第5章 Spring Boot数据访问

《Spring Boot企业级开发教程（第2版）》

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

学习目标/Target

熟悉Spring Data概述，能够说出Spring Data的项目结构和Spring Data的常用核心子接口

了解Spring Data JPA概述，能够说出使用Spring Data JPA进行数据访问的逻辑

掌握Spring Data JPA快速入门，能够根据方法命名规则定义的方法、JPQL，以及原生SQL的方式操作数据库中的数据

掌握Spring Boot整合Spring Data JPA，能够整合Spring Boot和Spring Data JPA，并使用Spring Data JPA进行基本的增删改查

了解MyBatis-Plus概述，能够说出MyBatis-Plus的特性

学习目标/Target

掌握MyBatis-Plus快速入门，能够使用通用Mapper、通用Service，以及条件构造器操作数据库中的数据

掌握Spring Boot整合MyBatis-Plus，能够整合Spring Boot和MyBatis-Plus，并使用MyBatis-Plus进行基本的增删改查

熟悉Redis快速入门，能够说出Redis的概念和优点、安装和启动Redis的方法、以及说出Redis支持的数据类型

掌握Spring Data Redis快速入门，能够说出Spring Data Redis的特性，以及应用Spring Data Redis的常见操作

掌握Spring Boot整合Redis，能够整合Spring Boot和Redis，并使用Spring Data Redis向Redis中存储和读取数据

章节概述/ Summary

一般情况下，应用程序中的数据都会使用数据库进行存储和管理，然后在应用程序中对数据库中的数据进行访问操作。Spring Boot在简化项目开发以及实现自动化配置的基础上，对常见的数据层解决方案提供了非常好的支持，用户只需要进行简单的配置，就可以使用数据库访问技术对数据库进行数据访问。下面将对Spring Boot项目中访问数据库的常用技术进行讲解。

目录/Contents

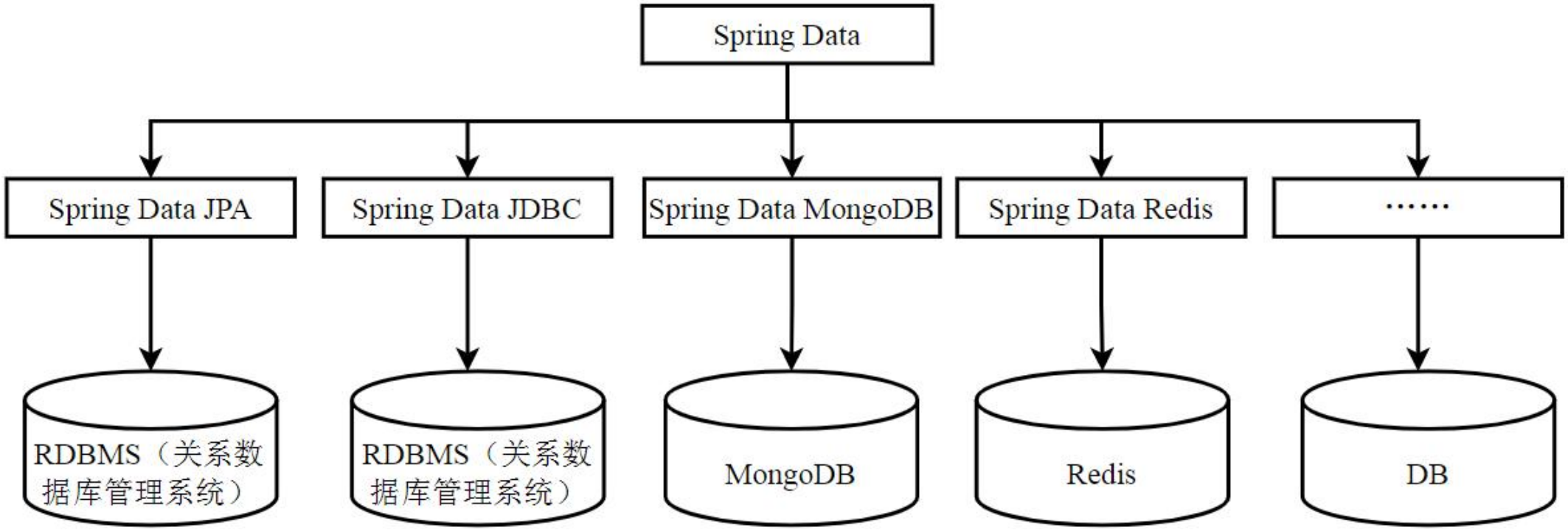
5.3 文件上传

5.1 Spring Data概述

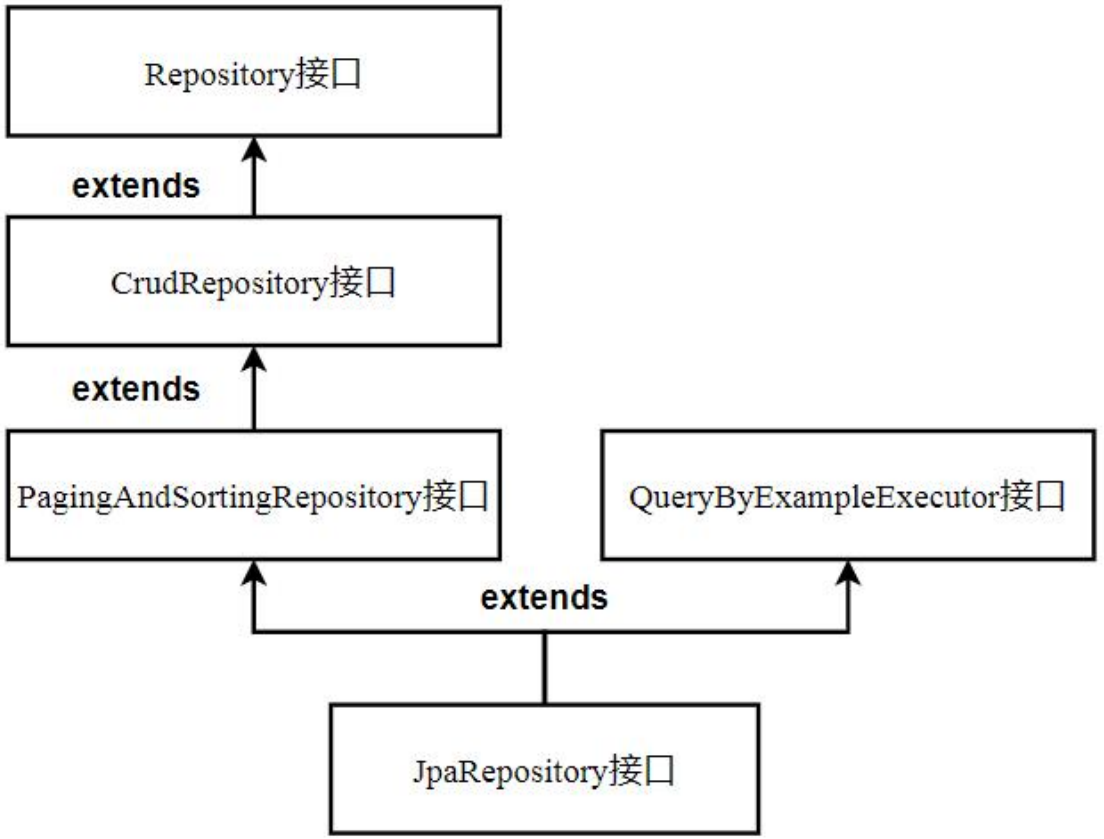
熟悉Spring Data概述，能够说出Spring Data的项目结构和Spring Data的常用核心子接口

Spring Boot默认采用整合Spring Data的方式统一处理数据访问层，Spring Data是Spring提供的开源框架，旨在统一和简化对各种类型数据库的持久化存储。Spring Data为大量的关系型数据库和非关系型数据库提供了数据访问的方案，并且提供了基于Spring的数据访问模型，同时保留了各存储系统的特殊性。

Spring Data为开发者提供了一套统一的API，开发者也可以相同的一套API操作不同存储系统中的数据，保持代码结构的一致性，大大减少开发工作量，提高开发效率。



Spring Data提供了一套统一的Repository接口实现方式，包括基本的增删改查操作、条件查询、排序查询、分页查询等。当数据访问对象需要进行增删改查、排序查询和分页查询等操作时，开发者仅需要按照一定的规范声明接口即可，不需要实现具体的查询方法。Spring Data会根据底层数据存储系统，在运行时自动实现真正的查询方法，执行查询操作，返回结果数据集。



- Repository接口：Repository是一个空接口，用于标示。
- CrudRepository接口：提供了各种增删改查方法。
- PagingAndSortingRepository接口：增加了分页和排序功能的方法。
- QueryByExampleExecutor接口：是进行条件封装查询的顶级父接口，允许通过Example实例执行复杂条件查询。
- JpaRepository接口：重写了一些查找和删除的方法。

5.2

Spring Boot整合Spring Data JPA

Spring Data作为Spring全家桶中重要的一员，在Spring项目全球使用市场份额排名中多次居前位，而在Spring Data子项目的使用份额排名中，Spring Data JPA也一直名列前茅。Spring Boot为Spring Data JPA提供了启动器，使Spring Data JPA在Spring Boot项目中的使用更加便利。下面将对Spring Data JPA的相关知识，以及Spring Boot整合Spring Data JPA进行讲解。

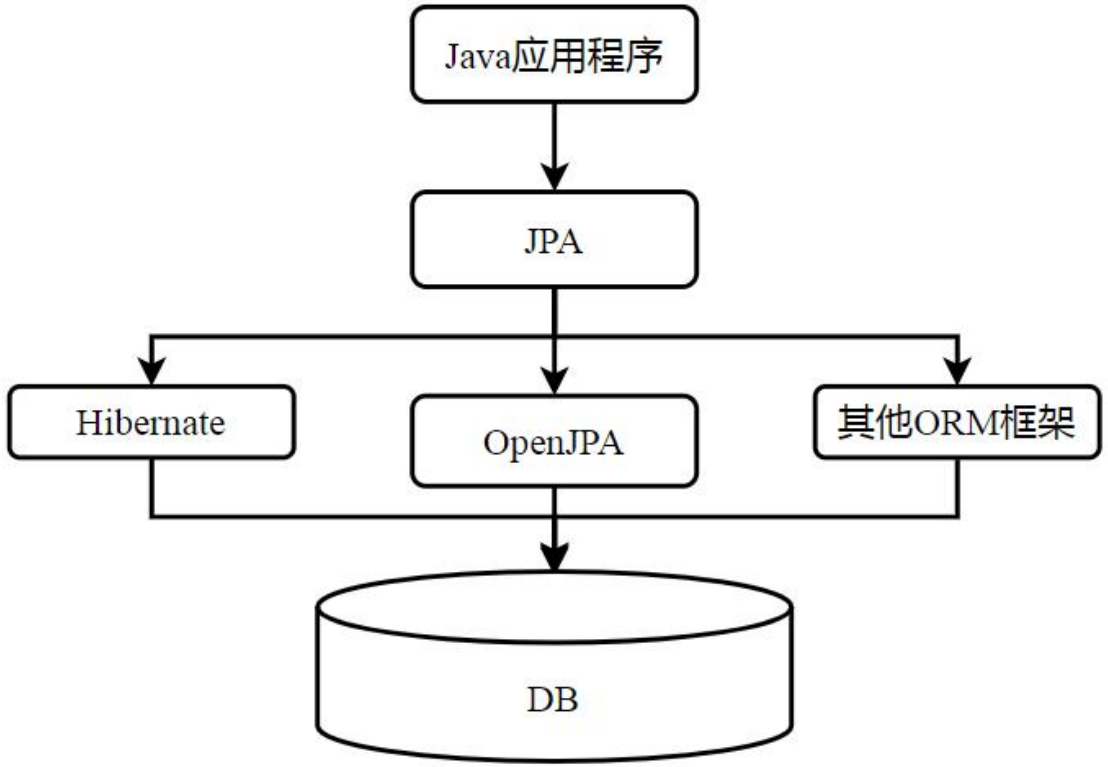
5.2.1 Spring Data JPA概述

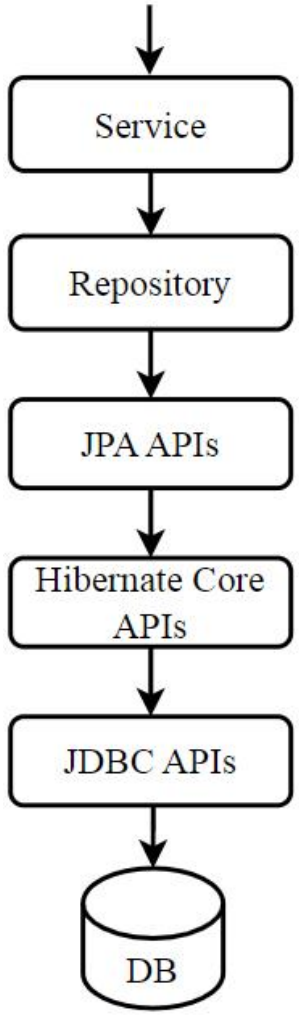
了解Spring Data JPA概述，能够说出使用Spring Data JPA进行数据访问的逻辑

5.2.1 Spring Data JPA概述

对象关系映射（Object Relational Mapping，ORM）框架在运行时可以参照映射文件的信息，把对象持久化到数据库中，可以解决面向对象与关系数据库存在的互不匹配的现象，常见的ORM框架有Hibernate、OpenJPA等。ORM框架的出现，使开发者从数据库编程中解脱出来，把更多的精力放在业务模型与业务逻辑上，但各ORM框架之间的API差别很大，使用了某种ORM框架的系统会严重受限于该ORM的标准，基于此，SUN公司提出JPA（Java Persistence API，Java持久化API）。

JPA是Sun官方提出的Java持久化规范，用于描述对象和关系表的映射关系，并将运行期的实体对象持久化到数据库中。JPA规范本质上是一套规范，它提供了一些编程的API接口，但具体实现则由服务厂商来提供基于JPA的数据访问。





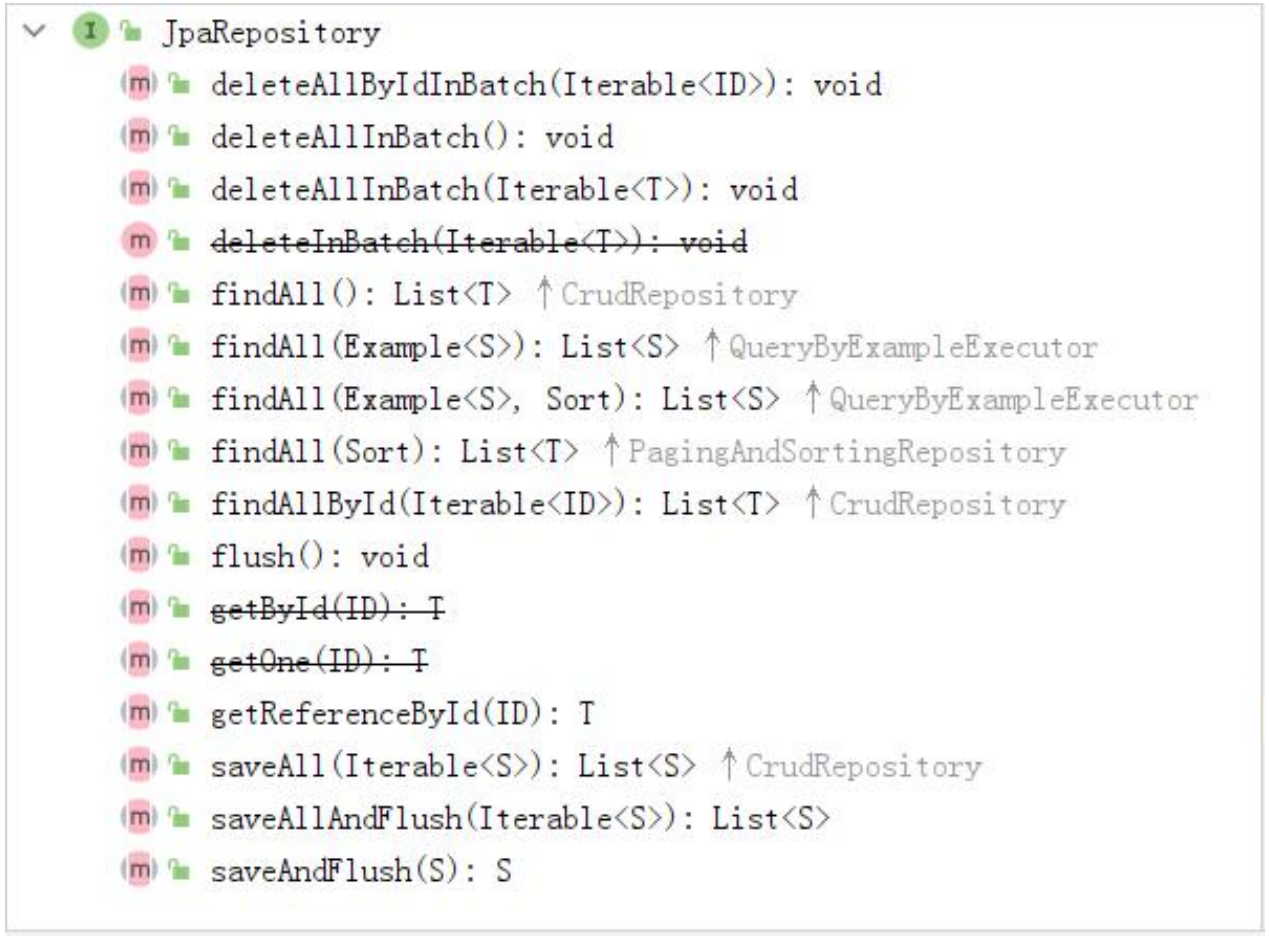
Spring Data JPA整体处理逻辑

掌握Spring Data JPA快速入门，能够根据方法命名规则定义的方法、JPQL，以及原生SQL的方式操作数据库中的数据

Spring Data JPA提供了很多模板代码，易于扩展，可以大幅提高开发效率，使开发者用极简的代码即可实现对数据的访问。使用Spring Data JPA可以通过Repository接口中的方法对数据库中的数据进行增删改查，也可以根据方法命名规则定义的方法、JPQL，以及原生SQL的方式进行操作，下面对Spring Data JPA提供的这些基本功能进行讲解。

1.父接口的方法

如果自定义接口继承了JpaRepository接口，则可以直接使用JpaRepository接口提供的方法。



2.根据方法命名规则定义方法

Spring Data中按照框架的规范自定义了Repository接口，除了可以使用接口提供的默认方法外，还可以按特定规则来定义查询方法，只要这些查询方法的方法名遵守特定的规则，不需要提供方法实现体，Spring Data就会自动为这些方法生成查询语句。Spring Data对这种特定的查询方法的定义规范如下。

- 以find、read、get、query、count开头。
- 涉及查询条件时，条件的属性使用条件关键字连接，并且条件属性的首字母大写。

2.根据方法命名规则定义方法

根据方法命名规范查询时，有时候需要使用关键字对查询条件的属性进行连接。

根据方法命名规则定义方法所支持的关键字

关键字	方法名示例	对应的JPQL片段
And	findByLastnameAndFirstname()	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname()	… where x.lastname = ?1 or x.firstname = ?2
Is，Equals	findByFirstname， findByFirstnameIs， findByFirstnameEquals()	… where x.firstname = ?1
Between	findByStartDateBetween()	… where x.startDate between ?1 and ?2

2.根据方法命名规则定义方法

根据方法命名规范查询时，有时候需要使用关键字对查询条件的属性进行连接。

根据方法命名规则定义方法所支持的关键字

关键字	方法名示例	对应的JPQL片段
And	findByLastnameAndFirstname()	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname()	… where x.lastname = ?1 or x.firstname = ?2
Is，Equals	findByFirstname， findByFirstnameIs， findByFirstnameEquals()	… where x.firstname = ?1
Between	findByStartDateBetween()	… where x.startDate between ?1 and ?2

2.根据方法命名规则定义方法

根据方法命名规则定义方法所支持的关键字

关键字	方法名示例	对应的JPQL片段
LessThan	findByAgeLessThan()	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual()	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan()	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual()	… where x.age >= ?1
After	findByStartDateAfter()	… where x.startDate > ?1
Before	findByStartDateBefore()	… where x.startDate < ?1
IsNull	findByAgeIsNull()	… where x.age is null
IsNotNull	findByAgeIsNotNull()	… where x.ageisnot null
NotNull	findByAgeNotNull()	… where x.age not null

2.根据方法命名规则定义方法

根据方法命名规则定义方法所支持的关键字

关键字	方法名示例	对应的JPQL片段
Like	findByFirstnameLike()	… where x.firstname like ?1
NotLike	findByFirstnameNotLike()	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith()	… where x.firstname like ?1 (绑定参数 %)
EndingWith	findByFirstnameEndingWith()	… where x.firstname like ?1 (绑定参数 %)
Containing	findByFirstnameContaining()	… where x.firstname like ?1 (绑定参数 %)
OrderBy	findByAgeOrderByLastnameDesc()	… where x.age = ?1 order by x.lastname desc

2.根据方法命名规则定义方法

根据方法命名规则定义方法所支持的关键字

关键字	方法名示例	对应的JPQL片段
Not	findByLastnameNot()	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByFirstnameIgnoreCase()	... where UPPER(x.firstame) = UPPER(?1)

3.JPQL

使用Spring Data JPA提供的查询方法已经可以满足大部分应用场景的需求，但是有些业务需要更灵活的查询条件，这时就可以使用@Query注解，结合JPQL的方式来完成查询。

JPQL是JPA中定义的一种查询语言，此种语言旨在让开发者忽略数据库表和表中的字段，而关注实体类及实体类中的属性。 JPQL语句的写法和SQL语句的写法十分类似，但是要把查询的表名换成实体类名称，把表中的字段名换成实体类的属性名称。

3.JPQL

JPQL支持命名参数和位置参数两种查询参数。

- 命名参数：在方法的参数列表中，使用@Param注解标注参数的名称，在@Query注解的查询语句中，使用“:参数名称” 匹配参数名称。
- 位置参数：在@Query注解的查询语句中，使用“?位置编号的数值” 匹配参数，查询语句中参数标注的编号需要和方法的参数列表中参数的顺序依次对应。

```
//命名参数绑定
@Query("from Book b where b.author=:author and b.name=:name")
List<Book> findByCondition1(@Param("author") String author,
 @Param("name") String name);
//位置参数绑定
@Query("from Book b where b.author=?1 and b.name=?2")
List<Book> findByCondition2(String author, String name);
```

3.JPQL

JPQL中使用like模糊查询、排序查询、分页查询子句时，其用法与SQL中的用法相同，区别在于JPQL处理的类的实例不同。

```
//like模糊查询
@Query("from Book b where b.name like %:name%")
 List<Book> findByCondition3(@Param("name") String name);
//排序查询
 @Query("from Book b where b.name like %:name% order by id desc")
List<Book> findByCondition4(@Param("name") String name);
//分页查询
@Query("from Book b where b.name like %:name%")
Page<Book> findByCondition5(Pageable pageable, @Param("name") String name);
```

3.JPQL

JPQL中除了可以使用字符串和基本数据类型的数据作为参数外，还可以使用集合和Bean作为参数，传入Bean进行查询时可以在JPQL中使用SpEL表达式接收变量。

```
//传入集合参数查询
@Query("from Book b where b.id in :ids")
List<Book> findByCondition6(@Param("ids") Collection<String> ids);
//传入Bean进行查询（使用SPEL表达式）
@Query("from Book b where b.author=:#{#Book.author} and " +
      " b.name=:#{#Book.name}")
Book findByCondition7(@Param("Book") Book Book);
```

4.原生SQL

如果出现非常复杂的业务情况，导致JPQL和其他查询都无法实现对应的查询，需要自定义SQL进行查询时，可以在@Query注解中定义该SQL。@Query注解中定义的是原生SQL时，需要在注解使用nativeQuery=true指定执行的查询语句为原生SQL，否则会将其当作JPQL执行

```
@Query(value="SELECT * FROM book WHERE id = :id",nativeQuery=true)
Book findByCondition8(@Param("id") Integer id);
```

小提示

使用@Query注解可以执行JPQL和原生SQL查询，但是@Query注解无法进行DML数据操纵语言,主要语句有INSERT、DELETE和UPDATE操作，如果需要更新数据库中的数据，需要在对应的方法上标注@Modifying注解，以通知Spring Data当前需要进行的是DML操作。需要注意的是JPQL只支持DELETE和UPDATE操作，不支持INSERT操作。

5.2.3 整合 Spring Data JPA

掌握Spring Boot整合Spring Data JPA，能够整合Spring Boot和Spring Data JPA，并使用Spring Data JPA进行基本的增删改查

下面使用Spring Boot和Spring Data JPA进行整合，进一步演示Spring Data JPA在Spring Boot项目中的基本使用。

1.创建项目

在IDEA中创建Spring Boot项目chapter05，读者可以根据自己当前情况选择使用Spring Initializr方式或者Maven方式进行创建，这里使用Maven方式创建项目。

2.配置依赖

在项目chapter05的pom.xml文件中配置Spring Boot整合 Spring Data JPA的依赖，包括Spring Boot父工程的依赖、MySQL驱动依赖和Spring Data JPA的启动器依赖，具体如文件5-1所示。

源 代 码

文件5-1

pom.xml

3.设置配置信息

使用Spring Data JPA需要操作数据库，所以需要在项目中设置一些和数据库连接相关的配置信息。

Spring Boot的自动装配提供了数据库连接的一些默认配置，例如数据源。Spring Boot 2.x 版本默认使用HikariCP作为数据源，如果没有显示指定使用其他数据源，项目启动后会自动使用HikariCP数据源获取数据库连接。

引入Spring Data JPA的启动器后，Spring Boot会自动装配对于JPA的默认配置，例如是否打印运行时的SQL语句和参数信息、是否根据实体自动建表等。

本项目选择采用默认的数据源，JPA的配置信息只修改打印运行时的SQL语句，其他都采用默认的配置信息。

3.设置配置信息

在项目的resources目录下创建application.yml文件，在该文件中指定数据库连接信息和JPA的配置信息，具体如文件5-2所示。

源 代 码

文件5-2

application.yml

4.创建实体类

通过JPA可以简单高效地管理Java实体类和关系数据库的映射，通常每个实体类与数据库中的单个数据表相关联，每个实体的实例表示数据库表格中的某一行记录。在项目的java目录下创建包com.itheima.chapter05.entity，并在该包下创建实体类Book，具体如文件5-3所示。

源 代 码

文件5-3

Book.java

5.自定义Repository接口

通过JPA可以简单高效的管理Java实体类和关系数据库的映射，通常每个实体类与数据库中的单个数

在java目录下创建包com.itheima.chapter05.dao，在该包下自定义接口BookRepository继承JpaRepository接口，并在BookRepository接口中添加操作数据库中图书信息的方法，具体如文件5-4所示。

源 代 码

文件5-4

BookRepository.java

6.创建数据库

至此，实体类和操作实体的接口都已经定义好，因为没有在JPA的配置信息中设置项目启动时根据实体类自动创建数据表，此时测试对图书信息的操作需要先创建对应的数据库和数据表，并插入对应的测试数据，相关SQL如文件5-5所示。

源 代 码

文件5-5

book.sql

7.创建项目启动类和测试类

在src\main\java目录的com.itheima.chapter05包下创建项目启动类Chapter05Application，具体如文件5-6所示。

源 代 码

文件5-6

Chapter05Application.java

7.创建项目启动类和测试类

在src\test\java目录下创建包com.itheima.chapter05，并在该包下创建测试类Chapter05ApplicationTests。在Chapter05ApplicationTests测试类中定义操作图书信息的测试方法，具体如文件5-7所示。

源 代 码

文件5-7

Chapter05ApplicationTests.java

8.测试操作图书信息

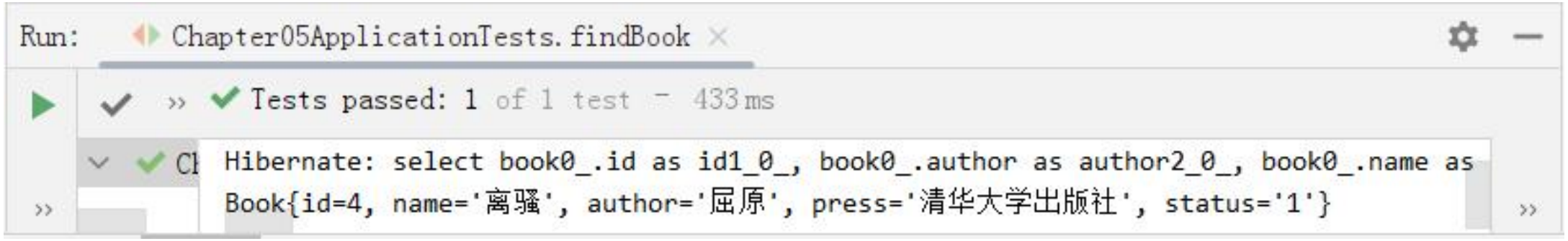
运行文件5-7中的saveBook()方法，测试新增图书信息。

8.测试操作图书信息

运行文件5-7中的editBook()方法，测试修改图书信息。

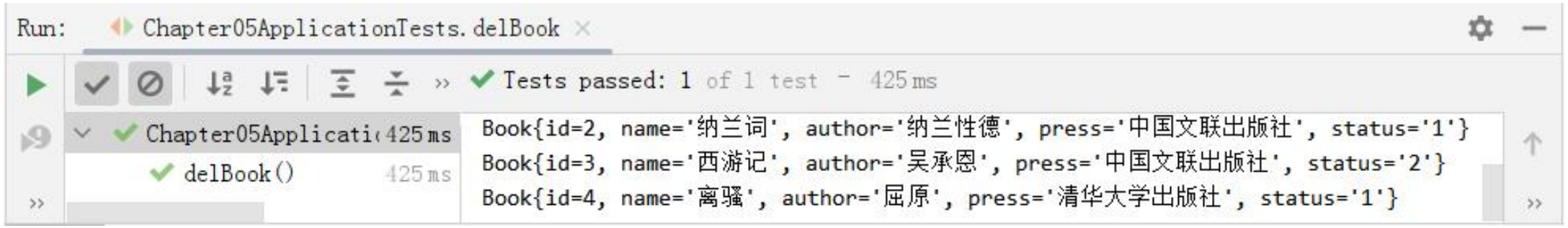
8.测试操作图书信息

运行文件5-7中的findBook()方法，测试查询图书信息。



8.测试操作图书信息

运行文件5-7中的delBook()方法，测试删除图书信息。



5.3

Spring Boot整合MyBatis-Plus

MyBatis是一个足够灵活的 DAO层解决方案，但其也存在一些不足。为了弥补这些不足，可以对MyBatis现有的功能进行增强，Mybatis-Plus就是这样的MyBatis增强工具。下面将对Mybatis-Plus的相关基础知识和Spring Boot整合Mybatis-Plus进行讲解。

了解MyBatis-Plus概述，能够说出MyBatis-Plus的特性

5.3.1 MyBatis-Plus概述

MyBatis是半自动化的ORM实现，支持定制化SQL、存储过程和高级映射，其封装性低于 Hibernate，但性能优秀、小巧、简单易学，在国内备受开发人员的喜爱。MyBatis本身也存在些许不足，例如，配置文件繁多，以及当编写一个业务逻辑时需要在Dao层写一个方法，再创建一个与之对应的映射文件或SQL语句。针对MyBatis的这些不足，MyBatis-Plus诞生了。

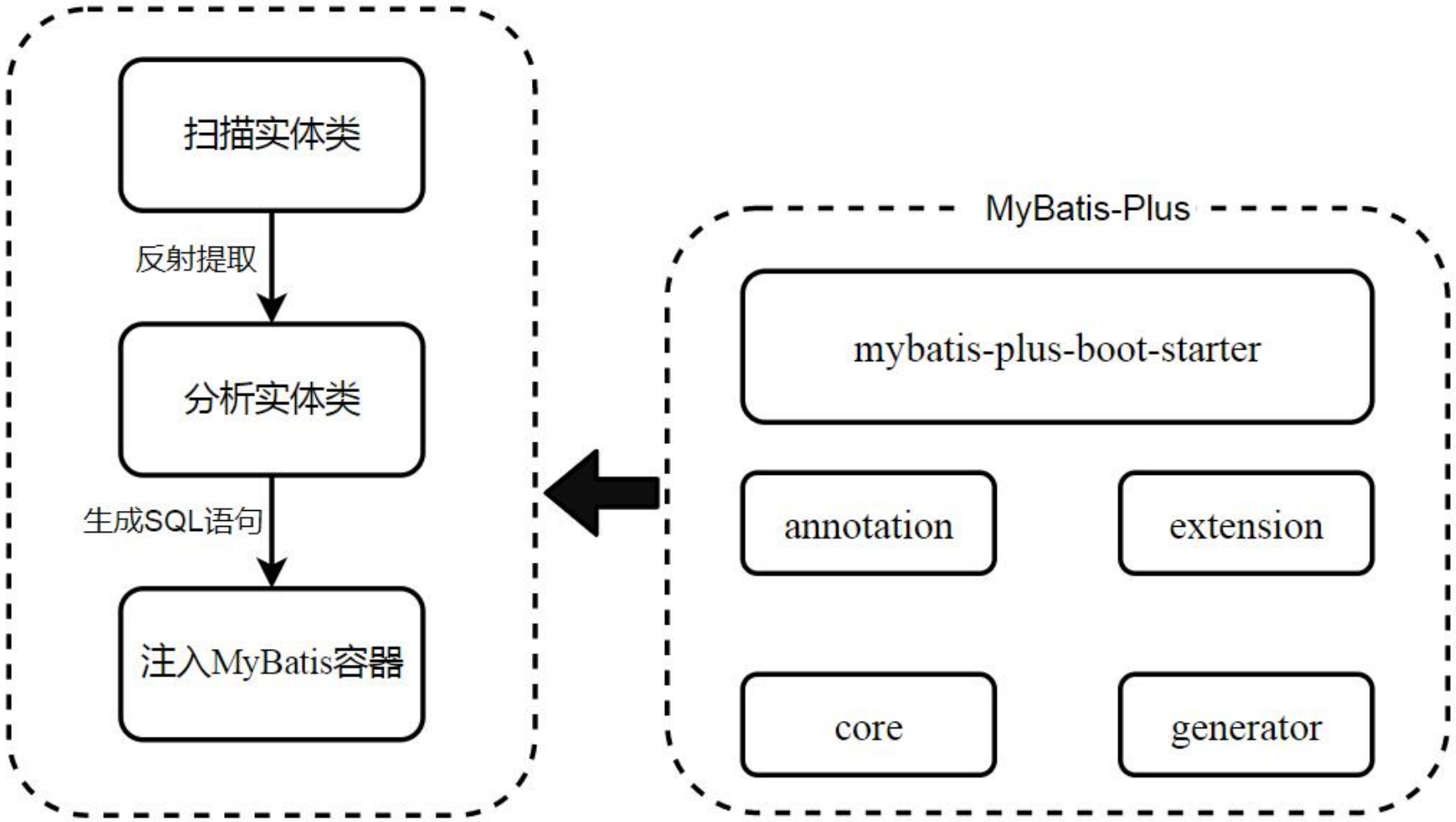
MyBatis-Plus是MyBatis 的增强工具，在MyBatis的基础上只做增强不做改变，支持MyBatis所有原生的特性，旨在简化开发、提高效率。

5.3.1 MyBatis-Plus概述

MyBatis-Plus除了弥补了MyBatis的些许不足外，还具有以下特性。

- 无侵入：只做增强不做改变，引入它不会对现有工程产生影响。
- 损耗小：启动即会自动注入基本增删改查方法，性能基本无损耗，直接面向对象操作。
- 强大的增删改查 操作：内置通用Mapper、通用Service，仅仅通过少量配置即可实现单表大部分增删改查操作，具有强大的条件构造器，可满足各类使用需求。
- 支持Lambda形式调用：通过Lambda表达式，可方便地编写各类查询条件，无须再担心字段写错。
- 支持主键自动生成：支持分布式唯一ID生成器Sequence在内的4种主键策略，可自由配置，完美解决了主键问题。
- 支持ActiveRecord模式：支持ActiveRecord形式调用，实体类只需继承 Model 类即可进行强大的增删改查 操作。

- 支持自定义全局通用操作：支持全局通用方法注入。
- 内置代码生成器：采用代码或者Maven 插件可快速生成Mapper 、Model、 Service、 Controller层代码，支持模板引擎。
- 内置分页插件：基于MyBatis物理分页，开发者无须关心具体操作，配置好插件之后，写分页等同于普通 List 查询。
- 分页插件支持多种数据库：支持MySQL、MariaDB、Oracle、DB2、H2、HSQL、SQLite、Postgre、SQLServer等多种数据库。
- 内置性能分析插件：可输出SQL语句以及其执行时间，建议开发测试时启用该功能，能快速揪出慢查询。
- 内置全局拦截插件：提供全表delete、 update操作智能分析阻断，也可自定义拦截规则，预防误操作。



5.3.2 MyBatis-Plus快速入门

掌握MyBatis-Plus快速入门，能够使用通用Mapper、通用Service，以及条件构造器操作数据库中的数据

通过学习MyBatis-Plus的特性可知，MyBatis-Plus内置了通用Mapper和通用Service，启动时会自动注入基本的增删改查方法，并提供了强大的条件构造器，下面对MyBatis-Plus的这些基本功能进行讲解。

1.通用Mapper

MyBatis-Plus的通用Mapper是指其BaseMapper接口，Mybatis-Plus启动时自动解析，将实体表关系映射转换为Mybatis内部对象注入到容器中。BaseMapper接口中封装了基本的增删改查方法。

（1）新增方法

```
int insert(T entity);
```

1.通用Mapper

（2）更新方法

```
// 根据 whereWrapper 条件，更新记录
int update(@Param(Constants.ENTITY) T updateEntity,
    @Param(Constants.WRAPPER) Wrapper<T> whereWrapper);
// 根据ID更新
int updateById(@Param(Constants.ENTITY) T entity);
```

1.通用Mapper

（3）删除方法

```
// 根据entity条件，删除记录
int delete(@Param(Constants.WRAPPER) Wrapper<T> wrapper);
// 根据ID批量删除
int deleteBatchIds(@Param(Constants.COLLECTION)      Collection<? extends Serializable> idList);
// 根据ID删除
int deleteById(Serializable id);
// 根据columnMap条件，删除记录
int deleteByMap(@Param(Constants.COLUMN_MAP)      Map<String, Object> columnMap);
```

1.通用Mapper

(4) 查询方法

```
// 根据id查询
T selectById(Serializable id);
// 根据entity 条件，查询一条记录
T selectOne(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 根据id批量查询
List<T> selectBatchIds(@Param(Constants.COLLECTION)      Collection<? extends Serializable> idList);
// 根据entity 条件，查询全部记录
List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 根据columnMap条件进行查询
List<T> selectByMap(@Param(Constants.COLUMN_MAP)     Map<String, Object> columnMap);
```

1.通用Mapper

(4) 查询方法

```
// 根据 Wrapper 条件，查询全部记录
List<Map<String, Object>> selectMaps(@Param(Constants.WRAPPER)  Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询全部记录。注意： 只返回第一个字段的值
List<Object> selectObjs(@Param(Constants.WRAPPER)        Wrapper<T> queryWrapper);
// 根据entity条件，查询全部记录并翻页
IPage<T> selectPage(IPage<T> page，@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询全部记录并翻页
IPage<Map<String, Object>> selectMapsPage(IPage<T> page,  @Param(Constants.WRAPPER)
Wrapper<T> queryWrapper);
// 根据 Wrapper 条件，查询总记录数
Integer selectCount(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

2.通用Service

除了通用Mapper，MyBatis-Plus还提供了通用Service，通用Service指的是其IService接口，该接口中也提供了基本的增删改查方法。编写Service层代码时，可以使用自定义的Service接口继承IService接口，调用IService接口中的方法时，不需要手动实现，即可实现基本的增删改查功能。

（1）插入方法

```
// 插入一条记录
boolean save(T entity);
// 批量插入
boolean saveBatch(Collection<T> entityList);
// 批量插入
boolean saveBatch(Collection<T> entityList, int batchSize);
```

2.通用Service

（2）更新方法

```
// 根据 UpdateWrapper 条件，更新记录 需要设置sqlset
boolean update(Wrapper<T> updateWrapper);
// 根据 whereWrapper 条件，更新记录
boolean update(T updateEntity, Wrapper<T> whereWrapper);
// 根据 ID 选择修改
boolean updateById(T entity);
// 根据ID 批量更新
boolean updateBatchById(Collection<T> entityList);
// 根据ID 批量更新
boolean updateBatchById(Collection<T> entityList, int batchSize);
```

2.通用Service

（3）插入或更新方法

```
// 主键存在就更新记录，否则插入一条记录
boolean saveOrUpdate(T entity);
// 根据updateWrapper尝试更新，否继续执行saveOrUpdate(T)方法
boolean saveOrUpdate(T entity, Wrapper<T> updateWrapper);
// 批量插入或更新
boolean saveOrUpdateBatch(Collection<T> entityList);
// 批量插入或更新
boolean saveOrUpdateBatch(Collection<T> entityList, int batchSize);
```

2.通用Service

(4) 删除方法

```
// 根据 entity 条件，删除记录
boolean remove(Wrapper<T> queryWrapper);
// 根据id删除
boolean removeById(Serializable id);
// 根据columnMap 条件，删除记录
boolean removeByMap(Map<String, Object> columnMap);
// 根据id批量删除
boolean removeByIds(Collection<? extends Serializable> idList);
```

2.通用Service

（5）查询方法

```
// 根据 ID 查询
T getById(Serializable id);
// 根据 Wrapper，查询一条记录。结果集，如果是多个会抛出异常
T getOne(Wrapper<T> queryWrapper);
// 根据 Wrapper，查询一条记录
T getOne(Wrapper<T> queryWrapper, boolean throwEx);
// 根据 Wrapper，查询一条记录
Map<String, Object> getMap(Wrapper<T> queryWrapper);
// 根据 Wrapper，查询一条记录
<V> V getObj(Wrapper<T> queryWrapper, Function<? super Object, V> mapper);
```

2.通用Service

（5）查询方法

```
// 查询所有
List<T> list();
// 查询列表
List<T> list(Wrapper<T> queryWrapper);
// 查询（根据ID 批量查询）
Collection<T> listByIds(Collection<? extends Serializable> idList);
// 查询（根据columnMap 条件）
Collection<T> listByMap(Map<String, Object> columnMap);
// 查询所有列表
List<Map<String, Object>> listMaps();
```

2.通用Service

（5）查询方法

```
// 查询列表
List<Map<String, Object>> listMaps(Wrapper<T> queryWrapper);
// 查询全部记录
List<Object> listObjs();
// 查询全部记录
<V> List<V> listObjs(Function<? super Object, V> mapper);
// 根据 Wrapper 条件，查询全部记录
List<Object> listObjs(Wrapper<T> queryWrapper);
```

2.通用Service

（6）分页查询方法

```
// 无条件分页查询
IPage<T> page(IPage<T> page);
// 条件分页查询
IPage<T> page(IPage<T> page, Wrapper<T> queryWrapper);
// 无条件分页查询
IPage<Map<String, Object>> pageMaps(IPage<T> page);
// 条件分页查询
IPage<Map<String, Object>> pageMaps(IPage<T> page,
    Wrapper<T> queryWrapper);
```

3.条件构造器

在开发中，有时候希望查询根据所指定的条件进行增删改查操作，而这个条件可能包含多种情况。在MyBatis-Plus中可以使用条件构造器Wrapper根据具体的需求定义来封装指定的条件

（1）eq

```
eq(R column, Object val)
```

column表示数据库字段名，val表示字段值。eq()方法用于匹配字段中值等于某个值的记录，例如，eq("name", "老王")用于匹配name字段中值等于"老王"的记录。

3.条件构造器

（2）ne

```
ne(R column, Object val)
```

ne()方法用于匹配字段中值不等于某个值的记录，例如，eq("name", "老王")用于匹配name字段中值不等于"老王"的记录。

（3）gt

```
gt(R column, Object val)
```

gt()方法用于匹配字段中值大于某个值的记录，例如， gt("age", 18)用于匹配age字段中值大于18的记录。

3.条件构造器

（4）ge

```
ge(R column, Object val)
```

ge()方法用于匹配字段中值大于或等于某个值的记录，例如， ge("age", 18)用于匹配age字段中值大于或等于18的记录。

（5）lt

```
lt(R column, Object val)
```

lt()方法用于匹配字段中值小于某个值的记录，例如， lt("age", 18)用于匹配age字段中值小于18的记录。

3.条件构造器

（6）le

```
le(R column, Object val)
```

le()方法用于匹配字段中值小于或等于某个值的记录，例如， le("age", 18)用于匹配age字段中值小于或等于18的记录。

（7）between

```
between(R column, Object val1, Object val2)
```

between()方法用于匹配字段中值在指定区间的记录，例如， between("age", 18, 30)用于匹配age字段中值大于18并且小于30的记录。

3.条件构造器

（8）like

```
like(R column, Object val)
```

like()方法用于模糊匹配字段中的值，例如，like("name", "王")用于匹配name字段中值包含“王”的记录。

（9）in

```
in(R column, Collection<?> value)
```

in()方法用于匹配字段的值在指定组合中的记录，例如，in("age",{1,2,3})用于匹配age字段中值为1或者2或者3的记录。

3.条件构造器

（10）groupBy

```
groupBy(R... columns)
```

groupBy()方法用于给指定字段进行分组，例如，groupBy("id", "name")用于对记录根据id字段和name字段进行分组。

5.3.3 整合 MyBatis-Plus

掌握Spring Boot整合MyBatis-Plus，能够整合Spring Boot和MyBatis-Plus，并使用MyBatis-Plus进行基本的增删改查

1.配置依赖

在项目chapter05的pom.xml文件中配置MyBatis-Plus整合Spring Boot的启动器依赖，由于该依赖不是Spring Boot提供的，需要自行配置对应的依赖版本号，具体如文件5-8所示。

源 代 码

文件5-8

pom.xml

2.设置配置信息

引入的MyBatis-Plus整合Spring Boot启动器依赖中，提供了数据库连接的一些默认配置。如果有单独的MyBatis配置，请将对应的配置文件路径配置到configLocation中。

本案例只演示MyBatis-Plus的基本操作，不需要配置额外的MyBatis 配置，所以直接使用文件5-2中原有的数据库连接配置即可。

3.创建实体类

MyBatis-Plus标注实体类的注解与Spring Data JPA不一样，所以需要使用MyBatis-Plus的注解标注实体类。在com.itheima.chapter05.entity包下创建实体类EBook，具体如文件5-9所示。

源 代 码

文件5-9

EBook.java

4.自定义Mapper接口

在com.itheima.chapter05.dao包下创建自定义接口BookMapper，并使用该接口继承BaseMapper接口，具体如文件5-10所示。

源 代 码

文件5-10

BookMapper.java

5.创建Service接口和实现类

在项目的java目录下创建包com.itheima.chapter05.service，并在该包下创建Service接口和实现类，具体如文件5-11和文件5-12所示。

源 代 码

文件5-11 BookService.java

文件5-12 BookServiceImpl.java

6.扫描Mapper接口

在启动类Chapter05Application上方使用@MapperScan注解扫描指定路径的Mapper并交由Spring管理，具体如文件5-13所示。

源 代 码

文件5-13

Chapter05Application.java

7.定义测试方法

在src\test\java目录的com.itheima.chapter05包下创建测试类Chapter05ApplicationMPTests，在Chapter05ApplicationMPTests测试类中定义操作图书信息的测试方法，具体如文件5-14所示。

源 代 码

文件5-14

Chapter05ApplicationMPTests.java

8.测试操作图书信息

运行文件5-14中的saveEBook()方法，测试新增图书信息。

8.测试操作图书信息

运行文件5-14中的findEBook()方法，测试查询图书信息。

8.测试操作图书信息

运行文件5-14中的editEBook()方法，测试修改图书信息。

8.测试操作图书信息

运行文件5-14中的delEBook()方法，测试删除图书信息。

5.4 Spring Boot整合Redis

随着互联网Web 2.0的兴起，关系型数据库在处理超大规模和高并发的Web 2.0网站的数据时存在一些不足，需要采用更适合解决大规模数据集合、多重数据种类的数据库，通常将这种类型的数据库统称为非关系型数据库(No SQL，Not Only SQL)。常见的非关系型数据库有MongoDb、Redis等，其中Redis以超高的性能、完美的文档和简洁易懂的源码受到广大开发人员的喜爱。下面将对Redis的相关知识和Spring Boot整合Redis进行讲解。

5.4.1 Redis快速入门

熟悉Redis快速入门，能够说出Redis的概念和优点、安装和启动Redis的方法、以及说出Redis支持的数据类型

1.Redis概述

Redis（Remote Dictionary Server，远程字典服务）是一个基于内存的键值型非关系型数据库，以Key-Value的形式存储数据。Redis中存储键(Key)、值(Value)的方式和Java中的HashMap类似，键和值是映射关系。在同一个库中，Key是唯一不可重复的，每一个Key对应一个Value。键值存储的本质就是使用Key标示Value，当想要检索Value时，必须使用与Value相对应的Key进行查找。

1.Redis概述

Redis与传统的关系型数据库截然不同，Redis没有提供手动创建数据库的语句，Redis启动后会默认创建16个数据库，用0~15进行编号，默认使用编号为0的数据库。

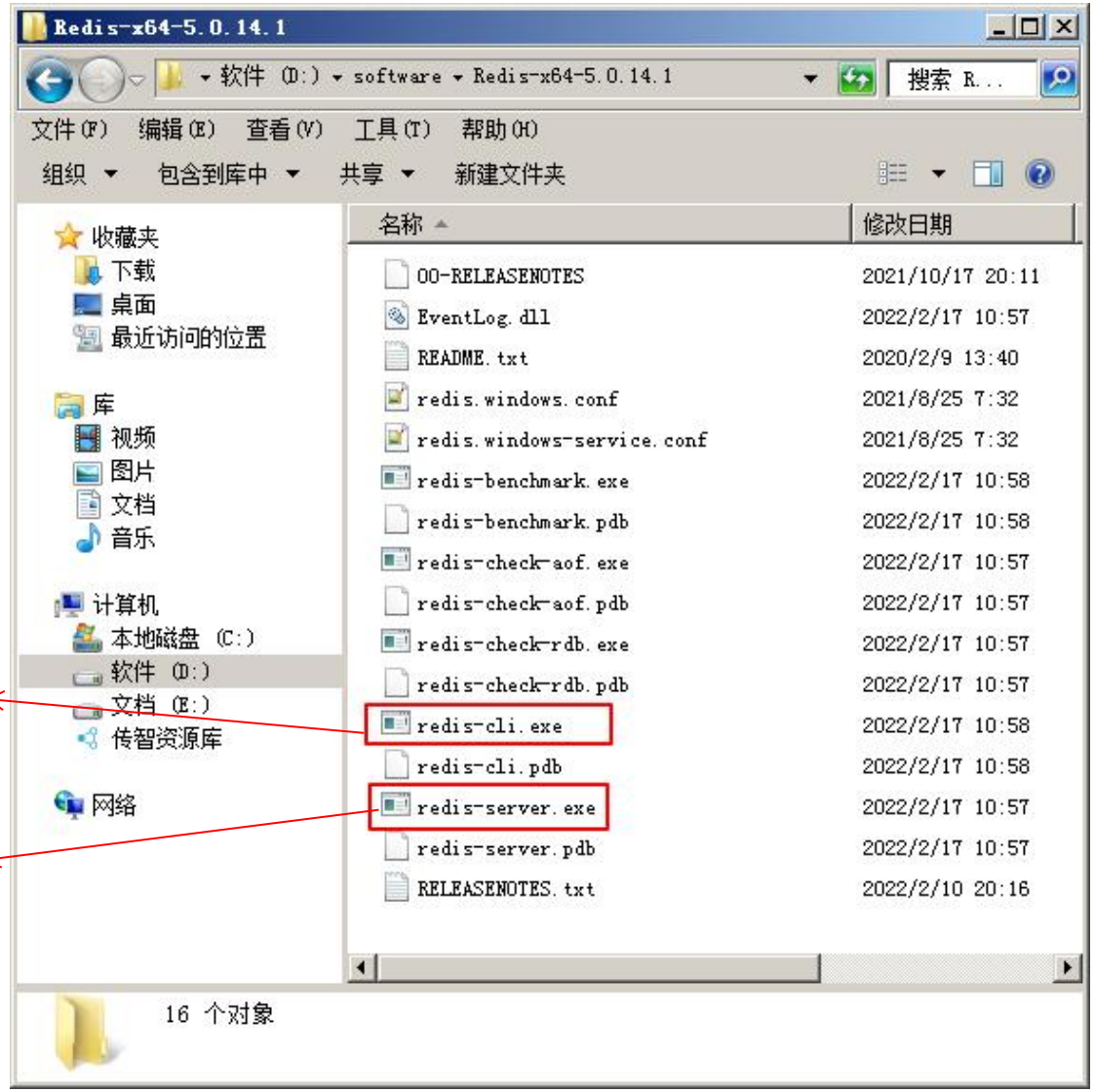
相较于其他的键值存储系统，Redis主要有以下优点。

- 存取速度快：Redis 基于内存来实现数据存取，相对于磁盘来说，其读写速度要高出好几个数量级，每秒可执行大约110000次的设置操作，或者执行81000次的读取操作。
- 支持丰富的数据类型：Redis支持开发人员常用的大多数数据类型，例如列表、集合、有序集合和散列等。
- 操作具有原子性：所有Redis操作都是原子操作，这使得两个客户端并发访问时，Redis服务器能接收更新后的值。
- 提供多种功能：Redis提供了多种功能特性，可用作非关系型数据库、缓存中间件、消息中间件等。

2.Redis安装和启动

要想使用非关系型数据库Redis，必须先安装Redis。Redis可以在Windows系统和Linux系统安装，也可以通过Docker镜像来安装，不同安装方式的安装过程也不相同。为了方便操作，此处选择在Windows平台下进行Redis安装。

2.Redis安装和启动

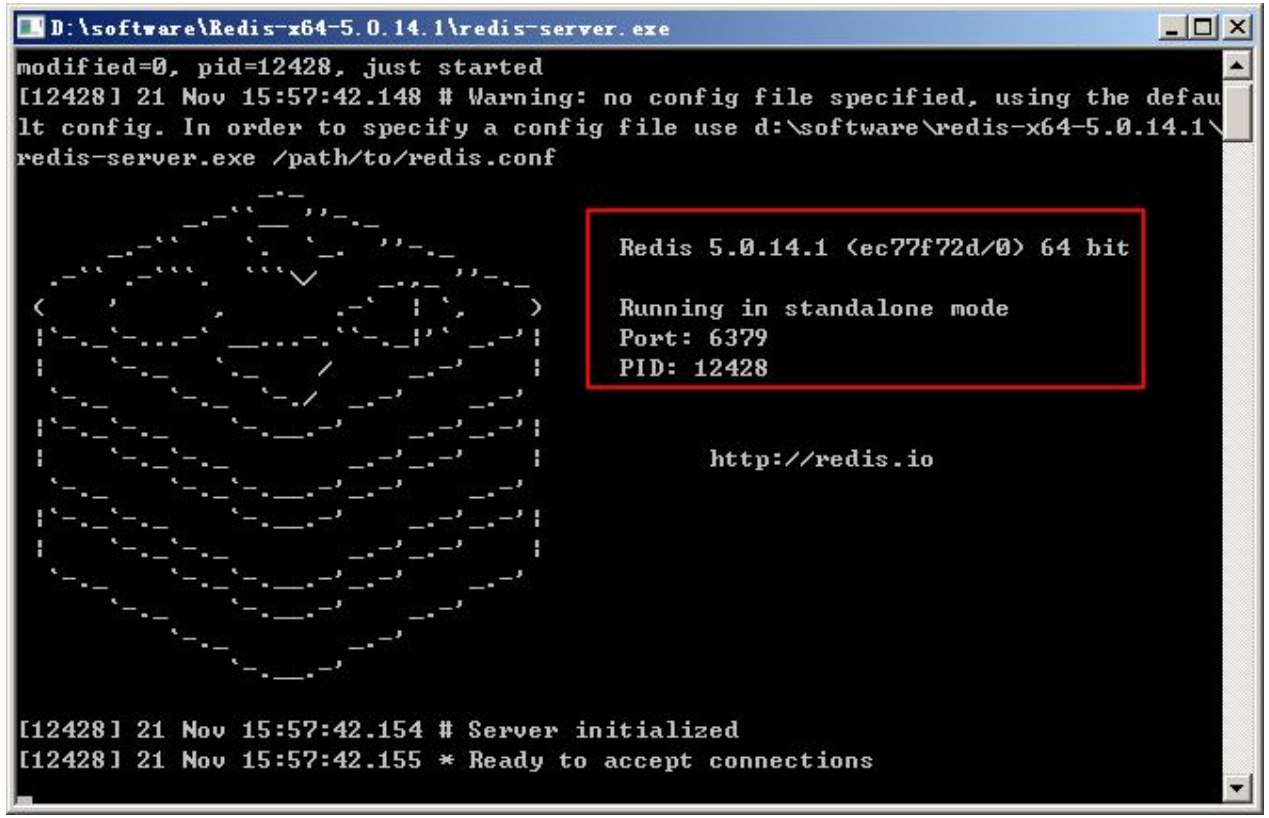


用于启动Redis服务。

客户端工具。

2.Redis安装和启动

双击redis-server.exe启动Redis服务。

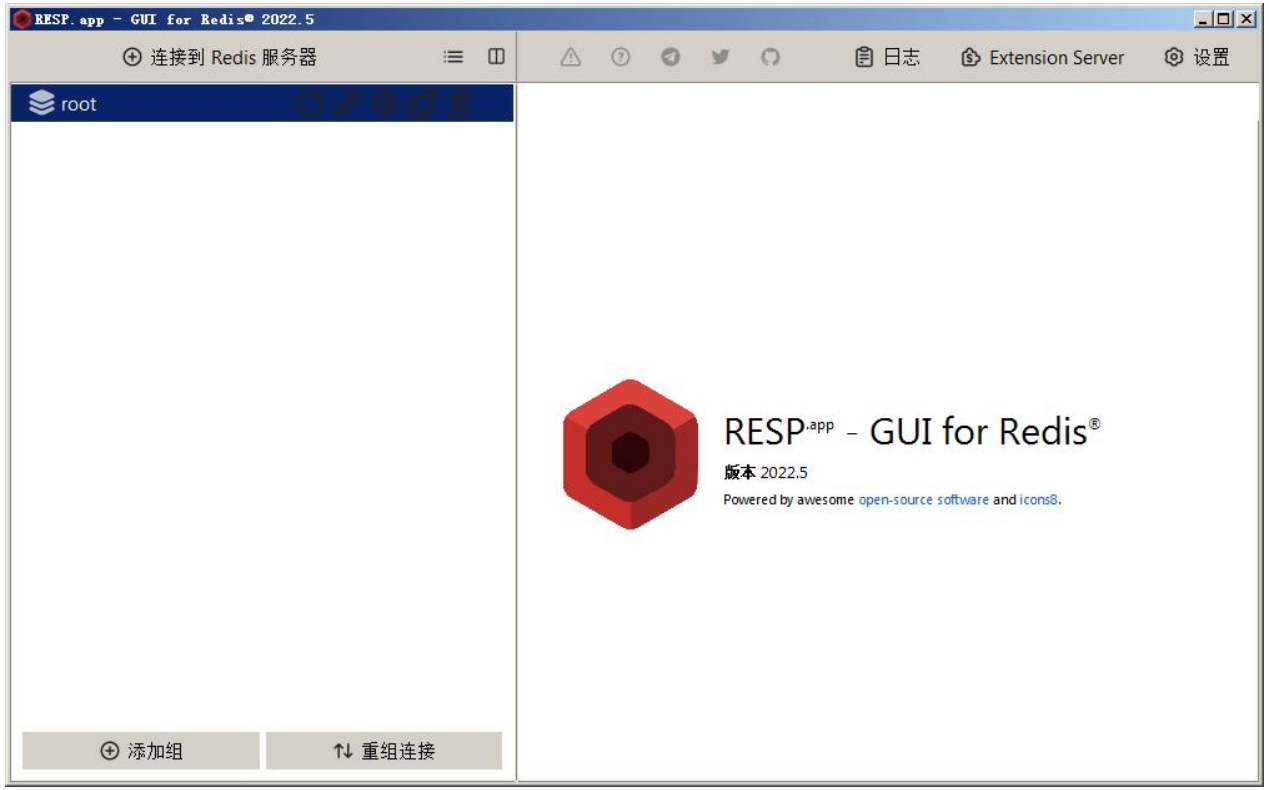


2.Redis安装和启动

双击redis-cli.exe启动客户端程序。

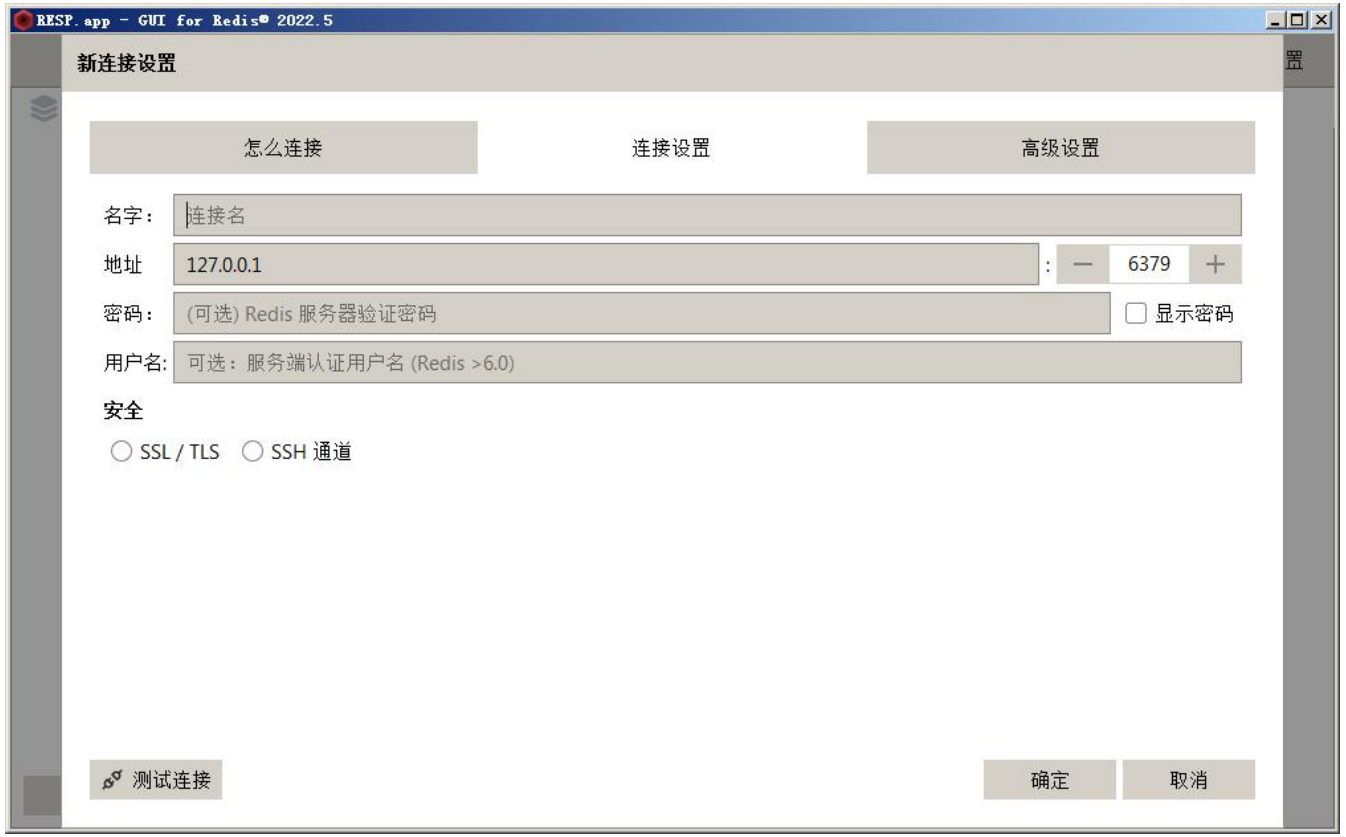
2.Redis安装和启动

Redis自带的客户端工具有时侯使用起来并不是特别方便，读者也可以使用一些图形化Redis客户端管理软件管理Redis。常用的有Redis Desktop Manager，其在2022年更名为RESP.app。



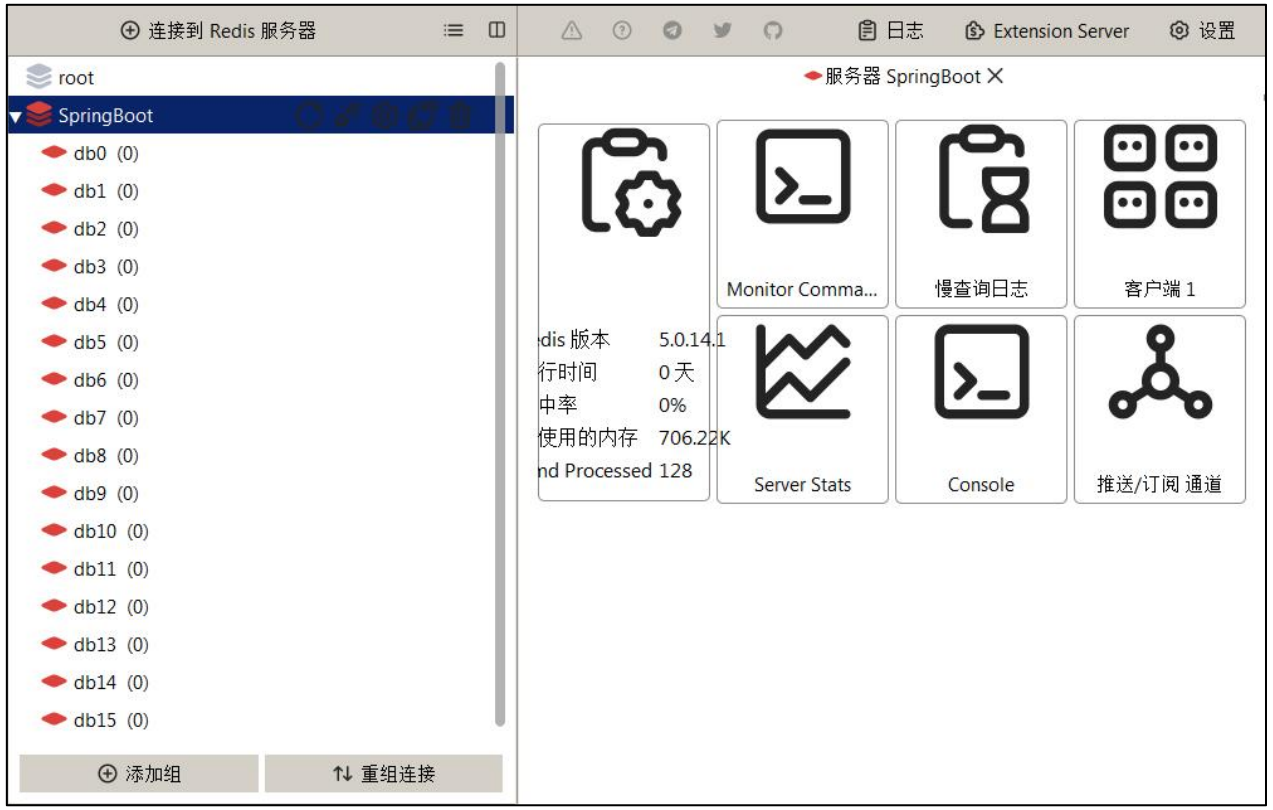
2.Redis安装和启动

单击RRESP.app主界面左侧的“连接到Redis服务器”，弹出“新连接设置”对话框。



2.Redis安装和启动

设置所连接的Redis的相关信息后，单击“确定”按钮创建连接。



3.Redis的数据类型

Redis中的数据库没有“数据表”的概念，通过Value不同的数据类型来实现存储数据的需求，不同的数据类型能够适应不同的应用场景，从而满足开发者的需求。Value的数据类型有五种常用数据类型，分别为String（字符串）、Hash（散列）、List（列表）、Set（集合）、SortedSet（有序集合）。

3.Redis的数据类型

（1）String

String可以灵活地表示字符串、整数、浮点数3种值，String有以下常见命令。

- SET key value：添加或者修改已经存在的键值对。
- GET key：根据键获取对应的值。
- MSET key1 value1 [key2 value2 ...]：批量添加多个键值对。
- MGET key1 [key2..]：根据一个或多个键获取对应的值。
- INCR key：将键存储的整数值自增1。
- INCRBY key increment：将键存储的整数值根据指定步长increment自增。
- INCRBYFLOAT key increment：将键存储的浮点数根据指定步长increment自增。
- SETNX key value：当且仅当Key不存在时，添加一个键值对。
- SETEX key seconds value：添加一个String类型的键值对，并且指定有效期为seconds秒。

3.Redis的数据类型

（2）List

Redis中的List与Java中的LinkedList类似，可以看作是一个双向链表结构，既可以支持正向检索，也可以支持反向检索。List类型的数据有序、元素可以重复、插入和删除速度快、查询速度一般。常见命令如下。

- LPUSH key value1 [value2...]：根据键向列表左侧插入一个或多个元素。
- LPOP key：根据键移除并返回列表左侧的第一个元素，没有则返回nil。
- RPUSH key value1 [value2 ...]：根据键向列表右侧插入一个或多个元素。
- RPOP key：根据键移除并返回列表右侧的第一个元素。
- LRANGE key star end：根据键返回指定范围内的所有元素。
- BLPOP和BRPOP：与LPOP和RPOP类似，只不过在没有元素时等待指定时间，而不是直接返回nil。

3.Redis的数据类型

（3）Set

Redis的Set类型与Java中的HashSet类似，可以看作是一个Value 为null的HashMap。Set类型的数据具有无序、元素不可重复、查找速度快和支持交集、并集、差集等功能的特征，常见命令如下。

- SADD key member1 [member2 ...]：将一个或多个member元素加入到集合key中。
- SREM key member1 [member2 ...] : 删除集合key中的一个或多个member元素。
- SCARD key： 返回集合key中元素的个数。
- SISMEMBER key member：判断元素member是否存在于集合key中。
- SMEMBERS key：获取集合key中的所有元素。
- SINTER key1 [key2 ...]：获取所有集合的交集。

3.Redis的数据类型

（4）SortedSet

Redis中的SortedSet是一个可排序的Set集合，SortedSet中的每一个元素都带有一个score属性，可以基于score属性对元素排序，底层的实现是一个跳表（SkipList）加Hash表。SortedSet常见命令如下。

- ZADD key score member：添加一个或多个元素到集合key中，如果已经存在则更新其score值。
- ZREM key member：删除集合key中的元素member。
- ZSCORE key member：获取集合key中的元素member 的score值。
- ZRANK key member：获取集合key中的元素member的排名。
- ZCARD key：获取集合key中的元素个数。
- ZCOUNT key min max：统计集合key中score值在给定范围内的所有元素的个数。
- ZINCRBY key increment member：让集合key中的元素member根据步长increment进行自增。

3.Redis的数据类型

（5）Hash

Hash是由字符串类型的field和value 组成的映射表，可以把它理解成一个包含了多个键值对的集合，一般用于存储对象。Hash有以下常见命令。

- HSET key field value：将集合key中字段field的值设为value。
- HGET key field：获取集合key中字段field的值。
- HMSET key field1 value1[field2 value2 ...]：将一个或多个field-value 对设置到集合key中。
- HMGET key field1 [field2 ...]：获取集合key中一个或多个field的值。
- HGETALL key：获取集合key中的所有的field和Value 。
- HKEYS key：获取集合key中的所有的field。
- HINCRBY key field increment：让集合key中的field字段根据步长increment自增。
- HSETNX key field value：在集合key中，当且仅当field不存在时，添加字段field，字段对应的值为value。

5.4.2 Spring Data Redis快速入门

掌握Spring Data Redis快速入门，能够说出Spring Data Redis的特性，以及应用Spring Data Redis的常见操作

为了方便开发者使用Redis，Redis官方为主流的编程语言都提供了对应的客户端，其中面向Java的客户端有Redisson、Jedis和Lettuce等。Jedis和Lettuce提供了Redis命令对应的API，因此操作Redis比较方便。如果一个项目中使用了Lettuce连接Redis，后来决定弃用Lettuce改用Jedis，就要面临修改代码的问题，对于此种问题，可以使用Spring Data Redis。下面对Spring Data Redis概述和常见操作进行讲解。

1.Spring Data Redis概述

Spring Data Redis是Spring Data在Spring管理的项目中对Redis操作的具体实现。Spring Boot为支持Redis提供了spring-boot-starter-data-redis.jar，该Starter使用Spring Data Redis对底层Lettuce和Jedis进行了封装，并为Lettuce和Jedis提供了自动配置。Spring Data Redis具有如下特性。

- 提供了对不同Redis客户端的整合Lettuce和Jedis。
- 提供了RedisTemplate统一API来操作Redis。
- 支持Redis的发布订阅模型。
- 支持Redis哨兵和Redis集群。
- 支持基于Lettuce的响应式编程。
- 支持基于JDK、JSON、字符串、Spring对象的数据序列化及反序列化。
- 支持基于Redis的JDKCollection实现。

2.Spring Data Redis常见操作

（1）RedisTemplate 常见 API

Spring Data Redis中提供了RedisTemplate工具类，该工具类封装了各种对Redis的操作，并且将不同数据类型的操作API封装到了不同的Operation接口对象中。

获取常用Operation接口对象的方法

方法	说明
ValueOperations<K, V> opsForValue()	获取操作String类型数据的对象。
ListOperations<K, V> opsForList()	获取操作List类型数据的对象。
SetOperations<K, V> opsForSet()	获取操作Set类型数据的对象。
ZSetOperations<K, V> opsForZSet()	获取操作SortedSet类型数据的对象。
HashOperations<K, HK, HV> opsForHash()	获取操作Hash类型数据的对象。

2.Spring Data Redis常见操作

为了能更便捷操作Redis，RedisTemplate还可以通过bound绑定指定的Key，绑定Key后再次进行一系列的操作时，无须显式的再次指定Key。

常用的绑定Key的方法

方法	说明
BoundValueOperations<K, V> boundValueOps(K key)	绑定映射String类型数据的Key
BoundListOperations<K, V> boundListOps(K key)	绑定映射List类型数据的Key
BoundSetOperations<K, V> boundSetOps(K key	绑定映射Set类型数据的Key
BoundZSetOperations<K, V> boundZSetOps(K key)	绑定映射SortedSet类型数据的Key
BoundHashOperations<K, HK, HV> boundHashOps(K key)	绑定映射Hash类型数据的Key

（2）Spring Data Redis的常用注解和序列化策略

Spring Data Redis操作的不是持久化类而是数据类，为了实现数据类与Redis之间的映射关系，Spring Data Redis提供了如下注解。

- @RedisHash：该注解在类上进行标注，用于指定将数据类映射到Redis中的存储空间。
- @Id：用于标示实体类主键。在Redis数据库中会默认生成字符串形式的HashKey，用于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：用于标示对应属性在Redis数据库中生成的二级索引。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，可以方便地进行数据条件查询。

（2）Spring Data Redis的常用注解和序列化策略

Redis的序列化策略有两种，一种是String的序列化策略，另一种是JDK的序列化策略。

RedisTemplate默认采用的是JDK的序列化策略，保存的Key和Value都采用此策略进行序列化保存。

RedisTemplate使用JDK的序列化策略保存数据时，会将数据先序列化成字节数组然后再存入Redis数据库，如果使用Redis根据查看对应的数据，数据是以字节数组显示的，而不是以原生可读的形式显示的。

RedisTemplate有一个子类StringRedisTemplate，StringRedisTemplate默认采用的是String的序列化策略，保存的key和value都是采用此策略序列化保存的，默认存入的数据就是原始的字符串。

5.4.3 整合Redis

掌握Spring Boot整合Redis，能够整合Spring Boot和Redis，并使用Spring Data Redis向Redis中存储和读取数据

1.配置依赖

在项目chapter05的pom.xml文件中配置Spring Boot提供的Spring Data Redis启动器依赖spring-boot-starter-data-redis。

2.设置配置信息

引入的spring-boot-starter-data-redis依赖中，提供了操作Redis的一些默认配置信息。例如，连接的Redis主机地址、主机的端口号、指定连接的数据库、连接池的连接数等信息。如果没有指定会默认连接本地端口为6379的主机。本案例演示的Redis安装在本地，所以不用进行额外的配置，使用默认的配置信息即可。

3.创建数据类

在com.itheima.chapter05.entity包下创建数据类User，具体如文件5-15所示。

源 代 码

文件5-15

User.java

4.自定义Repository接口

在com.itheima.chapter05.dao包下创建自定义接口UserRepository，并使用该接口继承CrudRepository接口，具体如文件5-16所示。

源 代 码

文件5-16

UserRepository.java

5.定义测试方法

在src\test\java目录的com.itheima.chapter05包下创建测试类Chapter05ApplicationRedisTests，在该测试类中定义操作用户信息的测试方法，具体如文件5-17所示。

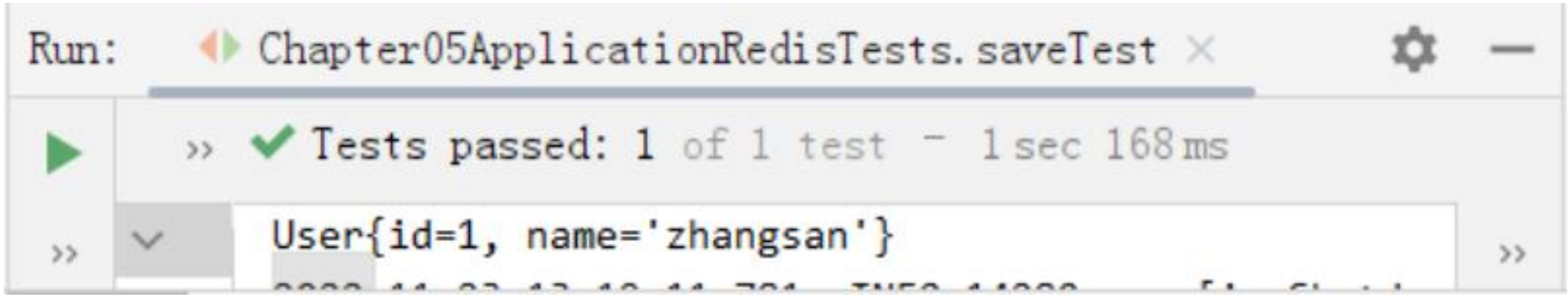
源 代 码

文件5-17

Chapter05ApplicationRedisTests.java

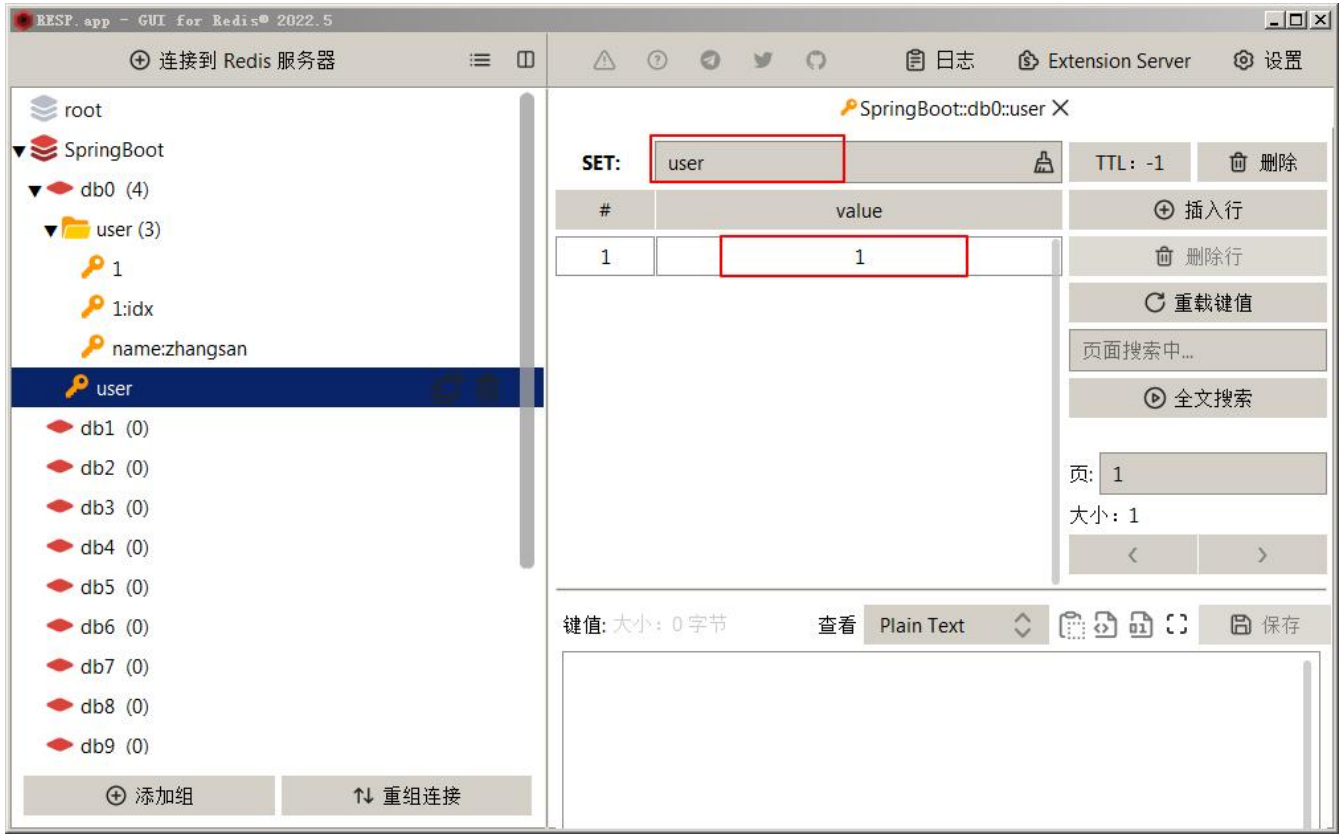
6.测试操作图书信息

启动Redis服务后，运行文件5-17中的saveTest()方法，测试保存和查询用户信息。



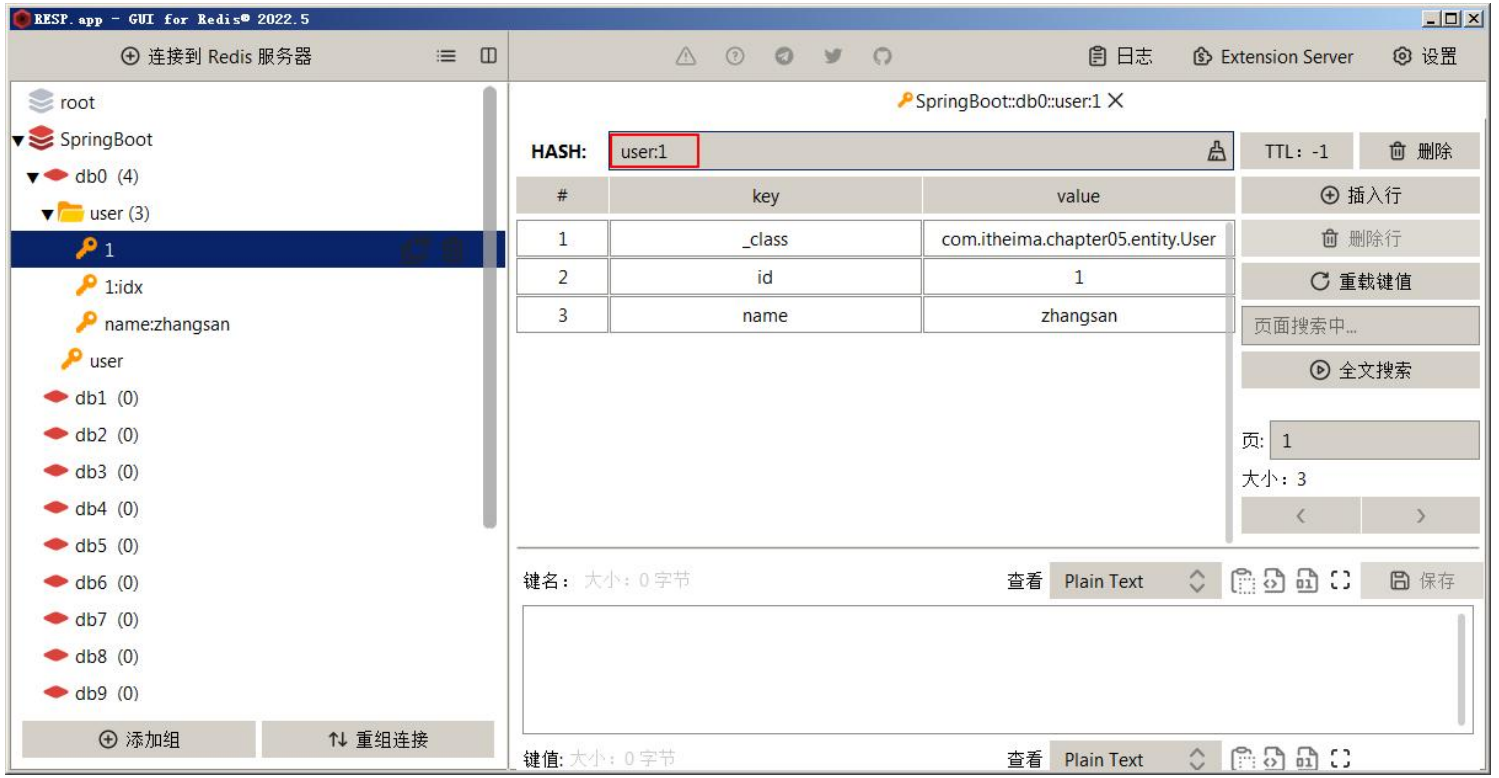
6.测试操作图书信息

启动RESP.app，查看此时Redis中存储的数据。



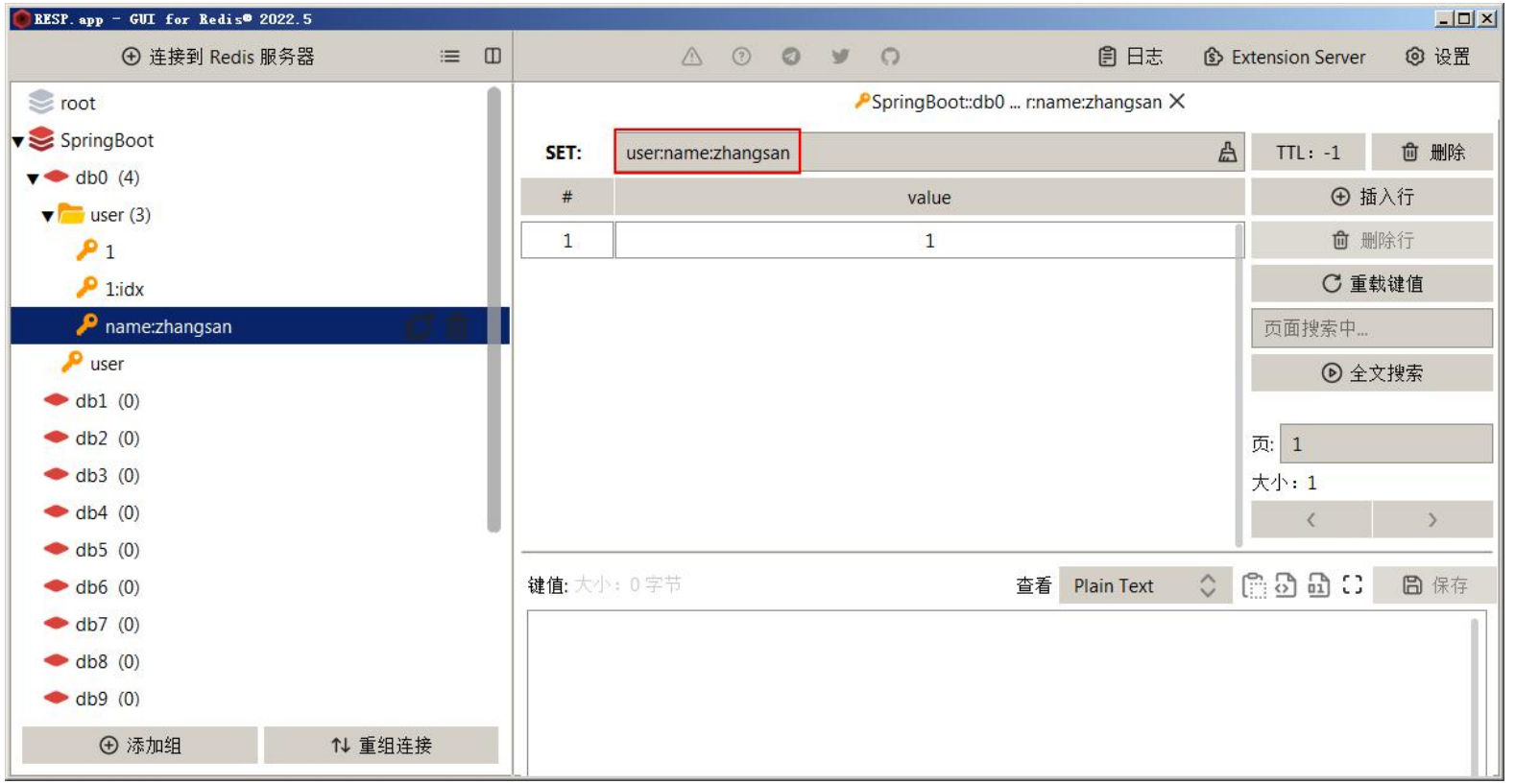
6.测试操作图书信息

单击RESP.app中名称为“user：1”的Key。



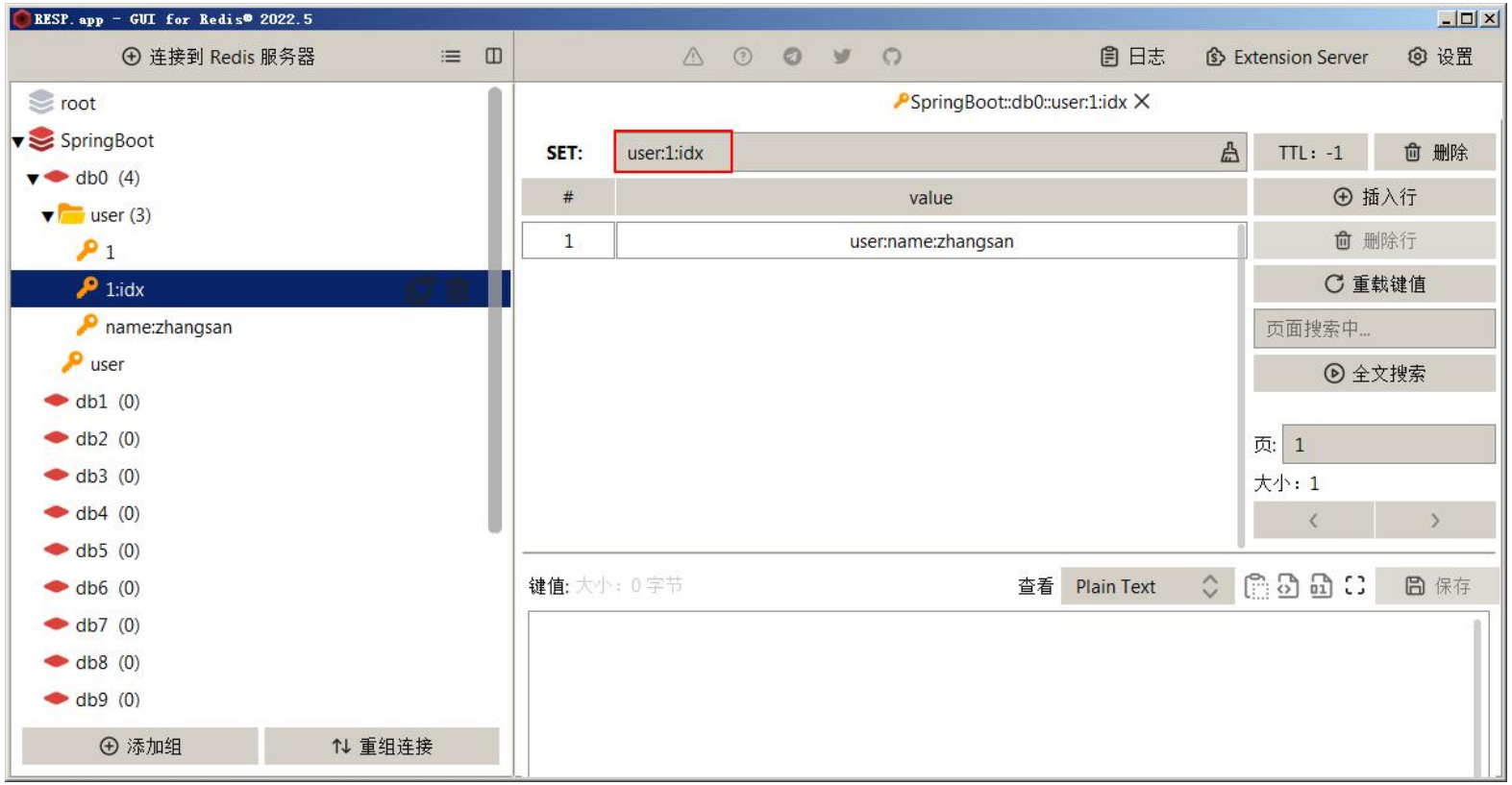
6.测试操作图书信息

单击RESP.app中名称为“user:name:zhangsan”的Key。



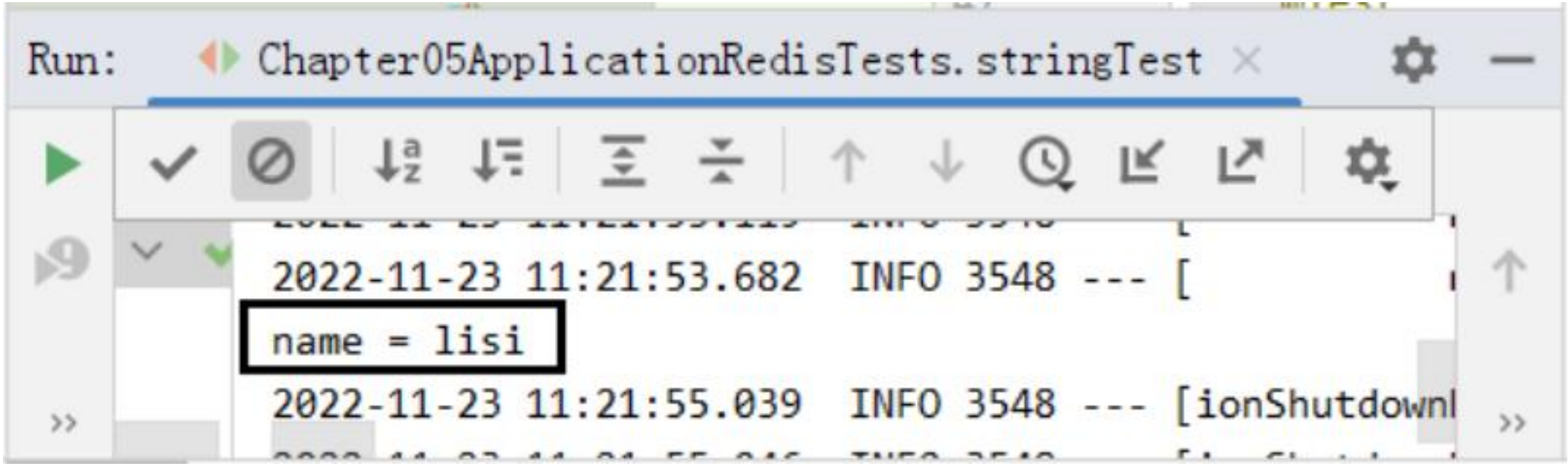
6.测试操作图书信息

单击RESP.app中名称为“user:1:idx”的Key。



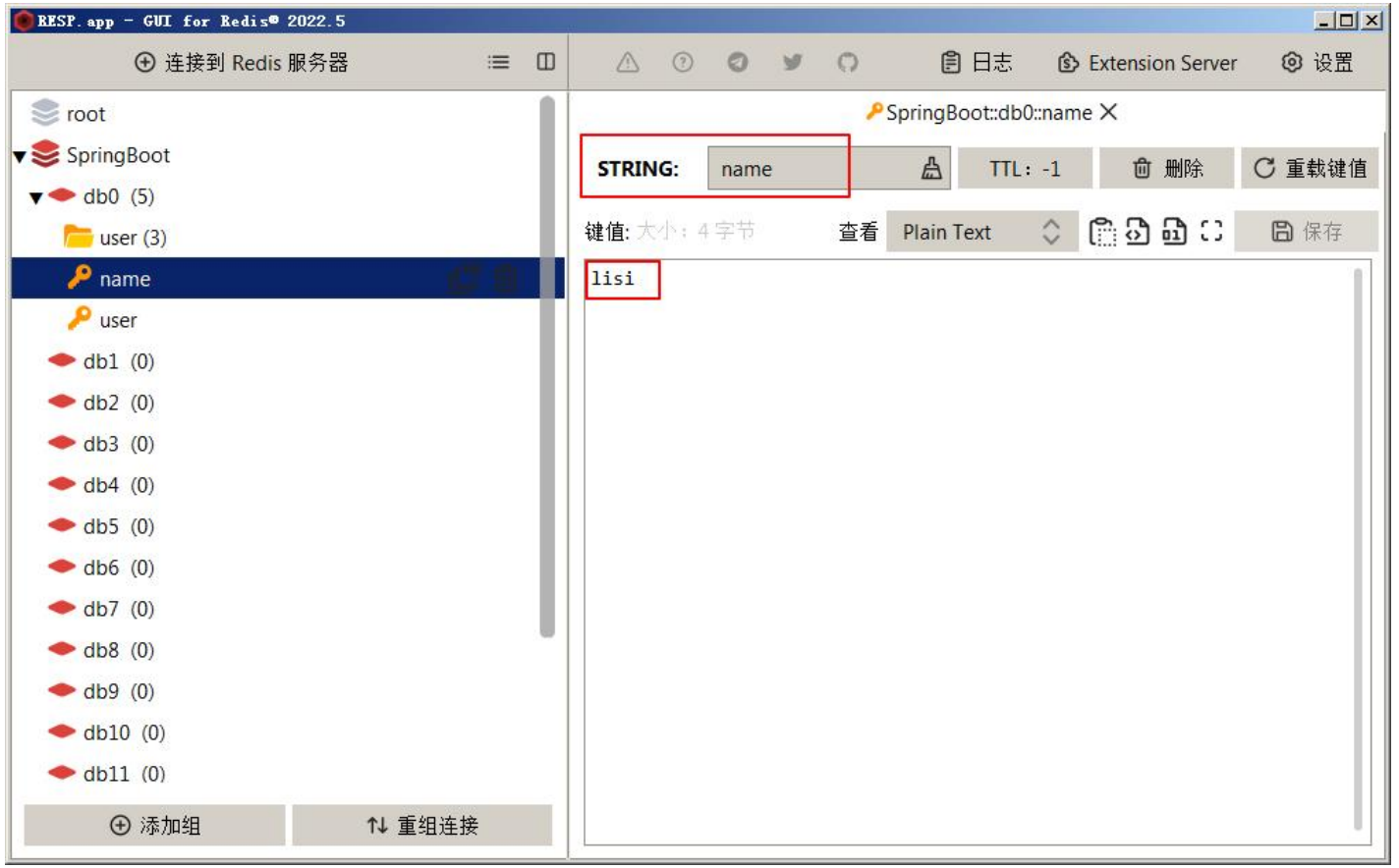
6.测试操作图书信息

运行文件5-17中的stringTest()方法，测试保存和查询String类型的数据。



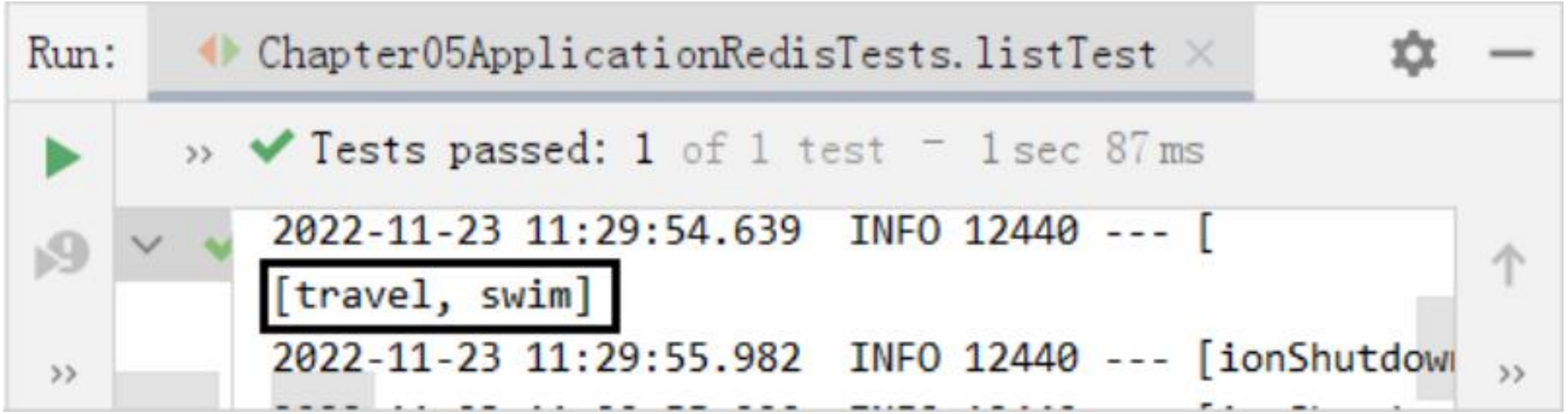
6.测试操作图书信息

在RESP.app中查看Key为name的数据。



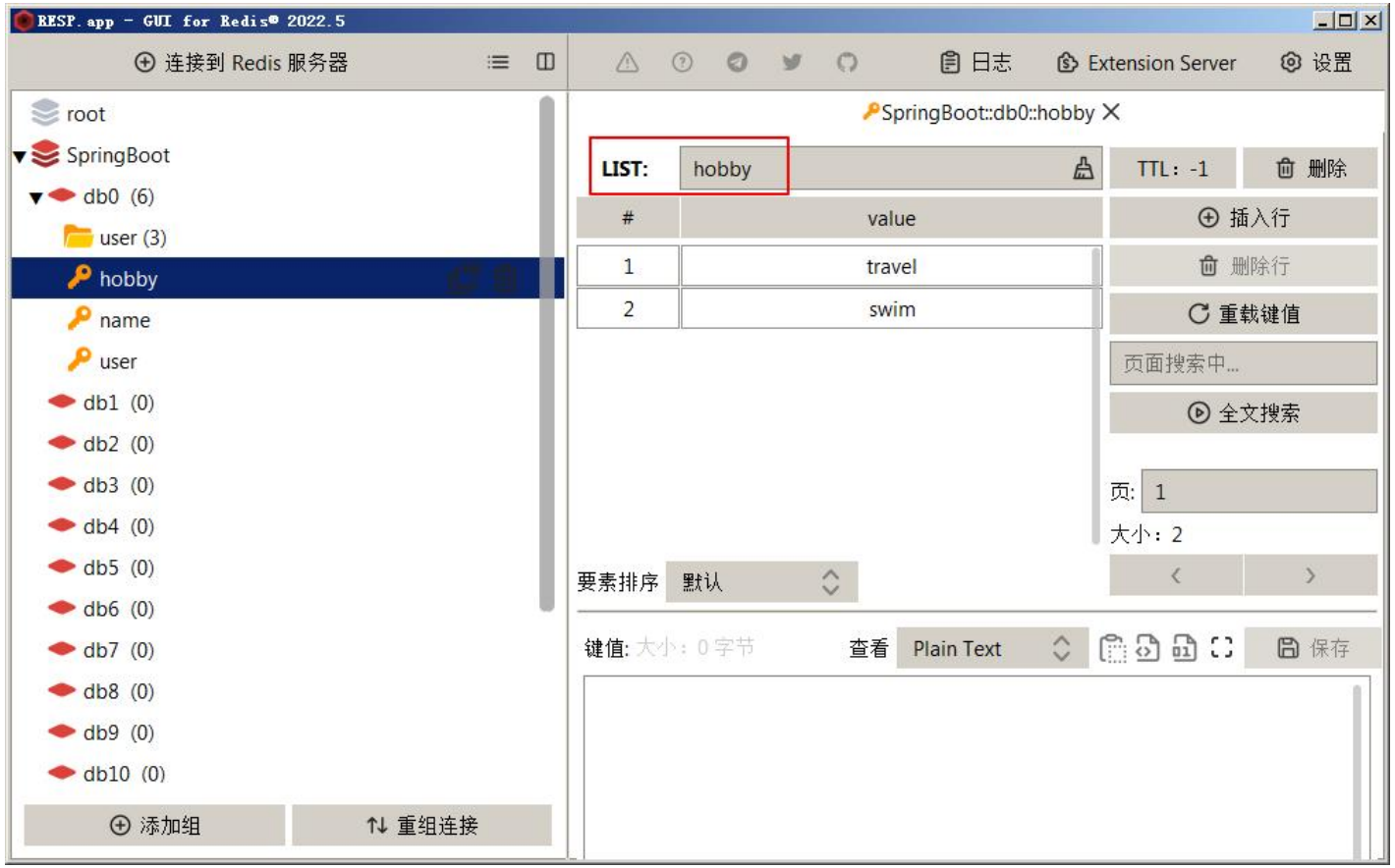
6.测试操作图书信息

运行文件5-17中的listTest()方法，测试保存和查询List类型的数据。



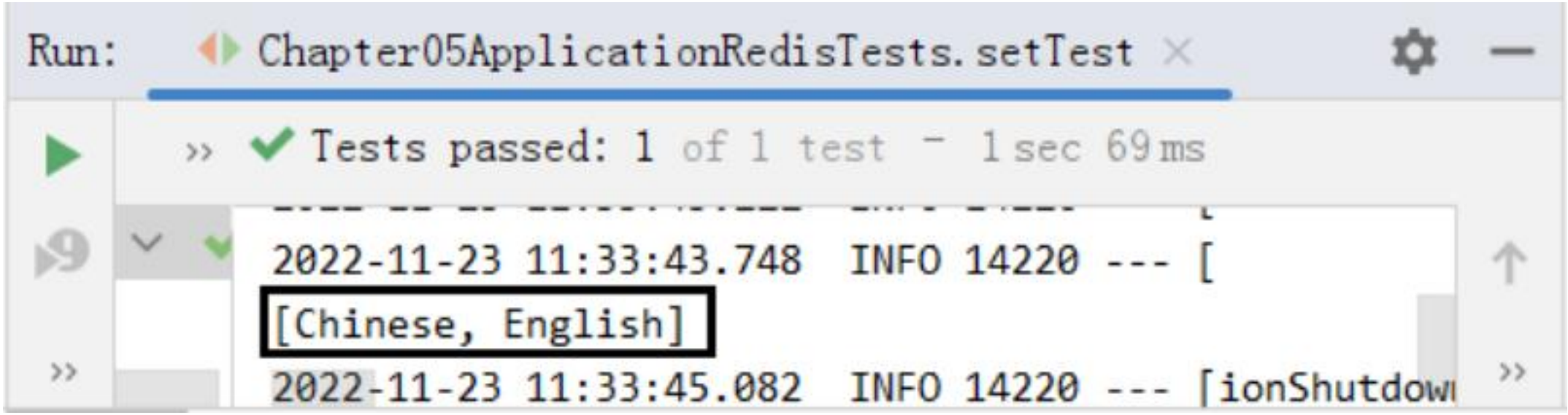
6.测试操作图书信息

在RESP.app中查看Key为hobby的数据。



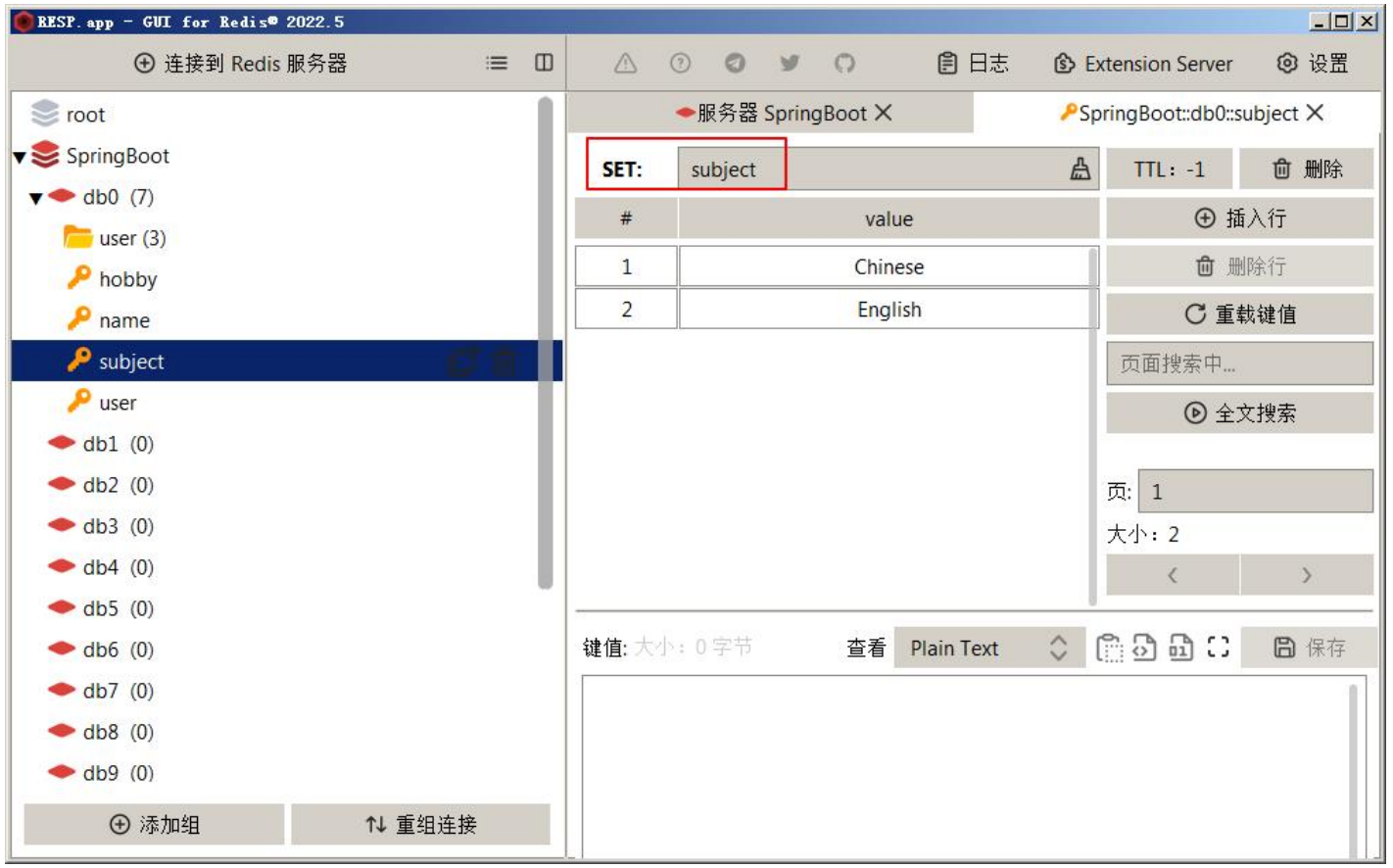
6.测试操作图书信息

运行文件5-17中的setTest()方法，测试保存和查询Set类型的数据。



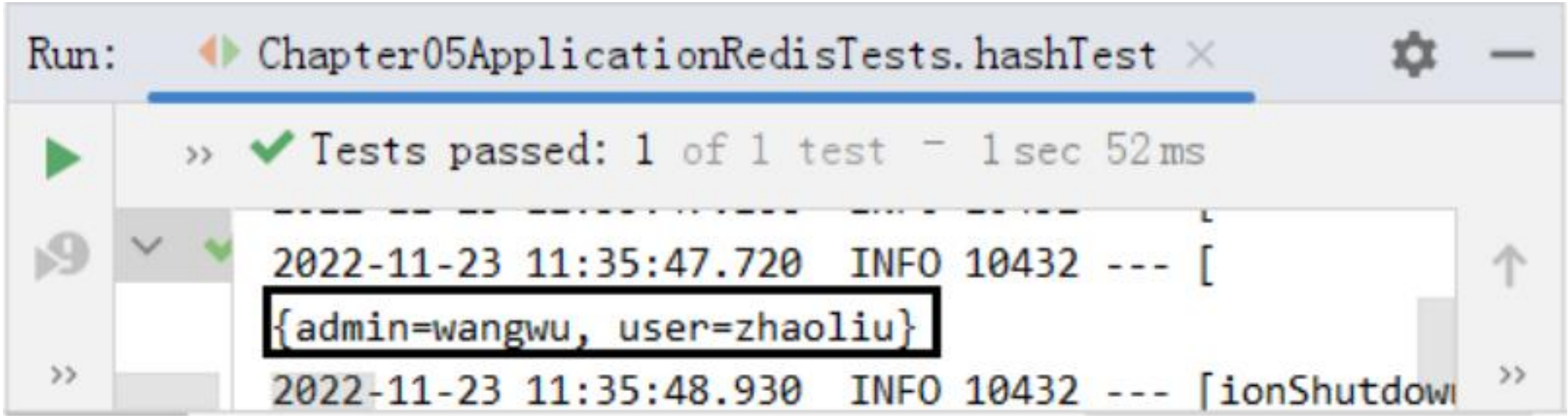
6.测试操作图书信息

在RESP.app中查看Key为subject的数据。



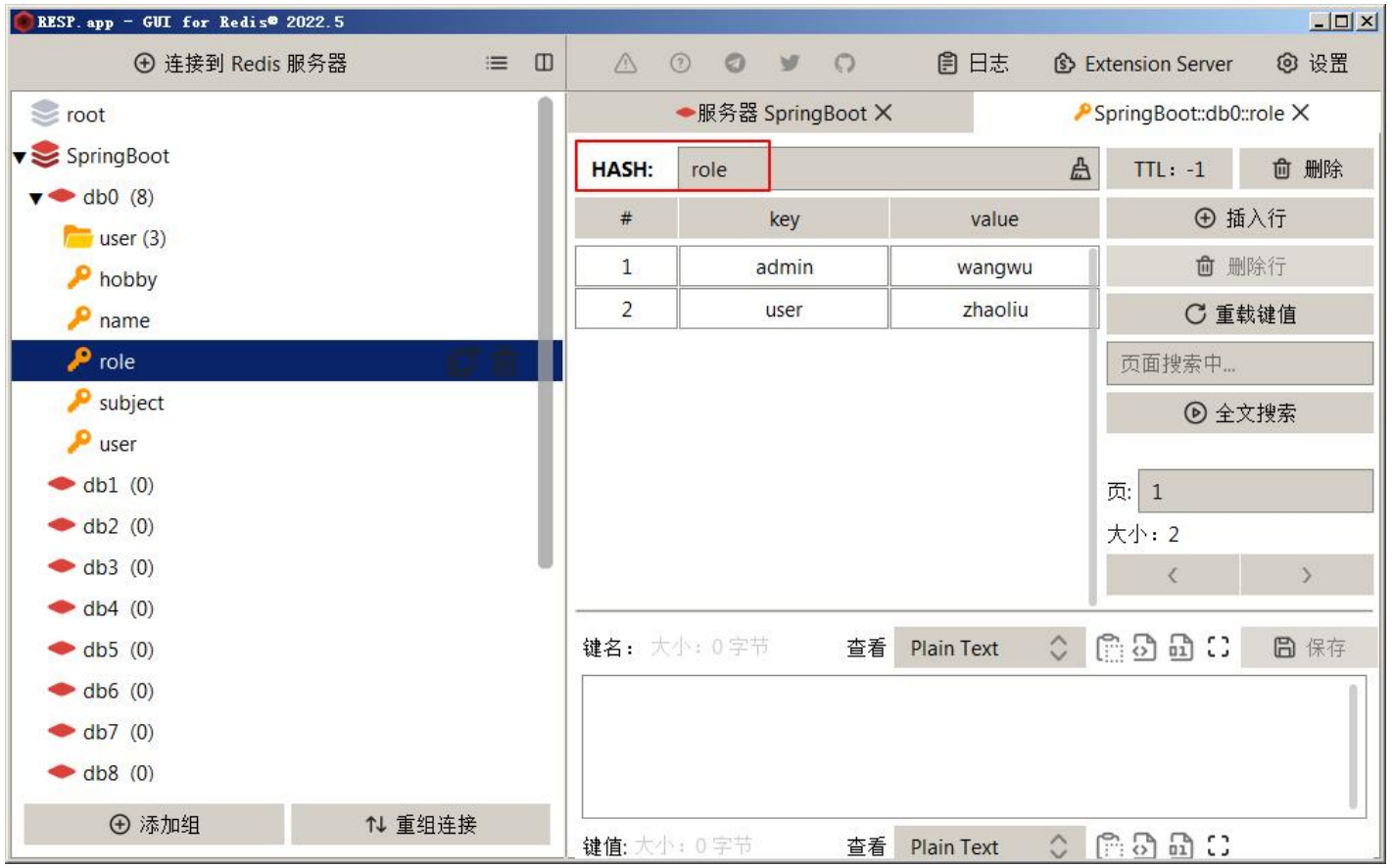
6.测试操作图书信息

运行文件5-17中的hashTest()方法，测试保存和查询Hash类型的数据。



6.测试操作图书信息

在RESP.app中查看Key为role的数据。



本章小结

本章主要对Spring Boot数据访问进行了讲解。首先讲解了Spring Data；然后讲解了Spring Data JPA的基础知识，以及Spring Boot整合Spring Data JPA；接着讲解了MyBatis-Plus的基础知识，以及Spring Boot整合MyBatis-Plus。最后讲解了Redis和Spring Data Redis的基础知识，以及Spring Boot整合Redis。通过本章的学习，希望大家可以在Spring Boot项目中正确应用数据访问技术，为后续更深入学习Spring Boot做好铺垫。

為千萬學生少走彎路而著書

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌