**谷粒商城**

版本：V 2.0

**商品详情页**

# 一 业务介绍

商品详情页，简单说就是以购物者的角度展现一个sku的详情信息。

这个页面不同于传统的crud的详情页，使用者并不是管理员，需要对信息进行查删改查，取而代之的是点击购买、放入购物车、切换颜色等等。

另外一个特点就是该页面的高访问量，虽然只是一个查询操作，但是由于频繁的访问所以我们必须对其性能进行最大程度的优化。

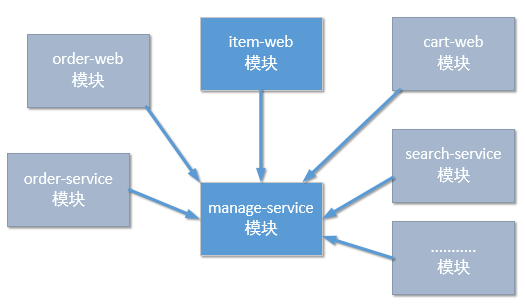
# 二 难点分析

1 光从功能角度上来说，并没有太多难点，唯一实现起来麻烦的就是用户对于不同销售属性的切换操作。



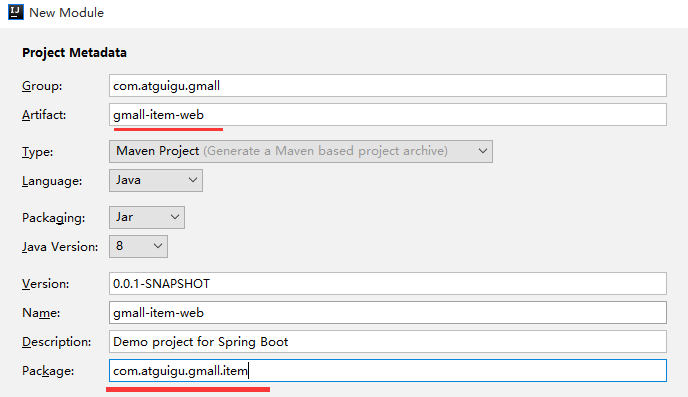
2 从性能角度来看，需要最大程度的提升页面的访问速度。

# 三 功能开发

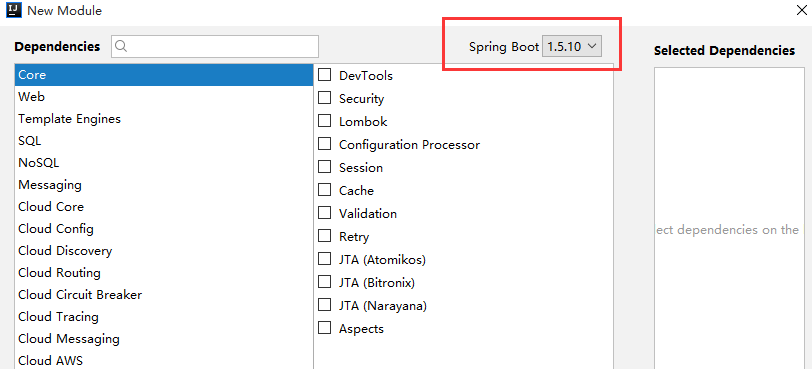


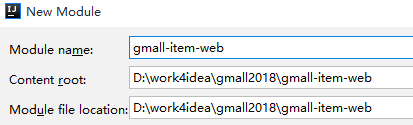
详情页功能，只增加一个web模块，后台调用商品管理的模块manage-service

item-web模块负责前端的页面渲染和控制层(controller)。



依赖包全都不选，SpringBoot版本选1.5.10



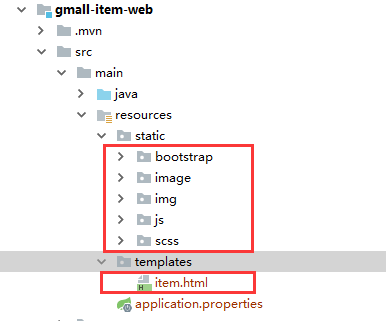


pom.xml

|  |
| --- |
| *<?*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">  <modelVersion>4.0.0</modelVersion>   <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-item-web</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>gmall-item-web</name>  <description>Demo project for Spring Boot</description>   <parent>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-parent</artifactId>  <version>1.0-SNAPSHOT</version>  </parent>    <dependencies>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-web-util</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>   </project> |

搭建完成后。

首先导入前端页面



静态页资源全部拷贝到static目录中，如果没有该目录请手工创建

动态的html文件拷贝到templates目录中

编写Controller类（入口方法）

|  |
| --- |
| @Controller **public class** ItemController {   @RequestMapping(**"/{skuId}.html"**)  **public** String getSkuInfo(@PathVariable(**"skuId"**) String skuId){  **return "item"**;  }  } |

application.properties

|  |
| --- |
| **server.port**=**8084  spring.thymeleaf.cache**=**false  spring.thymeleaf.mode**=**LEGACYHTML5** |

修改item.html改为绝对路径

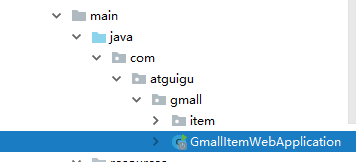
把item.html 所有src=”./ 替换成sr=”/

<link>标签的路径也改为/开头

|  |
| --- |
| <**head**>  <**meta charset="UTF-8"**>  <**title**></**title**>  <**link rel="stylesheet" type="text/css" href="/scss/shop.css"** />  <**link rel="stylesheet" type="text/css" href="/scss/main.css"**/>  <**link rel="stylesheet" href="/scss/header.css"** />  <**link rel="stylesheet" type="text/css" href="/bootstrap/css/bootstrap.css"**/> </**head**> |

把启动类GmallItemWebApplication提到和item平级的目录中。

或者增加@ComponentScan(basePackages = **"com.atguigu.gmall"**)



然后启动服务

启动测试



可以看到商品详情页的静态页面。

后台实现在gmall-item-web模块中

增加ItemController

|  |
| --- |
| @RequestMapping(**"/{skuId}.html"**) **public** String getSkuInfo(@PathVariable(**"skuId"**) String skuId, Model model){  SkuInfo skuInfo = **manageService**.getSkuInfo(skuId); model.addAttribute(**"skuInfo"**,skuInfo);  } |

gmall-manage-service中增加

后台实现类

|  |
| --- |
| **public** SkuInfo getSkuInfo(String skuId){   SkuInfo skuInfo = **skuInfoMapper**.selectByPrimaryKey(skuId);  **if**(skuInfo==**null**){  **return null**;  }  SkuImage skuImage=**new** SkuImage();  skuImage.setSkuId(skuId);  List<SkuImage> skuImageList = **skuImageMapper**.select(skuImage);  skuInfo.setSkuImageList(skuImageList);   SkuSaleAttrValue skuSaleAttrValue=**new** SkuSaleAttrValue();  skuSaleAttrValue.setSkuId(skuId);  List<SkuSaleAttrValue> skuSaleAttrValueList = **skuSaleAttrValueMapper**.select(skuSaleAttrValue);  skuInfo.setSkuSaleAttrValueList(skuSaleAttrValueList);   **return** skuInfo; } |

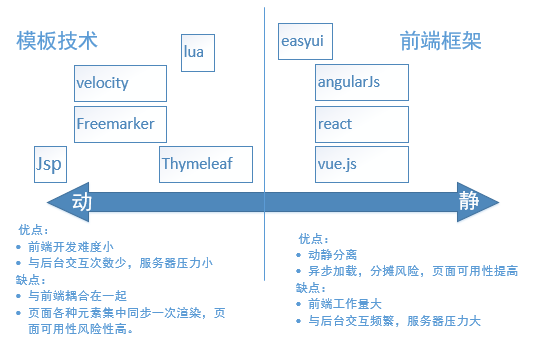
# 四 Thymeleaf

## 1 模板技术介绍

把页面中的静态数据替换成从后台数据库中的数据。这种操作用jsp就可以实现。但是Springboot 的架构不推荐使用Jsp，而且支持也不好，所以如果你是用springboot的话，一般使用Freemarker或者Thymeleaf。

而官方也是推荐使用Thymeleaf。

关于与前端有关的技术的比较



## 2 springboot亲儿子Themeleaf简介

Thymeleaf的主要目标是提供一个优雅和高度可维护的创建模板的方式。为了实现这一点，它建立在自然模板的概念上，将其逻辑注入到模板文件中，不会影响模板被用作设计原型。这改善了设计的沟通，弥合了设计和开发团队之间的差距。

比Jsp和Freemarker的优势，一般的模板技术都会在页面加各种表达式、标签甚至是java代码，而这些都必须要经过后台服务器的渲染才能打开。

但如果前端开发人员做页面调整，双击打开某个jsp或者ftl来查看效果，基本上是打不开的。

那么Thymeleaf的优势就出来了，因为Thymeleaf没有使用自定义的标签或语法，所有的模板语言都是扩展了标准H5标签的属性

比如

|  |
| --- |
| <**div th:text="${item.skuName} "**></div> |

它的效果和Jsp中的

|  |
| --- |
| <div>**${item.skuName}**</div> |

渲染后效果一样，但是如果你直接用浏览器打开页面文件，H5会把th:text这种不认识的属性忽略掉。效果就和<div></div> 没有区别，所以对于前端调页面影响更新。以上只是举了一个例子，如果是循环、分支的判断效果更明显。

## 3 快速入门：

### 3.1 所有头文件

就行Jsp的<%@Page %>一样 ，Thymeleaf的也要引入标签规范。不加这个虽然不影响程序运行，但是你的idea会认不出标签，不方便开发。

|  |
| --- |
| <**html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"**> |

### 3.2 取出请求域中的值，即取得request.attribute中的值

|  |
| --- |
| <p th:text="${hello}">打底值</p> |

### 3.3 循环

|  |
| --- |
| <table border="1">  <tr th:each="skuImage:${skuInfo.skuImageList}">  <td th:text="${skuImage.id}">  </td>  <td th:text="${skuImage.imgName}">  </td>  </tr> </table> |

### 3.4 判断

|  |
| --- |
| <td th:text="(${skuImage.id}=='10')?'是10':'不是10'"> </td> <td th:if="${skuImage.id}=='10'" >123456</td> |

### 3.5 取session中的属性

|  |
| --- |
| <div th:text="${session.userName}"> </div> |

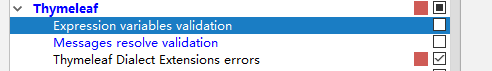
### 3.6 引用内嵌页

|  |
| --- |
| <div th:include="itemInner"/> |

如果idea编辑器质疑你页面中的元素是否存在



通常不准确，可以去掉验证，在settings->editor->Inspections中关掉验证。



# 五 详情页html

名称：

|  |
| --- |
| <**div class="box-name" th:text="${skuInfo.skuName}"**>  华为 HUAWEI Mate 10 6GB+128GB 亮黑色 移动联通电信4G手机 双卡双待 </**div**> |

价格：

|  |
| --- |
| <**span th:text="${#numbers.formatDecimal(skuInfo.price,1,2)}"**>4499.00</**span**> |

重量：

|  |
| --- |
| <**li th:text="${#numbers.formatDecimal(skuInfo.weight,1,2)}+' kg'"**></**li**> |

图片

|  |
| --- |
| <**div class="box-lh-one"**>  <**ul**>  <**li th:each="skuImage:${skuInfo.skuImageList}"**>  <**img th:src="${skuImage.imgUrl}"**/>  </**li**>  </**ul**> </**div**> |

**numbers.formatDecimal(<值>,<小数点左边的占位>,<小数点右边的保留位>)**

# 六 如何销售属性切换当前商品sku

## 1 思路：

1、查出该商品的spu的所有销售属性和属性值

2、标识出本商品对应的销售属性

3、点击其他销售属性值的组合，跳转到另外的sku页面

|  |
| --- |
| SELECT  sa.id,  sa.spu\_id,  sa.sale\_attr\_name,  sa.sale\_attr\_id,  sv.id sale\_attr\_value\_id,  sv.sale\_attr\_value\_name,  skv.sku\_id,  if(skv.sku\_id is not null,1,0) as isCheck  FROM  spu\_sale\_attr sa inner JOIN  spu\_sale\_attr\_value sv  on sa.sale\_attr\_id = sv.sale\_attr\_id and sa.spu\_id = sv.spu\_id and sa.spu\_id = 25  LEFT JOIN sku\_sale\_attr\_value skv  on skv.sale\_attr\_id = sa.sale\_attr\_id and skv.sale\_attr\_value\_id = sv.id and skv.sku\_id = 47  ORDER BY  sv.sale\_attr\_id |

|  |
| --- |
| SELECT skv.\*  FROM sku\_sale\_attr\_value skv  INNER JOIN sku\_info sk ON skv.sku\_id =sk.id and sk.spu\_id=25  ORDER BY skv.sku\_id , skv.sale\_attr\_id  select skv.\* from sku\_sale\_attr\_value skv , sku\_info sku  where skv.sku\_id =sku.id and sku.spu\_id=25  ORDER BY sku.id , skv.sale\_attr\_id |

## 2 查询出sku对应spu的销售属性

第1、2条通过此sql实现

|  |
| --- |
| SELECT sa.id ,sa.spu\_id, sa.sale\_attr\_name,sa.sale\_attr\_id,  sv.id sale\_attr\_value\_id,  sv.sale\_attr\_value\_name,  skv.sku\_id,  IF(skv.sku\_id IS NOT NULL,1,0) is\_check  FROM spu\_sale\_attr sa  INNER JOIN spu\_sale\_attr\_value sv ON sa.spu\_id=sv.spu\_id AND sa.sale\_attr\_id=sv.sale\_attr\_id  LEFT JOIN sku\_sale\_attr\_value skv ON skv.sale\_attr\_id= sa.sale\_attr\_id AND skv.sale\_attr\_value\_id=sv.id AND skv.sku\_id=10  WHERE sa.spu\_id=24  ORDER BY sv.sale\_attr\_id,sv.id |

此sql列出所有该spu的销售属性和属性值，并关联某skuid如果能关联上is\_check设为1，否则设为0。

页面开发部分

|  |
| --- |
| <div class="box-attr-2 clear" th:each="spuSaleAttr:${spuSaleAttrListCheckBySku}">  <dl>  <dt th:text="${spuSaleAttr.saleAttrName}">选择属性</dt>  <dd th:class="(${saleAttrValue.isCheck}=='1')?'redborder':''" th:each="saleAttrValue:${spuSaleAttr.spuSaleAttrValueList}">  <div th:value="${saleAttrValue.id}" th:text="${saleAttrValue.saleAttrValueName}">  属性值  </div>  </dd>  </dl> </div> |

其中th:class的设置，redborder是一个自定义的样式类，主要是边框设红。如果isCheck=1标识当前这个sku的所拥有的属性值，所以锁定为红边框。



需要增加修改的代码文件清单

|  |  |
| --- | --- |
| 增加SpuSaleAttrMapper.xml的方法 | selectSpuSaleAttrListCheckBySku  注意双参数的处理 |
| 增加 SpuSaleAttrMapper的方法 |  |
| 增加 ManageServiceImpl的方法 | getSpuSaleAttrListCheckBySku |
| 增加ManageService的方法 | getSpuSaleAttrListCheckBySku |
|  |  |
|  |  |
| 修改 SpuSaleAttrValue | 增加isCheck属性 |
|  |  |

代码见代码清单

|  |  |
| --- | --- |
| 增加 ManageServiceImpl的方法 | getSpuSaleAttrListCheckBySku |
| @Override **public** List<SpuSaleAttr> getSpuSaleAttrListCheckBySku(String skuId,String spuId){   List<SpuSaleAttr> spuSaleAttrList = **spuSaleAttrMapper**.selectSpuSaleAttrListCheckBySku(Long.*parseLong*(skuId),Long.*parseLong*(spuId));  **return** spuSaleAttrList;  } | |

|  |
| --- |
| 增加 SpuSaleAttrMapper的方法 |
| public List<SpuSaleAttr> selectSpuSaleAttrListCheckBySku(long skuId,long spuId); |

|  |  |
| --- | --- |
| 增加SpuSaleAttrMapper.xml的方法 | selectSpuSaleAttrListCheckBySku  注意双参数的处理 |
| <**select id ="selectSpuSaleAttrListCheckBySku" resultMap="spuSaleAttrMap"**>  SELECT sa.id ,sa.spu\_id, sa.sale\_attr\_name,sa.sale\_attr\_id,  sv.id sale\_attr\_value\_id,  sv.sale\_attr\_value\_name,  skv.sku\_id,  IF(skv.sku\_id IS NOT NULL,1,0) is\_check  FROM spu\_sale\_attr sa  INNER JOIN spu\_sale\_attr\_value sv ON sa.spu\_id=sv.spu\_id AND sa.sale\_attr\_id=sv.sale\_attr\_id  LEFT JOIN sku\_sale\_attr\_value skv ON skv.sale\_attr\_id= sa.sale\_attr\_id AND skv.sale\_attr\_value\_id=sv.id AND skv.sku\_id=#{arg0}  WHERE sa.spu\_id=#{arg1}  ORDER BY sv.sale\_attr\_id,sv.id </**select**> | |

|  |  |
| --- | --- |
| 修改 SpuSaleAttrValue | 增加isCheck属性 |

|  |  |
| --- | --- |
| 修改itemController | 调用getSpuSaleAttrListCheckBySku |
| List<SpuSaleAttr> spuSaleAttrListCheckBySku = **manageService**.getSpuSaleAttrListCheckBySku(skuInfo.getId(), skuInfo.getSpuId()); model.addAttribute(**"spuSaleAttrListCheckBySku"**,spuSaleAttrListCheckBySku); | |

|  |  |
| --- | --- |
| 修改item.html | 增加渲染的代码 |
| <div class="box-attr-2 clear" th:each="spuSaleAttr:${spuSaleAttrListCheckBySku}">  <dl>  <dt th:text="${spuSaleAttr.saleAttrName}">选择颜色</dt>  <dd th:class="(${saleAttrValue.isCheck}=='1')?'redborder':''" th:each="saleAttrValue:${spuSaleAttr.spuSaleAttrValueList}">  <div th:value="${saleAttrValue.id}" th:text="${saleAttrValue.saleAttrValueName}">  摩卡金  </div>  </dd>  </dl> </div> | |

## 3 点击其他销售属性值的组合，跳转到另外的sku页面

实现思路：

1 、从页面中获得得所有选中的销售属性进行组合比如：

“属性值1|属性值2|属性值3” 用这个字符串匹配一个对照表，来获得skuId。并进行跳转，或者告知无货。

2、后台要生成一个“属性值1|属性值2|属性值3：skuId”的一个json串以提供页面进行匹配。如

3、需要从后台数据库查询出该spu下的所有skuId和属性值关联关系。然后加工成如上的Json串。

实现：

skuSaleAttrValueMapper.xml中sql

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>* <mapper namespace="com.atguigu.gmall.manage.mapper.SkuSaleAttrValueMapper">  <select id ="selectSkuSaleAttrValueListBySpu" parameterType="long" resultMap="skuSaleAttrValueMap">  SELECT skv.\*  FROM sku\_sale\_attr\_value skv  INNER JOIN sku\_info sk ON skv.sku\_id =sk.id  where sk.spu\_id=#{spuId}  ORDER BY skv.sku\_id , skv.sale\_attr\_id  </select>  <resultMap id="skuSaleAttrValueMap" type="com.atguigu.gmall.bean.SkuSaleAttrValue" autoMapping="true">  <result property="id" column="id" ></result>  </resultMap> </mapper> |

要注意排序，方便后面整理。

实现类很简单：

|  |
| --- |
| **public** List<SkuSaleAttrValue> getSkuSaleAttrValueListBySpu(String spuId){  List<SkuSaleAttrValue> skuSaleAttrValueList = **skuSaleAttrValueMapper**.selectSkuSaleAttrValueListBySpu(Long.*parseLong*(spuId));  **return** skuSaleAttrValueList; } |

难点是整理成咱们要求的json串，这个写在controller类。

|  |
| --- |
| List<SkuSaleAttrValue> skuSaleAttrValueListBySpu  = **manageService**.getSkuSaleAttrValueListBySpu(skuInfo.getSpuId());  *//把列表变换成 valueid1|valueid2|valueid3 ：skuId 的 哈希表 用于在页面中定位查询*  String valueIdsKey=**""**;  Map<String,String> valuesSkuMap=**new** HashMap<>();  **for** (**int** i = 0; i < skuSaleAttrValueListBySpu.size(); i++) {  SkuSaleAttrValue skuSaleAttrValue = skuSaleAttrValueListBySpu.get(i);  **if**(valueIdsKey.length()!=0){  valueIdsKey= valueIdsKey+**"|"**;  }  valueIdsKey=valueIdsKey+skuSaleAttrValue.getSaleAttrValueId();   **if**((i+1)== skuSaleAttrValueListBySpu.size()||!skuSaleAttrValue.getSkuId().equals(skuSaleAttrValueListBySpu.get(i+1).getSkuId()) ){   valuesSkuMap.put(valueIdsKey,skuSaleAttrValue.getSkuId());  valueIdsKey=**""**;  }  }  *//把map变成json串* String valuesSkuJson = JSON.*toJSONString*(valuesSkuMap);  model.addAttribute("valuesSkuJson",valuesSkuJson); |

在item中增加隐藏域

|  |
| --- |
| <**input id="valuesSku" type="hidden" th:value= " ${valuesSkuJson}"** /> <**input id="skuId" type="hidden" th:value= " ${skuInfo.id}"** /> |

valuesSku存储属性值与skuid的对照json

skuId存放当前商品的skuId.

修改页面的js

|  |
| --- |
| *//红边框* **$**(**".box-attr-2 dd"**).click(**function**() {  **$**(**this**).addClass(**"redborder"**).siblings(**"dd"**).removeClass(**"redborder"**);  *switchSkuId*(); }) |

增加点击销售属性时触发切换sku方法

|  |
| --- |
| **function** *switchSkuId*() { **var** redborderDivs = **$**(**".redborder div"**); **var** valueIdkeys=**""**; **for**(**i**=0;**i**<redborderDivs.**length**;**i**++){  **var** redborderDiv= redborderDivs.eq(**i**);  **var** attrValueId = redborderDiv.attr(**"value"**);  **if**(**i**>0){  valueIdkeys+=**"|"**;  }  valueIdkeys+=attrValueId; } **console**.log(**"valueIdkeys:"**+valueIdkeys);  **var** valueIdSkuJson= **$**(**"#valueIdSkuJson"**).val();  **console**.log(**"valueIdSkuJson:"**+valueIdSkuJson);  **var** skuSelfId= **$**(**"#skuId"**).val();   **var** valueIdSku = ***JSON***.parse(valueIdSkuJson);  **var** skuIdTarget = valueIdSku[valueIdkeys];  **if**(!skuIdTarget){  **$**(**"#cartBtn"**).attr(**"class"**,**"box-btns-two-off"**);  **$**(**"#cartBtn"**).attr(**"canClick"**,**'0'**);  **$**(**"#cartBtn"**).css(**"cursor"**,**'not-allowed'**) ; }**else**{  **if**(skuSelfId!=skuIdTarget){  **window**.**location**.**href**=**"/"**+skuIdTarget+**".html"**;  }**else**{  **$**(**"#cartBtn"**).attr(**"class"**,**"box-btns-two"**);  **$**(**"#cartBtn"**).attr(**"canClick"**,**'1'**);  **$**(**"#cartBtn"**).css(**"cursor"**,**'pointer'**) ;  }  }   }  } |

最后测试：



# 七 性能优化

## 1 思路：

虽然咱们实现了页面需要的功能，但是考虑到该页面是被用户高频访问的，所以性能必须进行尽可能的优化。

一般一个系统最大的性能瓶颈，就是数据库的io操作。从数据库入手也是调优性价比最高的切入点。

一般分为两个层面，一是提高数据库sql本身的性能，二是尽量避免直接查询数据库。

提高数据库本身的性能首先是优化sql，包括：使用索引，减少不必要的大表关联次数，控制查询字段的行数和列数。另外当数据量巨大是可以考虑分库分表，以减轻单点压力。

这部分知识在mysql高级已有讲解，这里大家可以以详情页中的sql作为练习，尝试进行优化，这里不做赘述。

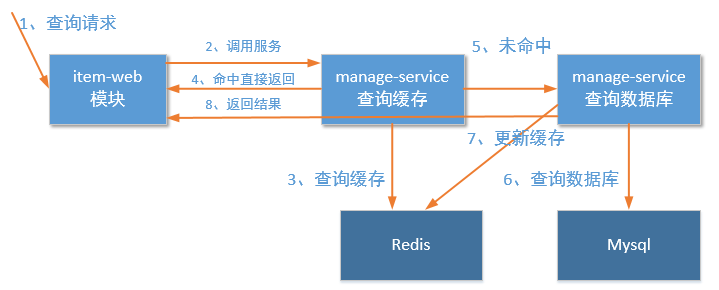
重点要讲的是另外一个层面：尽量避免直接查询数据库。

解决办法就是：缓存

缓存可以理解是数据库的一道保护伞，任何请求只要能在缓存中命中，都不会直接访问数据库。而缓存的处理性能是数据库10-100倍。

咱们就用Redis作为缓存系统进行优化。

结构图：



安装Redis ： 略

## 2 整合redis到工程中

由于redis作为缓存数据库，要被多个项目使用，所以要制作一个通用的工具类，方便工程中的各个模块使用。

而主要使用redis的模块，都是后台服务的模块，xxx-service工程。所以咱们把redis的工具类放到service-util模块中，这样所有的后台服务模块都可以使用redis。

首先引入依赖包

|  |
| --- |
| <!-- https://mvnrepository.com/artifact/redis.clients/jedis -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency> |

分别按照之前的方式放到parent模块和service-util的pom文件中。

然后在service-util中创建两个类RedisConfig和RedisUtil

RedisConfig负责在spring容器启动时自动注入，而RedisUtil就是被注入的工具类以供其他模块调用。

### 1 RedisUtil

|  |
| --- |
| **public class** RedisUtil {   **private** JedisPool **jedisPool**;   **public void** initPool(String host,**int** port ,**int** database){  JedisPoolConfig poolConfig = **new** JedisPoolConfig();  poolConfig.setMaxTotal(200);  poolConfig.setMaxIdle(30);  poolConfig.setBlockWhenExhausted(**true**);  poolConfig.setMaxWaitMillis(10\*1000);  poolConfig.setTestOnBorrow(**true**);  **jedisPool**=**new** JedisPool(poolConfig,host,port,20\*1000);  }   **public** Jedis getJedis(){  Jedis jedis = **jedisPool**.getResource();  **return** jedis;  }  } |

### 2 RedisConfig

|  |
| --- |
| @Configuration public class RedisConfig {   *//读取配置文件中的redis的ip地址* @Value("${spring.redis.host:disabled}")  private String host;   @Value("${spring.redis.port:0}")  private int port;   @Value("${spring.redis.database:0}")  private int database;   @Bean  public RedisUtil getRedisUtil(){  if(host.equals("disabled")){  return null;  }  RedisUtil redisUtil=new RedisUtil();  redisUtil.initPool(host,port,database);  return redisUtil;  }  } |

同时，任何模块想要调用redis都必须在application.properties配置，否则不会进行注入。

|  |
| --- |
| **spring.redis.host**=redis.server.com **spring.redis.port**=**6379 spring.redis.database**=**0** |

现在可以在manage-service中的getSkuInfo()方法测试一下

|  |
| --- |
| **try** {  Jedis jedis = **redisUtil**.getJedis();  jedis.get(**"test"**,**"text\_value"** ); }**catch** (JedisConnectionException e){  e.printStackTrace(); } |

## 3 使用redis进行业务开发

开始开发先说明redis key的命名规范，由于Redis不像数据库表那样有结构，其所有的数据全靠key进行索引，所以redis数据的可读性，全依靠key。

企业中最常用的方式就是：object:id:field

比如：sku:1314:info

user:1092:password

重构getSkuInfo方法

|  |
| --- |
| **public** SkuInfo getSkuInfo(String skuId){   Jedis jedis = **redisUtil**.getJedis();  String skuKey= RedisConst.***sku\_prefix***+skuId+RedisConst.***skuInfo\_suffix***;  String skuInfoJson = jedis.get(skuKey);  **if**(skuInfoJson!=**null** ){  System.***err***.println( Thread.*currentThread*().getName()+**"：命中缓存"** );  SkuInfo skuInfo = JSON.*parseObject*(skuInfoJson, SkuInfo.**class**);  jedis.close();  **return** skuInfo;  }**else**{  System.***err***.println( Thread.*currentThread*().getName()+**"：未命中缓存"** );   System.***err***.println( Thread.*currentThread*().getName()+**"： 查询数据##################### ##"** );  SkuInfo skuInfoDB = getSkuInfoDB(skuId);  String skuInfoJsonStr = JSON.*toJSONString*(skuInfoDB);  jedis.setex(skuKey,RedisConst.***skuinfo\_exp\_sec***,skuInfoJsonStr);  System.***err***.println( Thread.*currentThread*().getName()+**"：数据库更新完毕############### #####"** );  jedis.close();  **return** skuInfoDB;  } } |

以上基本实现使用缓存的方案。

## 4 高并发时可能会出现的问题

但在高并发环境下还有如下三个问题。

1. 如果redis宕机了，或者链接不上，怎么办？
2. 如果redis缓存在高峰期到期失效，在这个时刻请求会向雪崩一样，直接访问数据库如何处理？
3. 如果用户不停地查询一条不存在的数据，缓存没有，数据库也没有，那么会出现什么情况，如何处理？

|  |
| --- |
| **public** SkuInfo getSkuInfo(String skuId){  SkuInfo skuInfo = **null**;  **try** {  Jedis jedis = **redisUtil**.getJedis();  String skuInfoKey = ManageConst.***SKUKEY\_PREFIX*** + skuId + ManageConst.***SKUKEY\_SUFFIX***;  String skuInfoJson = jedis.get(skuInfoKey);   **if** (skuInfoJson == **null** || skuInfoJson.length() == 0) {  System.***err***.println(Thread.*currentThread*().getName()+**"缓存未命中！"**);  String skuLockKey = ManageConst.***SKUKEY\_PREFIX*** + skuId + ManageConst.***SKULOCK\_SUFFIX***;  String lock = jedis.set(skuLockKey, **"OK"**, **"NX"**, **"PX"**, ManageConst.***SKULOCK\_EXPIRE\_PX***);   **if** (**"OK"**.equals(lock) ){  System.***err***.println(Thread.*currentThread*().getName()+**"获得分布式锁！"**);  skuInfo = getSkuInfoFromDB(skuId);  **if**(skuInfo==**null**){  jedis.setex(skuInfoKey, ManageConst.***SKUKEY\_TIMEOUT***, **"empty"**);  **return null**;  }    String skuInfoJsonNew = JSON.*toJSONString*(skuInfo);  jedis.setex(skuInfoKey, ManageConst.***SKUKEY\_TIMEOUT***, skuInfoJsonNew);  jedis.close();  **return** skuInfo;  }**else**{  System.***err***.println(Thread.*currentThread*().getName()+**"未获得分布式锁，开始自旋！"**);  Thread.*sleep*(1000);  jedis.close();  **return** getSkuInfo( skuId);  }   } **else if**(skuInfoJson.equals(**"empty"**)){  **return null**;  } **else** {  System.***err***.println(Thread.*currentThread*().getName()+**"缓存已命中！！！！！！！！！！！！！！！！！！！"**);  skuInfo = JSON.*parseObject*(skuInfoJson, SkuInfo.**class**);  jedis.close();  **return** skuInfo;  }   }**catch** (JedisConnectionException e){  e.printStackTrace();  } **catch** (InterruptedException e) {  e.printStackTrace();  }   **return** getSkuInfoFromDB(skuId); } |

作业: 依照getSkuInfo的方法补全，getSpuSaleAttrListCheckBySku方法和getSkuSaleAttrValueListBySpu，实现详情页面全部信息都从缓存中查询。

skuInfo保存后，清除原缓存中数据。

# 八 缓存问题

## 1缓存穿透

* **缓存穿透**是指**查询一个一定不存在的数据**，由于缓存是不命中，将去查询数据库，但是数据库也无此记录，并且处于容错考虑，我们没有将这次查询的null写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。**在流量大时**，可能DB就挂掉了，要是有人利用不存在的key频繁攻击我们的应用，这就是漏洞。
* 解决

空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

## 2缓存雪崩

* 缓存雪崩是指在我们设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到DB，DB瞬时压力过重雪崩。
* 解决：

原有的失效时间基础上增加一个随机值，比如1-5分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

## 3缓存击穿

* 对于一些设置了过期时间的key，如果这些key可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：如果这个key在大量请求同时进来前正好失效，那么所有对这个key的数据查询都落到db，我们称为缓存击穿。和缓存雪崩的区别：
  + - 击穿是一个热点key失效



* + - 雪崩是很多key集体失效



* 缓存在某个时间点过期的时候，恰好在这个时间点对这个Key有大量的并发请求过来，这些请求发现缓存过期一般都会从后端DB加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把后端DB压垮。
* 解决：

分布式锁



## 4分布式锁



核心代码

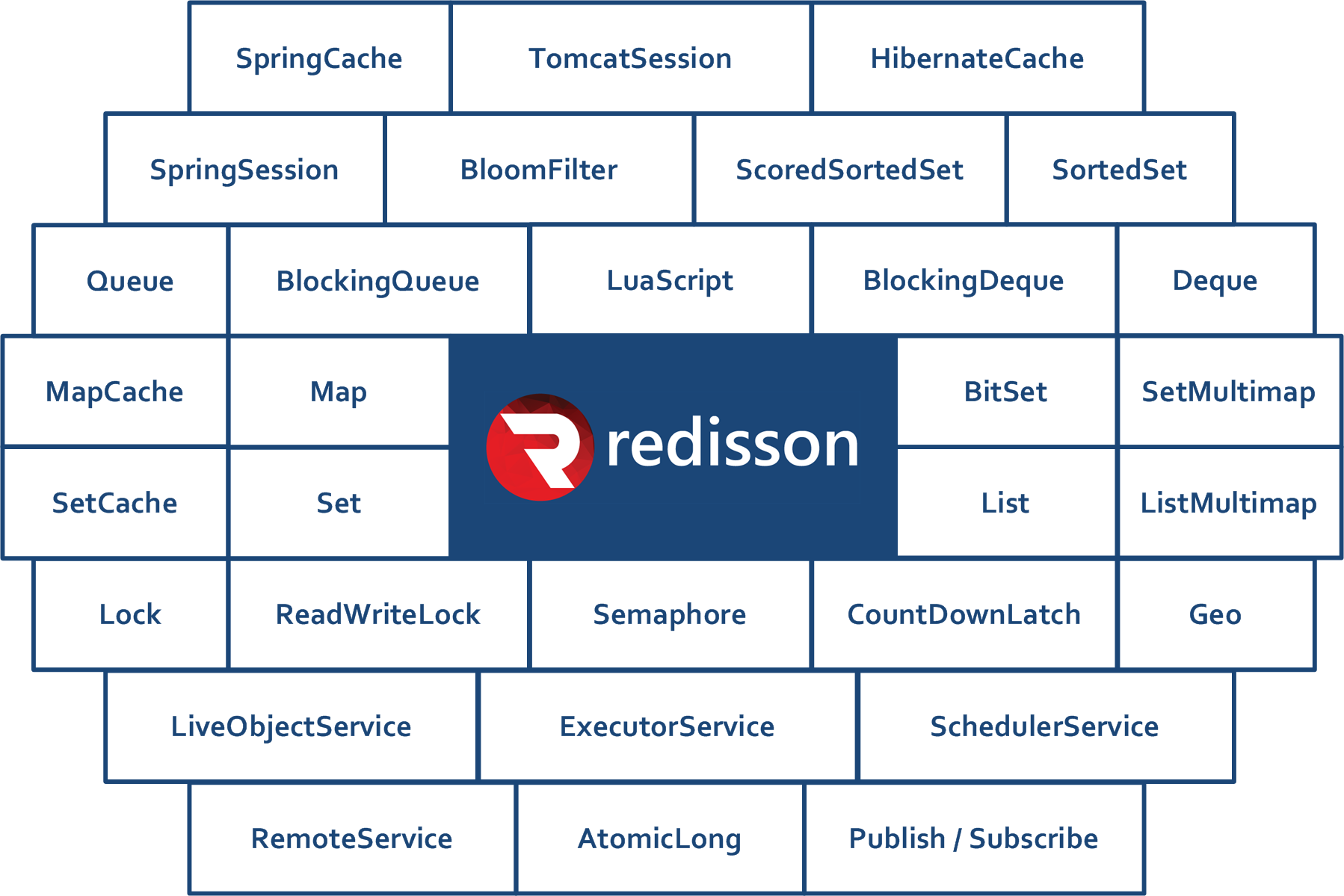
|  |
| --- |
| 加锁  String token = UUID.randomUUID().toString();  String lock = jedis.set(key, token, "NX", "EX",20); |

# 九 redisson

<https://github.com/redisson/redisson/wiki/1.-Overview>

## 1简介

Redisson是一个在Redis的基础上实现的Java驻内存数据网格（In-Memory Data Grid）。它不仅提供了一系列的分布式的Java常用对象，还提供了许多分布式服务。其中包括(BitSet, Set, Multimap, SortedSet, Map, List, Queue, BlockingQueue, Deque, BlockingDeque, Semaphore, Lock, AtomicLong, CountDownLatch, Publish / Subscribe, Bloom filter, Remote service, Spring cache, Executor service, Live Object service, Scheduler service) Redisson提供了使用Redis的最简单和最便捷的方法。Redisson的宗旨是促进使用者对Redis的关注分离（Separation of Concern），从而让使用者能够将精力更集中地放在处理业务逻辑上。



## 2整合

|  |
| --- |
| 引入pom  *<!-- https://mvnrepository.com/artifact/org.redisson/redisson -->* <**dependency**>  <**groupId**>org.redisson</**groupId**>  <**artifactId**>redisson</**artifactId**>  <**version**>3.10.5</**version**> </**dependency**> |
| 配置  **spring.redis.host**=**192.168.159.130 spring.redis.port**=**6379** |
| 配置类  @Configuration **public class** GmallRedissonConfig {   @Value(**"${spring.redis.host}"**)  **private** String **host**;   @Value(**"${spring.redis.port}"**)  **private** String **port**;   @Bean  **public** RedissonClient redissonClient(){  Config config = **new** Config();  config.useSingleServer().setAddress(**"redis://"**+**host**+**":"**+**port**);  RedissonClient redisson = Redisson.*create*(config);  **return** redisson;  } } |

## 3redisson锁

可重入锁（Reentrant Lock）

基于Redis的Redisson分布式可重入锁RLock Java对象实现了java.util.concurrent.locks.Lock接口。同时还提供了异步（Async）、反射式（Reactive）和RxJava2标准的接口。

|  |
| --- |
| RLock lock = redisson.getLock("anyLock");  // 最常见的使用方法  lock.lock(); |
| 另外Redisson还通过加锁的方法提供了leaseTime的参数来指定加锁的时间。超过这个时间后锁便自动解开了。  // 加锁以后10秒钟自动解锁  // 无需调用unlock方法手动解锁  lock.lock(10, TimeUnit.SECONDS);  // 尝试加锁，最多等待100秒，上锁以后10秒自动解锁  boolean res = lock.tryLock(100, 10, TimeUnit.SECONDS);  if (res) {  try {  ...  } finally {  lock.unlock();  }  } |
|  |

# 十 redisson分布式压力测试

## 安装

Linux

|  |
| --- |
| linux直接yum -y install httpd-tools，然后ab -V测试 |

Windows

|  |
| --- |
| 1查看80端口有没有被占用，netstat -ano | findstr "80"  2下载地址<https://www.apachehaus.com/cgi-bin/download.plx>    3解压后，找到安装目录下的httpd.conf，修改为自己的安装目录    4 启动服务    5 命令示例(并发200，一共1000个请求)  D:\apache24\bin>ab -c 200 -n 1000 http:nginx负载均衡/压力方法  6 测试结果  研究redisson的分布式锁在并发下的效果 |

## Ab命令参数

|  |
| --- |
| **-n即requests，用于指定压力测试总共的执行次数。**  **-c即concurrency，用于指定压力测试的并发数。**  -t即timelimit，等待响应的最大时间(单位：秒)。  -b即windowsize，TCP发送/接收的缓冲大小(单位：字节)。  -p即postfile，发送POST请求时需要上传的文件，此外还必须设置-T参数。  -u即putfile，发送PUT请求时需要上传的文件，此外还必须设置-T参数。  -T即content-type，用于设置Content-Type请求头信息，例如：application/x-www-form-urlencoded，默认值为text/plain。  -v即verbosity，指定打印帮助信息的冗余级别。  -w以HTML表格形式打印结果。  -i使用HEAD请求代替GET请求。  -x插入字符串作为table标签的属性。  -y插入字符串作为tr标签的属性。  -z插入字符串作为td标签的属性。  -C添加cookie信息，例如："Apache=1234"(可以重复该参数选项以添加多个)。  -H添加任意的请求头，例如："Accept-Encoding: gzip"，请求头将会添加在现有的多个请求头之后(可以重复该参数选项以添加多个)。  -A添加一个基本的网络认证信息，用户名和密码之间用英文冒号隔开。  -P添加一个基本的代理认证信息，用户名和密码之间用英文冒号隔开。  -X指定使用的代理服务器和端口号，例如:"126.10.10.3:88"。  -V打印版本号并退出。  -k使用HTTP的KeepAlive特性。  -d不显示百分比。  -S不显示预估和警告信息。  -g输出结果信息到gnuplot格式的文件中。  -e输出结果信息到CSV格式的文件中。  -r指定接收到错误信息时不退出程序。  -h显示用法信息，其实就是ab -help。 |

## 测试

|  |  |
| --- | --- |
| 测试代码  1 新建一个测试redisson控制层，加入测试代码   |  | | --- | | @RequestMapping(**"lockTest"**) @ResponseBody **public** String lockTest(){  Jedis jedis = **redisUtil**.getJedis();// redis链接  RLock lock = **redissonClient**.getLock(**"redis-lock"**);//分布锁  *//加锁* lock.lock();**try** {  String v = jedis.get(**"k"**);//获取value  System.***err***.print(**"==>"**+v);//打印value  **if**(StringUtil.*isBlank*(v)){  v = **"1"**;  }  **int** inum = Integer.*parseInt*(v);//获得value的值  jedis.set(**"k"**, inum+1+**""**);//value增加1  jedis.close();  } **finally** {  lock.unlock();  }  **return "success"**; } |   2 启动n个测试redisson控制层的微服务    3 配置nginx的负载均衡到测试redisson控制层的微服务  A配置nginx的upstream  upstream redisTest {  server 127.0.0.1:8071 weight=3;  server 127.0.0.1:8072 weight=3;  server 127.0.0.1:8073 weight=3;  }  B配置nginx的默认代理地址  location / {  #root D:/software/upload;  proxy\_pass http://redisTest;  index index.html index.htm;  }  4 用apache的ab压力测试命令访问nginx来压测分布式的redisson控制层的微服务  同时观察redis的被并发的k的增长情况 |

# 十一 nginx配置

应用级锁与分布式锁到底能解决什么问题。测试就知道了

|  |
| --- |
| nginx配置负载均衡  upstream tomcat {  server 192.168.159.1:8080 weight=20;  server 192.168.159.1:8081 weight=20;  server 192.168.159.1:8082 weight=20;  server 192.168.159.1:8083 weight=20;  }  location / {  proxy\_pass http://tomcat  } |
| C:\Users\lfy\AppData\Roaming\feiq\RichOle\4058099743.bmp |