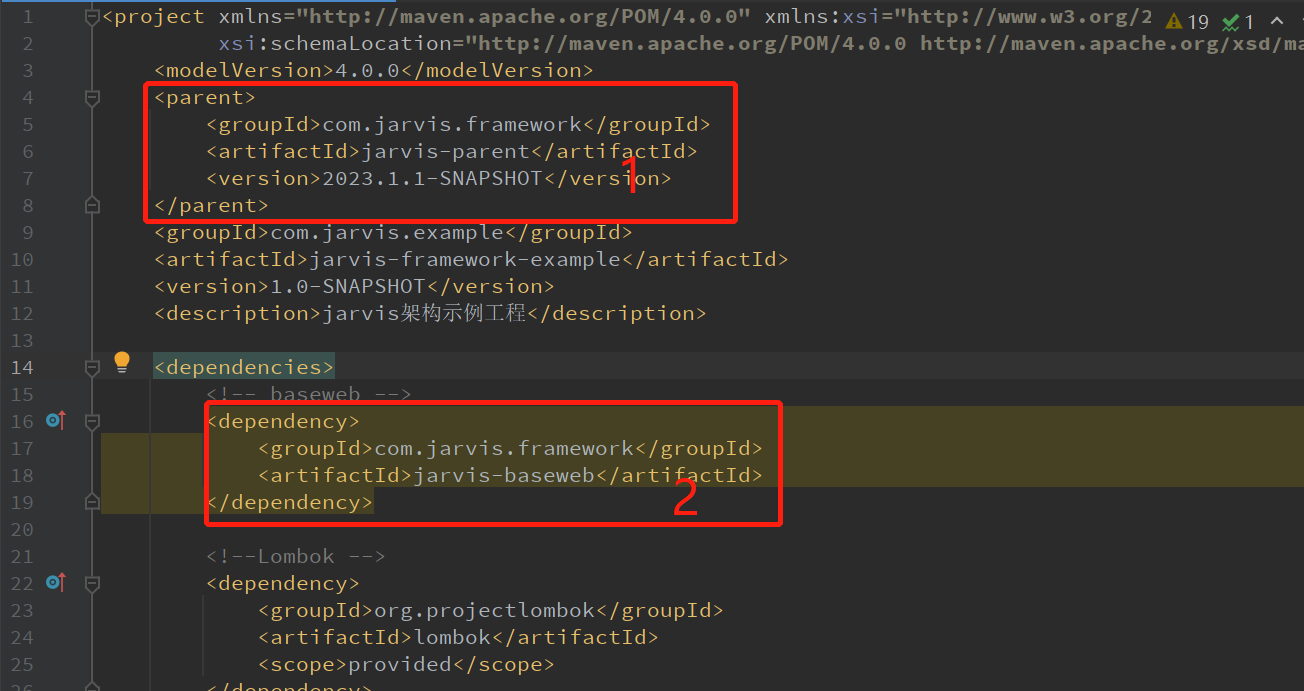
# 2023版架构使用说明

注：本文档是在单体应用上开发使用介绍，如果是微服务开发，在pom.xml里jar引用、appliction.yml配置及包结构上会有区别，但在使用上都一样的。

## Maven工程引用

在pom.xml文件中引入2023版本架构方式：

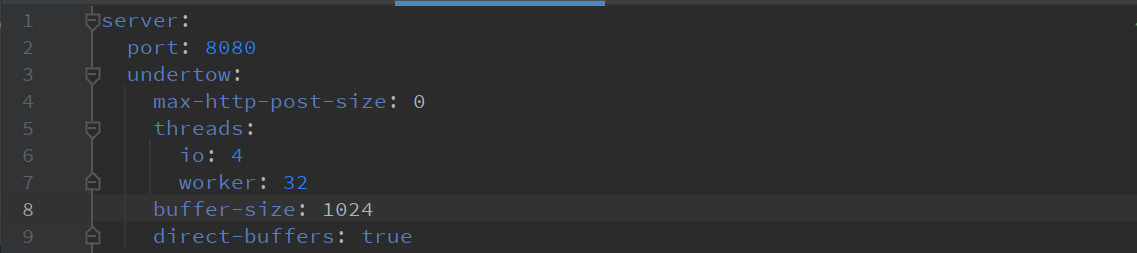
1. 在pom.xml中parent中引入jarvis-parent
2. 在pom.xml中dependencies中引入jarvis-baseweb



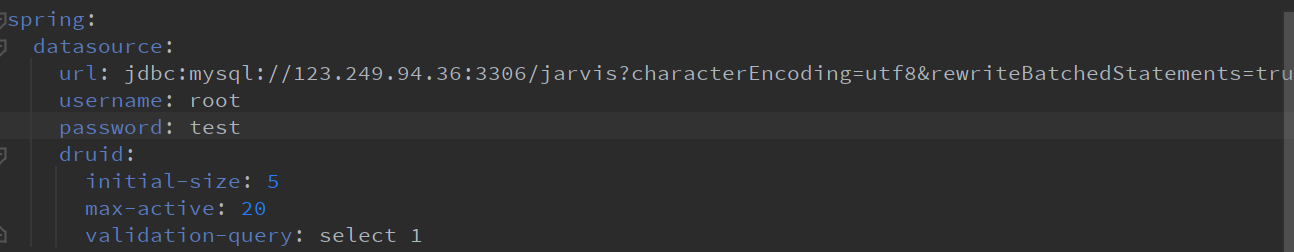
注：当前框架版本号是2023.1.1-SNAPSHOT，引入jarvis-parent后，就会自动管理框架依赖及spring boot所依赖的jar版本号，如上图中jarvis-baseweb、mysql和oracle驱动包不需要指定版本号（是因为框架中已经管理好了版本号）。具体有哪些jar不需要指定版本号，可以到本地maven仓库中查看jarvis-dependencies和spring-boot-dependencies中的pom.xml文件

## 配置文件application.yml

1. 配置容器（undertow），相当于tomcat



1. 配置数据源（使用Druid连接池）



1. Session过期时间配置（单位为秒）



1. 安全认证配置（默认表示用户登录了就可以访问）

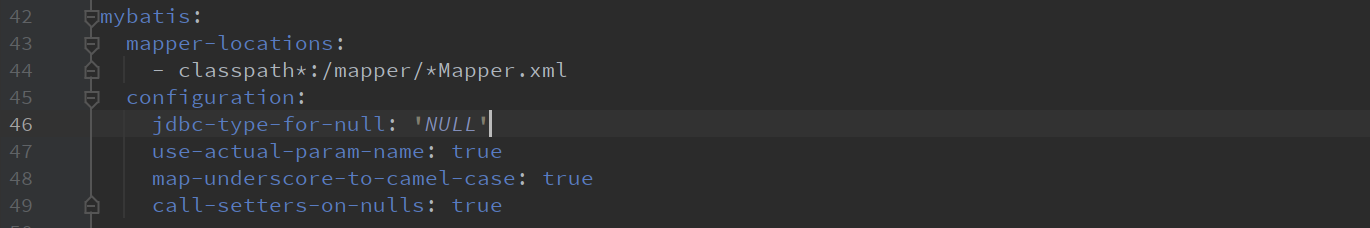


1. permit-urls表示不需要安全认证的url（如果登录时，可以获取到当前用户信息）
2. deny-urls表示禁止访问的url
3. authority-urls表示需要有权限的url，其中｛url: /city/\*\*, authorities: city, superadmin｝表示一个url，需要（city, superadmin）其中一个权限就可以访问了
4. ignore-urls 表示不需要认证的url（如果登录时，都获取不到当前用户信息），一般针对css/js等资源放行
5. 雪花ID配置



注：如果集群时，同一应用中不同服务上的worker-id要不一样，取值范围0~31

1. Mybatis配置



表示所有mapper.xml统一放在resources/mapper目录下，如下图所示



## lombok简单使用介绍

Lombok是一个可以通过注解来帮助我们简化消除一些必须有但显得很臃肿的Java代码的一种工具，通过使用对应的注解，可以在编译源码的时候动态添加源码。

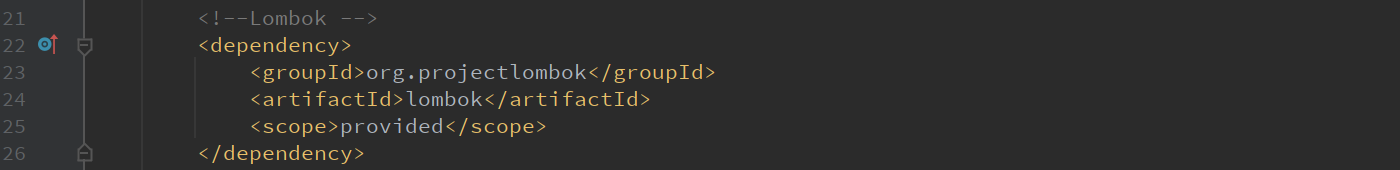
使用lombok时，开发工具IDE必须安装lombok插件，同时在pom.xml中引入lombok所需要的jar

例如在实体中经常见到一堆Getter和Setter方法，这些方法是必要的不可缺少的，但是这些代码感觉却像是“垃圾”，看起来重复而臃肿，看起来也不美观，也不简洁清爽，可以使用lombok，在类上直接使用@Getter @Setter 这两个注解或者@Data注解，那么代码在编译的时候会自动帮我们生成这个类中所有字段对应的Getter和Setter方法，因此实体类中只要有属性就可以了，这样实体类看起来变得简洁很多。

1. 开发工具IDE必须安装lombok插件

略

1. 在pom.xml中引入lombok所需要的jar

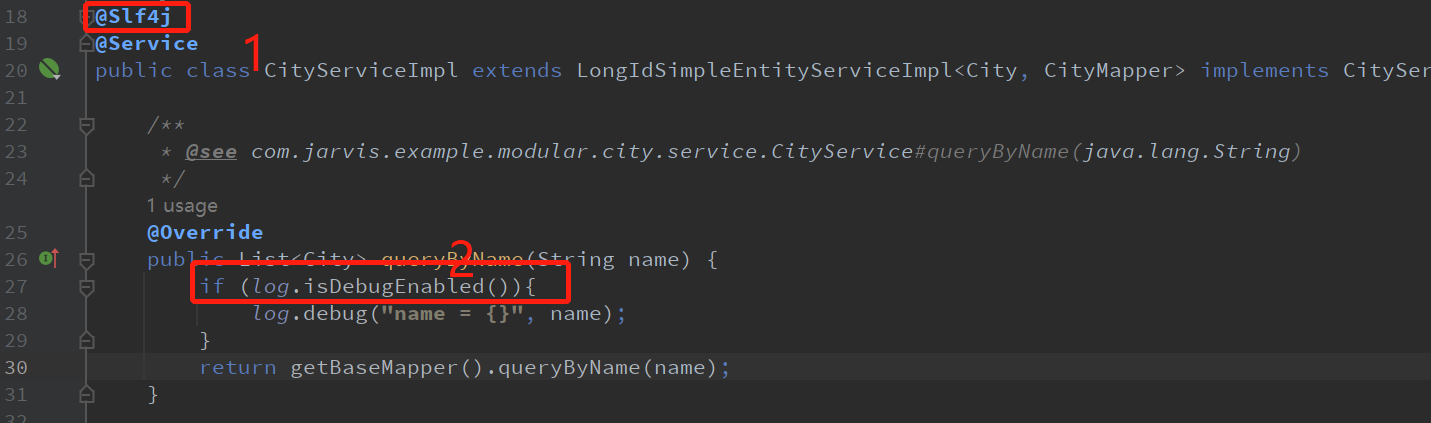


1. 实体类使用示例



其中@Getter @Setter就是为实体类自动生成Getter和Setter方法，@ToString为实体类生成toString方法，@EqualsAndHashCode为实体生成equals和hashCode方法

1. @Slf4j使用示例



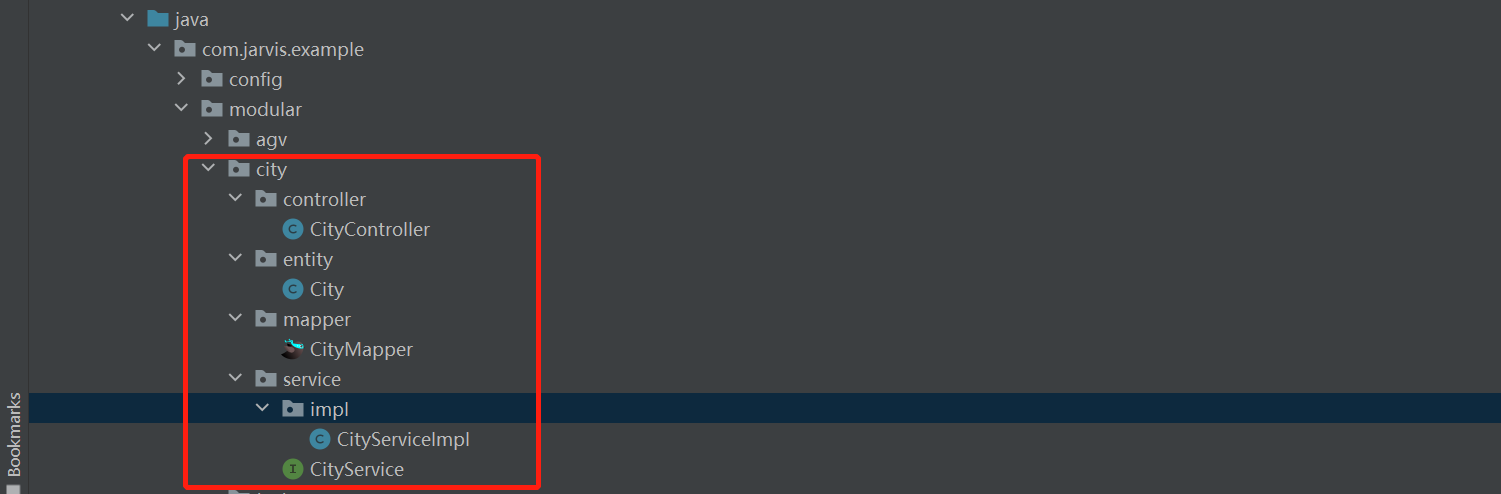
1. 在线学习参考地址：

<https://baijiahao.baidu.com/s?id=1663718812859963066&wfr=spider&for=pc>

## 功能开发

### 包结构

如下图所示：controller/service/mapper/entity



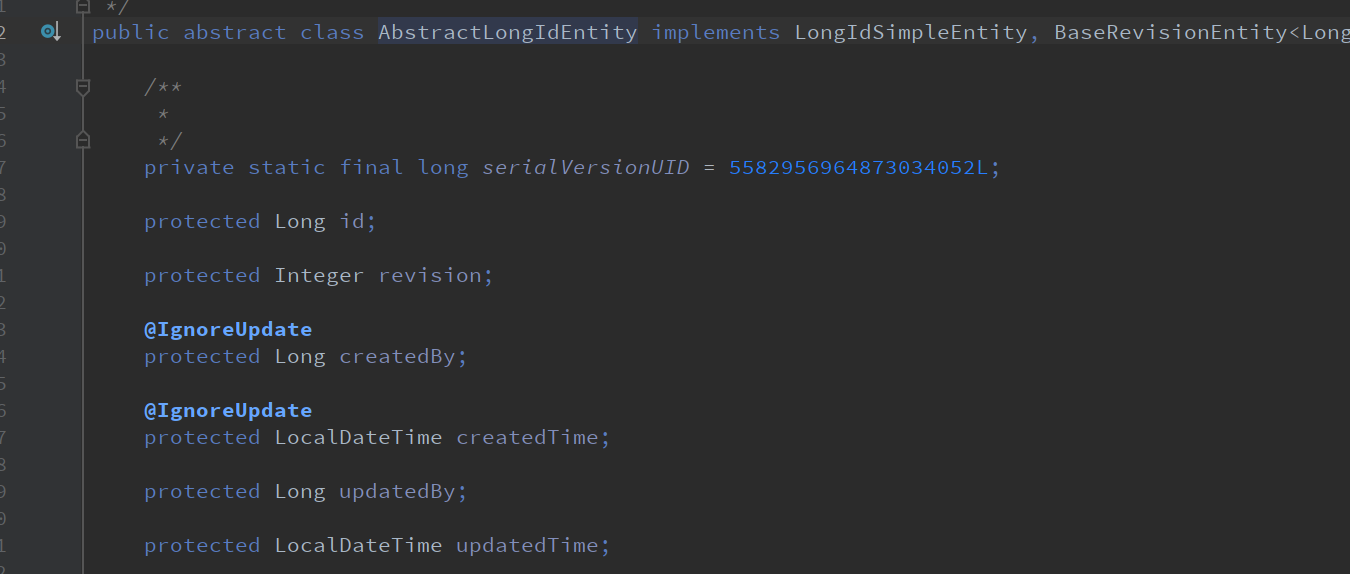
其中service中为面向接口编程，一个接口会对应一个实现类，实现类在impl包下，mapper包为DAO层

### 简单实体模块开发

#### 步骤一：创建简单实体

##### 要求

* 必须继承LongIdSimpleEntity或StringIdSimpleEntity或这两个类的子类（如：AbstractLongIdEntiy、AbstractSortLongIdEntiy），其中AbstractSortLongIdEntiy类多了一个sortOrder排序字段。



* 必须添加@Table注解，表明对应的物理表

##### 约定

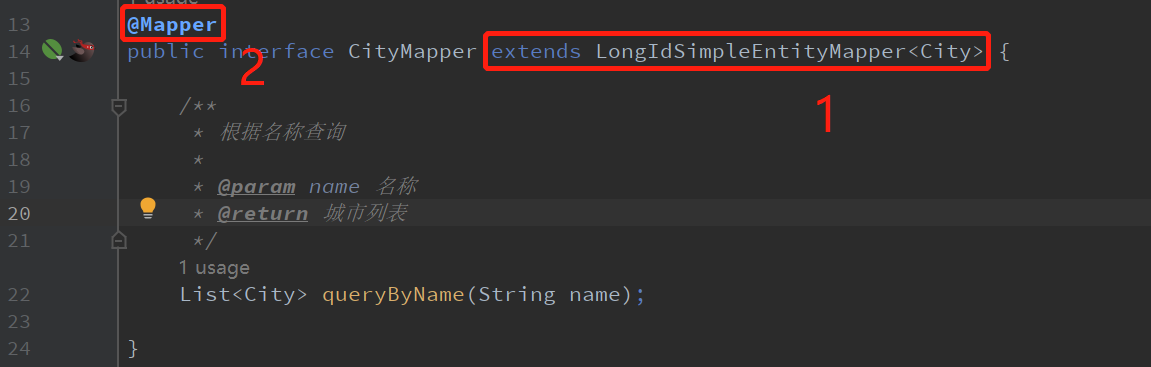
* 实体类的属性按小写驼峰格式转成对应字段名称，如实体属性showName，则对应的字段名称为show\_name
* 不需要持久化的属性用@Transient注解表明

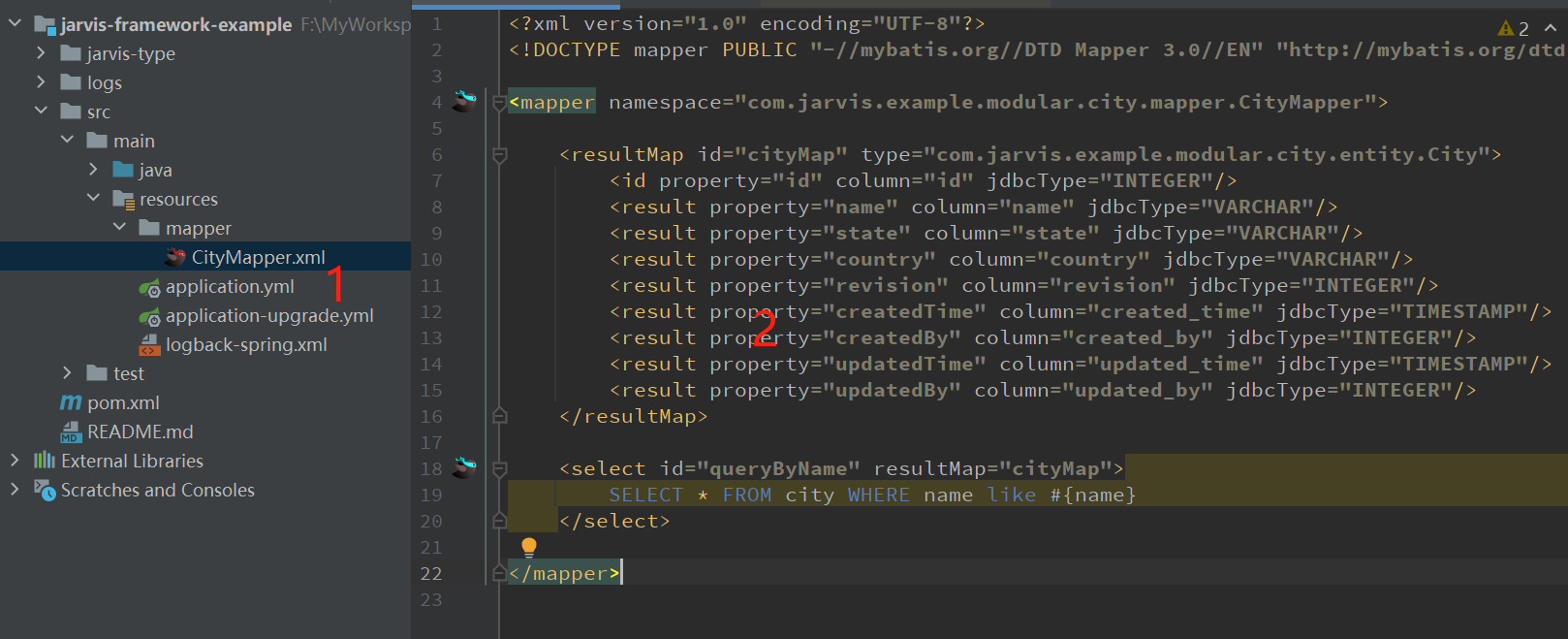


#### 步骤二：创建Dao类

##### 要求

* 继承LongIdSimpleEntityMapper或StringIdSimpleEntityMapper
* 添加@Mapper注解
* 如有复杂的操作要在xml自定义SQL，则在resources/mapper下创建对应的mapper的xml文件

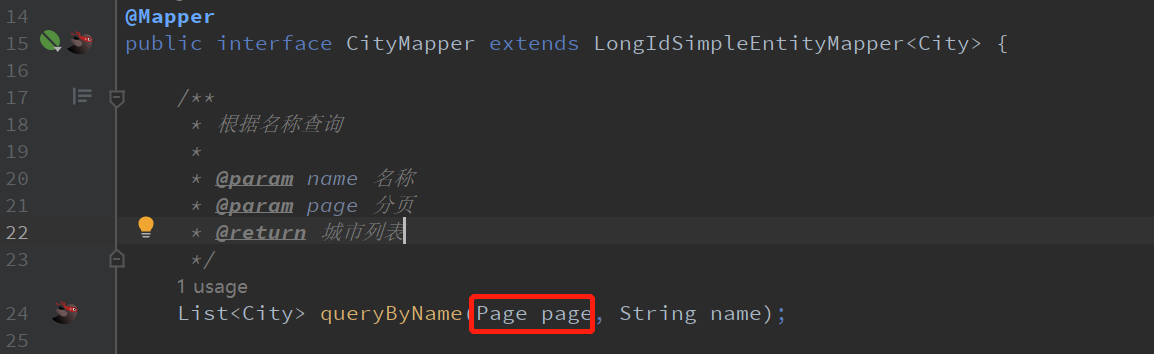




##### 方法介绍

1. insert：新增单个对象
2. insertAll：批量新增
3. update：更新单个对象
4. updateAll：批量更新
5. updateBy：根据条件更新
6. delete：删除单个对象
7. deleteById：根据ID删除
8. deleteAll：批量删除对象
9. deleteBy：根据条件删除对象
10. getById：根据ID获取单个对象
11. getBy：根据ID获取单个对象
12. queryBy：根据条件获取对象集合
13. count：根据条件统计
14. page：根据条件分页查询
15. exists: 根据条件判断是否存在

注：自定义分页查询，只要接口方法里添加一个Page参数可以完成分页功能（如果只要分页不需要统计总数，则把counted设置false，如：page.setCounted(false)），如下图所示：



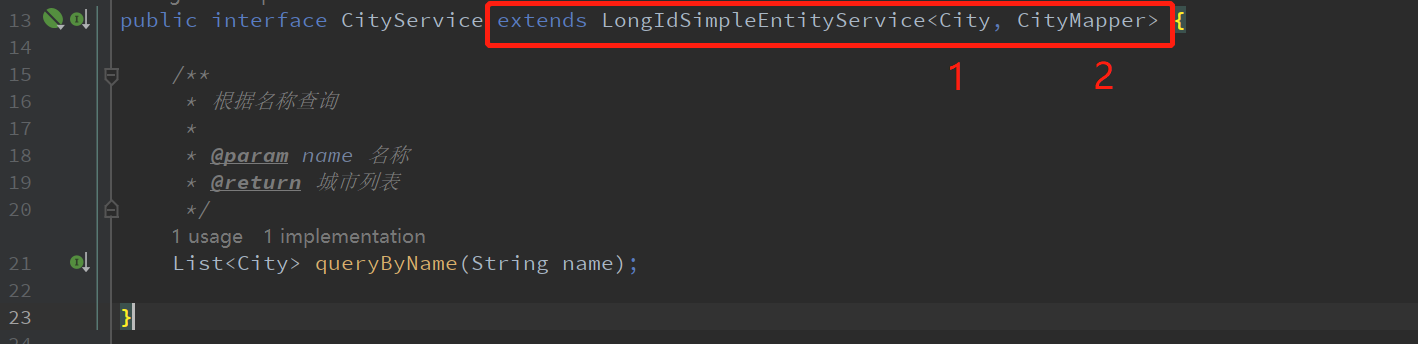
##### 在线帮助文档

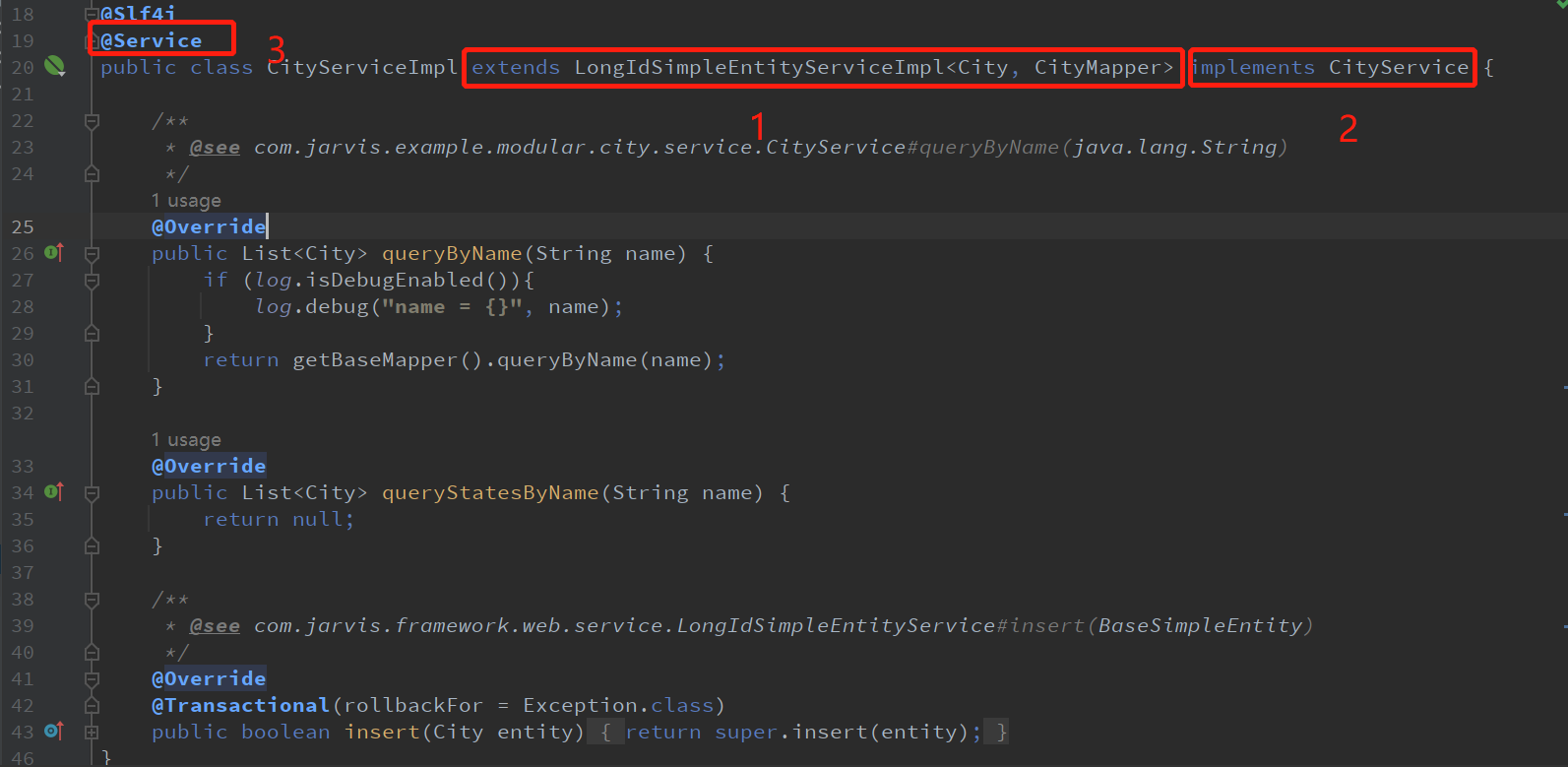
Mybatis在线帮助文档：<https://mybatis.org/mybatis-3/zh/index.html>

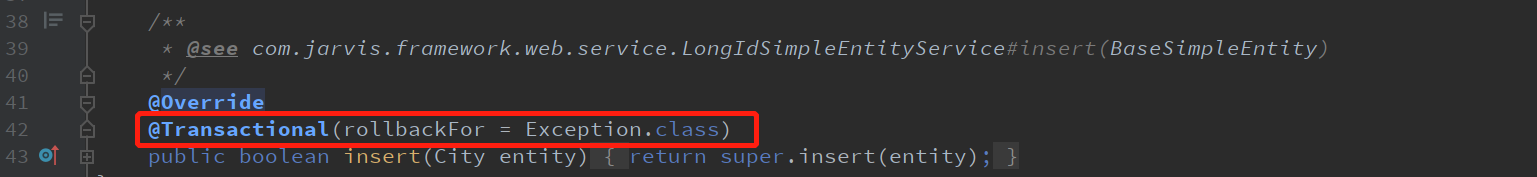
#### 步骤三：创建service类

##### 要求

* 接口继承LongIdSimpleEntityService或StringIdSimpleEntityService
* 实现类继承LongIdSimpleEntityServiceImpl或StringIdSimpleEntityServiceImpl，同时继承Service接口
* 实现类上添加@Component或@Service注解
* 实现类中如有数据库的增删改操作，必须添加@Transactional注解，同时指定rollbackFor=Exception.class
* 实现类如果需要Logger日志记录，添加@Slf4j注解







##### 常用方法

1. getBaseMapper：获取DAO
2. insert：新增单个对象，注：如果对象id有值，插入时就不自动生成id
3. insertAll：批量新增
4. update：更新单个对象
5. updateAll：批量更新
6. updateBy：根据条件更新
7. delete：删除单个对象
8. deleteById：根据ID删除
9. deleteAll：批量删除对象
10. deleteBy：根据条件删除对象
11. getById：根据ID获取单个对象
12. getBy：根据ID获取单个对象
13. queryBy：根据条件获取对象集合
14. count：根据条件统计
15. page：根据条件分页查询
16. exists: 根据条件判断是否存在

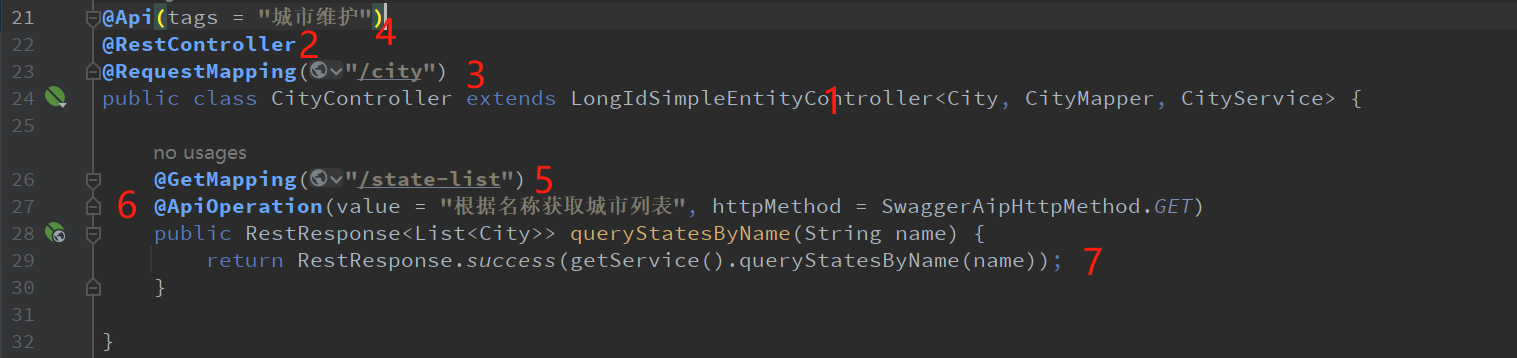
##### 建议

如果查询单个对象，则方法名使用getXxxx；如果查询集合对象，则方法名使用queryXxxx

#### 步骤四：创建controller类

##### 要求

* 继承LongIdSimpleEntityController或StringIdSimpleEntityController
* 添加@RestController注解
* 添加@RequestMapping注解，表明当前Controller访问地址
* 添加@Api注解，说明模块名称
* 前端入口的方法上添加@ApiOperation注解说明方法功能作用，同时加@GetMapping/@PostMapping/@PutMapping/@DeleteMapping表明前端访问入口
* 方法返回值统一返回RestResponse对象
* 注：controler里的所url统一使用kebab-case 风格（小写字符，单词与单词之间用中线分隔）



##### Restfull Api规范

* 按照restful接口设计规范

GET （SELECT）：从服务器检索特定资源，或资源列表。

POST （CREATE）：在服务器上创建一个新的资源。

PUT （UPDATE）：更新服务器上的资源，提供整个资源。

PATCH （UPDATE）：更新服务器上的资源，仅提供更改的属性。

DELETE （DELETE）：从服务器删除资源。

* 接口尽量使用名词，禁止使用动词，下面是一些例子
* GET /user/count：统计用户总数

GET /user/page：按分页列出用户列表

POST /user：新建一个用户

GET /user/{id}：获取某个指定用户的信息

PUT /user/{id}：更新某个指定用户的信息（提供该用户的全部信息）

PATCH /user/{id}：更新某个指定用户的信息（提供该用户的部分信息）

DELETE /user/{id}：删除某个用户

GET /user/{id}/role：列出某个指定用户的所有角色

DELETE /user/{uId}/role/{rId}：删除某个指定用户的指定角色

##### 功能介绍

通过以上4步就实现了一个模块新增、修改、删除、列表查询基础功能，访问地址：

新增：<http://localhost:8080/city> POST请求

修改：<http://localhost:8080/city> PUT请求

根据ID获取数据：<http://localhost:8080/city/1> GET请求

根据ID删除数据：[http://localhost:8080/cit](http://localhost:8080/city)y/1 DELETE请求

列表数据查询：[http://localhost:8080/cit](http://localhost:8080/city)y/page?pageNumber=1&pageSize=10 GET请求

列表总数查询：[http://localhost:8080/cit](http://localhost:8080/city)y/count GET请求

##### 常用方法

1. getService() ：获取对应的Service
2. getParameter(String name)：获取参数值

### 动态实体模块开发

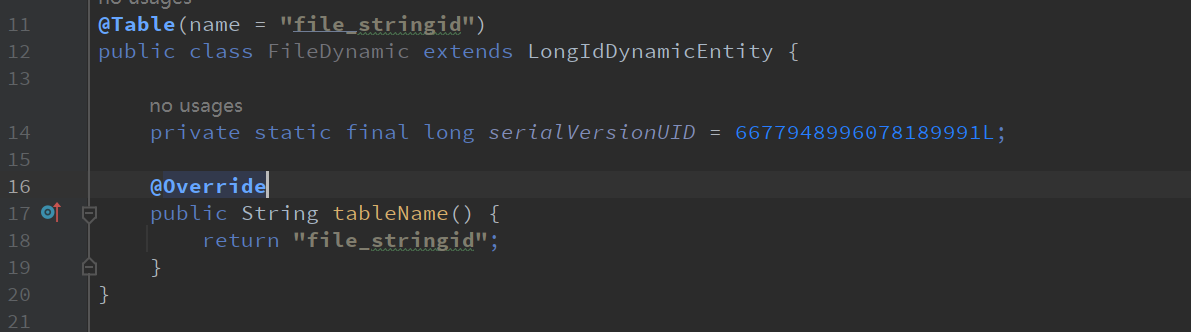
#### 步骤一：创建动态实体

##### 要求

* 必须继承LongIdDynamicEntity或StringIdDynamicEntity或这两个类的子类
* 通过tableName(String tableName)方法动态指定对应的物理表
* 如果表名是固定的，只有字段是动态的，可以通过@Table注解来指定物理表，复写tableName()方法
* 如果是动态表名，则不需要添加@Table注解

##### 约定

* 动态表实体没有固定属性，所有实体中没有属性，在使用时通过put(column, value)来动态添加
* 动态属性统一使用小写驼峰格式，如entity.put(“yearCode”, 2021)
* 不需要持久化的属性不要put进去，在使用时手动remove(column)掉

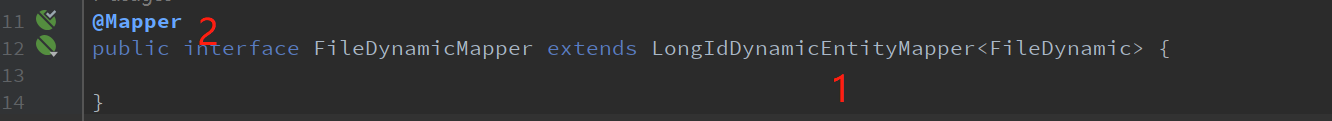


#### 步骤二：创建Dao类

##### 要求

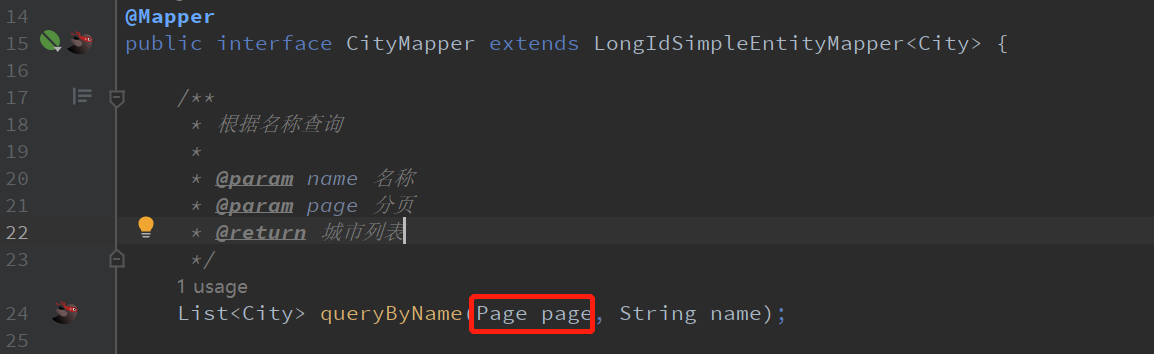
* 继承LongIdDynamicEntityMapper或StringIdDynamicEntityMapper
* 添加@Mapper注解

如有复杂的操作要在xml自定义SQL，则在resources/mapper下创建对应的mapper的xml文件



##### 方法介绍

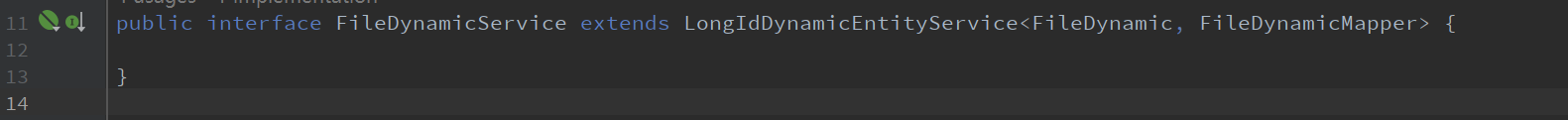
1. insert：新增单个对象
2. insertAll：批量新增
3. update：更新单个对象
4. updateAll：批量更新
5. updateBy：根据条件更新
6. delete：删除单个对象
7. deleteById：根据ID删除
8. deleteAll：批量删除对象
9. deleteBy：根据条件删除对象
10. getById：根据ID获取单个对象
11. getBy：根据ID获取单个对象
12. queryBy：根据条件获取对象集合
13. count：根据条件统计
14. page：根据条件分页查询
15. exists: 根据条件判断是否存在

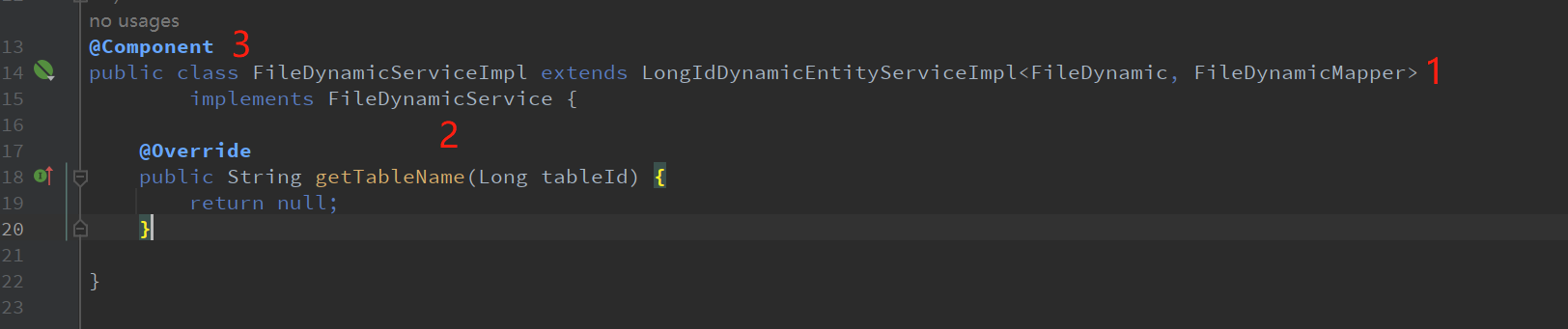
自定义分页查询，只要接口方法里添加一个Page参数可以完成分页功能（如果只要分页不需要统计总数，则把counted设置false，如：page.setCounted(false)），如下图所示：

#### 步骤三：创建service类

##### 要求

* 接口继承LongIdDynamicEntityService或StringIdDynamicEntityService
* 实现类继承LongIdDynamicEntityServiceImpl或StringIdDynamicEntityServiceImpl，同时继承Service接口
* 实现类上添加@Component或@Service注解
* 实现类中如有数据库的增删改操作，必须添加@Transactional注解，并指明rollbackFor = Exception.class
* 实现类如果需要Logger日志记录，添加@Slf4j注解





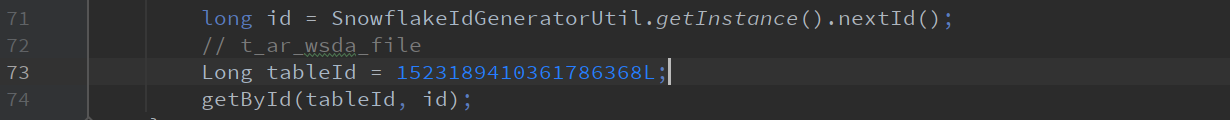
##### 常用方法

1. getBaseMapper：获取DAO
2. insert：新增单个对象
3. insertAll：批量新增
4. update：更新单个对象
5. updateAll：批量更新
6. updateBy：根据条件更新
7. delete：删除单个对象
8. deleteById：根据ID删除
9. deleteAll：批量删除对象
10. deleteBy：根据条件删除对象
11. getById：根据ID获取单个对象
12. getBy：根据ID获取单个对象
13. queryBy：根据条件获取对象集合
14. count：根据条件统计
15. page：根据条件分页查询
16. exists: 根据条件判断是否有数据

##### 注意事项

如果是动态表，接口中的tableId参数是必须设置的，同时columns方法参数，统一使用小写字段名称（如：fonds\_code），不能是属性（如：fondsCode）

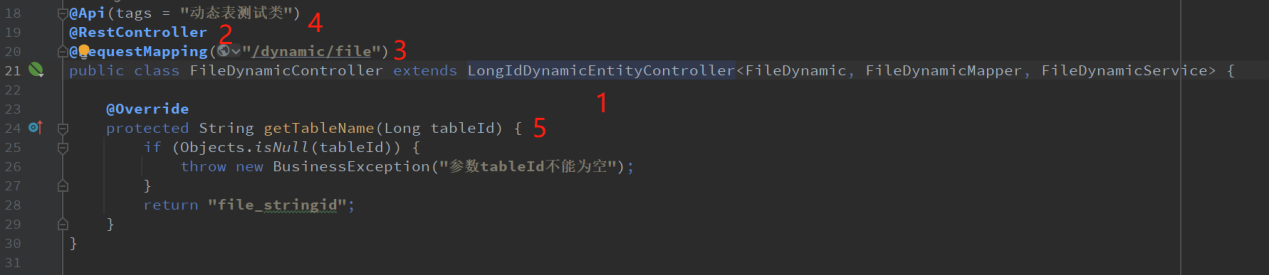




#### 步骤四：创建controller类

##### 要求

* 继承LongIdDynamicEntityController或StringIdDynamicEntityController
* 添加@RestController注解
* 添加@RequestMapping注解，表明当前Controller访问地址
* 添加@Api注解，说明模块名称
* 前端入口的方法上添加@ApiOperation注解说明方法功能作用，同时加@GetMapping/@PostMapping/@PutMapping/@DeleteMapping表明前端访问入口
* 方法返回值统一返回RestResponse对象
* 如果是动态表，则必须实现getTableName接口



##### 功能介绍

通过以上4步就实现了一个模块新增、修改、删除、列表查询基础功能，访问地址：

新增：<http://localhost:8080/city> POST请求

修改：<http://localhost:8080/city> PUT请求

根据ID获取数据：<http://localhost:8080/city/1> GET请求

根据ID删除数据：[http://localhost:8080/cit](http://localhost:8080/city)y/1 DELETE请求

列表数据查询：[http://localhost:8080/cit](http://localhost:8080/city)y/page?pageNumber=1&pageSize=10 GET请求

列表总数查询：[http://localhost:8080/cit](http://localhost:8080/city)y/count GET请求

##### 注意事项

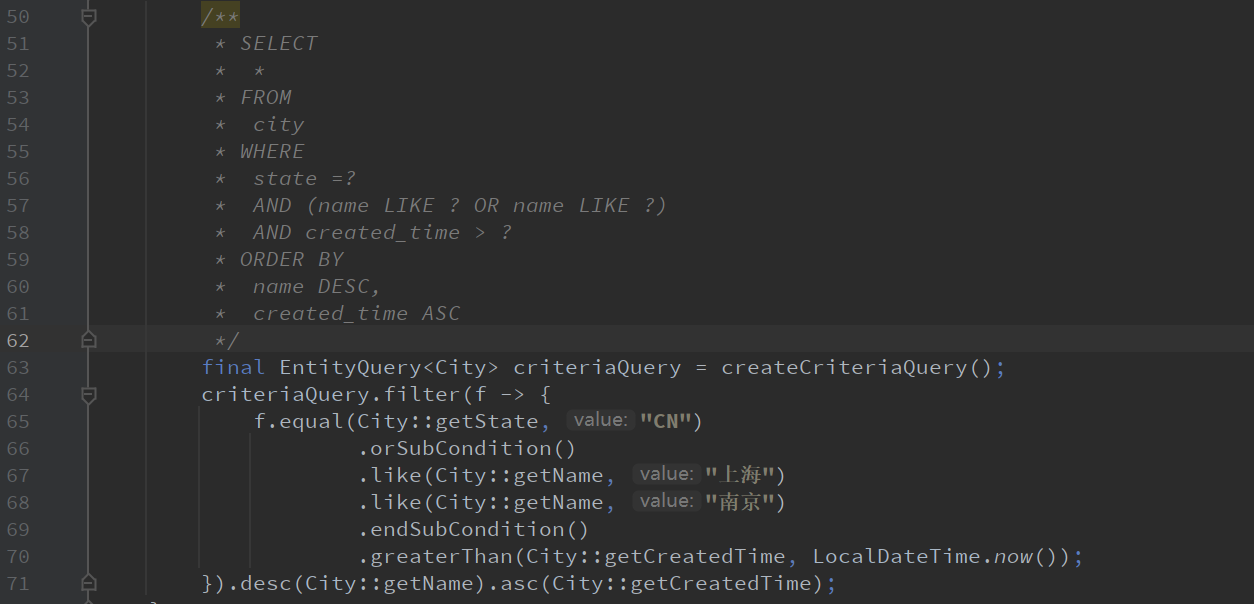
所有接口都有一个tableId参数，如果是动态表，则tableId必须传值

### 动态查询功能

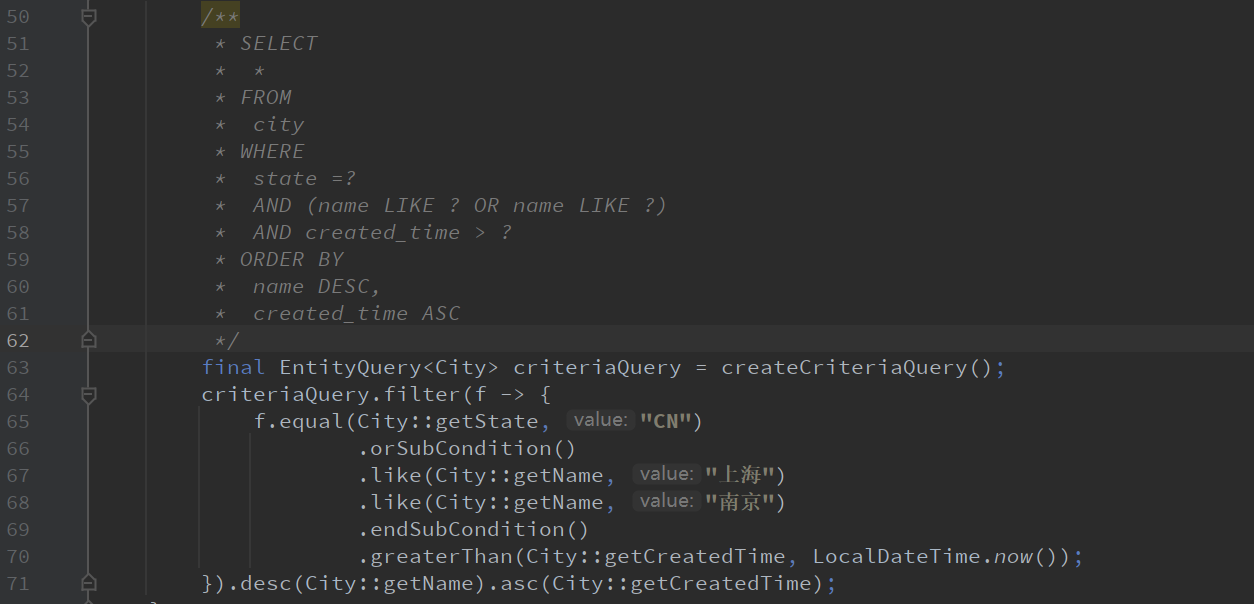
#### CriteriaQuery类用法

注：新版本中已弃用***~~ColumnFunction~~***,改用***Getter***或直接用***EntiyQuery***

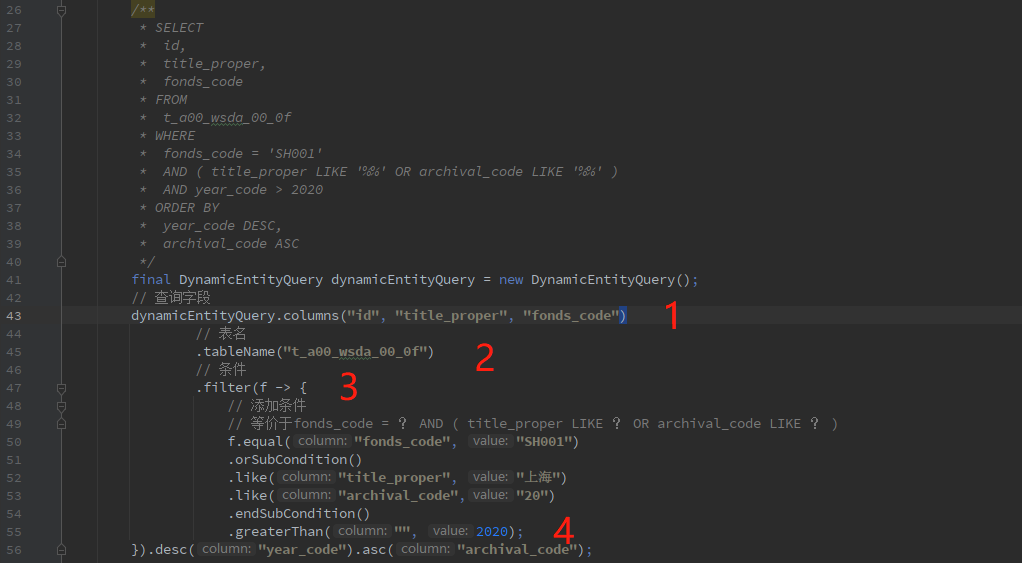
1. 简单实体普通用法



1. 简单实体链式用法



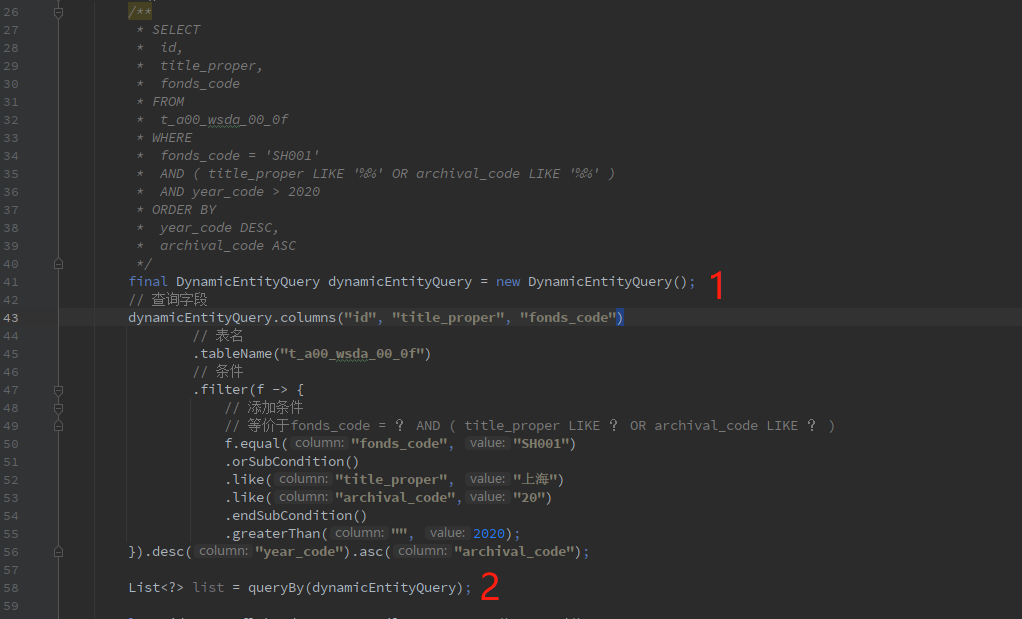
1. 动态表实体普通用法（注：columns参数必须是字段名称）



1. 动态表实链式用法（注：columns参数必须是字段名称）



1. Service中提供了查询方法使用CriteriaQuery
2. getBy
3. queryBy



注：在Service中可以通过createCriteriaQuery()来创建查询对象

### 分页功能（page接口使用介绍）



#### 分页参数

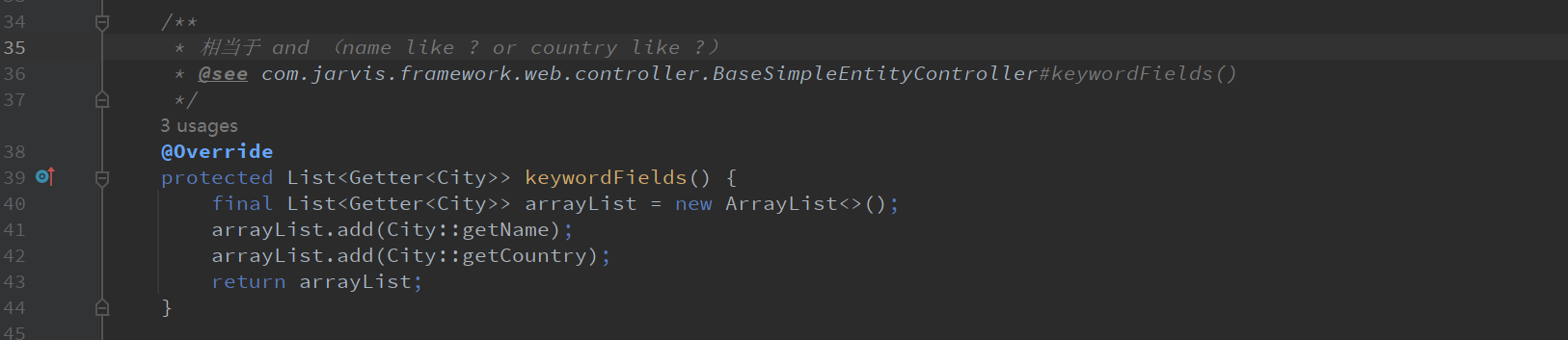
示例url: <http://localhost:8080/city/page?pageNumber=1&pageSize=10>

1. pageNumber：当前页数参数（从1开始）
2. pageSize：每页条数参数
3. keyword：关键字
4. Q\_[EQ|GT|LIKE|LT]\_属性名：动态传条件，可以有多个，如：Q\_EQ\_status=80

#### 一体化查询参数

示例url: <http://localhost:8080/city/page?pageNumber=1&pageSize=10&keyword=>中国

1. 参数名称：keyword
2. 在controller中要复写keywordFields接口，指定哪些字段要作为模糊查询



#### 条件查询参数

示例url: <http://localhost:8080/city/page?pageNumber=1&pageSize=10&Q_EQ_country=>中国&Q\_EQ\_name=上海&Q\_BT\_createdTime=2020-01-01 12:12:12,2021-01-01 23:59:59

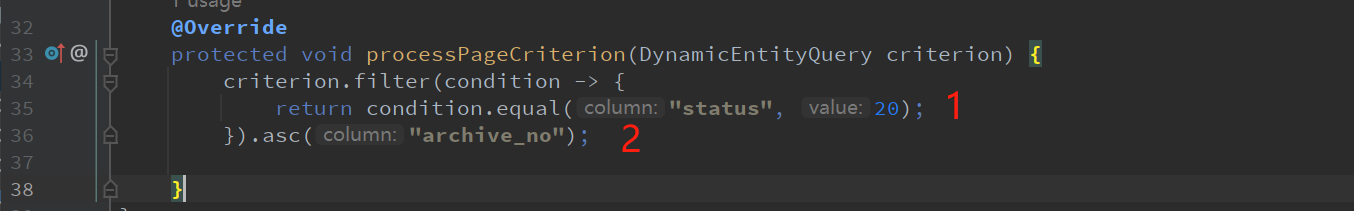
相当于查询条件： country=? and name=? and created\_time between ? and ?

1. 参数名称格式：Q\_xx\_field，其中field表示字段对应的属性名称，xx值如下表格所示

|  |  |  |  |
| --- | --- | --- | --- |
| xx值 | 说明 | xx值 | 说明 |
| EQ | 等价于SQL中的“=” | NEQ | 等价于SQL中的“!=” |
| LIKE | 等价于SQL中的“like ‘%xxx%’” | BT | 等价于SQL中的“between ? and ?” |
| GT | 等价于SQL中的“>” | GTE | 等价于SQL中的“>=” |
| LT | 等价于SQL中的“<” | LTE | 等价于SQL中的“<=” |
| NULL | 等价于SQL中的“is null” | NNULL | 等价于SQL中的“is not null” |
| SW | 等价于SQL中的“like ‘xxx%’” | EW | 等价于SQL中的“like ‘%xxx’” |
| IN | 等价于SQL中的“in(?,…)” | NIN | 等价于SQL中的“not in(?,…)” |

#### 查询的固定条件和排序设置

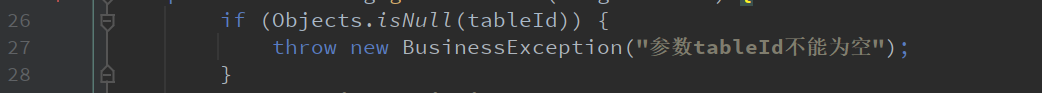
在controller中通过复写processPageCriterion方法来实现



### 异常使用

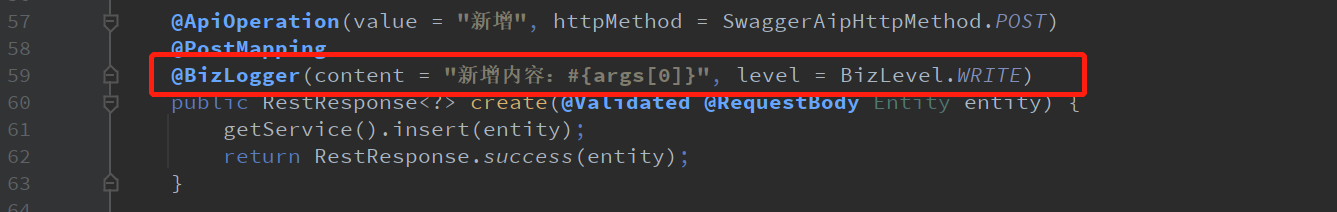
业务处理或逻辑校验异常统一使用BusinessException，并给出相应的提示，如：

throw new BusinessException(“题名不能为空”);



## 操作日志

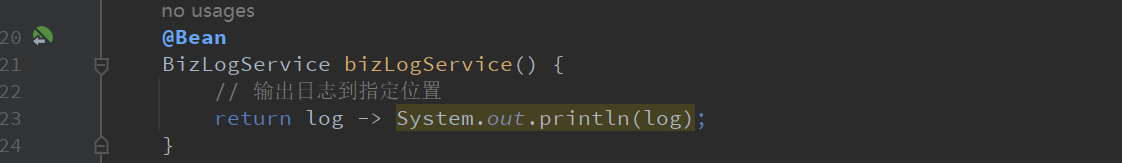
使用注解@BizLogger(module=”模块”, action=”操作”, content=”内容”, level=BizLevel.WRITE)，如下图所示：



* module：如果类上加上@Api注解，就可以不写，会自动使用@Api注解上的模块名称
* action: 如果方法上加上@ApiOperation注解，就可以不写，会自动使用@ApiOperation注解上的操作名称
* content: 日志内容，可以配合EL表达式，如content=”新增用户：#{args[0].name}”，其中args[0]为方法第一个参数，也可以使用#{result.body.xxx}，其中result表示方法返回值对象；如果要动态添加自定义日志内容，可以在方法里使用BizLogContentHolder.putContent(“日志内容”)，这样就可以不在注解里写content
* level：日志级别有OFF>WRITE>QUERY>READ>ALL
* 配置操作日志记录级别，如下图所示，默认只记WRITE级别的日志



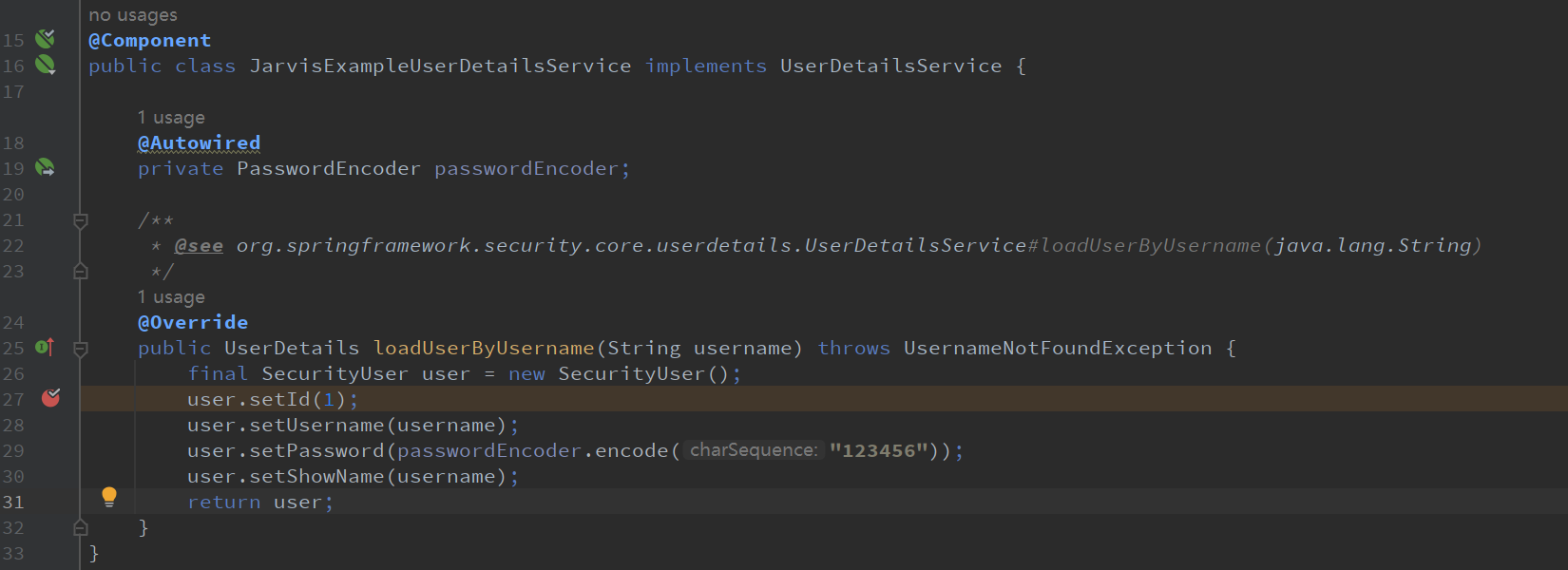
* BizLogService：表示日志存在什么地方，在系统实现这个接口并作为Spring Bean就可以了记录日志了



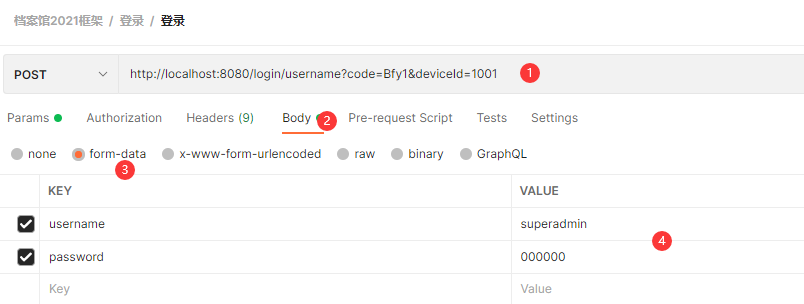
## 系统登录

系统默认登录用户密码是：superadmin/000000

可以通过实现UserDetailsService接口来对接应用系统具体用户登录逻辑



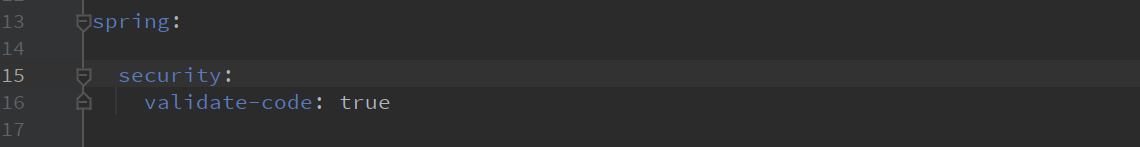
登录接口：



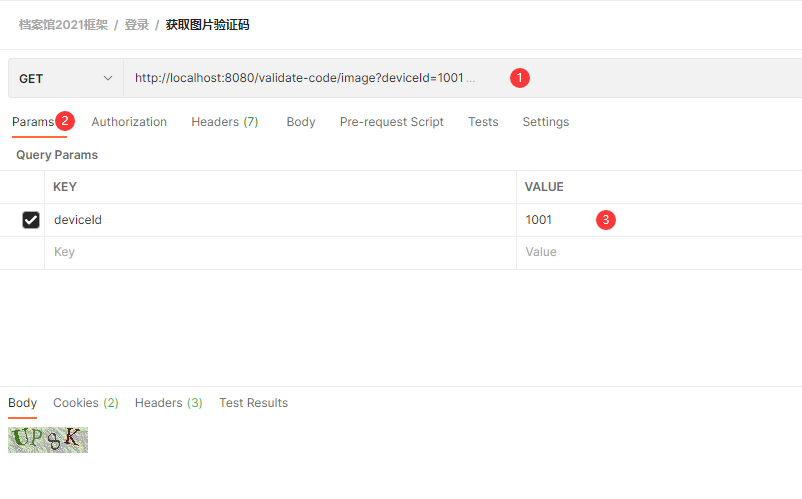
访问地址：<http://localhost:8080/login/username>

## 登录验证码

通过配置文件启用验证码，默认是关闭



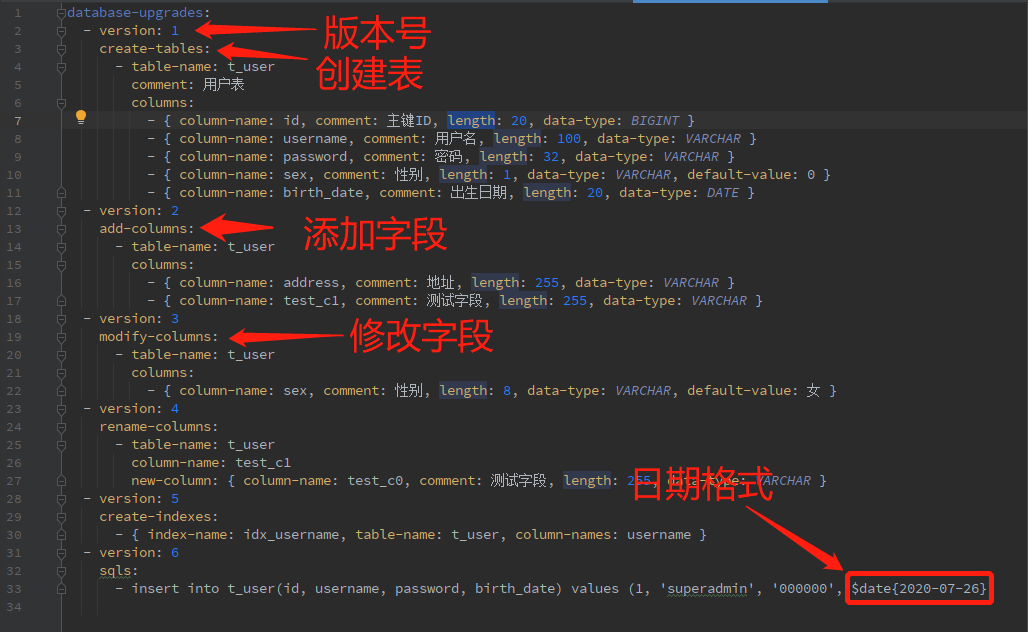
获取验证码接口，其中deviceId为随机值如uuid，只要保证获取验证码与登录时保持一致即可



访问地址：<http://localhost:8080/validate-code/image>

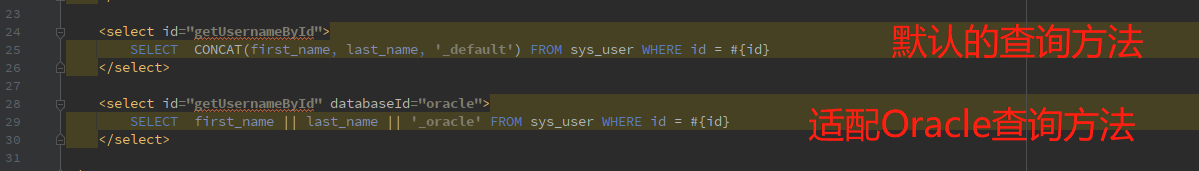
## 数据库升级脚本

要完成项目数据库初始化、客户数据库升级，只要一份升级脚本即可，在application-upgrade.yml配置升级脚本，如下图所示：



## 数据库适配

在mapper.xml文件中，通过***databaseId***来区分数据库，如果不加databaseId表示默认的，如下图所示：



databaseId值：oracle/mysql/sqlserver/dm/kingbase/kingbase8/oscar/h2等；

databaseId来源于数据库url配置：jdbc:xxxx://ip:port中的xxxx作为databaseId（全小写），示例如下：

datasource.url=jdbc:oracle://127.0.0.1:1521:orcl，则**databaseId=oracle**

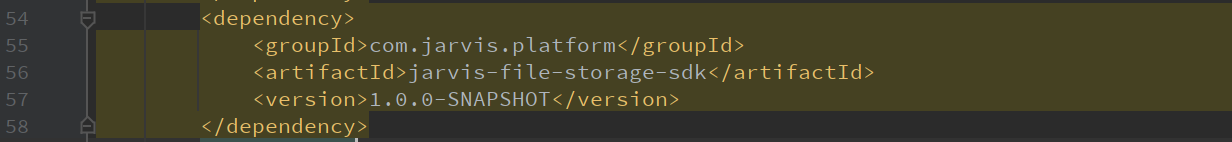
datasource.url=jdbc:oscar://127.0.0.1:2003/OSRDB，则**databaseId=oscar**

在Java类里，可以通过DatabaseIdHolder.getDatabaseId()工具类来获取databaseId值，同时这个类中预置一些常用的数据库判断，如isOracle(), isDm()等

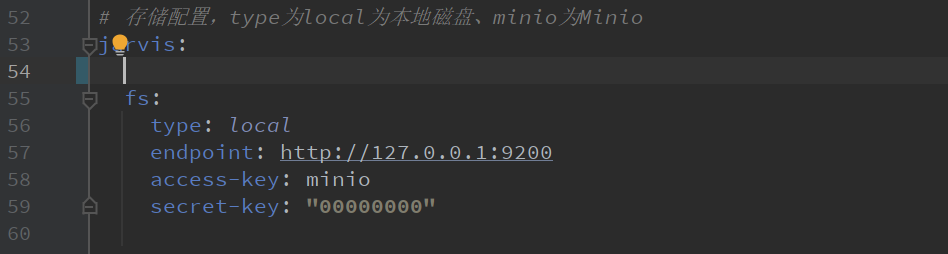
## 文件存储

Minio学习参考地址：http://docs.minio.org.cn/minio/baremetal/

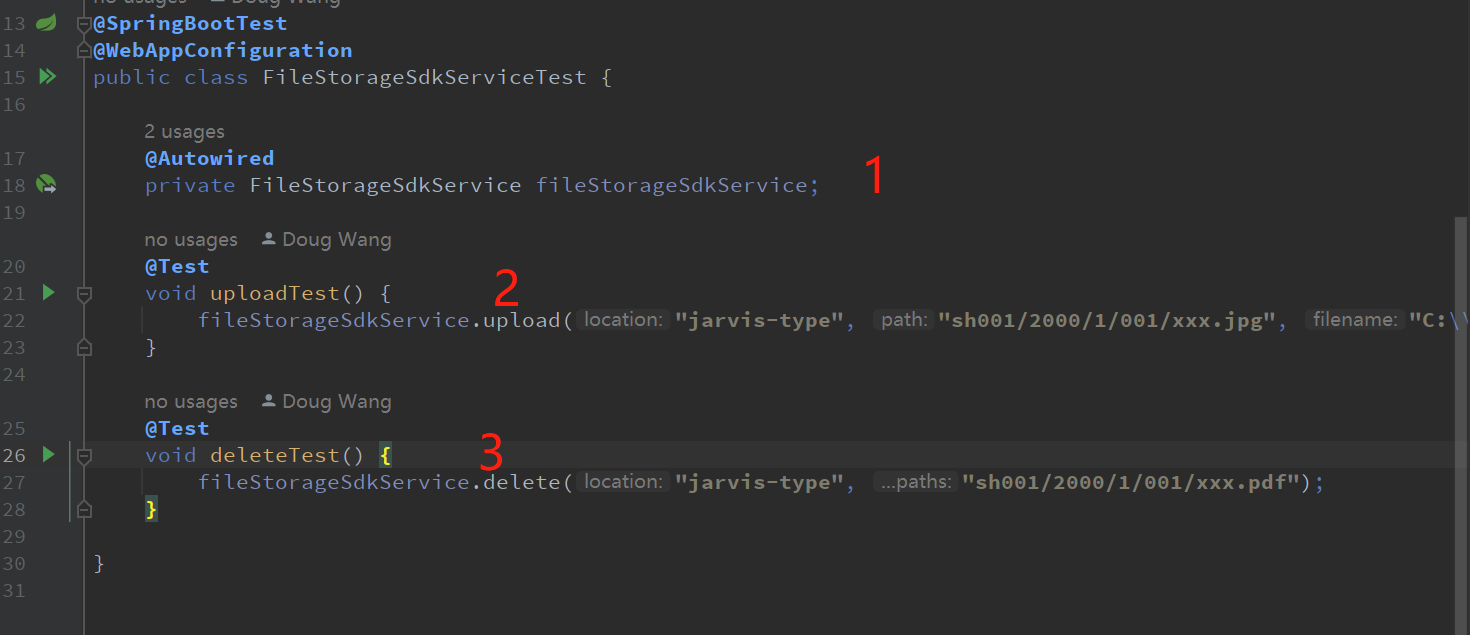
在pom.xml中引入如下jar



配置application.yml参数，type值：LOCAL（本地磁盘）/MINIO/ALI（阿里OSS）/HUAWEI（华为OSS）/FDFS（FastDFS）/TENCENT(腾讯OSS)等。



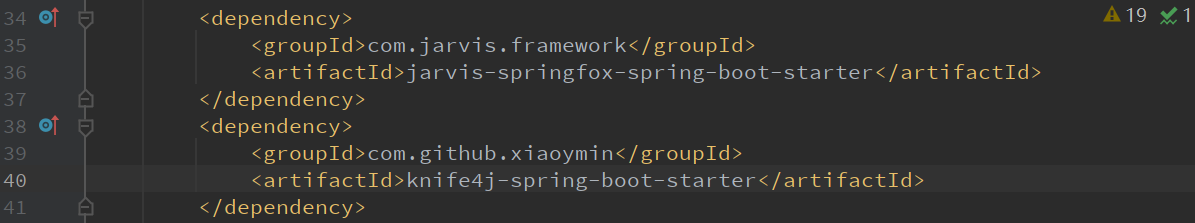
注入FileStorageSdkService就可以开始文件存储开发。



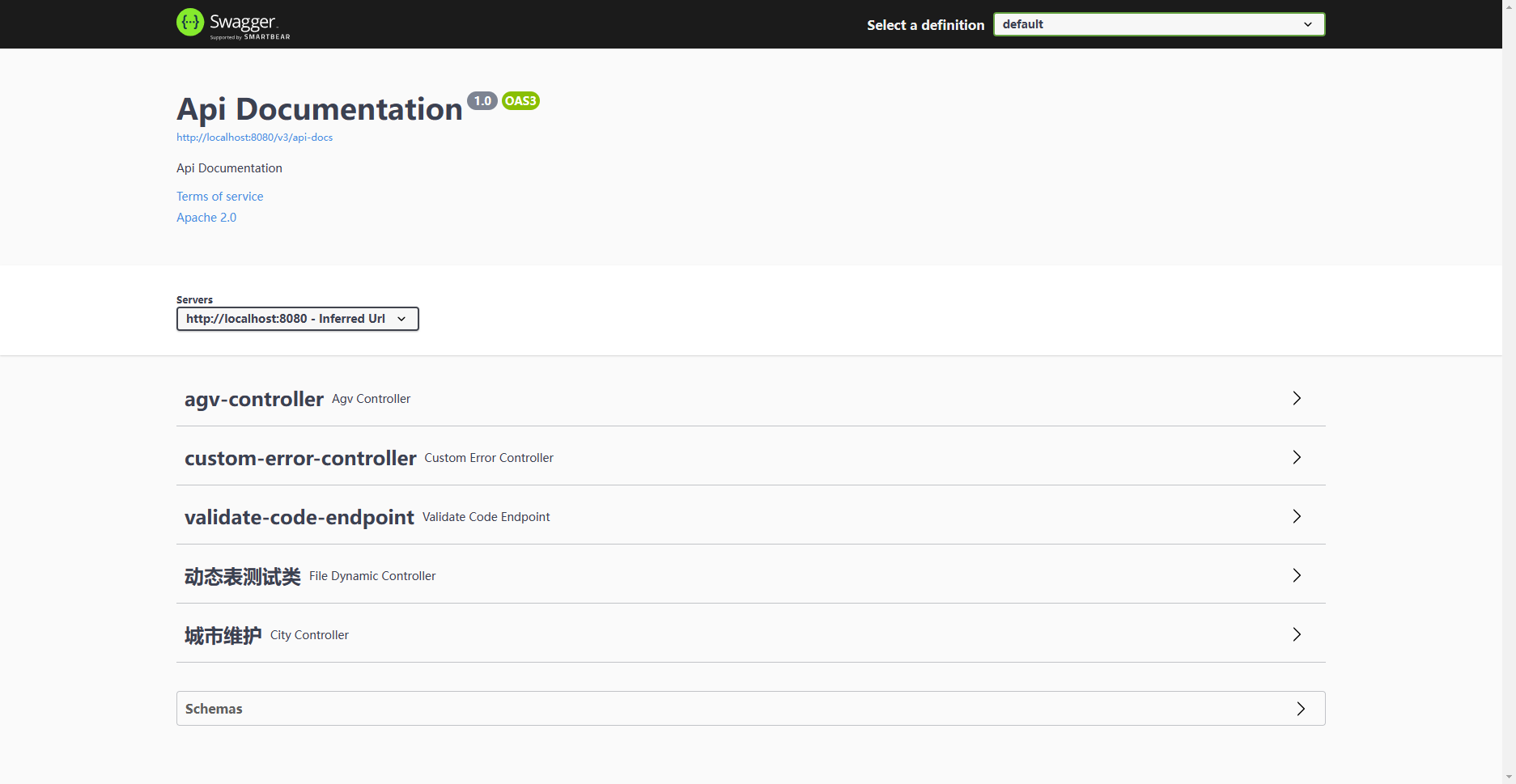
## Swagger文档

Swagger注解学习参考地址：<https://www.jianshu.com/p/d3bf446a2840>

在pom.xml中引入如下jar



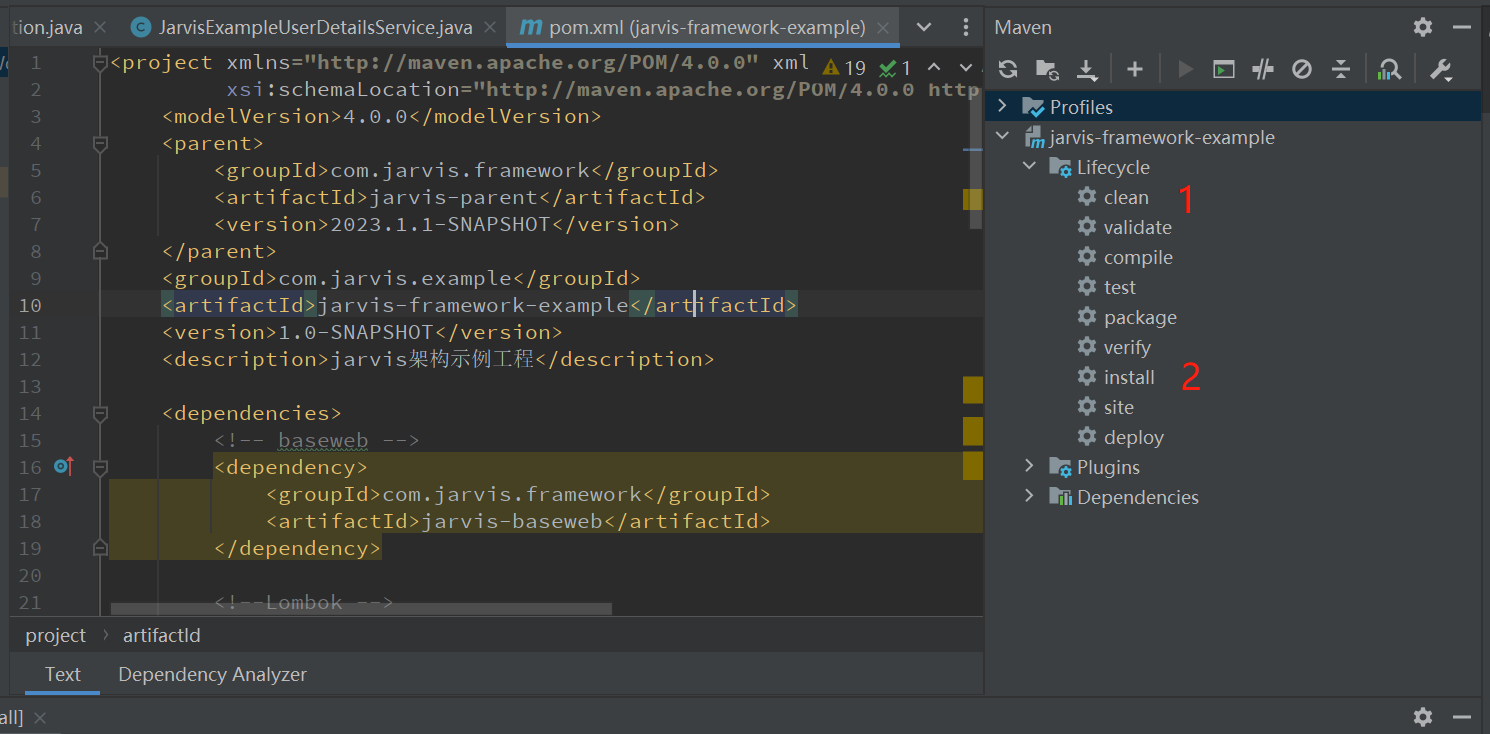
在我们系统中访问地址示例：[http://ip:port/swagger-ui/#/](http://ip:port/doc.html)打开后，会看到如下图所示页面



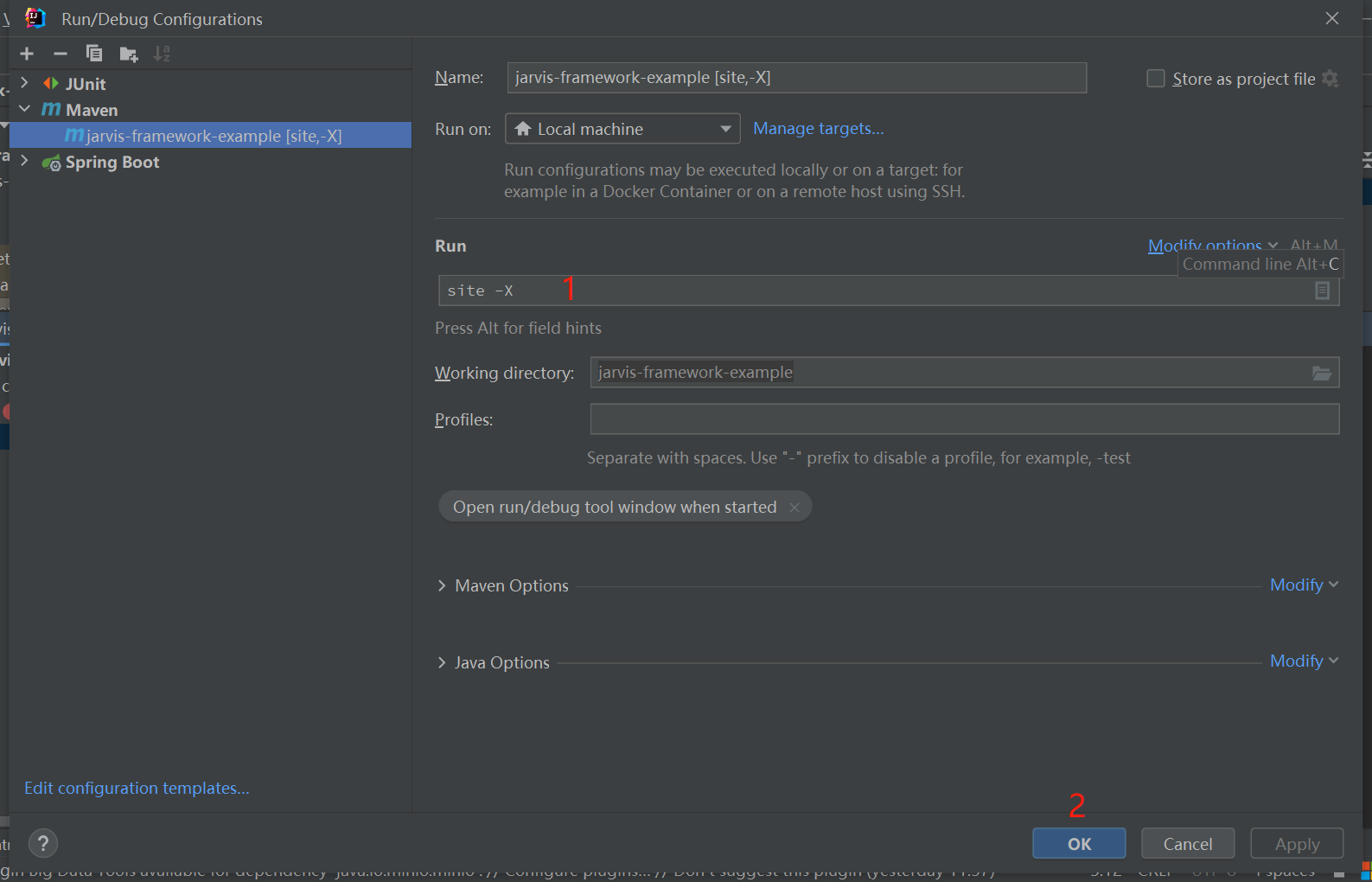
具体使用见“功能开发”->“简单实体模块开发”->“步骤四：创建controller类”，其中@Api和@ApiOperation两个注解就是Swagger文档使用的

## 代码检查

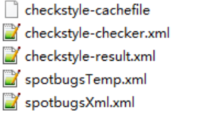
第一步：在主工程上右键，先Maven clean，再Maven install，如下图所示



第二步：运行 Maven build... ，出现如下图所示配置框，在Run框里输入：site -X，点击“Debug”或”Run”



第三步：运行完成后，到每个子工程的target目录下查看，如下图所示，表示代码检查完成



第四步：进site目录，打开index.html

第五步：index.html页面如下图所示，点击左侧”Project Reports”，右边列表有”Checkstyle”和“SpotBugs”

第六步：查看Checkstyle，如下图所示：

点击具体java文件，可以查看每个java类有哪些问题，如下图所示：

第七步：查看SpotBug，步骤与Checkstyle类似，如下图所示

## 示例工程下载

ssh://git@github.com:jarvisframework/jarvis-framework-example.git

## Postman工具使用

略

## 在线文档汇总

### Lombok

<https://baijiahao.baidu.com/s?id=1663718812859963066&wfr=spider&for=pc>

### Mybatis

<https://mybatis.org/mybatis-3/zh/index.html>

### Swagger

<https://www.jianshu.com/p/d3bf446a2840>

### Postman

<https://www.jianshu.com/p/76fdd0b4b256>

### Minio

http://docs.minio.org.cn/minio/baremetal/