

第八章 分布式锁与缓存

欧阳修







九章算法

www.jiuzhang.com 全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

1. 分布式锁



1. 什么是分布式锁

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

1. 什么是分布式锁

Java中的锁主要包括synchronized锁和JUC包中的锁，这些锁都是针对单个JVM实例上的锁，对于分布式环境如果我们需要加锁就显得无能为力。在单个JVM实例上，锁的竞争者通常是一些不同的线程，而在分布式环境中，锁的竞争者通常是一些不同的线程或者进程。如何实现在分布式环境中对一个对象进行加锁呢？这就需要用到分布式锁

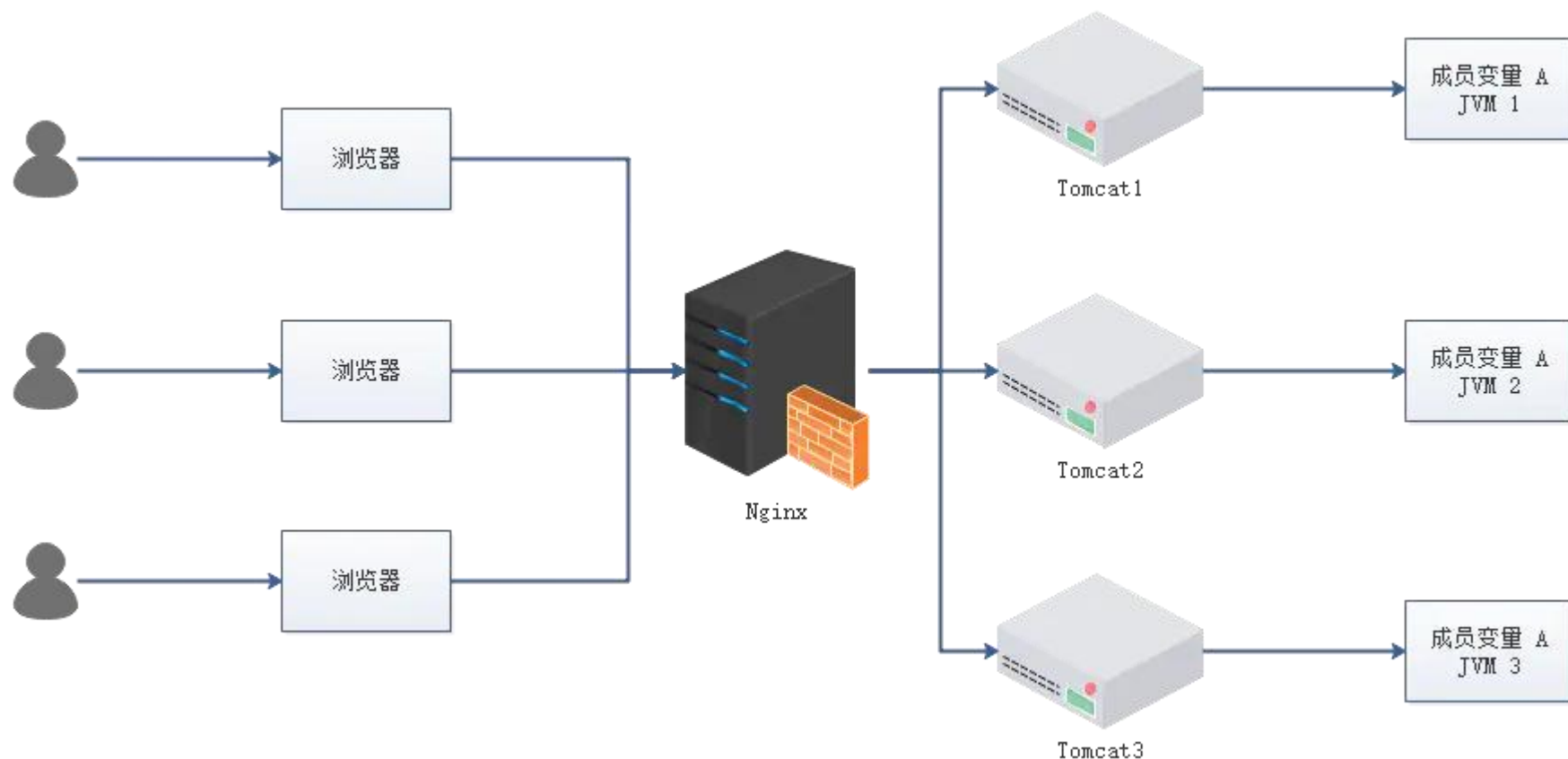


2. 为什么要使用分布式锁

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

2. 为什么要使用分布式锁

注意这是单机应用，后来业务发展，需要做集群，一个应用需要部署到几台机器上然后做负载均衡



2. 为什么要使用分布式锁

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

2. 为什么要使用分布式锁

注意这是单机应用，后来业务发展，需要做集群，一个应用需要部署到几台机器上然后做负载均衡

为了保证一个方法或属性在高并发情况下的同一时间只能被同一个线程执行，在传统单体应用单机部署的情况下，可以使用**并发**处理相关的功能进行**互斥控制**。

但是，随着业务发展的需要，原单体单机部署的系统被演化成分布式集群系统后，由于分布式系统多线程、多进程并且分布在**不同机器**上，这将使原单机部署情况下的**并发控制锁策略失效**，单纯的应用并不能提供分布式锁的能力。

为了解决这个问题就需要一种**跨机器的互斥机制**来控制共享资源的访问，这就是分布式锁要解决的问题！



3. 分布式锁的实现方式

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

分布式锁的实现方式

01

基于数据库的实现方式

02

基于Redis的分布式锁



3. 分布式锁的实现方式 - 数据库

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

1. 基于数据库的实现方式

1. 实现思路


在数据库中创建一个表，表中包含方法名等字段，并在方法名字段上创建唯一索引，想要执行某个方法，就使用这个方法，并向表中插入数据，成功插入则获取锁，执行完成后删除对应的行数据释放锁。



1. 基于数据库的实现方式

setp1:创建一个表

```
DROP TABLE IF EXISTS `method_lock`;
CREATE TABLE `method_lock` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT '主键',
  `method_name` varchar(64) NOT NULL COMMENT '锁定的方法名',
  `desc` varchar(255) NOT NULL COMMENT '备注信息',
  `update_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uidx_method_name` (`method_name`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8 COMMENT='锁定中的方法';
```

栏位	索引	外键	触发器	选项	注释	SQL 预览
名					类型	长度 小数点 不是 null
id					int	11 0 <input checked="" type="checkbox"/>  1
▶ method_name					varchar	64 0 <input checked="" type="checkbox"/>
desc					varchar	255 0 <input checked="" type="checkbox"/>
update_time					timestamp	0 0 <input checked="" type="checkbox"/>



3. 分布式锁的实现方式 - 数据库

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

1. 基于数据库的实现方式

setp2: 想要执行某个方法，就使用这个方法名向表中插入数据

```
INSERT INTO method_lock (method_name, desc) VALUES ('methodName', '测试的methodName');
```

因为我们对 method_name 做了唯一性约束，这里如果有多个请求同时提交到数据库的话，数据库会保证只有一个操作可以成功，那么我们就可以认为操作成功的那个线程获得了该方法的锁，可以执行方法体内容。



3. 分布式锁的实现方式 - 数据库

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

1. 基于数据库的实现方式

setp3: 执行完成后删除对应的行数据释放锁

```
delete from method_lock where method_name = 'methodName';
```



3. 分布式锁的实现方式 - 数据库

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

1. 基于数据库的实现方式

缺点

1

强依赖数据库的可用性，数据库的可用性和性能将直接影响分布式锁的可用性及性能

2

一旦数据库挂了，就不能使用了

3

没有失效时间，一旦解锁操作失败，就会导致锁记录一直在数据库中，其他线程无法再获得到锁。



3. 分布式锁的实现方式 - Redis

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

2. 基于Redis的分布式锁的实现方式

1. 加锁代码

```
/**
 * 获取分布式锁
 * @param lockKey
 * @param requestId
 * @param expireTime
 * @return
 */
public boolean tryGetDistributedLock(String lockKey, String requestId, int expireTime) {
    Jedis jedisClient = jedisPool.getResource();
    String result = jedisClient.set(lockKey, requestId, "NX", "PX", expireTime);
    if ("OK".equals(result)) {
        return true;
    }
    return false;
}
```



2. 基于Redis的分布式锁的实现方式

`jedis.set(String key, String value, String nxxx, String expx, int time)`, 这个`set()`方法一共有五个形参

1. 第一个为 **key**, 我们使用 **key** 来当锁, 因为key是唯一的。

2. 第二个为 **value**, 我们传的是 `requestId`,

为什么还要用到 `value`?

给 `value` 赋值为 `requestId`, 我们就知道这把锁是哪个请求加的了, 在解锁的时候就可以有依据。`requestId` 可以使用 `UUID.randomUUID().toString()`方法生成。

3. 第三个 **NX**, 意思是 SET IF NOT EXIST, 即当 `key` 不存在时, 我们进行 `set` 操作; 若 `key` 已经存在, 则不做任何操作;

4. 第四个 **PX**, 意思是我们要给这个 `key` 加一个过期的设置, 具体时间由第五个参数决定。

5. 第五个为 **time**, 与第四个参数相呼应, 代表 `key` 的过期时间。



3. 分布式锁的实现方式 - Redis

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

2. 基于Redis的分布式锁的实现方式

2. 解锁代码

```
/**
 * 释放分布式锁
 *
 * @param lockKey 锁
 * @param requestId 请求标识
 * @return 是否释放成功
 */
public boolean releaseDistributedLock(String lockKey, String requestId) {
    Jedis jedisClient = jedisPool.getResource();
    String script = "if redis.call('get', KEYS[1]) == ARGV[1] then return redis.call('del', KEYS[1]) else return 0 end";
    Long result = (Long) jedisClient.eval(script, Collections.singletonList(lockKey), Collections.singletonList(requestId));
    if (result == 1L) {
        return true;
    }
    return false;
}
```



2. 基于Redis的分布式锁的实现方式

2. 解锁代码

01

这段Lua代码的功能？

首先获取锁对应的value值，检查是否与requestId相等，如果相等则删除锁（解锁）

02

为什么要使用Lua语言来实现呢？

要确保上述操作是原子性的，eval命令执行Lua代码的时候，Lua代码将被当成一个命令去执行，并且直到eval命令执行完成，Redis才会执行其他命令



2. Redis 原子操作原理



1. 原子性(Atomicity)概念

原子性是数据库的事务中的特性。在数据库事务的情景下，原子性指的是：一个事务（transaction）中的**所有操作**，**要么全部完成**，**要么全部不完成**，不会结束在中间某个环节。

对于Redis而言，命令的原子性指的是：**一个操作的不可以再分**，**操作要么执行**，**要么不执行**。



2. Redis 如何保证原子性?

答案很简单，因为 Redis 是单线程的。



3. Redis 查询速度为什么这么快

- 1、完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。**数据存在内存中**，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是 $O(1)$ ；
- 2、**数据结构简单**，对数据**操作也简单**，Redis 中的数据结构是专门进行设计的；
- 3、采用**单线程**，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；
- 4、使用**多路I/O复用**模型，非阻塞IO。



3. 本地缓存



1. 本地缓存概念

在 **Client** 端使用本地缓存，从而降低了 redis 集群对 hot key 的访问量。

比如利用 GuavaCache 是一款面向本地缓存的，轻量级的 Cache，适合缓存少量数据。或者用一个 HashMap 都可以，在你发现热 key 以后，把热 key 加载到系统的 JVM 中。

针对这种热 key 请求，会直接从 jvm 中取，而不会走到 redis 层。



2.本地缓存作用

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

2.本地缓存作用

假设此时有十万个针对同一个 key 的请求过来，如果没有本地缓存，这十万个请求就直接怼到同一台 redis 上了。

现在假设，你的应用层有 50 台机器，OK，你也有jvm缓存了。这十万个请求平均分散开来，每个机器有2000个请求，会从 JVM 中取到 value 值，然后返回数据。避免了十万个请求怼到同一台 redis 上的情形。



4. 缓存穿透 & 缓存击穿 & 缓存雪崩



1. 缓存穿透

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

1. 缓存穿透

缓存穿透的概念

缓存穿透的概念很简单，用户想要查询一个数据，发现 **redis 内存中没有**，也就是缓存没有命中，于是向持久层**数据库**查询。发现**也没有**，于是本次查询失败。

当**用户很多**的时候，**缓存都没有命中**，于是都去请求了持久层数据库。这会给持久层数据库造成很大的压力，这时候就相当于出现了缓存穿透。



1. 缓存穿透

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

缓存穿透的两种解决方案

01

缓存空对象

02

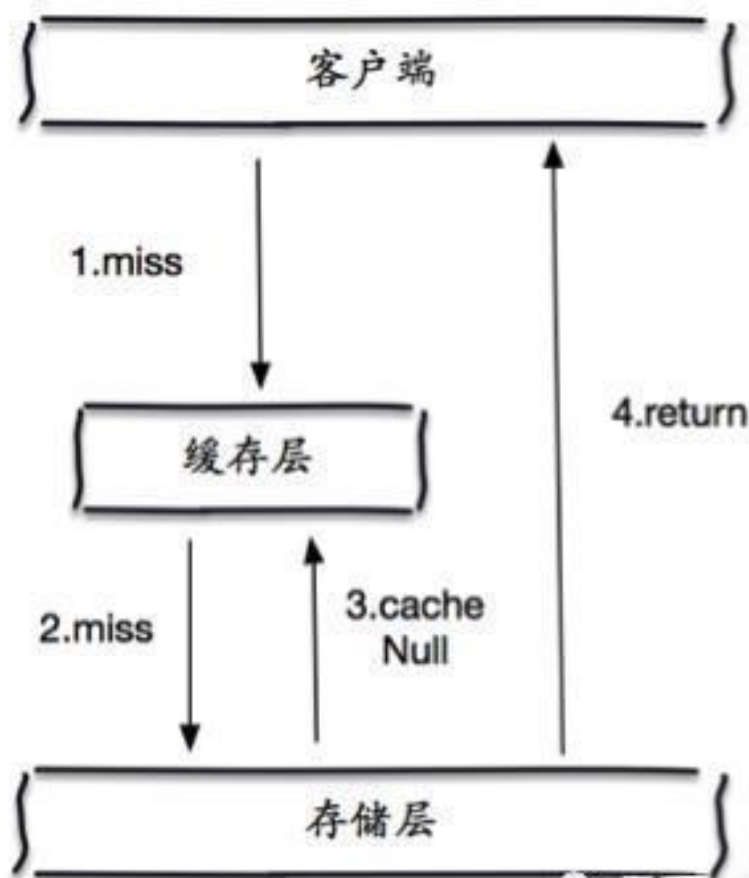
布隆过滤器 (Bloom Filter)



1. 缓存穿透-解决方案

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

缓存空对象



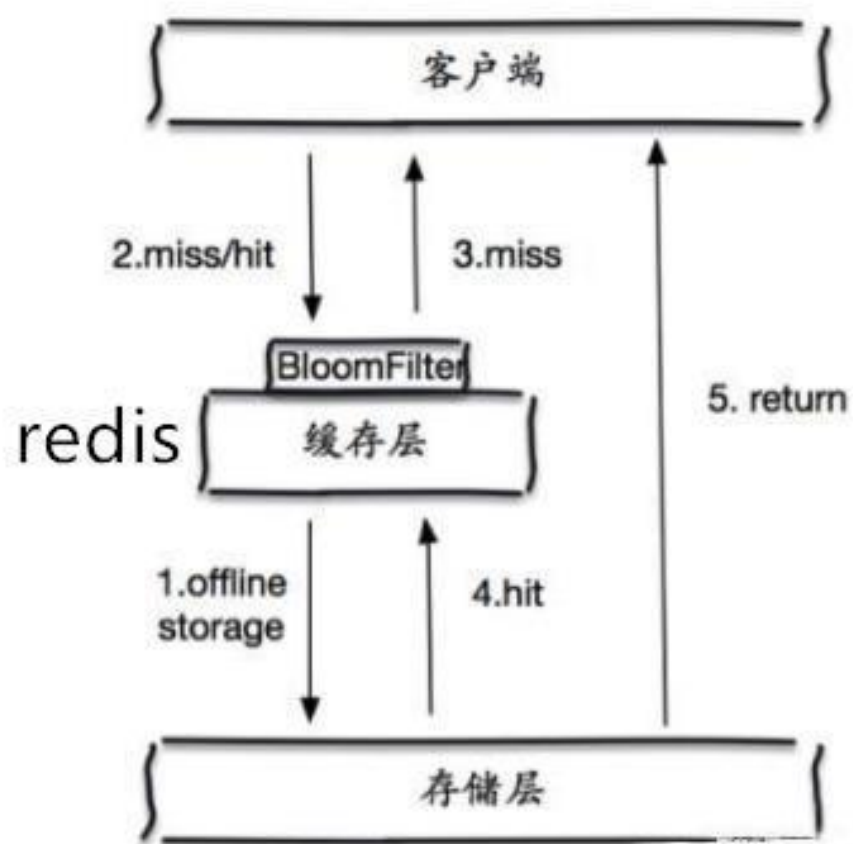
当存储层不命中后，即使返回的空对象也将其缓存起来，同时会设置一个过期时间，之后再访问这个数据将会从缓存中获取，保护了后端数据源



1. 缓存穿透-解决方案

全网一手不加密IT课程, 需要请加微信zzj97666九章来offer都有

布隆过滤器



首先对**所有可能查询的参数**以 hash 形式存储, 当用户想要查询的时候, 使用布隆过滤器**发现不在集合中**, 就直接**丢弃**, 不再对持久层查询。



2. 缓存击穿

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

2. 缓存击穿

缓存击穿的概念

缓存击穿，是指一个 **key** 非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个**key** 在失效的瞬间，持续的大并发就穿破缓存，直接**请求数据库**，就像在一个屏障上凿开了一个洞。



2. 缓存击穿

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

2. 缓存击穿

缓存击穿和缓存穿透的区别

缓存击穿和缓存穿透的区别是：

穿透：底层数据库**没有**数据 且 缓存内也**没有**数据

击穿：底层数据库**有**数据 而 缓存内**没有**数据

具体的现象是，当一个热点 key 从缓存内失效时，大量访问同时请求这个数据，就会将查询下沉到数据库层，此时数据库层的负载压力会骤增。



2. 缓存击穿

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

缓存击穿解决方案

延长热点 key 的过期时间或者设置永不过期

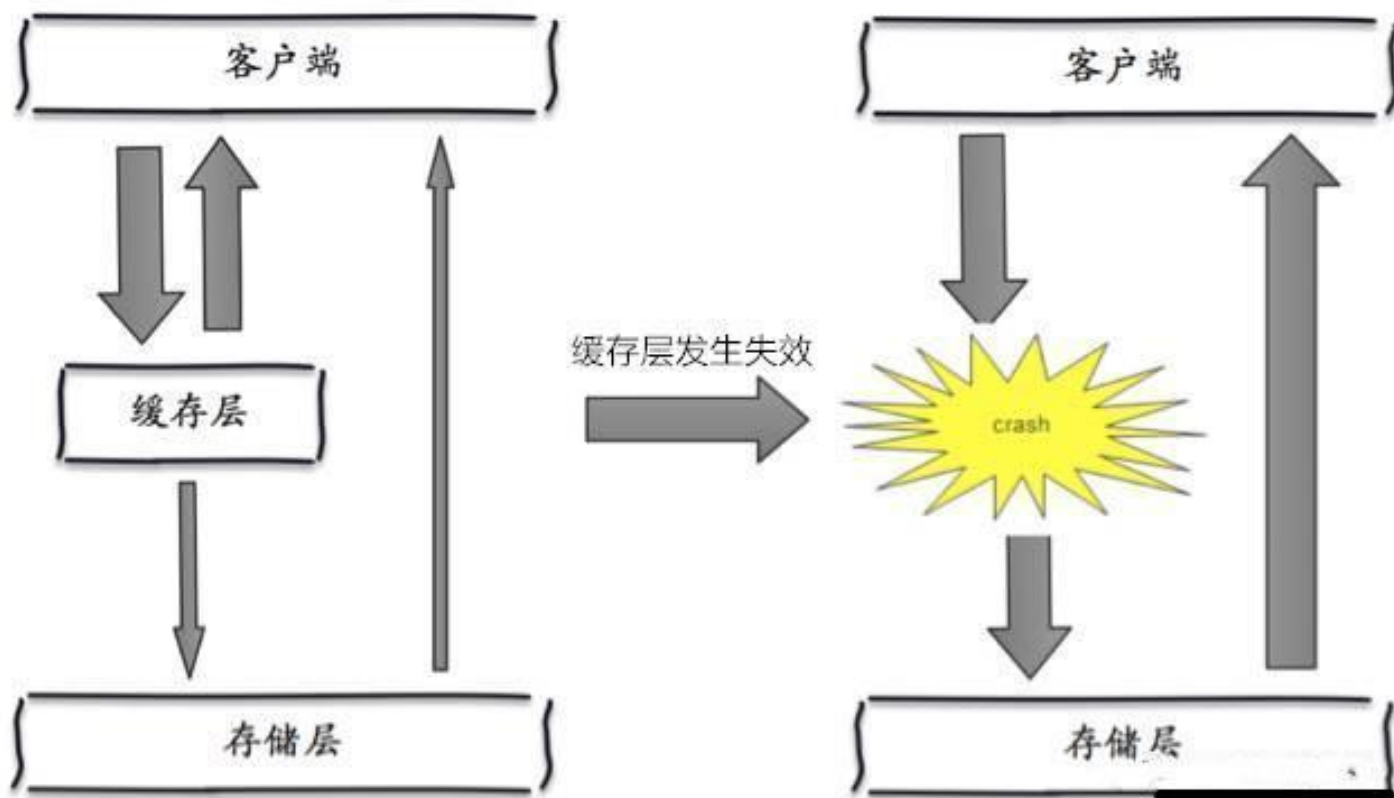


3. 缓存雪崩

全网一手不加密IT课程, 需要请加微信zzj97666九章来offer都有

3. 缓存雪崩

缓存雪崩的概念



当**缓存服务器**重启或者大量缓存集中在某一个时间段**失效**, 这样在失效的时候, 也会给后端系统(比如DB)带来很大压力, 造成**数据库后端故障**, 从而引起**应用服务器雪崩**。

3. 缓存雪崩

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

3. 缓存雪崩

产生缓存雪崩的例子

产生雪崩的原因之一，马上就要到双十一零点，很快就会迎来一波抢购，这波商品时间比较集中的放入了缓存，假设缓存**二个小时**（从11点到凌晨1点）。

那么到了凌晨一点钟的时候，这批商品的缓存就都集中**过期了**。

而对这批商品的访问查询，**都落到了数据库**上，对于数据库而言，就会产生周期性的压力波峰



3. 缓存雪崩

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

缓存雪崩解决方案



缓存数据的**过期时间设置随机**，防止同一时间大量数据过期现象发生



如果缓存数据库是分布式部署，将热点数据**均匀分布**在不同的缓存数据库中



设置热点数据**永远不过期**



5. 简单的打包部署



1. 修改pom

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

1. 修改 pom 文件

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

上面那个 plugin 是 Springboot 框架提供了一套自己的打包机制。下面的 plugin 可以指定项目源码的jdk版本，编译后的jdk版本，以及编码格式。

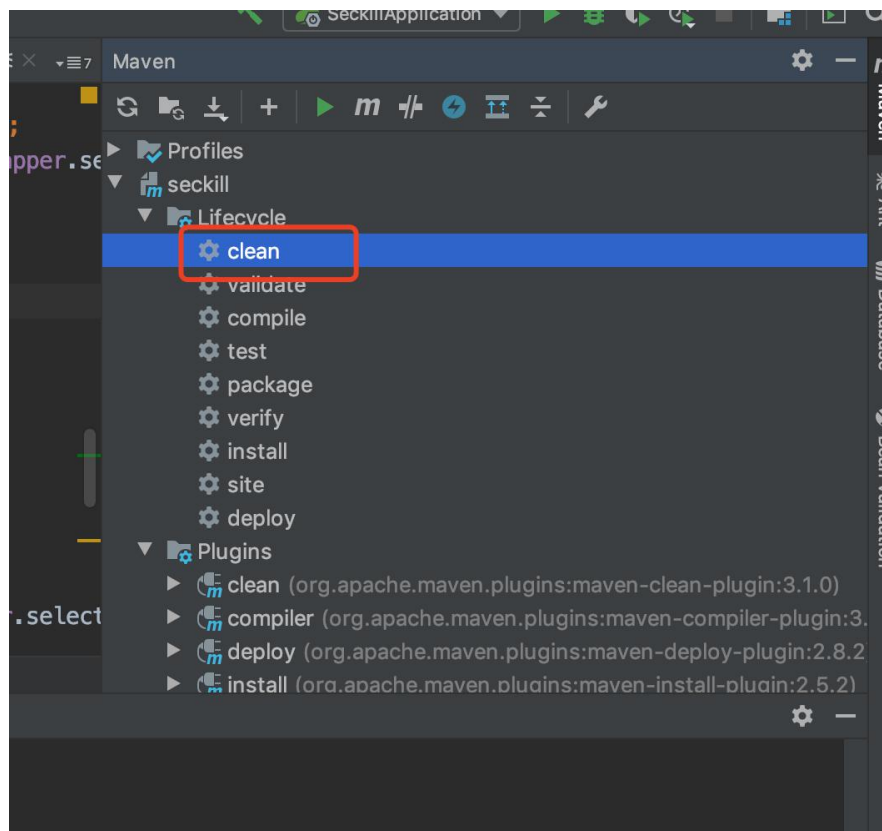


2. mvn clean 清理

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

2. mvn clean 清理

此时会清空 target 目录

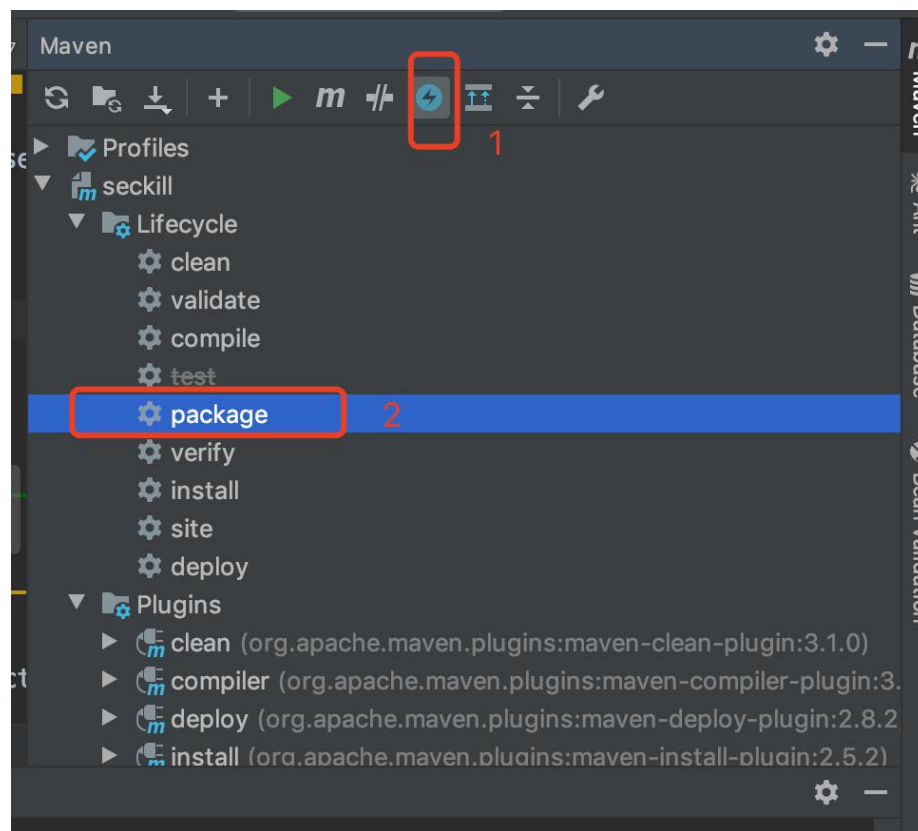


3. mvn package 打包

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

3. mvn package 打包

生成可以部署的 jar 包

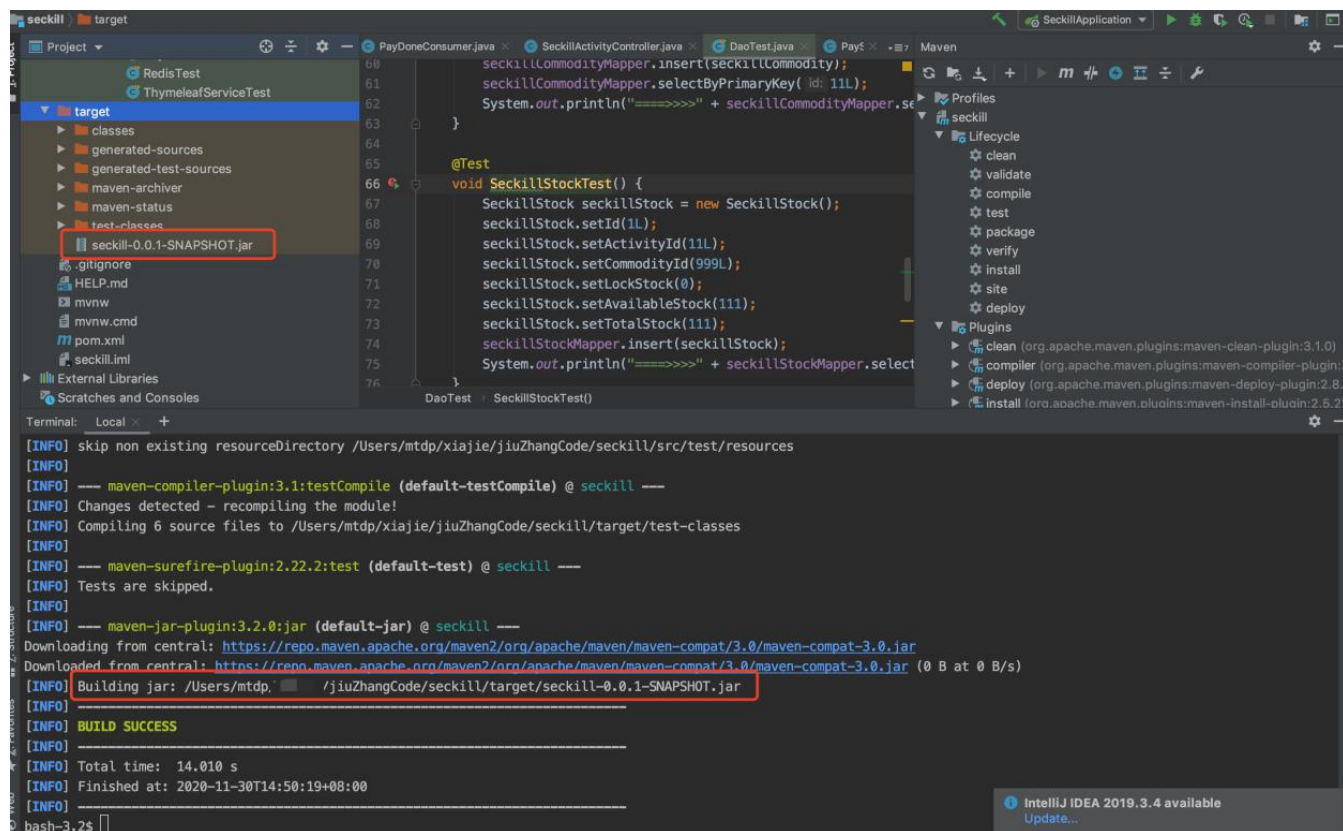


3. mvn package 打包

全网一手不加密IT课程，需要请加微信zzj97666九章来offer都有

3. mvn package 打包

jar 包生成后会保存在 target 目录下



```
Project: seckill
├── classes
├── generated-sources
├── generated-test-sources
├── maven-archiver
├── maven-status
├── test-classes
└── seckill-0.0.1-SNAPSHOT.jar

Terminal: Local
[INFO] skip non existing resourceDirectory /Users/mtdp/xiajie/jiuZhangCode/seckill/src/test/resources
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ seckill ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 6 source files to /Users/mtdp/xiajie/jiuZhangCode/seckill/target/test-classes
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ seckill ---
[INFO] Tests are skipped.
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ seckill ---
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-compat/3.0/maven-compat-3.0.jar
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-compat/3.0/maven-compat-3.0.jar (0 B at 0 B/s)
[INFO] Building jar: /Users/mtdp/.../jiuZhangCode/seckill/target/seckill-0.0.1-SNAPSHOT.jar
[INFO] BUILD SUCCESS
[INFO] Total time: 14.010 s
[INFO] Finished at: 2020-11-30T14:50:19+08:00
[INFO]
```



4. 运行项目

命令行中输入以下命令：

```
java -jar seckill-0.0.1-SNAPSHOT.jar
```

(后面的 jar 文件是你自己生成的文件名称)



5. 项目启动

启动项目后就可以在浏览器进行访问了

```
[~bash-3.2$ java -jar seckill-0.0.1-SNAPSHOT.jar]

:: Spring Boot ::
(V2.3.4.RELEASE)

2020-11-30 15:03:19.015 INFO 35679 --- [main] com.jiuzhang.seckill.SeckillApplication : Starting SeckillApplication v0.0.1-SNAPSHOT on MacBook-Pro-8.local with PID 35679 (/Users/mtdp/xiajie/jiuZhangCode/seckill/target/seckill-0.0.1-SNAPSHOT.jar started by mtdp in /Users/mtdp/xiajie/jiuZhangCode/seckill/target)
2020-11-30 15:03:19.018 INFO 35679 --- [main] com.jiuzhang.seckill.SeckillApplication : No active profile set, falling back to default profiles: default
2020-11-30 15:03:20.148 INFO 35679 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8082 (http)
2020-11-30 15:03:20.163 INFO 35679 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-11-30 15:03:20.163 INFO 35679 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.38]
2020-11-30 15:03:20.243 INFO 35679 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2020-11-30 15:03:20.244 INFO 35679 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1174 ms
2020-11-30 15:03:20.379 INFO 35679 --- [main] com.jiuzhang.seckill.config.JedisConfig : JedisPool注入成功!
2020-11-30 15:03:20.379 INFO 35679 --- [main] com.jiuzhang.seckill.config.JedisConfig : redis地址: 139.196.48.177:6379
Loading class 'com.mysql.jdbc.Driver'. This is deprecated. The new driver class is 'com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual loading of the driver class is generally unnecessary.
INFO: Sentinel log output type is: file
INFO: Sentinel log charset is: utf-8
INFO: Sentinel log base directory is: /Users/mtdp/logs/csp/
INFO: Sentinel log name use pid is: false
2020-11-30 15:03:21.585 INFO 35679 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-11-30 15:03:21.798 INFO 35679 --- [main] a.r.s.s.DefaultRocketMQListenerContainer : running container: DefaultRocketMQListenerContainer{consumerGroup='pay_done_group', nameServer='139.196.48.177:9876', topic='pay_done', consumeMode=CONCURRENTLY, selectorType=TAG, selectorExpression='*', messageModel=CLUSTERING}
2020-11-30 15:03:21.800 INFO 35679 --- [main] o.a.r.s.a.ListenerContainerConfiguration : Register the listener to container, listenerBeanName:payDoneConsumer, containerBeanName:org.apache.rocketmq.spring.support.DefaultRocketMQListenerContainer_1
2020-11-30 15:03:21.827 INFO 35679 --- [main] a.r.s.s.DefaultRocketMQListenerContainer : running container: DefaultRocketMQListenerContainer{consumerGroup='seckill_order_group', nameServer='139.196.48.177:9876', topic='seckill_order', consumeMode=CONCURRENTLY, selectorType=TAG, selectorExpression='*', messageModel=CLUSTERING}
2020-11-30 15:03:21.827 INFO 35679 --- [main] o.a.r.s.a.ListenerContainerConfiguration : Register the listener to container, listenerBeanName:orderConsumer, containerBeanName:org.apache.rocketmq.spring.support.DefaultRocketMQListenerContainer_2
2020-11-30 15:03:21.864 INFO 35679 --- [main] a.r.s.s.DefaultRocketMQListenerContainer : running container: DefaultRocketMQListenerContainer{consumerGroup='conmuserGrop-jiuzhang', nameServer='139.196.48.177:9876', topic='test-jiuzhang', consumeMode=CONCURRENTLY, selectorType=TAG, selectorExpression='*', messageModel=CLUSTERING}
2020-11-30 15:03:21.864 INFO 35679 --- [main] o.a.r.s.a.ListenerContainerConfiguration : Register the listener to container, listenerBeanName:consumerListener, containerBeanName:org.apache.rocketmq.spring.support.DefaultRocketMQListenerContainer_3
2020-11-30 15:03:21.901 INFO 35679 --- [main] a.r.s.s.DefaultRocketMQListenerContainer : running container: DefaultRocketMQListenerContainer{consumerGroup='pay_check_group', nameServer='139.196.48.177:9876', topic='pay_check', consumeMode=CONCURRENTLY, selectorType=TAG, selectorExpression='*', messageModel=CLUSTERING}
2020-11-30 15:03:21.901 INFO 35679 --- [main] o.a.r.s.a.ListenerContainerConfiguration : Register the listener to container, listenerBeanName:payStatusCheckListener, containerBeanName:org.apache.rocketmq.spring.support.DefaultRocketMQListenerContainer_4
2020-11-30 15:03:21.951 INFO 35679 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8082 (http) with context path ''
2020-11-30 15:03:21.960 INFO 35679 --- [main] com.jiuzhang.seckill.SeckillApplication : Started SeckillApplication in 3.253 seconds (JVM running for 3.628)
2020-11-30 15:03:53.884 INFO 35679 --- [nio-8082-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-11-30 15:03:53.884 INFO 35679 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2020-11-30 15:03:53.892 INFO 35679 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 7 ms
2020-11-30 15:03:54.003 INFO 35679 --- [nio-8082-exec-1] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2020-11-30 15:03:54.008 WARN 35679 --- [nio-8082-exec-1] com.zaxxer.hikari.util.DriverDataSource : Registered driver with driverClassName=com.mysql.jdbc.Driver was not found, trying direct instantiation.
2020-11-30 15:03:54.222 INFO 35679 --- [nio-8082-exec-1] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
```



回顾与总结

回顾并总结本节主要的知识点



