**品优购电商系统开发**

**第9章**

**广告展示与缓存解决方案**

传智播客.黑马程序员

# 课程目标

目标1：完成网站首页-广告展示功能

目标2：运用SpringDataRedis框架操作缓存

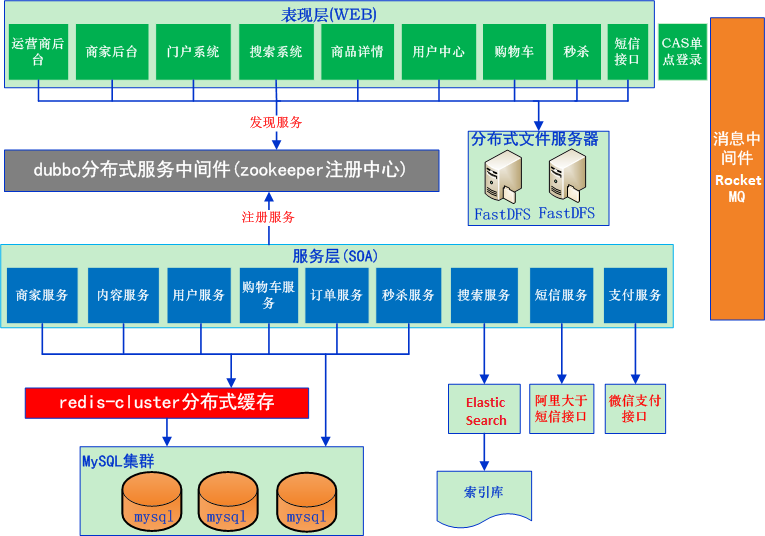
目标3：完成网站首页-缓存广告数据的功能

# 1.网站前台分析

## 1.1网站前台有哪些页面

* 网站首页
* 商家（店铺）首页
* 商品详细页
* 商品搜索页
* 购物车列表页
* 支付页
* 用户注册页
* 用户登录页
* 用户中心页等

# 搭建门户工程



## 2.1 pinyougou-portal-web(war)

与pinyougou-manager-web构建方式一样，拷贝pinyougou-manager-web模块中的web.xml(**去掉shiro配置**)、springmvc.xml、log4j.properties

pinyougou-portal-web/pom.xml：

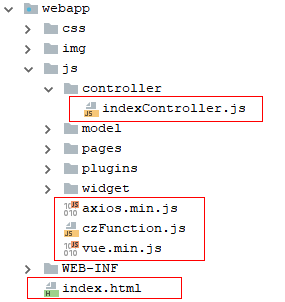
|  |
| --- |
| <**properties**>  *<!-- 更改tomcat端口号 -->* <**port**>**9103**</**port**> </**properties**> |

pinyougou-portal-web/src/main/resources/springmvc.xml：

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd  http://code.alibabatech.com/schema/dubbo  http://code.alibabatech.com/schema/dubbo/dubbo.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd"**>   ***<!-- 配置开启组件扫描 -->* <context:component-scan base-package="com.pinyougou.portal.controller"/>**  *<!-- 开启MVC注解驱动 -->* <**mvc:annotation-driven**>  *<!-- 配置消息转换 -->* <**mvc:message-converters**>  *<!-- 配置用fastjson作为json处理框架 -->* <**bean class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter"**>  *<!-- 配置支持的类型 -->* <**property name="supportedMediaTypes"  value="application/json;charset=UTF-8"**/>  </**bean**>  </**mvc:message-converters**>  </**mvc:annotation-driven**>  *<!-- 配置静态资源用WEB容器默认的servlet来处理 -->* <**mvc:default-servlet-handler**/>   *<!-- ############## 配置dubbo服务消费者 ############## -->  <!-- 配置当前应用的名称 -->* <**dubbo:application name="pinyougou-portal-web"**/>  *<!-- 配置zookeeper作为注册中心，注册服务地址 -->* <**dubbo:registry protocol="zookeeper" address="192.168.12.131:2181"**/>  *<!-- 配置采用包扫描来引用服务，产生服务接口的代理对象 -->* <**dubbo:annotation package="com.pinyougou.portal.controller"**/> </**beans**> |

## 2.2静态资源

拷贝“资料\网站首页静态资源”拷贝到pinyougou-portal-web模块的src/main/webapp目录下。



## 2.3配置域名访问

配置hosts文件:

127.0.0.1 [www.pinyougou.com](http://www.pinyougou.com)

127.0.0.1 pinyougou.com

配置nginx.conf反向代理:

|  |
| --- |
| server {  listen 80;  server\_name www.pinyougou.com pinyougou.com;  proxy\_set\_header X-Forwarded-Host $host;  proxy\_set\_header X-Forwarded-Server $host;  location / {  proxy\_pass http://127.0.0.1:9103;  proxy\_connect\_timeout 600;  proxy\_read\_timeout 600;  }  } |

重启Nginx: nginx -s reload

重启后，地址栏输入 [http://www.pinyougou.com](http://localhost:9103/index.html) 或 [http://pinyougou.com](http://localhost:9103/index.html) 即可看到首页效果：



# 3.网站首页【广告展示】

## 3.1需求分析

修改网站首页，轮播广告图根据后台设置的广告列表动态产生。

## 3.2前端代码

pinyougou-portal-web/src/main/webapp/index.html引入js

|  |
| --- |
| <**script src="/js/vue.min.js"**></**script**> <**script src="/js/axios.min.js"**></**script**> <**script src="/js/controller/indexController.js"**></**script**> |

pinyougou-portal-web/src/main/webapp/js/controller/indexController.js

|  |
| --- |
| *// 窗口加载完* ***window***.onload = **function** () {  **var** vue = **new Vue**({  **el** : **'#app'**, *// 元素绑定* **data** : { *// 数据模型* **contentList : [] *// 大广告数组*** },  **methods** : { *// 操作方法  // 根据广告分类id查询广告内容* **findContentByCategoryId : function (categoryId) {  axios.get("/content/findContentByCategoryId?categoryId="  + categoryId).then(  function(response){  vue.contentList = response.data;  }  );  }**  },  created : **function** () { *// 创建生命周期* **this.findContentByCategoryId(1);**  }  }); }; |

修改index.html首页轮播图524行

|  |
| --- |
| *<!--banner轮播-->* <**div id="myCarousel" data-ride="carousel" data-interval="4000"  class="sui-carousel slide"**>  <**ol class="carousel-indicators"**>  *<!-- 下面圆点控制 -->* <**li data-target="#myCarousel"  v-for="(c,index) in contentList"  :data-slide-to="index"  :class="index == 0 ? 'active' : ''"**></**li**>  </**ol**>  <**div class="carousel-inner"**>  <**div v-for="(c,index) in contentList"  :class="index == 0 ? 'active item' : 'item'"**>  <**a :href="c.url"**>  <**img :src="c.pic"**/>  </**a**>  </**div**>  </**div**>  <**a href="#myCarousel" data-slide="prev"  class="carousel-control left"**>‹</**a**>  <**a href="#myCarousel" data-slide="next"  class="carousel-control right"**>›</**a**> </**div**> |

## 3.3后端代码

pinyougou-portal-web/src/main/java/com.pinyougou.portal.controller包下创建ContentController.java

|  |
| --- |
| @RestController @RequestMapping(**"/content"**) **public class** ContentController {  @Reference(timeout=10000)  **private** ContentService **contentService**;  */\*\* 根据广告分类ID查询广告数据 \*/* @GetMapping(**"/findContentByCategoryId"**)  **public** List<Content> findContentByCategoryId(Long categoryId) {  **return contentService**.findContentByCategoryId(categoryId);  } } |

pinyougou-interface/src/main/java/com.pinyougou.service/ContentService.java增加方法

|  |
| --- |
| */\*\* 根据广告类型ID查询数据 \*/* List<Content> findContentByCategoryId(Long categoryId); |

pinyougou-content-service/src/main/java/com.pinyougou.content.service.impl的ContentServiceImpl.java增加方法

|  |
| --- |
| */\*\* 根据分类id查询广告内容 \*/* **public** List<Content> findContentByCategoryId(Long categoryId){  **try**{  *// 创建示范对象* Example example = **new** Example(Content.**class**);  *// 创建查询条件对象* Example.Criteria criteria = example.createCriteria();  *// 添加等于条件 category\_id = categoryId* criteria.andEqualTo(**"categoryId"**, categoryId);  *// 添加等于条件 status = 1* criteria.andEqualTo(**"status"**, **"1"**);  *// 排序(升序) order by sort\_order asc* example.orderBy(**"sortOrder"**).asc();  *// 条件查询*  **return contentMapper**.selectByExample(example);  }**catch** (Exception ex){  **throw new** RuntimeException(ex);  } } |

# 4. SpringDataRedis【介绍及使用】

## 4.1项目常见问题思考

我们目前的系统已经实现了广告后台管理和广告前台展示，但是对于首页每天有大量的人访问，对数据库造成很大的访问压力，甚至是瘫痪。那如何解决呢？我们通常的做法有两种：一种是数据缓存、一种是网页静态化。我们今天讨论第一种解决方案。

## 4.2 Redis

**[redis](http://lib.csdn.net/base/redis" \o "Redis知识库" \t "http://blog.csdn.net/zhang_red/article/details/_blank)**是一款开源的高性能的Key-Value**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/zhang_red/article/details/_blank)**，运行在内存中，由ANSI C编写。企业开发通常采用Redis来实现缓存。同类的产品还有memcached 、MongoDB等。

### 4.2.1 Linux安装Redis单机版【运维】

参考安装文档：资料\redis\【**Linux安装Redis单机版.docx**】

### 4.2.2 Linux安装Redis集群版【运维】

参考安装文档：资料\redis\【**Linux安装Redis集群版.docx**】

## 4.3 Jedis

Jedis是Redis官方推出的一款面向**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/zhang_red/article/details/_blank)**的客户端，提供了很多接口供Java语言调用。可以在Redis官网下载，当然还有一些开源爱好者提供的客户端，如Jredis、Redisson等等，推荐使用Jedis。

## 4.4 Spring Data Redis介绍

Spring-Data-Redis是Spring大家族的一部分，提供了在Spring应用中通过简单的配置访问Redis服务，对Redis底层开发包(Jedis、JRedis、Redisson)进行了高度封装，RedisTemplate提供了Redis各种操作、异常处理及序列化，支持发布订阅。

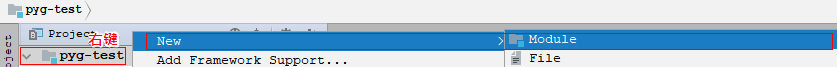
Spring-Data-Redis针对jedis提供了如下功能：  
1. 连接池自动管理，提供了一个高度封装的“RedisTemplate”类  
2. 针对jedis客户端中大量api进行了归类封装,将同一类型操作封装为operation接口如下：

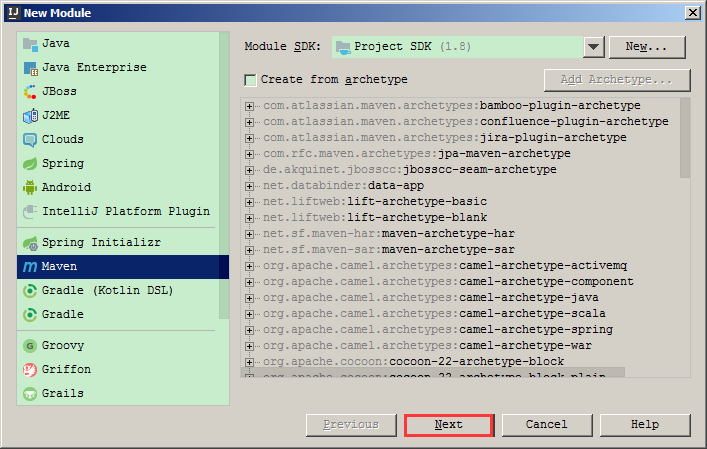
* ValueOperations：简单K-V操作
* SetOperations： set类型数据操作
* ZSetOperations： zset类型数据操作
* HashOperations： hash类型的数据操作
* ListOperations： list类型的数据操作

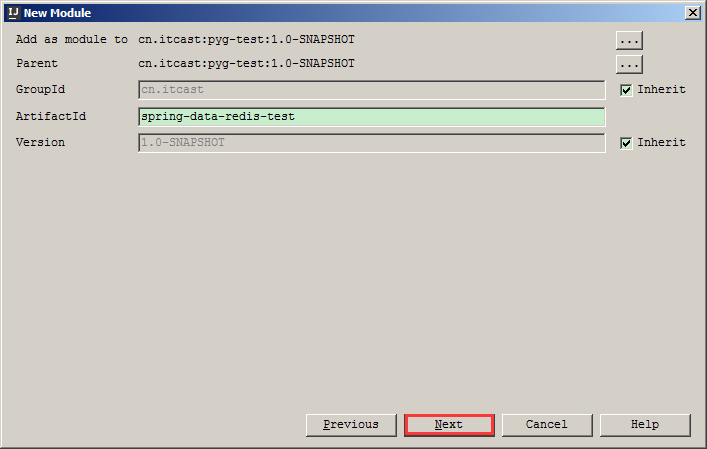
## 4.5 Spring Data Redis使用

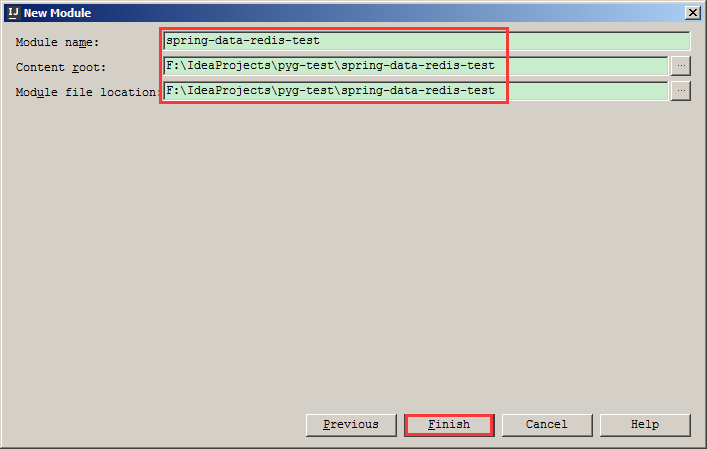
### 4.5.1搭建测试工程

构建Maven模块spring-data-redis-test(jar)









spring-data-redis-test/pom.xml，引入依赖

|  |
| --- |
| <**dependencies**>  *<!-- jedis -->* <**dependency**>  <**groupId**>redis.clients</**groupId**>  <**artifactId**>jedis</**artifactId**>  <**version**>2.9.0</**version**>  </**dependency**>  *<!-- spring-data-redis -->* <**dependency**>  <**groupId**>org.springframework.data</**groupId**>  <**artifactId**>spring-data-redis</**artifactId**>  <**version**>1.8.6.RELEASE</**version**>  </**dependency**>  *<!-- spring-context -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-context</**artifactId**>  <**version**>5.1.3.RELEASE</**version**>  </**dependency**>  *<!-- spring-test -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-test</**artifactId**>  <**version**>5.1.3.RELEASE</**version**>  <**scope**>test</**scope**>  </**dependency**> </**dependencies**> |

### 4.5.2 操作Redis单机版

#### **4.5.2.1配置部分**

在spring-data-redis-test/src/main/resources/目录下创建

applicationContext-redis.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd"**>  *<!-- ################ 配置Redis单机版 #################-->  <!-- 加载属性文件 -->* <**context:property-placeholder  location="classpath:redis-config.properties"**/>   *<!-- 配置连接工厂 -->* <**bean id="connectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"**>  <**property name="hostName" value="${redis.host}"**/>  <**property name="port" value="${redis.port}"**/>  </**bean**>   *<!-- 配置RedisTemplate -->* <**bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate"**>  <**property name="connectionFactory" ref="connectionFactory"**/>  </**bean**> </**beans**> |

spring-data-redis-test/src/main/resources/**redis-config.properties**

|  |
| --- |
| **redis.host**=**192.168.12.131 redis.port**=**6379** |

#### **4.5.2.2代码部分**

spring-data-redis-test/src/main/test/cn/itcast/redis/包下创建测试类Redis01Test.java

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**) @ContextConfiguration(locations = {**"classpath:applicationContext-redis.xml"**}) **public class** Redis01Test {  @Autowired  **private** RedisTemplate<Object,Object> **redisTemplate**;   */\*\* String类型 \*/* @Test  **public void** stringTest(){  */\*\* 设置值 \*/* **redisTemplate**.boundValueOps(**"name"**).set(**"itcast"**);  */\*\* 获取值 \*/* String str = (String) **redisTemplate**.boundValueOps(**"name"**).get();  System.***out***.println(str);  */\*\* 删除值 \*/* **redisTemplate**.delete(**"name"**);  }  */\*\* Set类型 \*/* @Test  **public void** setTest(){  */\*\* 设置值 \*/* **redisTemplate**.boundSetOps(**"name"**).add(**"李大华"**,**"李中华"**);  **redisTemplate**.boundSetOps(**"name"**).add(**"李小华"**);  */\*\* 获取值 \*/* Set members = **redisTemplate**.boundSetOps(**"name"**).members();  System.***out***.println(members);  */\*\* 删除集合中的某一个值 \*/* **redisTemplate**.boundSetOps(**"name"**).remove(**"李中华"**);  */\*\* 删除整个集合 \*/* **redisTemplate**.delete(**"name"**);  }  */\*\* List类型 \*/* @Test  **public void** listTest(){  */\*\* 右压栈：后添加的元素排在后面 \*/* **redisTemplate**.boundListOps(**"name1"**).rightPush(**"李大华"**);  **redisTemplate**.boundListOps(**"name1"**).rightPush(**"李中华"**);  **redisTemplate**.boundListOps(**"name1"**).rightPush(**"李小华"**);  */\*\* 获取值(全部) \*/* List list1 = **redisTemplate**.boundListOps(**"name1"**).range(0, -1);  System.***out***.println(list1);   */\*\* 左压栈：后添加的元素排在前面 \*/* **redisTemplate**.boundListOps(**"name2"**).leftPush(**"李大华"**);  **redisTemplate**.boundListOps(**"name2"**).leftPush(**"李中华"**);  **redisTemplate**.boundListOps(**"name2"**).leftPush(**"李小华"**);  */\*\* 获取值(全部) \*/* List list2 = **redisTemplate**.boundListOps(**"name2"**).range(0, -1);  System.***out***.println(list2);   */\*\* 获取集合中某个元素 \*/* String s = (String) **redisTemplate**.boundListOps(**"name1"**).index(1);  System.***out***.println(s);   */\*\* 删除集合中某个元素 第一个参数零:代表把匹配到的’李中华’全部删除\*/* **redisTemplate**.boundListOps(**"name1"**).remove(0, **"李中华"**);  }  */\*\* Hash类型 \*/* @Test  **public void** hashTest(){  */\*\* 设置值 \*/* **redisTemplate**.boundHashOps(**"name"**).put(**"a"**, **"李大华"**);  **redisTemplate**.boundHashOps(**"name"**).put(**"b"**, **"李中华"**);  **redisTemplate**.boundHashOps(**"name"**).put(**"c"**, **"李小华"**);   */\*\* 获取所有的key \*/* Set keys = **redisTemplate**.boundHashOps(**"name"**).keys();  System.***out***.println(keys);   */\*\* 获取所有的value \*/* List values = **redisTemplate**.boundHashOps(**"name"**).values();  System.***out***.println(values);   */\*\* 获取指定key的value \*/* Object obj = **redisTemplate**.boundHashOps(**"name"**).get(**"b"**);  System.***out***.println(obj);   */\*\* 删除指定的key与value \*/* **redisTemplate**.boundHashOps(**"name"**).delete(**"c"**);  } } |

### 4.5.3 操作Redis集群版

#### **4.5.3.1配置部分**

在spring-data-redis-test/src/main/resources目录下创建

applicationContext-redis-cluster.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd"**>   *<!-- ################ 配置Redis集群版 #################-->  <!-- 配置资源属性文件源 -->* <**bean id="propertySource" class="org.springframework.core.io.support.ResourcePropertySource"**>  <**constructor-arg value="redis-cluster.properties"**/>  </**bean**>  *<!-- 配置Redis集群节点信息 -->* <**bean id="clusterConfig" class="org.springframework.data.redis.connection.RedisClusterConfiguration"**>  <**constructor-arg ref="propertySource"**/>  </**bean**>  *<!-- 配置Jedis连接工厂 -->* <**bean id="connectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"**>  <**constructor-arg ref="clusterConfig"**/>  </**bean**>  *<!-- 配置RedisTemplate -->* <**bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate"**>  <**property name="connectionFactory" ref="connectionFactory"**/>  </**bean**> </**beans**> |

spring-data-redis-test/src/main/resources/**redis-cluster.properties**

|  |
| --- |
| *# 配置集群中的节点* **spring.redis.cluster.nodes**=**192.168.12.131:7001,192.168.12.131:7002,192.168.12.131:7003,192.168.12.131:7004,192.168.12.131:7005,192.168.12.131:7006** |

#### **4.5.3.2代码部分**

spring-data-redis-test/src/main/test/cn/itcast/redis/包下创建测试类Redis02Test.java

集群版型测试类代码与与单机版测试代码一样，变化的是加载的Spring配置文件不一样。

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**) @ContextConfiguration(locations = {**"classpath:applicationContext-redis-cluster.xml"**}) **public class** Redis02Test {  @Autowired  **private** RedisTemplate<Object,Object> **redisTemplate**;  ......  } |

# 5.项目集成SpringDataRedis

因为缓存对于我们整个的系统来说是通用功能。广告需要用，其它数据可能也会用到，所以我们将配置放在公共组件层（**pinyougou-common**）中较为合理。

## 5.1配置依赖

pinyougou-common/pom.xml 引入依赖：

|  |
| --- |
| <**dependencies**>  *<!-- jedis -->* <**dependency**>  <**groupId**>redis.clients</**groupId**>  <**artifactId**>jedis</**artifactId**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework.data</**groupId**>  <**artifactId**>spring-data-redis</**artifactId**>  </**dependency**> </**dependencies**> |

## 5.2属性文件

单机版属性文件：pinyougou-common/src/main/resources/props/

redis-config.properties

|  |
| --- |
| **redis.host**=**192.168.12.131 redis.port**=**6379** |

集群版属性文件：pinyougou-common/src/main/resources/props/

redis-cluster.properties

|  |
| --- |
| *# 配置Redis集群节点信息* **spring.redis.cluster.nodes**=**192.168.12.131:7001,192.168.12.131:7002,192.168.12.131:7003,192.168.12.131:7004,192.168.12.131:7005,192.168.12.131:7006** |

## 5.3配置文件

Spring-Data-Redis整合Redis配置文件：

pinyougou-common/src/main/resources/applicationContext-redis.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd"**>   *<!-- ################ 配置Redis单机版 #################-->  <!-- 配置Jedis连接工厂 -->* <**bean id="connectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"**>  <**property name="hostName" value="${redis.host}"**/>  <**property name="port" value="${redis.port}"**/>  </**bean**>   *<!-- ################ 配置Redis集群版 #################-->  <!-- 配置次源属性源对象 -->* <**bean id="propertySource" class="org.springframework.core.io.support.ResourcePropertySource"**>  <**constructor-arg value="classpath:props/redis-cluster.properties"**/>  </**bean**>  *<!-- 配置Redis集群节点信息 -->* <**bean id="clusterConfig" class="org.springframework.data.redis.connection.RedisClusterConfiguration"**>  <**constructor-arg ref="propertySource"**/>  </**bean**>  *<!-- 配置Jedis连接工厂 -->  <!--<bean id="connectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">  <constructor-arg ref="clusterConfig"/>  </bean>-->  <!-- 配置RedisTemplate -->* <**bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate"**>  <**property name="connectionFactory" ref="connectionFactory"**/>  </**bean**> </**beans**> |

# 6.网站首页【广告缓存】

## 6.1需求分析

现在我们首页的广告每次都是从数据库读取，这样当网站访问量达到高峰时段，对数据库压力很大，并且影响执行效率。我们需要将这部分广告数据缓存起来。

## 6.2添加缓存

### 6.2.1配置部分

pinyougou-content-service/src/main/resources/applicationContext-service.xml导入applicationContext-redis.xml文件

|  |
| --- |
| *<!-- 加载属性文件 -->* <**context:property-placeholder  location="classpath\*:props/\*.properties"**/> *<!-- 导入数据访问的Spring配置文件 -->* <**import resource="classpath:applicationContext-mapper.xml"**/> *<!-- 导入Redis的Spring配置文件 -->* <**import resource="classpath:applicationContext-redis.xml"**/> |

**classpath\*：代表加载多个jar包中的属性文件。**

**classpath：代表加载一个jar包中的属性文件。**

### 6.2.2代码部分

pinyougou-content-service/src/main/java/com.pinyougou.content.service.impl的ContentServiceImpl的findContentByCategoryId()方法

|  |
| --- |
| @Autowired **private** RedisTemplate **redisTemplate**;  */\*\* 根据分类id查询广告内容 \*/* **public** List<Content> findContentByCategoryId(Long categoryId){  */\*\* 定义广告数据 \*/* List<Content> contentList = **null**;  **try** {  */\*\* 从Redis中获取广告 \*/* contentList = (List<Content>) **redisTemplate** .boundValueOps(**"content"**).get();  **if** (contentList != **null** && contentList.size() > 0) {  **return** contentList;  }  }**catch** (Exception ex){}  **try**{  *// 创建示范对象* Example example = **new** Example(Content.**class**);  *// 创建查询条件对象* Example.Criteria criteria = example.createCriteria();  *// 添加等于条件 category\_id = categoryId* criteria.andEqualTo(**"categoryId"**, categoryId);  *// 添加等于条件 status = 1* criteria.andEqualTo(**"status"**, **"1"**);  *// 排序(升序) order by sort\_order asc* example.orderBy(**"sortOrder"**).asc();  */\*\* 条件查询 \*/* contentList = **contentMapper**.selectByExample(example);  **try** {  */\*\* 存入Redis缓存 \*/* **redisTemplate.boundValueOps("content").set(contentList);**  }**catch** (Exception ex){}  **return** contentList;  }**catch** (Exception ex){  **throw new** RuntimeException(ex);  } } |

## 6.3更新缓存

当广告数据发生变更时，需要将缓存数据清除，这样再次查询才能获取最新的数据

### 6.3.1新增广告后清除缓存

pinyougou-content-service/src/main/java/com/pinyougou/content/service/impl/ContentServiceImpl.java，修改save方法

|  |
| --- |
| */\*\* 添加广告 \*/* **public void** save(Content content){  **try**{  **contentMapper**.insertSelective(content);  */\*\* 清除Redis缓存 \*/* **redisTemplate.delete**(**"content"**);  }**catch**(Exception ex){  **throw new** RuntimeException(ex);  } } |

### 6.3.2修改广告后清除缓存

pinyougou-content-service/src/main/java/com/pinyougou/content/service/impl/ContentServiceImpl.java，修改update方法

|  |
| --- |
| */\*\* 修改广告 \*/* **public void** update(Content content){  **try**{  **contentMapper**.updateByPrimaryKeySelective(content);  */\*\* 清除Redis缓存 \*/* **redisTemplate**.delete(**"content"**);  }**catch**(Exception ex){  **throw new** RuntimeException(ex);  } } |

### 6.3.3删除广告后清除缓存

pinyougou-content-service/src/main/java/com/pinyougou/content/service/impl/ContentServiceImpl.java，修改deleteAll方法

|  |
| --- |
| */\*\* 批量删除 \*/* **public void** deleteAll(Serializable[] ids){  **try** {  *// 创建示范对象* Example example = **new** Example(Content.**class**);  *// 创建条件对象* Example.Criteria criteria = example.createCriteria();  *// 创建In条件* criteria.andIn(**"id"**, Arrays.*asList*(ids));  *// 根据示范对象删除* **contentMapper**.deleteByExample(example);   ***/\*\* 清除Redis缓存 \*/* redisTemplate.delete("content");**  }**catch** (Exception ex){  **throw new** RuntimeException(ex);  } } |