谷粒商城

版本：V 1.0

# 项目整合redis

## 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，所以性能需要优化。

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。

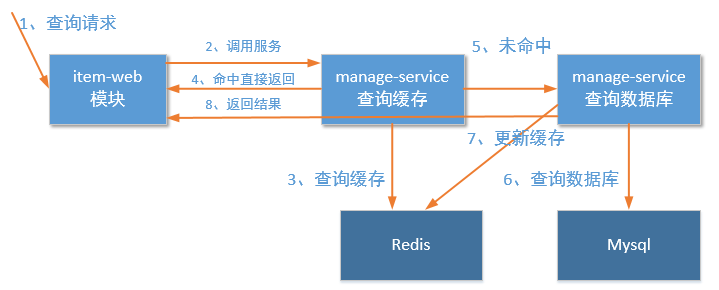
重点要讲的是另外一个层面：尽量避免直接查询数据库。

解决办法就是：缓存

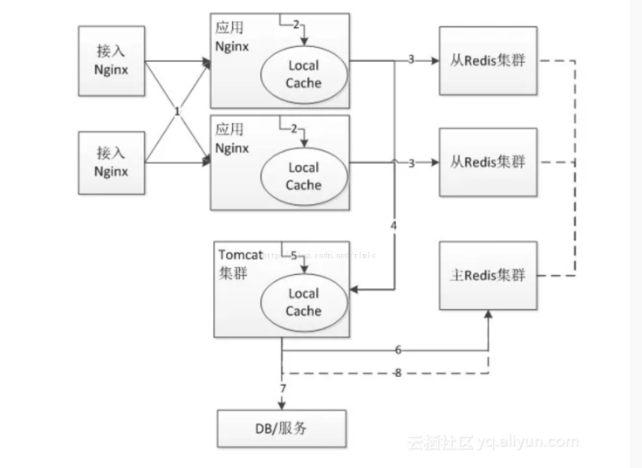
缓存可以理解是数据库的一道保护伞，任何请求只要能在缓存中命中，都不会直接访问数据库。而缓存的处理性能是数据库10-100倍。

咱们就用Redis作为缓存系统进行优化。

结构图：



多级缓存应用：



安装Redis ：

1. 安装依赖
2. 导入jar包
3. 创建安装目录
4. Make
5. Make install
6. 修改配置文件

## 2 整合redis到大工程中。

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。

Spring boot 项目 推荐使用注解方式来完成配置。

RedisUtil

|  |
| --- |
| **package** com.atguigu.gmall.config;  **public class** RedisUtil {   **private** JedisPool **jedisPool**;   **public void** initJedisPool(String host,**int** port,**int** database){  JedisPoolConfig jedisPoolConfig = **new** JedisPoolConfig();  *// 总数* jedisPoolConfig.setMaxTotal(200);  *// 获取连接时等待的最大毫秒* jedisPoolConfig.setMaxWaitMillis(10\*1000);  *// 最少剩余数* jedisPoolConfig.setMinIdle(10);  *// 如果到最大数，设置等待* jedisPoolConfig.setBlockWhenExhausted(**true**);  *// 在获取连接时，检查是否有效* jedisPoolConfig.setTestOnBorrow(**true**);  *// 创建连接池* **jedisPool** = **new** JedisPool(jedisPoolConfig,host,port,20\*1000);  }  **public** Jedis getJedis(){  Jedis jedis = **jedisPool**.getResource();  **return** jedis;  } } |

RedisConfig

@Configuration 相当于spring3.0版本的xml

|  |
| --- |
| @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**=**192.168.67.204 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.set(**"test"**,**"text\_value"** ); }**catch** (JedisConnectionException e){  e.printStackTrace(); } |
| **在启动类上加上@ComonentScan注解** |

## 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:info

:表示根据windows的 /一个意思

重构getSkuInfo方法

在gmall-manage-service项目中定义常量

|  |
| --- |
| **public static final** String ***SKUKEY\_PREFIX***=**"sku:"**;   **public static final** String ***SKUKEY\_SUFFIX***=**":info"**;   **public static final int *SKUKEY\_TIMEOUT***=24\*60\*60; |

以上基本实现使用缓存的方案。

如果说：redis服务没有启动，宕机！如何解决？

Try-catch 获取mysql数据返回

如果说：访问redis的时候，可能会有产生高并发，如何解决这种高并发访问？

使用分布式锁！

set test ok px 10000 nx

## 4 解决缓存击穿问题：

Redis:命令

# set sku:1:info “OK” NX PX 10000

EX second ：设置键的过期时间为 second 秒。 SET key value EX second 效果等同于 SETEX key second value 。

PX millisecond ：设置键的过期时间为 millisecond 毫秒。 SET key value PX millisecond 效果等同于 PSETEX key millisecond value 。

NX ：只在键不存在时，才对键进行设置操作。 SET key value NX 效果等同于 SETNX key value 。

XX ：只在键已经存在时，才对键进行设置操作。

|  |
| --- |
| @Override  **public** SkuInfo getSkuInfo\_redis(String skuId) {   SkuInfo skuInfoResult=**null**;  *//1 先查redis 没有再查数据库* Jedis jedis = **redisUtil**.getJedis();  **int** SKU\_EXPIRE\_SEC=100;  *// redis结构 ： 1 type string 2 key sku:101:info 3 value skuInfoJson* String skuKey=***SKUKEY\_PREFIX***+skuId+***SKUKEY\_INFO\_SUFFIX***;  String skuInfoJson = jedis.get(skuKey);  **if**(skuInfoJson!=**null**){  **if**(!**"EMPTY"**.equals(skuInfoJson)){  System.***out***.println(Thread.*currentThread*()+**"命中缓存！！"**);  skuInfoResult = JSON.*parseObject*(skuInfoJson, SkuInfo.**class**);  }   }**else**{  System.***out***.println(Thread.*currentThread*()+**"未命中！！"**);  *//setnx 1 查锁 exists 2 抢锁 set  //定义一下 锁的结构 type string key sku:101:lock value locked* String lockKey=***SKUKEY\_PREFIX***+skuId+***SKUKEY\_LOCK\_SUFFIX***; *// Long locked = jedis.setnx(lockKey, "locked"); // jedis.expire(lockKey,10);* String token=UUID.*randomUUID*().toString();  String locked = jedis.set(lockKey, token, **"NX"**, **"EX"**, 100);   **if**(**"OK"**.equals(locked)){  System.***out***.println(Thread.*currentThread*()+**"得到锁！！"**);   skuInfoResult = getSkuInfoDB(skuId);   System.***out***.println(Thread.*currentThread*()+**"写入缓存！！"**);  String skuInfoJsonResult=**null**;  **if**(skuInfoResult!=**null**){  skuInfoJsonResult = JSON.*toJSONString*(skuInfoResult);  }**else**{  skuInfoJsonResult=**"EMPTY"**;  }  jedis.setex(skuKey,SKU\_EXPIRE\_SEC,skuInfoJsonResult);  System.***out***.println(Thread.*currentThread*()+**"释放锁！！"**+lockKey);  **if**(jedis.exists(lockKey)&&token.equals(jedis.get(lockKey))){ *// 不完美 ，可以用lua解决* jedis.del(lockKey);  }   }**else**{  System.***out***.println(Thread.*currentThread*()+**"为得到锁，开始自旋等待！！"**);  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  getSkuInfo( skuId);  }   }   jedis.close();  **return** skuInfoResult;   } |

## 5 redisson 解决分布式锁

|  |
| --- |
| 1. 导入依赖 service-util   <dependency>  <groupId>org.redisson</groupId>  <artifactId>redisson</artifactId>  <version>3.11.1</version>  </dependency> |
| 1. 修改实现类   @Override **public** SkuInfo getSkuInfo(String skuId) {  SkuInfo skuInfoResult=**null**;  *//1 先查redis 没有再查数据库* Jedis jedis = **redisUtil**.getJedis();  **int** SKU\_EXPIRE\_SEC=100;  *// redis结构 ： 1 type string 2 key sku:101:info 3 value skuInfoJson* String skuKey=***SKUKEY\_PREFIX***+skuId+***SKUKEY\_INFO\_SUFFIX***;  String skuInfoJson = jedis.get(skuKey);  **if**(skuInfoJson!=**null**){  **if**(!**"EMPTY"**.equals(skuInfoJson)){  System.***out***.println(Thread.*currentThread*()+**"命中缓存！！"**);  skuInfoResult = JSON.*parseObject*(skuInfoJson, SkuInfo.**class**);  }  }**else**{  Config config = **new** Config();  config.useSingleServer().setAddress(**"redis://redis.gmall.com:6379"**);  RedissonClient redissonClient = Redisson.*create*(config);  String lockKey=***SKUKEY\_PREFIX***+skuId+***SKUKEY\_LOCK\_SUFFIX***;  RLock lock = redissonClient.getLock(lockKey);  *// lock.lock(10,TimeUnit.SECONDS);* **boolean** locked=**false** ;  **try** {  locked = lock.tryLock(10, 5, TimeUnit.***SECONDS***);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  **if**(locked) {  System.***out***.println(Thread.*currentThread*() + **"得到锁！！"**);  *// 如果得到锁后能够在缓存中查询 ，那么直接使用缓存数据 不用在查询数据库* System.***out***.println(Thread.*currentThread*()+**"再次查询缓存！！"**);  String skuInfoJsonResult = jedis.get(skuKey);  **if** (skuInfoJsonResult != **null**) {  **if** (!**"EMPTY"**.equals(skuInfoJsonResult)) {  System.***out***.println(Thread.*currentThread*() + **"命中缓存！！"**);  skuInfoResult = JSON.*parseObject*(skuInfoJsonResult, SkuInfo.**class**);  }   } **else** {  skuInfoResult = getSkuInfoDB(skuId);   System.***out***.println(Thread.*currentThread*() + **"写入缓存！！"**);   **if** (skuInfoResult != **null**) {  skuInfoJsonResult = JSON.*toJSONString*(skuInfoResult);  } **else** {  skuInfoJsonResult = **"EMPTY"**;  }  jedis.setex(skuKey, SKU\_EXPIRE\_SEC, skuInfoJsonResult);  }  lock.unlock();  }   }  **return** skuInfoResult; } |
|  |