# 商品详情的方案

## 网页静态化方案

1、创建商品详情的thymeleaf模板

2、创建一个RabbitMQ的消费者，监听新增或修改商品的消息，收到消息后创建静态页面

3、搭建nginx服务器，返回静态页面

## redis缓存商品信息方案

使用redis做缓存，业务逻辑：

1、根据商品id到redis中查询

​ 查得到，直接返回

2、查不到，查询mysql，

​ 数据放到redis中

​ 设置缓存的有效期一天的时间，可以根据实际情况调整。

需要使用String类型来保存商品数据，可以加前缀对redis中的key进行归类：

ITEM\_INFO:123456:BASE

ITEM\_INFO:123456:DESC

ITEM\_INFO:123456:PARAM

# 查询商品详情

创建usian\_detail\_web

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>usian\_parent</artifactId>  <groupId>com.usian</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>usian\_detail\_web</artifactId>  <dependencies>  <dependency>  <groupId>com.usian</groupId>  <artifactId>usian\_item\_feign</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>    </dependencies>  </project> |

Application.yml

|  |
| --- |
| spring:  application:  name: usian-detail-web  server:  port: 8097  eureka:  client:  service-url:  defaultZone: http://127.0.0.1:8761/eureka/ |

logback.xml

启动类

## 查询商品信息

usian\_detail\_web

|  |
| --- |
| @RestController  @RequestMapping("/frontend/detail")  public class DetailController {  @Autowired  private ItemServiceFeign itemServiceFeign;  /\*\*  \* 查询商品基本信息  \*/  @RequestMapping("/selectItemInfo")  public Result selectItemInfo(Long itemId) {  TbItem tbItem = itemServiceFeign.selectItemInfo(itemId);  if (tbItem != null) {  return Result.ok(tbItem);  }  return Result.error("查无结果");  }  /\*\*  \* 查询商品介绍  \*/  @RequestMapping("/selectItemDescByItemId")  public Result selectItemDescByItemId(Long itemId){  TbItemDesc tbItemDesc = itemServiceFeign.selectItemDescByItemId(itemId);  if(tbItemDesc != null){  return Result. ok (tbItemDesc);  }  return Result.error ("查无结果");  }  /\*\*  \* 根据商品 ID 查询商品规格参数  \*/  @RequestMapping("/selectTbItemParamItemByItemId")  public Result selectTbItemParamItemByItemId(Long itemId){  TbItemParamItem tbItemParamItem =  itemServiceFeign.selectTbItemParamItemByItemId(itemId);  if(tbItemParamItem != null){  return Result.ok(tbItemParamItem);  }  return Result. error ("查无结果");  }  } |

usian\_item\_service

application.yml：

|  |
| --- |
| #商品详情  ITEM\_INFO: ITEM\_INFO  BASE: BASE  DESC: DESC  PARAM: PARAM  ITEM\_INFO\_EXPIRE: 86400 |

Service：

|  |
| --- |
| /\*\*  \* 查询商品信息  \* @param itemId  \* @return  \*/  @Override  public TbItem selectItemInfo(Long itemId) {  //查询缓存  TbItem tbItem = (TbItem) redisClient.get(ITEM\_INFO + ":" + itemId + ":"+ BASE);  if(tbItem!=null){  return tbItem;  }  tbItem = tbItemMapper.selectByPrimaryKey(itemId);  //把数据保存到缓存  redisClient.set(ITEM\_INFO + ":" + itemId + ":"+ BASE,tbItem);  //设置缓存的有效期  redisClient.expire(ITEM\_INFO + ":" + itemId + ":"+ BASE,ITEM\_INFO\_EXPIRE);  return tbItem;  } |
| /\*\*  \* 根据商品 ID 查询商品描述  \* @param itemId  \* @return  \*/  @Override  public TbItemDesc selectItemDescByItemId(Long itemId) {  //查询缓存  TbItemDesc tbItemDesc = (TbItemDesc) redisClient.get(  ITEM\_INFO + ":" + itemId + ":"+ DESC);  if(tbItemDesc!=null){  return tbItemDesc;  }  TbItemDescExample example = new TbItemDescExample();  TbItemDescExample.Criteria criteria = example.createCriteria();  criteria.andItemIdEqualTo(itemId);  List<TbItemDesc> itemDescList =  this.tbItemDescMapper.selectByExampleWithBLOBs(example);  if(itemDescList!=null && itemDescList.size()>0){  //把数据保存到缓存  redisClient.set(ITEM\_INFO + ":" + itemId + ":"+ DESC,itemDescList.get(0));  //设置缓存的有效期  redisClient.expire(ITEM\_INFO + ":" + itemId + ":"+ DESC,ITEM\_INFO\_EXPIRE);  return itemDescList.get(0);  }  return null;  } |
| /\*\*  \* 根据商品id查询商品规格  \* @param itemId  \* @return  \*/  @Override  public TbItemParamItem selectTbItemParamItemByItemId(Long itemId) {  //查询缓存  TbItemParamItem tbItemParamItem = (TbItemParamItem) redisClient.get(  ITEM\_INFO + ":" + itemId + ":"+ PARAM);  if(tbItemParamItem!=null){  return tbItemParamItem;  }  TbItemParamItemExample example = new TbItemParamItemExample();  TbItemParamItemExample.Criteria criteria = example.createCriteria();  criteria.andItemIdEqualTo(itemId);  List<TbItemParamItem> tbItemParamItemList =  tbItemParamItemMapper.selectByExampleWithBLOBs(example);  if(tbItemParamItemList!=null && tbItemParamItemList.size()>0){  //把数据保存到缓存  redisClient.set(ITEM\_INFO + ":" + itemId + ":"+ PARAM,tbItemParamItemList.get(0));  //设置缓存的有效期  redisClient.expire(ITEM\_INFO + ":" + itemId + ":"+ PARAM,ITEM\_INFO\_EXPIRE);  return tbItemParamItemList.get(0);  }  return null;  } |

## 缓存同步

使用消息中间件实现redis中商品信息的同步：后台修改商品则直接删除redis中的商品

## 缓存穿透

缓存穿透是指缓存和数据库中都没有的数据，而用户不断发起请求，如发起为id为“-1”的数据或id为特别大不存在的数据。假如有恶意攻击，就可以利用这个漏洞，对数据库造成压力，甚至压垮数据库。

解决方案：

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（避免控制占用更多的存储空间），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；

## 缓存击穿

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。

解决方案：

1. 设置热点数据永远不过期。

2. 加分布式锁

## 缓存雪崩

缓存雪崩，是指在某一个时间段，缓存集中过期失效。

解决方案：缓存数据的过期时间设置随机，不同分类商品缓存不同周期或热门类目的商品缓存时间长一些