链滴

SpringBoot 入门教程 (十一) | 整合数据缓存 Cache

作者：JavaFish

原文链接：https://ld246.com/article/1621309062348

来源网站：链滴

01 前言

如题，今天介绍 SpringBoot 的数据缓存。做过开发的都知道程序的瓶颈在于数据库，我们也知道内的速度是大大快于硬盘的，当需要重复获取相同数据时，一次又一次的请求数据库或者远程服务，导大量时间耗费在数据库查询或远程方法调用上，导致性能的恶化，这便是数据缓存要解决的问题。

02 Spring 的缓存支持

Spring 定义了 org.springframework.cache.CacheManager 和 org.springframework.cache.Cache 接口用于统一不同的缓存技术。其中，CacheManager 是 Spring 提供的各种缓存技术的抽象接口，ache 接口则是包含了缓存的各种操作（增加，删除，获取缓存，一般不会直接和此接口打交道）。

1、Spring 支持的 CacheManager

针对不同的缓存技术，实现了不同的 CacheManager ，Spring 定义了下表所示的 CacheManager：

CacheManager	描述
SimpleCacheManager	使用简单的 Collection 存储缓存，主要用于测试
ConcurrentMapCacheManager	使用 ConcurrentMap 来存储缓存
NoOpCacheManager	仅测试用途，不会实际缓数据
EhCacheCacheManager	使用 EhCache 作为缓技术
GuavaCacheManager	使用 Google Guava 的 GavaCache 作为缓存技术
HazelcastCacheManager	使用 Hazelcast 作为存技术
JCacheCacheManager	支持 JCache(JSR-107) 准的实现作为缓存技术，如 ApacheCommonsJCS
RedisCacheManager	使用 Redis 作为缓存技术

在使用以上任意一个实现的 CacheManager 的时候，需注册实现的 CacheManager 的 Bean，如：

```
@Bean
public EhCacheCacheManager cacheManager(CacheManager
 ehCacheCacheManager){
    return new  EhCacheCacheManager(ehCacheCacheManager);
}
```

注意，每种缓存技术都有很多的额外配置，但配置 cacheManager 是必不可少的。

2、声明式缓存注解

Spring 提供了 4 个注解来声明缓存规则（又是使用注解式的 AOP 的一个例子）。4 个注解如下表示：

注解	解释
@Cacheable	在方法执行前 Spring 先查看缓存是否有数据，若有，则直接返回缓存数据；若无数据，调用方法将方法返回值放入缓存中
@CachePut	无论怎样，都会将方法的返回值放缓存中。
@CacheEvict	将一条或多条数据从缓存中删除
@Caching	可以通过 @Caching 注解组合多个解策略在一个方法上

@Cacheable、@CachePut、@CacheEvict 都有 value 属性，指定的是要使用的缓存名称；key 属指定的是数据在缓存中存储的键。

3、开启声明式缓存支持

开启声明式缓存很简单，只需在配置类上使用 @EnableCaching 注解即可，例如：

```
@Configuration
@EnableCaching
public class AppConfig{

}
```

03 SpringBoot 的支持

在 Spring 中使用缓存技术的关键是配置 CacheManager，而 SpringBoot 为我们配置了多个 Cach Manager 的实现。

它的自动配置放在 org.springframework.boot.autoconfigure.cache 包中。

在不做任何配置的情况下，默认使用的是 SimpleCacheConfiguration，即使用 ConcurrentMapCa heManager。SpringBoot 支持以前缀来配置缓存。例如：

```
spring.cache.type= # 可选 generic、ehcache、hazelcast、infinispan、jcache、redis、guava、s mple、none
spring.cache.cache-names= # 程序启动时创建的缓存名称
spring.cache.ehcache.config= # ehcache 配置文件的地址
spring.cache.hazelcast.config= # hazelcast配置文件的地址
spring.cache.infinispan.config= # infinispan配置文件的地址
spring.cache.jcache.config= # jcache配置文件的地址
spring.cache.jcache.provider= # 当多个 jcache 实现在类路径的时候，指定 jcache 实现
# 等等。。。
```

在 SpringBoot 环境下，使用缓存技术只需要在项目中导入相关缓存技术的依赖包，并在配置类中使用 @EnableCaching 开启缓存支持即可。

04 代码实现

本文将以 SpringBoot 默认的 ConcurrentMapCacheManager 作为缓存技术，演示 @Cacheable @CachePut、@CacheEvict。

1、准备工作

- IDEA
- JDK 1.8
- SpringBoot 2.1.3

2、Pom.xml 文件依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
MLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.3.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.nasus</groupId>
    <artifactId>cache</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>cache</name>
    <description>cache Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <!-- cache 依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-cache</artifactId>
        </dependency>
        <!-- JPA 依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <!-- web 启动类 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- mysql 数据库连接类 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
```

```
        <!-- lombok 依赖，简化实体 -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!-- 单元测试类 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

注释很清楚，无需多言。不会就谷歌一下。

3、Application.yaml 文件配置

```
spring:
  # 数据库相关
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=utf8&serverTi
ezone=UTC&useSSL=true
    username: root
    password: 123456
  # jpa 相关
  jpa:
    hibernate:
      ddl-auto: update   # ddl-auto: 设为 create 表示每次都重新建表
    show-sql: true
```

4、实体类

```
package com.nasus.cache.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    @Id
    @GeneratedValue
    private Integer id;

    private String name;

    private Integer age;

}
```

5、dao 层

```
package com.nasus.cache.repository;

import com.nasus.cache.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface StudentRepository extends JpaRepository<Student,Integer> {
}
```

6、service 层

```
package com.nasus.cache.service;

import com.nasus.cache.entity.Student;

public interface StudentService {

    public Student saveStudent(Student student);

    public void deleteStudentById(Integer id);

    public Student findStudentById(Integer id);

}
```

实现类：

```
package com.nasus.cache.service.impl;

import com.nasus.cache.entity.Student;
import com.nasus.cache.repository.StudentRepository;
import com.nasus.cache.service.StudentService;
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

@Service
public class StudentServiceImpl implements StudentService {

    // 使用 slf4j 作为日志框架
    private static final Logger LOGGER = LoggerFactory.getLogger(StudentServiceImpl.class);

    @Autowired
    private StudentRepository studentRepository;

    @Override
    @CachePut(value = "student",key = "#student.id")
    // @CachePut 缓存新增的或更新的数据到缓存，其中缓存名称为 student 数据的 key 是 student
的 id
    public Student saveStudent(Student student) {
        Student s = studentRepository.save(student);
        LOGGER.info("为id、key 为{}的数据做了缓存", s.getId());
        return s;
    }

    @Override
    @CacheEvict(value = "student")
    // @CacheEvict 从缓存 student 中删除 key 为 id 的数据
    public void deleteStudentById(Integer id) {
        LOGGER.info("删除了id、key 为{}的数据缓存", id);
        //studentRepository.deleteById(id);
    }

    @Override
    @Cacheable(value = "student",key = "#id")
    // @Cacheable 缓存 key 为 id 的数据到缓存 student 中
    public Student findStudentById(Integer id) {
        Student s = studentRepository.findById(id).get();
        LOGGER.info("为id、key 为{}的数据做了缓存", id);
        return  s;
    }
}
```

代码讲解看注释，很详细。

7、controller 层

```
package com.nasus.cache.controller;

import com.nasus.cache.entity.Student;
import com.nasus.cache.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.Mapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/student")
public class StudentController {

    @Autowired
    private StudentService studentService;

    @PostMapping("/put")
    public Student saveStudent(@RequestBody Student student){
        return studentService.saveStudent(student);
    }

    @DeleteMapping("/evit/{id}")
    public void deleteStudentById(@PathVariable("id") Integer id){
        studentService.deleteStudentById(id);
    }

    @GetMapping("/able/{id}")
    public Student findStudentById(@PathVariable("id") Integer id){
        return studentService.findStudentById(id);
    }

}
```

8、application 开启缓存功能

```
package com.nasus.cache;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;

@EnableCaching // 开启缓存功能
@SpringBootApplication
public class CacheApplication {

    public static void main(String[] args) {
        SpringApplication.run(CacheApplication.class, args);
    }
}
```

05 测试

测试前，先看一眼数据库当前的数据，如下：

id	age	name
2	24	詹姆斯
3	24	高斯林

1、测试 @Cacheable

访问 http://localhost:8080/student/able/2 控制台打印出了 SQL 查询语句，以及指定日志。说明一次程序是直接查询数据库得到的结果。

```
2019-02-21 22:54:54.651  INFO 1564 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
   : Completed initialization in 11 ms
Hibernate: select student0_.id as id1_0_0_, student0_.age as age2_0_0_, student0_.name as na
e3_0_0_ from student student0_ where student0_.id=?
2019-02-21 22:54:59.725  INFO 1564 --- [nio-8080-exec-1] c.n.c.service.impl.StudentServiceI
pl    : 为id、key 为2的数据做了缓存
```

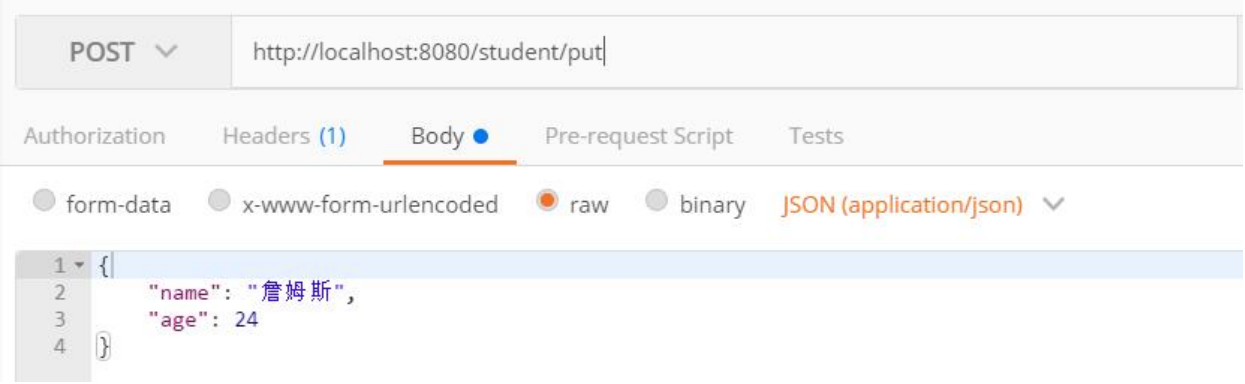
postman 第一次测试结果：

再次访问 http://localhost:8080/student/able/2 结果如下图。但控制台无 SQL 语句打印，也无**为i、key 为2的数据做了缓存**这句话输出。

说明 @Cacheable 确实做了数据缓存，第二次的测试结果是从数据缓存中获取的，并没有直接查数库。

2、测试 @CachePut

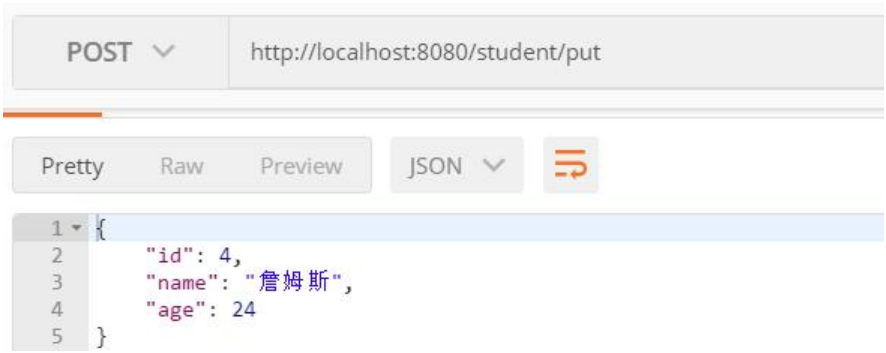
如下图，postman 访问 http://localhost:8080/student/put 插入数据：



下面是控制台打印出了 SQL Insert 插入语句，以及指定日志。说明程序做了缓存。

```
Hibernate: insert into student (age, name, id) values (?, ?, ?)
2019-02-21 23:12:03.688  INFO 1564 --- [nio-8080-exec-8] c.n.c.service.impl.StudentServiceImpl    : 为id、key 为4的数据做了缓存
```

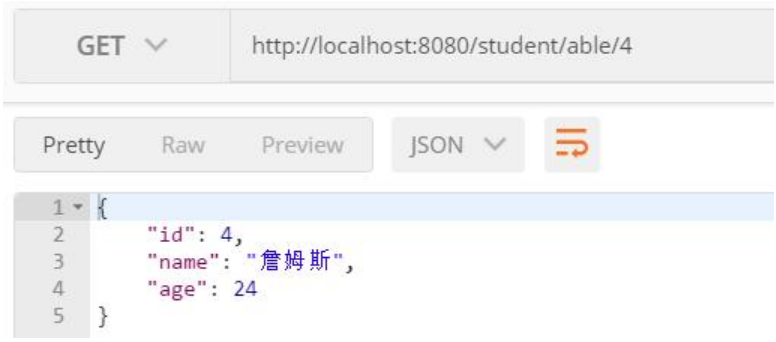
插入数据返回的结果：



数据库中的结果：

id	age	name
2	24	詹姆斯
3	24	高斯林
4	24	詹姆斯

访问 http://localhost:8080/student/able/4 Postman 结果如下图。控制台无输出，验证了 @CachePut 确实做了缓存，下图数据是从缓存中获取的。

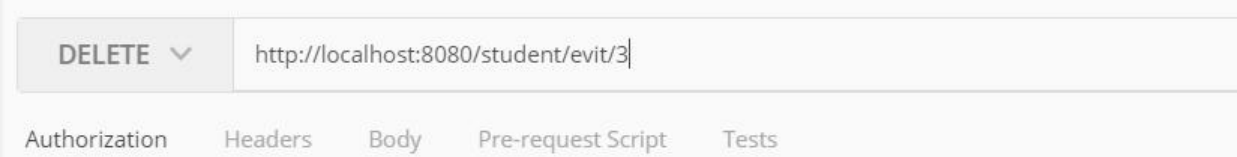


3、测试 @CacheEvict

postman 访问 http://localhost:8080/student/able/3 为 id = 3 的数据做缓存。

postman 再次访问 http://localhost:8080/student/able/3 确认数据是从缓存中获取的。

postman 访问 http://localhost:8080/student/evit/3



从缓存中删除 key 为 3 的缓存数据：

```
Hibernate: select student0_.id as id1_0_0_, student0_.age as age2_0_0_, student0_.name as na
e3_0_0_ from student student0_ where student0_.id=?
2019-02-21 23:26:08.516  INFO 8612 --- [nio-8080-exec-2] c.n.c.service.impl.StudentServiceI
pl    : 为id、key 为3的数据做了缓存
2019-02-21 23:27:01.508  INFO 8612 --- [nio-8080-exec-4] c.n.c.service.impl.StudentServiceI
pl    : 删除了id、key 为3的数据缓存
```

再次 postman 访问 http://localhost:8080/student/able/3 观察后台，重新做了数据缓存：

```
Hibernate: select student0_.id as id1_0_0_, student0_.age as age2_0_0_, student0_.name as na
e3_0_0_ from student student0_ where student0_.id=?
2019-02-21 23:27:12.320  INFO 8612 --- [nio-8080-exec-5] c.n.c.service.impl.StudentServiceI
pl    : 为id、key 为3的数据做了缓存
```

这一套测试流程下来，证明了 @CacheEvict 确实删除了数据缓存。

06 源码下载

https://github.com/turoDog/Demo/tree/master/springboot_cache_demo

07 切换缓存技术

切换缓存技术除了在 pom 文件加入相关依赖包配置以外，使用方式与上面的代码演示一样。

1、切换 EhCache

在 pom 中添加 Encache 依赖：

```
<!-- EhCache 依赖 -->
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
</dependency>
```

Ehcache 所需配置文件 ehcache.xml 只需放在类路径（resource 目录）下，SpringBoot 会自动扫，如：

```
<?xml version="1.0" encoding="UTF-8">
<ehcache>
   <cache name="student" maxElementsInMmory="1000">
```

```
<ehcache>
```

SpringBoot 会自动配置 EhcacheManager 的 Bean。

2、切换 Guava

只需在 pom 中加入 Guava 依赖即可：

```
<!-- GuavaCache 依赖 -->
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>18.0</version>
</dependency>
```

SpringBoot 会自动配置 GuavaCacheManager 的 Bean。

3、切换 RedisCache

与 Guava 一样，只需在 pom 加入依赖即可：

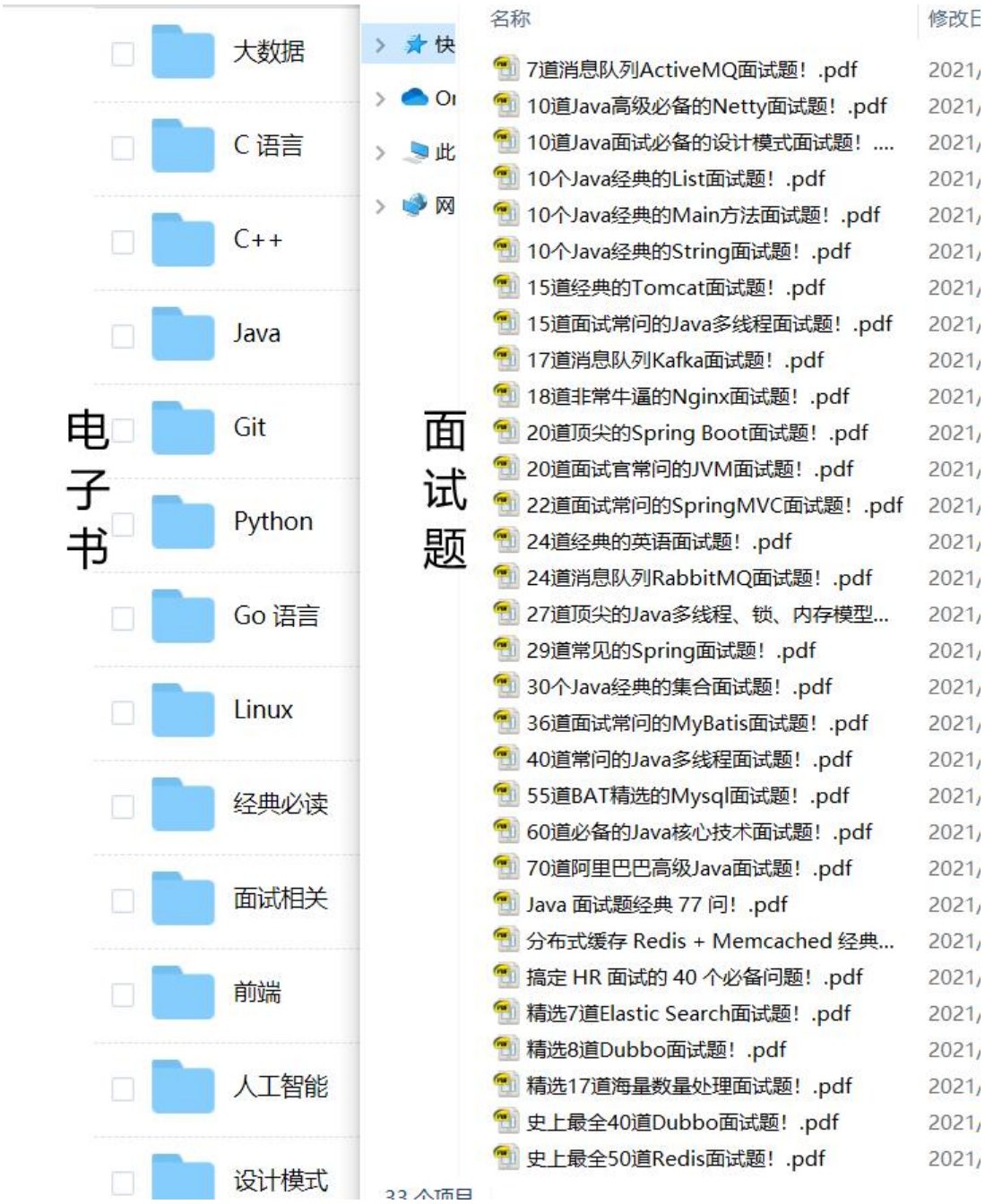
```
<!-- cache 依赖 -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-redis</artifactId>
</dependency>
```

SpringBoot 会自动配置 RedisCacheManager 以及 RedisTemplate 的 Bean。

此外，切换其他缓存技术的方式也是类似。这里不做赘述。

08 后语

以上为 SpringBoot 数据缓存的教程。



原文链接：SpringBoot 入门教程 (十一) | 整合数据缓存 Cache

JavaFish

微信扫描二维码，关注我的公众号

原文链接：SpringBoot 入门教程 (十一) | 整合数据缓存 Cache