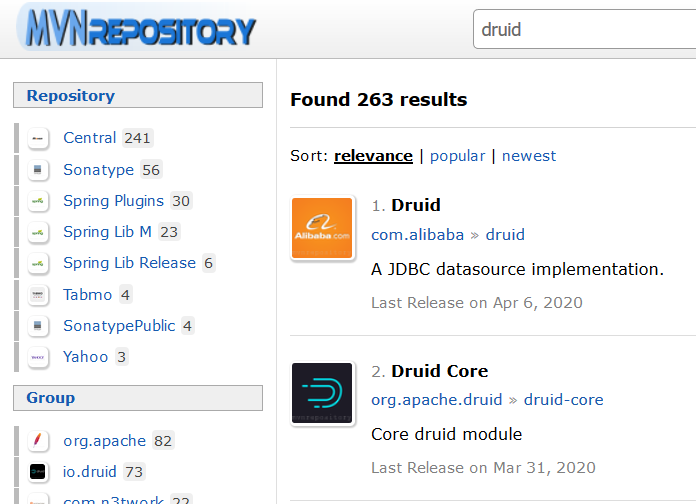
# Druid数据源

## 介绍

Druid是阿里巴巴开源的一个数据源，主要用于java数据库连接池，相比spring推荐的DBCP和hibernate推荐的C3P0、Proxool数据库连接池，Druid在市场上占有绝对的优势；

## 基本使用

1. 下载druid的jar包，如下：



将下载好的jar包放入到lib目录中；

1. 在配置数据源时使用durid数据源，如下：



配置完成后，即可使用；

## 数据监控

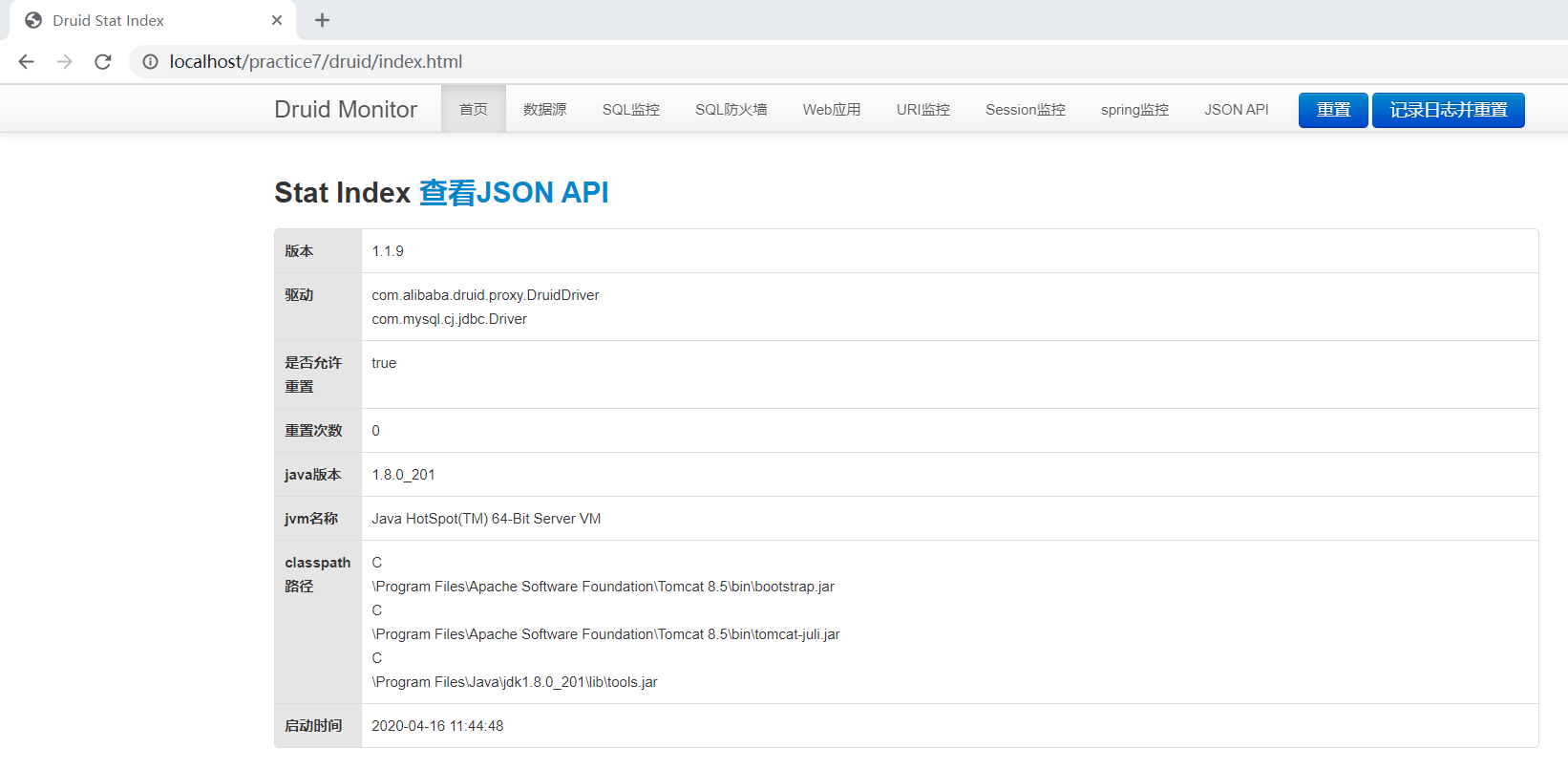
当项目中使用了阿里巴巴的druid数据源，如果想对该项目的数据源以及SQL情况进行查看，则可以配置数据监控。

1. 在web.xml文件中配置数据监控后台的访问路径，如下：



经过以上配置，数据监控后台的访问路径为<http://localhost/项目名/druid>

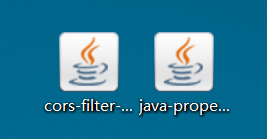
1. 启动项目，访问后台，如下：



# Ajax跨域处理

（关于Ajax跨域问题可参考<https://segmentfault.com/a/1190000012469713>）

1. 下载cors-filter.jar包和java-property-utils.jar包，放入lib目录中，如下：



需要注意的是版本搭配的问题，在这里的cors-filter.jar为2.4版本，java-property-utils.jar版本为1.9.1版本；

1. 在web.xml文件中配置跨域过滤器，将这个过滤器作为第一个，如下：



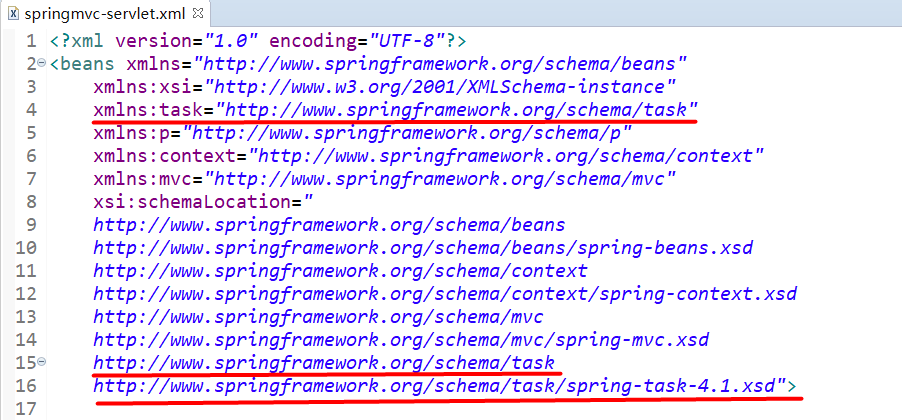


网上关于跨域的解决方案大致有四种，这是其中之一，如果有时间最好都应用测试一下；

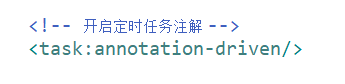
# 定时任务

## 开启定时支持

1. 在springmvc配置文件中引入定时任务约束



1. 在springmvc配置文件中开启定时任务注解，如下：

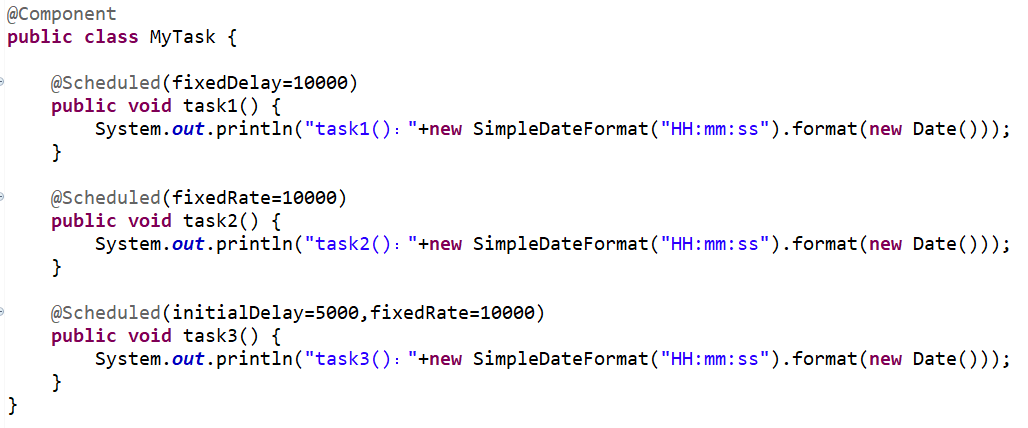


开启定时支持后，在项目就可以使用定时任务了。

## 简单使用

定时任务是使用@Scheduled注解标注的方法，这些承载的定时任务的方法所在的类必须使用@Component注解标注、这些类所在的包必须被扫描；

定义一个定时任务类，如下：



@Scheduled(fixedDelay=10000)表示该方法在项目启动后立即执行一次，之后每隔十秒执行一次；

@Scheduled(fixedRate=10000)表示该方法在项目启动后立即执行执行一次，之后每隔十秒立即执行一次；

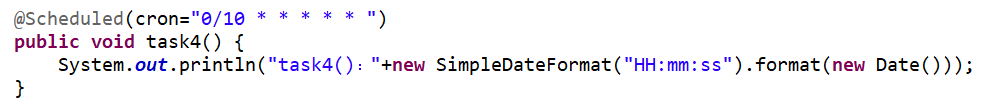
@Scheduled(initialDelay=5000,fixedRate=10000)表示该方法在项目启动后不立即执行，而是隔5秒后执行，之后每隔10秒执行一次；

**【注意：作为定时执行的方法，返回值只能为空-void】**

## corn表达式

通过@Scheduled的参数（如fixedDelay、fixedRate、initialDelay等）来设置该方法多久执行一次；但这些参数只能设置任务每隔多久重复执行，而不能设置成在具体的时间点执行；为此，可以使用corn表达式来完成，同时corn表达式还具有@Scheduled注解全部参数的功能，也能设置任务每隔多久重复执行。

其使用方法如下：



表示每隔10秒执行一次该方法；

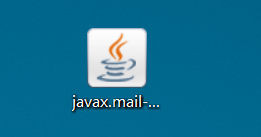
关于corn表达式的用法可以参考如下文章：

<https://blog.csdn.net/NRlovestudy/article/details/80915757>

# 邮件服务

## 基本配置

1. 添加java.mail.jar包，如下：



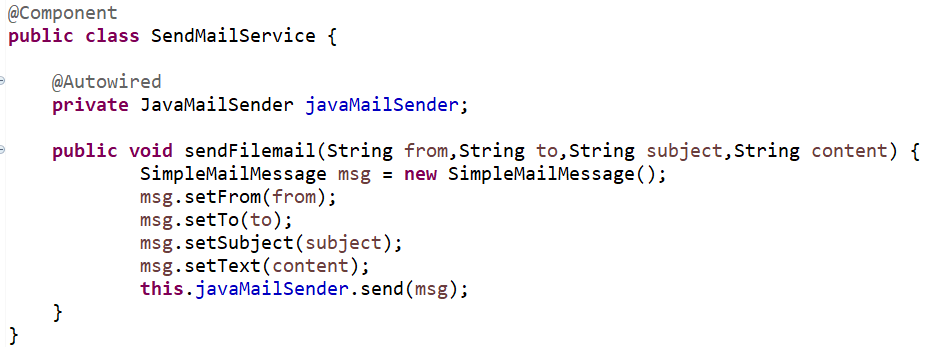
1. 在applicationContext.xml中配置邮件发送需要的信息，如下：



## 使用

### 发送简单邮件

1. 创建一个类用于发送邮件（记得扫描该包），如下：



from表示发送人；to表示收件人；subject表示邮件主题；content表示内容，最后使用send方法发送邮件；

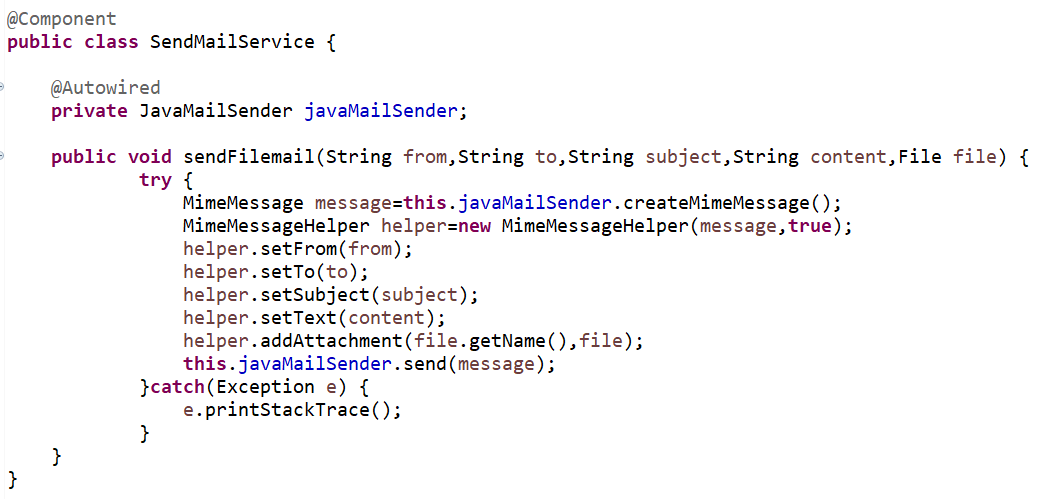
1. 进行测试，如下：



启动项目，访问该路径，查看收件人邮箱是否收到邮件。

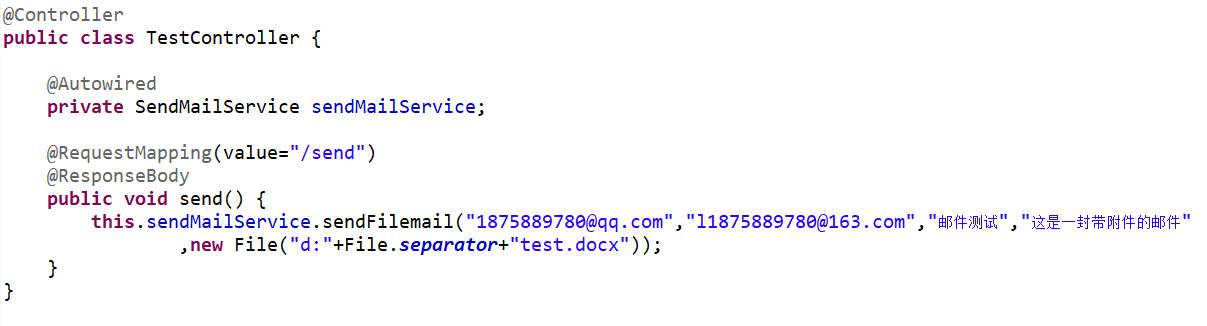
### 发送带附件的邮件

1. 创建邮件服务类，如下：



通过调用addAttachment方法增加附件，如果有多个附件，则调用多次；该方法的第一个参数表示附件名称，第二个参数表示具体的附件；

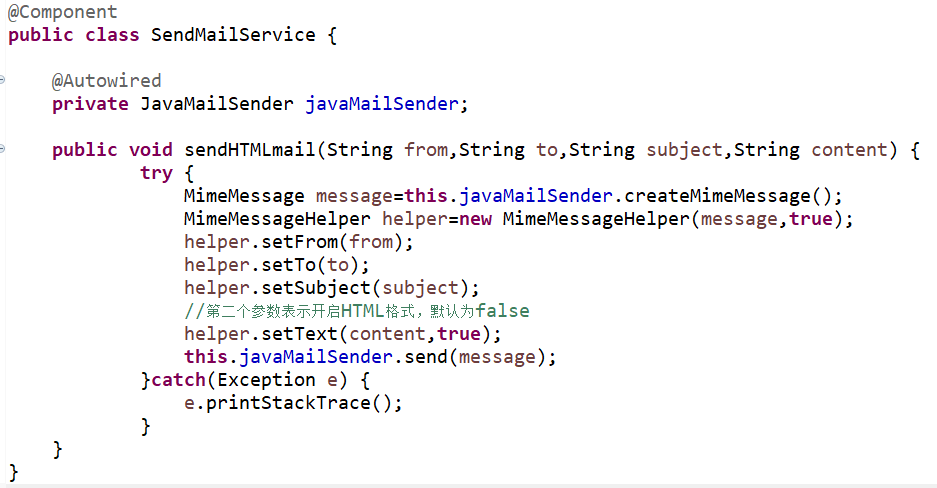
1. 进行测试，如下：



访问路径，查看收件者邮箱是否收到邮件。

### 发送HTML格式的邮件

1. 创建邮件服务类，如下：



在调用setText方法设置邮件内容时，该方法的第二个参数为true时，表示发送HTML格式的文本内容，默认为false。

1. 进行测试，如下：



启动项目，访问该路径，查看收件人邮箱是否收到邮件。

# MyBatis缓存

MyBatis缓存分为一级缓存和二级缓存，一级缓存默认情况下是开启的，二级缓存默认是关闭的，需要手动开启和配置。

缓存的查找顺序是先去二级缓存中找，如果没有就去一级缓存中找，一级缓存中没有则取数据库中取。

## 一级缓存

一级缓存也称本地缓冲，在默认情况下就是开启的，无需配置。

### 使用

一级缓存默认开启，使用一级缓存，会将缓存保存在一个sqlSession对象中，如果该sqlSession对象被关闭，则缓存也会被释放；不同sqlSession对象不能使用其他sqlSession中保存的缓存。

当MyBatis与Spring整合后，一级缓存会失效，所以此时需要搭配二级缓存使用。

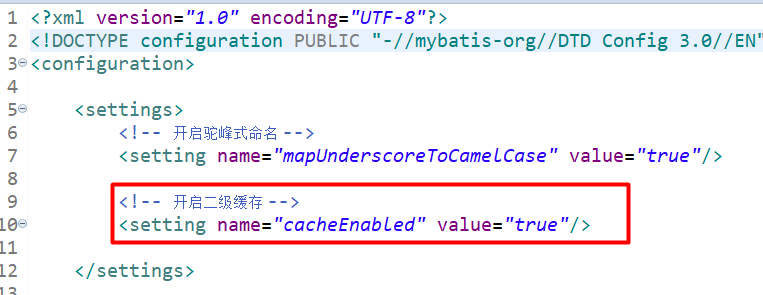
## 二级缓存

二级缓存也称全局缓存，是基于Mapper文件级别的缓存，一个Mapper文件对应一个二级缓存。

### 使用

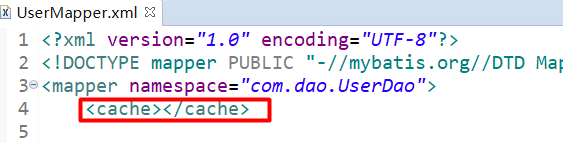
使用二级缓存很简单，只需要3步；

1. 在mybatis配置文件中开启全局二级缓存，如下：

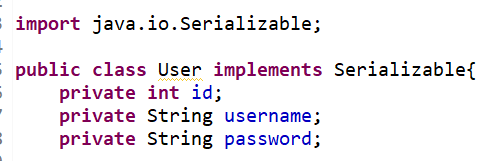


为false表示关闭二级缓存；

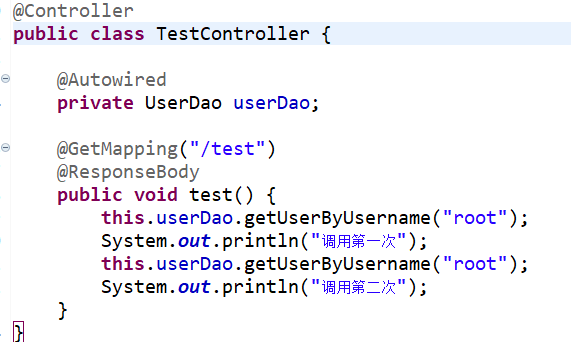
1. 因为二级缓存是基于Mapper级别的缓存，所以需要在XXXMapper.xml文件中进行配置，表示该mapper需要使用二级缓存。如下：



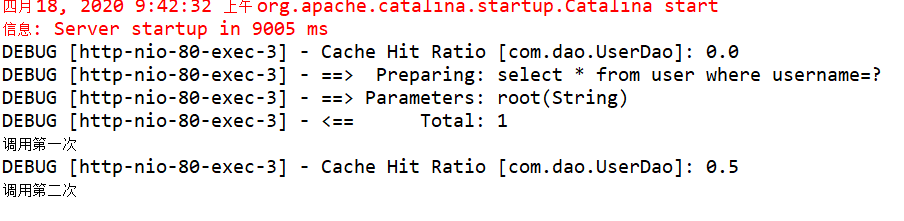
3）pojo类需要实现序列化接口，如下：



1. 进行测试，如下：



观察打印台，如下：

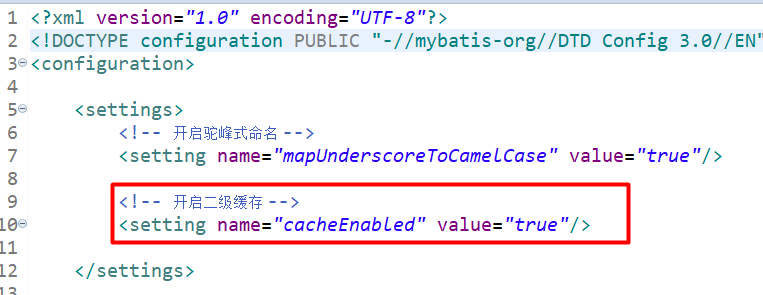


可以观察出，虽然调用了两次，但实际上只查询了一次，而第二次是使用的缓存；

Cache Hit Ratio表示缓存命中率；

### 常用设置

1. 在MyBatis配置文件中，通过cacheEnabled开启二级缓存，如下：



cacheEnabled是二级缓存的开关，当值为true表示开启二级缓存，值为false表示关闭二级缓存，默认值是true，所以不配置该项也可以；

1. 二级缓存是基于Mapper文件级别的缓存，在mapper文件中添加<cache>标签表示该mapper文件将启用二级缓存，这个<cache>标签有4个常用的可选属性，如下：



eviction：指定缓存的回收策略：

·LRU – 最少使用的；移除长时间不被使用的缓存对象（默认策略）

·FIFO – 先进先出，按对象进入缓存的顺序进行回收；

·SOFT – 软引用，移除基于垃圾回收器状态和软引用规则的对象；

·WEAK – 弱引用，更积极的移除基于垃圾收集器状态和弱引用规则的

对象；

flushInterval：指定缓存刷新间隔，即缓存多久清空一次，单位毫秒；（默认不

清空）

readonly：指定缓存是否只读：

·true – 表示只读，MyBatis将会认为所有从缓存中获取的操作都是只

读操作，不会修改数据；Mybatis为了加快获取缓冲的速度，直接就会将数据在缓存中的引用交给用户；不安全但速度快；

·false – 表示非只读，Mybatis将会认为数据可能会被修改，从而利用

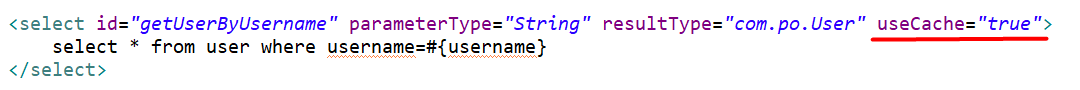
序列化与反序列化技术克隆一份新的技术给你；安全但速度慢，

（默认为false）

size：指定缓存存放多少个元素；

对于这些可选属性，一般默认即可，不需要单独设置；

1. 在mapper文件中的每个<select>标签都有一个userCache属性，该属性表示该select语句是否使用二级缓存，默认值为true。如下：



当值为false表示该select语句不使用二级缓存；

1. mapper文件中每个<insert>（增）、<update>（改）、<delete>（删）标签上都有一个flushCache属性，如下：



表示执行增删改操作后是否清空一级和二级缓存，默认为true；

<select>标签也有flushCache属性，但默认值为false；

一般这个属性不需要修改，取默认值即可。

1. 调用localCacheScope方法会清除一级缓存，但不会清除二级缓存；

### 二级缓存的缺点

MyBatis二级缓存是基于Mapper级别的，但如果存在多表关联，则其他Mapper中的缓存不会被刷新，此时最好将这些关联表使用同一个缓存，如下：



虽然这样，但还是无法掩盖MyBatis二级缓存的缺点，所以推荐如果这个Mapper文件中的SQL只涉及到单表的话，使用二级缓存是没啥问题的，但如果SQL语句涉及到多表，则推荐放弃使用二级缓存，在业务层使用可控制的缓存。

## MyBatis整合第三方缓存

### EhCache

EhCache是一个纯Java的进程内缓存框架，具有快速、精干等特点，它具有内存和磁盘存储。其特点如下：

1. 快速、简单；
2. 多种缓存策略；
3. 缓存数据有两级：内存和磁盘，因此无需担心容量问题；
4. 缓存的数据会在虚拟机重启的过程中写入磁盘；

特点不限于此；

### 配置

使用EhCache都需要创建一个XML配置文件，用于配置缓存策略，通常是在classpath路径下创建一个ehcache.xml文件，在该文件中配置缓存策略，在配置缓存策略时，至少需要配置一个默认的缓存策略，也可以配置多个缓存策略；

如定义一个默认的缓存策略（使用<defaultCache>标签定义）



name：指定缓存策略的名称，默认的缓存策略不需要此属性；

path：缓存数据持久化存放的目录；

maxElementsInMemory: 内存中缓存的element的最大数目；

maxElementsOnDisk：在磁盘中缓存的element的最大数目，默认值为0，表示不限制；

eternal：设置elements是否过期，如果为true，则表示缓存始终有效，如果为false，

将根据timeToIdleSeconds属性和timeToLiveSeconds属性判断；

timeToIdleSeconds：对象空闲时间，只有当eternal属性值为false才会生效（秒）；

timeTOLiveSecnods：对象存活时间，只有当eternal属性值为false才会生效（秒）；

overflowToDisk：内存中数据超过限制，是否需要缓存到磁盘；

diskPersistent：是否在磁盘上持久化，重启jvm后，数据是否有效；

diskExpiryThreadIntervalSeconds：磁盘缓存的清理线程的运行间隔；

memoryStoreEvictionPolicy：超过内存空间，向磁盘存储数据的策略，可选值为LRU（最

近最少使用）、FIFO（先进先出）、LFU（最少使用），通常值设置为LRU；

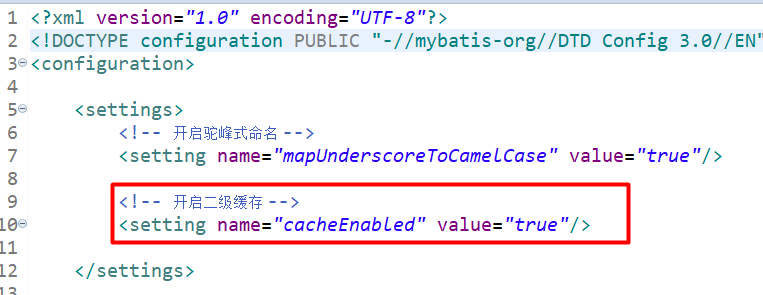


### 使用

1. 导入EhCache核心包以及与MyBatis的整合包，如下：



1. 开启MyBatis二级缓存，如下：



1. 在Mapper.xml文件中开启二级缓存，并指定使用EhCache缓存，如下：



1. 在classpath下创建ehcache.xml配置文件，用于定义缓存策略，如下：



这样就可以使用EhCache缓存了。

关于MyBatis整合EhCache文章可以参考：

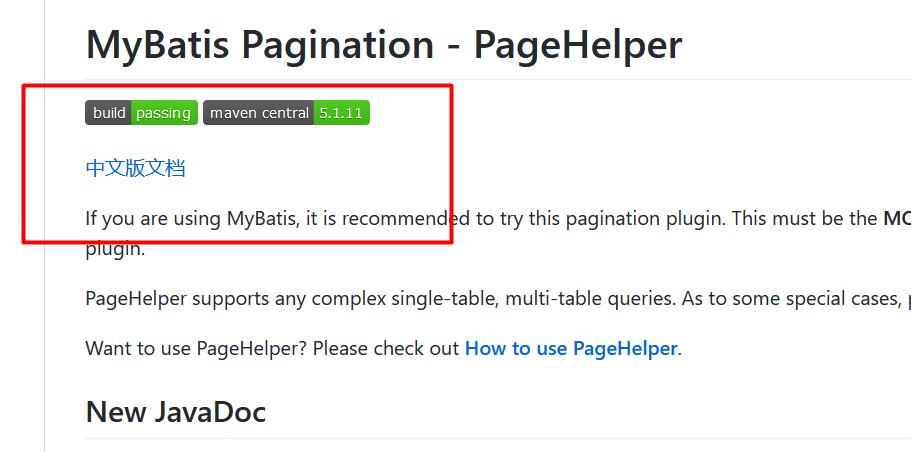
<https://blog.csdn.net/m0_37448438/article/details/82252850>

<https://www.cnblogs.com/sdream/p/5966668.html>

# MyBatis分页插件

## 介绍

PageHelper插件是MyBatis一款非常强大的分页插件，你可以访问这个插件的开源地址<https://github.com/pagehelper/Mybatis-PageHelper>，它提供了文档教你如何使用这个分页插件：

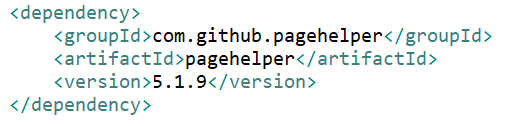


## 使用前提

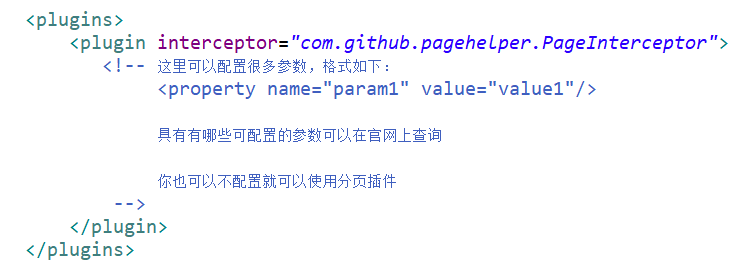
1. 使用分页插件需要引入到这两个jar包，



在Maven项目中，只需要引入pagehelper包即可，因为Maven会自动引入依赖包：



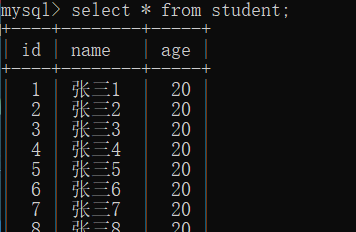
1. 在mybatis配置文件中配置插件：



注意：这个插件在mybatis配置文件的位置必须严格遵循mybatis配置文件的规范（即配置要在相对的位置）。

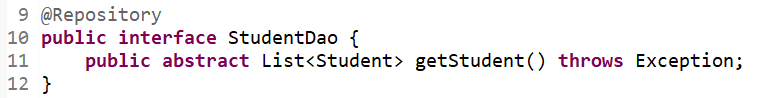
## 使用

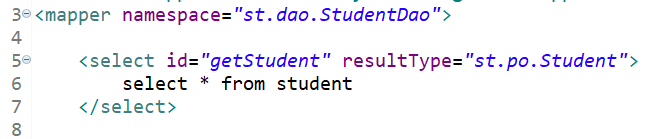
1）创建数据表student



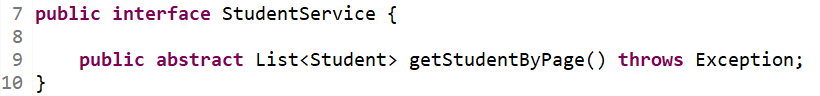
在表中插入若干数据，作为要显示的分页数据；

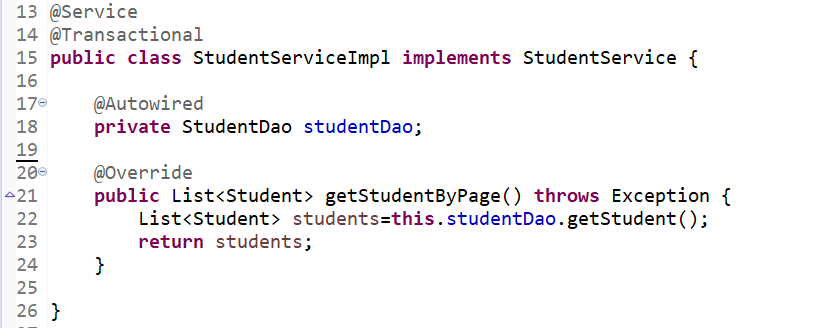
1. 为student表创建实体类后，编写dao接口和mapper文件，关键代码如下：



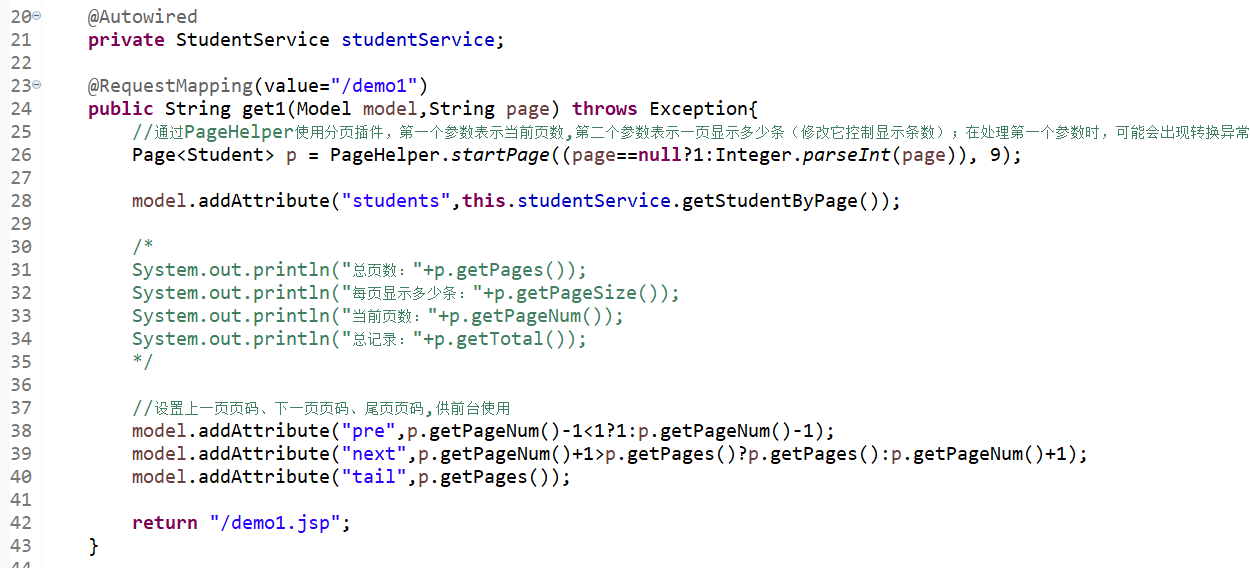


1. 创建业务层接口StudentService和实现类StudentServiceImpl，关键代码如下：



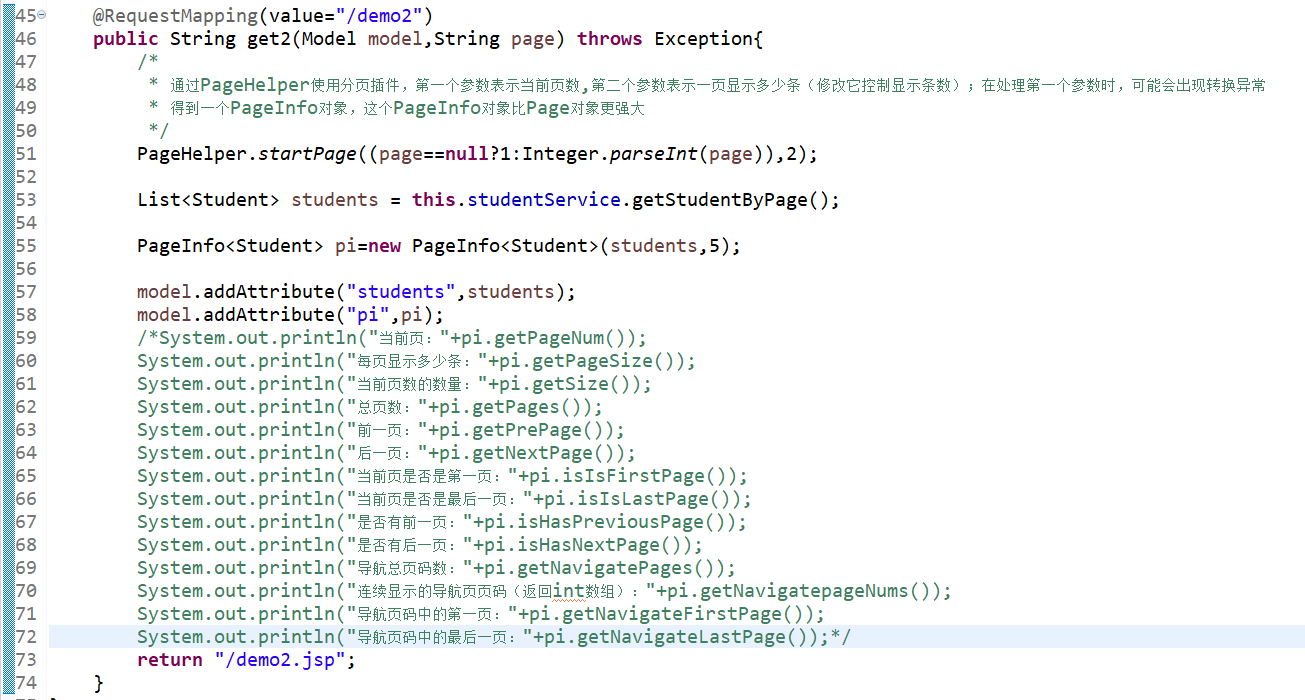


1. 创建控制层，在控制层实现分页功能：



关键代码在于第26行，通过PageHelper调用startPage方法实现分页，startPage方法的第一个参数表示你要显示第几页的，第二个参数表示每页显示的条数，调用startPage方法会返回一个Page对象，这个Page对象有很多属性，如总页数、每页显示数、当前所在页数、总记录数等等，这些属性的值由startPage传入的两个参数所决定的，并且需要注意的是startPage这个方法必须在执行SQL语句之前进行调用；

1. 除了通过Page对象搭配PageHelper.srartPage实现分页，也可以通过PageInfo对象搭配PageHelperPage实现分页，PageInfo对象比Page对象的功能更强大。如下：

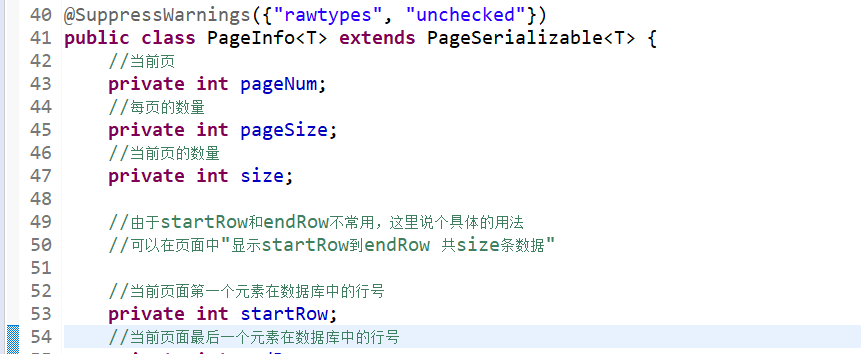


关键代码在于第51行、第53行、第55行；

## 小结

实现分页的关键点：

1. 使用PageHelper类调用startPage方法进行分页；
2. 选择Page对象或PageInfo对象保存分页后的结果或各种属性，PageInfo比Page更强大；
3. 你可以查看Page或PageInfo的开源代码，它们都提供了中文注释：

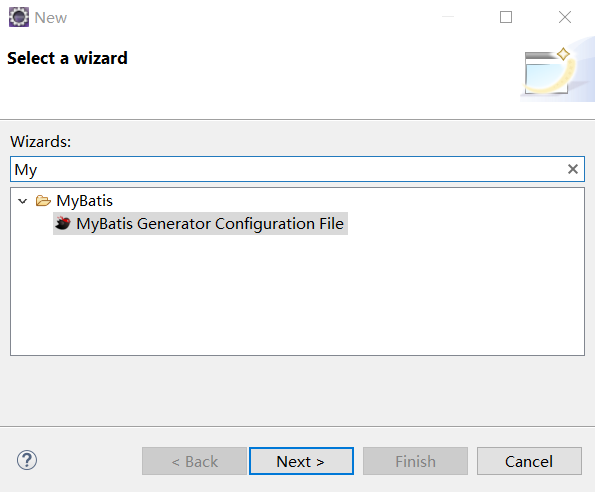


# 使用Mybatis Generator

MyBatis Generator是一个代码生成插件。

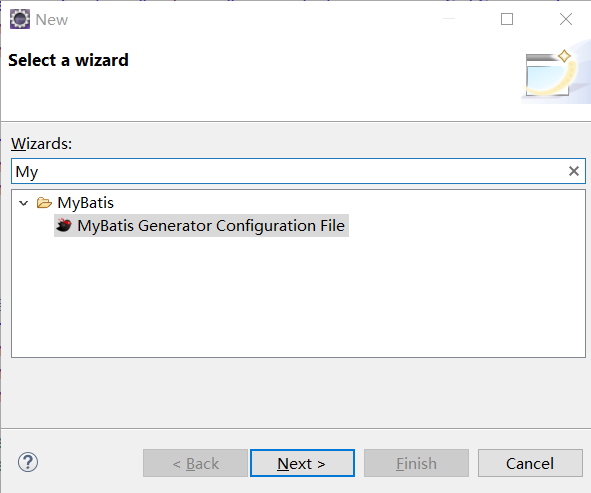
## 安装

1. 在<https://dl.bintray.com/mybatis/mybatis-generator/zipped/>下载mybatis generator插件，解压缩后将其中的features和plugin放到Eclipse安装目录中的dropins目录中；
2. 在项目上鼠标右键点击【New】-->【Other】，如果出现了MyBatis插件，则表示插件安装成功。



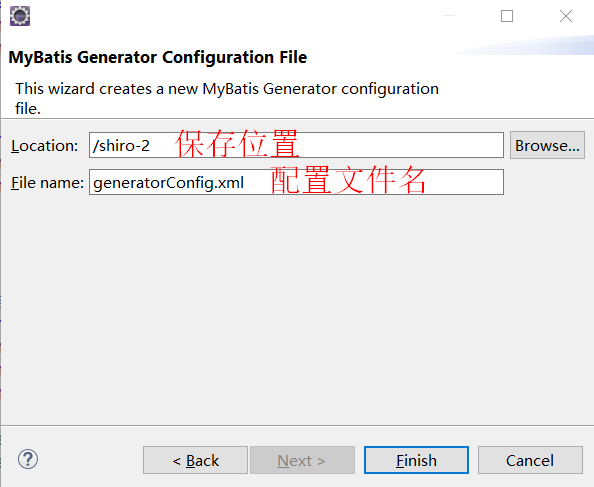
## 使用

1. 选中项目，点击【New】-->【Other】，选择MyBatis下的【MyBatis Generator Configuration File】：



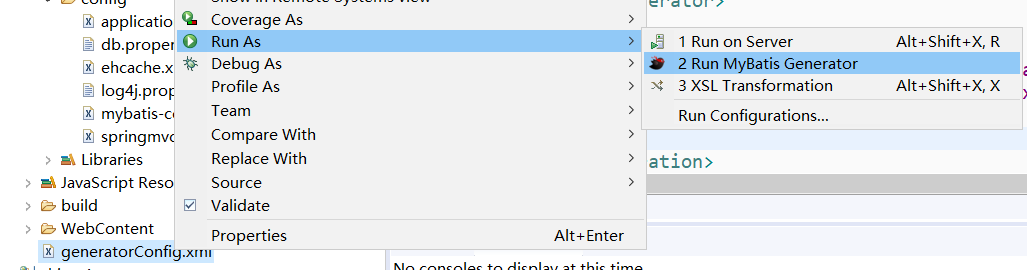
它表示生成mybatis generator插件的配置文件；

1. 填写mybatis generator插件配置文件的相关信息：



点击【Finish】后，会在指定位置生存一个mybatis generator插件的配置文件；

1. 打开mybatis generator配置文件，编写生成规则：
2. 配置完成后，运行该配置文件：



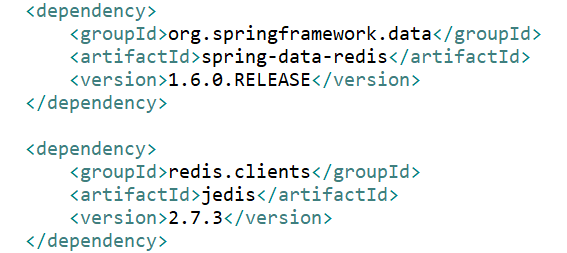
# SSM整合Redis缓存

<https://blog.csdn.net/qq_39056805/article/details/79948304>

## 入门案例

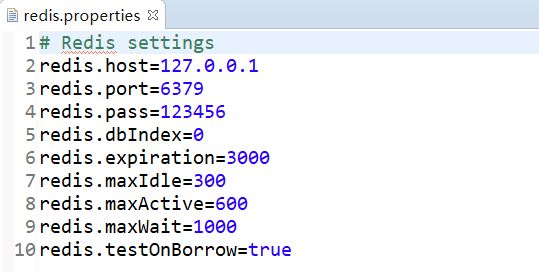
此案例里是使用Redis作为SSM的缓存，并非是MyBatis的二级缓存。

1. 导入以下两个jar包：

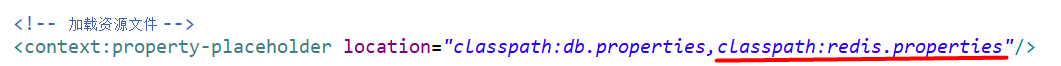


注意存在版本问题；

1. 编写redis数据库配置文件，如下：



1. 在applicationContext.xml配置文件中引入redis配置文件，并进行整合，如下：

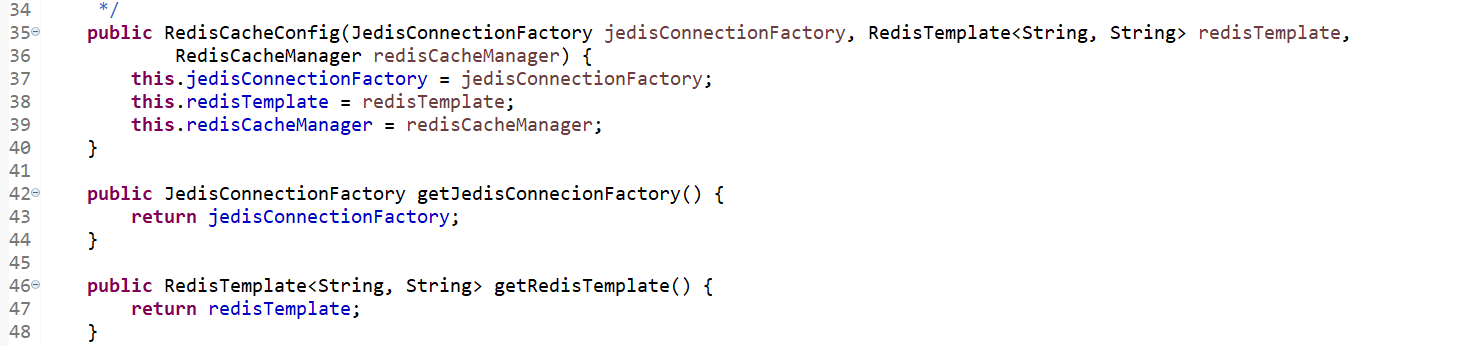




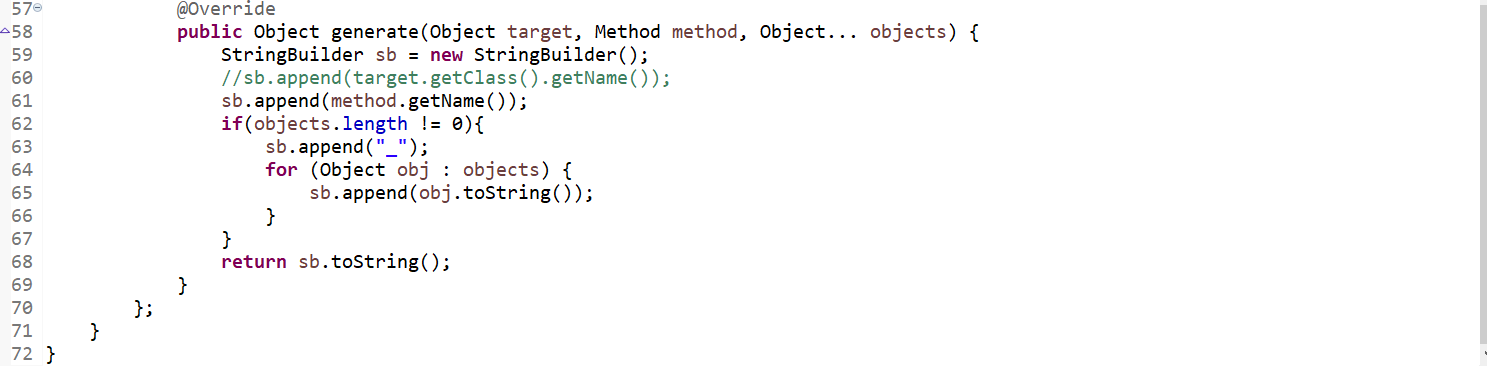


1. 创建RedisCacheConfig类，它所在的包路径与整合Redis时指定的一致，类中内容如下：

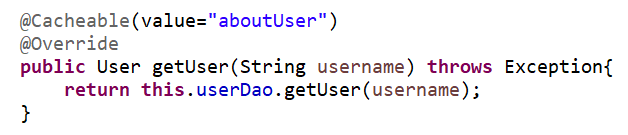




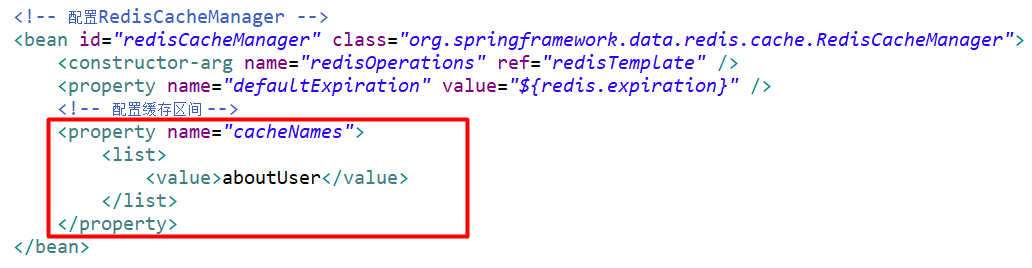




1. 在Service层使用@Cacheable实现缓存，如下：

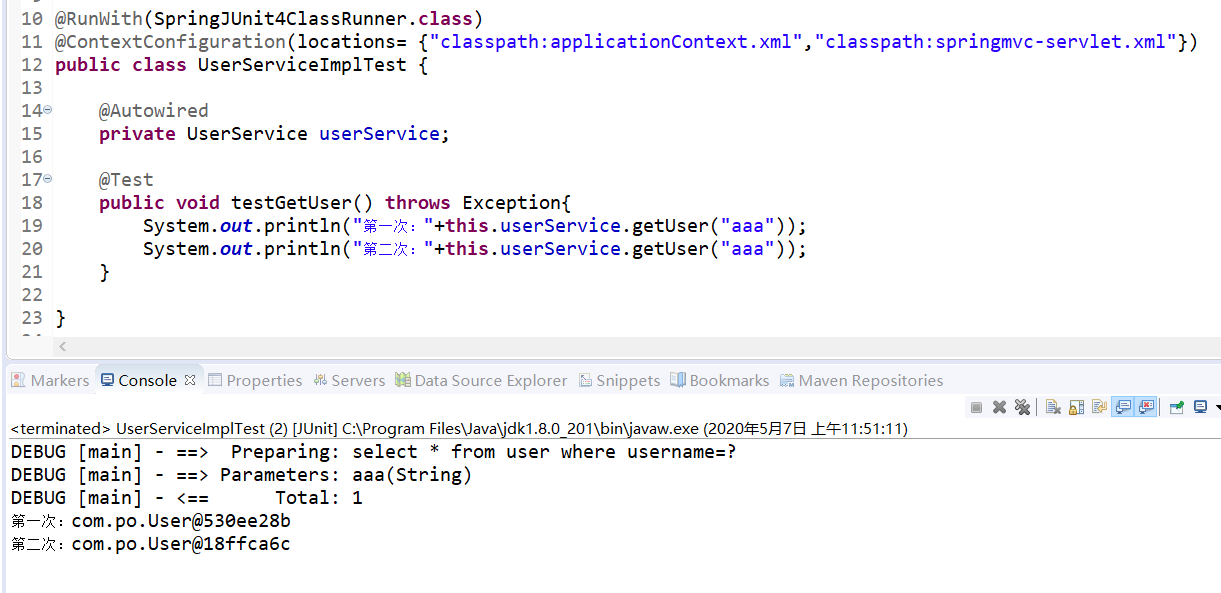


value属性用于指定缓存区间的名称，这个缓存区间的名称可以是在Spring与Redis整合时定义的，如下：



如果没有定义则会自动创建；

1. 编写单元测试，测试缓存是否成功（执行测试前记得开启Redis服务器），如下：



从执行结果上看，只查询了一次，第二次使用了缓存（实体类需要实现序列化接口）；

## 常用注解

SSM与缓存相关的注解有3个：@CacheConfig，@Cacheable，@CachePut，@CacheEnict

1）@CacheConfig

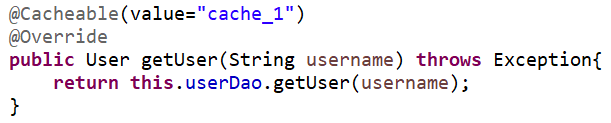
@CacheConfig注解可以标注在类上，用于指定要使用的缓存的名称，如下：



它是可选的，可以不使用它，而直接在@Cacheable注解中指定缓存的名称；

2）@Cacheable

被@Cacheable标注的方法表示要对这个方法进行缓存，如下：

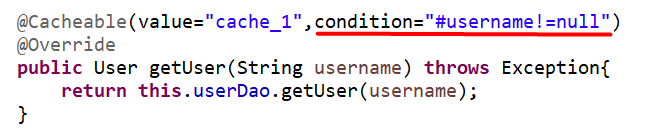


@Cacheable有3个常用的可选属性：value、key、condition

value用于指定要使用的缓存区，如果该类上没有使用@CacheConfig指定，那么必须使用value参数指定要使用的缓存区；

key用于指定缓存中key的名称，如果使用该参数，则必须按照SpEL编写，如果不指定，在默认情况下，缓存的key是方法的参数值，缓存的value是方法的返回值，当在其他类中调用此方法时，首先会根据参数查看缓存中是否有相关数据，若有，则直接使用缓存数据，该方法不会被执行，否则执行该方法，执行成功后再将返回值缓存起来，需要注意：在当前类中调用该方法缓存是不会生效的；（这个key除了可以按照SpEL自定义，还可以根据keyGenerator自定义）

condition用于指定缓存时机，如：



表示当形参username的值不为null时，才对这个方法进行缓存；

3）@CachePut

@CachePut一般标注在数据更新的方法上，与@Cacheable不同，在执行@CachePut注解标注的方法时不会检查缓存中是否有数据，而是直接执行方法，然后将方法的执行结果缓存起来，如果key对应的数据已经被缓存起来了，就会覆盖之前的数据，这样就可以避免加载到脏数据；它的属性与@Cacheable相似；

4）@CacheEvict

@CacheEvict注解一般标注在删除数据的方法上，表示移除一个key对应的缓存，@CacheEvict注解有五个常用的属性：value、key、condition、allEntries和beforeInvocation：

value表示清除操作是发生在哪些Cache上的（缓存名称）；

key用于指定要移除的key的名称；

condition表示清除操作发生的条件

allEntries表示是否将所有的缓存都移除，默认为false；

beforeInvocation表示是否在方法执行之前移除缓存中的数据，默认为false，

即在方法执行之后再移除缓存中的数据，这样方法如果因为抛出异常而未能成功返回时也不会触发清除操作；

# 单元测试

关于单元测试可以参考以下文章：

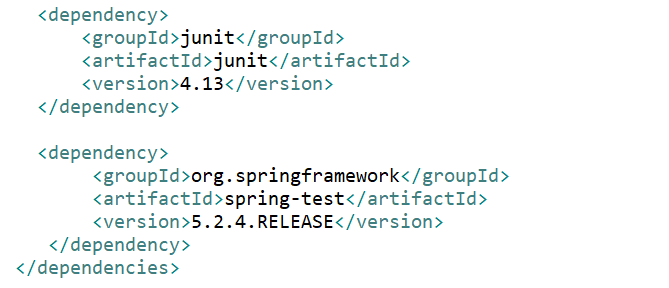
<https://louluan.blog.csdn.net/article/details/37562165>

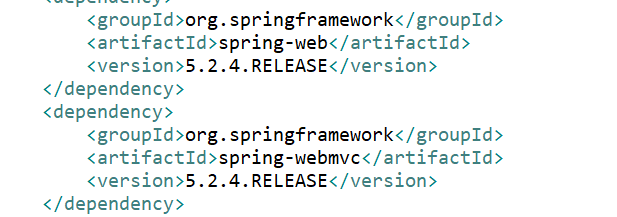
<https://jaune162.blog.csdn.net/article/details/40025861>

<https://blog.csdn.net/yl405001832/article/details/99894622>

## 准备

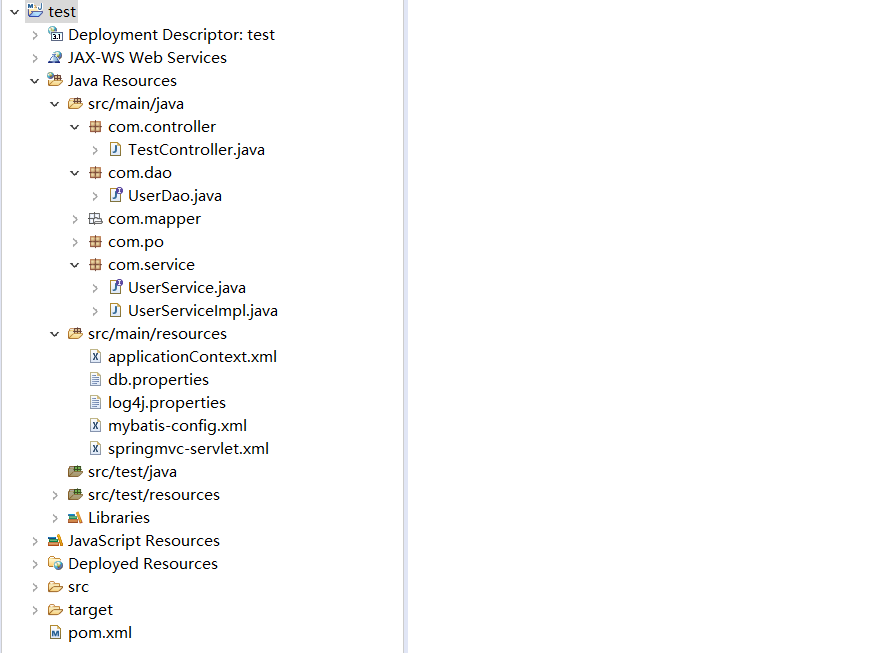
导入以下两个jar包：



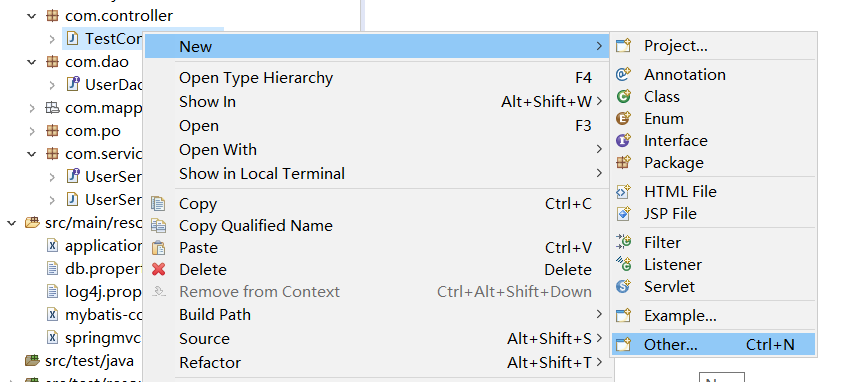
 注意：spring-test的版本应该和其他spring系列jar包的版本一致，如下：

## 测试控制层

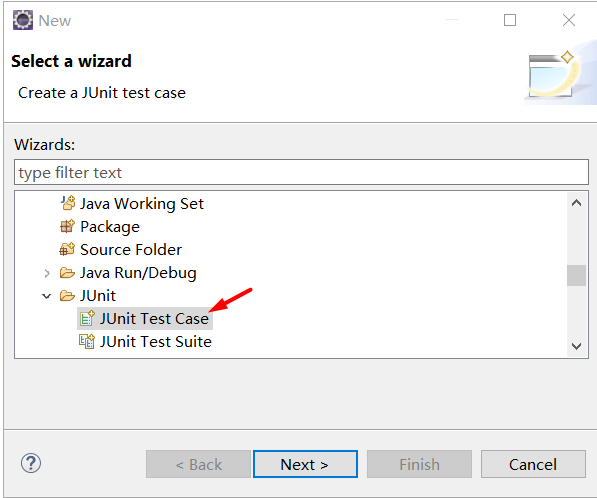
项目结构如下：



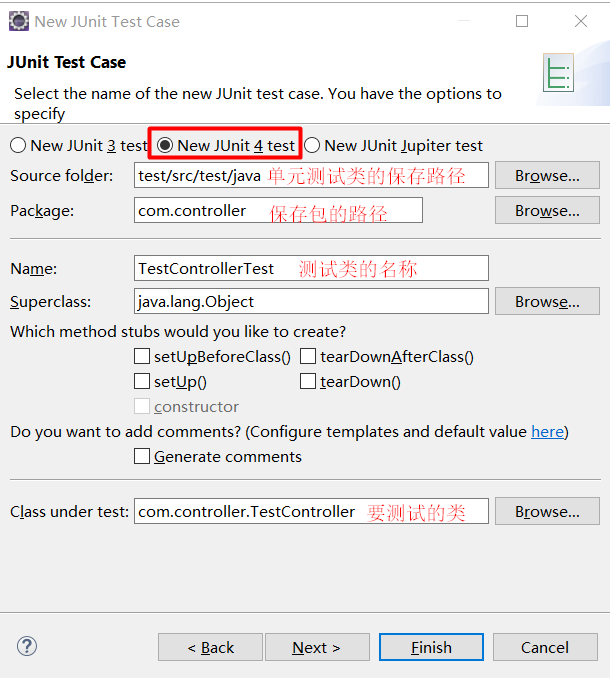
1. 选中要测试的Controller类，这里以TestController这个控制类为例，鼠标右击，打开【New】下的【Other】（如果没有则打开【Other】寻找），如下：



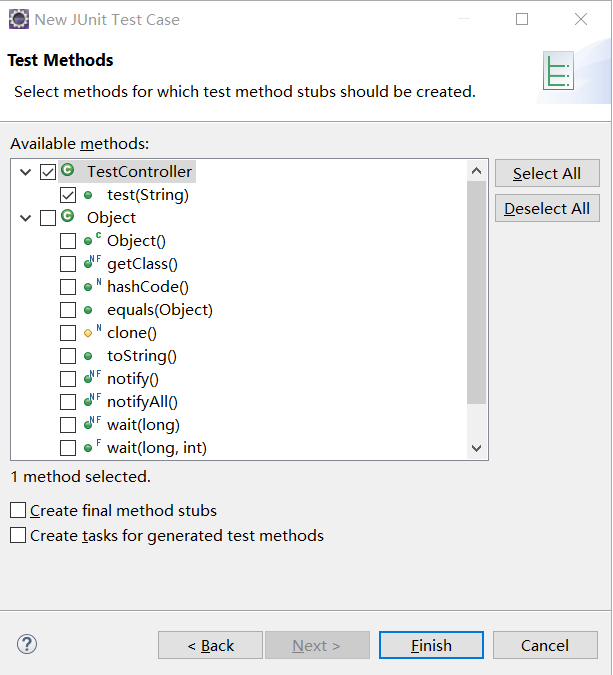
打开【Junit】下的【Junit Test Case】，如下：



1. 填写相关信息，如下：

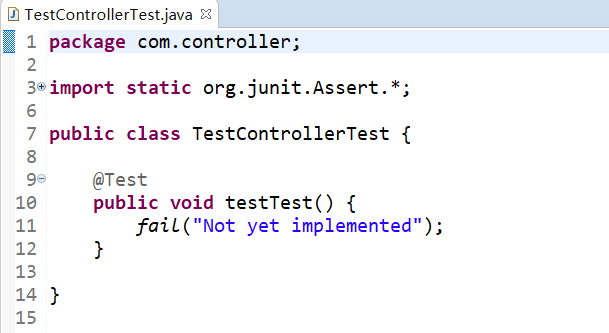


填写完成后，选择要测试的方法：



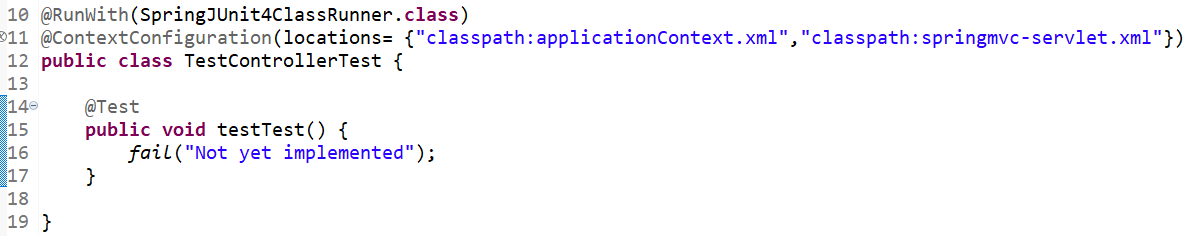
由于测试的TestController类中只定义了一个test方法，所以选择这个方法进行测试。

点击【Finish】后，生成的类如下：



@Test标注的方法表示要执行测试的方法，它的命名规则为“要测试的方 法的名称+Test”；

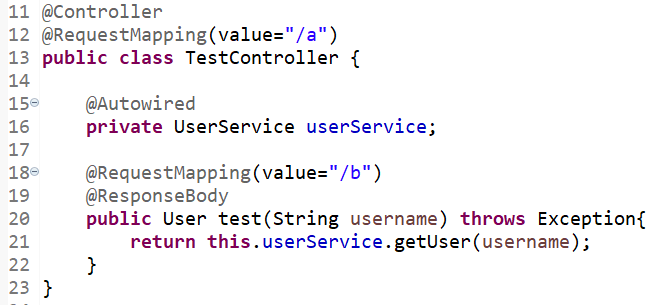
1. 在测试类上标注@RunWith注解和@ContextConfiguration注解，如下：

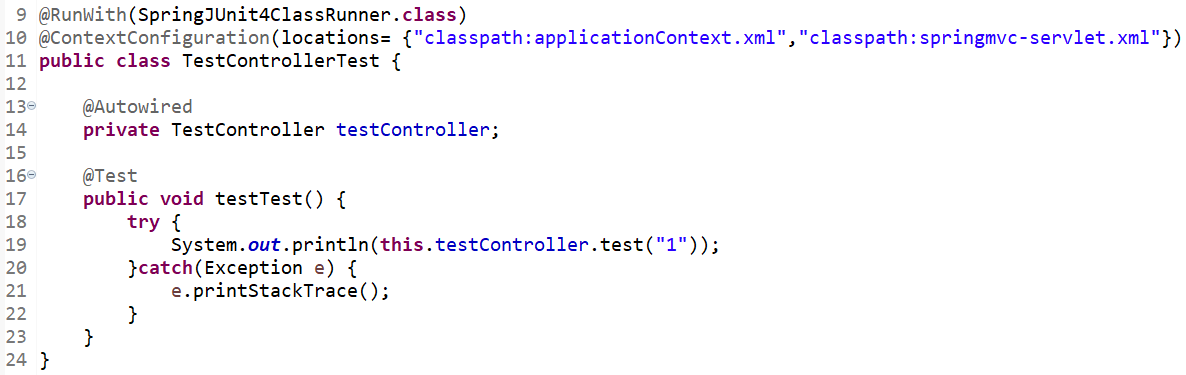


@ContextConfiguration注解用于指定配置文件所在的位置，这些配置文件是控制器执行时依赖的配置文件；

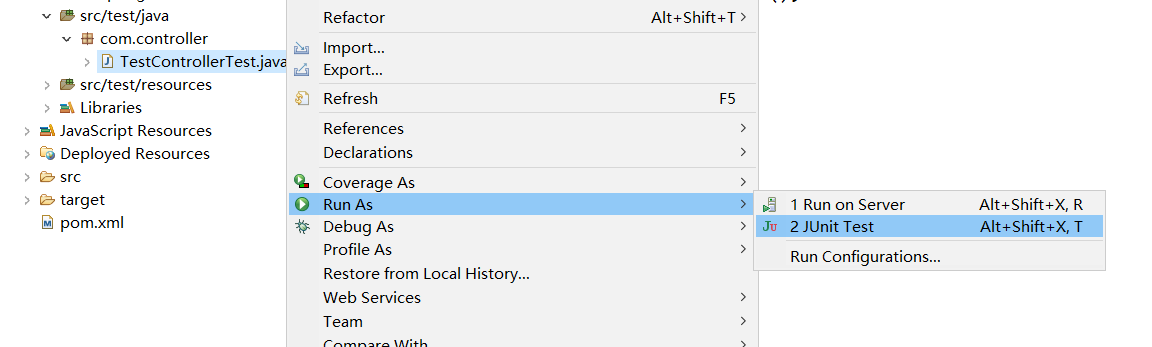
1. 根据要测试的方法，编写内容，进行测试

要测试的test的方法如下：

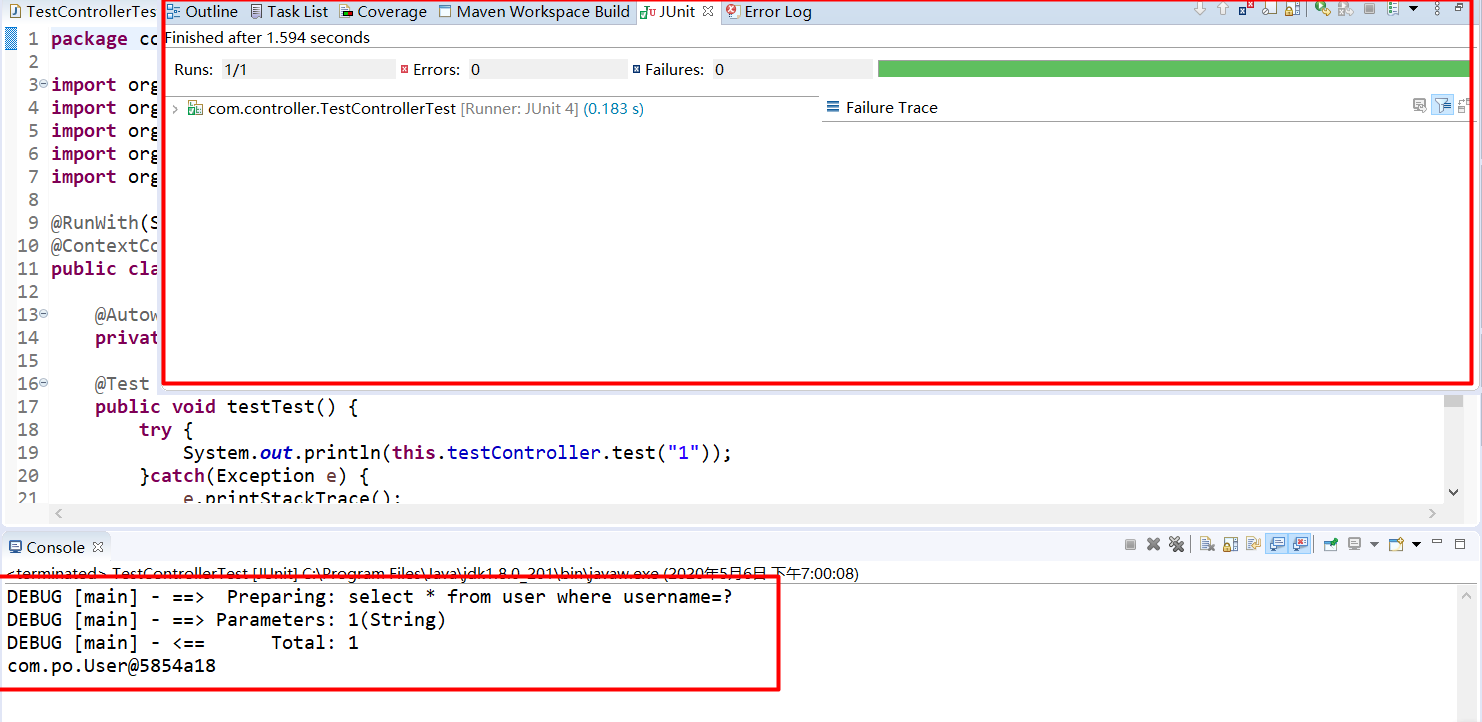


 编写测试，如下：

1. 执行测试类，如下：

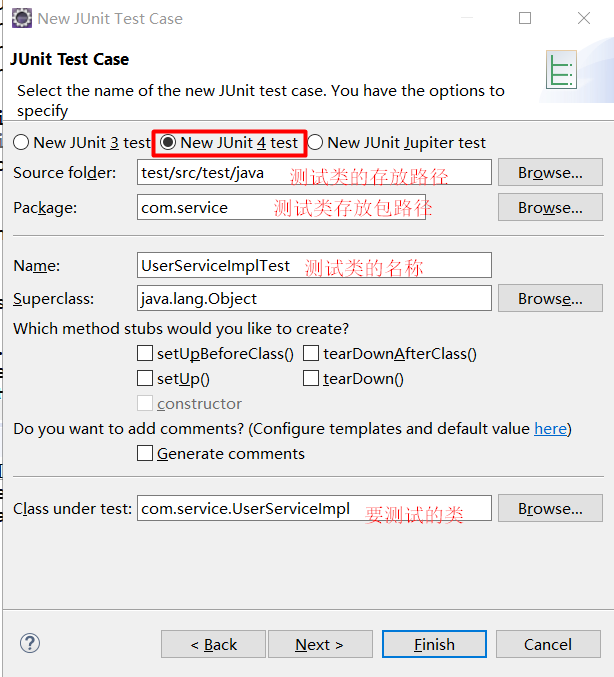


结果如下图所示：

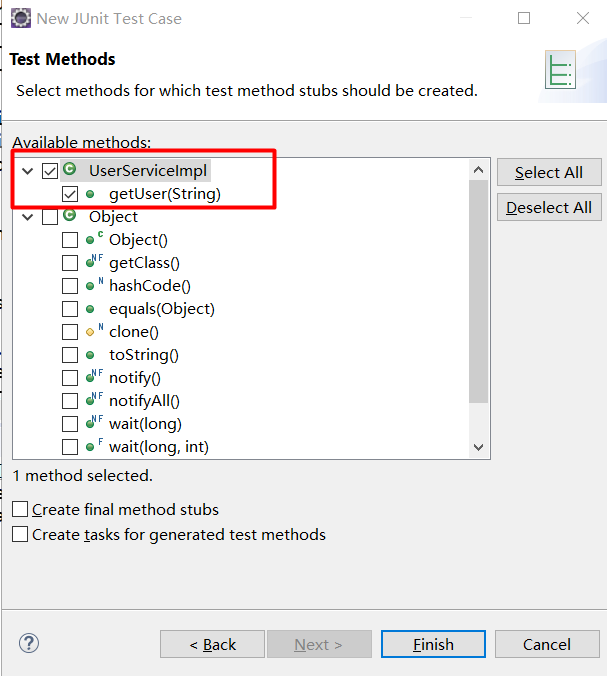


## 测试业务层

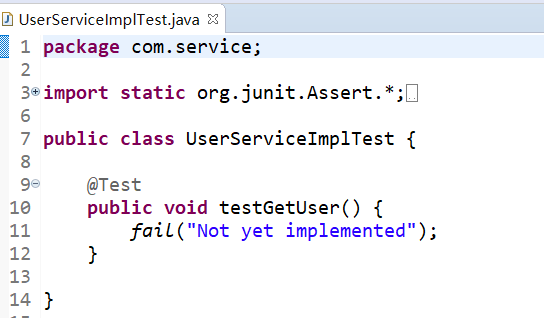
1. 这里以要测试的UserServiceImpl这个类为例，选中该类，鼠标右击，打开【New】下的【Other】，找到【Junit】下的【Junit Test Case】并打开它，打开后，填写相关信息，如下：



点击【Next】，选择要测试的方法，如下：

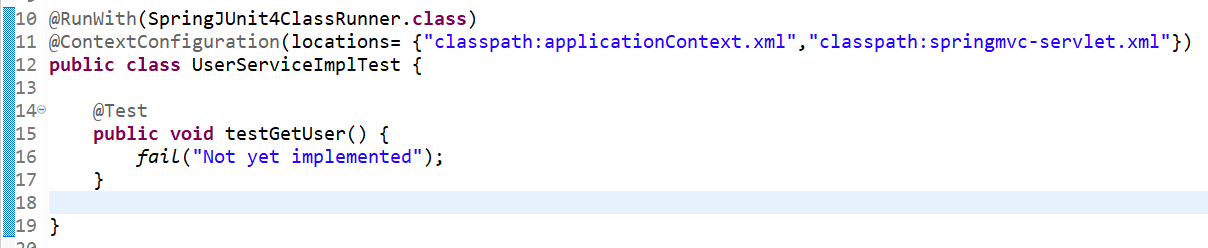


点击【Finish】，生成的测试类如下：



@Test标注的方法表示要执行的方法；

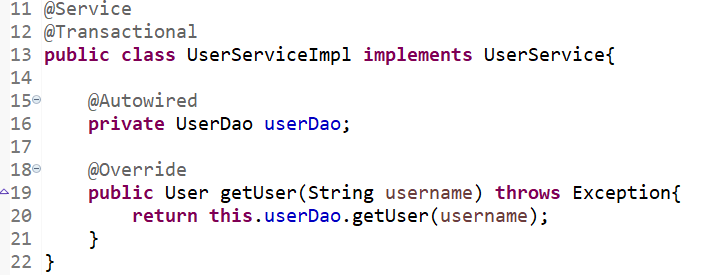
1. 添加@RunWith注解和@ContextConfiguration注解，如下：



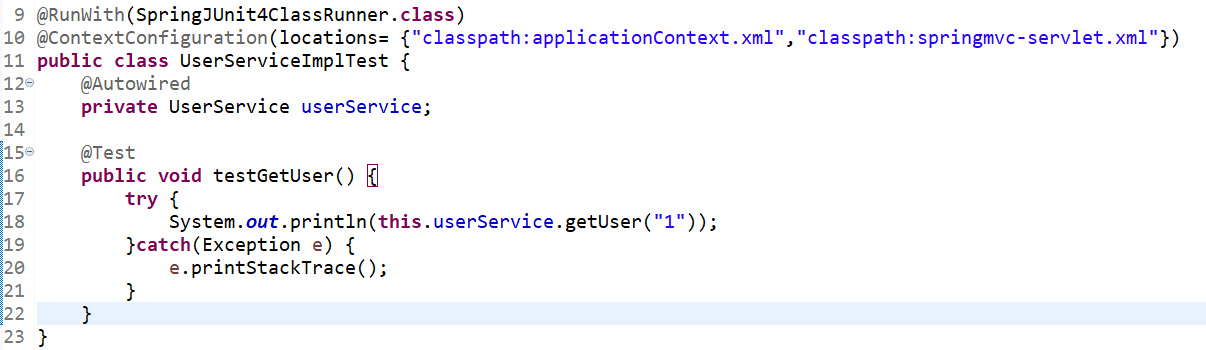
@ContextConfiguration注解用于指定配置文件的位置，配置文件时执行要测试的方法所依赖的配置文件；

1. 根据要测试的方法，编写测试内容

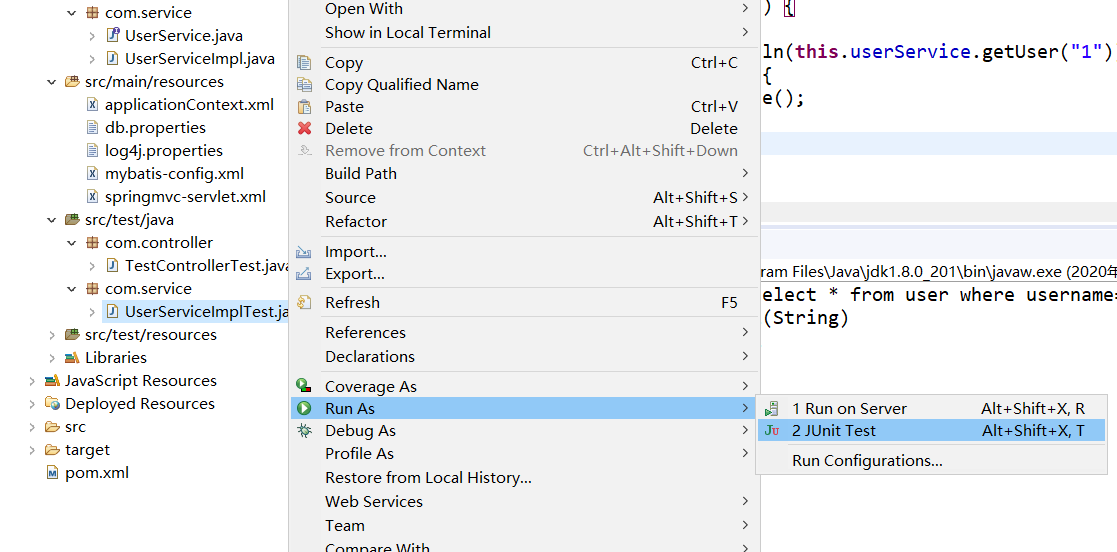
UserServiceImpl中要测试的getUser方法如下：



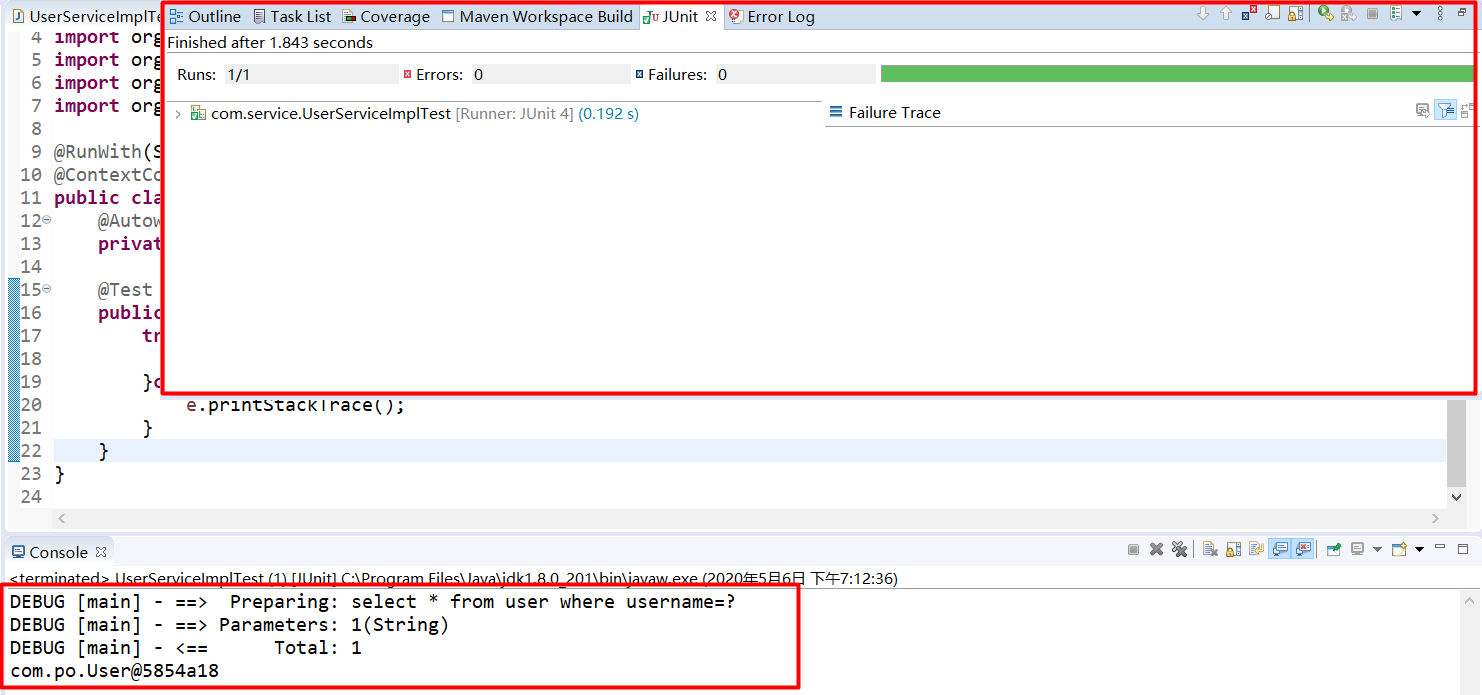
根据这个方法编写测试内容，如下：



1. 执行测试，如下：

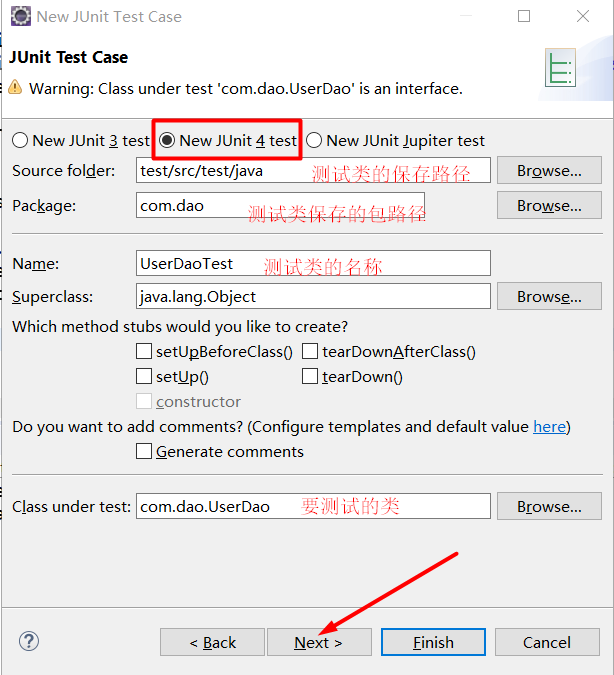


执行结果如下：

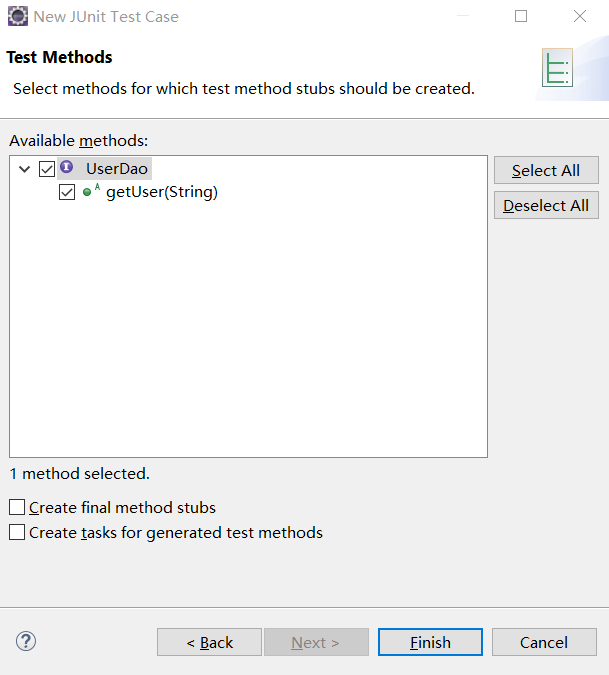


## 测试持久层

1. 选择要测试的类，这里以UserDao为例，选中该类，鼠标右键打开【New】下的【Other】，打开【JUnit】下的【Junit Test Case】，填写相关信息，如下：

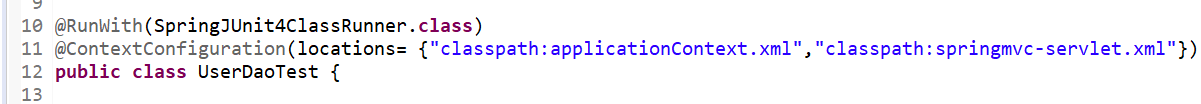


点击【Next】，选择要测试的方法，如下：



点击【Finish】完成。

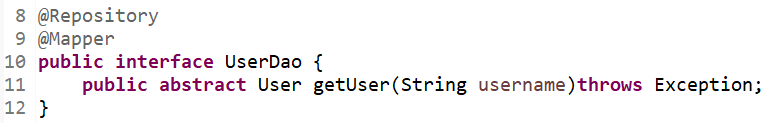
1. 添加@RunWith注解和@ContextConfiguration注解，如下：

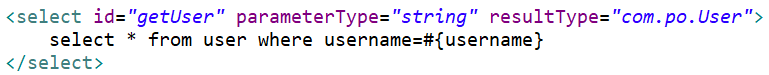


@ContextConfiguration注解用于指定测试要依赖的配置文件；

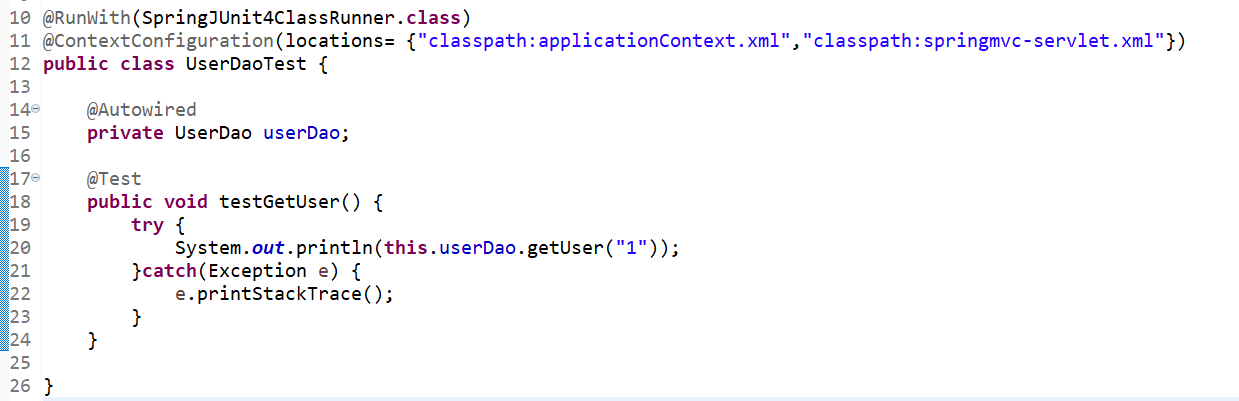
1. 根据要测试的方法，编写测试内容

要测试的UserDao接口中的getUser方法如下：

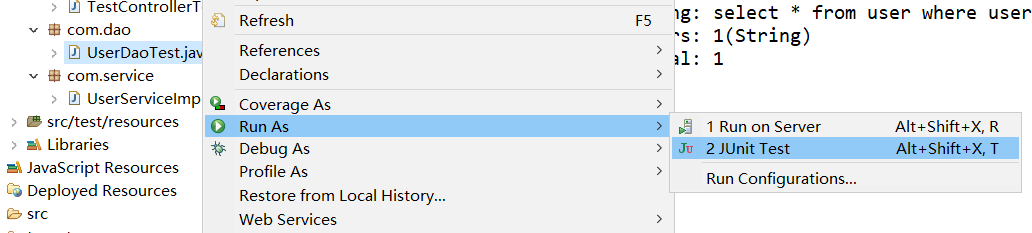




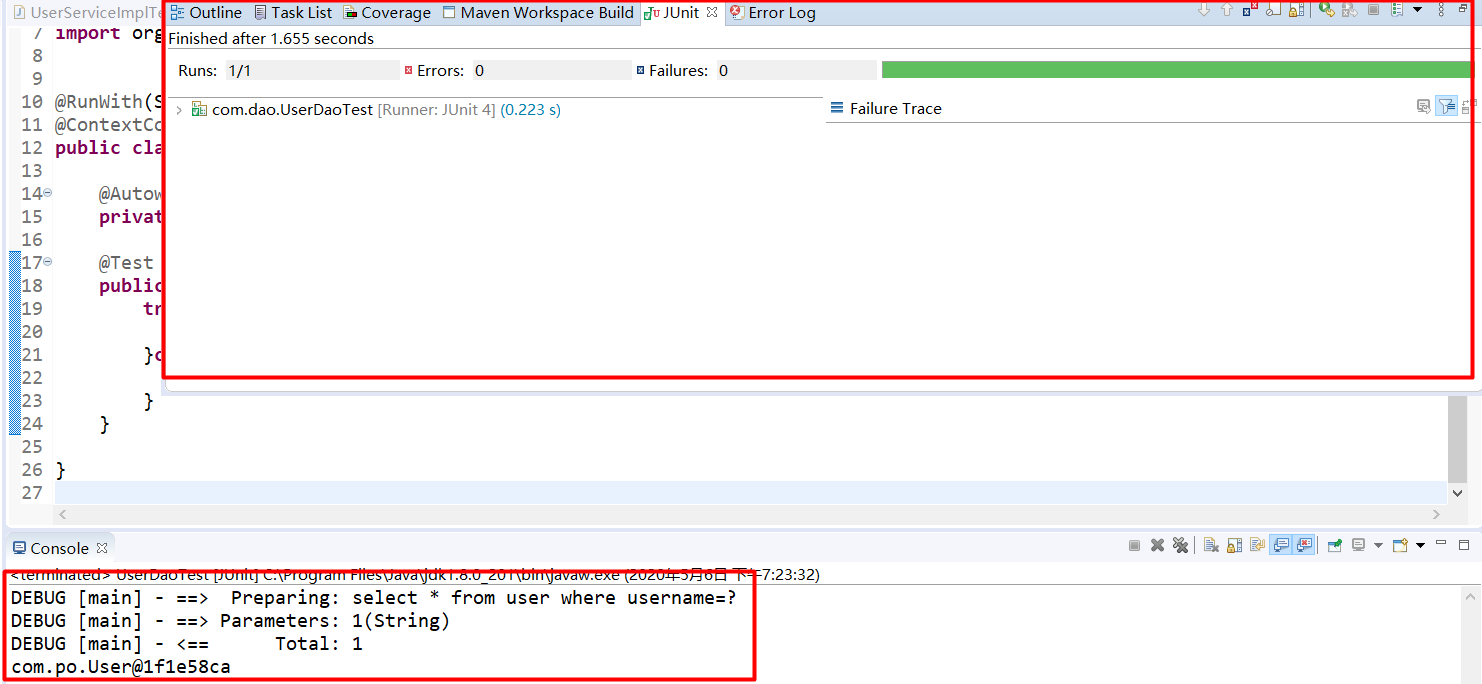
编写测试的内容如下：



1. 运行测试类，如下：



执行结果如下：



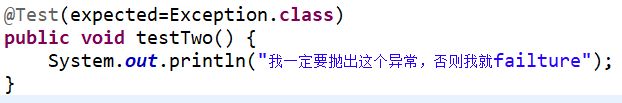
## 常用注解

单元测试常用的注解有@RunWith，@ContextConfiguration，@Test，@Before，@After，@BeforeClass，@AfterClass，@Ignore

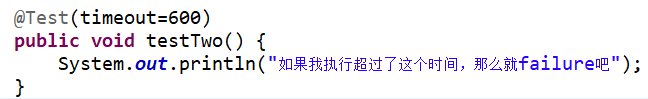
1）@ContextConfiguration注解用于添加单元测试所依赖的配置文件；

2）@Test注解用于标注测试方法，每个使用@Test标注的方法都是一个可以被Junit执行的test case，它有两个可选属性：expected和timeout

·expected：它的值可以是一个Throwable的子类对象，如果@Test加上这个属性，表示JUnite期待这个方法抛出该异常，如果没有抛出，则单元测试failure；如：

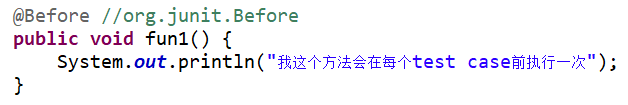


·timeout：timeout值是一个long类型，如果@Test注解加上该属性，表示方法的运行时间不能超过该数值，否则单元测试failture；如：



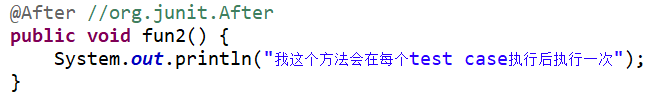
3）@Before注解标注的public void方法将会在每个test case，可用户初始

化操作，开启资源等。如下：



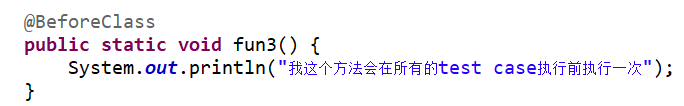
4）@After注解标注的public void 方法将会在每个test case执行后执行一次，一般关闭

资源等。如下：



5）@BeforeClass注解标注的方法必须是public static void类型的，并且无参数，对于整

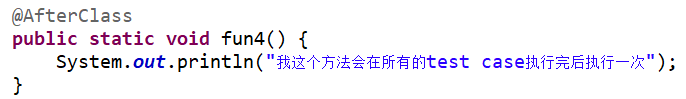
合测试类需要统一初始化处理的逻辑放到这个方法里，比如数据库连接等。如下：



只执行一次；

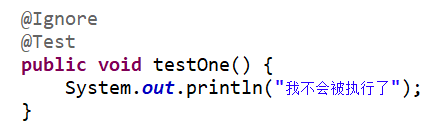
6）@AfterClass注解标注的方法必须是public static void类型，并且无参数。对于整个测

试类需要统一善后处理的逻辑可以放到这个方法里，比如关闭数据库连接等。如下：

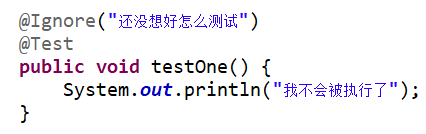


只执行一次；

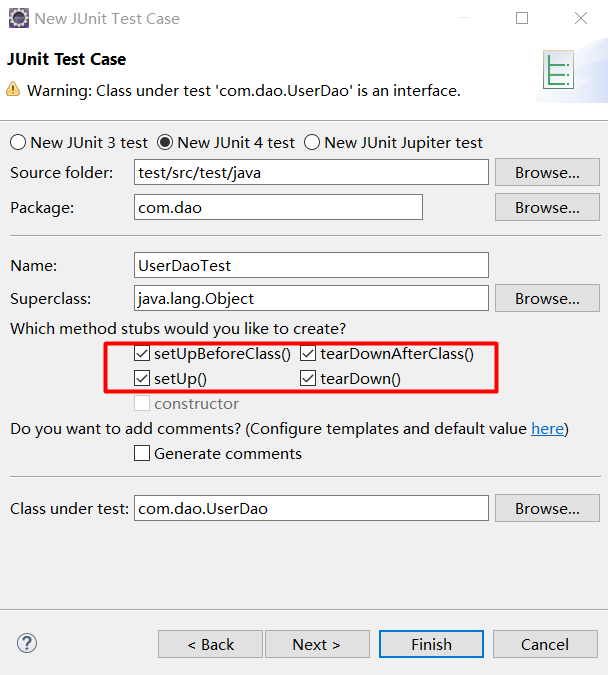
7）@Ignore注解可以标注在方法或类上，被@Ignore标注的test case将不会被执行，被@Ignore标注的类，该类中的所有test case将不会被执行。如标注在方法上：



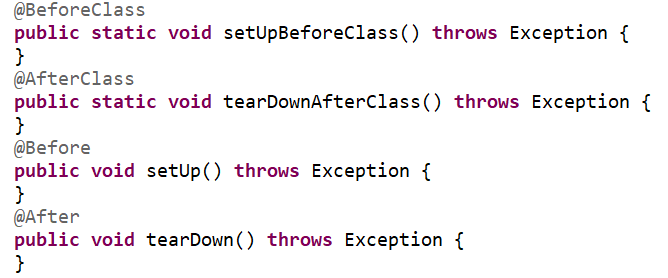
@Ignore有一个可选属性value，用于定义注释，如下：



@Before、@After、@BeforeClass、@AfterClass标注的方法可以在设置单元测试时生成，如下：



生成后如下：



## 小结

1. 单元测试不能证明是对的，只能证明没有错误；
2. 在测试类中，所有被@Test标注的方法都可以被JUnit执行，它们必须是public void类型的方法，且不能带有参数；
3. 测试类不应该与开发的源代码存放在一起，而是应该单独存放在某个目录中，并其包路径与被测试类的包路径一致；
4. 测试类的命名规范为“被测试类的名称+Test”，测试方法的命名规范为“test+被测试方法的名称（驼峰式）”，但不是强制的；

# MD5密码加密

如果不使用Shiro，Spring也自带了MD5加密工具类。使用案例如下：



运行结果如下：

