# 今日大纲

1. 学习Redis
2. 使用Redis完成项目中缓存需求
3. 实现商品详情页功能

# 缓存的需求

大广告位数据无需每次查询后台系统的接口，可以在前台系统添加缓存，提高访问首页的速度。

商品类目的数据也可以缓存起来。

实现：

使用Redis实现缓存。

目前缓存的主流技术：

1. Redis
2. Memcached

二者谁的性能更高？

1. 单纯从缓存命中的角度来说，是Memcached要高，Redis和Memcache的差距不大
2. 但是，Redis提供的功能更加的强大

二者的区别是什么？

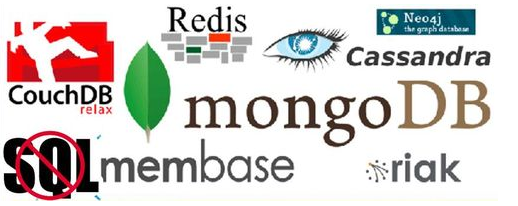
1. Memcache是多线程
2. Redis是单线程

# Redis

## NoSQL

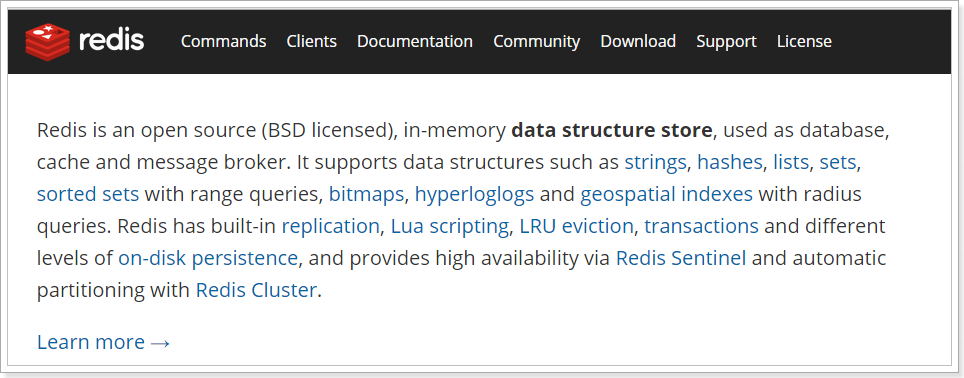


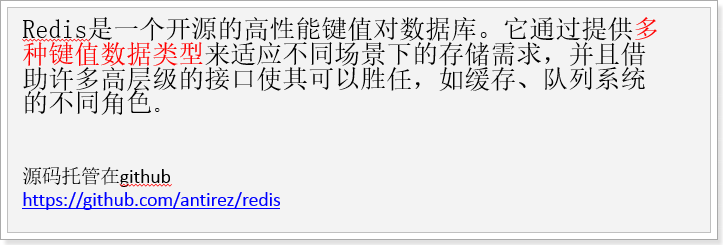
## 主流的NoSQL产品

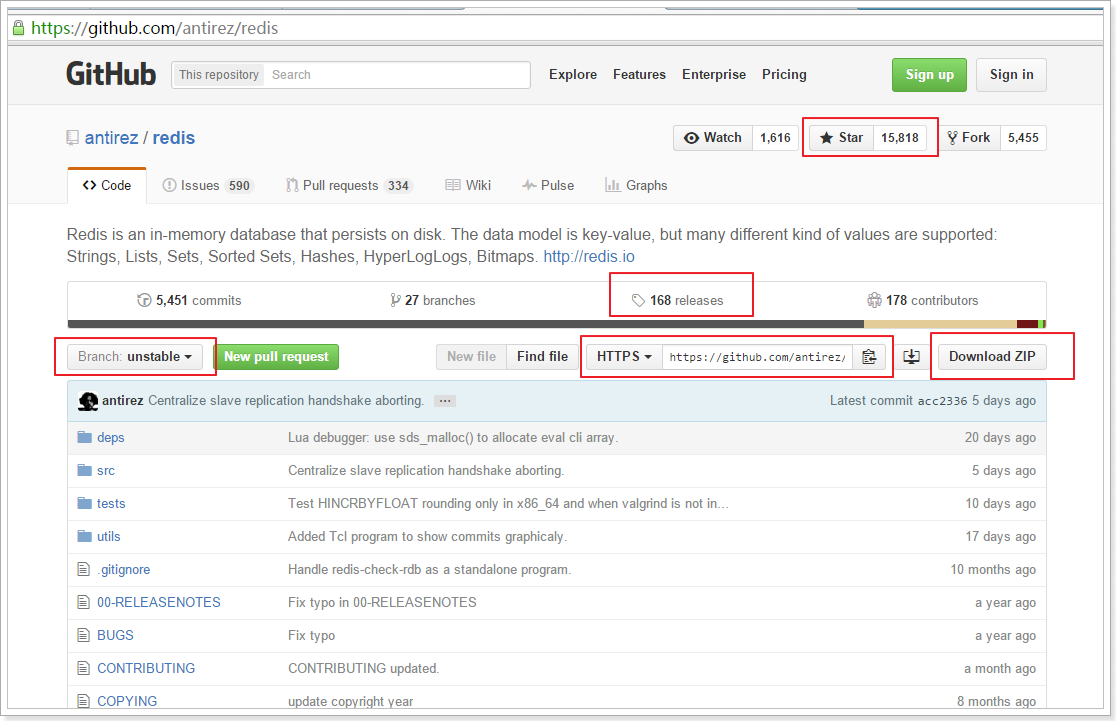


## Redis简介

Redis官网: <http://redis.io/>



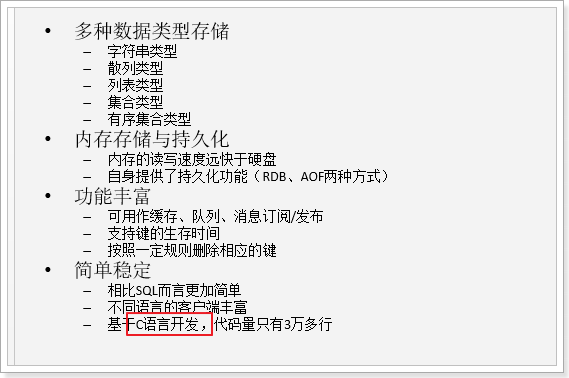




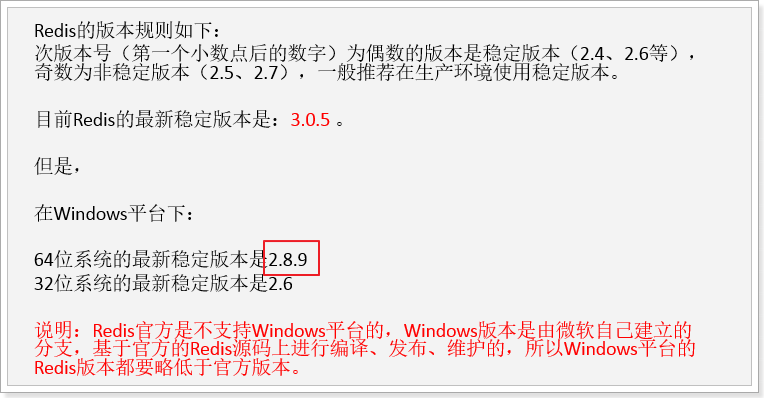
## 历史与发展



## Redis的特性



## Redis版本说明



## 下载Redis

Linux版本 2.8.11 ：

<http://download.redis.io/releases/redis-2.8.11.tar.gz>

Windows(64位)版本 2.8.9 ：

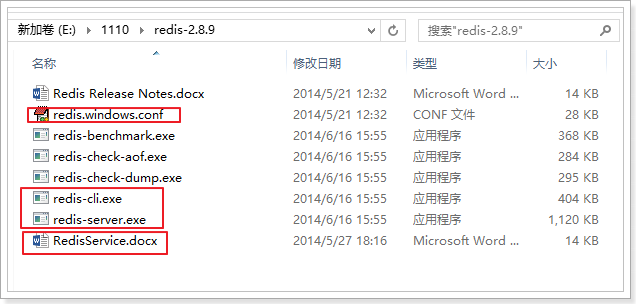
<https://github.com/MSOpenTech/redis/blob/2.8/bin/release/redis-2.8.9.zip?raw=true>

Windows(32位)版本 2.6 ：

<https://github.com/MSOpenTech/redis/blob/2.6/bin/release/redisbin.zip?raw=true>

## Redis的安装

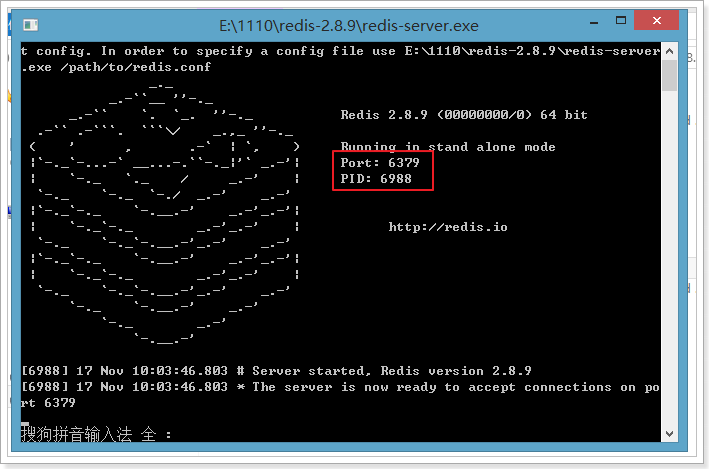
### 安装文件



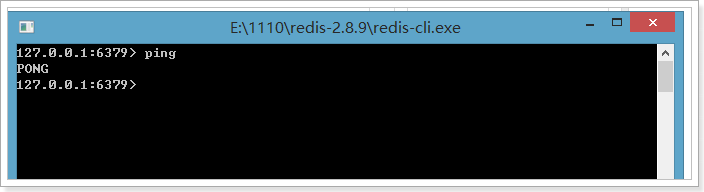


### 安装方式一

双机打开redis-server.exe即可：

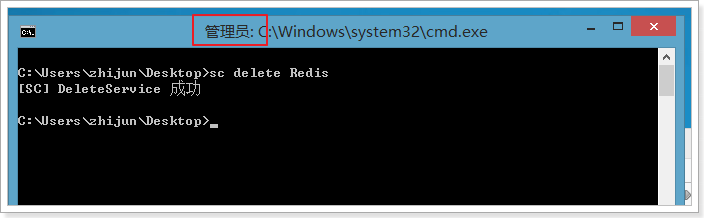


测试：

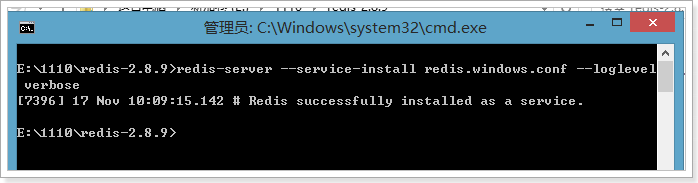


### 安装方式二（安装到系统服务）

先删除原有的系统服务：



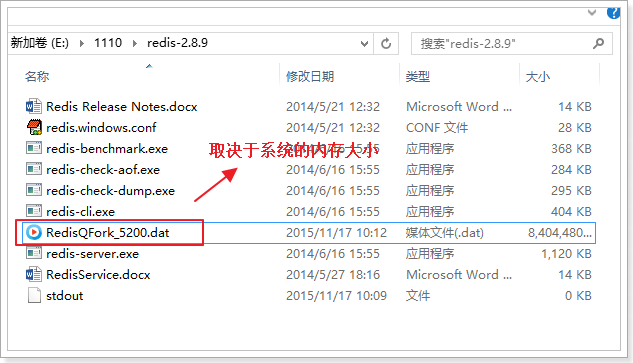
安装服务：



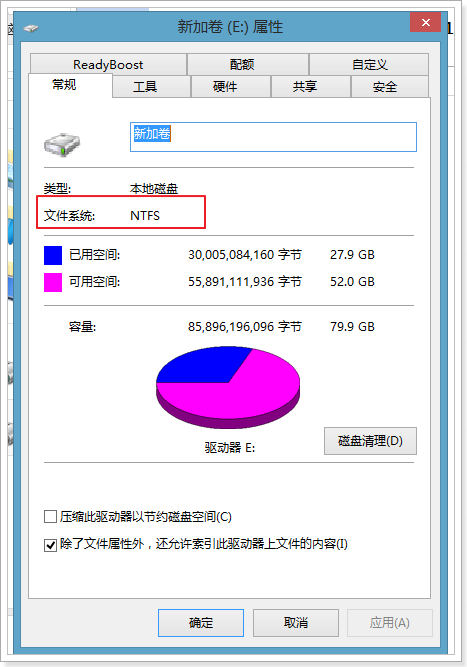
### 32位操作系统安装

只能通过双击打开redis-server.exe启动，不能安装到系统服务。

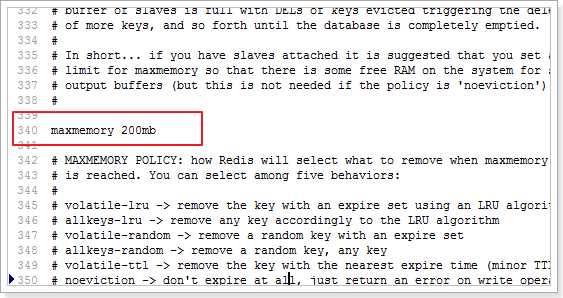
### 注意事项

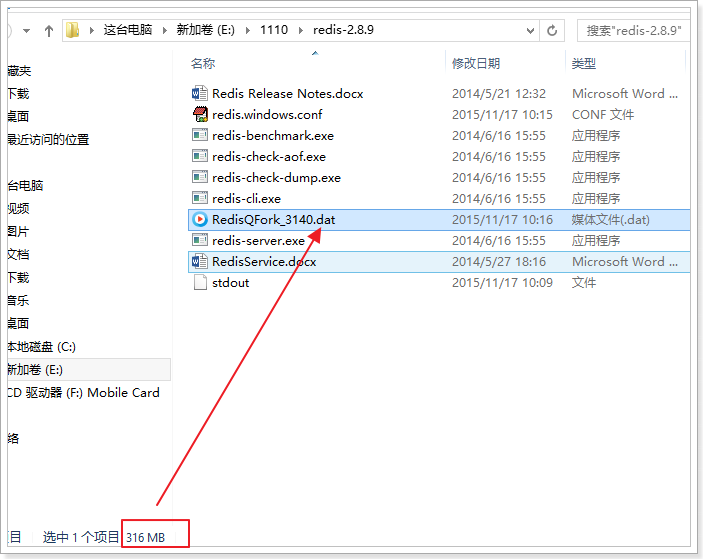


由于文件系统非NTFS，导致Redis启动失败：



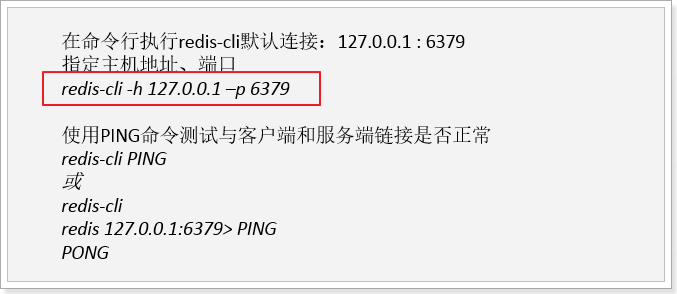
限制Redis的最大内存：



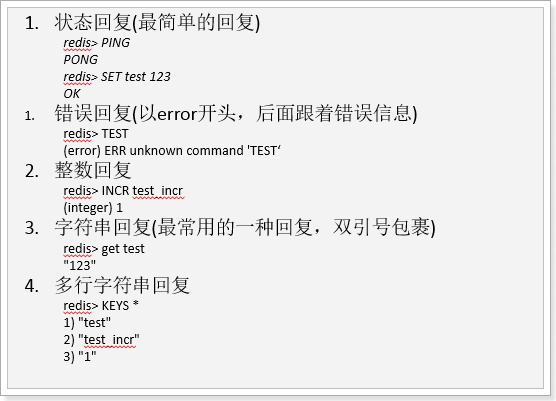


## Redis-cli使用

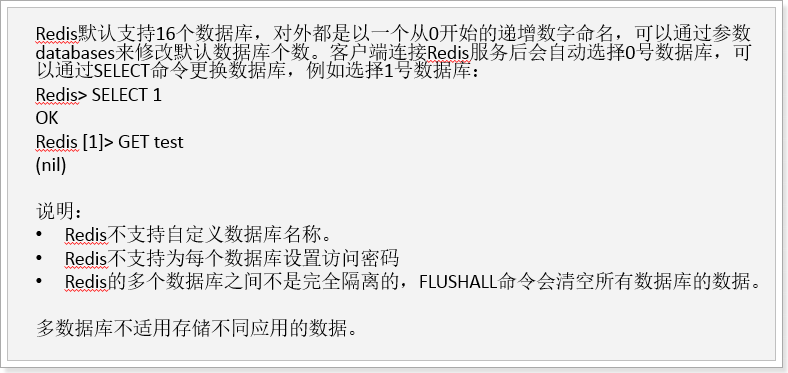
### redis-cli的使用之发送命令



### redis-cli的使用之命令返回值



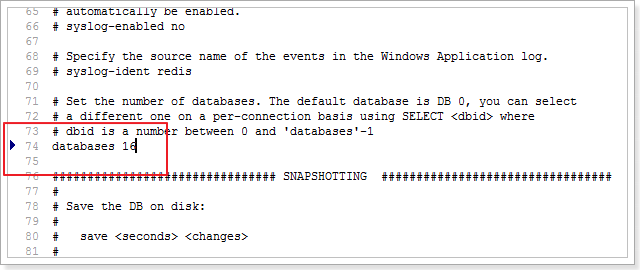
## Redis的多数据库



FLUSHALL -- 清空所有数据库的所有数据

FLUSHDB -- 清空当前所在数据库的数据

### 配置数据库数量

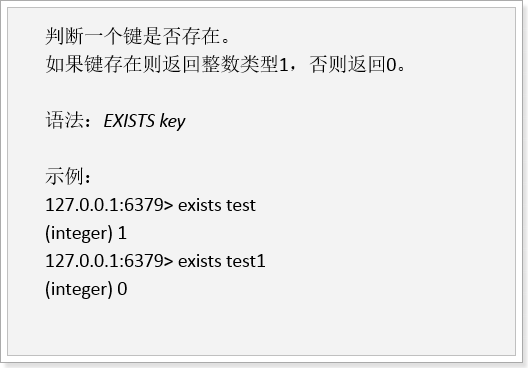


## Redis的基本命令

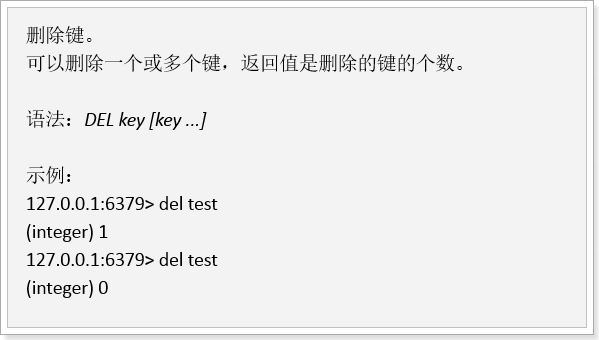
### KEYS



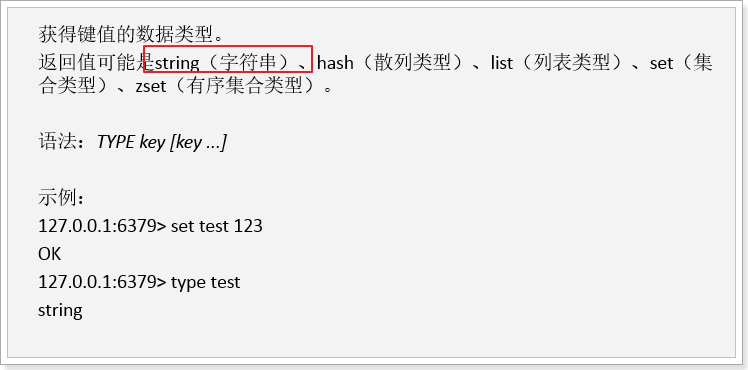
### EXISTS



### DEL



### TYPE



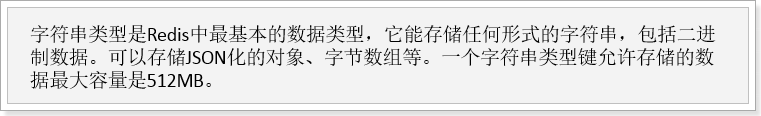
### HELP

HELP 空格 tab键

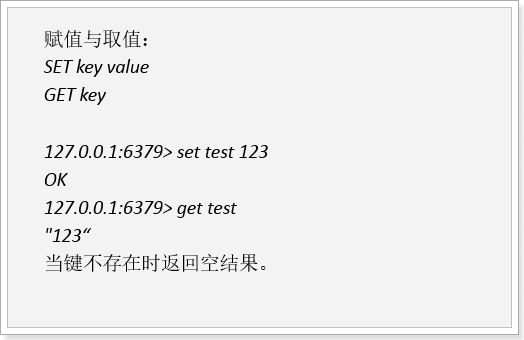


# Redis的字符串数据类型

## 字符串类型



## GET、SET



## INCR



## INCRBY



## DECR、DECRBY



## APPEND



## STRLEN

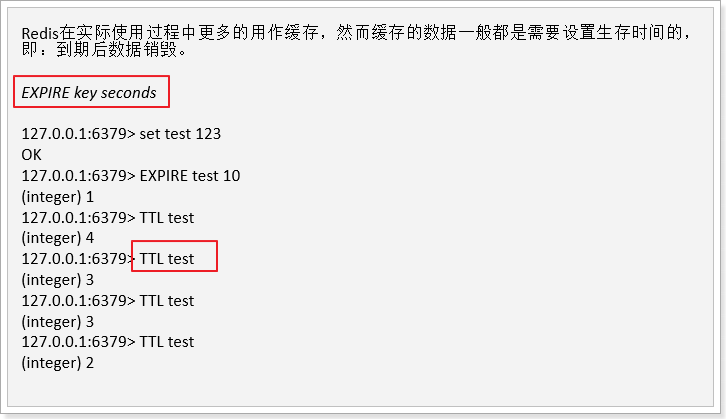


## MSET、MGET



# Redis之生存时间

## 设置生存时间



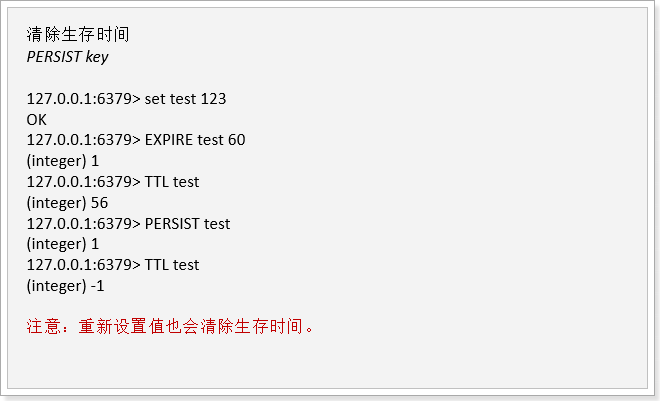
TTL返回值：

大于0的数字：剩余生存时间，单位为秒

-1 ： 没有生存时间，永久存储

-2 ： 数据已经被删除

## 清除生存时间

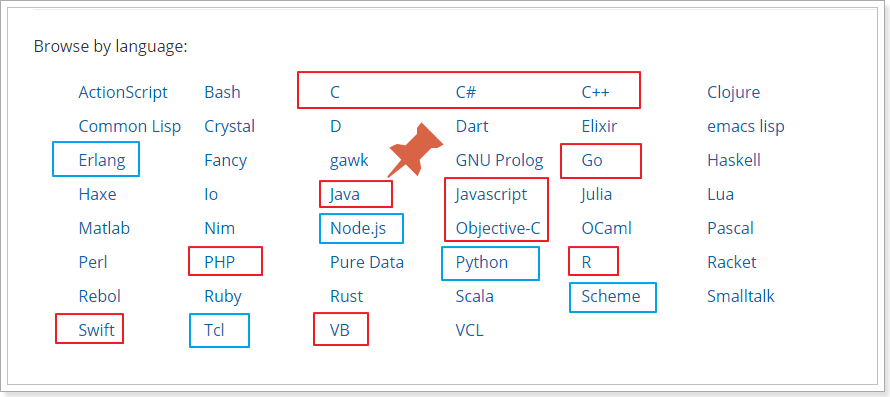


## 设置单位为毫秒

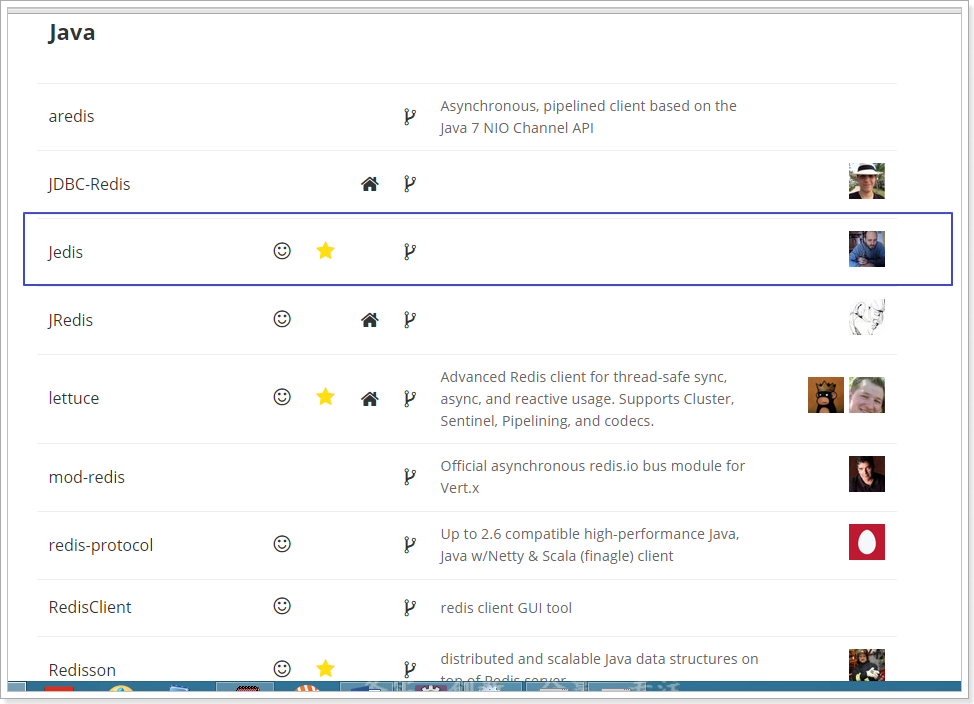


# 客户端

## 支持的语言

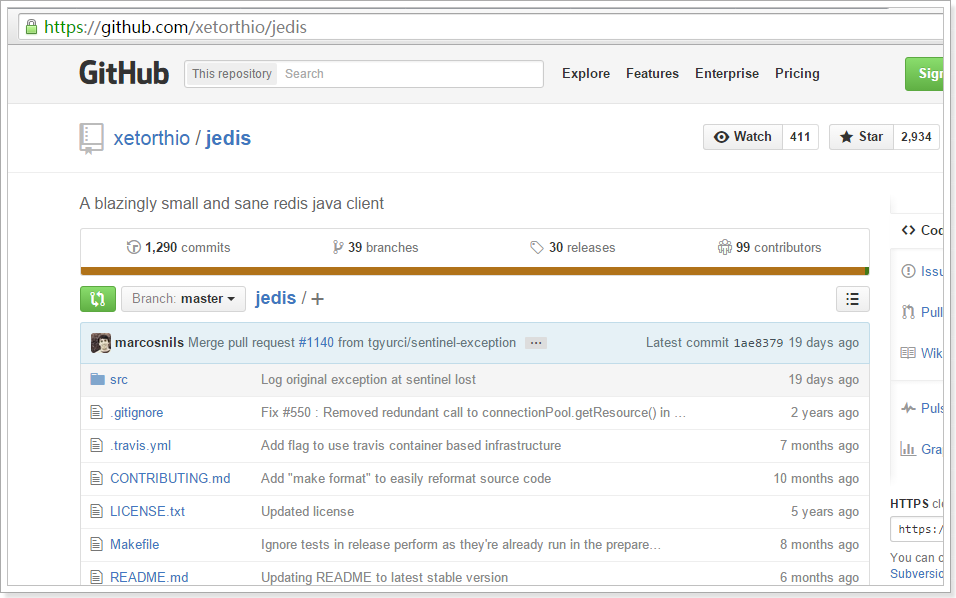


## Jedis



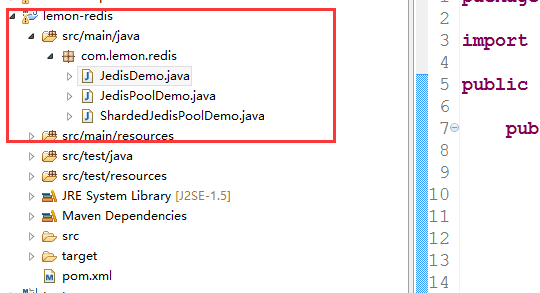


官网：

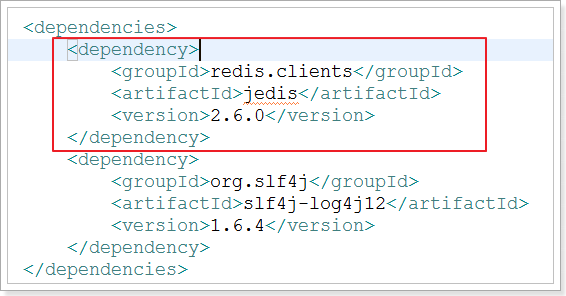


## Jedis的使用

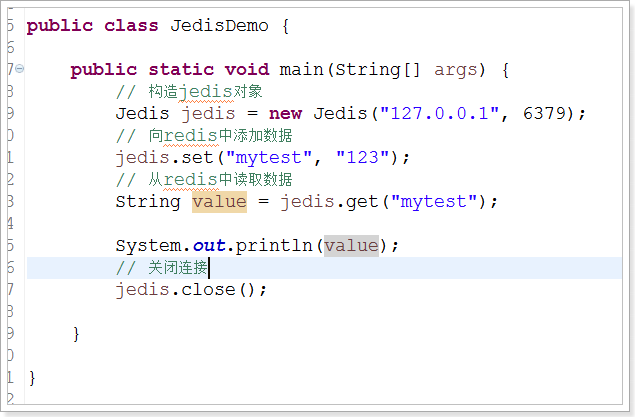
### 导入lemon-redis



### 导入依赖



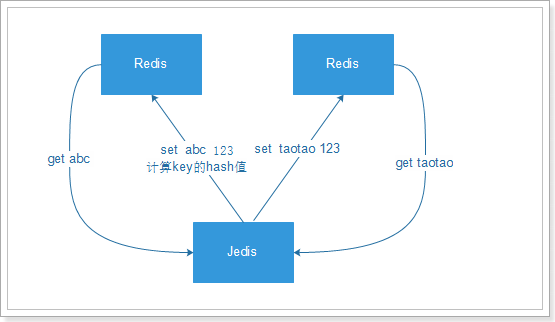
### 简单示例



### 连接池使用



### 分片式集群



存在的问题：无法动态增加减少服务节点。

### 分片式集群的使用

**public** **class** ShardedJedisPoolDemo {

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

// 构建连接池配置信息

JedisPoolConfig poolConfig = **new** JedisPoolConfig();

// 设置最大连接数

poolConfig.setMaxTotal(50);

// 定义集群信息

List<JedisShardInfo> shards = **new** ArrayList<JedisShardInfo>();

shards.add(**new** JedisShardInfo("127.0.0.1", 6379));

shards.add(**new** JedisShardInfo("192.168.29.112", 6379));

// 定义集群连接池

ShardedJedisPool shardedJedisPool = **new** ShardedJedisPool(poolConfig, shards);

ShardedJedis shardedJedis = **null**;

**try** {

// 从连接池中获取到jedis分片对象

shardedJedis = shardedJedisPool.getResource();

// for (int i = 0; i < 100; i++) {

// shardedJedis.set("key\_" + i, "value\_" + i);

// }

System.***out***.println(shardedJedis.get("key\_49"));

System.***out***.println(shardedJedis.get("key\_7"));

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (**null** != shardedJedis) {

// 关闭，检测连接是否有效，有效则放回到连接池中，无效则重置状态

shardedJedis.close();

}

}

// 关闭连接池

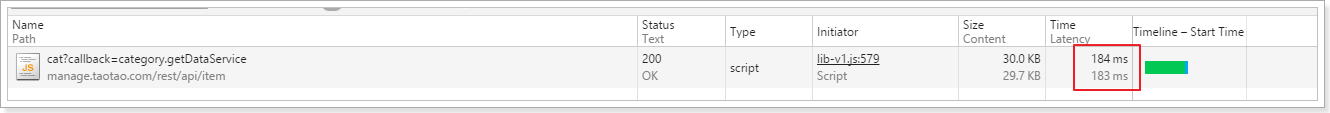
shardedJedisPool.close();

}

}

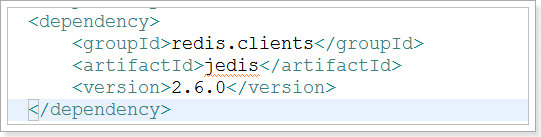
# 将Jedis集成到项目中

## 添加缓存前的测试



## 后台系统中添加缓存

### 在Service中添加依赖



### Spring和Jedis的整合



### 封装RedisService

**package** com.taotao.manage.service;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** redis.clients.jedis.ShardedJedis;

**import** redis.clients.jedis.ShardedJedisPool;

@Service

**public** **class** RedisService {

@Autowired

**private** ShardedJedisPool shardedJedisPool;

/\*\*

\* 执行set操作

\*

\* **@param** key

\* **@param** value

\* **@return**

\*/

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

ShardedJedis shardedJedis = **null**;

**try** {

// 从连接池中获取到jedis分片对象

shardedJedis = shardedJedisPool.getResource();

**return** shardedJedis.set(key, value);

} **finally** {

**if** (**null** != shardedJedis) {

// 关闭，检测连接是否有效，有效则放回到连接池中，无效则重置状态

shardedJedis.close();

}

}

}

/\*\*

\* 执行get操作

\*

\* **@param** key

\* **@return**

\*/

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

ShardedJedis shardedJedis = **null**;

**try** {

// 从连接池中获取到jedis分片对象

shardedJedis = shardedJedisPool.getResource();

**return** shardedJedis.get(key);

} **finally** {

**if** (**null** != shardedJedis) {

// 关闭，检测连接是否有效，有效则放回到连接池中，无效则重置状态

shardedJedis.close();

}

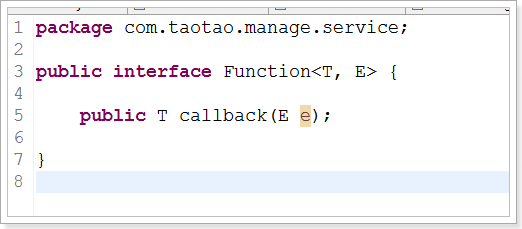
}

}

}

### 优化RedisService

定义接口：



RedisService的实现：

**package** com.taotao.manage.service;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** redis.clients.jedis.ShardedJedis;

**import** redis.clients.jedis.ShardedJedisPool;

@Service

**public** **class** RedisService {

@Autowired

**private** ShardedJedisPool shardedJedisPool;

**private** <T> T execute(Function<T, ShardedJedis> fun) {

ShardedJedis shardedJedis = **null**;

**try** {

// 从连接池中获取到jedis分片对象

shardedJedis = shardedJedisPool.getResource();

**return** fun.callback(shardedJedis);

} **finally** {

**if** (**null** != shardedJedis) {

// 关闭，检测连接是否有效，有效则放回到连接池中，无效则重置状态

shardedJedis.close();

}

}

}

/\*\*

\* 执行set操作

\*

\* **@param** key

\* **@param** value

\* **@return**

\*/

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

**return** **this**.execute(**new** Function<String, ShardedJedis>() {

@Override

**public** String callback(ShardedJedis e) {

**return** e.set(key, value);

}

});

}

/\*\*

\* 执行get操作

\*

\* **@param** key

\* **@return**

\*/

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

**return** **this**.execute(**new** Function<String, ShardedJedis>() {

@Override

**public** String callback(ShardedJedis e) {

**return** e.get(key);

}

});

}

/\*\*

\* 执行删除操作

\*

\* **@param** key

\* **@return**

\*/

**public** Long del(**final** String key) {

**return** **this**.execute(**new** Function<Long, ShardedJedis>() {

@Override

**public** Long callback(ShardedJedis e) {

**return** e.del(key);

}

});

}

/\*\*

\* 设置生存时间，单位为：秒

\*

\* **@param** key

\* **@param** seconds

\* **@return**

\*/

**public** Long expire(**final** String key, **final** Integer seconds) {

**return** **this**.execute(**new** Function<Long, ShardedJedis>() {

@Override

**public** Long callback(ShardedJedis e) {

**return** e.expire(key, seconds);

}

});

}

/\*\*

\* 执行set操作并且设置生存时间，单位为：秒

\*

\* **@param** key

\* **@param** value

\* **@return**

\*/

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

**return** **this**.execute(**new** Function<String, ShardedJedis>() {

@Override

**public** String callback(ShardedJedis e) {

String str = e.set(key, value);

e.expire(key, seconds);

**return** str;

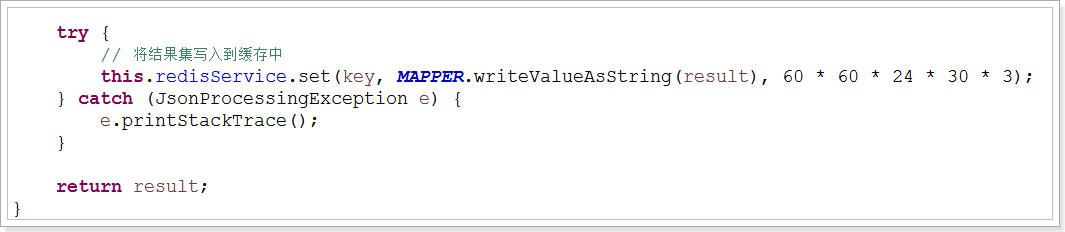
}

});

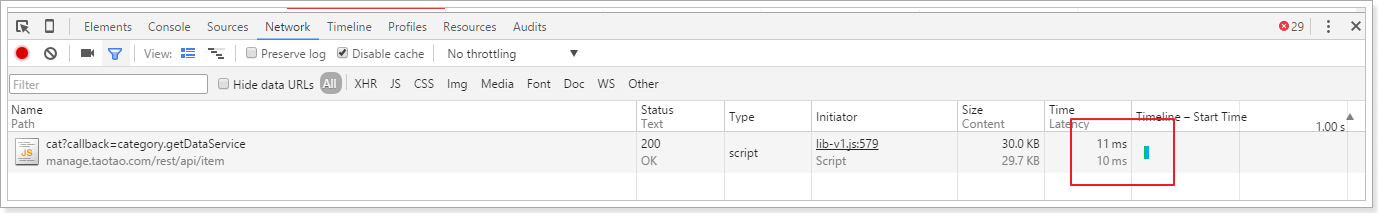
}

}

### 实现缓存逻辑

1. 先从缓存中命中，命中则返回  
   
2. 将结果写入到Redis中  
   

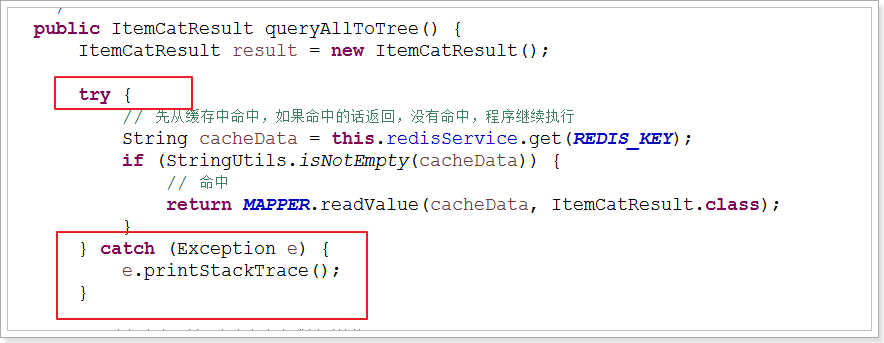
### 测试

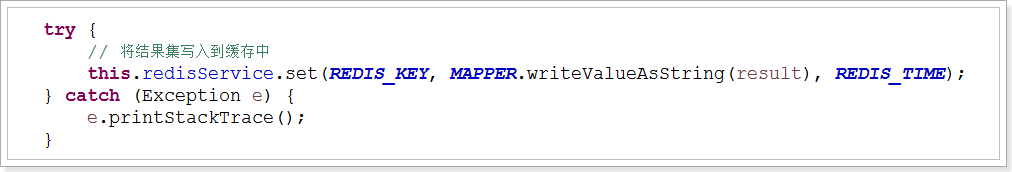


### 原则

原则：缓存逻辑不能影响原有的业务逻辑执行。

### 优化之后的缓存实现





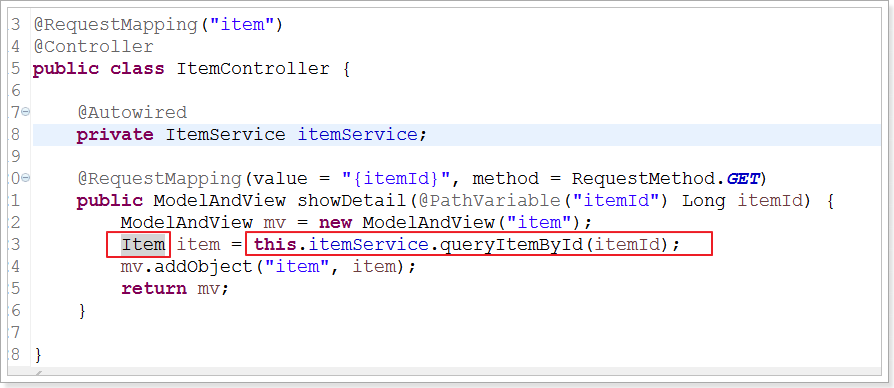
# 商品详情页

## 进入商品详情页地址

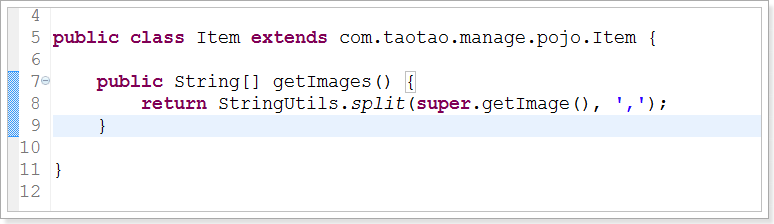
[http://www.taotao.com/item/{itemId}.html](http://www.taotao.com/item/%7bitemId%7d.html)

## 商品基本数据显示

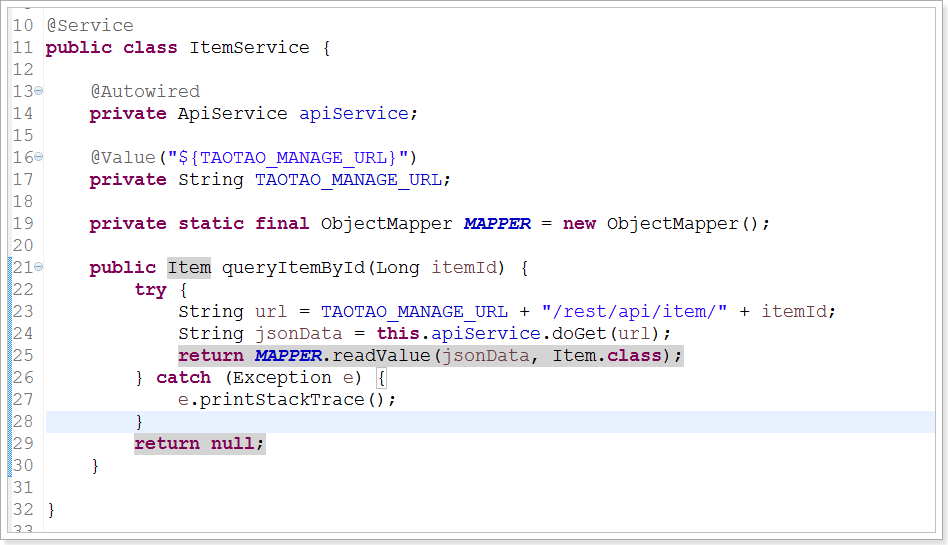
### Controller



### Item对象



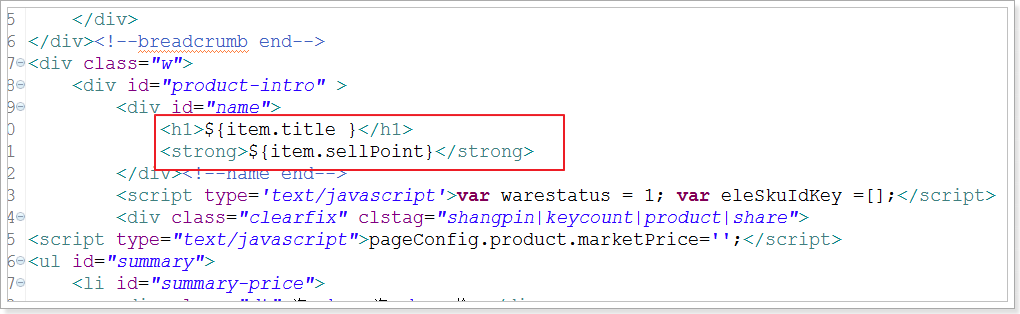
### Service



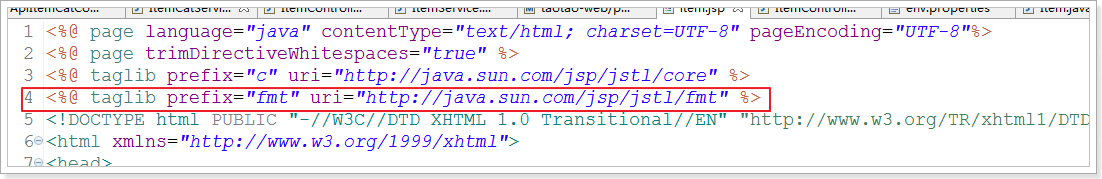
### 后台系统提供接口服务



### Item.jsp中使用模型数据



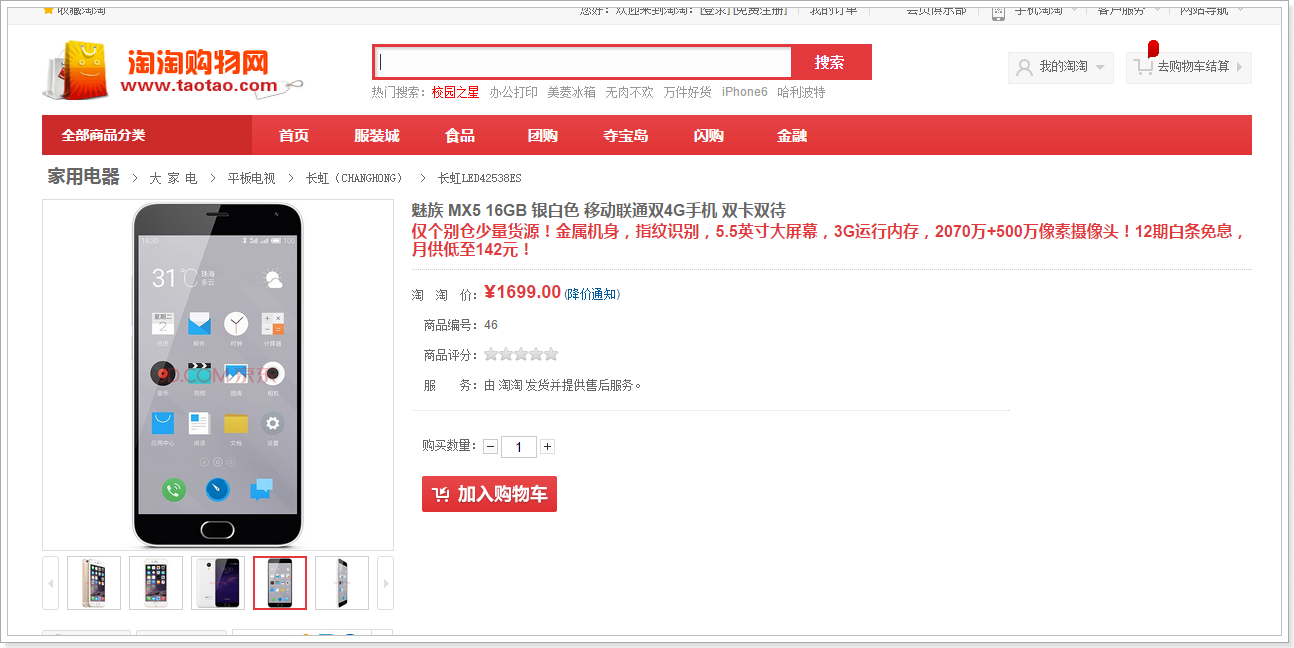
显示商品价格：



<fmt:formatNumber groupingUsed="false" maxFractionDigits="2" minFractionDigits="2" value="${item.price / 100 }"/>



### 测试

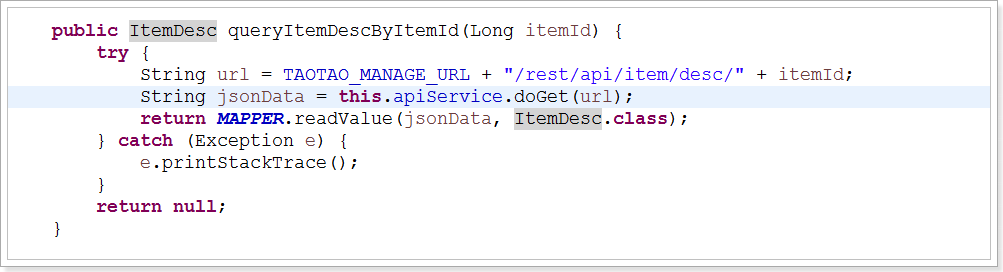


## 显示商品描述数据

### Controller



### Service



### 后台系统中开放接口服务



### Item.jsp中显示数据



### 效果



## 显示规格参数数据

### Controller



### Service



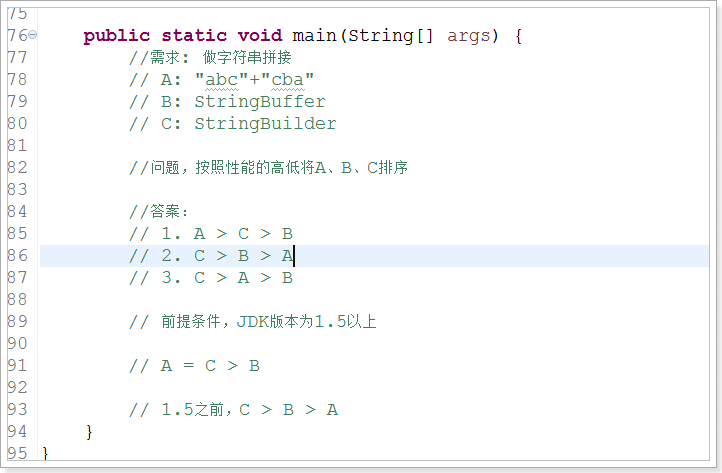
### Item.jsp



### 效果



## 字符串拼接面试题



# 商品详情页添加缓存

## 为什么要添加缓存？

加快显示商品 详情页的速度。

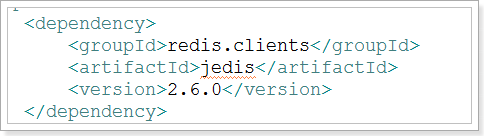
## 在哪里添加缓存？

前台系统？ 还是 后台系统？ --- 都加。

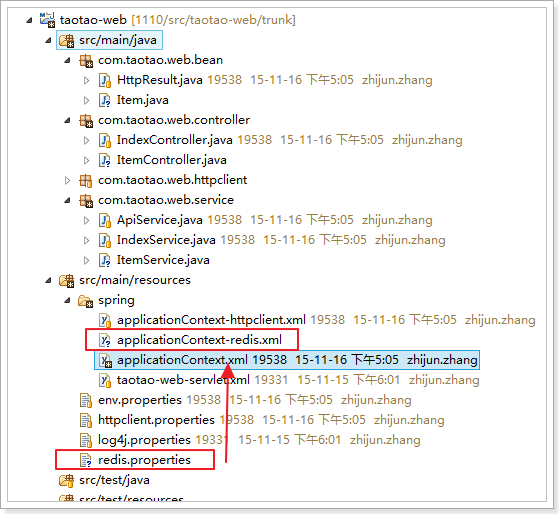
每个团队都应该为自己的系统功能负责任，保证其性能。

## 前台系统添加缓存

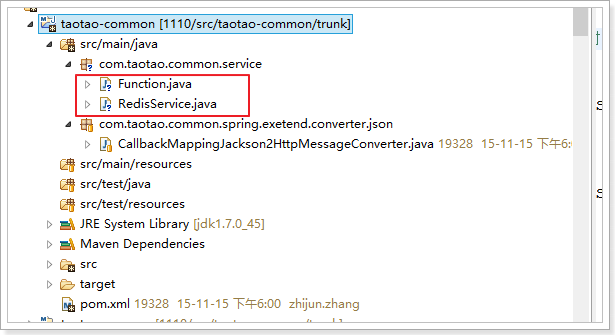
### 导入依赖



### 导入Spring和Jedis的整合文件



### 将RedisService和Function接口移动至taotao-common





### Service中添加缓存逻辑

**private** **static** **final** String ***REDIS\_KEY*** = "TAOTAO\_WEB\_ITEM\_DETAIL\_";

**private** **static** **final** Integer ***REDIS\_TIME*** = 60 \* 60 \* 24;

**public** Item queryItemById(Long itemId) {

// 从缓存中命中

String key = ***REDIS\_KEY*** + itemId;

**try** {

String cacheData = **this**.redisService.get(key);

**if** (StringUtils.*isNotEmpty*(cacheData)) {

**return** ***MAPPER***.readValue(cacheData, Item.**class**);

}

} **catch** (Exception e1) {

e1.printStackTrace();

}

**try** {

String url = TAOTAO\_MANAGE\_URL + "/rest/item/" + itemId;

String jsonData = **this**.apiService.doGet(url);

**if** (StringUtils.*isEmpty*(jsonData)) {

**return** **null**;

}

**try** {

// 将数据写入到缓存中

**this**.redisService.set(key, jsonData, ***REDIS\_TIME***);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** ***MAPPER***.readValue(jsonData, Item.**class**);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **null**;

}

# 商品数据同步问题（TODO）

## 问题

后台系统中将商品修改，前台系统没有进行数据的同步，导致前端系统不能够实时显示最新的数据。

## 解决

后台系统中商品修改后向其他系统发送通知，其他系统做出对应的处理即可。

## 怎么通知？

1. 在前台系统中开发一个接口  
   
2. 在后台系统中调用该接口  
   

## ApiService移动至taotao-common



## 实现数据同步存在的问题

通知的实现，代码的耦合度太高了。

怎么解决？ -- 消息队列。