《JavaEE 平台技术》实验报告

实验名称:	Redis 缓存的效率
实验日期:	2021.10.27
实验地点:	宿舍
提交日期:	2021.10.29

组号:	1-07
组名:	这队更是重量级
专业年级:	软件工程 2019 级
学年学期:	21-22 学年上半学期

一、 实验目的

1、掌握 Redis 缓存的使用方法

2、比较使用和不使用 Redis 缓存的效果

二、 实验环境

1、服务器 A: Ubuntu 18.04 服务器 2 核 4G 内存虚拟机一台，图形界面，安装 JDK 11，Maven、git，Redis 6.2.4
2、服务器 B：Ubuntu 18.04 服务器 2 核 2G 内存虚拟机一台，命令行界面，安装 JDK 11，Maven、git，JMeter 5.4.1
3、服务器 C：Ubuntu 18.04 服务器 2 核 2G 内存虚拟机一台，命令行界面，安装 JDK 11，Maven、git，MySQL 8.0

三、 实验内容及要求

使用 Redis 可以有效的提升系统的性能，商品信息是一个会经常读的数据，请用 Redis 缓存和不用 Redis 缓存分别实现以下 RESTful API，利用 JMeter 测试两者的性能，并给出 分析报告。

四、 实验设计

对于 Product 的获取，我们采取同样的策略：使用 ResultMap 在 SQL 层完成对象的关联
虽然实验三中，我们的结论是在数据量很大的情况下，尽量使用单表查询，然后在 Dao 层完成对象的关联，但考虑到本次实验中需要获取的是 Product 对象，而一个 Product 只能和一个 Goods 关联，因此我们可以忽略关联的影响，直接在 SQL 层即完成关联

对于 Product 的筛选

```xml
<select id="findProduct" parameterType="ProductPo" resultMap="ProductWithGoodsMap">
    select p.id as id,
           p.name,
           p.product_sn,
           p.detail,
           p.image_url,
           p.state,
           p.disabled,
           p.original_price,
           p.weight,
           p.gmt_create,
           p.gmt_modified,
           g.id as goods_id,
           g.name as goods_name,
           g.goods_sn as goods_sn,
           g.image_url as goods_image_url,
           g.gmt_create as goods_gmt_create,
           g.gmt_modified as goods_gmt_modified,
           g.disabled as goods_disabled
    from product p left join goods g
    on p.goods_id = g.id
    where p.id = #{id}
</select>
```

相应 ResultMap 的定义

```xml
<resultMap id="ProductWithGoodsMap" type="ProductPo" autoMapping="true">
    <id property="id" column="id"/>
    <collection property="goodsPoList" ofType="com.example.exp_4.model.GoodsPo">
        <id property="id" column="goods_id"/>
        <result property="name" column="goods_name"/>
        <result property="goodsSn" column="goods_sn"/>
        <result property="imageUrl" column="goods_image_url"/>
        <result property="gmtCreate" column="goods_gmt_create"/>
        <result property="gmtModified" column="goods_gmt_modified"/>
        <result property="disabled" column="goods_disabled"/>
    </collection>
</resultMap>
```

对于是否使用 Redis 缓存，我们只需要在 Dao 层中区分即可

不使用 Redis 缓存的 findProduct

```java
public ReturnObject<List<Product>> findProduct(ProductPo productPo) {
    List<ProductPo> productPoList = productMapper.findProduct(productPo);
    List<Product> retProducts = new ArrayList<>(productPoList.size());

    for(ProductPo p : productPoList) {
        Product product = new Product(p);
        List<GoodsPo> goodsPoList = p.getGoodsPoList();
        List<Goods> goodsList = new ArrayList<>(goodsPoList.size());
        for(GoodsPo g : goodsPoList) {
            Goods goods = new Goods(g);
            goodsList.add(goods);
        }
        product.setGoodsList(goodsList);
        retProducts.add(product);
    }

    return new ReturnObject<>(retProducts);
}
```

使用 Redis 缓存的 findProductWithRedis

```java
public ReturnObject<List<Product>> findProductWithRedis(ProductPo productPo) {

    List<Product> retProduct = null;
    String key = null;
    if(null != productPo.getId()) {
        key = "p_" + productPo.getId();
        Product product = (Product) redisUtil.get(key);
        if(null != product) {
            retProduct = new ArrayList<>( initialCapacity: 1);
            retProduct.add(product);
            return new ReturnObject<>(retProduct);
        }
    }

    List<ProductPo> productPoList = productMapper.findProduct(productPo);
    retProduct = new ArrayList<>(productPoList.size());

    for(ProductPo p : productPoList) {
        Product product = new Product(p);
        List<GoodsPo> goodsPoList = p.getGoodsPoList();
        List<Goods> goodsList = new ArrayList<>(goodsPoList.size());
        for(GoodsPo g : goodsPoList) {
            Goods goods = new Goods(g);
            goodsList.add(goods);
        }
        product.setGoodsList(goodsList);
        retProduct.add(product);
    }

    if(null != productPo.getId()) {
        if(retProduct.size() != 0) {
            redisUtil.set(key, retProduct.get(0), productTimeout);
        } else {
            redisUtil.set(key, value: null, productTimeout);
        }
    }

    return new ReturnObject<>(retProduct);
}
```

值得注意的是，在 RedisUtil 中，我们对于 RedisTemplate 的定义是 RedisTemplate<String, Serializable>，也就是把 key 作为 String 来用，这是为了调试方便，同时也考虑到本次实验数据量较小，不会占用太多内存空间。实际使用中，最好使用二进制作为 key，可以节省更多内存

```java
public class RedisUtil {

    @Autowired
    private RedisTemplate<String, Serializable> redisTemplate;

    /**
```

实验要求中，要求查询一个 Product，考虑到一个商品可以有多个规格，但一个规格只能对应一个商品，我们添加一个 name 为 mobilephone 的 Goods，然后在 product 表中为其添加 200 个规格，以验证一个商品有多个规格情况下是否使用 Redis 缓存的差异

```
ysql> select id,name,gmt_create from goods;
+----+-------------+---------------------+
| id | name        | gmt_create          |
+----+-------------+---------------------+
|  1 | mobilephone | 2021-10-26 20:47:21 |
+----+-------------+---------------------+
 row in set (0.01 sec)

ysql>
```

```
mybaby@c17fd8acc2094d86943f6a2d694e9c98slave3: ~
File  Edit  Tabs  Help
|  94 |        1 | iphone      | 2021-10-28 22:57:00 |
|  95 |        1 | ipod        | 2021-10-28 22:57:00 |
|  96 |        1 | chuizi      | 2021-10-28 22:57:00 |
|  97 |        1 | heimei      | 2021-10-28 22:57:00 |
|  98 |        1 | xiaolingtong | 2021-10-28 22:57:00 |
|  99 |        1 | iphone      | 2021-10-28 22:57:00 |
| 100 |        1 | ipod        | 2021-10-28 22:57:00 |
| 101 |        1 | chuizi      | 2021-10-28 22:57:00 |
| 102 |        1 | heimei      | 2021-10-28 22:57:00 |
| 103 |        1 | xiaolingtong | 2021-10-28 22:57:00 |
| 104 |        1 | iphone      | 2021-10-28 22:57:00 |
| 105 |        1 | ipod        | 2021-10-28 22:57:00 |
| 106 |        1 | chuizi      | 2021-10-28 22:57:00 |
| 107 |        1 | heimei      | 2021-10-28 22:57:00 |
| 108 |        1 | xiaolingtong | 2021-10-28 22:57:00 |
| 109 |        1 | nokia       | 2021-10-28 22:57:00 |
| 110 |        1 | xiaolingtong | 2021-10-28 22:57:48 |
| 111 |        1 | iphone      | 2021-10-28 22:57:48 |
| 112 |        1 | ipod        | 2021-10-28 22:57:48 |
| 113 |        1 | chuizi      | 2021-10-28 22:57:48 |
| 114 |        1 | heimei      | 2021-10-28 22:57:48 |
| 115 |        1 | xiaolingtong | 2021-10-28 22:57:48 |
| 116 |        1 | nokia       | 2021-10-28 22:57:48 |
| 117 |        1 | xiaolingtong | 2021-10-28 22:57:48 |
```

在实验之前，我们考虑可能发生阻塞以及干扰实验数据的因素，并逐个排除，主要考虑以下方面：

1. 将 Redis、jvm、mysql 部署在三台服务器上，其中 Redis 部署在内存较大的 slave1 图形

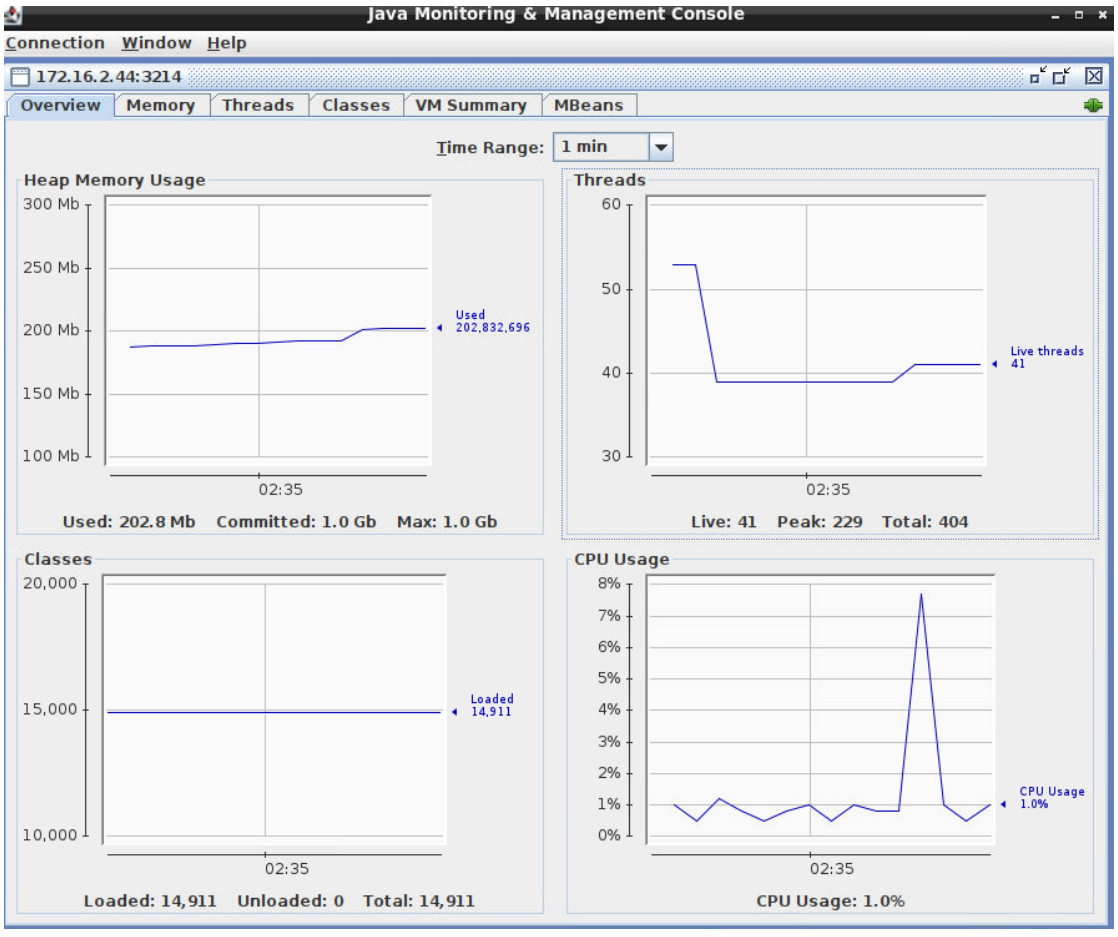
化界面服务器中（slave1 总内存为 4G）

2. Druid、Redis 设置相同的空闲连接数——100，测试计划的线程数不超过 50，方便观察与对照
3. 考虑到 GUI 模式下的 jmeter 测试数据并不准确，因此将实验计划导出为 jmx，使用 non-GUI 模式运行
4. 需要对比 Mybatis 和 Redis 缓存，因此我们查询同一 id 的规格，以便对比二者缓存的速度
5. 预热 JVM：先运行 1000 个线程的测试计划，让 jvm 预热，防止干扰
6. 在进行 Redis 测试时，**注意 key 的过期时间对实验结果的影响**，在 redis-cli 中使用 ttl + key 查看剩余过期时间

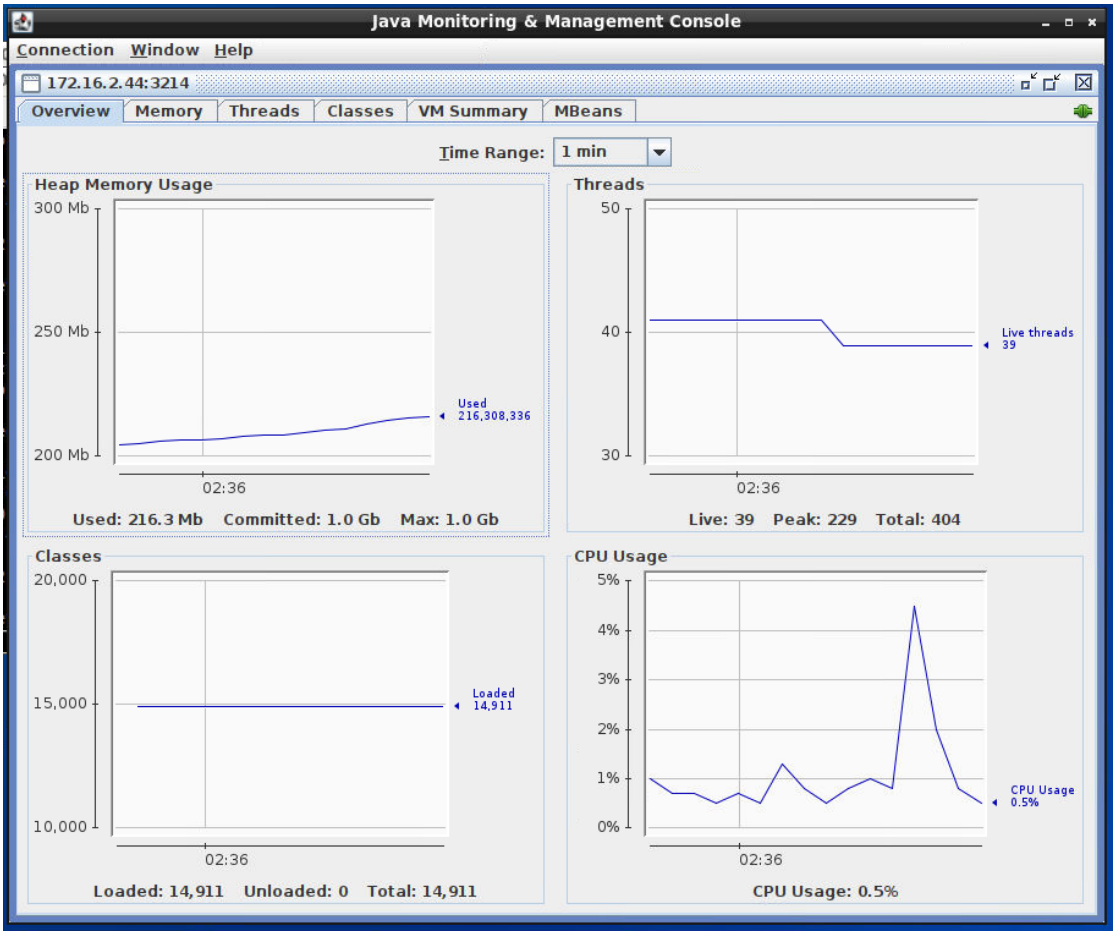
不使用 Redis 的 url：/products/100
使用 Redis 的 url：/products/redis/100

五、 具体测试与结果分析

不使用 Redis 下，应用服务器的负载情况：



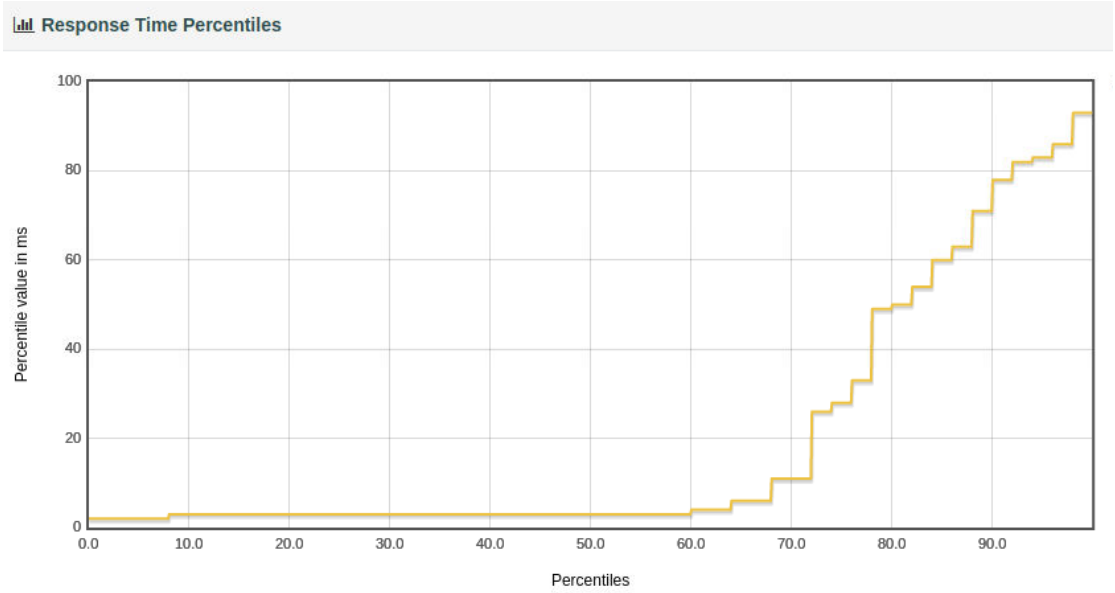
使用 Redis 的情况下，应用服务器的负载情况



可以看到，在两种情况下，应用服务器的负载是差不多的，均处于低负载情况
（因为我们在数据库层即完成了对象的关联，因此不会给予应用服务器很高的负载）

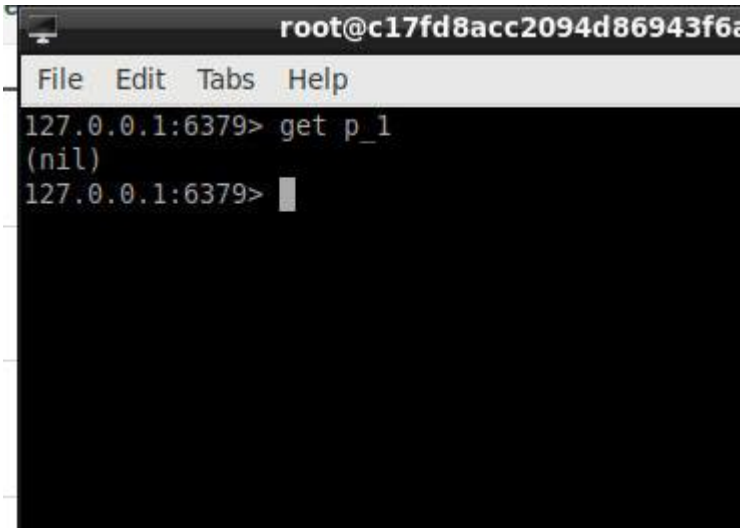
下面先进行不使用 Redis 的情况下的测试
测试线程数——50
测试结果：

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**50**	**0**	**0.00%**	**19.68**	**2**	**93**	**3.00**	**77.30**	**84.35**	**93.00**	**61.73**	**42.62**	**7.78**
HTTP Request	50	0	0.00%	19.68	2	93	3.00	77.30	84.35	93.00	61.73	42.62	7.78

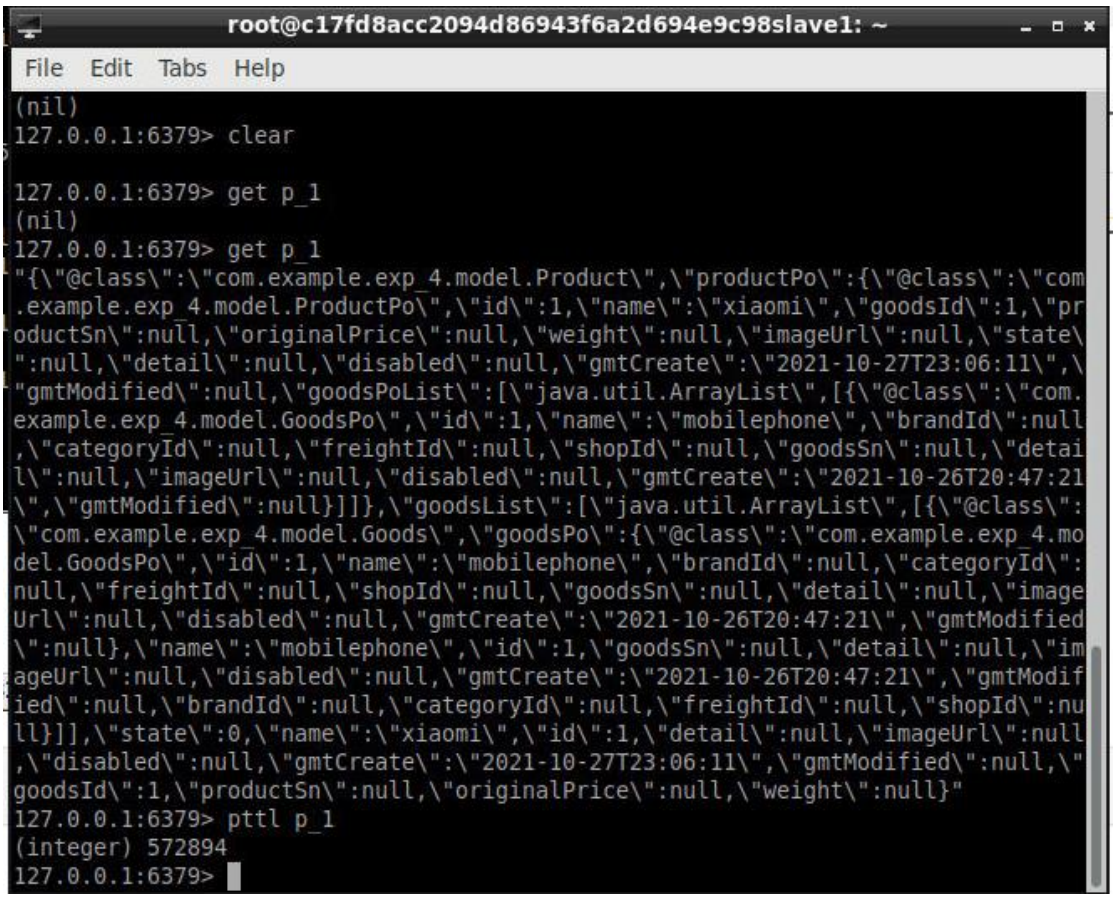


在不使用 Redis 的情况下，只是单纯的 MyBatis 读取，平均响应时间为 19.64ms
在响应时间百分比中，约 72%的响应时间低于 20ms，90%的响应时间在约 77ms 以内，我们可以认为阻塞边界大约是 50 * 70% = 35 个线程数

下面是使用 Redis 的情况下的测试：
先确保 Redis 没有对应键值：



运行测试计划之后，导出结果为 ResultWithRedis.jtl，此时在 Redis 命令行中获取键值，可以看到已经有对应的 value



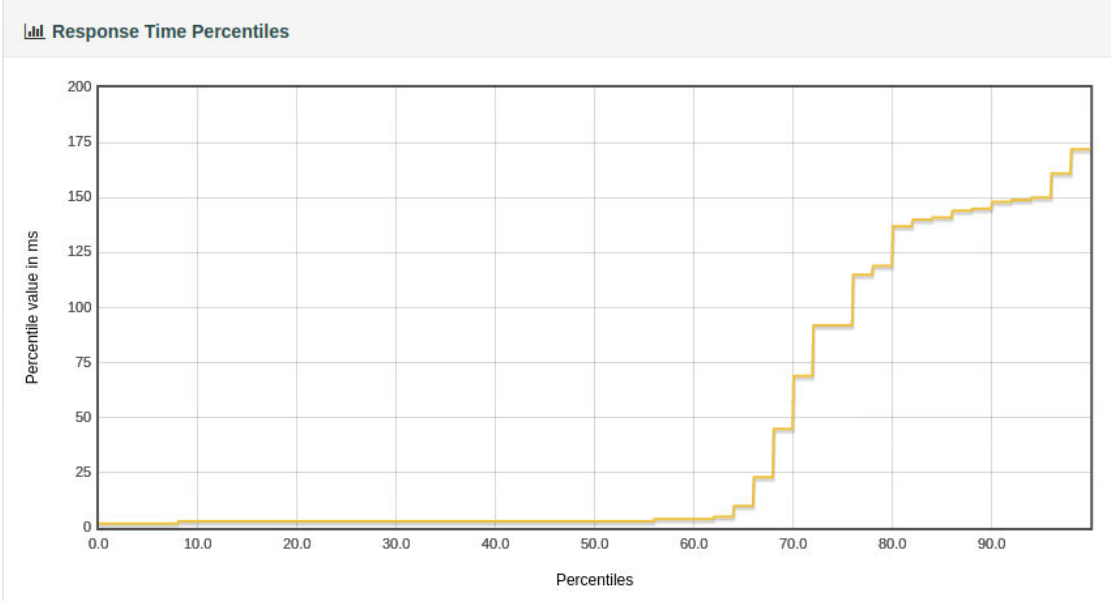
在 key 过期之前进行第二次测试，导出结果为 ResultWithRedis1.jtl

查看两次结果：

第一次：

Statistics

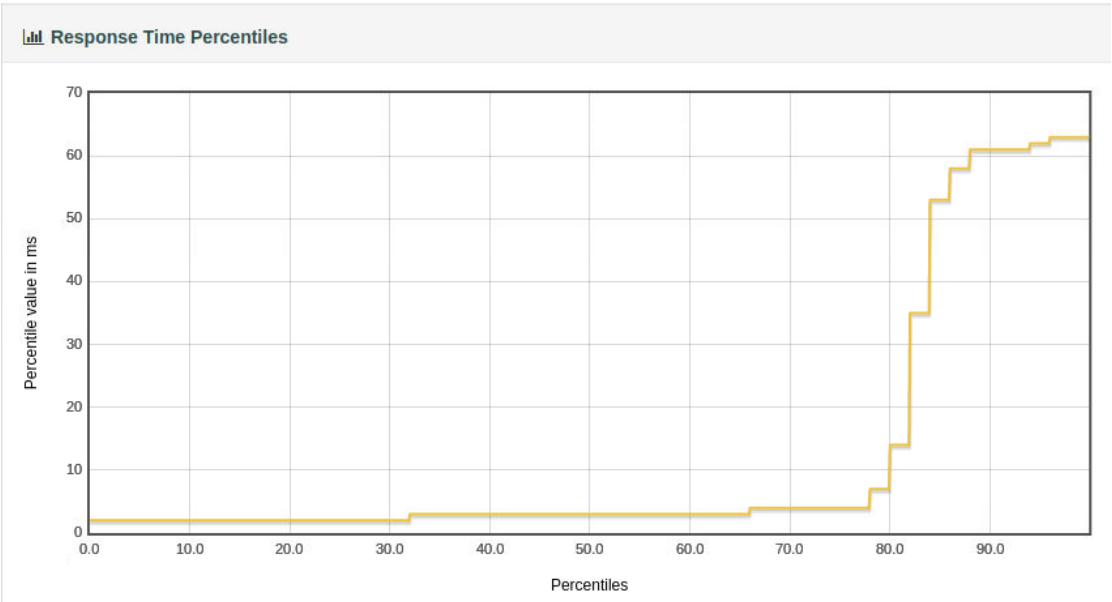
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	50	0	0.00%	42.98	2	172	3.00	147.70	154.95	172.00	63.13	43.59	8.32
HTTP Request	50	0	0.00%	42.98	2	172	3.00	147.70	154.95	172.00	63.13	43.59	8.32



第二次：

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**50**	**0**	**0.00%**	**12.90**	**2**	**63**	**3.00**	**61.00**	**62.45**	**63.00**	**60.17**	**41.54**	**7.93**
HTTP Request	50	0	0.00%	12.90	2	63	3.00	61.00	62.45	63.00	60.17	41.54	7.93



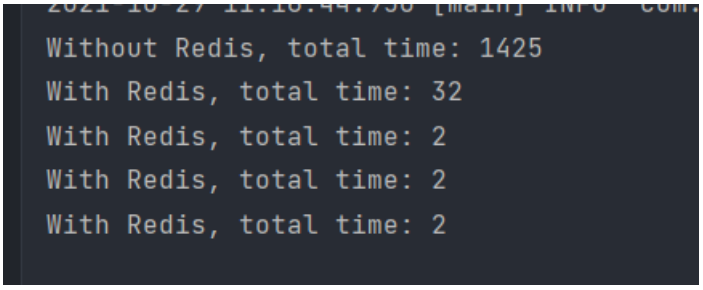
这两次中，可以明显看到，第二次的测试快于第一次的测试，原因可能在于在第一次的访问中，key 并没有对应键值，因此需要先访问数据库，拿到对应 value，然后存储到 redis 中，这样就会造成更长的响应时间

第二次测试中，因为 key 并未过期，因此每次需要访问对应数据时，直接从 Redis 中取出数据即可，不需要经过数据库，因此第二次测试比第一次测试更快，甚至快于之前的不使用 Redis

对于这个可能的原因，我们在本机代码中添加 currentTimeMills 进行测试，来验证我们的猜想：

```
public ReturnObject<List<Product>> findProductWithRedis(ProductPo productPo) {

    long startTime = System.currentTimeMillis();
    List<Product> retProduct = null;
    String key = null;
    if(null != productPo.getId()) {
        key = "p_" + productPo.getId();
        Product product = (Product) redisUtil.get(key);
        if(null != product) {
            retProduct = new ArrayList<>( initialCapacity: 1);
            retProduct.add(product);
            long endTime1 = System.currentTimeMillis();
            System.out.println("With Redis, total time: " + (endTime1 - startTime));
            return new ReturnObject<>(retProduct);
        }
    }

    List<ProductPo> productPoList = productMapper.findProduct(productPo);
    retProduct = new ArrayList<>(productPoList.size());

    for(ProductPo p : productPoList) {
        Product product = new Product(p);
        List<GoodsPo> goodsPoList = p.getGoodsPoList();
        List<Goods> goodsList = new ArrayList<>(goodsPoList.size());
        for(GoodsPo g : goodsPoList) {
            Goods goods = new Goods(g);
            goodsList.add(goods);
        }
        product.setGoodsList(goodsList);
        retProduct.add(product);
    }

    if(null != productPo.getId()) {
        if(retProduct.size() != 0) {
            redisUtil.set(key, retProduct.get(0), productTimeout);
        } else {
            redisUtil.set(key,  value: null, productTimeout);
        }
    }
    long endTime = System.currentTimeMillis();
    System.out.println("Without Redis, total time: " + (endTime - startTime));

    return new ReturnObject<>(retProduct);
}
```

使用 springboot:run，在浏览器中多次访问/products/redis/1，结果如下：



虽然这种手动测试方式得到的结果并不是准确结果，但至少可以说明，是否有 Redis，对于访问速度的有很大的影响

综上，我们可以得出结论：
Redis 在没有对应的 key 和 value 的情况下，访问时间会比不使用 Redis 更长
如果使用了 Redis，并且 Redis 的有缓存且缓存并未失效，那么使用 Redis 的速度更快
对于有多个规格的商品，是否使用 Redis 对查询速度几乎没有影响

六、 附加文件

github 地址：https://github.com/529106896/HeavyTeam/tree/main/Experiment/Exp_4
gitee 地址：https://gitee.com/yjz6666774/heavy-team/tree/master/Experiment/Exp_4
其余还包括 jtl 文件和 sql 文件，一并放在压缩包中
（提示：sql 文件是 mysql 8.0 直接导出，在一些低版本的 mysql 中无法使用，请保证版本一致，或者将对应的 utf-8 编码修改至低版本 mysql 适用的格式）