**淘淘商城**

**第六天**

# 课程计划

1. 实现首页大广告
2. 添加缓存分析
3. Redis安装
4. Redis单机版
5. Redis集群版
6. 项目集成Redis
7. 添加缓存

# 实现首页大广告

前端团队：前端页面效果实现

后端团队：根据前端编写的页面，从后台获取数据进行动态展示

## 开发分析

1. 对页面进行分析，查看首页大广告所需要的数据在什么地方

2. 对前端展示所需要的数据结构进行分析

3. 从后台服务获取数据，封装并返回给前台

## 查找数据位置

taotao-portal是门户系统。里面的index.jsp页面就是首页的jsp

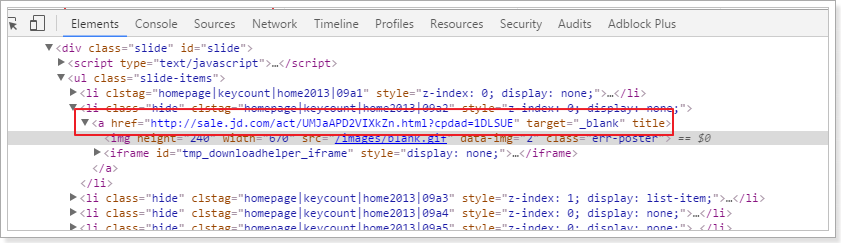
我们需要实现的大广告位就在其中，可以在该页面中进行查找

查找方式（使用谷歌浏览器）：

1. 在大广告位图片点击右键
2. 点击检查

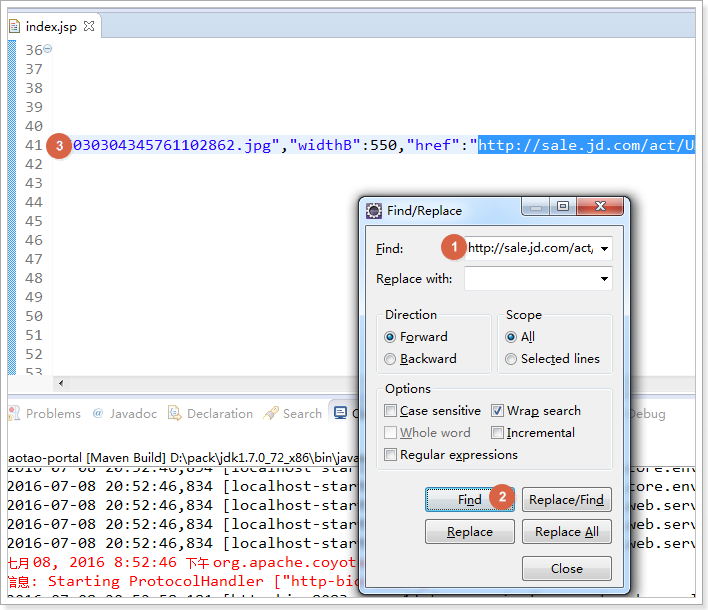


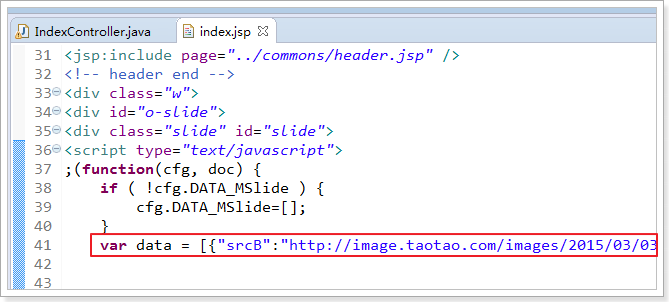
3. 在弹出的窗口复制网址，这是点击图片超链接的url的地址



1. 在taotao-portal的index.jsp页面，搜索该网址

在页面的41行找到数据



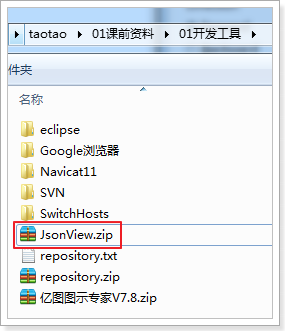


## 分析数据结构

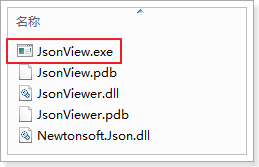
查看数据，发现是json格式的数据

可以使用JsonView对数据进行格式化，方便分析

### jsonView工具

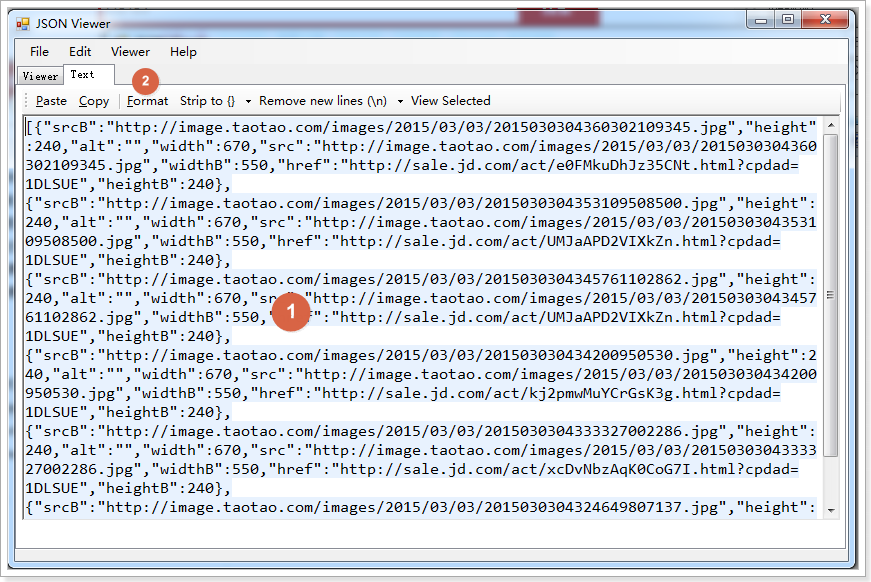


解压效果：



使用：

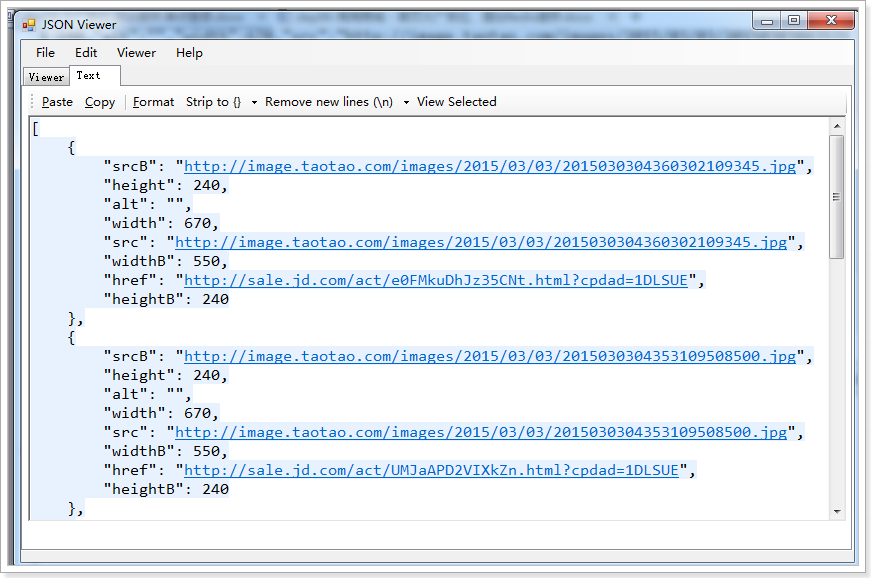
复制json数据进去，点击Format即可



### 分析数据

页面的json数据有6组数据，对应大广告位的6张图片

srcB和src是图片地址，href是图片的超链接，height，width，widthB，heightB是图片的尺寸属性。我们暂时用不到，就保持和原始数据一致



## 实现大广告展示

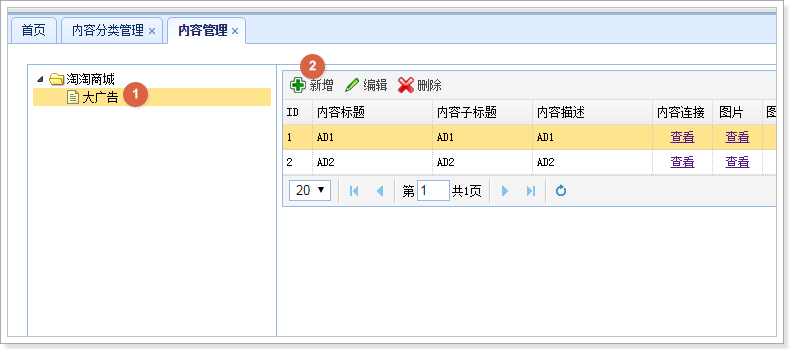
### 数据准备

在后台管理系统中，使用内容管理功能，增加大广告数据

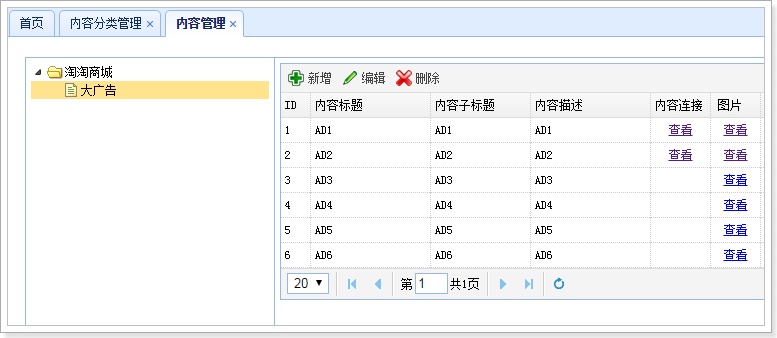
首页的大广告展示模块 对应 内容管理系统的大广告分类

大广告的6条信息 对应 内容管理系统的大广告的内容数据

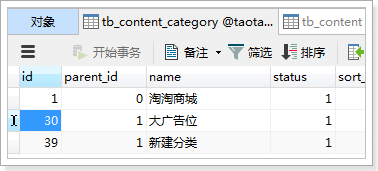
注意：只能保存6条数据，不能多也不能少

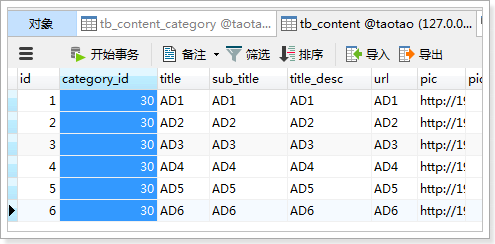


效果



查看数据库内容分类表，可以根据大广告的id查询内容表，就是我们需要的结果

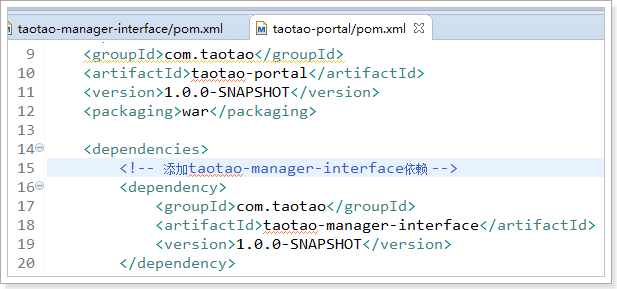




### 实现功能

#### 添加依赖

taotao-portal需要调用ContentService服务获取大广告数据，所以需要加入taotao-manager-interface依赖



#### 声明服务调用

在taotao-portal中，声明服务调用



#### 编写Controller

在taotao-portal修改

@Controller

@RequestMapping("index")

**public** **class** IndexController {

@Autowired

**private** ContentService contentService;

// http://www.taotao.com/index.html

// http://www.taotao.com

/\*\*

\* 首页

\*

\* **@return**

\*/

@RequestMapping

**public** String index(Model model) {

// 使用服务获取大广告的数据

String AD = **this**.contentService.queryAD();

// 把首页大广告数据放到模型中

model.addAttribute("AD", AD);

**return** "index";

}

}

#### 编写service接口

/\*\*

\* 查询首页大广告数据

\* **@return**

\*/

String queryAD();

#### 实现service接口方法

// Jackson的工具类，可以把对象转json格式的数据，也可以把json格式的数据转为对象，还可以方便的解析json格式的数据

**private** **static** **final** ObjectMapper ***MAPPER*** = **new** ObjectMapper();

@Override

**public** String queryAD() {

// 从数据库查询大广告需要的数据，其实就是查询CategoryId为30的数据

Content param = **new** Content();

param.setCategoryId(30l);

List<Content> list = **super**.queryListByWhere(param);

// 把查到的数据进行封装，List<Map>

// 声明容器封装数据

List<Map<String, Object>> results = **new** ArrayList<>();

**for** (Content content : list) {

Map<String, Object> map = **new** HashMap<>();

map.put("srcB", content.getPic());

map.put("height", 240);

map.put("alt", "");

map.put("width", 670);

map.put("src", content.getPic());

map.put("widthB", 550);

map.put("href", content.getUrl());

map.put("heightB", 240);

// 把封装好的map放到容器中

results.add(map);

}

// 把封装好的list<Map>,转为json格式的数据

String json = "";

**try** {

json = ***MAPPER***.writeValueAsString(results);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

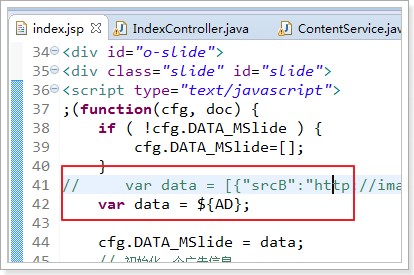
}

**return** json;

}

#### 修改页面展示数据

修改taotao-portal的index.jsp页面，数据是从后台获取



### 效果



# 添加缓存分析

## 首页大广告添加缓存

为什么要添加缓存？

随着访问用户越来越多，并发越来越大，直接从MySQL数据库查询性能较低。

可以使用把数据进行缓存，从缓存读取速度更快

什么样的数据需要添加缓存？

1. 访问频率高
2. 更新频率低

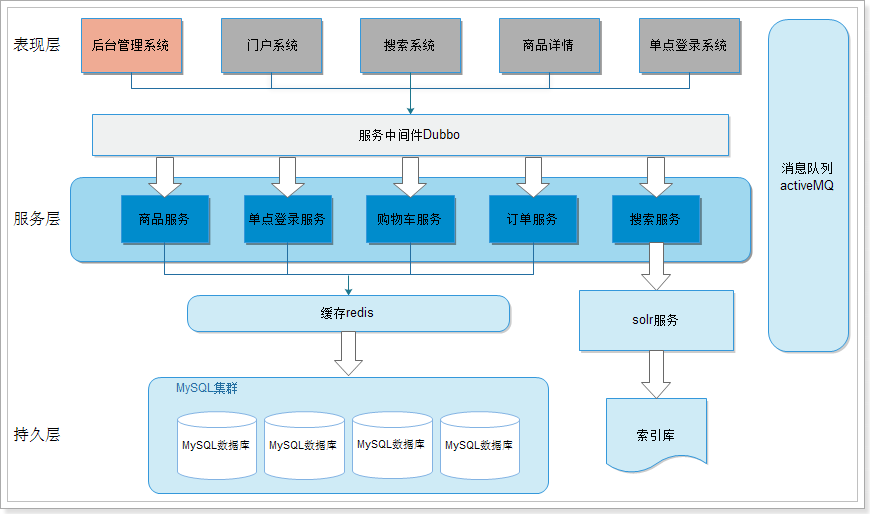
首页访问量很大，大广告的更新频率较低，所以可以添加缓存

可以使用redis增加缓存

因为redis的读写速度快

我们的架构是把缓存放在服务层

服务层把需要缓存的数据放在redis中，先从redis查询数据，如果redis没有数据，再去MySQL查询。



# redis安装

## redis回顾

redis是是非关系型数据库，即NoSql数据库

存储的数据模型是key-value

特点是访问速度快，可以作为缓存使用

Redis把数据存放到内存中，内存的读写速度比硬盘的读写速度快

持久化方案（了解）

RDB 快照

AOF 保存命令日志

支持多种数据类型：

1. String类型。Value只能是String类型。
2. Hash类型。可以对key进行分类。
3. List类型。所有元素是有序的
4. Set类型。元素是无序的，元素是不能重复。集合操作，并集，交集，差集
5. SortedSet（zset）有序的set，元素不能重复且有序。

redis是单线程，但是Redis的读写速度太快了，

官方的一个简单测试：

　　测试完成了**50个并发**执行**100000个请求**。

　　设置和获取的值是一个256字节字符串。

　　结果:**读的速度是110000次/s,写的速度是81000次/s**

## redis单机版安装

### 环境准备

redis安装在Linux上，redis是c语言开发的，需要gcc-c++环境

查看是否安装gcc-c++

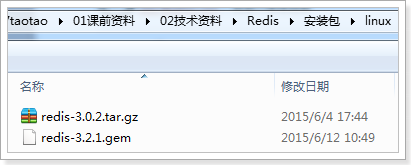
[root@itcast-01 ~]# rpm -qa | grep gcc-c++

gcc-c++-4.4.7-17.el6.i686

如果没有安装，可以使用yum命令进行网络安装

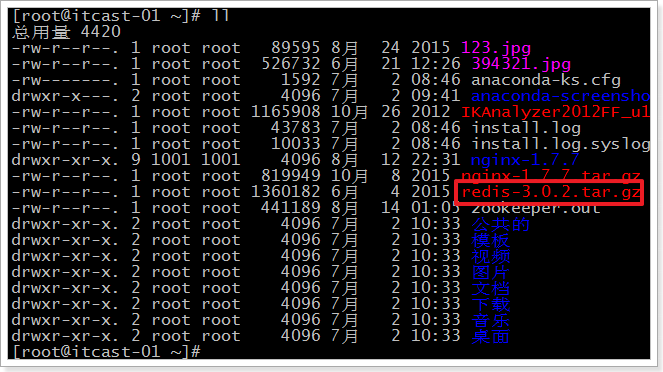
[root@itcast-01 ~]# yum install gcc-c++

### 上传安装包



在Linux的CRT客户端，按alt+p进行上传，可以直接把安装包拖动上去

进入root目录，查看上传包



### 解压安装包

[root@itcast-01 ~]# tar -zxvf redis-3.0.2.tar.gz

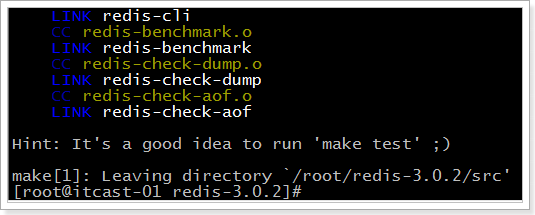
### 执行安装

进入解压的文件夹，进行编译

[root@itcast-01 ~]# cd redis-3.0.2

[root@itcast-01 redis-3.0.2]# make

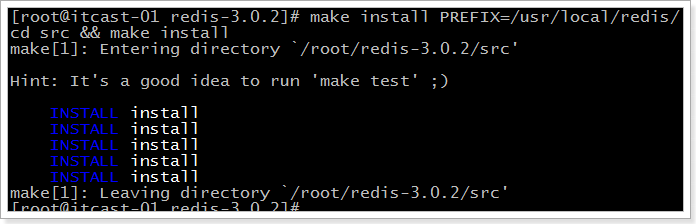
成功效果：



执行安装，指定安装路径

[root@itcast-01 redis-3.0.2]# make install PREFIX=/usr/local/redis

成功效果：



### 查看安装目录

进入目录

[root@itcast-01 redis-3.0.2]# cd /usr/local/redis/bin/



常用的是客户端和服务端

redis-cli 是客户端

redis-server 是服务端

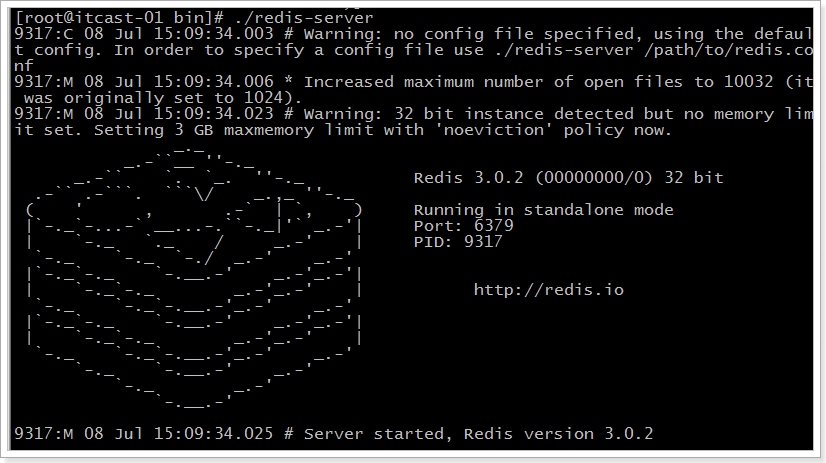
### 启动redis

#### 前端启动

启动命令

[root@itcast-01 bin]# ./redis-server

确认redis安装正常，效果：



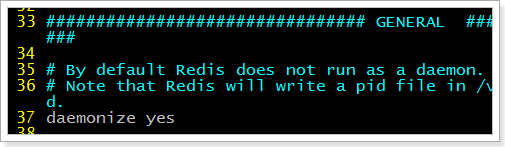
#### 修改为后端启动

从安装包中复制redis.conf文件到/usr/local/redis/bin中

[root@itcast-01 bin]# cp /root/redis-3.0.2/redis.conf ./

修改redis.conf配置文件，修改为yes

[root@itcast-01 bin]# vim redis.conf

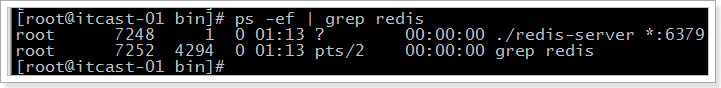


启动命令

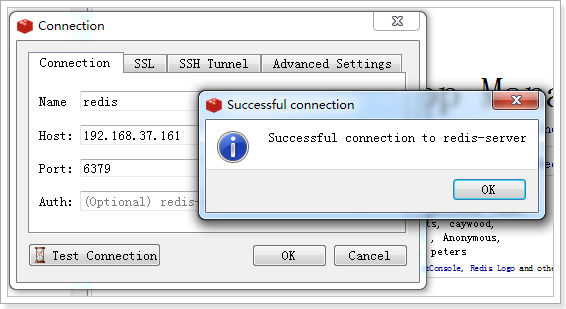
[root@itcast-01 bin]# ./redis-server redis.conf

查看启动效果：

[root@itcast-01 bin]# ps -ef | grep redis

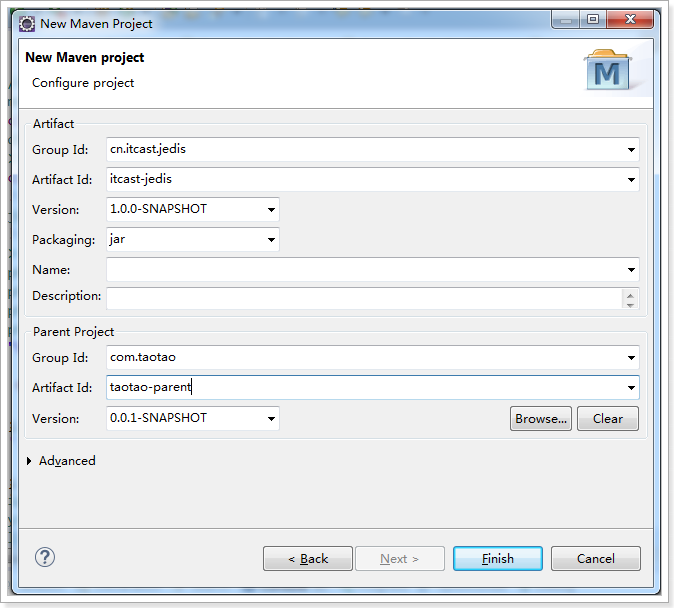


使用图形化界面测试访问成功

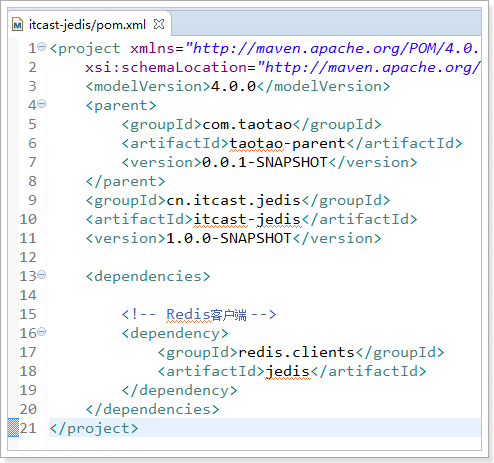


## Jedis回顾

### 创建itcast-jedis工程



### 加入依赖



### 测试jedis

**public** **class** JedisTest {

**public** **static** **void** main(String[] args) {

// 1. 创建jedis对象，需要ip和port

Jedis jedis = **new** Jedis("192.168.37.161", 6379);

// 2. 使用jedis操作redis，jedis的方法和命令是一样的

jedis.select(2);

String pong = jedis.ping();

System.***out***.println(pong);

String key = "jedis";

String setResult = jedis.set(key, "hello redis!");

System.***out***.println(setResult);

String getResult = jedis.get(key);

System.***out***.println(getResult);

// 3. 释放资源，关闭jedis

jedis.close();

}

}

### 测试jedisPool

使用jedisPool连接池操作,项目使用连接池

**public** **class** JedisPoolTest {

**public** **static** **void** main(String[] args) {

// 1. 创建连接池JedisPool

JedisPool jedisPool = **new** JedisPool("192.168.37.161", 6379);

// 2. 从连接池获取连接jedis

Jedis jedis = jedisPool.getResource();

// 3. 使用连接jedis操作redis

jedis.select(2);

String pong = jedis.ping();

System.***out***.println(pong);

String key = "jedisPoll";

String setResult = jedis.set(key, "hello redis!");

System.***out***.println(setResult);

String getResult = jedis.get(key);

System.***out***.println(getResult);

// 4.释放资源，每次用完jedis连接，必须close（）,指的就是还回连接池

jedis.close();

// 程序结束，销毁连接池

jedisPool.close();

}

}

## Redis集群版原理

### Redis-cluster架构图

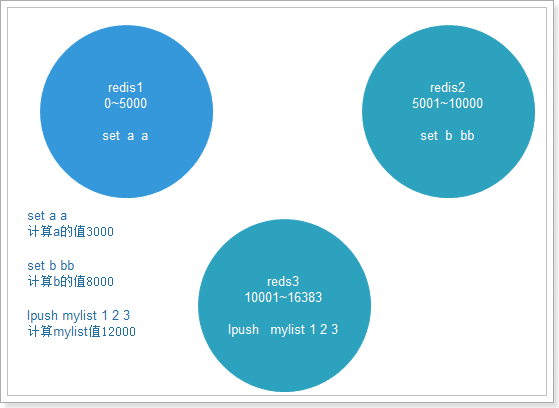


1. 所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽。
2. 节点的fail是通过集群中超过半数的节点检测失效时才生效。

(3)客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可。

(4)redis-cluster把所有的物理节点映射到[0-16383]slot上,cluster 负责维护node<->slot<->value

Redis 集群中内置了 16384 个哈希槽，当需要在 Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点



### redis-cluster投票:容错



(1)投票过程是集群中所有master参与,如果半数以上master节点与master节点通信超时(cluster-node-timeout),认为当前master节点挂掉.

(2):什么时候整个集群不可用(cluster\_state:fail)?

    a:如果集群任意master挂掉,且当前master没有slave.集群进入fail状态,也可以理解成集群的slot映射[0-16383]不完整时进入fail状态. ps : redis-3.0.0.rc1加入cluster-require-full-coverage参数,默认关闭,打开集群兼容部分失败.

    b:如果集群超过半数以上master挂掉，无论是否有slave，集群进入fail状态.

## Redis集群搭建

为了保证可以进行投票，需要至少3个主节点。

每个主节点都需要至少一个从节点,所以需要至少3个从节点

一共需要6台redis服务器

可以使用6个redis实例

6个redis实例的端口号,7001~7006

不要求学员搭建集群，只需要打开Redis集群的节点，启动命令：

[root@itcast-01 ~]# cd /usr/local/redis/redis-cluster

[root@itcast-01 ~]# ./start-all.sh

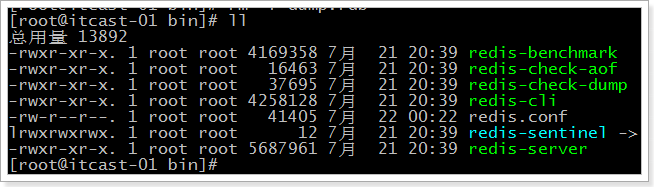
### 准备6个redis实例

1. 把bin目录里面的rdb,和aof文件删除,准备干净的redis

[root@localhost bin]# rm -rf appendonly.aof

[root@localhost bin]# rm -f dump.rdb

效果：



1. 把bin复制6份

[root@localhost bin]# cd ..

[root@localhost redis]# cp -r bin redis-cluster/redis1

[root@localhost redis]# cp -r bin redis-cluster/redis2

[root@localhost redis]# cp -r bin redis-cluster/redis3

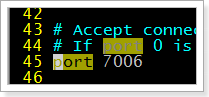
[root@localhost redis]# cp -r bin redis-cluster/redis4

[root@localhost redis]# cp -r bin redis-cluster/redis5

[root@localhost redis]# cp -r bin redis-cluster/redis6

1. 修改端口号

依次修改端口号为7001~7006



1. 启动6个redis实例

编写脚本实现

1. 编写脚本

[root@localhost redis-cluster]# vim start-all.sh

cd redis1

./redis-server redis.conf

cd ..

cd redis2

./redis-server redis.conf

cd ..

cd redis3

./redis-server redis.conf

cd ..

cd redis4

./redis-server redis.conf

cd ..

cd redis5

./redis-server redis.conf

cd ..

cd redis6

./redis-server redis.conf

cd ..

1. 设置脚本启动权限

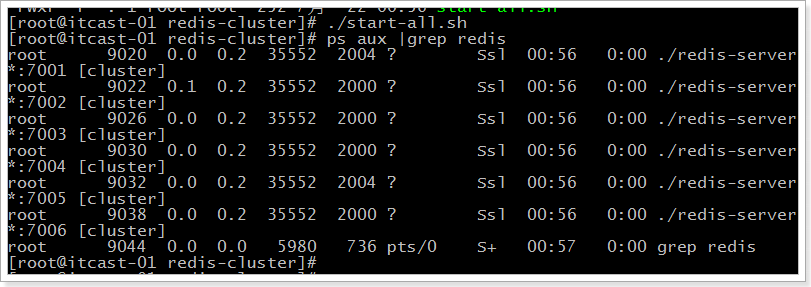
[root@localhost redis-cluster]# chmod u+x start-all.sh

1. 执行脚本

[root@localhost redis-cluster]# ./start-all.sh

[root@localhost redis-cluster]# ps aux | grep redis

效果



### 准备redis集群的安装环境

redis集群的管理工具使用的是ruby脚本语言，安装集群需要ruby环境。

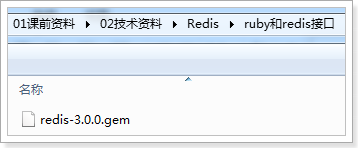
安装ruby环境(老师提供的虚拟机，环境已经装好)

[root@itcast-01 redis-cluster]# yum install ruby

安装Ruby的打包系统

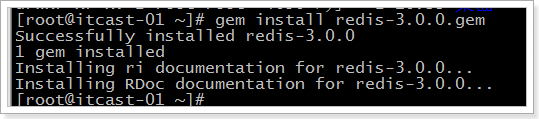
[root@itcast-01 redis-cluster]# yum install rubygems

安装redis的ruby接口库



先上传，再执行命令安装

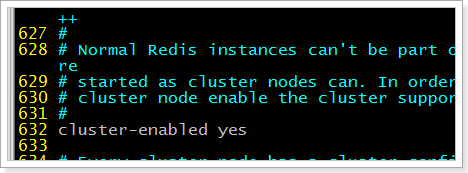
[root@itcast-01 ~]# gem install redis-3.0.0.gem



### 安装redis集群

需要修改每个实例的redis.conf配置文件,开启redis-cluster

[root@localhost redis-cluster]# vim redis1/redis.conf



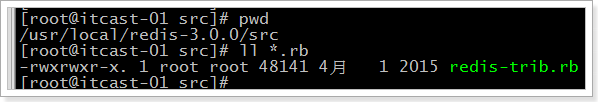
需要修改6个节点

重启redis实例

[root@localhost redis-cluster]# ./start-all.sh

集群管理工具在redis解压文件夹的src的文件夹中

使用redis-cluster的集群管理工具启动集群



先进入集群管理工具所在的路径

[root@localhost redis-cluster]# cd /root/redis-3.0.0/src/

再启动命令（注意修改为自己的ip最好不用127.0.0.1和port）

[root@localhost src]# ./redis-trib.rb create --replicas 1 192.168.37.161:7001 192.168.37.161:7002 192.168.37.161:7003 192.168.37.161:7004 192.168.37.161:7005 192.168.37.161:7006

启动信息

>>> Creating cluster

Connecting to node 192.168.37.131:7001: OK

Connecting to node 192.168.37.131:7002: OK

Connecting to node 192.168.37.131:7003: OK

Connecting to node 192.168.37.131:7004: OK

Connecting to node 192.168.37.131:7005: OK

Connecting to node 192.168.37.131:7006: OK

>>> Performing hash slots allocation on 6 nodes...

Using 3 masters:

192.168.37.131:7001

192.168.37.131:7002

192.168.37.131:7003

Adding replica 192.168.37.131:7004 to 192.168.37.131:7001

Adding replica 192.168.37.131:7005 to 192.168.37.131:7002

Adding replica 192.168.37.131:7006 to 192.168.37.131:7003

M: 955567f988830cdf4328854f815719ea28082ca8 192.168.37.131:7001

slots:0-5460 (5461 slots) master

M: 4f3eeced04b930aa48193699301745a05a70697a 192.168.37.131:7002

slots:5461-10922 (5462 slots) master

M: 871a684dbbc0f43dcc16107710d7bd2f4e6de76a 192.168.37.131:7003

slots:10923-16383 (5461 slots) master

S: baca3ce2223dfcd9c636a7193b12998b1dbb2431 192.168.37.131:7004

replicates 955567f988830cdf4328854f815719ea28082ca8

S: 0b599863ddf2e03b0326c75b874a1af8ae430d2e 192.168.37.131:7005

replicates 4f3eeced04b930aa48193699301745a05a70697a

S: 92f712d954c62f2743e2e572f6582a6ef9a163e0 192.168.37.131:7006

replicates 871a684dbbc0f43dcc16107710d7bd2f4e6de76a

Can I set the above configuration? (type 'yes' to accept): yes

>>> Nodes configuration updated

>>> Assign a different config epoch to each node

>>> Sending CLUSTER MEET messages to join the cluster

Waiting for the cluster to join..

>>> Performing Cluster Check (using node 192.168.37.131:7001)

M: 955567f988830cdf4328854f815719ea28082ca8 192.168.37.131:7001

slots:0-5460 (5461 slots) master

M: 4f3eeced04b930aa48193699301745a05a70697a 192.168.37.131:7002

slots:5461-10922 (5462 slots) master

M: 871a684dbbc0f43dcc16107710d7bd2f4e6de76a 192.168.37.131:7003

slots:10923-16383 (5461 slots) master

M: baca3ce2223dfcd9c636a7193b12998b1dbb2431 192.168.37.131:7004

slots: (0 slots) master

replicates 955567f988830cdf4328854f815719ea28082ca8

M: 0b599863ddf2e03b0326c75b874a1af8ae430d2e 192.168.37.131:7005

slots: (0 slots) master

replicates 4f3eeced04b930aa48193699301745a05a70697a

M: 92f712d954c62f2743e2e572f6582a6ef9a163e0 192.168.37.131:7006

slots: (0 slots) master

replicates 871a684dbbc0f43dcc16107710d7bd2f4e6de76a

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

## redis集群连接

redis的单机版，默认是16个数据库，但是redis-Cluster集群版，只有一个数据库

### 使用redis命令行客户端连接

[root@localhost bin]# ./redis-cli -h 192.168.37.131 -p 7006 **-c**

192.168.37.131:7006> set hello money

-> Redirected to slot [866] located at 192.168.37.131:7001

OK

一定要加-c参数,节点之间就可以互相跳转

### redis的java客户端连接(Jedis)

使用的是JedisCluster类进行连接

**public** **class** JedisClusterTest {

**public** **static** **void** main(String[] args) {

// 1. 创建JedisCluster对象

// 设置节点信息,放到set中

Set<HostAndPort> nodes = **new** HashSet<>();

nodes.add(**new** HostAndPort("192.168.37.136", 7001));

nodes.add(**new** HostAndPort("192.168.37.136", 7002));

nodes.add(**new** HostAndPort("192.168.37.136", 7003));

nodes.add(**new** HostAndPort("192.168.37.136", 7004));

nodes.add(**new** HostAndPort("192.168.37.136", 7005));

nodes.add(**new** HostAndPort("192.168.37.136", 7006));

// 创建对象

JedisCluster jedisCluster = **new** JedisCluster(nodes);

// 2. 使用jedisCluster操作redis,方法和jedis是一样的

String key = "jedisCluster";

String setResult = jedisCluster.set(key, "Hello redis !");

String getResult = jedisCluster.get(key);

System.***out***.println(setResult + ":" + getResult);

// 3. 关闭jedisCluster连接(在程序执行完之后,才能关闭,他的内部已经封装了连接池)

jedisCluster.close();

}

}

# 添加缓存

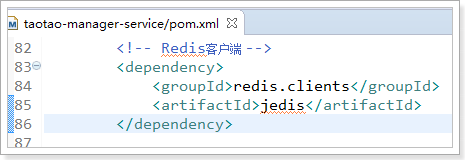
## 项目集成Redis

根据架构，需要给内容管理服务添加缓存

缓存的添加在service层，所以把redis集成到taotao-manager-service中

### 添加依赖

在taotao-manager-service添加redis依赖



### redis整合分析

因为集群需要的服务器至少是6台，使用Redis集群的成本很高，所以如果没有业务需求，不会使用Redis集群。

而使用Redis集群的公司，为了节省成本，一般只会在生产环境使用集群，而开发环境使用Redis单机版，所以我们在整合项目的时候，需要单机版和集群版都要有。

我们可以创建一个接口，再编写单机版和集群版的实现类，使用spring进行管理，在部署时，使用哪种Redis，就切换那种实现类。

在taotao-manager-service创建com.taotao.manager.redis包进行管理

### 编写RedisUtils接口

在taotao-manager-service编写

**public** **interface** RedisUtils{

/\*\*

\* 保存

\*

\* **@param** key

\* **@param** value

\*/

**public** **void** set(String key, String value);

/\*\*

\* 根据key查询

\*

\* **@param** key

\* **@return**

\*/

**public** String get(String key);

/\*\*

\* 删除

\*

\* **@param** key

\*/

**public** **void** del(String key);

/\*\*

\* 根据key设置生存时间

\*

\* **@param** key

\* **@param** seconds

\*/

**public** **void** expire(String key, Integer seconds);

/\*\*

\* 保存并设置生存时间

\*

\* **@param** key

\* **@param** value

\* **@param** seconds

\*/

**public** **void** set(String key, String value, Integer seconds);

/\*\*

\* value加一

\*

\* **@param** key

\* **@return**

\*/

**public** Long incr(String key);

}

### 编写RedisUtils实现类

#### 单机版

在taotao-manager-service编写

**public** **class** RedisPool **implements** RedisUtils {

@Autowired

**private** JedisPool jedisPool;

@Override

**public** **void** set(String key, String value) {

Jedis jedis = **this**.jedisPool.getResource();

jedis.set(key, value);

jedis.close();

}

@Override

**public** String get(String key) {

Jedis jedis = **this**.jedisPool.getResource();

String result = jedis.get(key);

jedis.close();

**return** result;

}

@Override

**public** **void** del(String key) {

Jedis jedis = **this**.jedisPool.getResource();

jedis.del(key);

jedis.close();

}

@Override

**public** **void** expire(String key, Integer seconds) {

Jedis jedis = **this**.jedisPool.getResource();

jedis.expire(key, seconds);

jedis.close();

}

@Override

**public** **void** set(String key, String value, Integer seconds) {

Jedis jedis = **this**.jedisPool.getResource();

jedis.set(key, value);

jedis.expire(key, seconds);

jedis.close();

}

@Override

**public** Long incr(String key) {

Jedis jedis = **this**.jedisPool.getResource();

Long count = jedis.incr(key);

jedis.close();

**return** count;

}

}

#### 集群版

在taotao-manager-service编写

**public** **class** RedisCluster **implements** RedisUtils {

@Autowired

**private** JedisCluster jedisCluster;

@Override

**public** **void** set(String key, String value) {

**this**.jedisCluster.set(key, value);

}

@Override

**public** String get(String key) {

String result = **this**.jedisCluster.get(key);

**return** result;

}

@Override

**public** **void** del(String key) {

**this**.jedisCluster.del(key);

}

@Override

**public** **void** expire(String key, Integer seconds) {

**this**.jedisCluster.expire(key, seconds);

}

@Override

**public** **void** set(String key, String value, Integer seconds) {

**this**.jedisCluster.set(key, value);

**this**.jedisCluster.expire(key, seconds);

}

@Override

**public** Long incr(String key) {

Long count = **this**.jedisCluster.incr(key);

**return** count;

}

}

### 配置Spring管理Redis

在taotao-manager-service创建applicationContext-redis.xml

**注意：使用的时候开启一个配置即可**

<?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:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

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-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 配置jedis连接池 -->

<bean id=*"jedisPool"* class=*"redis.clients.jedis.JedisPool"*>

<constructor-arg name=*"host"* value=*"${redis.host}"* />

<constructor-arg name=*"port"* value=*"${redis.port}"* />

</bean>

<!-- 配置操作单机版redis的实现类 -->

<bean id=*"poolJedisClient"* class=*"com.taotao.manager.redis.impl.RedisPool"* />

<!-- 配置JedisCluster -->

<bean id=*"jedisCluster"* class=*"redis.clients.jedis.JedisCluster"*>

<!-- 配置集群信息 -->

<constructor-arg name=*"nodes"*>

<set>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host1}"* />

<constructor-arg name=*"port"* value=*"${cluster.port1}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host2}"* />

<constructor-arg name=*"port"* value=*"${cluster.port2}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host3}"* />

<constructor-arg name=*"port"* value=*"${cluster.port3}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host4}"* />

<constructor-arg name=*"port"* value=*"${cluster.port4}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host5}"* />

<constructor-arg name=*"port"* value=*"${cluster.port5}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster.host6}"* />

<constructor-arg name=*"port"* value=*"${cluster.port6}"* />

</bean>

</set>

</constructor-arg>

</bean>

<!-- 配置操作集群版实现类 -->

<bean id=*"clusterJedisClient"* class=*"com.taotao.manager.redis.impl.RedisCluster"*></bean>

</beans>

### 添加redis.properties

在taotao-manager-service添加redis.properties

#redis单机版信息

redis.host=192.168.37.161

redis.port=6379

#redis集群信息

cluster.host1=192.168.37.161

cluster.port1=7001

cluster.host2=192.168.37.161

cluster.port2=7002

cluster.host3=192.168.37.161

cluster.port3=7003

cluster.host4=192.168.37.161

cluster.port4=7004

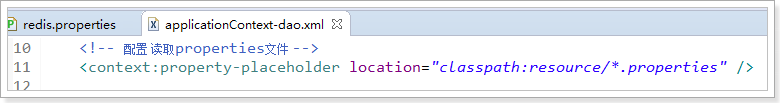
cluster.host5=192.168.37.161

cluster.port5=7005

cluster.host6=192.168.37.161

cluster.port6=7006

修改applicationContext-dao.xml加载redis.properties



## 首页大广告添加缓存

添加缓存的步骤

1. 从缓存中命中
2. 没有命中执行原有逻辑，从MySQL中查询
3. 把数据保存在redis中

添加缓存的原则

不能够影响正常的业务逻辑，可以进行try/catch

// Jackson的工具类，可以把对象转json格式的数据，也可以把json格式的数据转为对象，还可以方便的解析json格式的数据

**private** **static** **final** ObjectMapper ***MAPPER*** = **new** ObjectMapper();

@Autowired

**private** RedisUtils redisUtils;

@Value("${TAOTAO\_PORTAL\_AD}")

**private** String TAOTAO\_PORTAL\_AD;

@Override

**public** String queryAD() {

// 1. 从缓存中命中

// 为了很好的管理和维护redis，需要redis的key是有意义的

**try** {

String redisJson = **this**.redisUtils.get(**this**.TAOTAO\_PORTAL\_AD);

// 判断是否为空，如果不为空表示命中了，直接返回

**if** (StringUtils.*isNotBlank*(redisJson)) {

**return** redisJson;

}

} **catch** (Exception e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

// 2. 如果没有命中，执行原有逻辑，查MySQL

// 从数据库查询大广告需要的数据，其实就是查询CategoryId为30的数据

Content param = **new** Content();

param.setCategoryId(30l);

List<Content> list = **super**.queryListByWhere(param);

// 把查到的数据进行封装，List<Map>

// 声明容器封装数据

List<Map<String, Object>> results = **new** ArrayList<>();

**for** (Content content : list) {

Map<String, Object> map = **new** HashMap<>();

map.put("srcB", content.getPic());

map.put("height", 240);

map.put("alt", "");

map.put("width", 670);

map.put("src", content.getPic());

map.put("widthB", 550);

map.put("href", content.getUrl());

map.put("heightB", 240);

// 把封装好的map放到容器中

results.add(map);

}

// 把封装好的list<Map>,转为json格式的数据

String json = "";

**try** {

json = ***MAPPER***.writeValueAsString(results);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**try** {

// 3. 把查到的数据放到redis中

**this**.redisUtils.expire(**this**.TAOTAO\_PORTAL\_AD, json, 60 \* 60 \* 24);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** json;

}

在taotao-manager-service的配置文件jedis.properties增加配置

