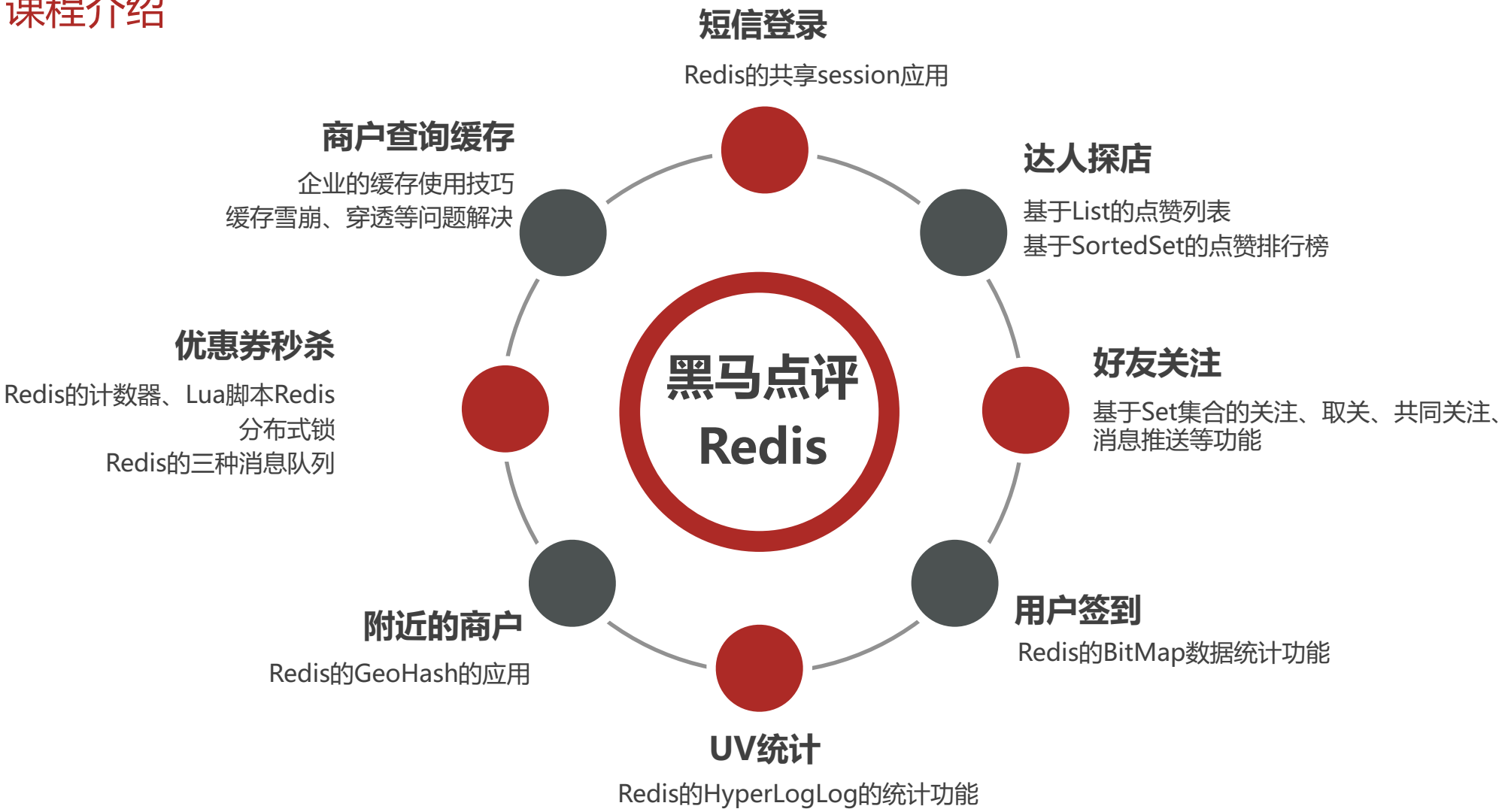
Redis企业实战

Redis的企业应用案例

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

今日课程介绍


短信登录
Redis的共享session应用
商户查询缓存
企业的缓存使用技巧
缓存雪崩、穿透等问题解决
达人探店
基于List的点赞列表
基于SortedSet的点赞排行榜
优惠券秒杀
Redis的计数器、Lua脚本Redis
分布式锁
Redis的三种消息队列
黑马点评
Redis
好友关注
基于Set集合的关注、取关、共同关注、
消息推送等功能
附近的商户
Redis的GeoHash的应用
用户签到
Redis的BitMap数据统计功能
UV统计
Redis的HyperLogLog的统计功能

目录

Contents

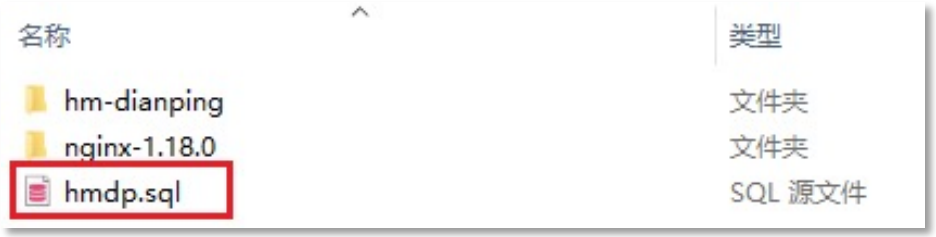
01 短信登录

目录

Contents

导入黑马点评项目

首先，导入课前资料提供的SQL文件：



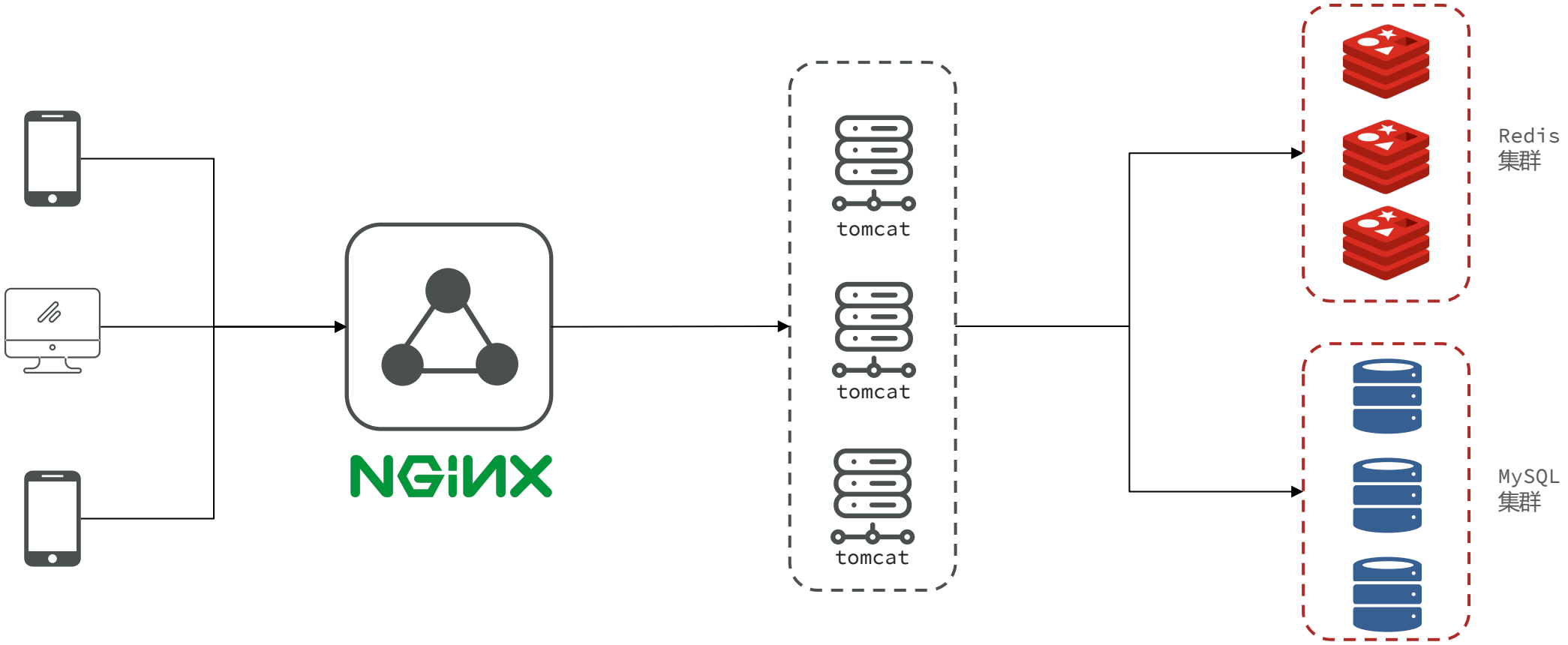
其中的表有：

- tb_user：用户表
- tb_user_info：用户详情表
- tb_shop：商户信息表
- tb_shop_type：商户类型表
- tb_blog：用户日记表（达人探店日记）
- tb_follow：用户关注表
- tb_voucher：优惠券表
- tb_voucher_order：优惠券的订单表

注意

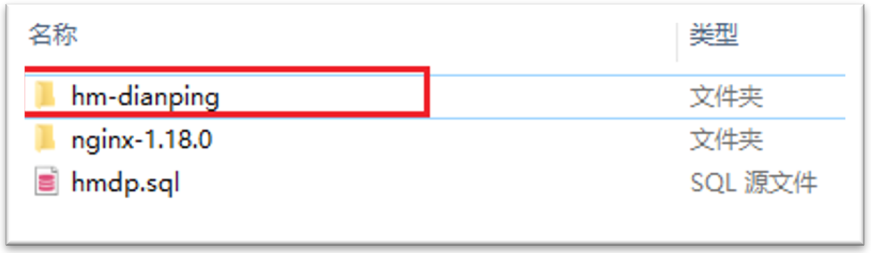
Mysql的版本采用5.7及以上版本

导入黑马点评项目



导入后端项目

在资料中提供了一个项目源码：



将其复制到你的idea工作空间，然后利用idea打开即可：

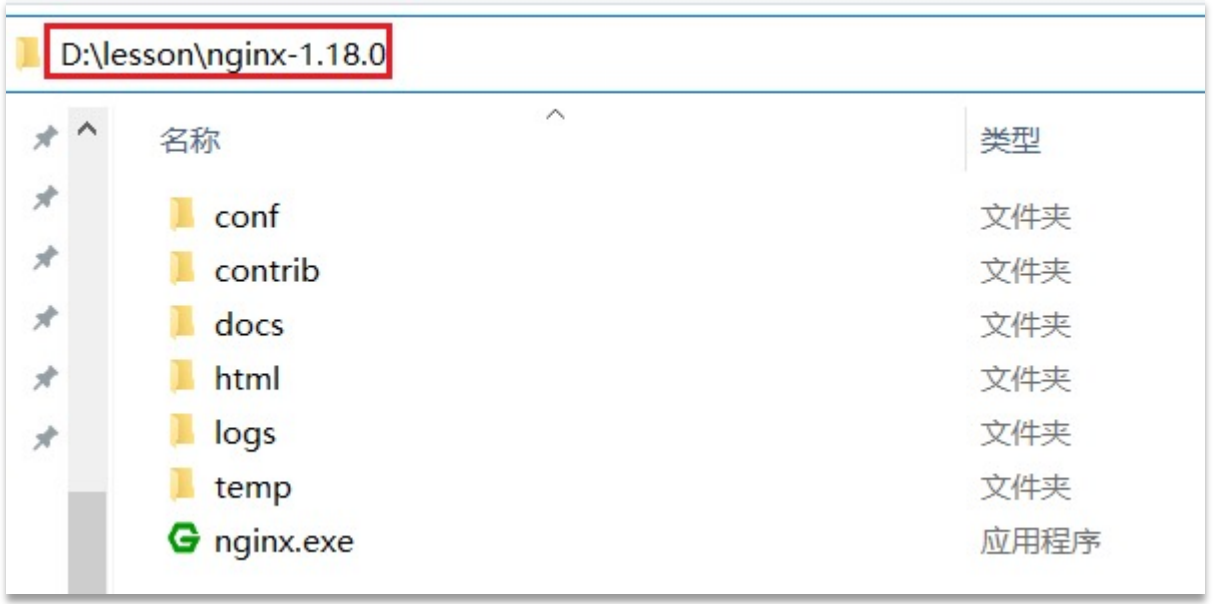
启动项目后，在浏览器访问：http://localhost:8081/shop-type/list ，如果可以看到数据则证明运行没有问题

> **注意**
>
> 不要忘了修改application.yaml文件中的mysql、redis地址信息

导入前端项目

在资料中提供了一个nginx文件夹：

将其复制到任意目录，要确保该目录不包含中文、特殊字符和空格，例如：



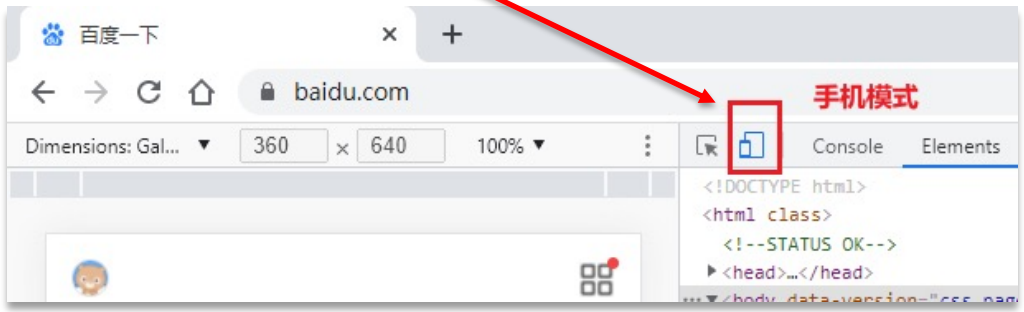
运行前端项目

在nginx所在目录下打开一个CMD窗口，输入命令：

```
start nginx.exe
```

打开chrome浏览器，在空白页面点击鼠标右键，选择检查，即可打开开发者工具：

然后打开手机模式：

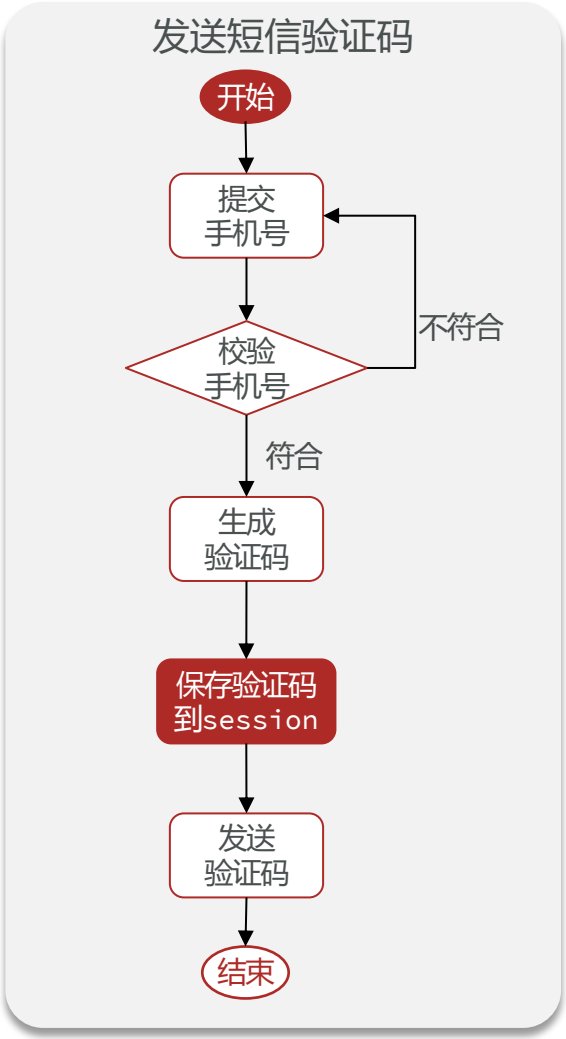


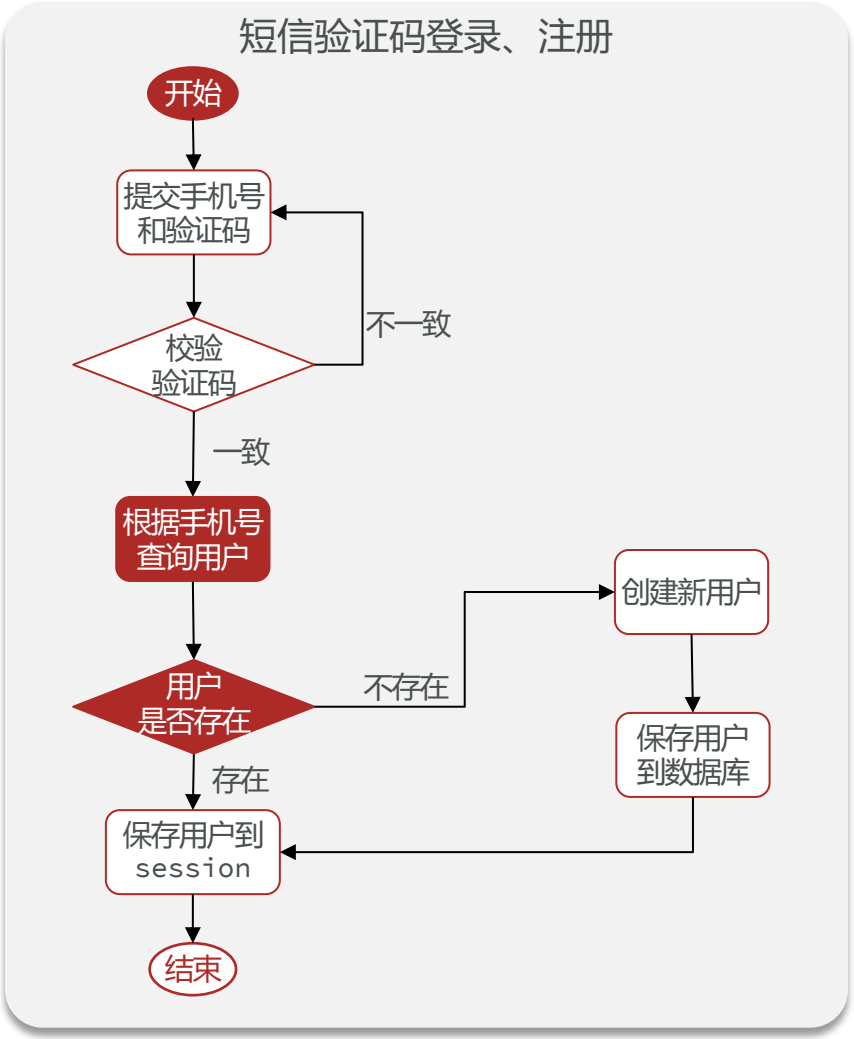
然后访问: http://127.0.0.1:8080 ，即可看到页面：

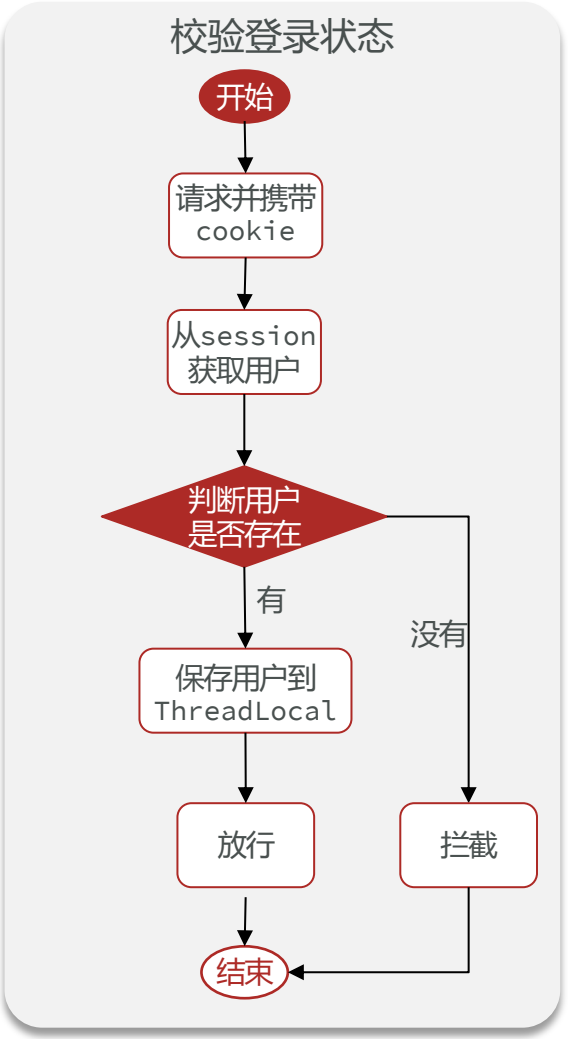
目录

Contents

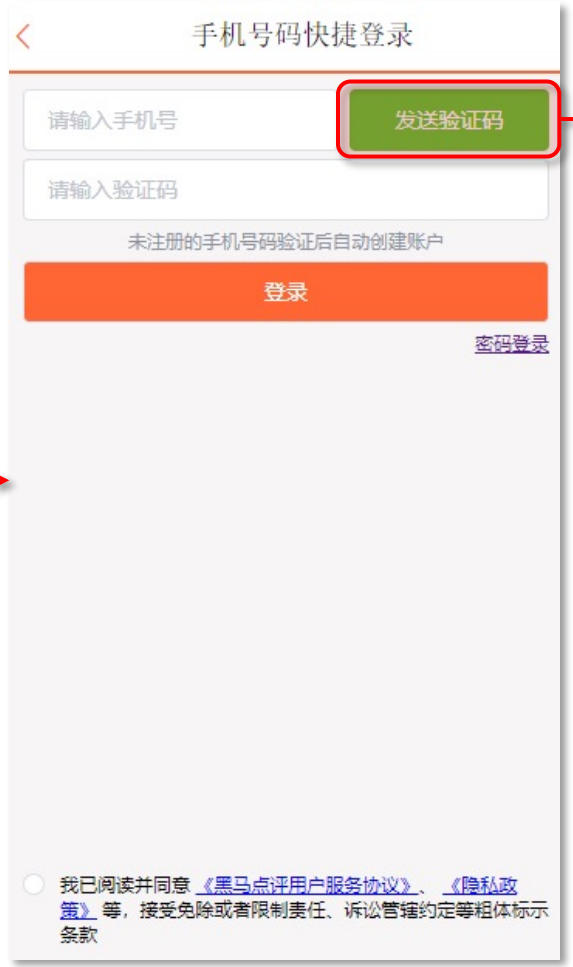
基于Session实现登录





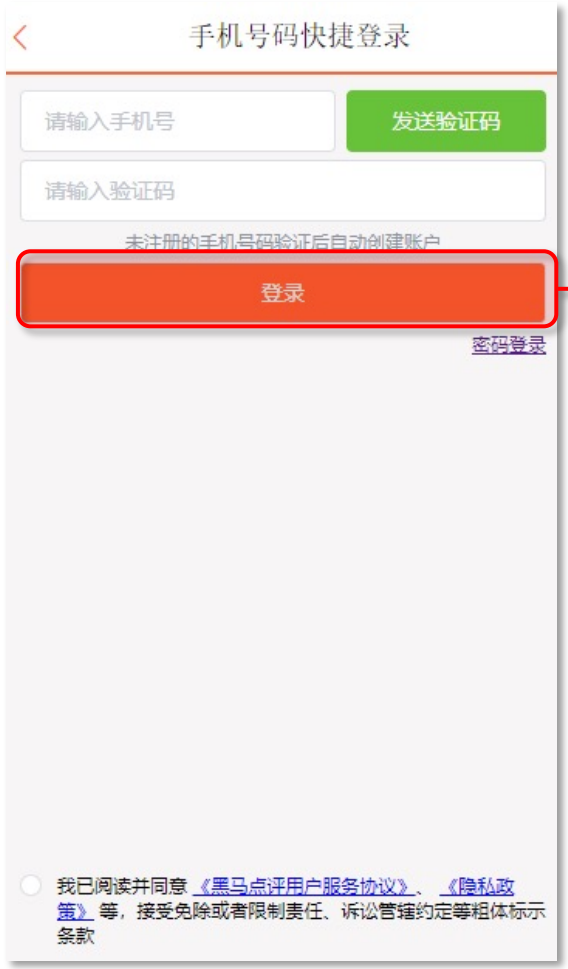


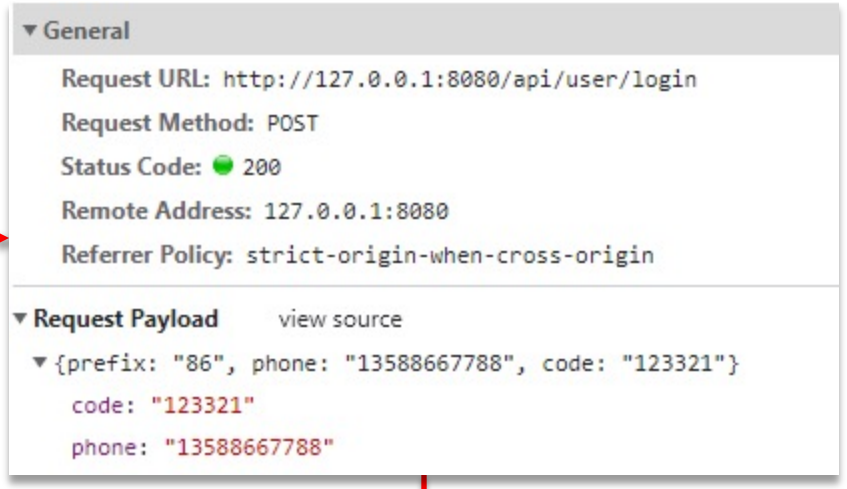
发送短信验证码



	说明
请求方式	POST
请求路径	/user/code
请求参数	phone,电话号码
返回值	无

短信验证码登录





	说明
请求方式	POST
请求路径	/user/login
请求参数	phone:电话号码；code:验证码
返回值	无

登录验证功能

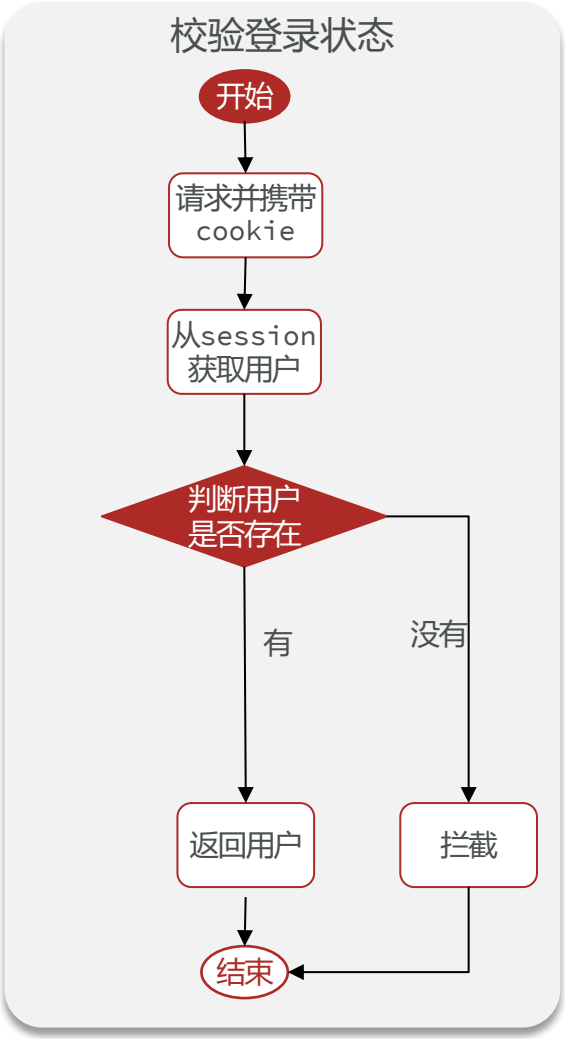
General

Request URL: http://localhost:8080/api/user/me
Request Method: GET
Status Code: 200
Remote Address: 127.0.0.1:8080
Referrer Policy: strict-origin-when-cross-origin

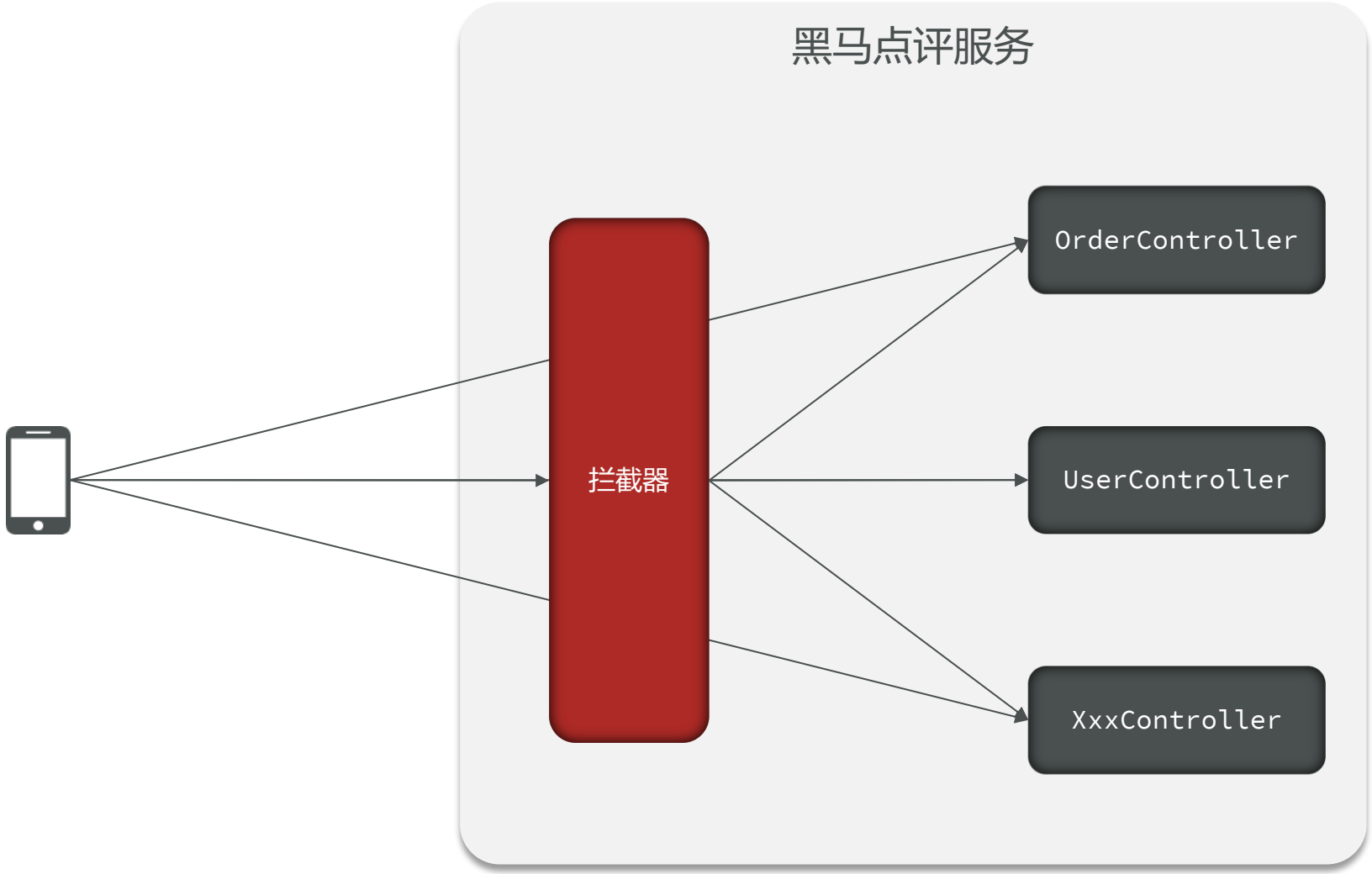
Response Headers (5)

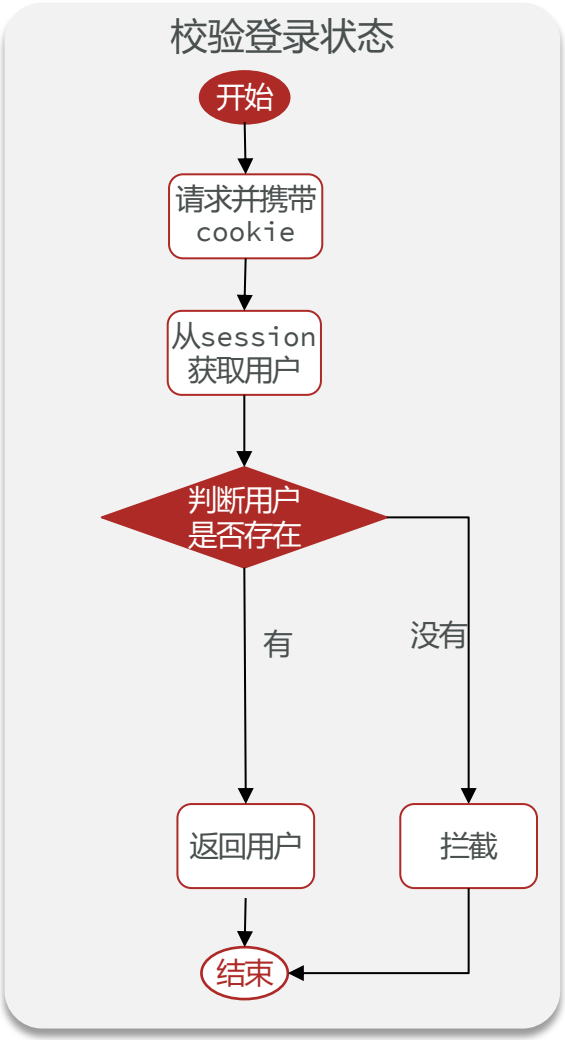
Request Headers View source

Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en-US;q=0.8,en;q=0.7
Connection: keep-alive
Cookie: Idea-2e2d9f91=4ee107b6-b215-4cec-9d9e-b6804781dffd; JSESSIONID=40DF9A8FA7A0BDA229C5B41B2054092D
Host: localhost:8080
Referer: http://localhost:8080/info.html

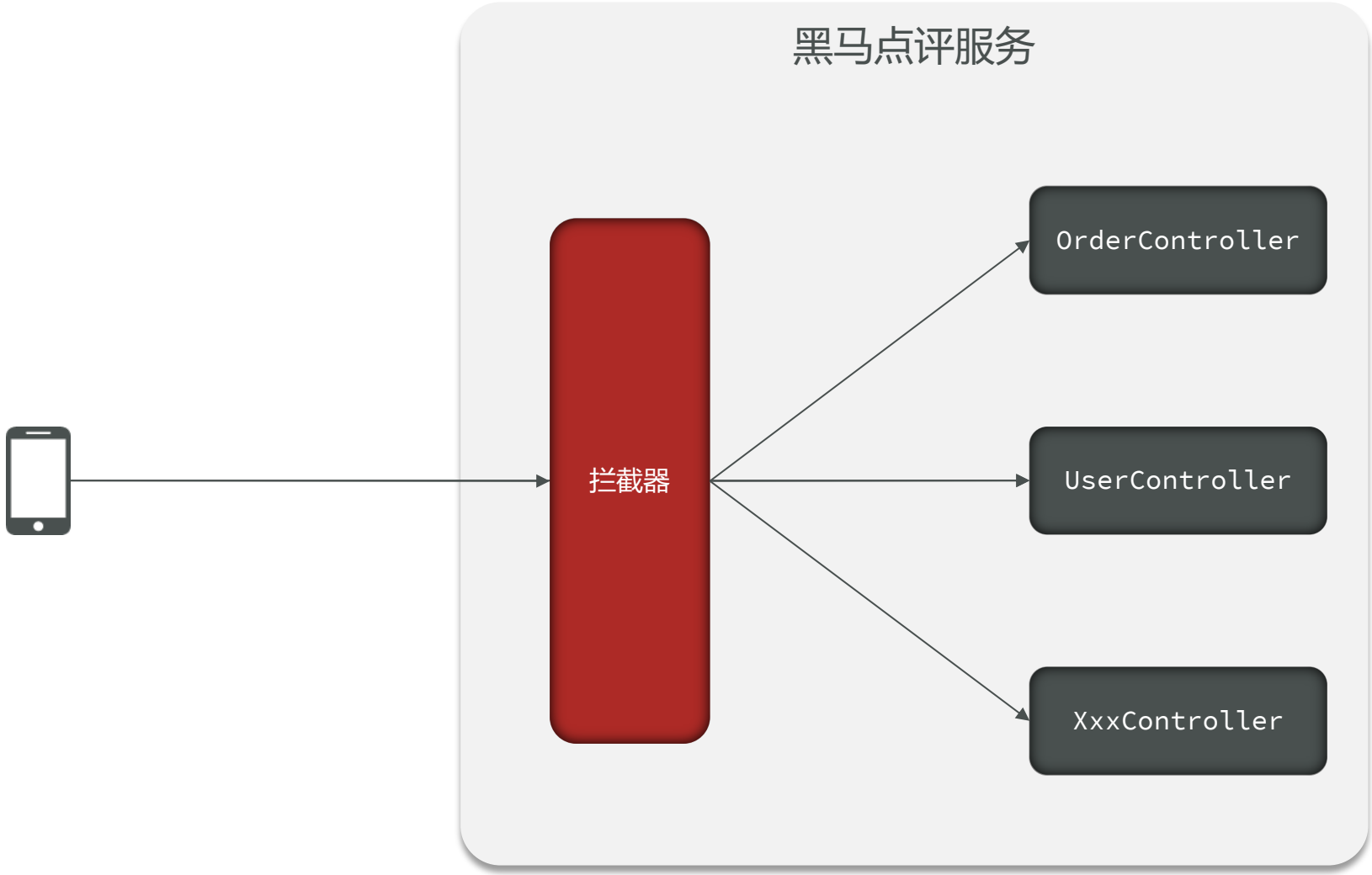


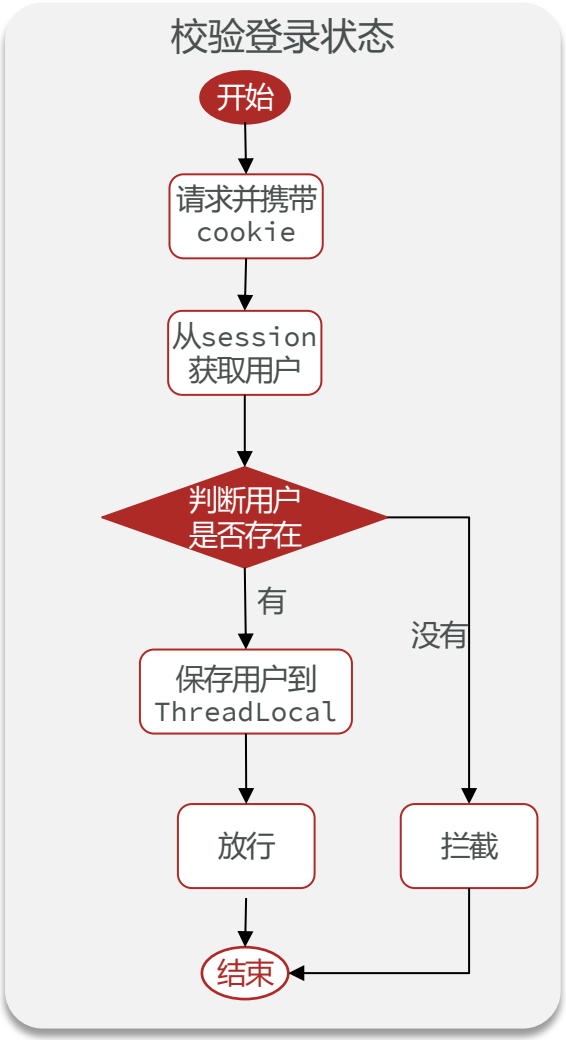
登录验证功能


黑马点评服务
拦截器
OrderController
UserController
XxxController


校验登录状态
开始
请求并携带cookie
从session获取用户
判断用户是否存在
有
没有
返回用户
拦截
结束

登录验证功能


黑马点评服务
拦截器
OrderController
UserController
XxxController


校验登录状态
开始
请求并携带cookie
从session获取用户
判断用户是否存在
有
没有
保存用户到ThreadLocal
放行
拦截
结束

目录

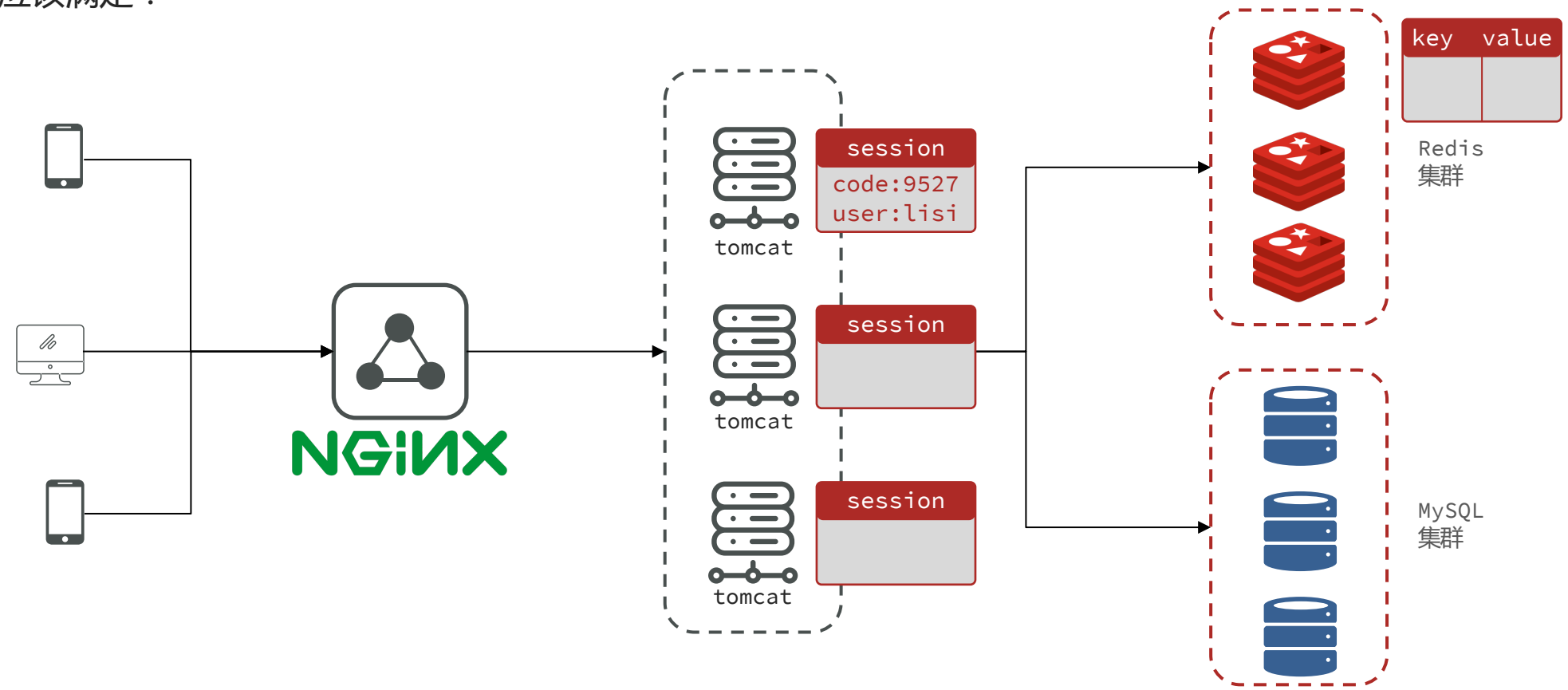
Contents

集群的session共享问题

session共享问题：多台Tomcat并不共享session存储空间，当请求切换到不同tomcat服务时导致数据丢失的问题。

session的替代方案应该满足：

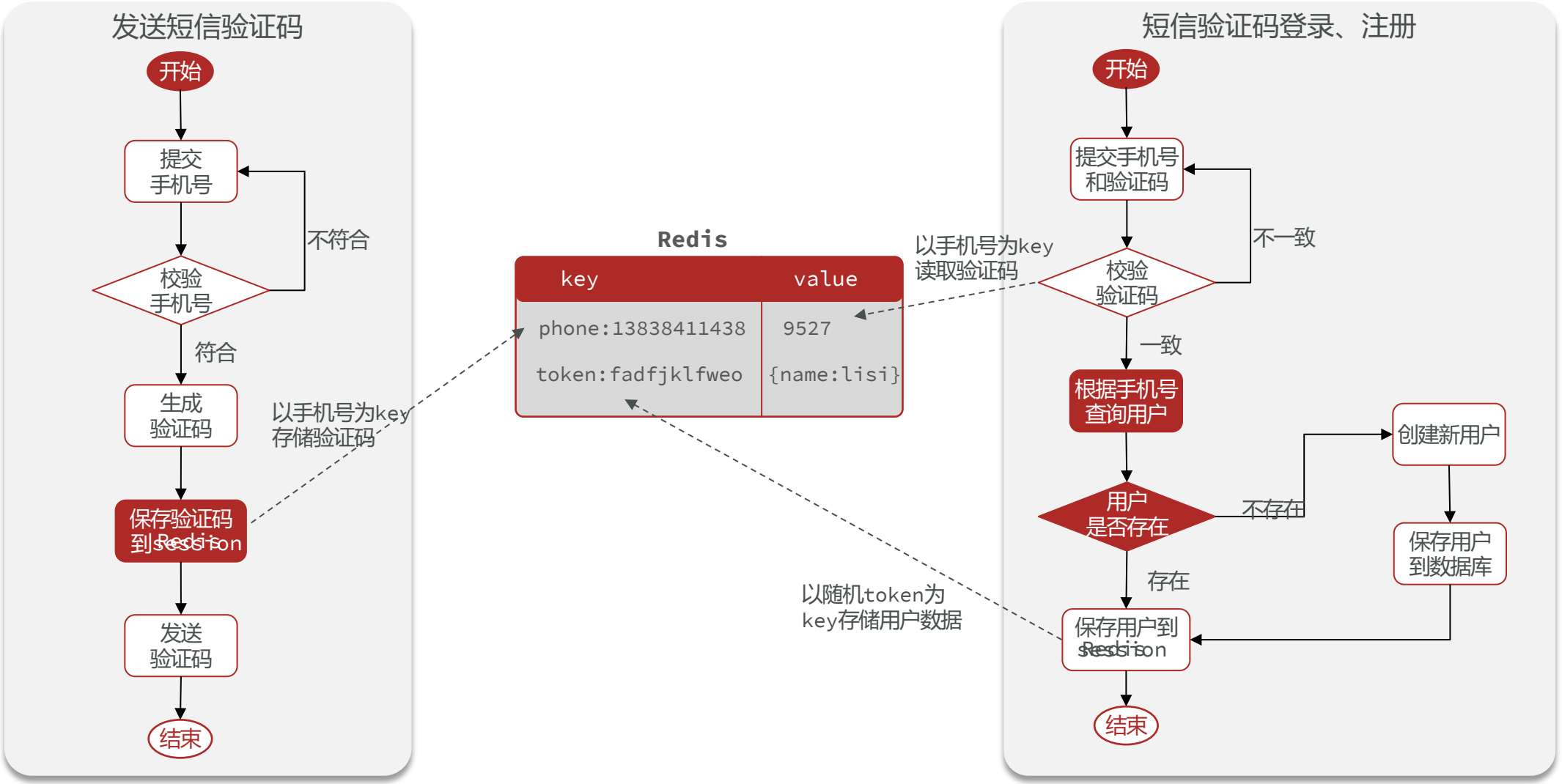
- 数据共享
- 内存存储
- key、value结构



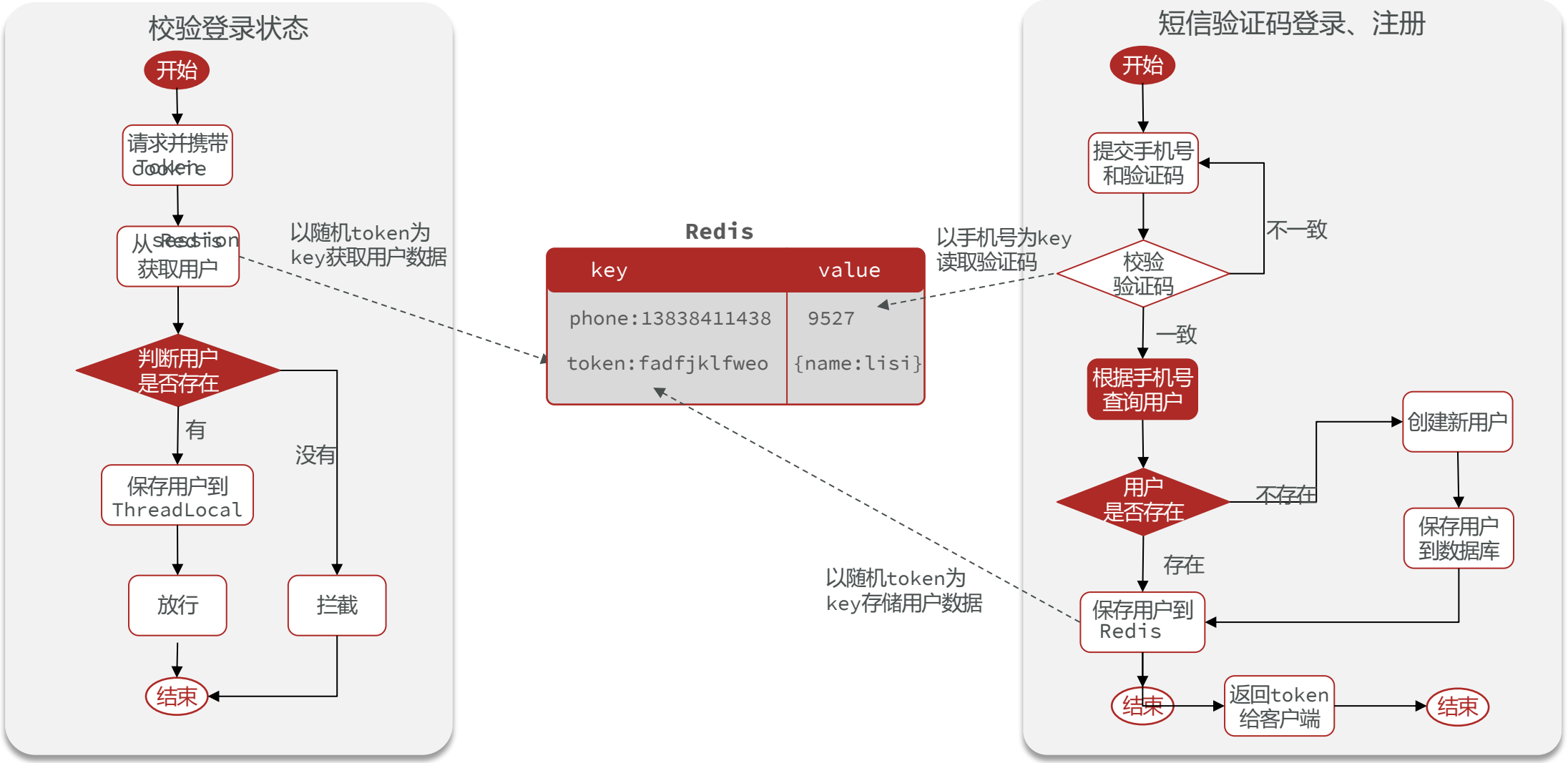
目录

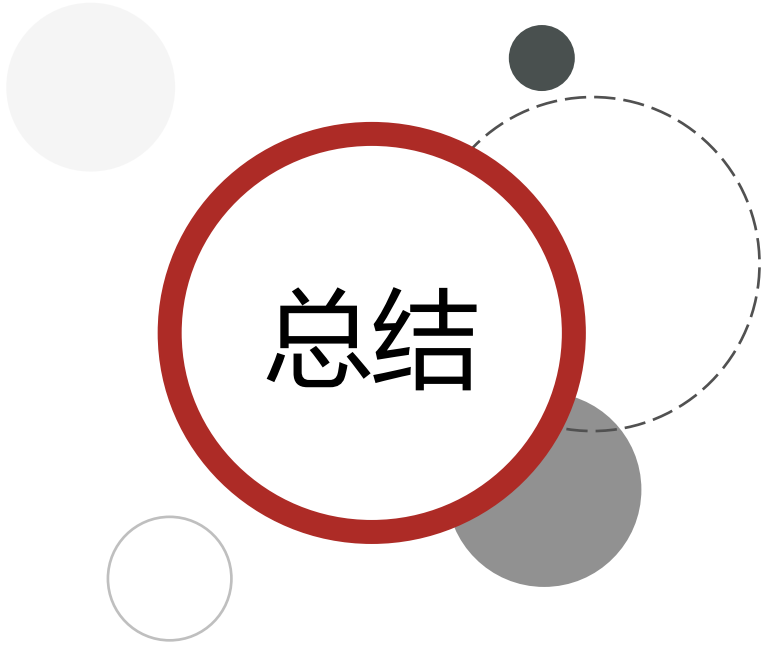
Contents

基于Redis实现共享session登录



基于Redis实现共享session登录





Redis代替session需要考虑的问题：

- 选择合适的数据结构
- 选择合适的key
- 选择合适的存储粒度

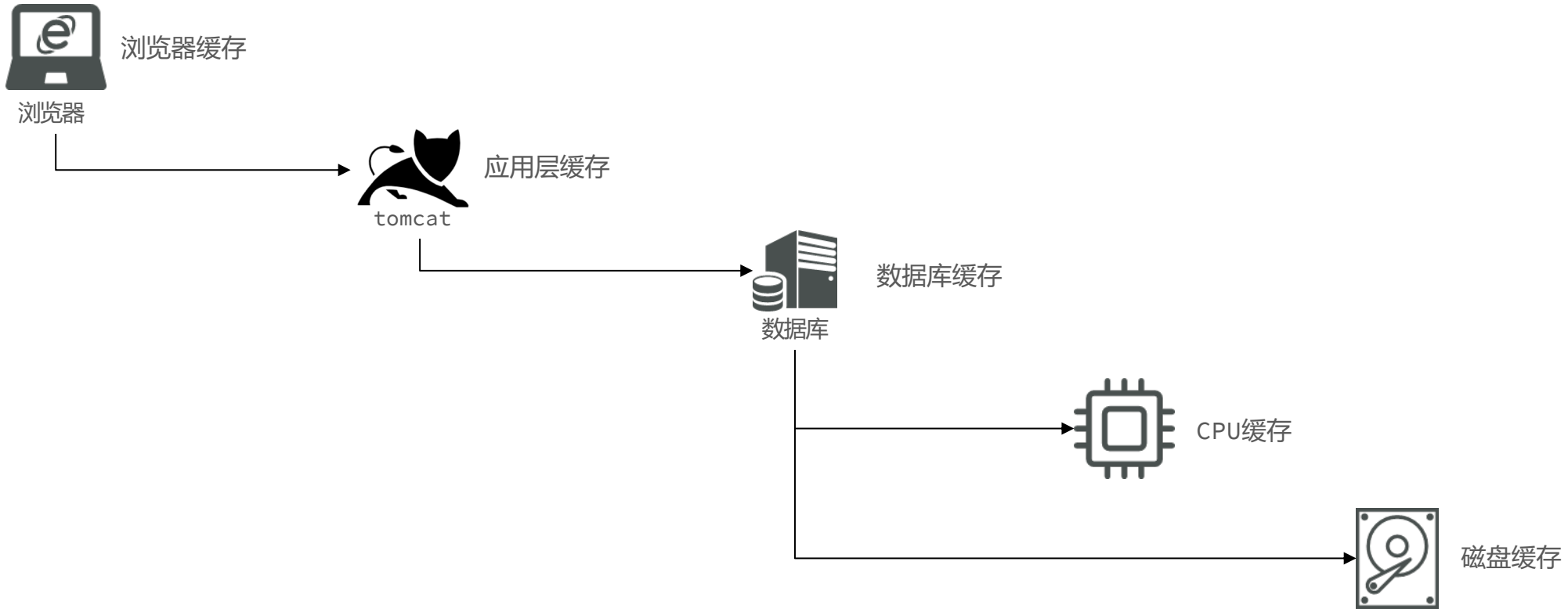
02 商户查询缓存

目录

Contents

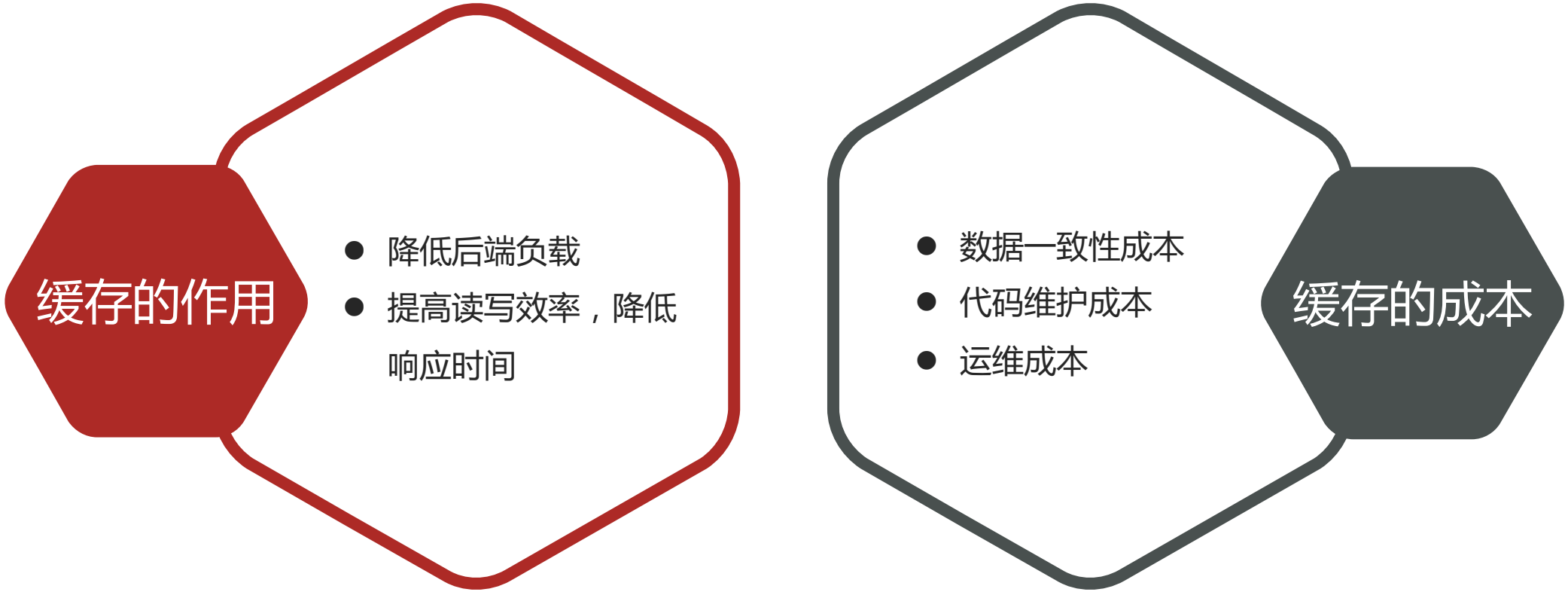
什么是缓存

缓存就是数据交换的缓冲区（称作Cache），是存贮数据的临时地方，一般读写性能较高。



什么是缓存

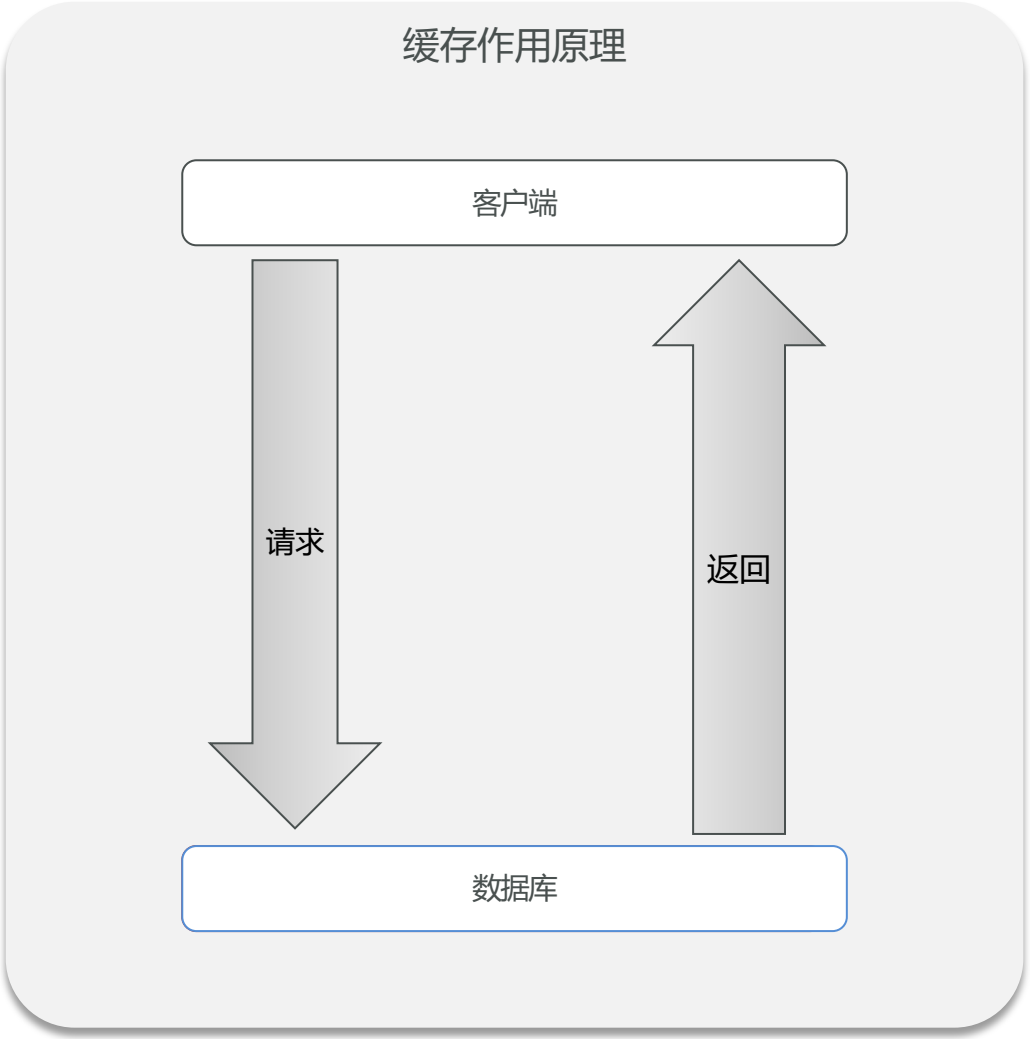
缓存就是数据交换的缓冲区（称作Cache），是存贮数据的临时地方，一般读写性能较高。



目录

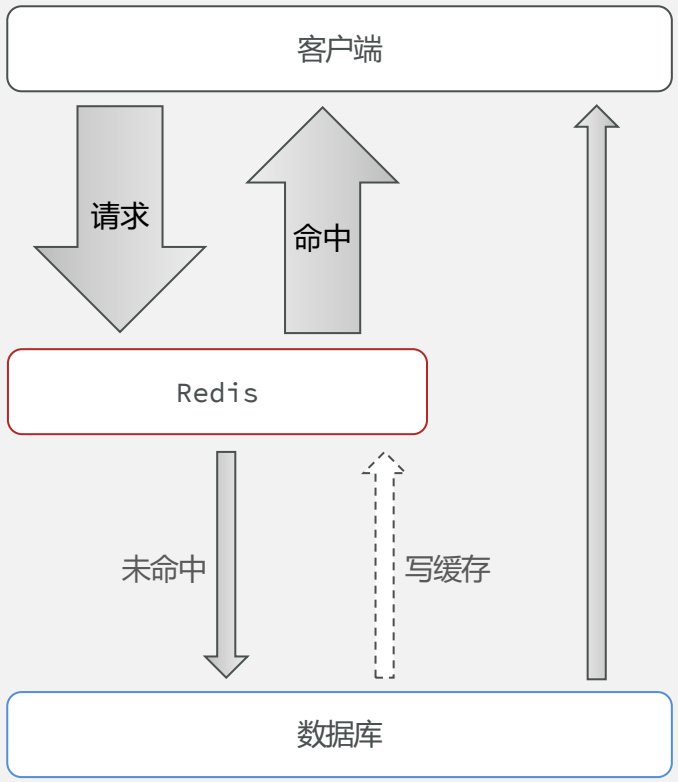
Contents

添加Redis缓存

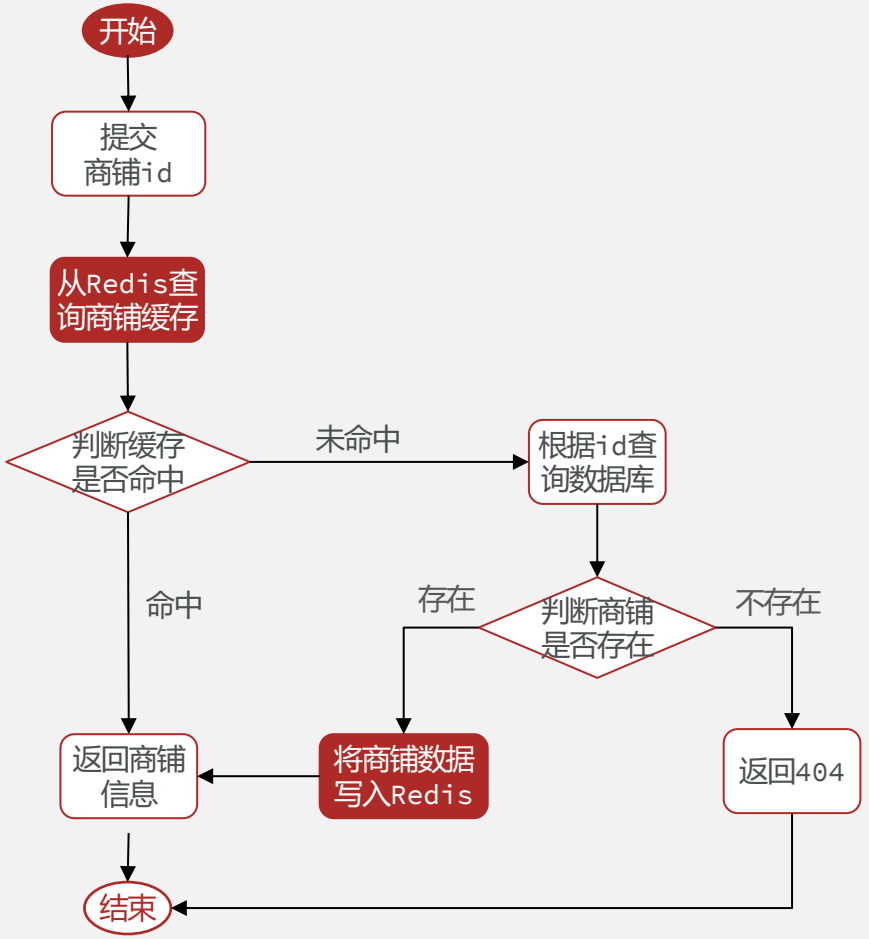


添加Redis缓存

缓存作用原理



根据id查询商铺缓存的流程



练习

使用Hash结构来缓存商铺信息

需求：

- 修改缓存存储的逻辑，用hash结构来替代JSON字符串
- 商铺的每一个字段作为一个hash结构的field

练习 给店铺类型查询业务添加缓存

店铺类型在首页和其它多个页面都会用到，如图：

需求：修改ShopTypeController中的queryTypeList方法，添加查询缓存

```
public class ShopTypeController {
    @Resource
    private IShopTypeService typeService;

    @GetMapping("list")
    public Result queryTypeList() {
        List<ShopType> typeList = typeService
                .query().orderByAsc( column: "sort").list();
        return Result.ok(typeList);
    }
}
```

目录

Contents

缓存更新策略

	内存淘汰	超时剔除	主动更新
说明			
一致性			
维护成本			

业务场景：

- 低一致性需求：使用内存淘汰机制。例如店铺类型的查询缓存
- 高一致性需求：主动更新，并以超时剔除作为兜底方案。例如店铺详情查询的缓存

主动更新策略

01

Cache Aside Pattern

由缓存的调用者，在更新数据库的同时更新缓存

02

Read/Write Through Pattern

缓存与数据库整合为一个服务，由服务来维护一致性。调用者调用该服务，无需关心缓存一致性问题。

03

Write Behind Caching Pattern

调用者只操作缓存，由其它线程异步的将缓存数据持久化到数据库，保证最终一致。

主动更新策略

01

Cache Aside Pattern

由缓存的调用者，在更新数据库的同时更新缓存

操作缓存和数据库时有三个问题需要考虑：

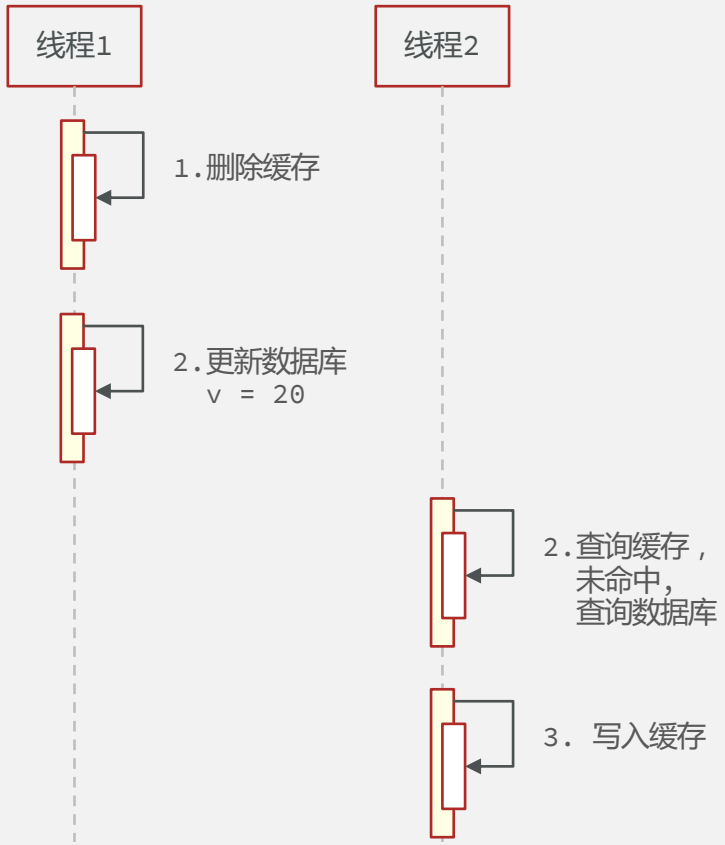
1. 如何保证缓存与数据库的操作的同时成功或失败？
   - 单体系统，将缓存与数据库操作放在一个事务
   - 分布式系统，利用TCC等分布式事务方案
2. 删除缓存还是更新缓存？
   - 更新缓存：每次更新数据库都更新缓存，无效写操作较多 ✗
   - 删除缓存：更新数据库时让缓存失效，查询时再更新缓存 ✓
3. 先操作缓存还是先操作数据库？
   - 先删除缓存，再操作数据库
   - 先操作数据库，再删除缓存

Cache Aside Pattern

缓存 20

数据库 20

先删除缓存，再操作数据库

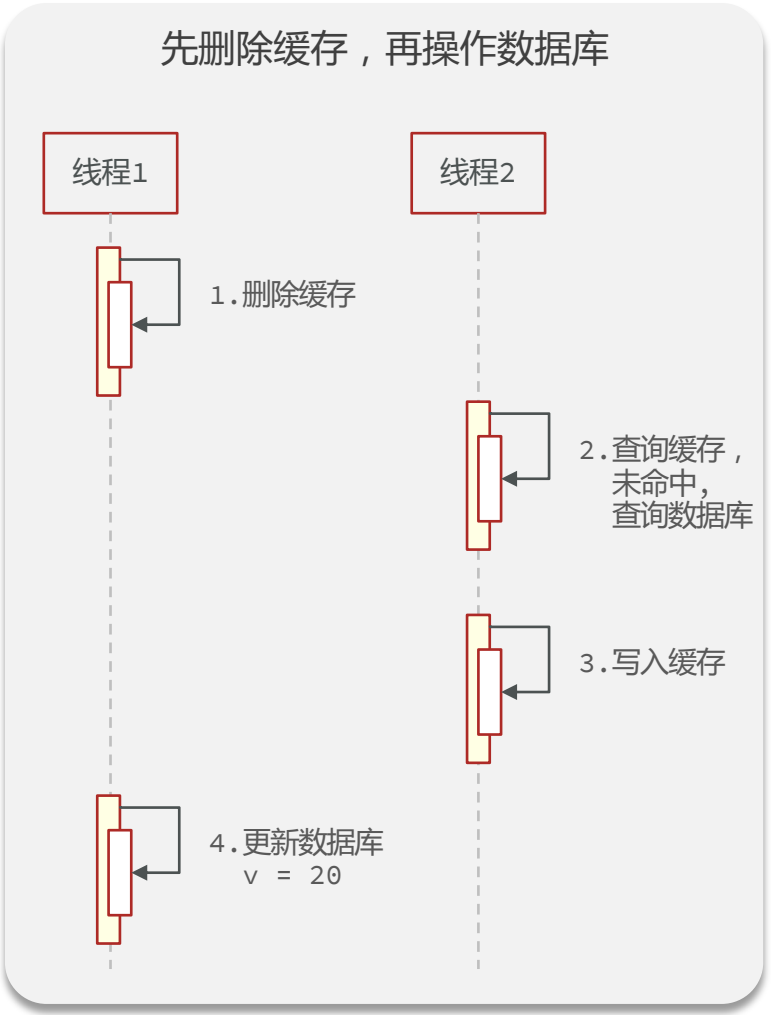


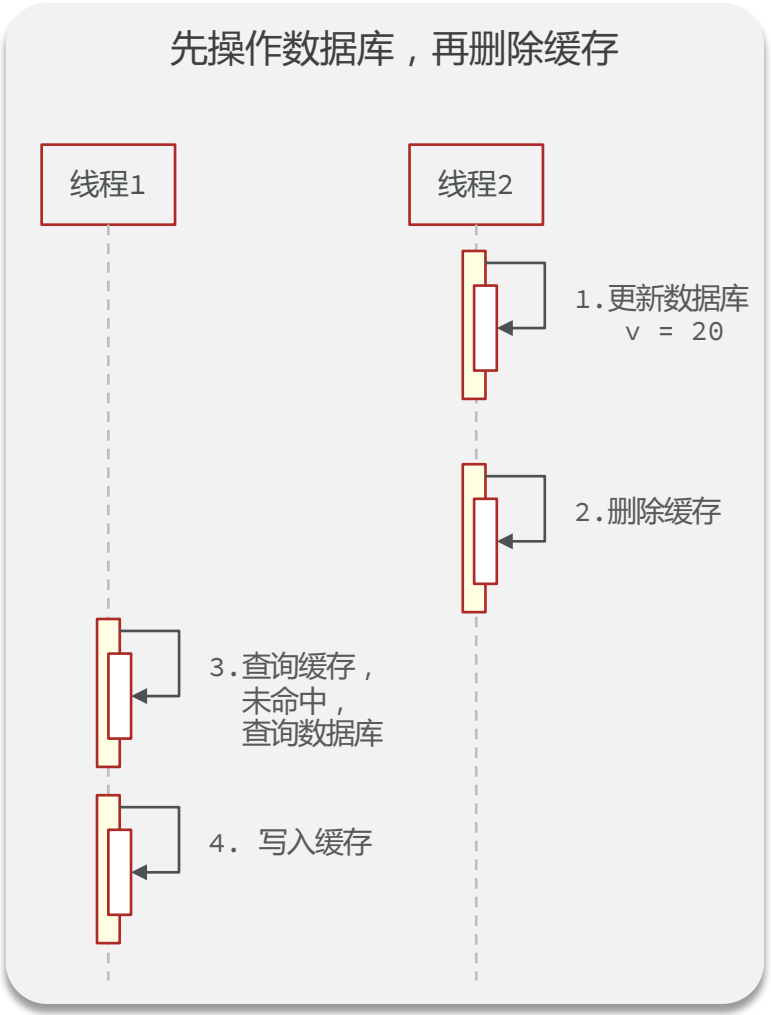
先操作数据库，再删除缓存

Cache Aside Pattern

缓存 10

数据库 10

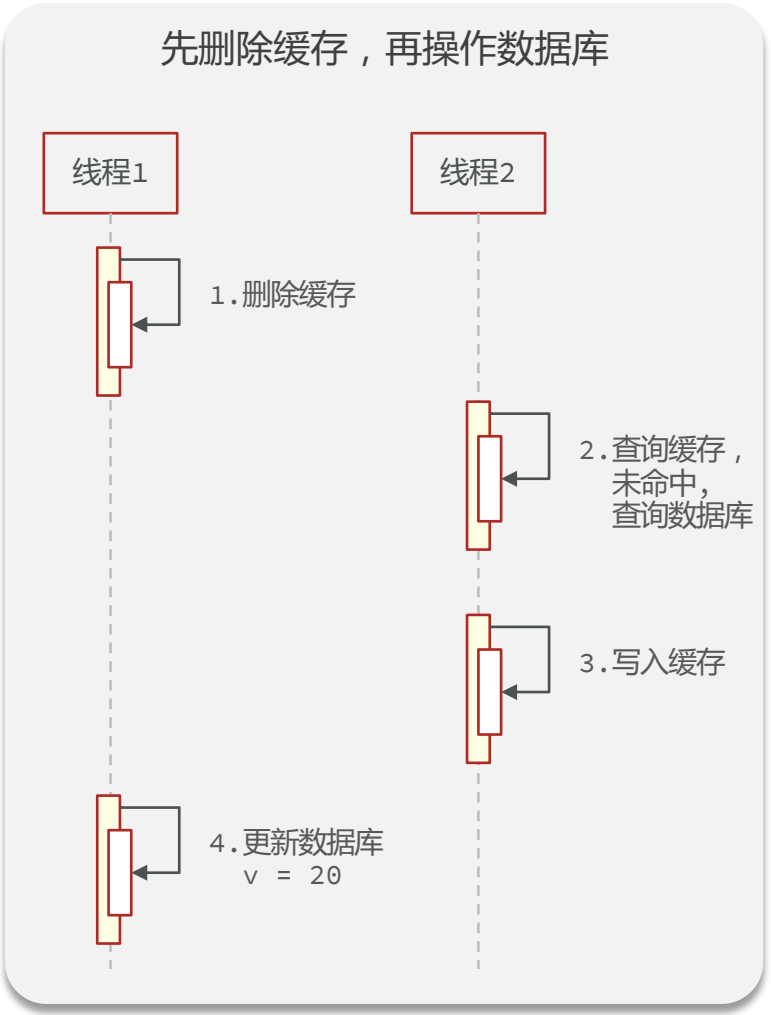


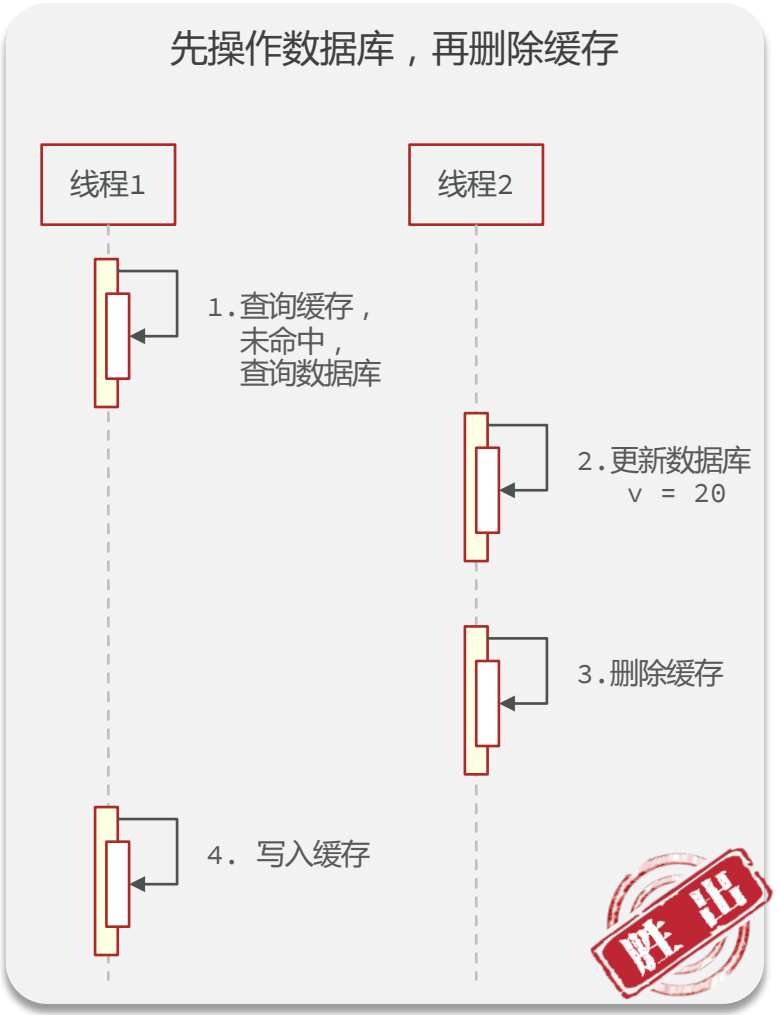


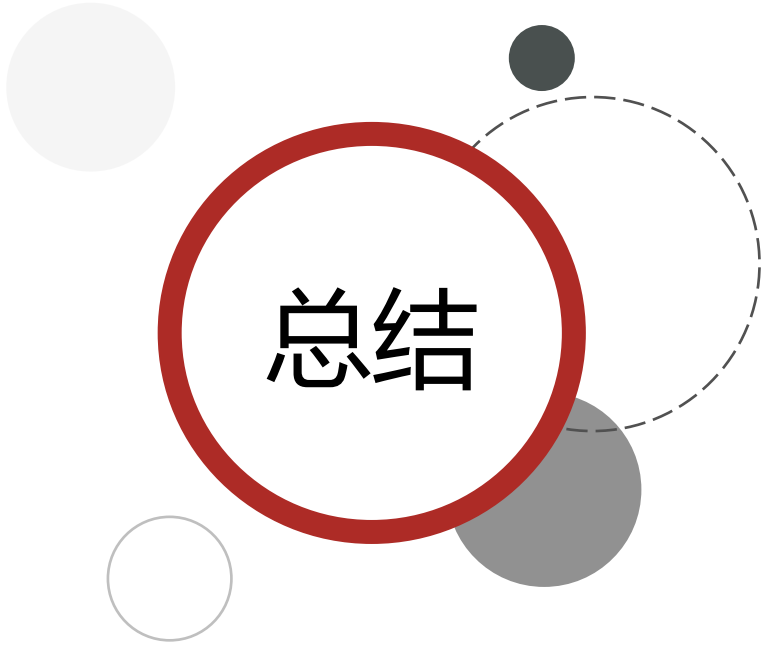
Cache Aside Pattern

缓存 10

数据库 20







缓存更新策略的最佳实践方案：

1. 低一致性需求：使用Redis自带的内存淘汰机制
2. 高一致性需求：主动更新，并以超时剔除作为兜底方案
   - ◆ 读操作：
     - 缓存命中则直接返回
     - 缓存未命中则查询数据库，并写入缓存，设定超时时间
   - ◆ 写操作：
     - 先写数据库，然后再删除缓存
     - 要确保数据库与缓存操作的原子性

案例 给查询商铺的缓存添加超时剔除和主动更新的策略

修改ShopController中的业务逻辑，满足下面的需求：

① 根据id查询店铺时，如果缓存未命中，则查询数据库，将数据库结果写入缓存，并设置超时时间

② 根据id修改店铺时，先修改数据库，再删除缓存

目录

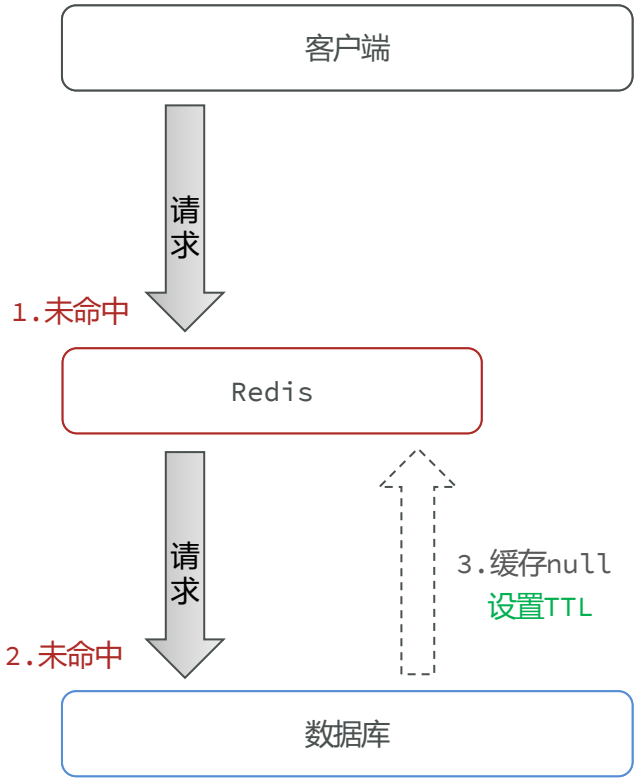
Contents

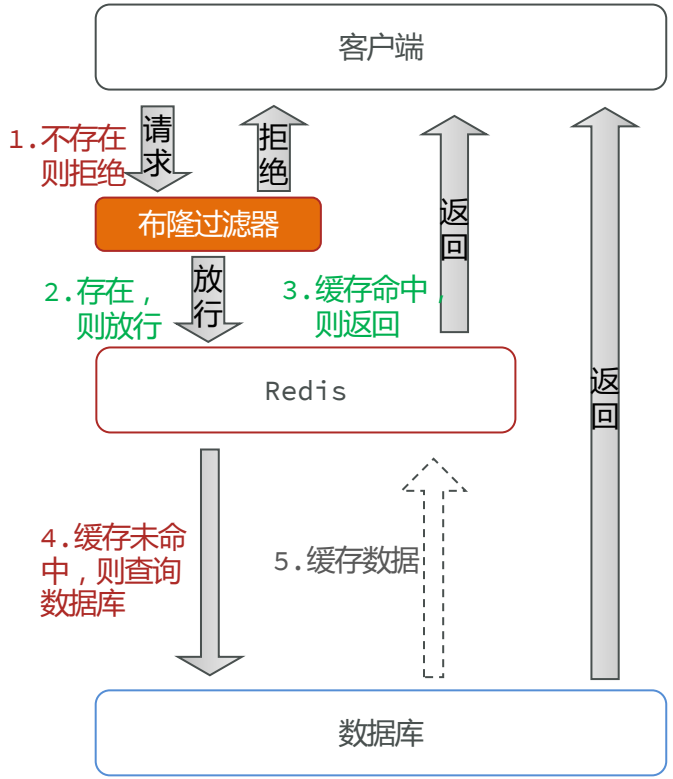
缓存穿透

缓存穿透是指客户端请求的数据在缓存中和数据库中都不存在，这样缓存永远不会生效，这些请求都会打到数据库。

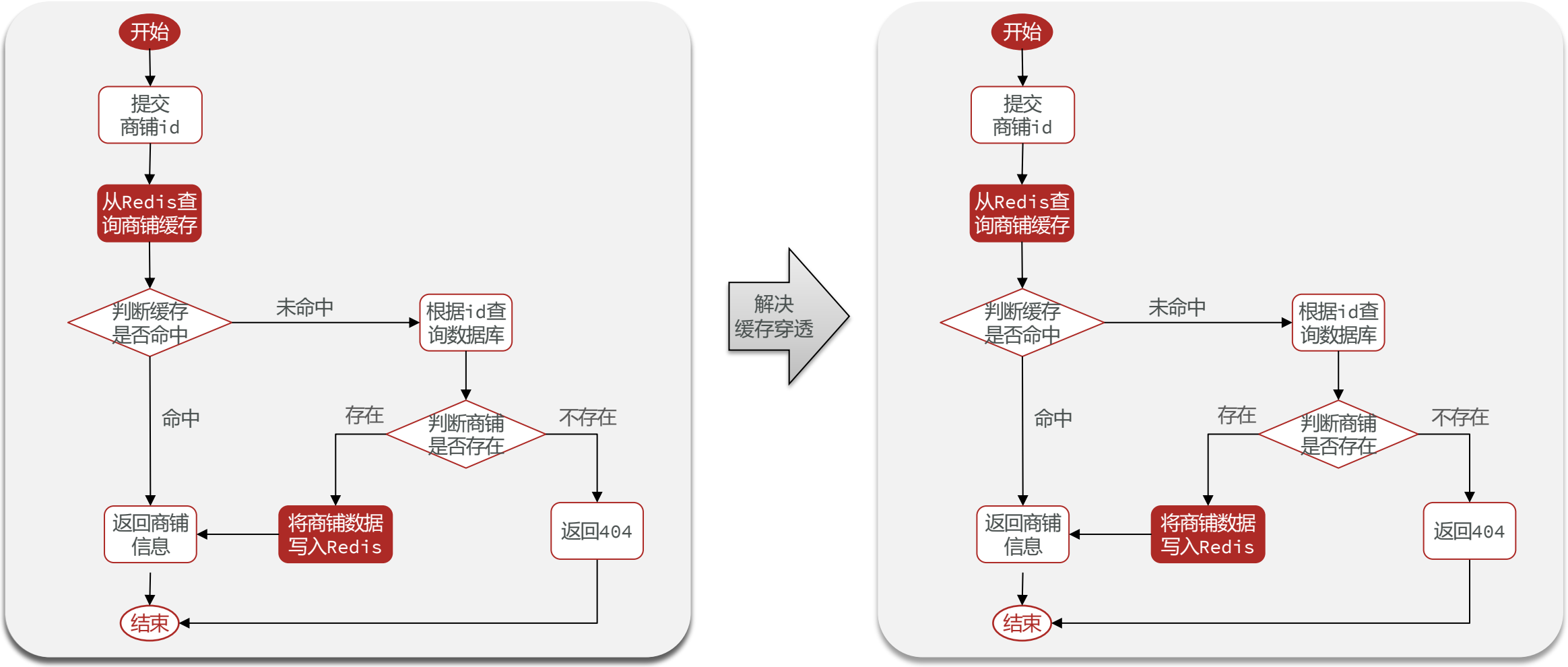
常见的解决方案有两种：

- 缓存空对象
  - 优点：实现简单，维护方便
  - 缺点：
    - 额外的内存消耗
    - 可能造成短期的不一致
- 布隆过滤
  - 优点：内存占用较少，没有多余key
  - 缺点：
    - 实现复杂
    - 存在误判可能

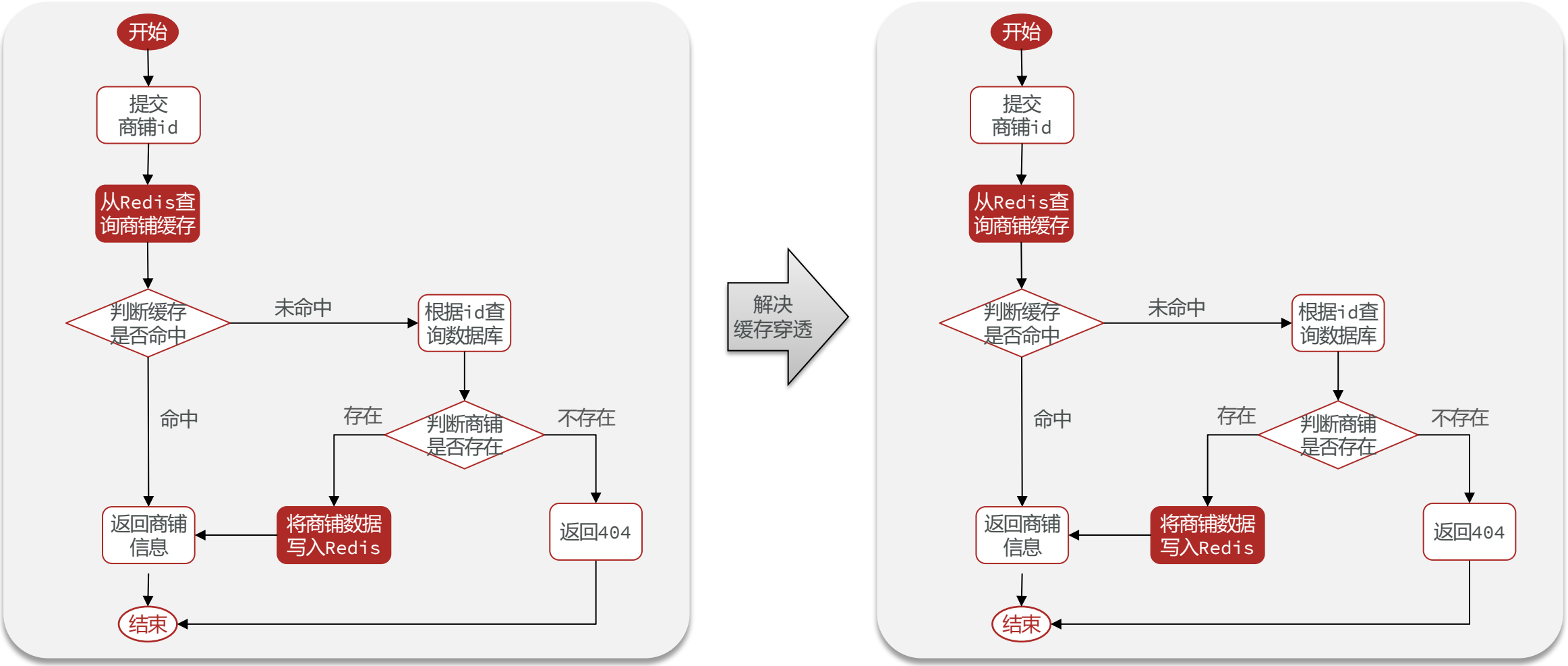




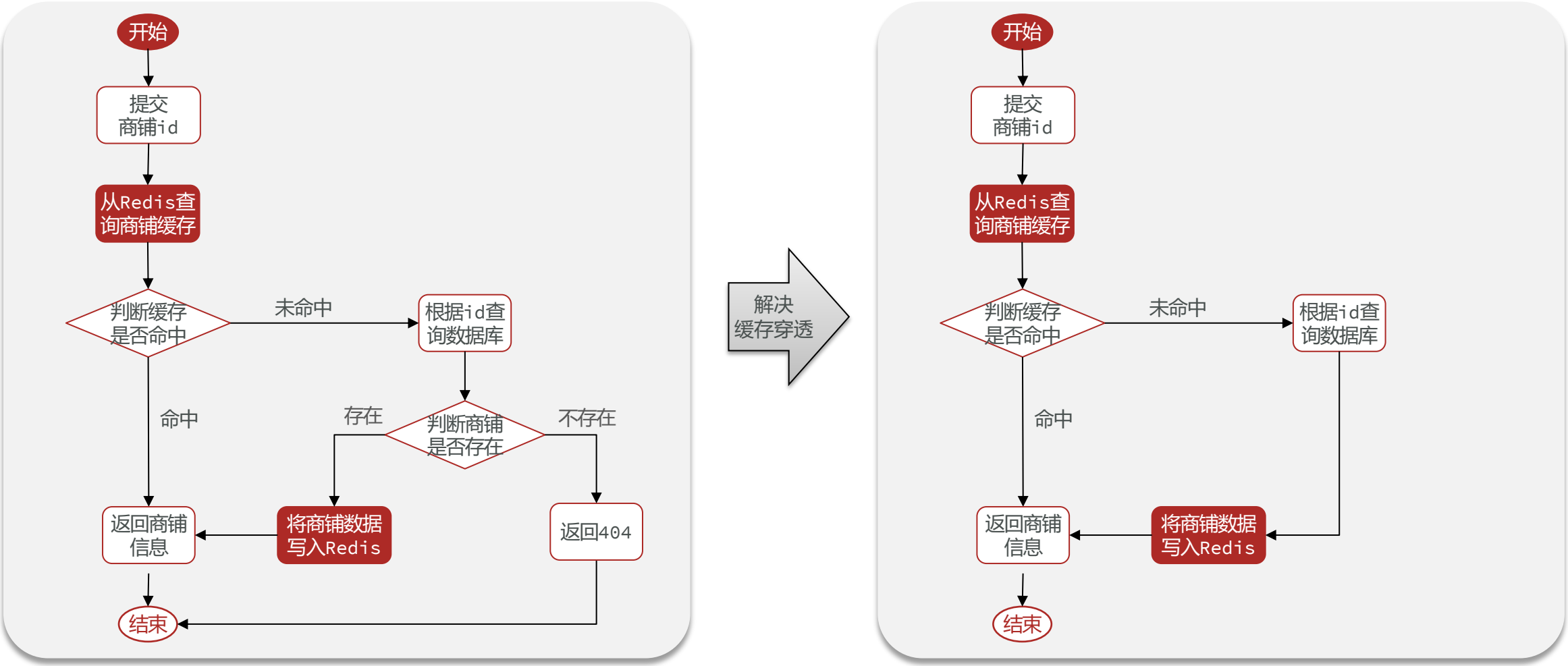
缓存穿透

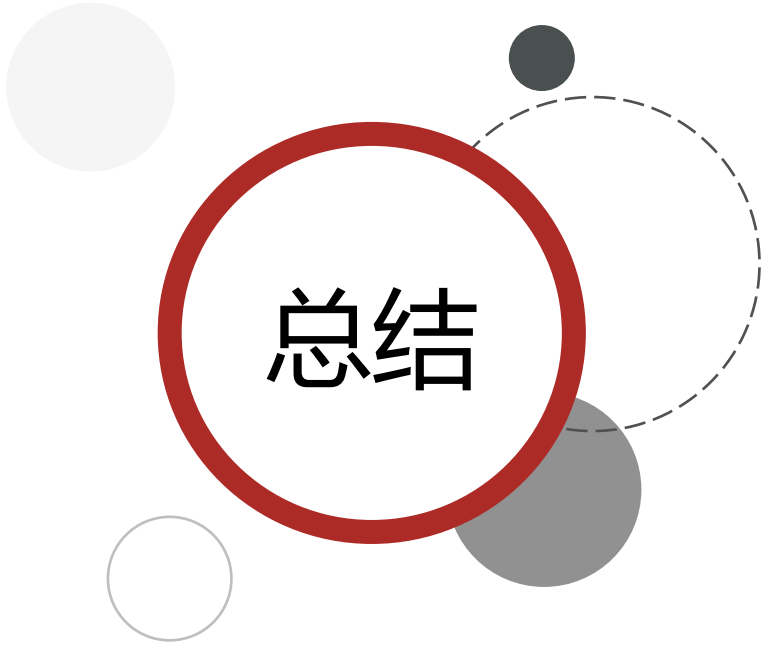

开始
提交商铺id
从Redis查询商铺缓存
判断缓存是否命中
未命中
根据id查询数据库
命中
存在
判断商铺是否存在
不存在
返回商铺信息
将商铺数据写入Redis
返回404
结束
解决缓存穿透
开始
提交商铺id
从Redis查询商铺缓存
判断缓存是否命中
未命中
根据id查询数据库
命中
存在
判断商铺是否存在
不存在
返回商铺信息
将商铺数据写入Redis
返回404
结束

缓存穿透


开始
提交商铺id
从Redis查询商铺缓存
判断缓存是否命中
未命中
根据id查询数据库
命中
存在
判断商铺是否存在
不存在
返回商铺信息
将商铺数据写入Redis
返回404
结束
解决缓存穿透
开始
提交商铺id
从Redis查询商铺缓存
判断缓存是否命中
未命中
根据id查询数据库
命中
存在
判断商铺是否存在
不存在
返回商铺信息
将商铺数据写入Redis
返回404
结束

缓存穿透


开始
提交商铺id
从Redis查询商铺缓存
判断缓存是否命中
未命中
根据id查询数据库
命中
存在
判断商铺是否存在
不存在
返回商铺信息
将商铺数据写入Redis
返回404
结束
解决缓存穿透
开始
提交商铺id
从Redis查询商铺缓存
判断缓存是否命中
未命中
根据id查询数据库
命中
返回商铺信息
将商铺数据写入Redis
结束



缓存穿透产生的原因是什么？

- 用户请求的数据在缓存中和数据库中都不存在，不断发起这样的请求，给数据库带来巨大压力

缓存穿透的解决方案有哪些？

- 缓存null值
- 布隆过滤
- 增强id的复杂度，避免被猜测id规律
- 做好数据的基础格式校验
- 加强用户权限校验
- 做好热点参数的限流

目录

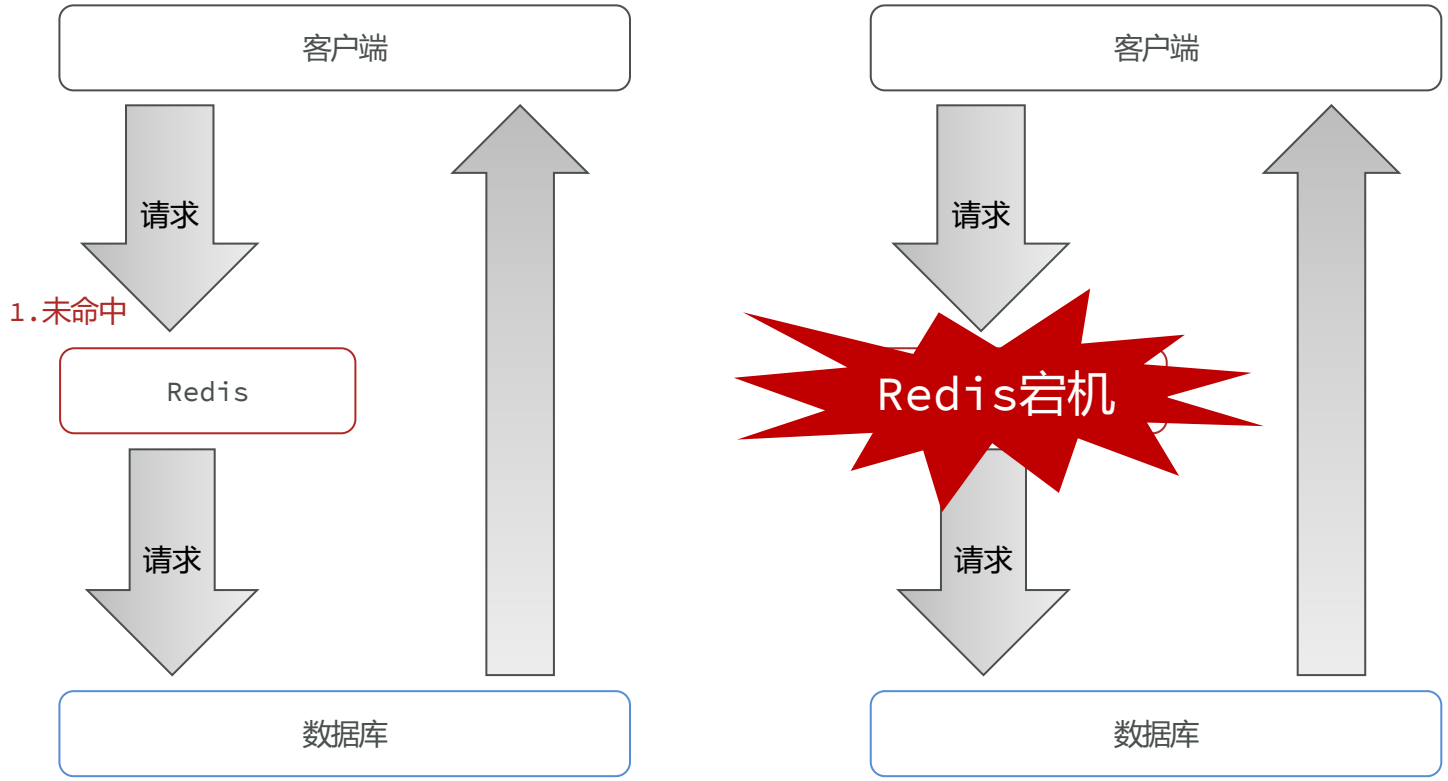
Contents

缓存雪崩

缓存雪崩是指在同一时段大量的缓存key同时失效或者Redis服务宕机，导致大量请求到达数据库，带来巨大压力。

解决方案：

- 给不同的Key的TTL添加随机值
- 利用Redis集群提高服务的可用性
- 给缓存业务添加降级限流策略
- 给业务添加多级缓存



目录

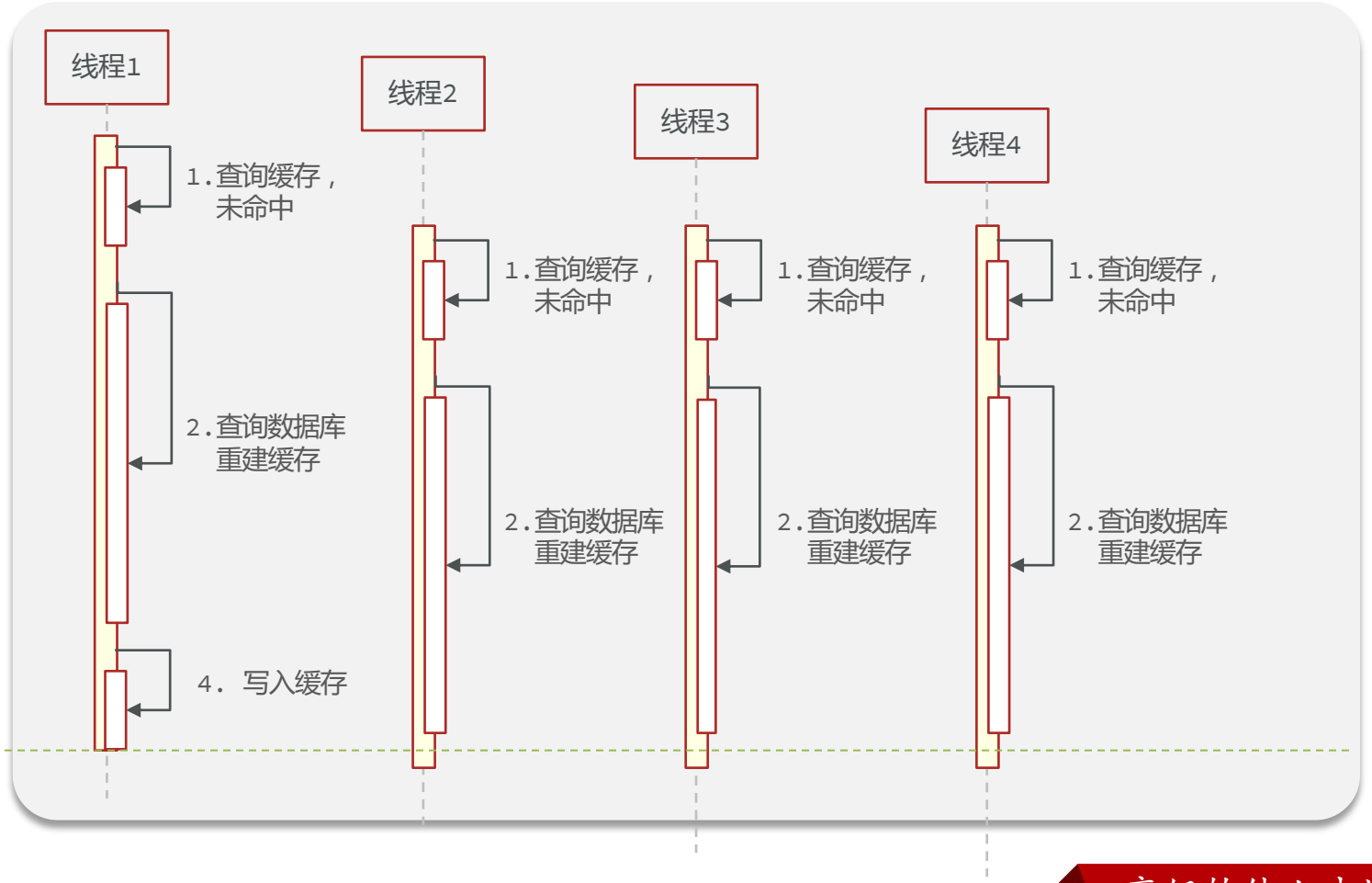
Contents

缓存击穿

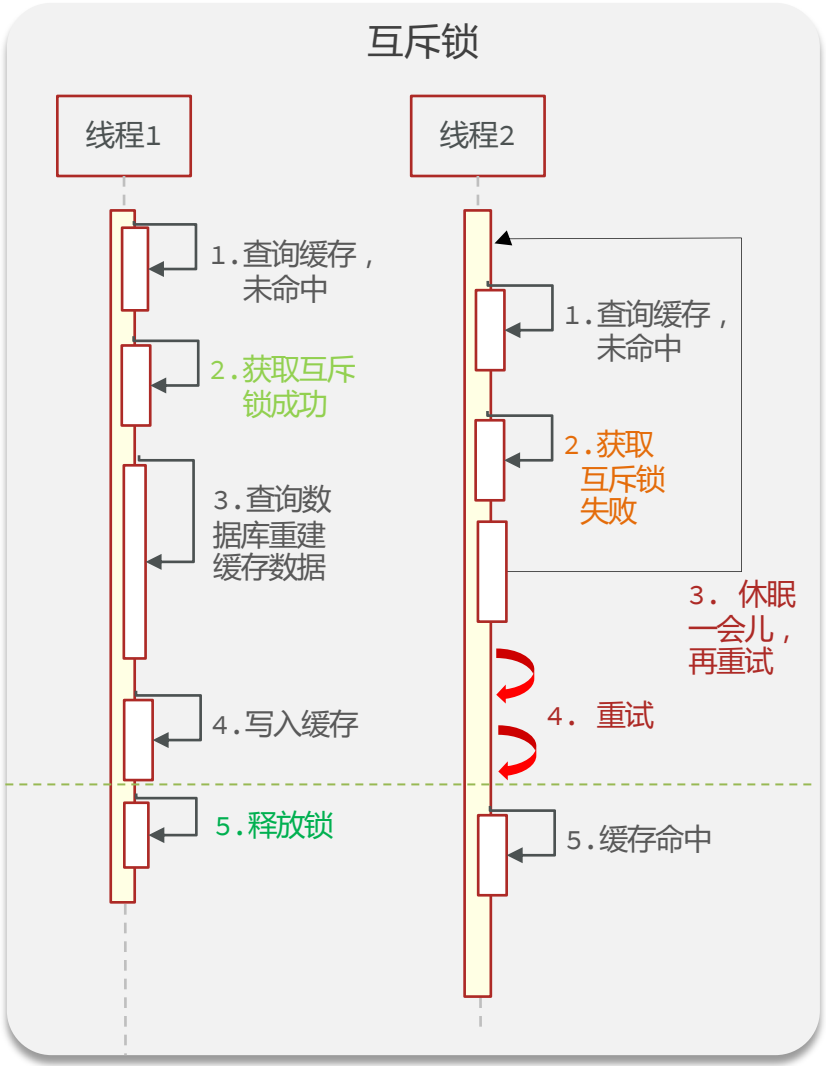
缓存击穿问题也叫热点Key问题，就是一个被**高并发访问**并且**缓存重建业务较复杂**的key突然失效了，无数的请求访问会在瞬间给数据库带来巨大的冲击。

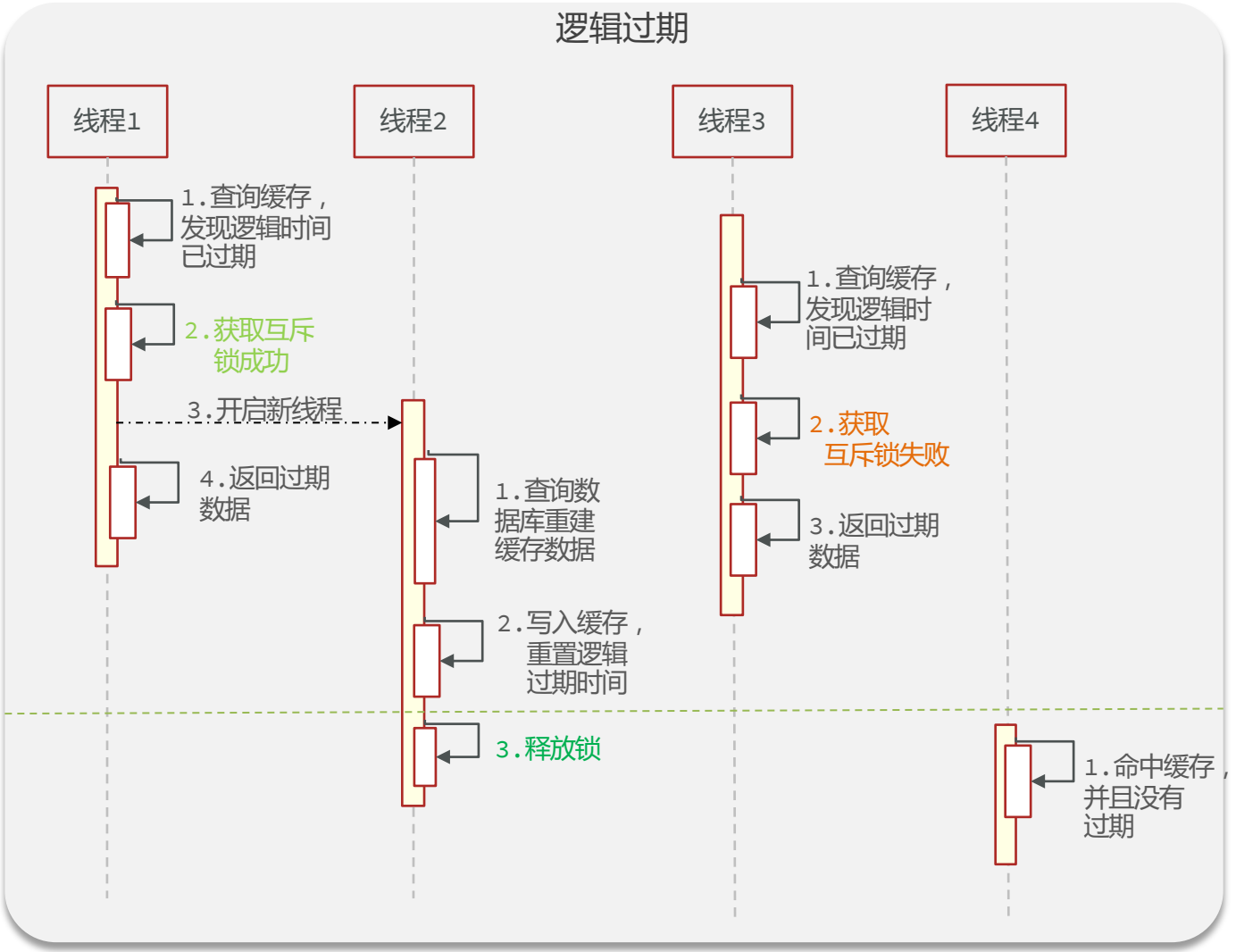
常见的解决方案有两种：

- 互斥锁
- 逻辑过期



缓存击穿


互斥锁
线程1
线程2
1.查询缓存，未命中
2.获取互斥锁成功
3.查询数据库重建缓存数据
4.写入缓存
5.释放锁
1.查询缓存，未命中
2.获取互斥锁失败
3. 休眠一会儿，再重试
4. 重试
5.缓存命中

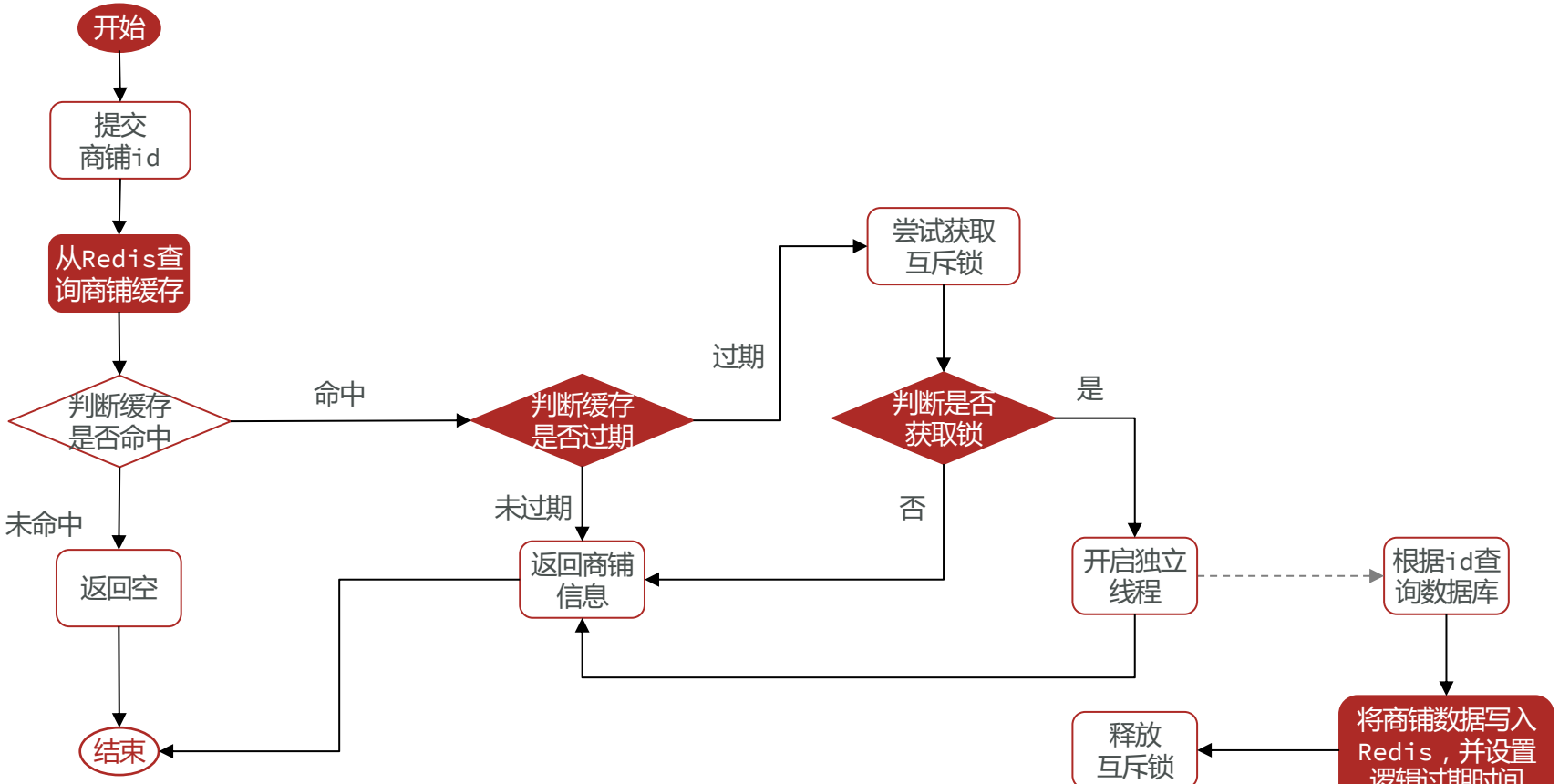

逻辑过期
线程1
线程2
线程3
线程4
1.查询缓存，发现逻辑时间已过期
2.获取互斥锁成功
3.开启新线程
4.返回过期数据
1.查询数据库重建缓存数据
2.写入缓存，重置逻辑过期时间
3.释放锁
1.查询缓存，发现逻辑时间已过期
2.获取互斥锁失败
3.返回过期数据
1.命中缓存，并且没有过期

缓存击穿

解决方案	优点	缺点
互斥锁	• 没有额外的内存消耗 • 保证一致性 • 实现简单	• 线程需要等待，性能受影响 • 可能有死锁风险
逻辑过期	• 线程无需等待，性能较好	• 不保证一致性 • 有额外内存消耗 • 实现复杂

案例 基于逻辑过期方式解决缓存击穿问题

需求：修改根据id查询商铺的业务，基于逻辑过期方式来解决缓存击穿问题



目录

Contents

案例 缓存工具封装

基于StringRedisTemplate封装一个缓存工具类，满足下列需求：

- ✓ 方法1： set(String key, Object value, Long time, TimeUnit unit) ，利用json实现对象序列化并存储在string类型的key中，并且可以设置过期时间
- ✓ 方法2：setWithLogicalExpire(String key, Object value, Long time, TimeUnit unit) ，利用json实现对象序列化并存储在string类型的key中，并且可以设置逻辑过期时间，用于处理缓存击穿问题
- ✓ 方法3：<R, K> R getPassThrough(String keyPrefix, K id, Class<R> type, Function<K, R> dbFallback, Long time, TimeUnit unit)，根据Key查询json数据，并根据type的类型反序列化。查询失败时，利用dbFallback功能去数据库查询，并写入缓存，设置过期时间。需要利用缓存null值来解决缓存穿透问题
- ✓ 方法4：<R, K> R getHotKey(String keyPrefix, K id, Class<R> type, Function<K, R> dbFallback, Long time, TimeUnit unit)，热点Key的查询，根据Key查询json数据，并根据type的类型反序列化。查询失败时，利用dbFallback功能去数据库查询，并写入缓存，设置过期时间。需要利用逻辑过期解决缓存击穿问题

03 优惠券秒杀

目录

Contents

传智教育旗下高端IT教育品牌