1、代码规范

1.1 命名风格

1.1.1 Java命名风格

Java文件以大写字母开头，全部使用英文，禁止使用中英结合的命名方式。这其中包括你的类名称、方法名称、变量名称和数据库字段等等。比如：UserInfoController.java / IpUtils.java；禁止：yongHuInfo.java

public String findUserName(); / public List<UserInfo> findUserInfoList(Long id); 禁止：public String findYongHuName(); private IMcRoleFunctionDao dao 禁止：private IJueSeGongNengDao dao;

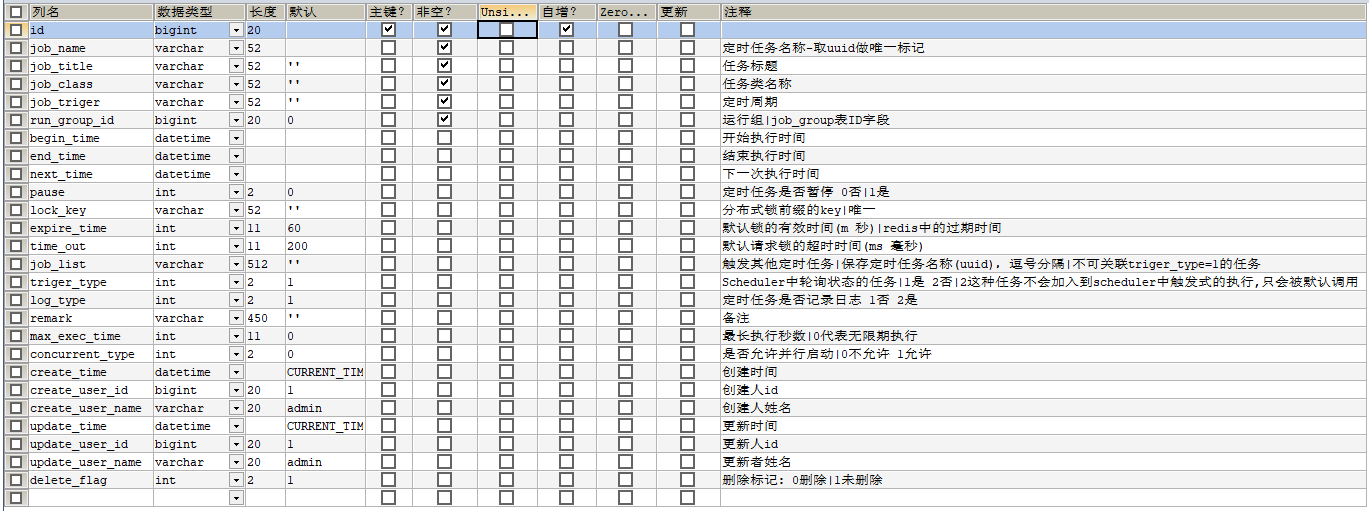
1.1.2 数据库字段命名风格

数据库字段命名以小写英文加下滑线的方式组合，例如：create\_user\_id，表意为创建人主键。禁止使用中文进行命名，禁止使用中英混合命名，禁止使用Java驼峰命名；错误的命名风格如下：chuang\_jian\_ren\_id、

CREATE\_USER\_ID、geng\_xin\_shijian、updateTime。

数据库中每新加一张表，必须包含如下基础字段：create\_time|create\_user\_id|create\_user\_name|update\_time|update\_user\_id|update\_user\_name|delete\_flag；且每个字段后面必须包含对应的注释。下图将展示

一张符合规范的数据库表的样式：



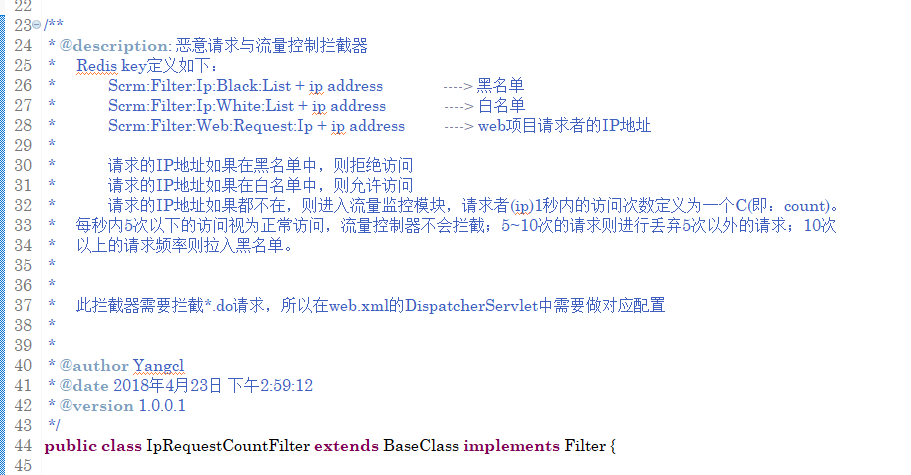
1.2 类与方法的注释

注意！注释风格必须统一，与本节中图片中注释的样式一致，不允许团队成员私自更改注释结构和样式风格。

1.2.1 类注释

类开头必须有注释标识，不允许为空，不允许拷贝他人注释且不做修改。通常类的开头注释包含如下内容：@description 描述这个类做什么；@author 创建人；@date 创建时间；@version 版本号(默认1.0.0.1)。

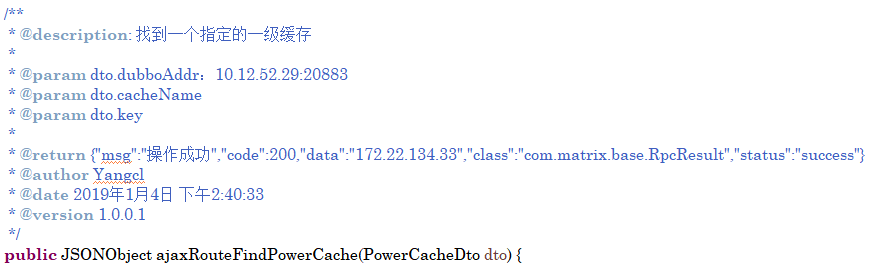
务必详细描述这个类是做什么的，不要让接手你代码的人去猜。一个符合约定的类注释如下：



1.2.2 方法注释

无论私有方法还是公有方法，都要添加注释。相比类注释，方法注释需要额外添加@param，用来描述这个方法的入参。如果方法有返回值，那么需要增加@return，描述返回的内容。原则：让看的人，一目了然，不用猜。

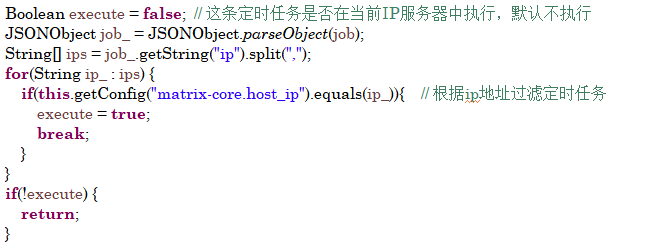
一个符合约定的方法注释如下：

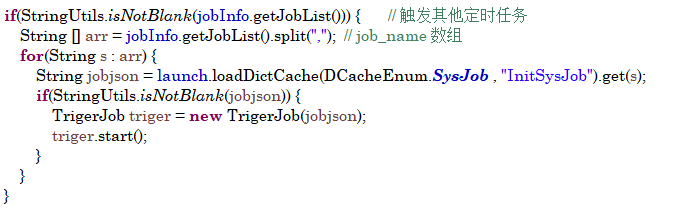


1.2.3 字段注释

针对业务复杂或封装抽象的代码，需要加入适当的注释，便于代码维护与扩展。例如：**public** **abstract** **class** RootJob **extends** BaseClass **implements** Job, IbaseJob{……}，抽象类中封装过于复杂、晦涩，

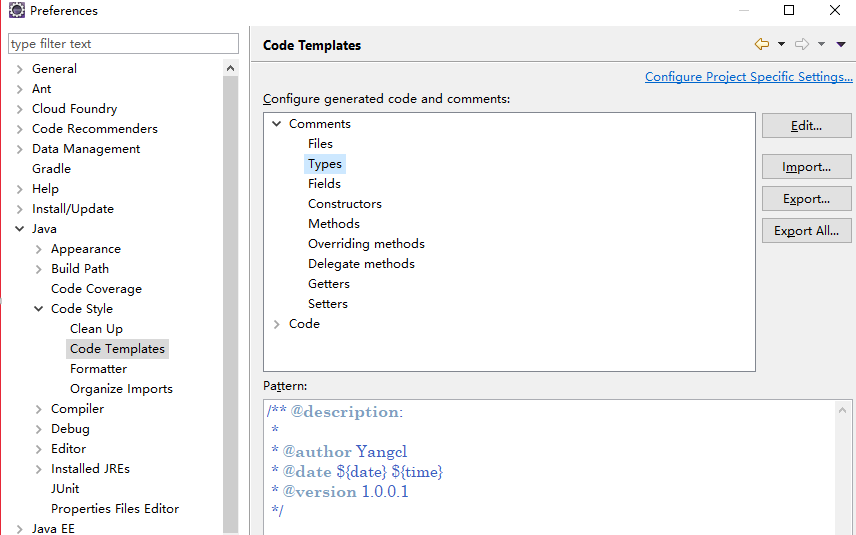
需要在代码中加入说明，示例如下：





1.2.4 Eclipse自定义注释

Eclipse添加自定义注释的方式如下：



类注释模板：

/\*\* **@description**:

\*

\* **@author** 此处填写你的名字，如：ShaoHua.Ma

\* **@date** ${date} ${time}

\* **@version** 1.0.0.1

\*/

方法注释模板：

/\*\* **@description**:

\*

\* ${tags}

\* **@author** 此处填写你的名字，如：ShaoHua.Ma

\* **@date** ${date} ${time}

\* **@version** 1.0.0.1

\*/

如果使用IDEA作为代码编辑器，请自行解决注释风格不统一的问题。

1.3 代码层级约定

1.3.1 Controller层约定



如右图所示，展示了一个标准的Controller层代码。首先标明注释，清晰的业务描述，告诉后续的使用者这个控制器是做什么的，从业务上划分好层级。

在Controller层中，一个具体的业务方法，如左图所示。在Controller层调用Service层中的代码，所有的业务处理、拼装都要放到Service层中，在Controller层禁止出现任何形式的与业务处理相关的代码，这样做可以最大限度的提高代码的复用性。

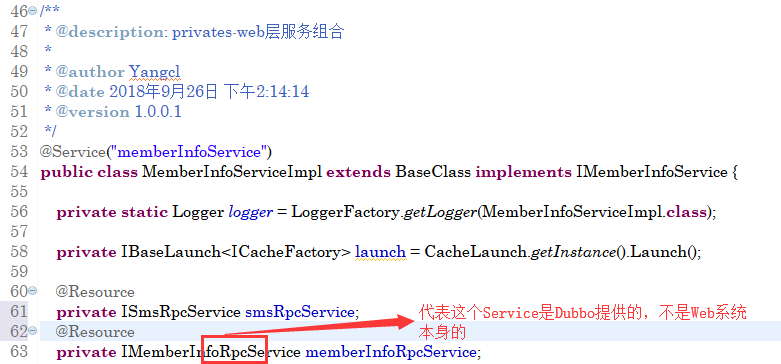
在左图中，ajaxApiProjectlist()方法传递参数使用的是一个Entity类，这里你也可以使用DTO作为参数的接收者。系统中约定：禁止使用基础对象传递参数，比如：String name,Long id等等形式的传参；因为这样不利于快速复用Service的代码。进入方法处理体内，添加一条日志。

@RequestMapping 标识：ajax\_apicenter\_api\_project\_list；**我们约定**，异步数据交互以ajax\_开头，apicenter作为当前Controller类映射标识作为第二段，api\_project\_list作为第三段，属于业务描述。其中\_list代表列表，\_add代表添加，\_update代表更新，\_delete代表删除，\_find代表查找一条记录。

针对JSP页面跳转，需要将ajax\_改成page\_，其他基础约定不变。

@param 中列出所有涉及到的请求参数，以及含义。如果请求参数复杂，请详细备注说明；原则就是让看到的人一眼知道你想做什么事情，同时会涉及到哪些参数，最大限度的减少猜测时间。

1.3.2 Service层约定

这里所说的Service层约定特指Web项目的Service和Dubbo服务的Service层，之所以特别声明是Dubbo服务的Service，是因为在Dubbo项目中，我们真正对外提供Rpc接口服务的层是RpcService层。RpcService层会包装、组织Service层中数据；这么做是为了将来有一天，如果公司想做技术转型，不再使用Dubbo，转而使用Spring-Cloud，那么现有的RpcService层中的代码，只需要改成Controller即可，我们对应的每个服务立刻就能变成Spring-boot组件，迅速的接入到Spring-Cloud框架中。在右图中，一个约定格式的Service包括

第56行：日志声明；第58行：缓存声明；以及下面的接口引入。

针对接口引入，分为两种情况

1. web项目本身的Service接口。
2. Dubbo提供的RpcService接口。

我们引入服务的书写规范是去掉大写字母“I”，同时将第二个大写字母小写，遵循驼峰格式。针对Dubbo提供的Rpc Service，为了将它和web本身的Service区分开，所以他们的结尾全部以RpcService结束。



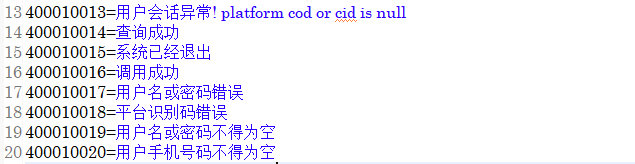
在Web系统中(注意！右边图片特指Web系统的Service)，针对Service层提供的每一个方法，其返回值一律使用com.alibaba.fastjson.JSONObject对象。这么做是因为JSON对象包含更加丰富和灵活的表意能力，能极大的简化开发人员的工作量、提高方法与方法间的通用性和复用性；另一点相对传统的返回一个视图类的做法，这种方式能够有效的防止“类爆”(类爆：随着项目的不断壮大，产生大量的视图类，相互冲突，命名困难，不易维护的现象)。

针对返回值的两个必填状态：status：success or errror；msg：消息提示，根据具体业务场景有所不同。

所有msg对应的消息提示信息都会保存在指定路径下的配置文件中，这些信息随着项目的启动被读取到缓存中。这个路劲为约定好的：src/main/resources/META-INF/matrix/info/info.project-name.\*000.properties

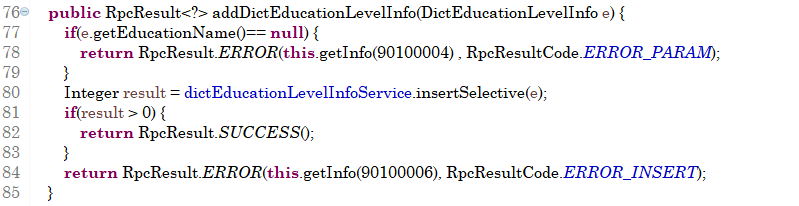
每个项目都有这个文件夹，根据project-name的不同加以区分；\*000则是在Ehcache中存在的Key，为了保证这些信息不重复，每一个项目的起始数字都是不一样的。这些由架构设计者统一控制，开发者无需关心。

如：info.matrix-manager.4000.properties(简称：info配置文件)，具体配置后的样式如下图



对于Dubbo中的Service层，通常只返回一个对应的数字、实体类或列表；因为微服务化后，体现出服务的单一职能，不会出现复杂的数据组合。不可以使用JSONObject，这个对象不支持序列化，Dubbo服务会报错。

1.3.3 Dubbo Rpc Service层约定

 这一层的约定相对简单，只需要根据具体情况返回一个RpcResult<?>对象即可。这个类的用法

请参照1.3.4节的描述。

1.3.4 Rpc层通用返回值约定 – 请认真阅读

Rpc层的通用返回对象使用RpcResult.java，存在于matrix-core项目中，路径为：com.matrix.base.RpcResult.java。项目约定：禁止任何开发人员使用自己的定义的返回对象，所有人都要统一，并且保持一致!

针对这个类，使用原则声明如下：

1. 针对查询、分页、导出等等这一类的行为，你的RPC实现类和RPC接口类(interface)返回的RpcResult对象中，泛型需要约束为一个具体的Java对象；这个泛型对象将携带调用者

所需要的信息，它也是你查询的结果。比如：

public RpcResult<PageInfo<MemberInfo>> memberInfoPageList(MemberInfoDto dto);

public RpcResult<MemberInfoView> findMemberInfo(Long id);

1. 针对添加、修改、删除等等这一类行为，如果没有特别需求，你的RPC实现类和RPC接口类(interface)返回的RpcResult对象中，泛型需要约束为一个通配符(约定优于配置，此处为

团队内部约定后的结果)。比如：

public RpcResult<?> addMemberInfo(MemberInfo e);

public RpcResult<?> editMemberInfo(MemberInfo e);

public RpcResult<?> deleteMemberInfo(MemberInfo e);

理论上来讲，这里可以约束为任何对象，因为这一类行为我们需要的只有一个结果：成功或失败，失败的话原因是什么，状态码是多少。但也会出现极少数的情况，比如添加成功后，我们需要拿到这条记录的主键ID，此时通常会将实体对象一同返回，比如：

public RpcResult<MemberInfo> addMemberInfoGotId(MemberInfo e);

实际业务中还会有更多的情况，此处不再一一列举，请开发者根据实际情况决定。

1.4 项目基础类以及常用方法

1.4.1 BaseClass

存在于matrix-core项目中，路径为：com.matrix.base.BaseClass。Matrix框架底层核心类，提供多样化的注解功能，配置文件的访问功能和日志的功能。

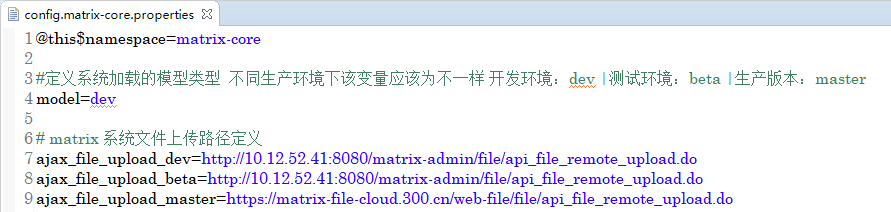
在核心底层的代码中，大部分类都会继承BaseClass。1.4.2节到1.4.5节会详细介绍他提供的4个方法。

1.4.2 BaseClass -> getConfig(String key)

通过访问每一个项目的 config.\*\*\*\*\*.properties文件，获取其配置内容；key：配置主键，会返回配置在项目中的内容。这**一类**配置文件在：

src/main/resources/META-INF/matrix/config文件夹下。Matrix的每一个项目都遵守同样的包路径结构，同时利用路径同源性的特点，极大增强系统的灵活性。

比如一个具体的配置文件：src/main/resources/META-INF/matrix/config/config.matrix-core.properties



@this$namespace=matrix-core是一个固定的标签，根据每一个项目名称的不同，名字也不一样。但通常他是和配置文件名称的中段保持一致，比如：config.matrix-core.properties。

这些配置文件的信息会随系统启动而加载到当前应用程序的Ehcache缓存中。当你的web项目或Dubbo项目引入了多个Matrix框架的子项目时，他们配置文件中的信息都会自动扫描并引入。

使用方法如下：

首先继承BaseClass.java，然后：String defaultDomains = **this**.getConfig("matrix-api.default\_leader\_service\_list\_" + **this**.getConfig("matrix-core.model")); // 取出默认跨域服务器列表

**this**.getConfig("matrix-core.model")会取出当前运行环境是什么，比如：dev/test/master等等。

1.4.3 BaseClass -> getInfo(long infoCode , Object… parms)

通过访问每一个项目的 info.\*\*\*\*.\*\*\*\*.properties文件，获取其消息提示内容。其中infoCode为文本编号，parms为拼接的字符串。其典型的两种使用方法如下：

cache.put("msg", **this**.getInfo(600010006)); // 600010006=数据查询成功!

result.put("msg", **this**.getInfo(600010077 , d.getTarget()) ); // 600010077=目标接口: {0} 不存在!

这一类配置文件在：src/main/resources/META-INF/matrix/info文件夹下，与config文件夹在同一个包下。系统启动会自动加载其中配置好的消息提示信息。

**系统约定**：

这些info配置文件的主要作用就是对项目进行国际化扩展，当公司的业务发展到足够壮大，需要跨语言来运营的时候，我们可以通过修改这些配置文件就可以立刻满足需求，

所以业务代码的消息提示中禁止出现中文，必须使用this.getInfo()方法来获取系统中配置好的消息，同时为了方便同事阅读，代码的开发者需要将对应的中文释义备注到代码右侧，

形成良好的开发习惯，不要耽误其他同事的时间。

1.4.4 BaseClass -> getLogger(Logger logger)

获取日志，打印消息等。该方法会获取BaseLog.java对象，这个类提供了系统统一的日志输出风格：[-----时间戳-----][-----当前类-----][-----消息提示信息-----]。样式如下：

[2019-01-21 16:06:24][com.matrix.cache.redis.core.RedisTemplate][获取缓存开始增量计次|Redis Key = xs-MipScoreBaseRule-2 当前增量值 = 6]

BaseLog.java提供了众多的方法，其中一个比较常用的使用方式如下：

**this**.getLogger(**null**).sysoutInfo("用户：" + view.getUserName() + " 接口名称：" + apiInfo.getString("name") + " target = " + head.getTarget(), **this**.getClass());

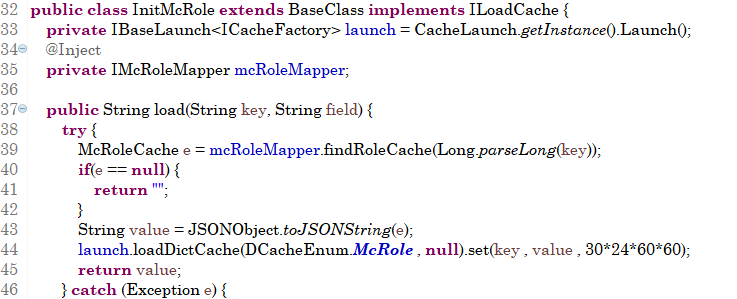
1.4.5 BaseClass -> @Inject标签

Matrix系统提供的核心标签。随着项目的扩大，分包越来越多，如果都让Spring配置文件去一个个扫描，项目的启动时间会很长。所以为了解决这个问题，系统在Spring

加载的时候，只会扫描com.matrix.controller、com.matrix.service和com.matrix.dao三个层；如果是Dubbo服务项目，还会额外扫描com.matrix.rpcimpl。Spring加载

完成后，BaseClass会获取当前Spring的上下文环境，拥有操作其中所有Bean的能力。也就是说，其他的类中，如果用到service下的类、dao下的类，只需要用@Inject标签

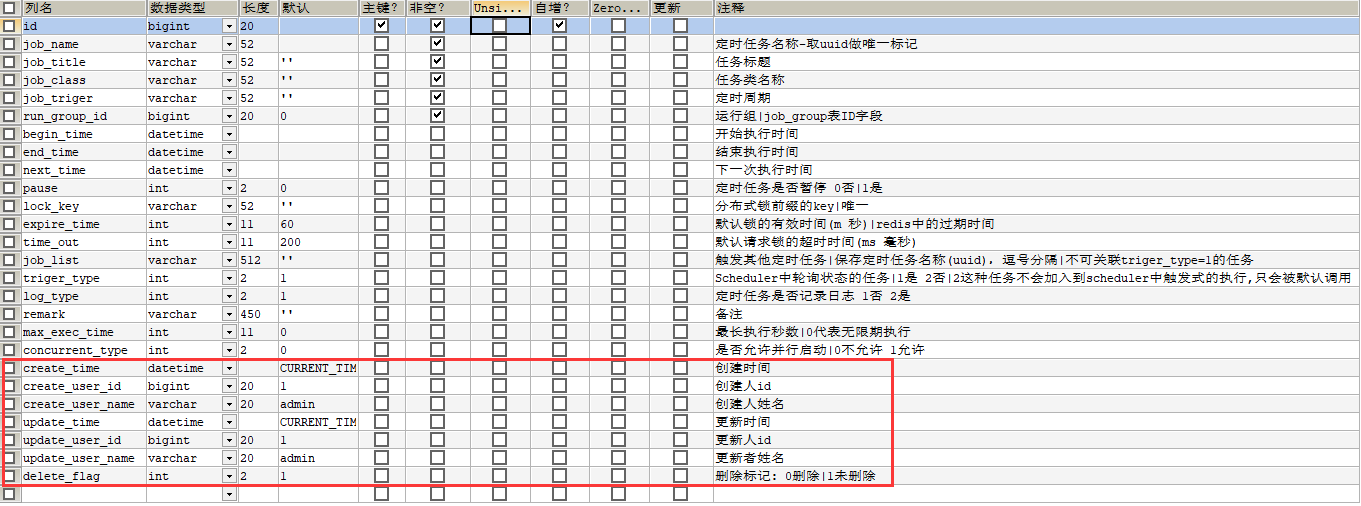
引入即可，非常灵活。比如：



1.4.6 快速开发指南

本节将重点介绍，一张数据库表和整个业务系统的融合过程。

1、首先建立好一张表：



数据表必须包含红框中的7个固定字段。

2、利用Mybatis generator插件生成对应包下的实体类和xml配置文件：

* com.matrix.pojo.entity.JobInfo.java

与数据库表一一对应的实体类，即：Entity。由Generator插件自动生成，但还需要你在其基础上删除上图红框中的7个固定字段，同时继承BaseEntity.java，细节如下图：

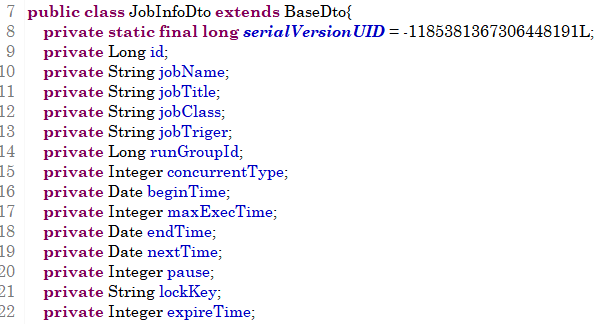


* com.matrix.pojo.dto.JobInfoDto.java

DTO数据传输模型。为了不污染实体类中的字段，让实体类始终保持与数据库字段的一致性，通常来讲不会使用Entity来传递页面或者接口发送过来的请求

而是使用Dto来传递。Dto中可以添加实体类中以外的其他字段，这些字段通常所属于很多张表，然后在Service层重组、分发。Entity主要用了给他对应的那张

数据库表添加数据、更新数据用；同时添加和更新这两个操作是Mybatis Generator插件自动生成好的，使用的正好是Entity。示例如下：



* com.matrix.pojo.view.JobInfoView.java

数据联合查询视图类。View-视图的概念来源于数据库，数据库中的一种结构叫做联合视图，就是将多表联合后展示信息。

与Dto的情况稍有不同，Dto是用来传输数据使用，View则是专门用来返回数据使用。与Dto相同点在于也是为了避免污染Entity中的字段。通常在数据查询的时候可能会用到多表联查的情况(在分布式系统

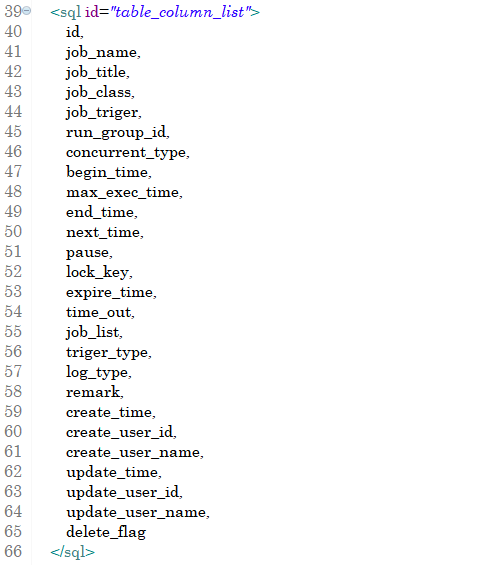
不提倡多表联查，最好避免)，每个表中各有几个字段返回，此时禁止将非实体类中的字段写入到实体类。系统约定使用视图类来保存其他表的字段。还有一种情况，在分布式系统中，Mycat作为代理进行分库后，

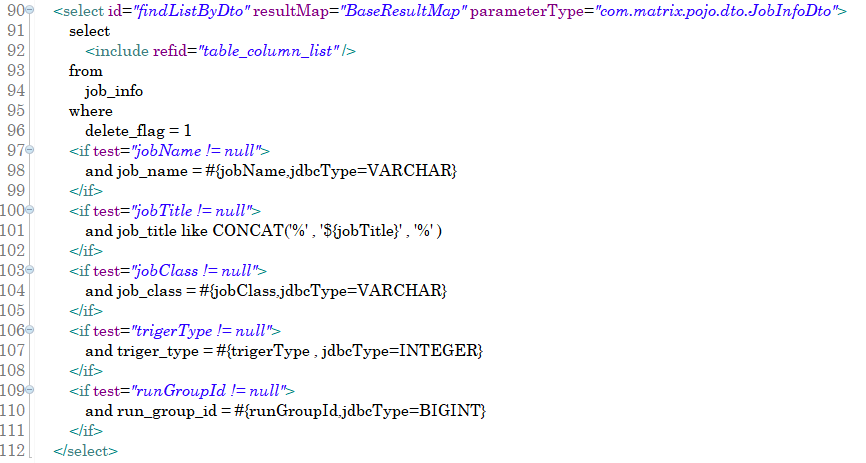
每个独立的服务进行单表查询返回记录，此时需要把多个服务返回的信息进行重组，然后保存到视图类中。如下图：



* JobInfoMapper.xml

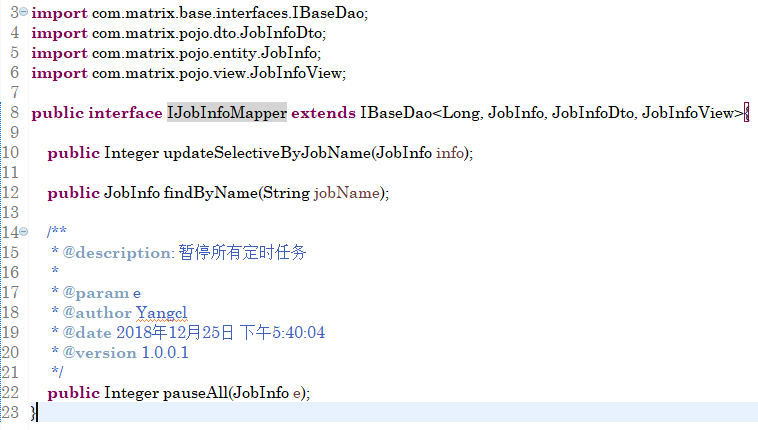
Mybatis Generator自动生成的xml配置文件。生成之初，其中的代码是凌乱的，需要开发者进行整理，规范好脚本代码格式，原则是：xml文本中的代码对其，SQL脚本错落有致。

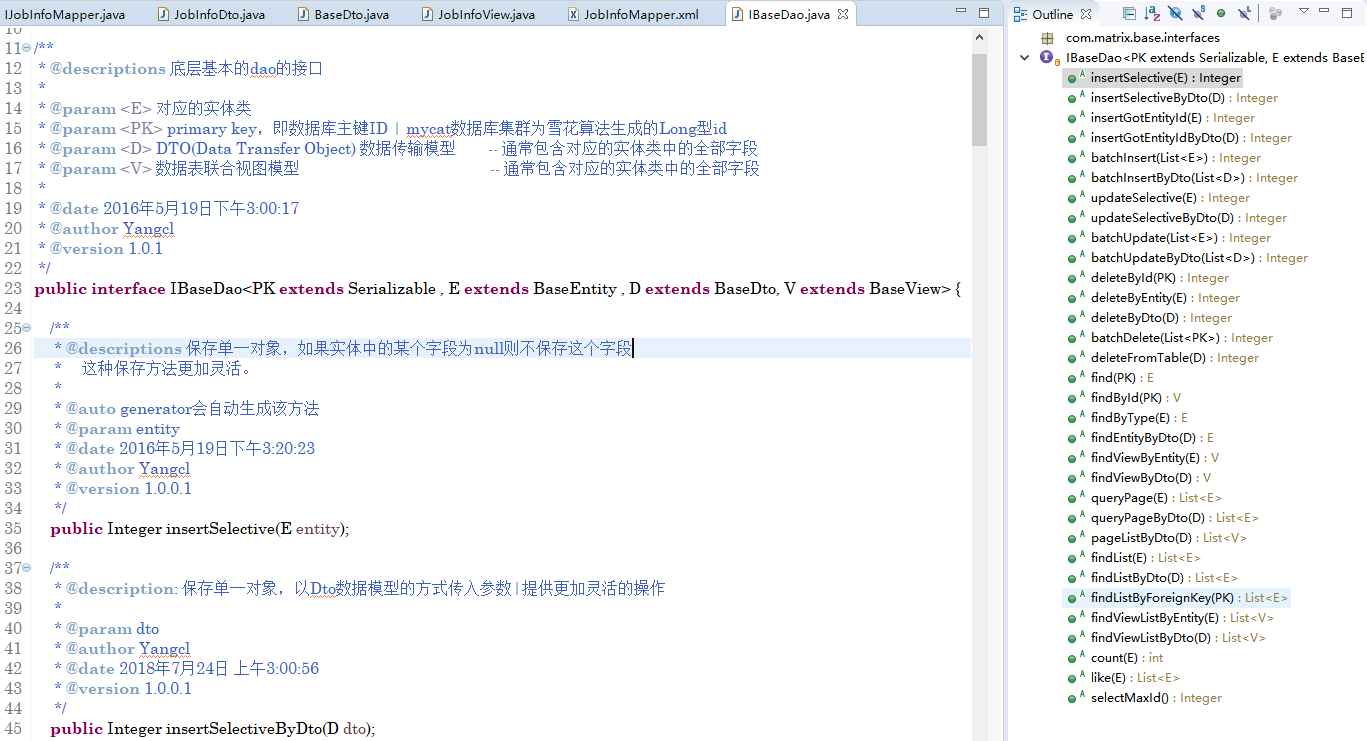


3、上述基础准备完成后，开始准备接受泛型约束。

Mybatis Generator自动生成的IjobInfoMapper.java的基础上，删除自带的增删改查方法，继承IbaseDao，如下图所示



com.matrix.base.interfaces.IbaseDao.java存在matrix-core项目中，封装了大量实际项目中会用到的方法：

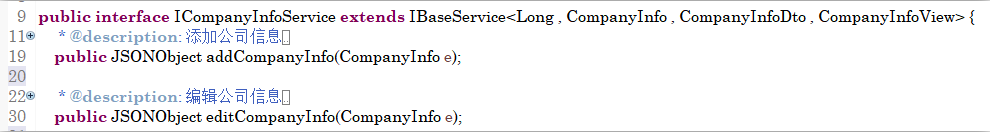


泛型约束的第一步完成。达到这一步，系统基本的增删改查就已经全部具备，只有部分特定的脚本需要开发者对应好BaseDao中提供的方法，在xml文件中填写好即可。

另一种情况是泛型约束Service。假如这个表会是你的主表，那么你可以继续约束到Service层，com.matrix.base.interfaces. IbaseService是接口类，com.matrix.base. BaseServiceImpl是实现类。

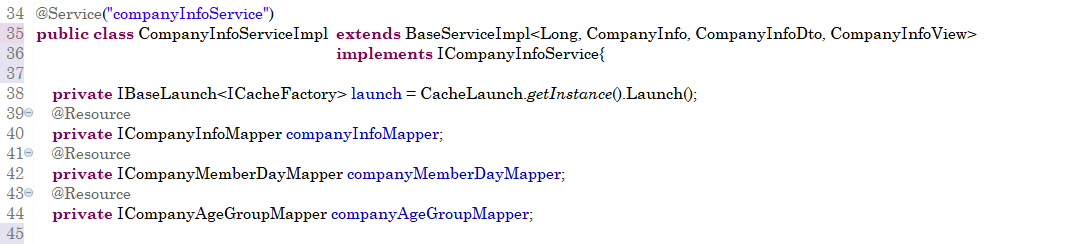
以CompanyInfo：公司信息为例，单纯的单表操作，关联性小，那么他可以在上面的基础上继续进行约束。

**接口层**：



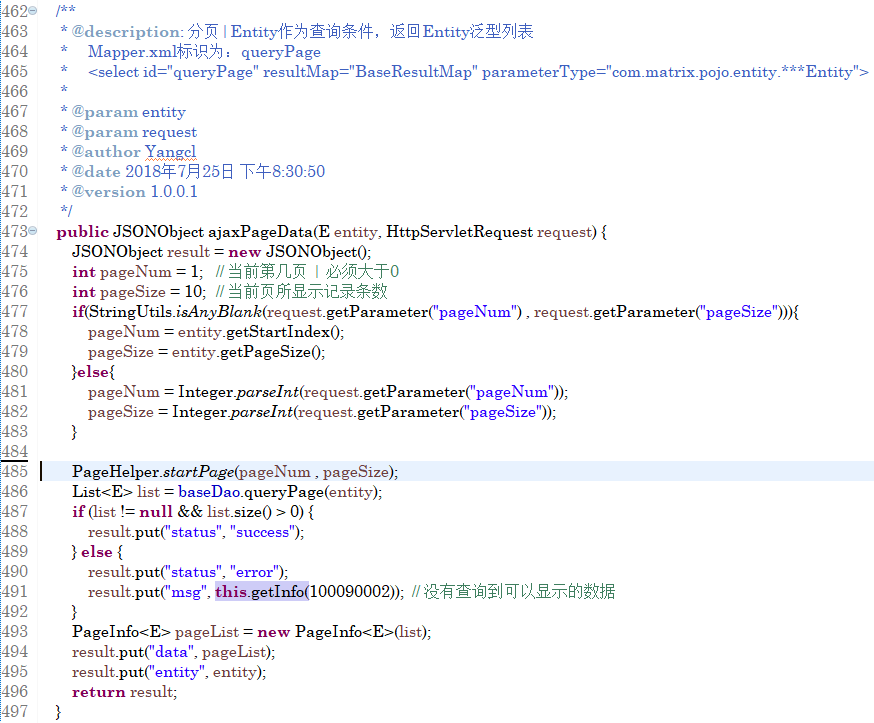
基础方法不满足使用，我们可以自己定义方法。

**实现类**：

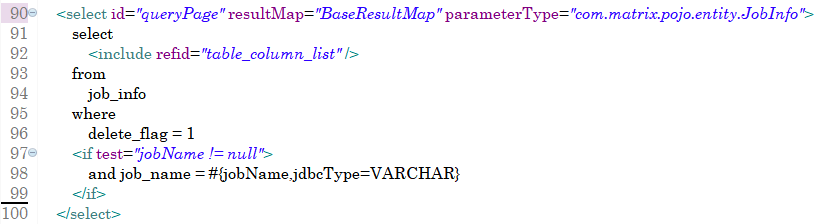


1.4.7 分页操作

在com.matrix.base. BaseServiceImpl.java中提供了丰富的分页查询方法(5种)，如果不满足你的需求，都可以作为你参考的蓝本，如下图所示，Web系统查询的分页方法封装：



在这个方法中，对应的数据库查询声明是：List<E> list = baseDao.queryPage(entity); 其中queryPage方法在mppper.xml配置文件中需要写好对应的SQL语句，比如：



可以看到，这里无需传入limit参数来限制分页，无需书写Count查询。项目在底层结合了pagehelper-4.0.3.jar，其中的com.github.pagehelper.PageHelper方法可以帮助我们

完成所有分页相关的操作。也就是说，对于整个分页的操作来讲，只需要写这么一句话就可以了，这对于业务开发者非常便捷。下图针对Dubbo Rpc层的分页封装，原理上与上面的代码大同小异：



具体请看代码，这里只列举出给Web系统查询公共库的和给Dubbo查询业务库的两种方法。

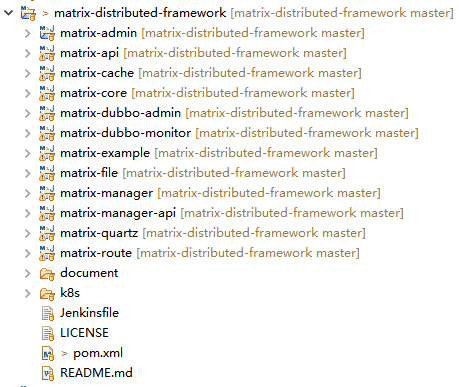
2、项目结构规范

本节中将重点介绍基于matrix-distributed-framework(以下简称Matrix)通用底层框架开发具体业务的方法和规范。这里会具体展示如何快速构建一个基于Dubbo的微服务生态环境，以及底层框架各个子项目的

分工、用途以及关联关系。

2.1 项目结构介绍

Matrix项目结构如下图：



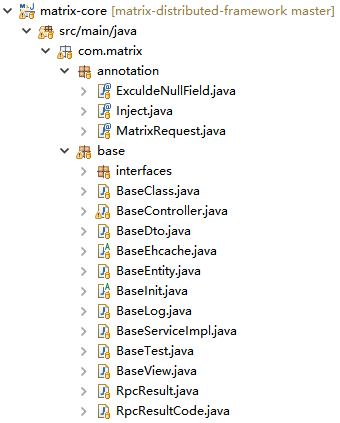
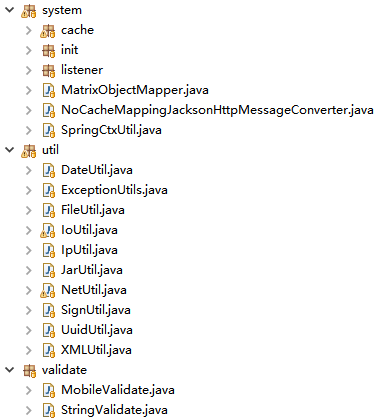
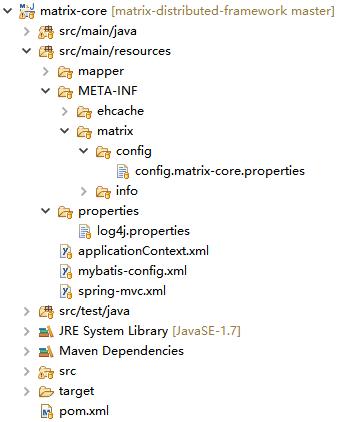
2.1.1 matrix-admin

矩阵管理平台，web项目，基于SpringMVC架构，提供底层技术管理功能，如：平台权限系统、Redis缓存系统、平台API系统、DubboAdmin控制台(部分功能内嵌，70%功能尚未迁入)、

分布式节点路由功能、分布式定时任务管理等。

2.1.2 matrix-core

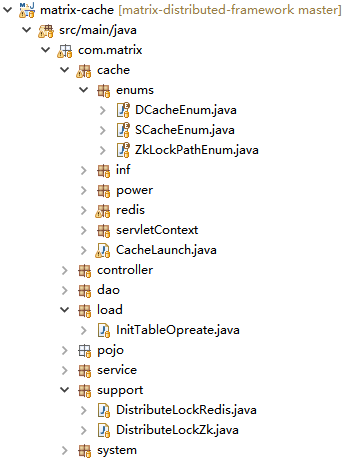
分布式系统的核心底层项目，提供大量基础工具与脚手架支持，所有衍生子项目、业务项目都需要在POM中集成这个Jar包。项目结构如下图：

2.1.3 matrix-cache

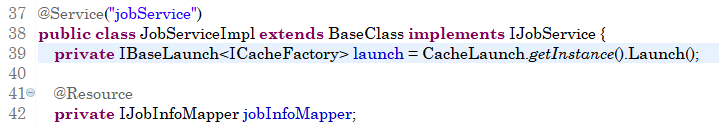
系统缓存封装，此项目提供Servlet Context、Ehcache和Redis三种缓存模式，Matrix系统使用Ehcache作为系统一级缓存，Redis作为系统二级缓存，Mycat做数据库分库中间件。系统一级缓存的读写效率

是二级缓存的上万倍，但缺点是需要项目经验丰富、技术能力强的人才能在具体业务场景中使用。由于是分布式系统，故对Ehcache不了解的人请先详细了解他的使用方法，以免出现大的差错。



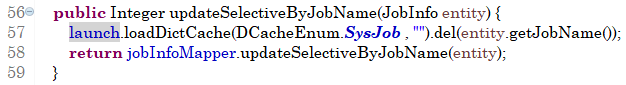
DcacheEnum.java定义了数据字典相关枚举信息，ScacheEnum.java定义了业务字典相关枚举信息。CacheLaunch.java是整个缓存的入口类。DistributeLockRedis.java提供基于Redis的高性能分布式锁，DistributeLockZk.java提供

基于Zookeeper的高可靠分布式锁，这两种锁可以根据具体的业务场景，自由选择。缓存的使用方式如下：

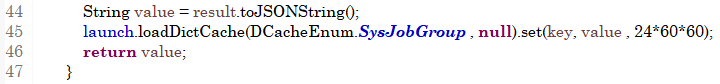


在业务的Service层，将缓存声明，然后在具体的方法中进行调用，除了launch.loadDictCache(DCacheEnum.***SysJob*** , "").get("redis-key"); 方法，其他的loadDictCache或的第二个参数都可以为空("" or null)

如下图：



或



当设置一个缓存的时候，请务必添加超时时间。针对缓存的Get操作，这里有特殊要求，部分代码片段如下：



从左图的代码段可知，当获取缓存的时候是通过反射指定的类来实现的，具体使用方式如下：

String value = launch.loadDictCache(DCacheEnum.**SysGroup** , "InitSysGroup").get("uuid-1");

这其中的InitSysGroup标识的是一个类，他存在于系统特定的包下：com.matrix.load。这个类的命名约定为DcacheEnum枚举中的**SysGroup**变量的前面添加“Init”。矩阵管理后台会根据这个规则来定位和获取其对应的值。

系统把数据库查询、拼接数据的业务的代码统一归纳到一个具体的处理类中，这么做的好处是查库放到缓存的这个过程，只有一个入口，业务开发人员不必担心会有多个地方存在同一个缓存的获取行为，以架构约束的方式，将与业务代码无关的缓存获取操作从Spring的Service层中分离出来，单独保存，最大限度的减少因为遗忘、疏漏而产生的缓存不对称的bug。

同时，此方法做了防雪崩处理，如果一个缓存Key在10分钟内尝试20次连续的数据库查询操作，那么将会在10分钟内连续返回空字符串；也就是说业务开发人员无需在考虑防雪崩的问题。如下代码段展示了一个简单的缓存未在Redis中命中，从而去数据库中查询数据、拼装JSON串儿的过程：

public class InitStoreInfo extends BaseClass implements ILoadCache {

private IBaseLaunch<ICacheFactory> launch = CacheLaunch.getInstance().Launch();

@Inject

private IStoreInfoMapper storeInfoMapper;

public String load(String key, String field) {

StoreInfoDto dto=new StoreInfoDto();

dto.setCid(Long.parseLong( key.split("-")[0]));

dto.setId(Long.parseLong( key.split("-")[1]));

StoreInfo storeInfo = storeInfoMapper.findEntityByDto(dto);

String value="";

if (storeInfo!=null){

value = JSONObject.toJSONString(storeInfo);

launch.loadDictCache(DCacheEnum.StoreInfo , null).set(key , value , 30\*24\*60\*60);

}

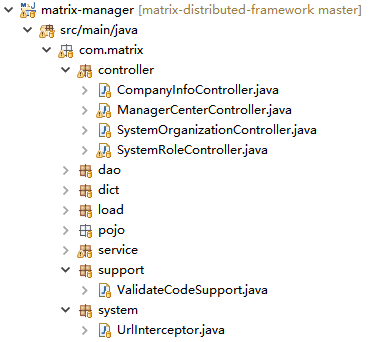
return value;

}

}

2.1.4 matrix-manager

系统权限相关项目。



CompanyInfoController：公司信息控制器

ManagerCenterController：权限分配与登录信息控制器

SystemOrganizationController：组织机构相关控制器

SystemRoleController：系统功能与角色相关控制器

ValidateCodeSupport：验证码支持

UrlInterceptor：系统权限拦截器

通用的权限控制中心，系统会在矩阵后台为子平台分配超级管理员权限的账号；然后子平台的超级管理员根据所分到的功能列表，再次创建属于他们公司的角色和用户。

2.1.5 matrix-manager-api

封装了matrix-manager项目，使用项目中的Service层，结合matrix-api项目，对外提供API支持，不使用Controller。

2.1.6 matrix-api

平台内置的API网关项目。所有基于Tomcat的容器型项目，在POM中加载这个Jar包后，都具有API接口提供能力。API项目采用MD5加密的方式进行验签，矩阵系统会对请求者提供对应的公钥和私钥。

一个完整的接口请求数据结构如下：

{

"head": {

"target": "MANAGER-API-100",

"accessToken": "",

"client": 3,

"version": "vsesion-2.0.0.1",

"requestTime": "2018-12-14 16:47:09",

"channel": "mip会员平台PC前端",

"key": "133C9CB27DA0",

"value": "58b6e0bd4d34a35b9773d4762be0f521"

},

"data": {

"userName": "admin-lqx",

"password": "xxxxxx",

"validateCodeKey": "b89e4919-2620-4c52-a371-240e452fbc3b",

"validateCode": "SOFH",

"platform": "133EFB922DF3"

}

}

target：api标识符

accessToken：登录后的令牌，过期时间半小时

client：3代表客户端，固定值

version：版本号，vsesion-2.0.0.1，固定值

requestTime：本次请求发起时间，md5加密依据之一

channel：渠道，固定值

key：公钥，由Leader平台系统进行分配

私钥：此处不体现，由系统分配给对应客户端的请求者。

value：md5加密后的结果。

data：请求所携带的参数，其中platform为固定值，每个平台对应的值不同，由数商管理平台统一分发。

在系统设计中，所有的API信息均以缓存的方式存在于Redis中，每一个接口都可以针对指定的域名进行跨域操作，接口间的跨域问题由底层统一解决，不用开发者处理。完整的API JSON结构信息如下：

{

"id": 80160001,

"name": "订单信息", ------------------------------------------------ 接口名称

"target": "API-ORDER-INFO", -------------------------------- 接口标识

"discard": 0, --------------------------------------------------------- 这个api是否废弃|0:废弃 1:使用中

"createUserId": 1,

"updateUserId": 1,

"seqnum": 1,

"module": "matrix-api", ----------------------------------------- 接口所属项目

"remark": "ORDER-INFO", ----------------------------------- 备注信息

"updateTime": 1512032208000,

"list": [ --------------------------------------------------------------- 当前接口跨域名单列表

"http://api.baidu.com",

"http://sub.model.firos.com.cn"

],

"processor": "private.order.OrderInfomation", --------- 对应的处理类

"parentId": 1,

"domainIds": [

"4",

"5"

],

"createTime": 1511922954000,

"atype": "private", ------------------------------------------------- 私有接口，公司内部项目间调用

"domain": 1 ---------------------------------------------------------- 标识当前接口可跨域请求

}

2.1.7 matrix-file

文件操作。这个项目中提供了操作文件的接口，当一个Tomcat项目在POM中引入此Jar包后，他即可成为一个简单的单点文件服务器。

2.1.8 matrix-dubbo-admin/ matrix-dubbo-monitor

Dubbo Admin控制台相关，只迁移了30%的功能。

2.1.9 matrix-example

提供简单的系统示例，减轻开发人员的学习成本

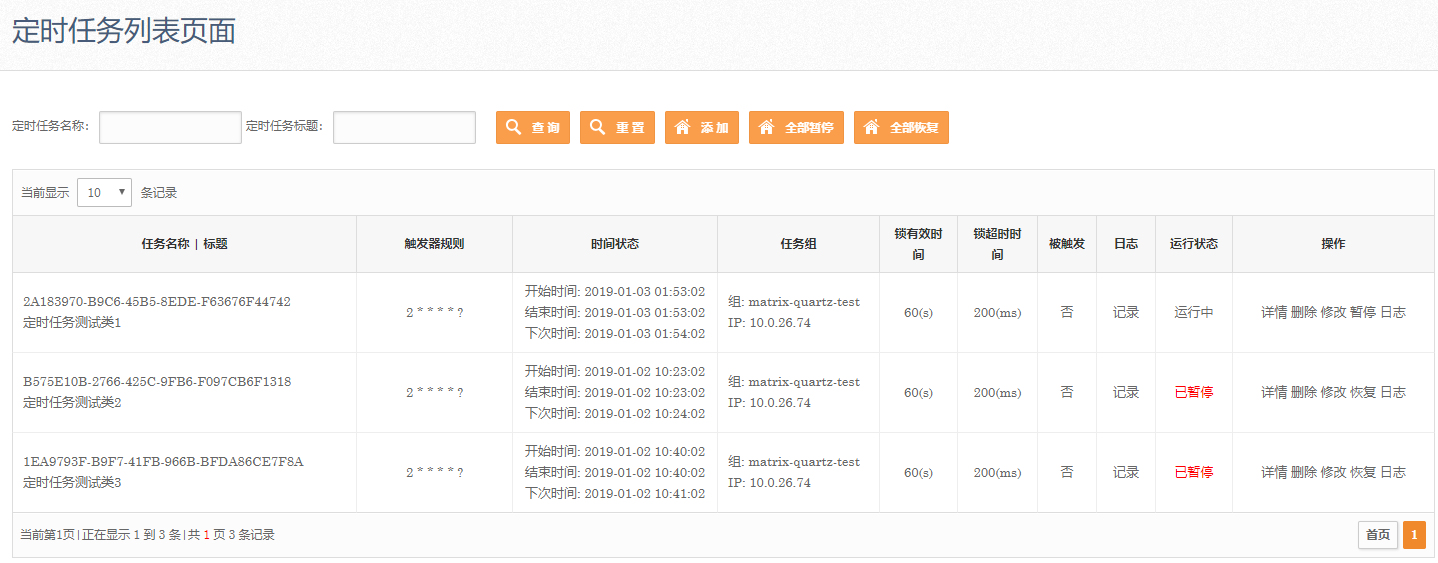
2.1.10 matrix-quartz

为分布式系统定时任务提供支持。基于Quartz，系统对其再次进行了高度封装，使得开发人员可以轻而易举的完成定时任务的开发工作。在矩阵后台，有非常人性化的界面来配置、修改、查看和关闭你想要操作的定时任务。

系统将定时任务的并发控制放到了底层，不需要业务开发人员干预。当一个定时任务执行完成后，通过配置还可以顺序性的触发另一个定时任务。定时任务可以配置是否记录日志，如果记录则可以查看定时任务的执行情况。定时

任务分组功能，可以让定时任务在指定的服务器上执行，每个分组对应一个或几个服务器的IP地址。定时任务项目：matrix-quartz可以配置在基于Tomcat的web项目中，也可以配置在Dubbo服务中，注意：无需在Spring配置文

件中添加任何xml配置，后台界面全部都能帮你搞定。下面将展示相对应的功能截图：











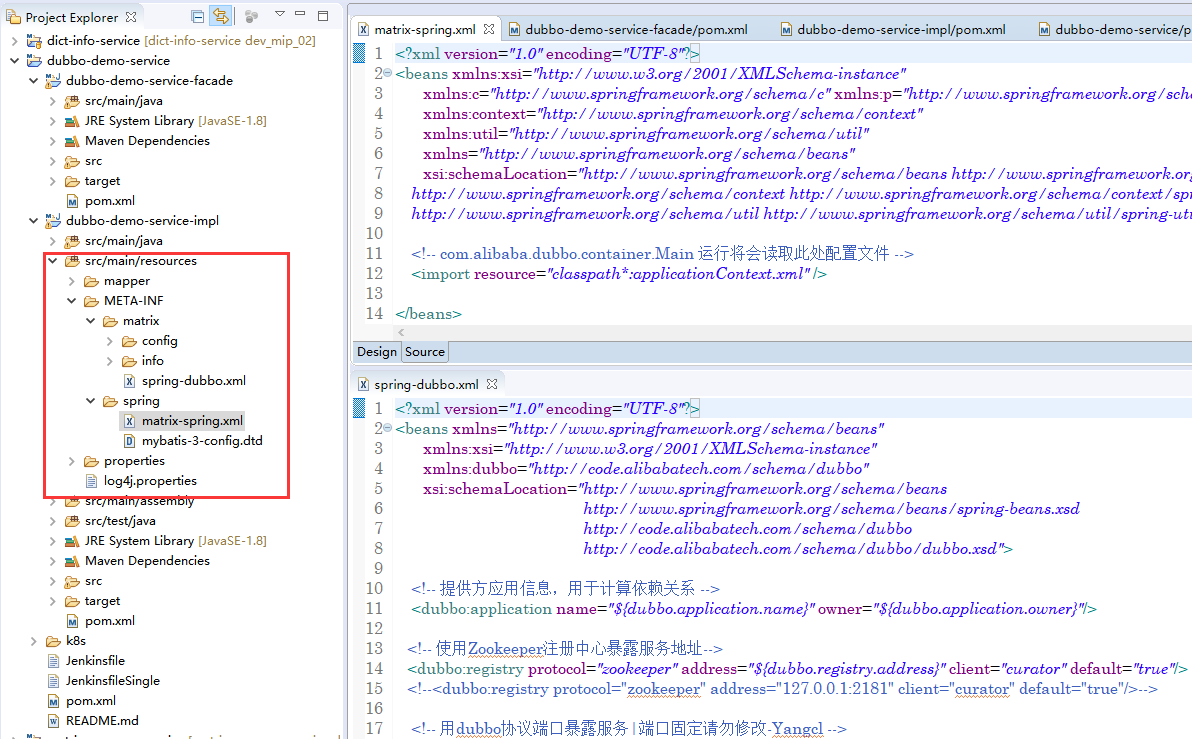
2.1.11 matrix-route

分布式系统节点路由，这个功能让矩阵系统拥有控制所有基于Matrix底层的服务节点的能力。系统高级功能，技术过于偏向底层，业务上基本使用不到，故不做介绍。

2.2 新项目搭建过程与细节

Matrix在快速构建一个系统的时候，不会从头开始搭建整个项目框架，而是选择复制！因为没有什么比这样来的更快捷。在复制一个项目的时候，需要改动一些名称，即可使用。在Matrix的架构规划中，无论是搭建

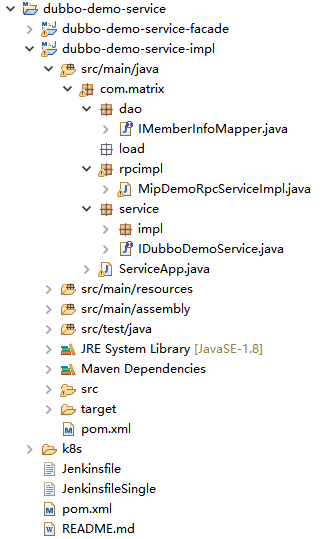
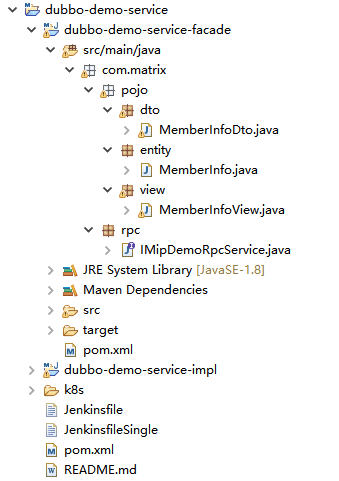
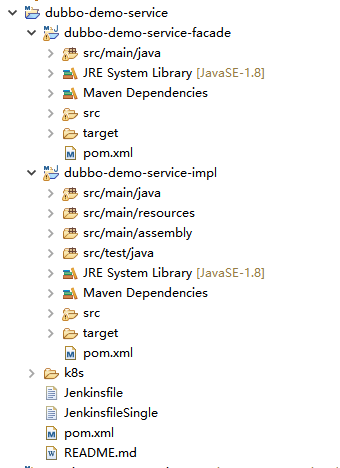
基于Dubbo的微服务项目还是基于Tomcat运行环境的Web项目，他们都会共用一套核心底层代码，不需要这些项目添加属于自己Spring xml配置文件。如下图：



可以看到，他们不包含任何与Spring属性配置相关的文件，只是一些引用(web项目原理相同，截图不再展示)。这些基础配置全部由matrix-core项目来负责，业务项目只需要引入这个jar包的maven坐标即可。

了解完这些项目搭建的先决条件后，我们开始分两个分支来详细介绍Dubbo项目和Tomcat项目的搭建过程。

2.2.1 基于Dubbo的微服务项目搭建过程



每新加一个Dubbo微服务项目，通常会复制dubbo-demo-service项目。然后根据具体的业务为他重新命名即可，从而达到快速复制一个项目的目的。dubbo-demo-service遵从Maven的项目结构，他是一个父级项目

同时包含2个子项目：dubbo-demo-service-façade(门面项目) 和 dubbo-demo-service-impl(实现类项目)。关于父项目中的文件夹和文件：

k8s：docker相关文件。

Jenkinsfile：Jenkins部署使用，项目上线需要和运维同事沟通，怎么改。

* **关于子项目**：

dubbo-demo-service-façade：

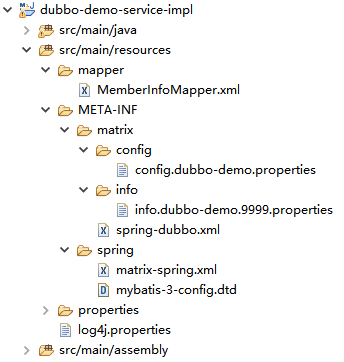
项目中的dto|entity|view相关的pojo存放在这里，方便其他项目进行引用。

dubbo-demo-service-impl：

真正的服务提供者，提供包括dao层、rpcimpl、service、resources、assemble构建和基于Mybatis的xml配置文件等等内容。其中resources和assemble/maven-conf下的内容需要做调整。

通常这个项目会引入：matrix-core、matrix-route、matrix-cache、matrix-file和dubbo-demo-service-façade等基础jar包作为支持。

**resources**：



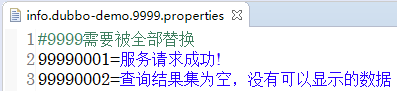
**mapper**：文件夹下存放数据库的xml配置文件；

**META-INO/matrix**：

config：系统配置文件，@this$namespace为固定标识；dubbo-demo为项目名(去掉service)，建立新项目，此处需要修改。示例如下图：



info：系统消息提示文件(国际化配置文件)，所有的消息提示禁止写入到代码中，必须写入到info配置文件中。

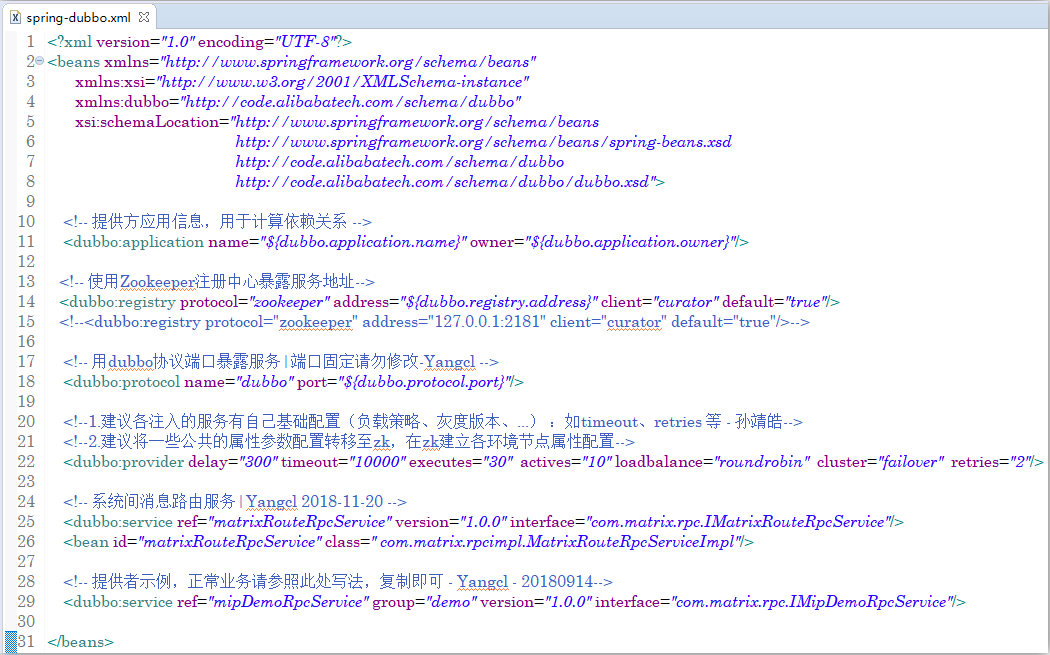


info.dubbo-demo.9999.properties命名原则：dubbo-demo项目名称(去掉service)；9999顺序描述码，9999不可以直接使用，需要

架构设计者统一分配给每一个新加入的项目。在/matrix-distributed-framework/document/Readme.txt中记录了每一个项目的顺序码。

顺序码不可重复！否则会被系统覆盖。

**META-INO/matrix/spring-dubbo.xml**：Dubbo提供者和消费者的配置文件。



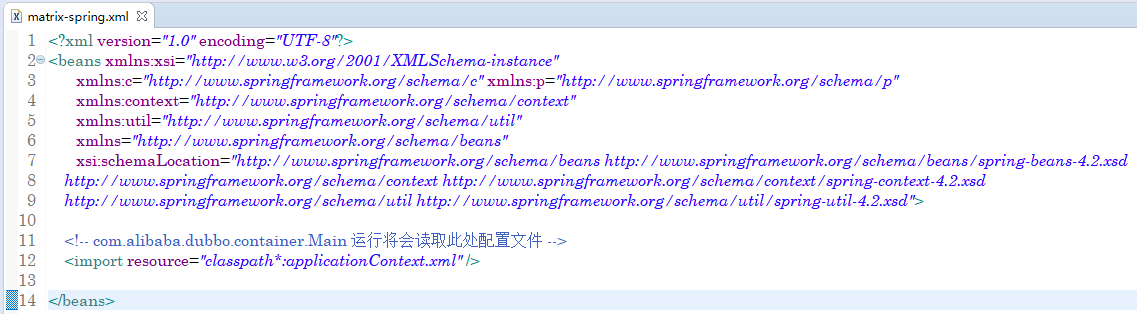
matrixRouteRpcService：

系统间消息路由服务是打通各个节点间底层通信的关键服务，如果删除，则此节点形成孤岛。

mipDemoRpcService：

一个提供示例，使用完成后删除即可。

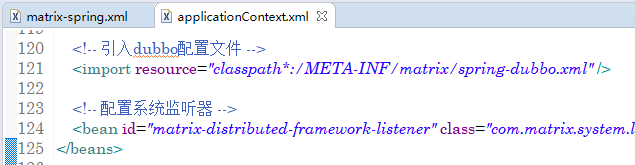
**META-INO/spring/matrix-spring.xml**：



META-INO/spring是Dubbo约定的配置文件加载位置，而Matrix将所有与Spring配置文件相关的行为托管于matrix-core中，也就是说matrix-core中的applicationContext.xml文件会

加载所有Spring额外的配置文件。

如下图：



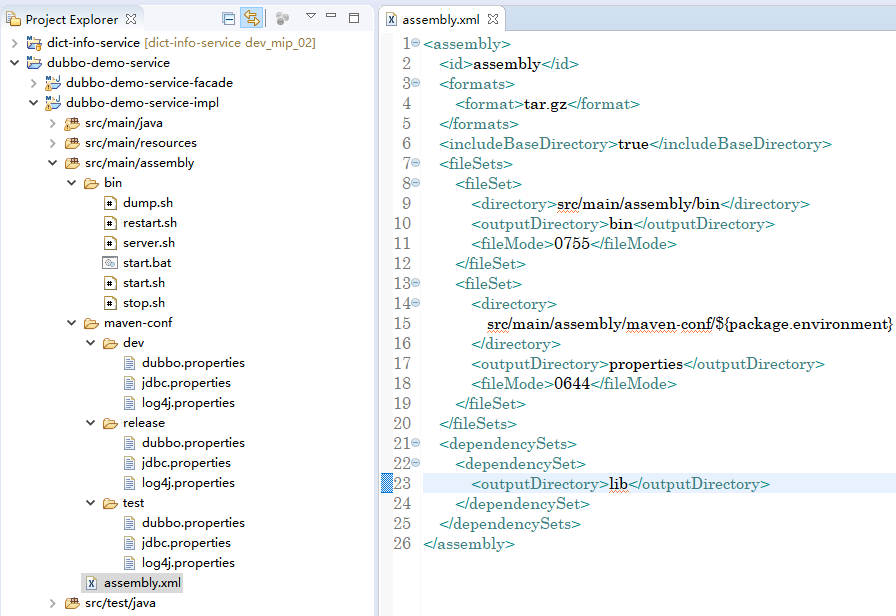
也就是说，当以com.alibaba.dubbo.container.Main作为容器来运行的时候，Dubbo容器首先从META-INO/spring文件夹下

遍历所有Spring配置文件，matrix-spring.xml被Dubbo容器加载；而matrix-spring.xml引入了matrix-core中的applicationContext

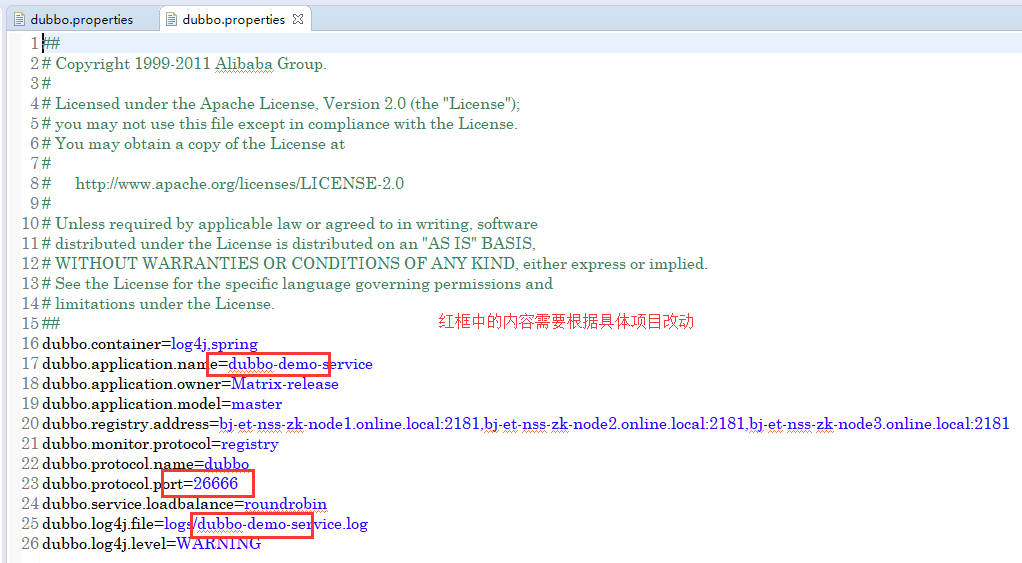
.xml，这个配置文件会完成一系列Spring框架的配置，比如：自动扫描使用注解的包、配置数据源、配置事务通知属性等等；但他最重要的一点是：加载Dubbo的服务提供者/消费者的配置文件，完成Bean的初始化。

**properties**：本地开发模式会读取此处配置，包含dubbo.properties和jdbc.properties。

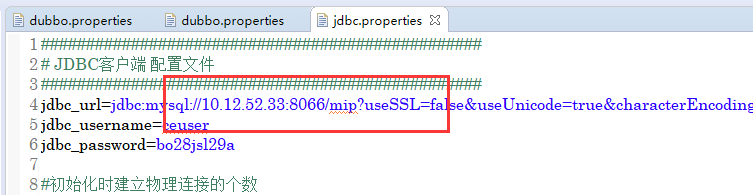
**assemble/maven-conf**：项目的差异化配置文件



**dubbo.properties**：



**jdbc.properties**：根据具体的业务，这里的数据源有所不同，可以选择链接公共库或者业务库。



2.2.2 基于Tomcat运行环境的Web项目搭建过程