1.0 |  |
| 17 | 弹出选择框 | 基于模态对话框，支持列表或树数据选择的组件。简化调用。 |  |  |  | 倾向于使用layer，dom方式不容易切分代码，iframe方式速度太慢 |
| 18 | 布局 |  | <div> | layout | 1.4.4 | 能解决border布局 |
| 19 | 样式 |  |  |  |  |  |

## 标准JSP页面

### 页面顶部：

JSP标准页面，应当在指定后端开发语言、字符集以及页面文档格式：

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!DOCTYPE html>

### 页面head

引入本模html页面的head部分，一般需依次定义meta元数据定义、引用css样式表、引用js代码文件。而且各部分顺序有一定要求，不可混乱。

由于MC前端框架涉及的文件较多，关系较复杂。为了简化使用，只需要引用/mc/common/mc\_all.jspf即可。



具体怎样建立资源引用体系，在下一小节描述。

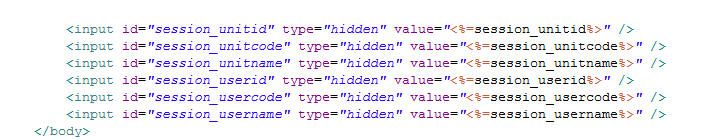
### JSP页面标准变量和Session全局变量

在mc\_all.jspf中，除了定义通用的meta元数据定义之外，还预定义了一部分全局变量，可在页面上直接使用。

这部分全局变量是登录后写入session的，因此前台后台都可以使用。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **jsp变量** | **含义** | **服务端访问工具类** | **访问方法** |
| 1 | session\_unitid | 当前用户所属公司Id | com.newtouch.cloud.common.session.M8Session | getCompanyID() |
| 2 | session\_unitcode | 当前用户所属公司编码 | getCompanyCode() |
| 3 | session\_unitname | 当前用户所属公司名称 | getCompanyName() |
| 4 | session\_userid | 当前用户Id | getUserID() |
| 5 | session\_usercode | 当前用户登录名 | getUserCode() |
| 6 | session\_username | 当前用户姓名 | getUserName() |

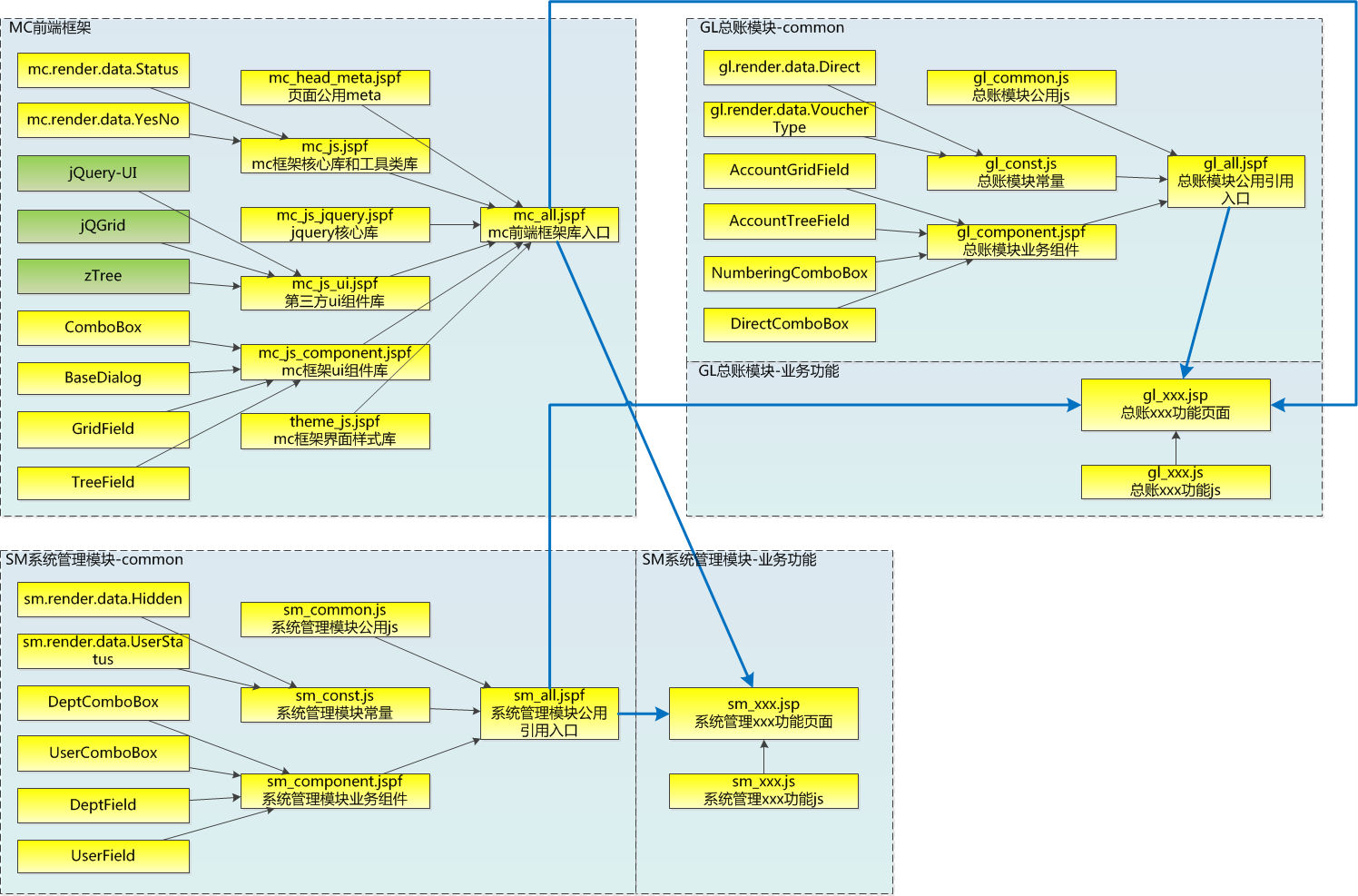
如果在页面中需要使用这些变量，可以在body中加入以下html代码，即可在页面上访问这些值。



如果项目有需要，可以在登录后在Session写入新的全局变量。

注意Session全局变量需要立足在项目整体进行规划，不能滥用。否则可能会影响性能，以及导致各功能/模块之间耦合更紧密，也会不利于前后端分离。

## JSP页面资源引用体系



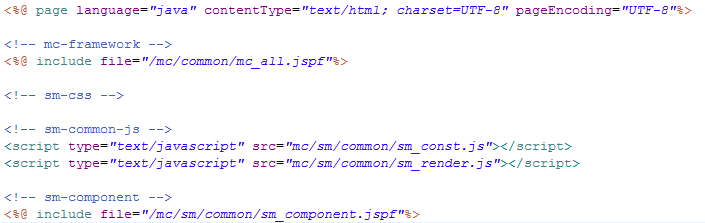
从上图可以看到MC前端框架的入口是/mc/common/mc\_all.jspf文件，其中包含了第三方类库、mc基础类库、mc基础组件、mc基础样式等资源。

如果没有特殊要求，我们新项目的jsp页面，可以直接引用mc\_all.jspf页面，就可以使用MC前端框架所有的功能。

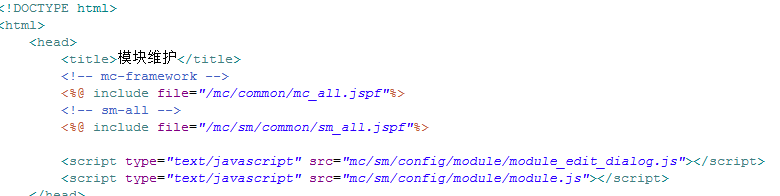
但是我们的新项目往往不会这么简单，通常也会包含我们自己项目/模块内部的公用js代码、公用业务组件、甚至会在mc框架基础上增加了自己的css样式表。

我们可以在自己的项目/模块中建立一个自己项目/模块的公共jspf引用文件，依次引用自己的css、js等各类资源。

参考mc.系统管理模块的src/main/webapp/mc/sm/common/sm\_all.jspf



最后在实际功能JSP页面开发过程中，就可以直接引用mc\_all.jspf和sm\_all.jspf



参考以下功能代码：

src/main/webapp/mc/sm/config/module/module.jsp

src/main/webapp/mc/sm/config/menu/menu.jsp

# 功能开发

## 增删改查

**敬请期待**

## 上传下载

**敬请期待**

## 导入导出

**敬请期待**

# 主框架调整与集成

# 迁移

从老ssh框架迁移过来，需要做以下事项：

* jsp页面，去掉struts标签库引用（一般在页面顶部）。因为新框架取消了struts组件

<%@ taglib prefix="s" uri="/struts-tags" %>

* 重写Action，取消从BaseAction或ActionSupport继承，改为普通类(POJO类)。
* Action如果使用成员属性接受参数，改为方法的参数，并用@ResquestParam注解。注意如果接受参数过多（一般不要超过3个），请打包为json后传输。
* 返回值取消ActionJSONUtil工具类，改用ActionResultUtil工具类，API返回值改为ActionResult
* 检查BP和DAO，如果使用成员属性，取消改为方法内的成员属性。
* Action、BP、DAO都是用注解自动装配，取消xml配置文件。
* 取消在BP、DAO层使用M8Session，只允许在Action层使用M8Session。如果需要在BP、DAO层中使用M8Session内值，通过Action传入相关值。

# 附录