3. SpringBoot数据访问

SpringData是Spring提供的一个用于简化数据库访问、支持云服务的开源框架。它是一个伞形项目，包含了大量关系型数据库及非关系型数据库的数据访问解决方案，其设计目的是使我们可以快速且简单地使用各种数据访问技术。Spring Boot默认采用整合SpringData的方式统一处理数据访问层，通过添加大量自动配置，引入各种数据访问模板xxxTemplate以及统一的Repository接口，从而达到简化数据访问层的操作。

Spring Data提供了多种类型数据库支持，对支持的的数据库进行了整合管理，提供了各种依赖启动器，接下来，通过一张表罗列提供的常见数据库依赖启动器，如表所示。

名称	描述
spring-boot-starter-data-jpa	使用Spring Data JPA与Hibernate
spring-boot-starter-data-mongodb	使用MongoDB和Spring Data MongoDB
spring-boot-starter-data-neo4j	使用Neo4j图数据库和Spring Data Neo4j
spring-boot-starter-data-redis	使用Redis键值数据存储与Spring Data Redis和Jedis客户端

除此之外，还有一些框架技术，Spring Data项目并没有进行统一管理， Spring Boot官方也没有提供对应的依赖启动器，但是为了迎合市场开发需求、这些框架技术开发团队自己适配了对应的依赖启动器，例如，mybatis-spring-boot-starter支持MyBatis的使用

3.1 Spring Boot整合MyBatis

MyBatis 是一款优秀的持久层框架，Spring Boot官方虽然没有对MyBatis进行整合，但是MyBatis团队自行适配了对应的启动器，进一步简化了使用MyBatis进行数据的操作

因为Spring Boot框架开发的便利性，所以实现Spring Boot与数据访问层框架（例如MyBatis）的整合非常简单，主要是引入对应的依赖启动器，并进行数据库相关参数设置即可

基础环境搭建：

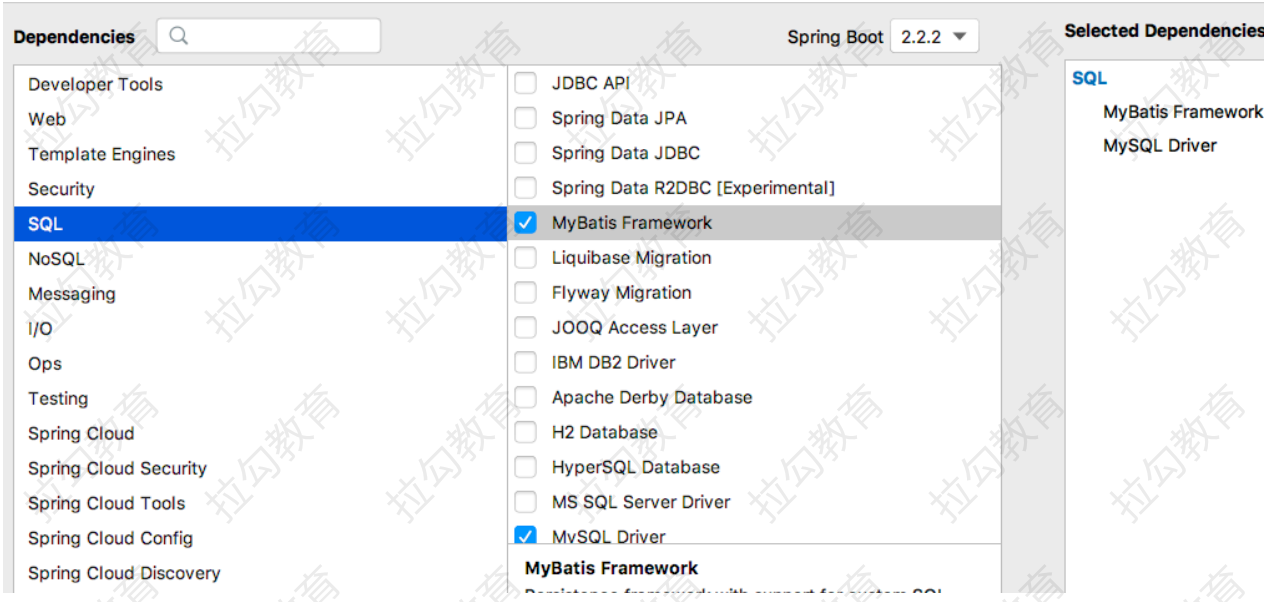
1.数据准备

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', 'luccy', '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;
    // 省略属性getXX()和setXX()方法
    // 省略toString()方法
}
```

```
public class Article {

    private Integer id;
    private String title;
    private String content;
    // 省略属性getXX()和setXX()方法
    // 省略toString()方法
}
```

4.编写配置文件

（1）在application.properties配置文件中进行数据库连接配置

```
# MySQL数据库连接配置
spring.datasource.url=jdbc:mysql://localhost:3306/springbootdata?
serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
```

注解方式整合Mybatis

（1）创建一个用于对数据库表t_comment数据操作的接口CommentMapper

```
@Mapper
public interface CommentMapper {
    @Select("SELECT * FROM t_comment WHERE id =#{id}")
    public Comment findById(Integer id);
}
```

@Mapper注解表示该类是一个MyBatis接口文件，并保证能够被Spring Boot自动扫描到Spring容器中

对应的接口类上添加了@Mapper注解，如果编写的Mapper接口过多时，需要重复为每一个接口文件添加@Mapper注解

为了解决这种麻烦，可以直接在Spring Boot项目启动类上添加@MapperScan("xxx")注解，不需要再逐个添加

@Mapper注解，@MapperScan("xxx")注解的作用和@Mapper注解类似，但是它必须指定需要扫描的具体包名

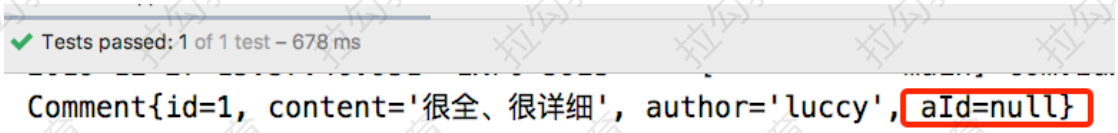
（2）编写测试方法

```
@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：

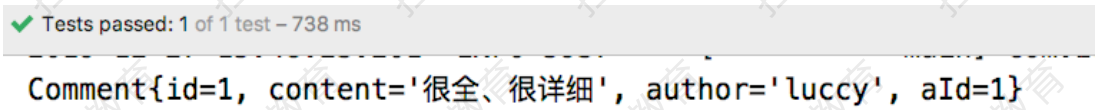


控制台中查询的Comment的aId属性值为null，没有映射成功。这是因为编写的实体类Comment中使用了驼峰命名方式将t_comment表中的a_id字段设计成了aId属性，所以无法正确映射查询结果。

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：



使用配置文件的方式整合MyBatis

（1）创建一个用于对数据库表t_article数据操作的接口ArticleMapper

```
@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.lagou.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

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

（4）编写单元测试进行接口方法测试

```
@Autowired
private ArticleMapper articleMapper;
@Test
public void selectArticle() {
    Article article = articleMapper.selectArticle(1);
    System.out.println(article);
}
```

打印结果：

```
✔ Tests passed: 1 of 1 test – 723 ms
Article{id=1, title='Spring Boot基础入门', content='从入门到精通讲解...'}
```

3.2 Spring Boot整合JPA

（1）添加Spring Data JPA依赖启动器。在项目的pom.xml文件中添加Spring Data JPA依赖启动器，示例代码如下

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

（2）编写ORM实体类。

```
@Entity(name = "t_comment")  // 设置ORM实体类，并指定映射的表名
public class Comment {

    @Id   // 表明映射对应的主键id
    @GeneratedValue(strategy = GenerationType.IDENTITY) // 设置主键自增策略
    private Integer id;
    private String content;
    private String author;

    @Column(name = "a_id")  //指定映射的表字段名
    private Integer aId;
    // 省略属性getXX()和setXX()方法
    // 省略toString()方法
}
```

（3）编写Repository接口： CommentRepository

```
public interface CommentRepository extends JpaRepository<Comment,Integer> {

}
```

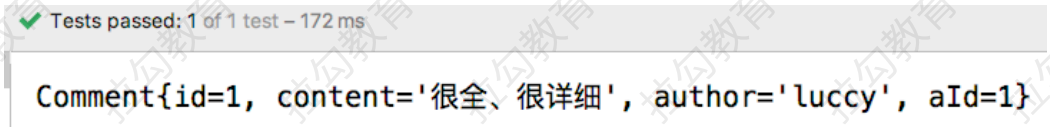
（4）测试

```
    @Autowired
    private CommentRepository repository;

    @Test
    public void selectComment() {
        Optional<Comment> optional = repository.findById(1);
            if(optional.isPresent()){
                System.out.println(optional.get());
            }
           System.out.println();

    }
```

打印：



3.3 Spring Boot整合Redis

除了对关系型数据库的整合支持外，Spring Boot对非关系型数据库也提供了非常好的支持。Spring Boot与非关系型数据库Redis的整合使用

（1）添加Spring Data Redis依赖启动器。先在项目的pom.xml文件中添加Spring Data Redis依赖启动器，示例代码如下

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

（2）编写实体类。此处为了演示Spring Boot与Redis数据库的整合使用，在项目的com.lagou.domain包下编写几个对应的实体类

```
  @RedisHash("persons")  // 指定操作实体类对象在Redis数据库中的存储空间
  public class Person {
    @Id        // 标识实体类主键
    private String id;
    @Indexed  // 标识对应属性在Redis数据库中生成二级索引
    private String firstname;
    @Indexed
    private String lastname;
    private Address address;
    // 省略属性getXX()和setXX()方法
    // 省略有参和无参构造方法
    // 省略toString()方法
  }
```

Address：

```
  public class Address {
    @Indexed
    private String city;
    @Indexed
    private String country;
      // 省略属性getXX()和setXX()方法
      // 省略有参和无参构造方法
      // 省略toString()方法
}
```

实体类示例中，针对面向Redis数据库的数据操作设置了几个主要注解，这几个注解的说明如下：

- @RedisHash("persons")：用于指定操作实体类对象在Redis数据库中的存储空间，此处表示针对Person实体类的数据操作都存储在Redis数据库中名为persons的存储空间下。
- @Id：用于标识实体类主键。在Redis数据库中会默认生成字符串形式的HashKey表示唯一的实体对象id，当然也可以在数据存储时手动指定id。
- @Indexed：用于标识对应属性在Redis数据库中生成二级索引。使用该注解后会在Redis数据库中生成属性对应的二级索引，索引名称就是属性名，可以方便的进行数据条件查询。

（3）编写Repository接口。Spring Boot针对包括Redis在内的一些常用数据库提供了自动化配置，可以通过实现Repository接口简化对数据库中的数据进行增删改查操作

```
public interface PersonRepository extends CrudRepository<Person,String> {
    List<Person> findByAddress_City(String 北京);
}
```

- 需要说明的是，在操作Redis数据库时编写的Repository接口文件需要继承最底层的CrudRepository接口，而不是继承JpaRepository，这是因为JpaRepository是Spring Boot整合JPA特有的。当然，也可以在项目pom.xml文件中同时导入Spring Boot整合的JPA依赖和Redis依赖，这样就可以编写一个继承JpaRepository的接口操作Redis数据库

（4）Redis数据库连接配置。在项目的全局配置文件application.properties中添加Redis数据库的连接配置，示例代码如下

```
# 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服务器连接密码（默认为空）
spring.redis.password=
```

（5）编写单元测试进行接口方法测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class RedisTests {

    @Autowired
    private PersonRepository repository;

    @Test
    public void savePerson() {
        Person person = new Person();
        person.setFirstname("张");
        person.setLastname("三");

        Address address = new Address();
```