# 1 注解方式整合Mybatis

基础环境准备：

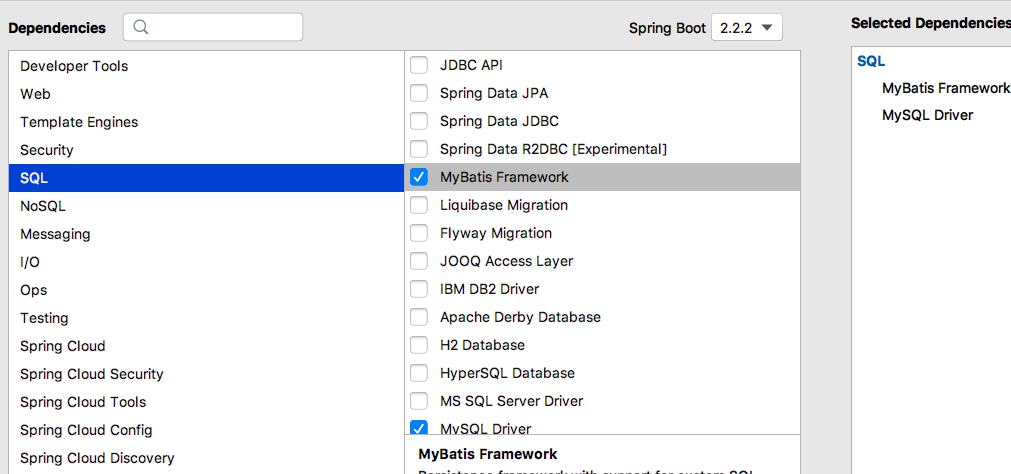
## 1.1 基础环境准备

（1）数据准备

在MySQL中，先创建了一个数据库springbootdata，然后创建了两个表t\_article和t\_comment并向表中插入数据。其中评论表t\_comment的a\_id与文章表t\_article的主键id相关联

|  |
| --- |
| # 创建数据库  CREATE DATABASE springbootdata;  # 选择使用数据库  USE springbootdata;  # 创建表t\_article并插入相关数据  DROP TABLE IF EXISTS t\_article;  CREATE TABLE t\_article (  id int(20) NOT NULL AUTO\_INCREMENT COMMENT '文章id',  title varchar(200) DEFAULT NULL COMMENT '文章标题',  content longtext COMMENT '文章内容',  PRIMARY KEY (id)  ) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;  INSERT INTO t\_article VALUES ('1', 'Spring Boot基础入门', '从入门到精通讲解...');  INSERT INTO t\_article VALUES ('2', 'Spring Cloud基础入门', '从入门到精通讲解...');    # 创建表t\_comment并插入相关数据  DROP TABLE IF EXISTS t\_comment;  CREATE TABLE t\_comment (  id int(20) NOT NULL AUTO\_INCREMENT COMMENT '评论id',  content longtext COMMENT '评论内容',  author varchar(200) DEFAULT NULL COMMENT '评论作者',  a\_id int(20) DEFAULT NULL COMMENT '关联的文章id',  PRIMARY KEY (id)  ) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;  INSERT INTO t\_comment VALUES ('1', '很全、很详细', 'lucy', '1');  INSERT INTO t\_comment VALUES ('2', '赞一个', 'tom', '1');  INSERT INTO t\_comment VALUES ('3', '很详细', 'eric', '1');  INSERT INTO t\_comment VALUES ('4', '很好，非常详细', '张三', '1');  INSERT INTO t\_comment VALUES ('5', '很不错', '李四', '2'); |

（2）创建项目，引入相应的启动器



（3）编写与数据库表t\_comment和t\_article对应的实体类Comment和Article

|  |
| --- |
| public class Comment {  private Integer id;  private String content;  private String author;  private Integer aId;  } |

|  |
| --- |
| public class Article {  private Integer id;  private String title;  private String content;  } |

（4）编写配置文件

|  |
| --- |
| # MySQL数据库连接配置  spring:  datasource:  url: jdbc:mysql://localhost:3306/springbootdata?serverTimezone=UTC&characterEncoding=UTF-8  username: root  password: 123456 |

## 1.2 整合

需求：实现通过ID查询Comment信息

（1）创建一个对t\_comment表数据操作的接口CommentMapper

​  
public interface CommentMapper {  
   @Select("SELECT \* FROM t\_comment WHERE id =#{id}")  
   public Comment findById(Integer id);  
}

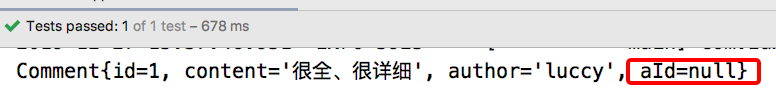
（2）在Spring Boot项目启动类上添加@MapperScan("xxx")注解

@SpringBootApplication  
@MapperScan("com.demo.mapper")  
public class Springboot02MybatisApplication {  
​  
   public static void main(String[] args) {  
       SpringApplication.run(Springboot02MybatisApplication.class, args);  
  }  
​  
}

（3）编写测试方法

@RunWith(SpringRunner.class)  
@SpringBootTest  
class SpringbootPersistenceApplicationTests {  
​  
   @Autowired  
   private CommentMapper commentMapper;  
​  
   @Test  
   void contextLoads() {  
       Comment comment = commentMapper.findById(1);  
       System.out.println(comment);  
  }  
}

打印结果：

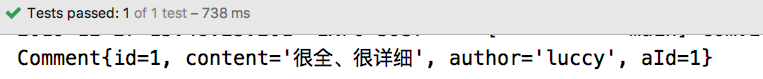


计成了aId属性，所以无法正确映射查询结果。

为了解决上述由于驼峰命名方式造成的表字段值无法正确映射到类属性的情况，可以在Spring Boot全局配置文件application.properties中添加开启驼峰命名匹配映射配置，示例代码如下

|  |
| --- |
| #开启驼峰命名匹配映射  mybatis:  configuration:  map-underscore-to-camel-case: true |

打印结果：



# 2 配置文件方式整合MyBatis

第一、二步骤使用Free Mybatis plugin插件生成



## （1）创建ArticleMapper接口

@Mapper  
public interface ArticleMapper {  
   public Article selectArticle(Integer id);  
}

## （2）创建XML映射文件

resources目录下创建一个统一管理映射文件的包mapper，并在该包下编写与ArticleMapper接口方应的映射文件ArticleMapper.xml

<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
         "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<mapper namespace="com.demo.mapper.ArticleMapper">  
   <select id="selectArticle" resultType="Article">  
      select \* from Article  
   </select>  
</mapper>

## （3）配置XML映射文件路径

在项目中编写的XML映射文件，Spring Boot并无从知晓，所以无法扫描到该自定义编写的XML配置文件，还必须在全局配置文件application.properties中添加MyBatis映射文件路径的配置，同时需要添加实体类别名映射路径，示例代码如下：

mybatis:  
#配置MyBatis的xml配置文件路径  
mapper-locations: classpath:mapper/\*.xml  
#配置XML映射文件中指定的实体类别名路径  
type-aliases-package: com.demo.base.pojo  
​

## （4）编写单元测试进行接口方法测试

   @Autowired  
   private ArticleMapper articleMapper;  
​  
   @Test  
   void contextLoads2() {  
       Article article = articleMapper.selectByPrimaryKey(1);  
       System.out.println(article);  
  }

# 3 Spring Boot整合Redis

## （1）添加Redis依赖包

在项目的pom.xml中添加如下：

<!-- redis依赖包 -->  
<dependency>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-starter-data-redis</artifactId>  
</dependency>

## （2）配置Redis数据库连接

在application.properties中配置redis数据库连接信息，如下：

#redis配置  
#Redis服务器地址  
spring.redis.host=127.0.0.1  
#Redis服务器连接端口  
spring.redis.port=6379  
#Redis数据库索引（默认为0）  
spring.redis.database=0    
#连接池最大连接数（使用负值表示没有限制）  
spring.redis.jedis.pool.max-active=50  
#连接池最大阻塞等待时间（使用负值表示没有限制）  
spring.redis.jedis.pool.max-wait=3000  
#连接池中的最大空闲连接  
spring.redis.jedis.pool.max-idle=20  
#连接池中的最小空闲连接  
spring.redis.jedis.pool.min-idle=2  
#连接超时时间（毫秒）  
spring.redis.timeout=5000

## （3）编写Redis操作工具类

将RedisTemplate实例包装成一个工具类，便于对redis进行数据操作。

package com.demo.utils;  
​  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.stereotype.Component;  
​  
import java.util.concurrent.TimeUnit;  
​  
@Component  
public class RedisUtils {  
​  
   @Autowired  
   private RedisTemplate redisTemplate;  
​  
   /\*\*  
    \* 读取缓存  
    \*  
    \* @param key  
    \* @return  
    \*/  
   public Object get(final String key) {  
       return redisTemplate.opsForValue().get(key);  
  }  
​  
   /\*\*  
    \* 写入缓存  
    \*/  
   public boolean set( String key, Object value) {  
       boolean result = false;  
       try {  
           redisTemplate.opsForValue().set(key, value,1, TimeUnit.DAYS);  
           result = true;  
      } catch (Exception e) {  
           e.printStackTrace();  
      }  
       return result;  
  }  
​  
   /\*\*  
    \* 更新缓存  
    \*/  
   public boolean getAndSet(final String key, String value) {  
       boolean result = false;  
       try {  
           redisTemplate.opsForValue().getAndSet(key, value);  
           result = true;  
      } catch (Exception e) {  
           e.printStackTrace();  
      }  
       return result;  
  }  
​  
   /\*\*  
    \* 删除缓存  
    \*/  
   public boolean delete(final String key) {  
       boolean result = false;  
       try {  
           redisTemplate.delete(key);  
           result = true;  
      } catch (Exception e) {  
           e.printStackTrace();  
      }  
       return result;  
  }  
​  
}  
​

## （4）测试

写一个测试用例类来完成对redis的整合

@RunWith(SpringRunner.class)  
@SpringBootTest  
class Springboot02MybatisApplicationTests {  
​  
   @Autowired  
   private RedisUtils redisUtils;  
​  
​  
   @Autowired  
   private CommentMapper commentMapper;  
​  
​  
    @Test  
   public void setRedisData() {  
      redisUtils.set("article\_1",articleMapper.selectByPrimaryKey(1));  
       System.out.println("success");  
  }  
​  
   @Test  
   public void getRedisData() {  
       Article article = (Article) redisUtils.get("article\_1");  
       System.out.println(article);  
  }  
​