Item商品详情

# 1 商品详情的介绍

|  |
| --- |
| 1 电商的前台系统和后台系统区别  前台页面更丰富、前台消费者用户、前台系统压力更大、前台系统多用缓存负载均衡，数据的一致性  后台页面更专业、后台管理人员或商家、后台系统压力更小、后台系统直连数据库，数据的一致性  2电商的前台系统和后台系统联系  数据结构是同一套，后台系统的一些设置可以控制前台系统的运行方式 |

# 2 前台系统中包含的业务功能

|  |
| --- |
| 1 首页（静态化）  2 检索页（搜索引擎）  3 详情页（缓存、切换、推荐）  4 购物车页（cookie、redis）  5 结算页(订单页)（一致性校验、安全）  6 支付页（安全、对接支付平台） |

# 3 item工程的创建

Manage

SkuService、spuService

|  |  |  |
| --- | --- | --- |
| 1 创建一个item-web的商品详情工程  2 配置item-web，pom，properites  3 引入thymeleaf的依赖，thymeleaf是springboot的原生渲染模板，它使用的是html的扩展标签，需要在页面上声明thymeleaf的模板约束  <**html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"**>  4 为thymeleaf加入松校验、热部署   |  | | --- | | *<!-- thymeleaf支持包、可以设置松校验、热部署 -->* <**dependency**>  <**groupId**>net.sourceforge.nekohtml</**groupId**>  <**artifactId**>nekohtml</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>xml-apis</**groupId**>  <**artifactId**>xml-apis</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>org.apache.xmlgraphics</**groupId**>  <**artifactId**>batik-ext</**artifactId**> </**dependency**> |   热部署配置   |  | | --- | | *# 关闭thymeleaf的缓存(热部署)* **spring.thymeleaf.cache**=**false** *# 松校验* **spring.thymeleaf.mode**=**LEGACYHTML5** |   5 在th模板中使用js并且在js中使用th的标签表达式传参    6 将静态页面引入项目 |

# 4 sku根据销售属性的动态切换

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 数据模型(根据销售属性切换一个sku的其他兄弟姐妹)  Pms\_sku\_info pms\_sku\_image  Pms\_sku\_sale\_attr\_value  Pms\_spu\_sale\_attr  Pms\_spu\_sale\_attr\_value  2 如何实现这些数据模型对应的业务功能  A页面得到销售属性列表(当前sku对应spu的id)  Pms\_spu\_sale\_attr  Pms\_spu\_sale\_attr\_value   |  | | --- | | SELECT  \*  FROM  pms\_product\_sale\_attr sa,  pms\_product\_sale\_attr\_value sav  WHERE  sa.product\_id = sav.product\_id  AND sa.sale\_attr\_id = sav.sale\_attr\_id  and sa.product\_id = ? |  |  | | --- | | SELECT  sa.\*, sav.\*, if(ssav.sku\_id,1,0) as isChecked  FROM  pms\_product\_sale\_attr sa  INNER JOIN pms\_product\_sale\_attr\_value sav ON sa.product\_id = sav.product\_id  AND sa.sale\_attr\_id = sav.sale\_attr\_id  AND sa.product\_id = 66  LEFT JOIN pms\_sku\_sale\_attr\_value ssav ON sav.id = ssav.sale\_attr\_value\_id  AND ssav.sku\_id = 108 |   B页面根据销售属性的选择的组合，定位到关联的sku  通过页面被选择属性值id  查询中间表pms\_sku\_sale\_attr\_value  得到skuId   |  | | --- | | SELECT  sku\_id  FROM  pms\_sku\_sale\_attr\_value ssav  WHERE  ssav.sale\_attr\_id IN (销售属性id1,销售属性id2)  AND ssav.sale\_attr\_value\_id IN (销售属性值id1,销售属性值id2) |   C根据skuId查询到sku对象返回到页面  D功能优化  在用户进入某一个spu领域后，将该spu所包含的sku们和这些sku对应的销售属性值  生成一个k是销售属性值组合，v是skuId的hash表格，放到页面上   |  |  | | --- | --- | | k:239|243 | v:106 | | k:239|244 | v:107 | | k:240|245 | v:108 |   在用户选择完销售属性之后，根据hash的组合找到对应的skuid |

复习

|  |
| --- |
| 商品详情页面的数据结构  Pms\_sku\_info  pms\_sku\_image  销售属性集合  pms\_spu\_sale\_attr  pms\_spu\_sale\_attr\_value  pms\_sku\_sale\_attr\_value |

销售属性值

销售属性

Sku销售属性关联

Sku信息表

|  |
| --- |
| SELECT  sa.\*, sav.\*,  IF (ssav.sku\_id, 1, 0) AS isChecked  FROM  pms\_product\_sale\_attr sa  INNER JOIN pms\_product\_sale\_attr\_value sav ON sa.product\_id = sav.product\_id  AND sa.sale\_attr\_id = sav.sale\_attr\_id  AND sa.product\_id = 66  LEFT JOIN pms\_sku\_sale\_attr\_value ssav ON sa.sale\_attr\_id = ssav.sale\_attr\_id  AND sav.id = ssav.sale\_attr\_value\_id  AND ssav.sku\_id = 108 |

|  |
| --- |
| **(${saleAttrValue.isChecked}=='1'?'redborder':'')** 字符串：redborder/’’ |

# 5 将后台查出的同一个spu的sku和sku销售属性的列表组合成一个静态的hash表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 sql查询sku和关联的销售属性  SELECT  si.id , ssav.sale\_attr\_value\_id  FROM  pms\_sku\_info si,  pms\_sku\_sale\_attr\_value ssav  WHERE  si.product\_id = 66  AND si.id = ssav.sku\_id  skuInfo集合  sku\_sale\_attr\_value集合  将集合转化为   |  |  | | --- | --- | | k:239|243 | v:106 | | k:239|244 | v:107 | | k:240|245 | v:108 |   spuId\_66.js |

Cpu的寄存器

一级缓存、二级缓存

内存

机械磁盘

# 6 使用缓存redis解决页面并发问题

Redis

Mysql

|  |
| --- |
| 1 缓存使用的简单设计  *// 链接缓存 // 查询缓存 // 如果缓存中没有，查询mysql // mysql查询结果存入redis*  2 reids的整合步骤  A将redis整合到项目中(redis+spring)  Spring-conf  Spring  B设计一个数据存储策略  企业中的存储策略（核心就是，设计key）  数据对象名：数据对象id：对象属性  User:123:passpword  User:123:username  Sku:108:info |

# 7 reids的整合过程

|  |
| --- |
| 1 引入pom依赖信息(本工程所有redis统一放入service-util)  <**dependency**>  <**groupId**>redis.clients</**groupId**>  <**artifactId**>jedis</**artifactId**> </**dependency**>  2 写一个reids的工具类(用来将redis的池初始化到spring容器中)  **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;  } }  3 写一个spring整合redis的配置类  将redis的链接池创建到spring的容器中  @Configuration public class RedisConfig {  *//读取配置文件中的redis的ip地址* @Value("${spring.redis.host:disabled}")  private String host = “192.168.222.20”;  @Value("${spring.redis.port:0}")  private int port = “6179”;  @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;  } }  4 注意  每个以用工程引入service-util后，单独配置自己的redis的配置文件  Service-util的配置文件没有作用 |

Service1

Service1

Service2

Service2

# 8 redis常见问题

|  |
| --- |
| 1 cont get a connection from the pool  2缓存在高并发和安全压力下的一些问题  缓存击穿  是某一个热点key在高并发访问的情况下，突然失效，导致大量的并发打进mysql数据库的情况  缓存穿透  是利用redis和mysql的机制(redis缓存一旦不存在，就访问mysql)，直接绕过缓存访问mysql，而制造的db请求压力  一般在代码中防止该现象的发生  解决：*// 为了防止缓存穿透将，null或者空字符串值设置给redis*  缓存雪崩  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，导致的db崩溃  解决：设置不同的缓存失效时间  3 如何解决击穿缓存的问题 |

# 9 如何解决缓存击穿的问题

|  |
| --- |
| Redis失效  Mysql  Redis的nx |

|  |
| --- |
| 穿透：利用不存在的key去攻击mysql数据库  雪崩：缓存中的很多key失效，导致数据库负载过重宕机  击穿：在正常的访问情况下，如果缓存失效，如果保护mysql，重启缓存的过程  使用redis数据库的分布式锁，解决mysql的访问压力问题  第一种分布式锁：redis自带一个分布式锁,set px nx    第二种分布式锁：redisson框架，一个redis的带有juc的lock功能的客户端的实现(既有jedis的功能，又有juc的锁功能) |

Redisson

Redis

自带的分布式锁

Jedis

Redis

Jedis

错误自旋代码



Redis

Redis

分布锁

Mysql

# 10 lua脚本

|  |
| --- |
| 问题1 如果在redis中的锁已经过期了，然后锁过期的那个请求又执行完毕，回来删锁,删除了其他线程的锁，怎么办？  问题2 如果碰巧在查询redis锁还没删除的时候，正在网络传输时，锁过期了  怎么办？  String script ="if redis.call('get', KEYS[1]) == ARGV[1] then return redis.call('del', KEYS[1]) else return 0 end";  jedis.eval(script, Collections.singletonList("lock"),Collections.singletonList(token)); |

# 11 redis的分布式工具框架redisson

|  |
| --- |
| Redisson实现了juc的lock锁，并且可以在分布式的redis环境下使用 |

Apache

Nginx

商品的redis缓存服务

Redisson

Redis

商品的redis缓存服务

商品的redis缓存服务

|  |
| --- |
| 设置非单例模式启动 |

|  |
| --- |
| 配置nginx |

# 12 下载安装apache测试工具(apache)

|  |  |
| --- | --- |
| 1 下载地址  <https://www.apachehaus.com/cgi-bin/download.plx>    2 安装即解压    3 修改apache服务的配置文件(服务器的根目录)    修改服务的根目录路径：    4 启动服务    5 压测命令    6 压力测试代码   |  | | --- | | **public** String testRedisson(){  Jedis jedis = **redisUtil**.getJedis();  RLock lock = **redissonClient**.getLock(**"lock"**);*// 声明锁* lock.lock();*//上锁* **try** {  String v = jedis.get(**"k"**);  **if** (StringUtils.*isBlank*(v)) {  v = **"1"**;  }  System.***out***.println(**"->"** + v);  jedis.set(**"k"**, (Integer.*parseInt*(v) + 1) + **""**);  }**finally** {  jedis.close();  lock.unlock();*// 解锁* }  **return "success"**; } |   7 测试结果 |