# 数据库优化

## 优化策略

### 实操中如何优化数据库

1. 优化sql语句.

原则:

A:能用单表不用关联查询. 正常的数据库设计需要满足三范式要求

为了实现业务模块查询效率高,一般在企业中使用反三范式的设计.

B:使用关联时尽可能提早确定数据. 否则会进行大量的笛卡尔积操作.直接影响程序执行性能

C:尽可能根据主键查询.

SELECT t.id,t.title,d.item\_desc FROM

(SELECT \* FROM tb\_item WHERE id = 536563)t

LEFT JOIN

tb\_item\_desc d

ON t.id = d.item\_id

1. 创建索引

对于经常查询的数据,为其创建索引文件

作业:自己了解索引的创建方式

1. 添加缓存

为经常查询的数据,将数据添加到缓存中,提高查询效率.

主流的缓存有 mamchche,redis,Map

1. 定期转储数据

采用历史表策略 将旧的数据保存到历史表中.当前表中只维护现在使用的数据.

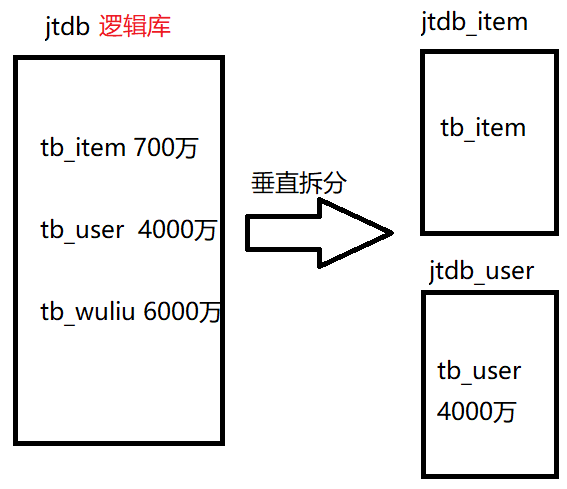
1. **分库分表**

因为数据库瓶颈问题,无法以现在的优化策略,优化数据库,这时需要将数据库进行分库分表操作.也就是将一个数据库动态的拆分为多个数据库.

## 数据库分库分表

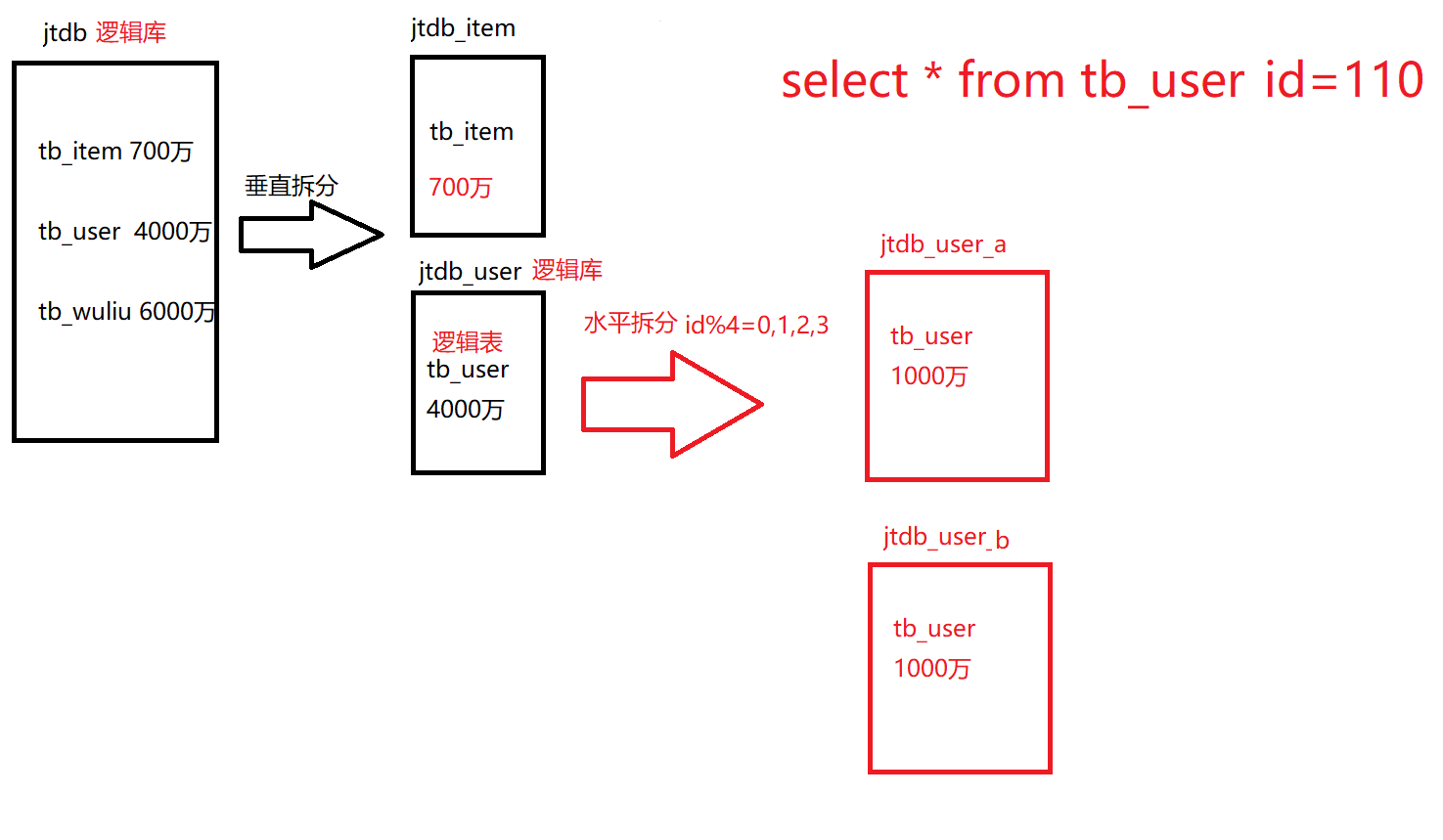
### 数据库垂直拆分

说明:根据不同的数据库,将有关联关系的数据表拆分到一个数据库中,将没有关联关系的数据表拆分到不同的数据库中.(根据业务拆分)



### 数据库水平拆分

说明:将数据库中的表数据拆分到多个数据库中得表中



Mycat配置详情,参数配置文件的资料.

# 缓存机制

## 缓存服务配置

### 缓存机制

缓存实际上就是数据库数据的备份.只要的作用就是降低了用户对物理设备的访问的频次.同时缓存数据都保存到内存中.用户查询时如果缓存中有数据则直接返回,如果缓存中没有数据则查询数据库之后保存到缓存中.方便后续用户查询.

### 设计缓存考虑哪些因素

1. 缓存数据采用哪种数据类型存储??? K-v结构
2. 缓存数据如何与数据库数据保持一致?? 需要通过程序控制
3. 缓存如何维护内存空间大小??? Lru算法(查询最近最少使用的数据进行删除操作)
4. 缓存中的数据如何保证不丢失. 定期序列化到磁盘中.

## Redis

### Redis介绍

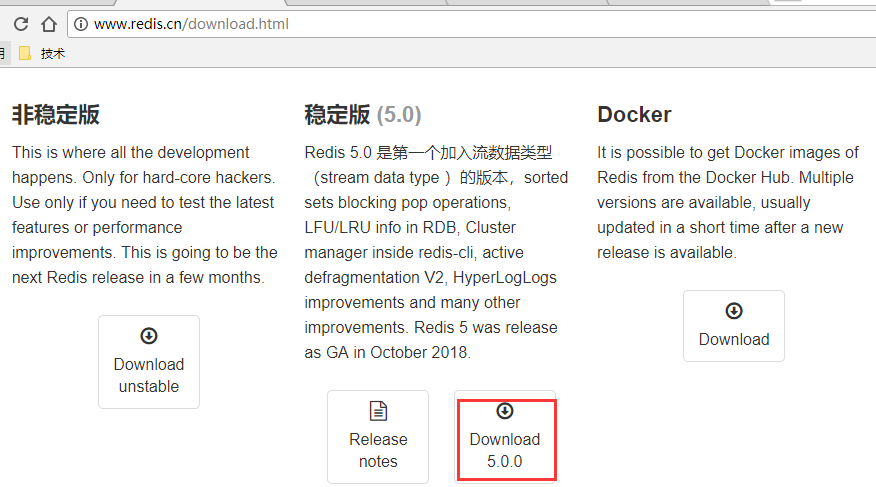
redis是一个**key-value**[**存储系统**](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E7%B3%BB%E7%BB%9F)。和Memcached类似，它支持存储的value类型相对更多，**包括string(字符串)、list(**[**链表**](https://baike.baidu.com/item/%E9%93%BE%E8%A1%A8)**)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。**这些[数据类型](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B)都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是**原子性**的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

用途:**redis可以当做数据库/缓存/消息队列(MQ)**

## Redis安装

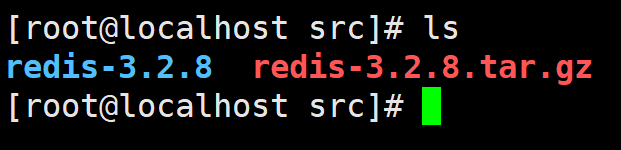
### Redis下载

Redis下载Linux版本.



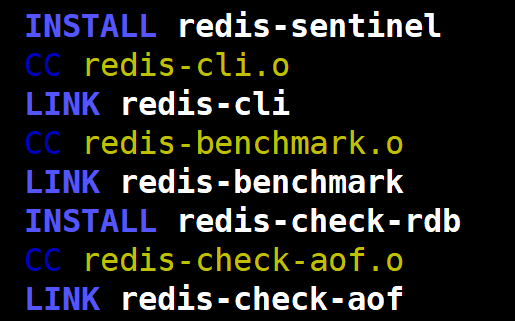
### 安装Redis

1. 解压文件



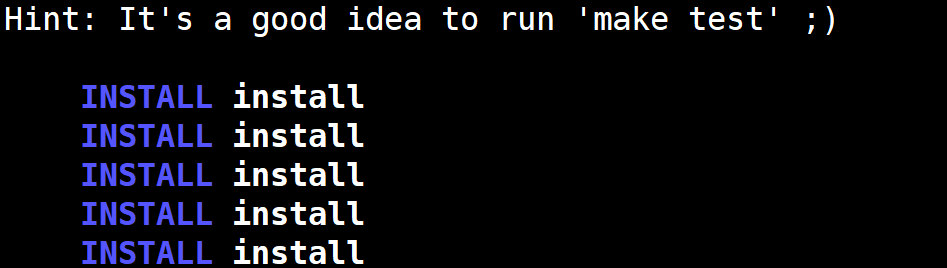
1. 编译redis

Make



安装:

make install



### Redis启动/关闭/客户端

1. Redis启动

redis-server

redis-server redis.conf

1. 关闭redis

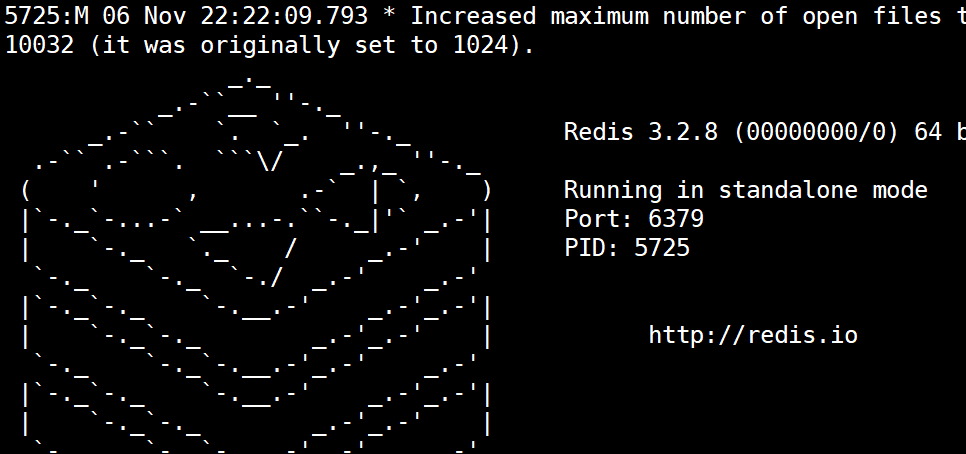
Kill -9 PID

redis-cli shutdown 该命令 只能关闭端口号为6379

redis-cli -p 6379 shutdown

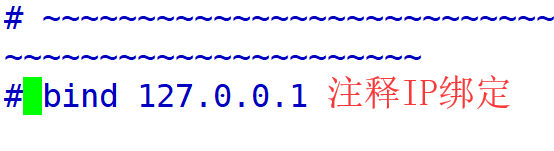
3.redis客户端进入

redis-cli -p 6379

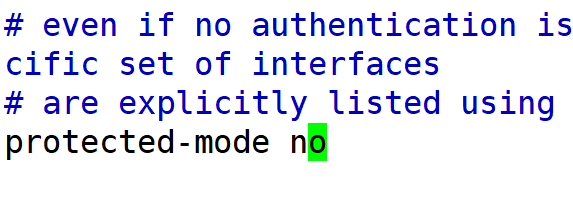


### 修改redis配置文件

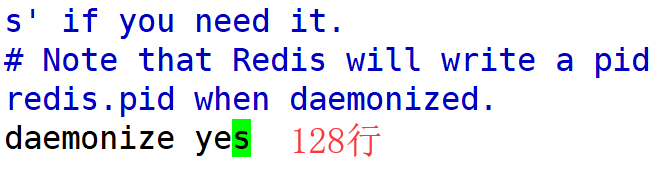
修改IP绑定:



关闭保护模式



开启后台启动



## Redis命令

### String类型

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| set | 添加key-value | set username admin |
| get | 根据key获取数据 | get username |
| strlen | 获取key的长度 | strlen key |
| exists | 判断key是否存在 | exists name  返回1存在 0不存在 |
| del | 删除redis中的key | del key |
| Keys | 用于查询符合条件的key | keys \* 查询redis中全部的key  keys n?me 使用占位符获取数据  keys nam\* 获取nam开头的数据 |
| mset | 赋值多个key-value | mset key1 value1 key2 value2 key3 value3 |
| mget | 获取多个key的值 | mget key1 key2 |
| append | 对某个key的值进行追加 | append key value |
| type | 检查某个key的类型 | type key |
| select | 切换redis数据库 | select 0-15 redis中共有16个数据库 |
| flushdb | 清空单个数据库 | flushdb |
| flushall | 清空全部数据库 | flushall |
| incr | 自动加1 | incr key |
| decr | 自动减1 | decr key |
| incrby | 指定数值添加 | incrby 10 |
| decrby | 指定数值减 | decrby 10 |
| expire | 指定key的生效时间 单位秒 | expire key 20  key20秒后失效 |
| pexpire | 指定key的失效时间 单位毫秒 | pexpire key 2000  key 2000毫秒后失效 |
| ttl | 检查key的剩余存活时间 | ttl key |
| persist | 撤销key的失效时间 | persist key |

### Hash类型

说明:可以用散列类型保存对象和属性值

例子:User对象{id:2,name:小明,age:19}

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| hset | 为对象添加数据 | hset key field value |
| hget | 获取对象的属性值 | hget key field |
| hexists | 判断对象的属性是否存在 | HEXISTS key field  1表示存在 0表示不存在 |
| hdel | 删除hash中的属性 | hdel user field [field ...] |
| hgetall | 获取hash全部元素和值 | HGETALL key |
| hkyes | 获取hash中的所有字段 | HKEYS key |
| hlen | 获取hash中所有属性的数量 | hlen key |
| hmget | 获取hash里面指定字段的值 | hmget key field [field ...] |
| hmset | 为hash的多个字段设定值 | hmset key field value [field value ...] |
| hsetnx | 设置hash的一个字段,只有当这个字段不存在时有效 | HSETNX key field value |
| hstrlen | 获取hash中指定key的长度 | HSTRLEN key field |
| hvals | 获取hash的所有值 | HVALS user |

### List类型

说明:Redis中的List集合是双端循环列表,分别可以从左右两个方向插入数据.

List集合可以当做**队列**使用,也可以当做**栈**使用

队列:存入数据的方向和获取数据的**方向相反**

栈:存入数据的方向和获取数据的**方向相同**

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| lpush | 从队列的左边入队一个或多个元素 | LPUSH key value [value ...] |
| rpush | 从队列的右边入队一个或多个元素 | RPUSH key value [value ...] |
| lpop | 从队列的左端出队一个元素 | LPOP key |
| rpop | 从队列的右端出队一个元素 | RPOP key |
| lpushx | 当队列存在时从队列的左侧入队一个元素 | LPUSHX key value |
| rpushx | 当队列存在时从队列的右侧入队一个元素 | RPUSHx key value |
| lrange | 从列表中获取指定返回的元素 | LRANGE key start stop  Lrange key 0 -1 获取全部队列的数据 |
| lrem | 从存于 key 的列表里移除前 count 次出现的值为 value 的元素。 这个 count 参数通过下面几种方式影响这个操作：   * count > 0: 从头往尾移除值为 value 的元素。 * count < 0: 从尾往头移除值为 value 的元素。 * count = 0: 移除所有值为 value 的元素。 | LREM list -2 “hello” 会从存于 list 的列表里移除最后两个出现的 “hello”。  需要注意的是，如果list里没有存在key就会被当作空list处理，所以当 key 不存在的时候，这个命令会返回 0。 |
| Lset | 设置 index 位置的list元素的值为 value | LSET key index value |

### Redis事务命令

说明:redis中操作可以添加事务的支持.一项任务可以由多个redis命令完成,如果有一个命令失败导致入库失败时.需要实现事务回滚.

|  |  |  |
| --- | --- | --- |
| 命令 | 说明 | 案例 |
| multi | 标记一个事务开始 | 127.0.0.1:6379> MULTI  OK |
| exec | 执行所有multi之后发的命令 | 127.0.0.1:6379> EXEC  OK |
| discard | 丢弃所有multi之后发的命令 |  |

### Redis优点

作用:

1. 使用缓存可以有效的减少数据的访问次数.保护数据库.
2. 快速的获取用户的数据.
3. Redis在大型项目经常使用. 例如消息队列

优点:

1.因为redis底层实现C语言,工作在内存中,异常快速

存:8万/秒 取:12万/秒

2.Redis内部采用LRU算法实现内存自动调优.

3.Redis支持多种数据类型

4.Redis内部支持数据持久化(aof/rdb)

## Redis入门案例

### 引入jar包

<!-- jedis -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>${jedis.version}</version>

</dependency>

<!--添加spring-datajar包 -->

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

<version>1.4.1.RELEASE</version>

</dependency>

### 测试String类型

//测试字符串 IP:6379

@Test

**public** **void** testString(){

Jedis jedis = **new** Jedis("192.168.126.166",6379);

jedis.set("aa","1807班");

System.***out***.println(jedis.get("aa"));

}

### 测试hash数据

//测试hash数据存储

@Test

**public** **void** testHash(){

Jedis jedis = **new** Jedis("192.168.126.166",6379);

jedis.hset("dog", "id", "100");

jedis.hset("dog", "name", "中华田园犬");

jedis.hset("dog", "age", "88");

System.***out***.println(jedis.hget("dog", "name"));

System.***out***.println(jedis.hgetAll("dog"));

}

### RedisList类型

//测试List集合

@Test

**public** **void** testList(){

Jedis jedis = **new** Jedis("192.168.126.166",6379);

jedis.lpush("idList", "1","2","3");

System.***out***.println(jedis.lpop("idList"));

}

### 事务控制

//redis中事务控制

@Test

**public** **void** testTx(){

Jedis jedis = **new** Jedis("192.168.126.166",6379);

Transaction transaction = jedis.multi(); //开启事务.

transaction.set("aa", "aaa");

transaction.set("bb", "bbb");

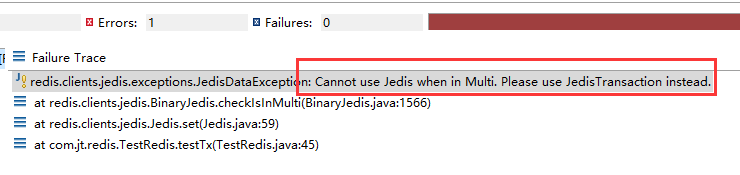
transaction.exec(); //事务提交

System.***out***.println("执行成功!!!");

//transaction.discard(); //事务回滚

}

使用事务控制时,应该使用transaction对象操作



### 关于缓存使用问题

1. 缓存击穿

当遇到大规模并发操作时,请求大量的访问缓存.但是缓存中的数据为空,或者命中率过低,这时用户的请求会直接访问真实的数据库,可能造成宕机的现象.

## ObjectMapper对象

### 说明

该对象支持将java对象转化为JSON串.同时支持JSON串转化为对象.

### 导入jar包

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>${jackson.version}</version>

</dependency>

### 将对象转化为JSON

//将对象转化为JSON数据

@Test

**public** **void** objectToJSON() **throws** IOException{

User user = **new** User();

user.setId(1);

user.setName("tom");

user.setAge(18);

ObjectMapper mapper = **new** ObjectMapper();

String json =

mapper.writeValueAsString(user);

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

//将json数据转化为对象

User u = mapper.readValue(json,User.**class**);

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

}

@Test

**public** **void** listToJSON() **throws** IOException{

User user = **new** User();

user.setId(1);

user.setName("tom");

user.setAge(18);

User user2 = **new** User();

user2.setId(1);

user2.setName("tom");

user2.setAge(18);

User user3 = **new** User();

user3.setId(1);

user3.setName("tom");

user3.setAge(18);

List<User> list = **new** ArrayList<>();

list.add(user);

list.add(user2);

list.add(user3);

ObjectMapper mapper = **new** ObjectMapper();

String listJSON =

mapper.writeValueAsString(list);

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

List<User> uList =

mapper.readValue(listJSON,list.getClass());

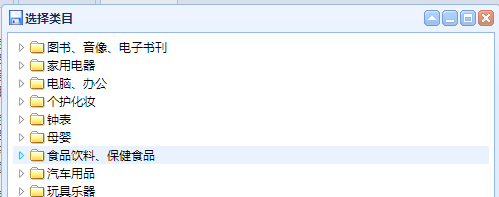
System.***out***.println(uList);

}

## 商品分类实现redis缓存

### 业务需求

将商品分类目录信息保存到redis中



实现步骤:

1. Spring管理jedis对象,哪里需要直接注入(IOC)
2. 用户查询时,先查询缓存中的数据.

有数据:

将从redis中获取的JSON串**转化**为对象,之后返回数据

没数据:

查询数据库,将数据插入缓存中,将集合转化为JSON数据, key:不重复即可.vlaue:ListJSON数据

### Spring整合Jedis对象

<!--spring实例化jedis

Jedis jedis = new Jedis("192.168.126.166",6379);

-->

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

<constructor-arg name=*"host"* value=*"192.168.126.166"*/>

<constructor-arg name=*"port"* value=*"6379"*/>

</bean>

### 编辑Controller

@RequestMapping("/cat/list")

@ResponseBody

**public** List<EasyUITree> findItemCatListById

(@RequestParam(value="id",defaultValue="0") Long parentId){

//查询一级商品分类标题

**return** itemCatService.findCacheItemCat(parentId);

}

### 编辑Service

@Override

**public** List<EasyUITree> findCacheItemCat(Long parentId) {

List<EasyUITree> treeList = **new** ArrayList<>();

//用户查询先查询缓存

String key = "ITEM\_CAT\_"+parentId;

String json = jedis.get(key);

**try** {

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

//表示缓存中没有数据

treeList = findItemCatListById(parentId);

//将数据保存到缓存中

String listJSON =

objectMapper.writeValueAsString(treeList);

//将数据保存到redis中

jedis.set(key, listJSON);

System.***out***.println("第一次查询数据库");

}**else**{

//表示缓存中有数据,直接转化为对象

treeList =

objectMapper.readValue(json, treeList.getClass());

System.***out***.println("缓存查询成功!!!");

}

} **catch** (Exception e) {

e.printStackTrace();

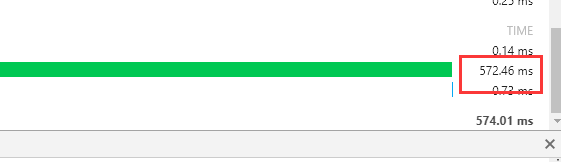
}

**return** treeList;

}

### 速度比较

不使用缓存:



使用redis缓存:

