第 4-7 课：Spring Boot 简单集成 MongoDB

MongoDB 如今是最流行的 NoSQL 数据库，被广泛应用于各行各业中，很多创业公司数据库选型就直接使用了 MongoDB，但对于大部分公司，使用 MongoDB 的场景是做大规模数据查询和离线分析。MongoDB 一经推出就受到了广大社区的热爱，可以说是对程序员最友好的一种数据库，下面我们来了解一下它的特性。

MongoDB 简介

MongoDB（Humongous，庞大）是可以应用于各种规模的企业、各个行业以及各类应用程序的开源数据库，作为一个适用于敏捷开发的数据库，MongoDB 的数据模式可以随着应用程序的发展而灵活地更新。与此同时，它也为开发人员提供了传统数据库的功能：二级索引、完整的查询系统及严格一致性等。MongoDB 能够使企业更加具有敏捷性和可扩展性，各种规模的企业都可以通过使用 MongoDB 来创建新的应用，来提高与客户之间的工作效率，加快产品上市时间，以及降低企业成本。

MongoDB 是专门为可扩展性、高性能和高可用性而设计的数据库，它可以从单服务器部署扩展到大型、复杂的多数据中心架构。利用内存计算的优势，MongoDB 能够提供高性能的数据读写操作。 MongoDB 的本地复制和自动故障转移功能使应用程序具有企业级的可靠性和操作灵活性。

MongoDB 相关概念

在学习 MongoDB 之前需要先了解一些专业术语，常说 MongoDB 是最像关系数据库的 NoSQL 数据库，我们用关系数据库和 MongoDB 做一下对比，方便更清晰地认识它。

SQL 术语/概念	MongoDB 术语/概念	解释/说明
DataBase	DataBase	数据库
Table	Collection	数据库表/集合
Row	Document	数据记录行/文档
Column	Field	数据字段/域
index	index	索引
Table joins		表连接，MongoDB 不支持
primary key	primary key	主键，MongoDB 自动将 _id 字段设置为主键

MongoDB 和关系数据库一样有库的概念，一个 MongoDB 数据库可以有很多数据库，MongoDB 中的集合就相当于我们关系数据库中的表，文档就相当于关系数据库中的行，域就相当于关系数据库中的列，MongoDB 也支持各种索引有唯一主键，但不支持表连接查询。

Spring Boot MongoDB

Spring Boot 操作数据库的各种 Starters 都继承于 Spring Data，Spring Data 是 Spring 为了简化数据库操作而封装的一个组件包，提供了操作多种数据库的支持，其 API 简洁、调用方便。

spring-boot-starter-data-mongodb 是 Spring Data 的一个子模块。目标是为 MongoDB 提供一个相近的、一致的、基于 Spring 的编程模型。spring-boot-starter-data-mongodb 核心功能是映射 POJO 到 Mongo 的 DBCollection 中的文档，并且提供 Repository 风格数据访问层。

我们使用 Spring Boot 进行 MongoDB 连接，需要使用 spring-boot-starter-data-mongodb 包。spring-boot-starter-data-mongodb 继承 Spring Data 的通用功能外，针对 MongoDB 的特性开发了很多定制的功能，让我们使用 Spring Boot 操作 MongoDB 更加简便。

Spring Boot 操作 MongoDB 有两种比较流行的使用方法，一种是将 MongoTemplate 直接注入到 Dao 中使用，一种是继承 MongoRepository，MongoRepository 内置了很多方法可直接使用。

我们分别来介绍它们的使用。

MongoTemplate 的使用

MongoTemplate 提供了非常多的操作 MongoDB 方法，它是线程安全的，可以在多线程的情况下使用。

MongoTemplate 实现了 MongoOperations 接口, 此接口定义了众多的操作方法如 find、findAndModify、findOne、insert、remove、save、update and updateMulti 等。它转换 domain object 为 DBObject，并提供了 Query、Criteria and Update 等流式 API。

我们后面在使用动态查询时候会用到 DBObject 对象的操作。

（1）pom 包配置

pom 包里面添加 spring-boot-starter-data-mongodb 包引用：

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

（2）在 application.properties 中添加配置

```
spring.data.mongodb.uri=mongodb://name:pass@localhost:27017/test
```

多个 IP 集群可以采用以下配置：

```
spring.data.mongodb.uri=mongodb://user:pwd@ip1:port1,ip2:port2/database
```

当然了密码和用户名如果没有设置可以不用添加，像这样：

```
spring.data.mongodb.uri=mongodb://localhost:27017/test
```

> 注：Mongo 3.0 Java 驱动不支持 spring.data.mongodb.host 和 spring.data.mongodb.port，对于这种情况，spring.data.mongodb.uri 需要提供全部的配置信息。

（3）创建数据实体

```
public class User implements Serializable {
        private static final long serialVersionUID = -3258839839160856613L;
        private Long id;
        private String userName;
        private String passWord;

      //getter、setter 省略
}
```

（4）对实体进行增删改查操作

Repository 层实现了 User 对象的增、删、改、查功能。

首先将 MongoTemplate 注入到实体类中：

```
@Component
public class UserRepositoryImpl implements UserRepository {

    @Autowired
    private MongoTemplate mongoTemplate;

}
```

做增、删、改、查功能时，直接使用即可。

添加数据：

```
@Override
public void saveUser(UserEntity user) {
    mongoTemplate.save(user);
}
```

save 方法中会进行判断，如果对象包含了 ID 信息就会进行更新，如果没有包含 ID 信息就自动保存。

根据用户名查询对象：

```
@Override
public User findUserByUserName(String userName) {
    Query query=new Query(Criteria.where("userName").is(userName));
    User user =  mongoTemplate.findOne(query , User.class);
    return user;
}
```

更新对象比较灵活，可以选择更新一条数据，或者更新多条数据。

```
@Override
public long updateUser(User user) {
    Query query=new Query(Criteria.where("id").is(user.getId()));
    Update update= new Update().set("userName", user.getUserName()).set("passWord"
, user.getPassWord());
    //更新查询返回结果集的第一条
    UpdateResult result =mongoTemplate.updateFirst(query,update,User.class);
    //更新查询返回结果集的所有
    // mongoTemplate.updateMulti(query,update,UserEntity.class);
    if(result!=null)
        return result.getMatchedCount();
    else
        return 0;
}
```

删除对象：

```
@Override
public void deleteUserById(Long id) {
    Query query=new Query(Criteria.where("id").is(id));
    mongoTemplate.remove(query,User.class);
}
```

（5）开发对应的测试方法

测试时注入封装好的 UserRepository，根据情况调用对应的方法即可：

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserRepositoryTest {
    @Autowired
    private UserRepository userRepository;
}
```

测试插入数据：

```
@Test
public void testSaveUser() throws Exception {
    User user=new User();
    user.setId(2l);
    user.setUserName("小明");
    user.setPassWord("fffooo123");
    userRepository.saveUser(user);
}
```

测试查询方法：

```
@Test
public void findUserByUserName(){
    User user= userRepository.findUserByUserName("小明");
   System.out.println("user is "+user);
}
```

测试更新：

```
@Test
public void updateUser(){
    User user=new User();
    user.setId(2l);
    user.setUserName("天空");
    user.setPassWord("fffxxxx");
    userRepository.updateUser(user);
}
```

最后测试删除：

```
@Test
public void deleteUserById(){
    userRepository.deleteUserById(1L);
}
```

（6）查看验证结果

可以使用工具 Robo 3T 来连接后直接图形化展示查看，也可以登录服务器用命令来查看。

a.登录 mongos：

> bin/mongo -host localhost -port 20000

b.切换到 test 库：

> use test

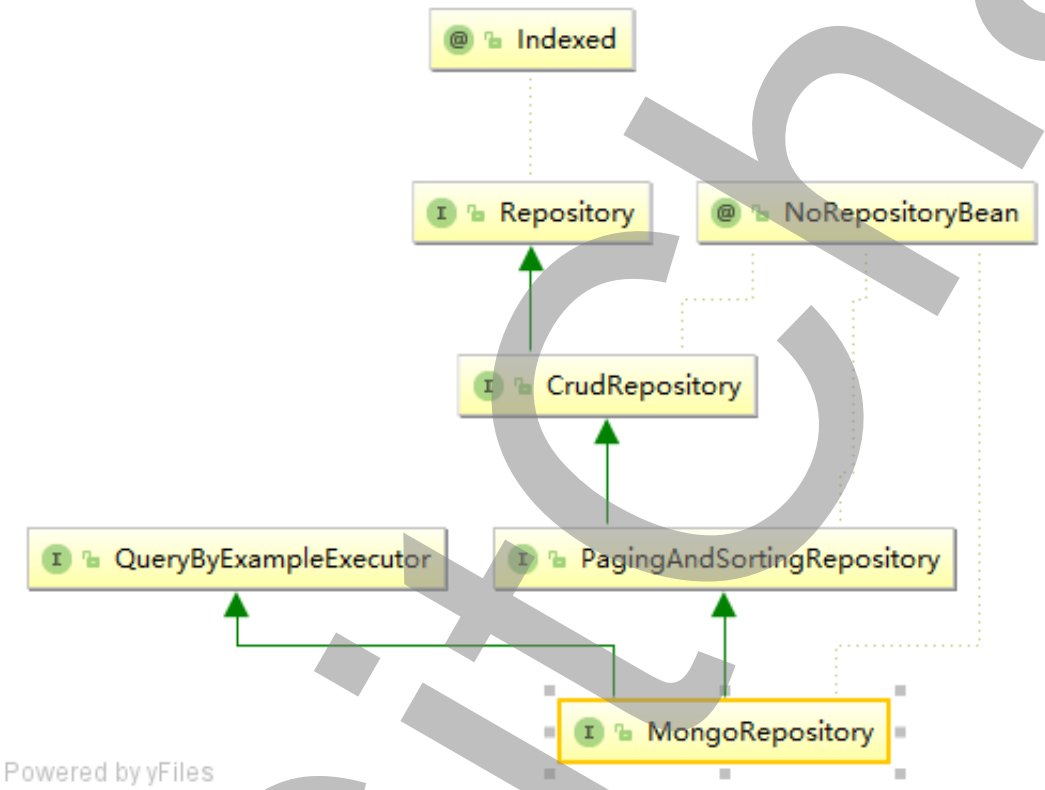
c.查询 userEntity 集合数据：

```
db.user.find()
```

根据 c 查询的结果来观察测试用例的执行是否正确，到此 MongoTemplate 的增、删、改、查功能已经全部实现。

MongoRepository 的使用

MongoRepository 继承于 PagingAndSortingRepository，再往上就是 CrudRepository，根据下面的类图可以看出 MongoRepository 和前面 JPA、Elasticsearch 的使用比较类似，都是 Spring Data 家族的产品，最终使用方法也就和 JPA、ElasticSearch 的使用方式类似，可以根据方法名自动生成 SQL 来查询。



MongoRepository 项目结构和上面类似，只有 Repository 层的代码有所变化，如下：

```
public interface UserRepository extends MongoRepository<UserEntity, Long> {
    UserEntity findByUserName(String userName);
}
```

我们新建 UserRepository 需要继承 MongoRepository，这样就可直接使用 MongoRepository 的内置方法。

使用 UserRepository 进行增、删、改、查功能，首先将 UserRepository 注入到类中：

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserDaoTest {
    @Autowired
    private UserRepository userRepository;
}
```

集成 MongoRepository 会默认实现很多的内置方法，其中就包括保持、删除等操作。

测试保存数据：

```
@Test
public void testSaveUser() throws Exception {
    User user=new User();
    user.setId(i);
    user.setUserName("小明"+i);
    user.setPassWord("fffooo123");
    userRepository.save(user);
}
```

测试根据属性查询：

```
@Test
public void findUserByUserName(){
  User user= userRepository.findByUserName("小明");
   System.out.println("user is "+user);
}
```

测试更新属性，保存和更新都使用的是 save() 方法，内部会根据实体是否有注解来判断是用来更新还是用来保存。

```
@Test
public void updateUser(){
    User user=new User();
    user.setId(2l);
    user.setUserName("天空");
    user.setPassWord("fffxxxx");
    userRepository.save(user);
}
```

测试删除数据：

```
@Test
public void deleteUserById(){
    userRepository.delete(1l);
}
```

我们发现 UserRepository 已经自动实现了常用的数据库操作，不用像使用 MongoTemplate 那样需要手动来实现。

如果想使用分页怎么办？只需要在 UserRepository 再加入一个空方法，不需要自己实现。

```
Page<UserEntity> findAll(Pageable var1);
```

首先使用 save 方法插入 100 个 UserEntity：

```
@Test
public void testSaveUser() throws Exception {
    for (long i=0;i<100;i++) {
        User user=new User();
        user.setId(i);
        user.setUserName("小明"+i);
        user.setPassWord("fffooo123");
        userRepository.save(user);
    }
}
```

返回测试分页，设置以 ID 的倒叙排序，每页 10 条数据，取第二页的用户列表。

```
@Test
public void testPage(){
    Sort sort = new Sort(Sort.Direction.DESC, "id");
    Pageable pageable = PageRequest.of(2, 10, sort);
    Page page=userRepository.findAll(pageable);
    System.out.println("users: "+page.getContent().toString());
}
```

- Pageable 是 Spring Data 库中定义的一个接口，该接口是所有分页相关信息的一个抽象，通过该接口，可以得到和分页相关所有信息（如 pageNumber、pageSize 等），这样，JPA 就能够通过 pageable 参数来得到一个带分页信息的 SQL 语句。
- Page 类也是 Spring Data 提供的一个接口，该接口表示一部分数据的集合以及其相关的下一部分数据、数据总数等相关信息，通过该接口，可以得到数据的总体信息（数据总数、总页数...）以及当前数据的信息（当前数据的集合、当前页数等）。

在 Spring Boot 2.0 中 new PageRequest(2, 10, sort) 已经过期不再推荐使用，2.0 中推荐使用 PageRequest.of(2, 10, sort) 来做排序参数控制。

我们发现使用 Repository 的方式和前几课介绍的 Spring Boot JPA 非常相似，其实 spring-boot-starter-data-mongodb 和 Spring-boot-starter-data-jpa 都来自于 Spring Data 大家族，它们的实现原理基本一致，因此使用 Repository 完全可以参考前面课程 JPA 用法。

多数据源 MongoDB 的使用

我们来学习在多数据源的情况下，如何使用 MongoDB。

（1）添加配置

首先在配置文件中添加不同的数据源，配置文件添加两条数据源，如下：

```
mongodb.primary.uri=mongodb://192.168.0.1:27017
mongodb.primary.database=primary
mongodb.secondary.uri=mongodb://192.168.0.1:27017
mongodb.secondary.database=secondary
```

如果使用的是 MongoDB 集群可以这样配置：

```
mongodb.xxx.uri=mongodb://192.168.0.1:27017,192.168.0.2:27017,192.168.0.3:27017
mongodb.xxx.database=xxx
```

（2）配置数据源

封装读取以 MongoDB 开头的两个配置文件：

```
@Data
@ConfigurationProperties(prefix = "mongodb")
public class MultipleMongoProperties {

    private MongoProperties primary = new MongoProperties();
    private MongoProperties secondary = new MongoProperties();
    //省略 getter、setter
}
```

创建 MultipleMongoConfig 类分别创建两个数据源的 MongoTemplate。

```
@Configuration
public class MultipleMongoConfig {
    @Autowired
    private MultipleMongoProperties mongoProperties;
}
```

根据配置文件信息构建第一个数据源的 MongoDbFactory。

```
@Bean
@Primary
public MongoDbFactory primaryFactory(MongoProperties mongo) throws Exception {
    MongoClient client = new MongoClient(new MongoClientURI(mongoProperties.getPri
mary().getUri()));
    return new SimpleMongoDbFactory(client, mongoProperties.getPrimary().getDataba
se());
}
```

利用上面构建好的 MongoDbFactory 创建对应的 MongoTemplate 。

```
@Primary
@Bean(name = "primaryMongoTemplate")
public MongoTemplate primaryMongoTemplate() throws Exception {
    return new MongoTemplate(primaryFactory(this.mongoProperties.getPrimary()));
}
```

最后将 MongoTemplate 信息注入到对应的包路径下：

```
@Configuration
@EnableConfigurationProperties(MultipleMongoProperties.class)
@EnableMongoRepositories(basePackages = "com.neo.repository.primary",
        mongoTemplateRef = "primaryMongoTemplate")
public class PrimaryMongoConfig {
}
```

以上第一个数据源就配置完成了，第二个数据源的配置过程类似，大家可以直接看课程演示项目。总结一下封装过程：读取配置信息封装为 Properties，根据 Properties 信息构建 Factory，再由 Factory 构建出最后需要使用的 MongoTemplate，MongoTemplate 在根据包路径配置注入到对应的包下。

（3）创建两个库分别对应的对象和 Repository

```
public class User implements Serializable {
        private static final long serialVersionUID = -3258839839160856613L;
        private String  id;
        private String userName;
        private String passWord;
        //省略getter setter
}
```

对应的 Repository：

```
public interface PrimaryRepository extends MongoRepository<User, String> {
}
```

继承了 MongoRepository 会默认实现很多基本的增、删、改、查功能，省了很多自己写 Dao 层的代码，Secondary 和上面的代码类似就不贴出来了。

（4）最后测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class MuliDatabaseTest {
    @Autowired
    private PrimaryRepository primaryRepository;
    @Autowired
    private SecondaryRepository secondaryRepository;

    @Test
    public void TestSave() {
        this.primaryRepository.save(new User("小张", "123456"));
        this.secondaryRepository.save(new User("小王", "654321"));
        List<User> primaries = this.primaryRepository.findAll();
        for (User primary : primaries) {
            System.out.println(primary.toString());
        }
        List<User> secondaries = this.secondaryRepository.findAll();
        for (User secondary : secondaries) {
            System.out.println(secondary.toString());
        }
    }
}
```

输出内容如下：

```
com.neo.entity.UserEntity@2a3a299[id=5a0ed2d7439f6618ac294ba6,userName=小张,passWord=123456]
com.neo.entity.UserEntity@5b989dc7[id=5a0ed2d7439f6618ac294ba7,userName=小王,passWord=654321]
```

使用 Robo 3T 分别查看两个库的数据，均已经有对应的 User 对象，说明多数据源使用测试成功。

总结

两种方式都可以方便地操作 MongoDB 数据库，MongoTemplate 模式比较灵活可以根据自己的使用情况进行组装，MongoRepository 的使用方式更简单，因为继承了 Spring Data，所以默认实现了很多基础的增、删、改、查功能，业务直接拿来使用即可。简单日常的使用推荐 MongoRepository，简单方便利于项目快速开发，复杂多变的查询推荐使用 MongoTemplate 自行进行封装。

> 点击这里下载源码。