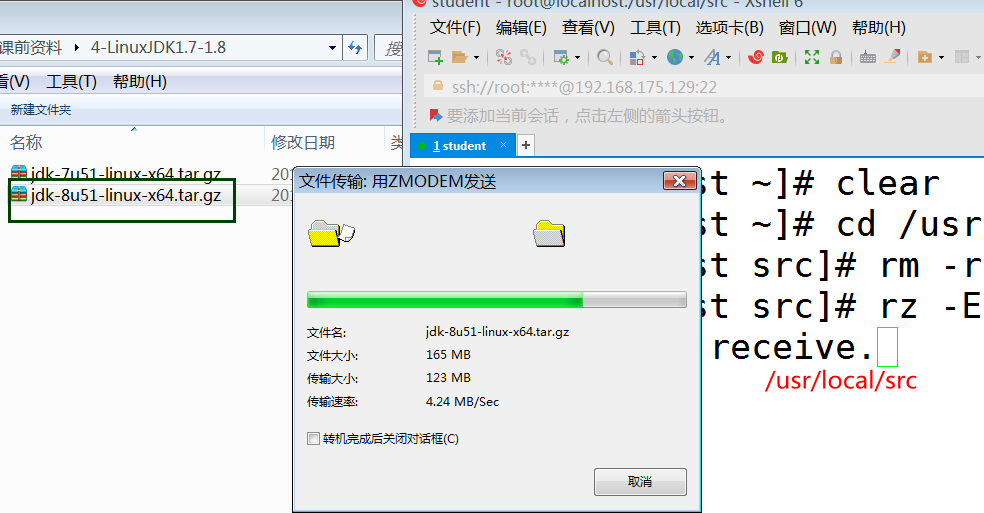
# 部署Linuxtomcat集群

## 安装JDK

### 安装JDK步骤

1. 准备JDK文件.
2. 配置环境变量.标识java\_host/path

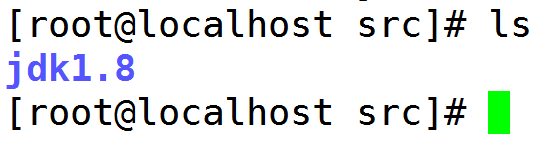
### 上传JDK安装包



### 解压JDK

tar -xvf jdk-8u51-linux-x64.tar.gz

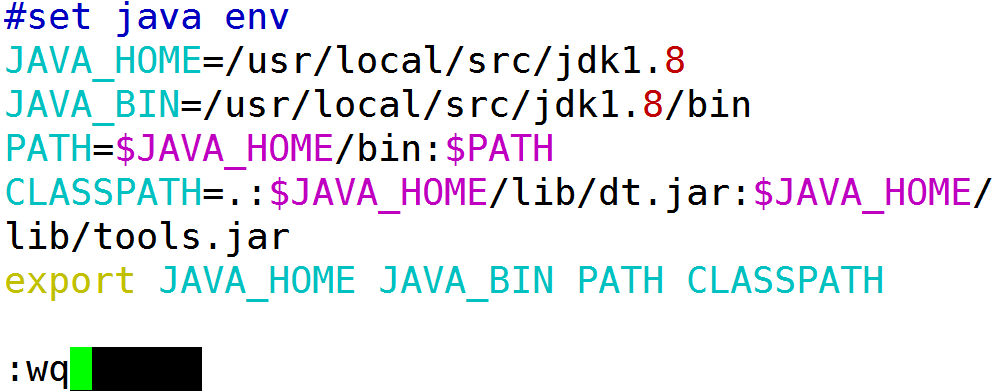
将原有的安装文件删除,并且将JDK文件名称修改



### 编辑环境变量

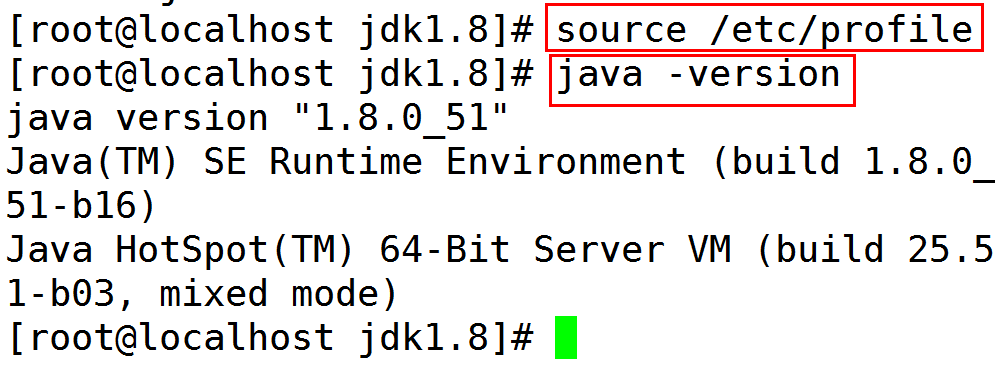
vim /etc/profile

编辑环境变量



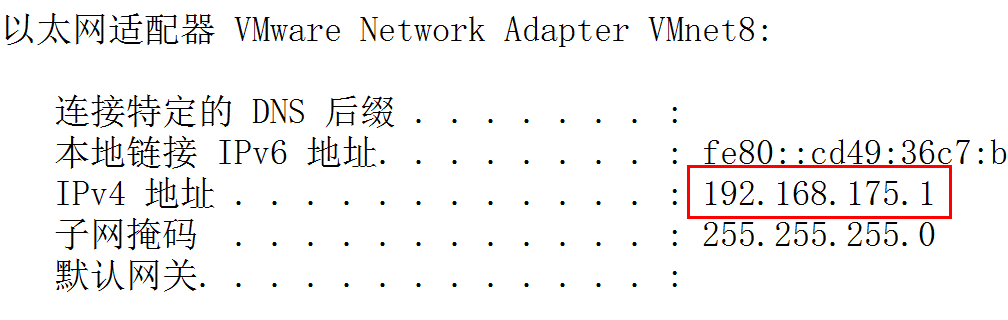
### JDK生效

命令:source /etc/profile



### 修改Mysql链接

说明:检查windows系统中的net8网卡IP地址.



根据IP地址修改YML配置文件的数据库链接

spring:

datasource:

#引入druid数据源

type: com.alibaba.druid.pool.DruidDataSource

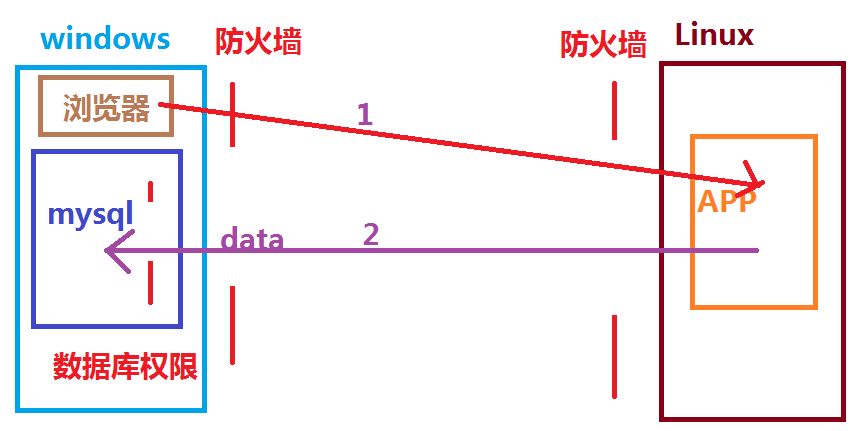
driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://192.168.175.1:3306/jtdb?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

username: root

password: root

### 服务器链接策略



### 关于数据库链接问题

说明:当用户修改为net8网卡IP地址之后.重启服务器时发现数据库不能正常链接.则需要关闭防护墙好数据库权限

语法：

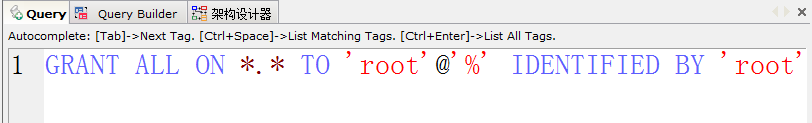
grant [权限] on [数据库名].[表名] to ['用户名']@['web服务器的ip地址'] identified by ['密码'];

grant all on \*.\* to 'root'@'%' identified by 'root';

或者指定IP地址

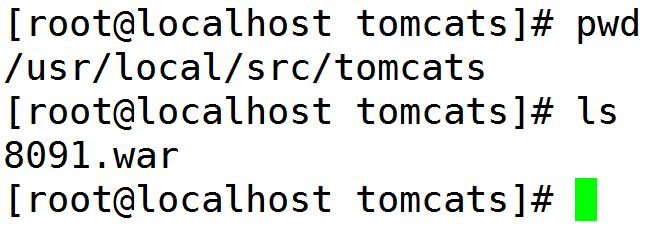
grant all on \*.\* to 'root'@'192.168.1.103' identified by 'root';

在sqlYog中执行数据库开放权限的指令,之后测试



### 项目打包

修改完成之后,将jt-manage项目打包之后长传到/usr/local/src/tomcats目录中



### 项目启动

将项目发布到Linux中之后启动服务器

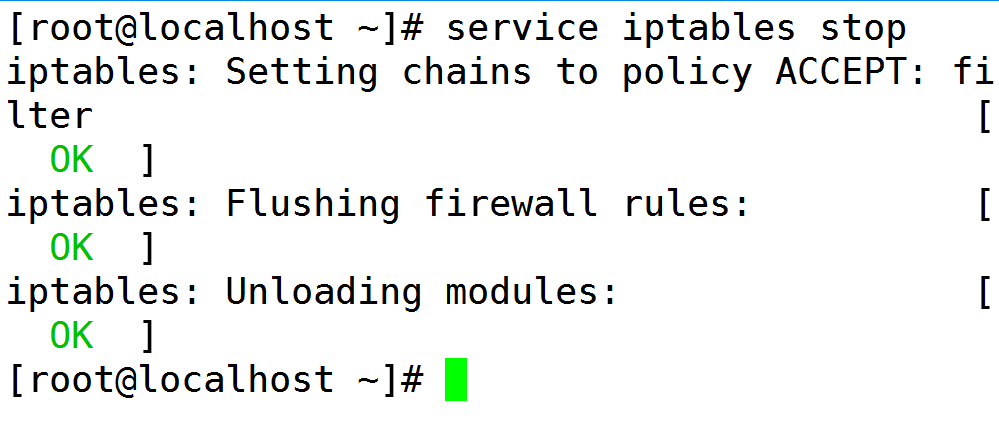
Java –jar 8091.war

### 关闭Linux防火墙

1.临时关闭 当Linux服务器重启时防火墙重新启动

service iptables stop

service iptables start 启动防火墙



2.永久关闭防护墙

chkconfig iptables off

chkconfig iptables on

### 效果测试



## Tomcat服务器启动事项

### 指令介绍

#### 关闭终端: Crtl + C

#### 连续操作指令:

java –jar 8091.war &

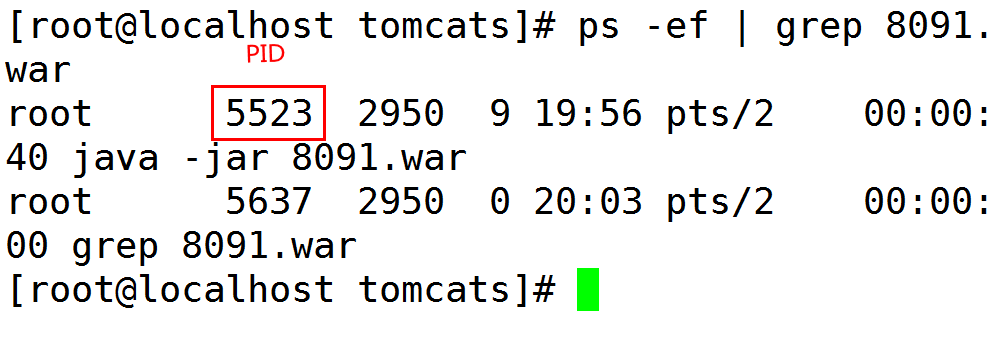
java –jar 8092.war &

#### 关闭tomcat服务器

1. 查询tomcat服务器PID

ps –ef | grep 服务名称

规则:管道之前查询的结果当做管道之后查询的条件



1. 杀死进程号PID

强制杀死进程 Kill -9 5523

友好的杀死 kill -15 5523

4).通过命令查找服务

ps -ef |grep **java\***

root 5664 2950 47 20:10 pts/2 00:00:27 java -jar 8091.war

root 5665 2950 47 20:10 pts/2 00:00:27 java -jar 8092.war

root 5774 5706 1 20:11 pts/1 00:00:00 grep java\*

## Nginx实现tomcat集群

### 编辑配置文件

编辑Nginx配置文件,配置Linux服务器集群信息.之后重启nginx服务器.

#准备Linux服务器配置

upstream jt-linux {

server 192.168.175.129:8091;

server 192.168.175.129:8092;

}

#后台管理服务器 用户访问manage.jt.com时访问localhost:8091

server {

listen 80;

server\_name manage.jt.com;

location / {

#代理路径

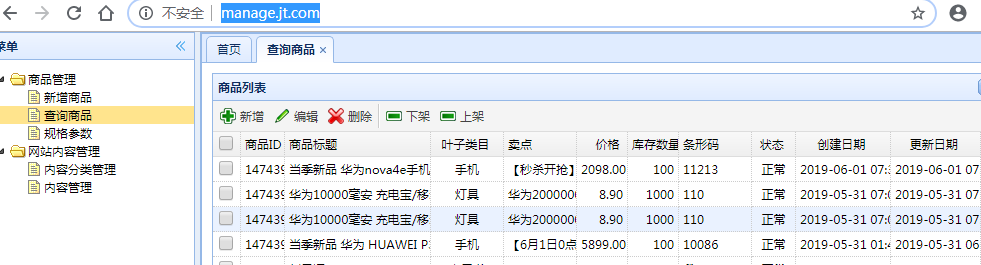
proxy\_pass http://jt-linux;

}

}

### 启动测试

通过域名访问Linux中tomcat服务器.



# Redis缓存

## 什么是缓存

### 缓存机制

引入缓存实际上就是降低了用户访问数据的频次.同时缓存中的数据就是数据库中的数据,将来要保证数据的一致性.提高了用户查询的效率.

### 实现缓存需要考虑的问题

1. 存储结构 k-v key不能重复
2. 开发语言 C
3. 缓存数据都保存到内存中
4. 将内存中的数据定期持久化到本地磁盘中
5. 定期对内存数据优化

LRU算法:可以将内存中不经常使用的数据提前删除

LFU算法:可以将内存中使用次数最少的数据提前删除

## Redis

### Redis介绍

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。 它支持多种类型的数据结构，如 [**字符串（strings）**](http://www.redis.cn/topics/data-types-intro.html#strings)， [**散列（hashes）**](http://www.redis.cn/topics/data-types-intro.html#hashes)， [**列表（lists）**](http://www.redis.cn/topics/data-types-intro.html#lists)， [**集合（sets）**](http://www.redis.cn/topics/data-types-intro.html#sets)， [**有序集合（sorted sets**）](http://www.redis.cn/topics/data-types-intro.html#sorted-sets) 与范围查询， [bitmaps](http://www.redis.cn/topics/data-types-intro.html#bitmaps)， [hyperloglogs](http://www.redis.cn/topics/data-types-intro.html#hyperloglogs) 和 [地理空间（geospatial）](http://www.redis.cn/commands/geoadd.html) 索引半径查询。 Redis 内置了 [复制（replication）](http://www.redis.cn/topics/replication.html)，[LUA脚本（Lua scripting）](http://www.redis.cn/commands/eval.html)， [LRU驱动事件（LRU eviction）](http://www.redis.cn/topics/lru-cache.html)，[事务（transactions）](http://www.redis.cn/topics/transactions.html) 和不同级别的 [磁盘持久化（persistence）](http://www.redis.cn/topics/persistence.html)， 并通过 [Redis哨兵（Sentinel）](http://www.redis.cn/topics/sentinel.html)和自动 [分区（Cluster）](http://www.redis.cn/topics/cluster-tutorial.html)提供高可用性（high availability）。

### Redis性能

读:11.2万/秒

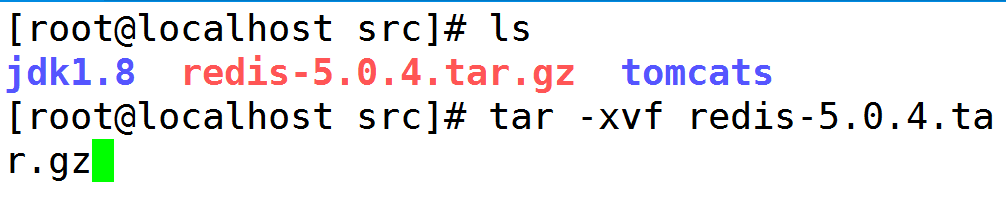
写:8.6万/秒

平均: 10万/秒 吞吐量

## Redis安装和使用

### 上传文件

上传redis安装文件.之后解压



### 安装redis

要求:命令的执行值redis的根目录中运行.

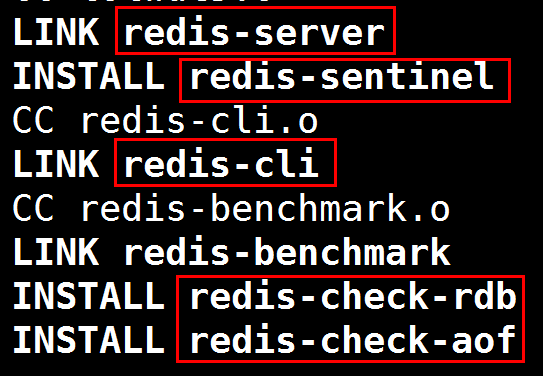
命令:

1.make 表示编译文件

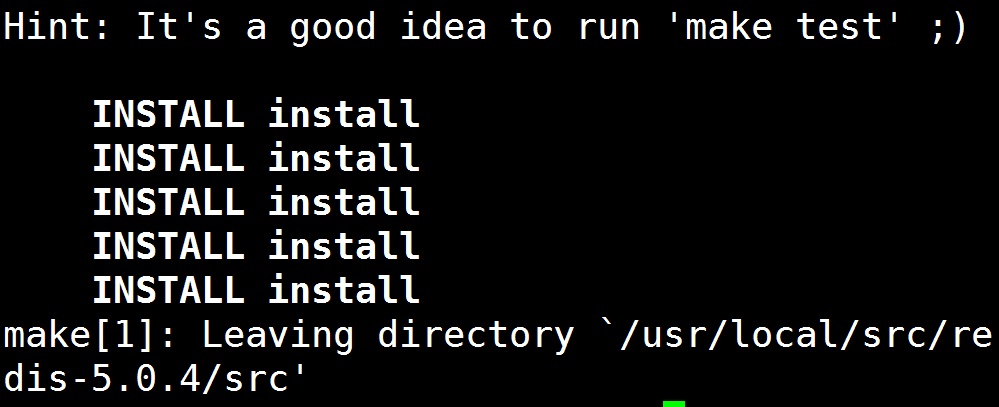
2.make install 表示安装redis服务.

命令执行:

1.make



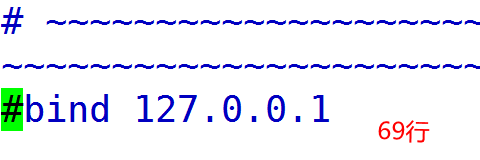
2.make install



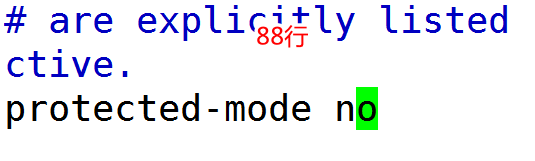
### 修改配置文件

vim redis.conf

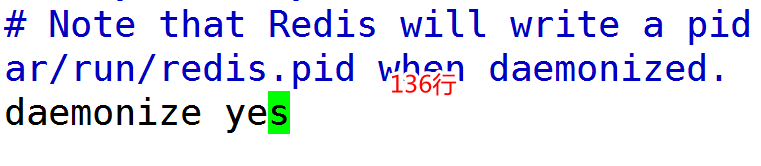
1. 去除IP绑定



1. 关闭保护模式

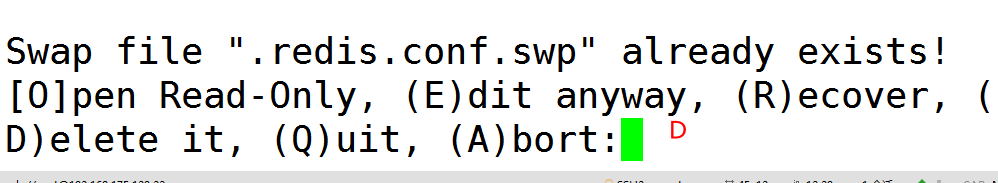


1. 开启后台启动



### 关于VIM常见报错

报错演示:



报错原因:

在vim编辑状态下.程序异常外退出导致Linux程序启动保护模式.导致异常.

解决方案:

1. 根据选项 点击 D
2. 利用删除指令 直接删除临时文件 rm –rf .xxxxxx.swap

### Redis运行命令

1. 启动redis

redis-server 忽略 一切加载都是默认项

**redis-server redis.conf** 目的读取配置文件

1. **进入redis客户端**

redis-cli –p 6379

exit 退出客户端

ctrl + c 退出

简化:如果Redis默认的端口号为6379则命令可以简化

redis-cli

1. redis关闭

redis-cli –p 6379 shutdown

kill -9 PID PID2

redis-cli shutdown 简化命令

## 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.

补充:一般使用list集合一般当做消息队列.

消息队列作用:缓解服务器压力.实现了异步操作.

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高级应用

## Redis入门案例

### 添加jar包文件

说明:在JT-PARENT项目中添加jar包文件

<!-- 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>

### 入门案例-String

/\*\*

\* 连接单台redis

\* 参数介绍:

\* redisIP地址.

\* redis:6379

\*/

@Test

**public** **void** test01(){

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

jedis.set("redis", "redis入门案例");

System.***out***.println

("获取redis中的数据:"+jedis.get("redis"));

//为数据设定超时时间 单位秒

jedis.setex("1804", 100, "1804班");

}

### 入门案例-hash

@Test

**public** **void** test01(){

Jedis jedis = **new** Jedis("192.168.126.148", 6379);

jedis.hset("user", "id", "1");

jedis.hset("user", "name", "tomcat");

jedis.hset("user", "age", "18");

System.***out***.println("操作完成!!!"+jedis.hget("user", "id"));

Map<String,String> map = jedis.hgetAll("user");

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

}

结果展现:

操作完成!!!1

{name=tomcat, age=18, id=1}

### 入门案例-List

@Test

**public** **void** test02(){

Jedis jedis = **new** Jedis("192.168.126.148", 6379);

Long number = jedis.lpush("list", "a","b","c","d","e");

System.***out***.println("获取数据"+number);

List<String> list= jedis.lrange("list", 0, -1);

System.***out***.println("获取参数:"+list);

}

结果展现:

获取数据5

获取参数:[e, d, c, b, a]

### 事务控制

//redis事务控制

**public** **void** TestTx() {

Jedis jedis = **new** Jedis("192.168.175.129",6379);

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

**try** {

transaction.set("ww", "wwww");

transaction.set("dd", **null** );

transaction.exec();

} **catch** (Exception e) {

transaction.discard();

}

}

## SpringBoot整合Redis

### 导入jar包

<!--spring整合redis -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

</dependency>

<dependency>

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

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

</dependency>

### 入门案例

//String类型操作方式 配置文件3处 防火墙

//IP:端口号

@Test

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

Jedis jedis =

**new** Jedis("192.168.175.129",6379);

jedis.set("1902","1902班");

jedis.expire("1902", 10);

System.***out***.println(jedis.get("1902"));

}

### 超时时间定时

//设定数据的超时方法

//分布式锁!!!!!

@Test

**public** **void** testTimeOut() **throws** InterruptedException {

Jedis jedis = **new** Jedis("192.168.175.129",6379);

jedis.setex("aa", 2, "aa");

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

Thread.*sleep*(3000);

//当key不存在时操作正常.当key存在时,则操作失败

Long result = jedis.setnx("aa","bb");

System.***out***.println("获取输出数据:"+result+":"+jedis.get("aa"));

}

## 对象转化JSON

### 转化说明

利用ObjectMapper实现对象的转化.

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

//对象转化为JSON

String json串 = mapper.writeValueAsString(对象);

//JSON转化为对象

对象 T = mapper.readValue(json, 对象.**class**);

### 对象与JSON互转

/\*\*

\* 3.利用Redis保存业务数据 数据库

\* 数据库数据: 对象 Object

\* String类型要求只能存储字符串类型

\* item ~~~ JSON ~~~ 字符串

\*/

@Test

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

ItemDesc itemDesc = **new** ItemDesc();

itemDesc.setItemId(1000L)

.setItemDesc("测试方法")

.setCreated(**new** Date())

.setUpdated(**new** Date());

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

//转化JSON时必须 get/set方法

String json =

mapper.writeValueAsString(itemDesc);

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

//将json串转化为对象

ItemDesc desc2 =

mapper.readValue(json, ItemDesc.**class**);

System.***out***.println("测试对象:"+desc2);

}

### List与JSON互转

//实现List集合与JSON转化

@SuppressWarnings("unused")

@Test

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

ItemDesc itemDesc1 = **new** ItemDesc();

itemDesc1.setItemId(1000L)

.setItemDesc("测试方法");

ItemDesc itemDesc2 = **new** ItemDesc();

itemDesc2.setItemId(1000L)

.setItemDesc("测试方法");

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

list.add(itemDesc1);

list.add(itemDesc2);

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

String json = mapper.writeValueAsString(list);

System.***out***.println("集合转化为JSON"+json);

//将数据保存到redis中

Jedis jedis = **new** Jedis("192.168.175.129",6379);

jedis.set("itemDescList", json);

//从redis中获取数据

String result = jedis.get("itemDescList");

List<ItemDesc> descList = mapper.readValue(result,list.getClass());

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

}

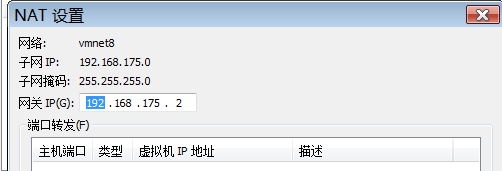
## 设定静态IP地址

### 配置过程

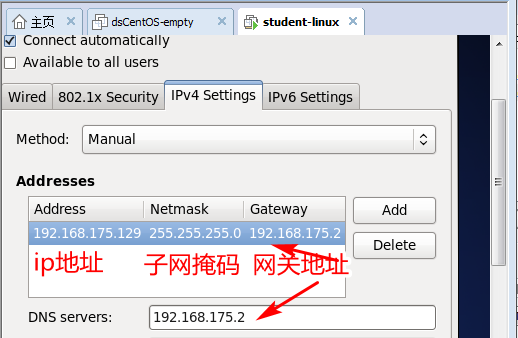
1. 检查当前IP地址

192.168.175.129

1. 检查网关地址



1. 配置静态IP



# Spring整合Redis

## 关于对象与JSON转化过程

### 对象转化JSON

/\*\*

\* 1.首先获取对象的getXXXX方法.

\* 2.将get去掉,之后首字母小写获取属性的名称

\* 3.之后将属性名称 : 属性的值进行拼接.

\* 4.形成json串(字符串)

\* **@throws** JsonProcessingException

\*/

@Test

**public** **void** userToJSON() **throws** JsonProcessingException {

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

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

user.setId(1000);

user.setName("json测试");

user.setAge(18);

user.setSex("男");

String json =

mapper.writeValueAsString(user);

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

}

### Json串转化对象

/\*\*

\* 1.获取userJSON串

\* 2.通过json串获取json中key

\* 2.根据class类型的反射机制实例化对象

\* 3.根据key调用setKey方法为对象赋值.

\* 4.最终生成对象.

\* 5.可以利用

\* **@JsonIgnoreProperties**(ignoreUnknown = true)

\* 注解忽略未知属性

\*

\* **@throws** IOException

\*/

@Test

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

//该方法准备json串

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

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

user.setId(1000);

user.setName("json测试");

user.setAge(18);

user.setSex("男");

String json =

mapper.writeValueAsString(user);

//以下方法实现了数据的转化

User user2 = mapper.readValue(json, User.**class**);

System.***out***.println(user2);

}

## 编辑工具类

### 对象转化JSON

在jt-common中.添加工具类方法.将对象转化为JSON串

//编辑工具类实现对象与json转化

**public** **class** ObjectMapperUtil {

**private** **static** **final** ObjectMapper ***MAPPER***

= **new** ObjectMapper();

**public** **static** String toJSON(Object target) {

String json = **null**;

**try** {

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

} **catch** (Exception e) {

e.printStackTrace();

//将检查异常转化为运行时异常

**throw** **new** RuntimeException();

}

**return** json;

}

}

### 编辑JSON转化为对象

**public** **static** <T> T toObject(String json,Class<T> targetClass) {

T target = **null**;

**try** {

target = ***MAPPER***.readValue(json, targetClass);

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException();

}

**return** target;

}

### 修改配置文件

1. 修改yml文件

spring:

datasource:

#引入druid数据源

type: com.alibaba.druid.pool.DruidDataSource

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/jtdb?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

username: root

password: root

1. 修改Nginx配置文件

#后台管理服务器 用户访问manage.jt.com时访问localhost:8091

server {

listen 80;

server\_name manage.jt.com;

location / {

#代理路径

proxy\_pass http://127.0.0.1:8091;

}

}

}

修改完成之后,重启nginx

## Spring整合Redis

### 编辑pro文件

jedis.host=192.168.175.129

jedis.port=6379

### 编辑配置类

//表示redis配置类

@Configuration

@PropertySource("classpath:/properties/redis.properties")

**public** **class** RedisConfig {

@Value("${jedis.host}")

**private** String host;

@Value("${jedis.port}")

**private** Integer port;

@Bean

**public** Jedis jedis() {

**return** **new** Jedis(host, port);

}

}

## 实现商品分类缓存

### 实现思路

1. 修改业务调用逻辑改为先查询缓存
2. 如果redis中根据key查询时没有数据则查询数据库
3. 将查询到的数据转化为JSON串之后保存到redis中
4. 如果重redis中查询数据时有数据.将json串转化为对象返回.

### 修改Controller中方法

//查询全部数据的商品分类信息 id=560

//需要获取任意名称的参数,为指定的参数赋值.

//@RequestParam value/name 接收参数名称 defaultValue="默认值" required = true/false 是否必须传值

@RequestMapping("/list")

**public** List<EasyUITree> findItemCatByParentId(@RequestParam(value="id",defaultValue="0")Long parentId){

//return itemCatService.findItemCatByParentId(parentId);

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

}

### 编辑Service

@Override

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

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

String result = jedis.get(key);

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

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

//如果为null,查询数据库

treeList = findItemCatByParentId(parentId);

//将数据转化为json

String json = ObjectMapperUtil.*toJSON*(treeList);

jedis.setex(key, 7\*24\*3600, json);

System.***out***.println("业务查询数据库!!!!!");

}**else** {

//表示缓存中有数据

treeList = ObjectMapperUtil.*toObject*(result,treeList.getClass());

System.***out***.println("业务查询redis缓存!!!!!");

}

**return** treeList;

}

### Redis速度测试

说明:将redis中缓存数据清空,之后通过F12开发者工具测试

没有缓存和有缓存时间差异.

没有redis缓存:



有redis缓存:



作业:利用AOP机制动态添加缓存.

通知选择: 环绕通知

编辑自定义注解: @Cache(xxx,xxx,xxx)

要求:用户使用越简单越好

## Redis分片机制

### Redis单台问题

1. 使用redis单台时内存会明显不足.如果一味的扩容内存,则运行的效率低.
2. 如果redis单台服务器宕机.

### Redis分片(分区)介绍

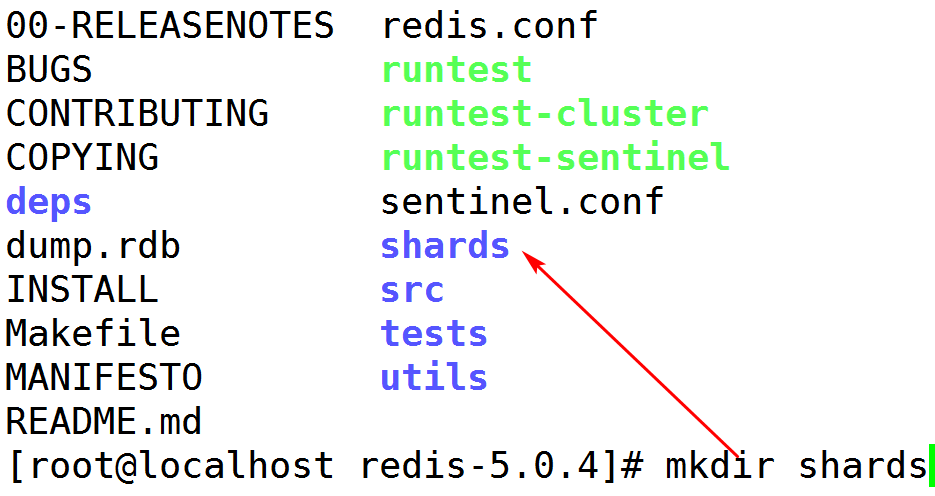
核心特点:实现了redis内存的扩容.

说明:使用多个redis节点,共同为用户提供服务.内存空间翻倍.

用户使用时当做一个整体.并且内存保存的数据不一.

## 分片搭建

### 创建新文件

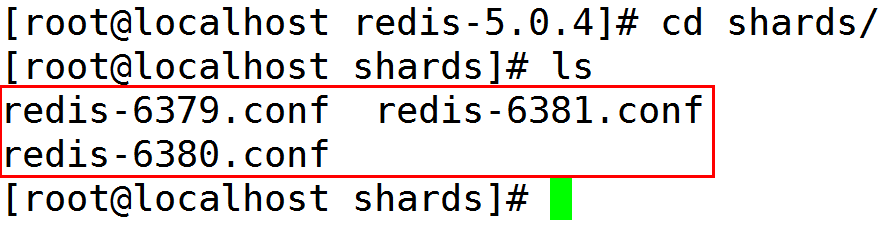


### 复制3个配置文件

[root@localhost redis-5.0.4]# cp redis.conf shards/redis-6379.conf

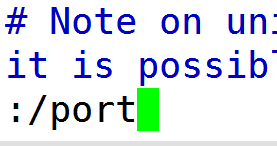
[root@localhost redis-5.0.4]# cp redis.conf shards/redis-6380.conf

[root@localhost redis-5.0.4]# cp redis.conf shards/redis-6381.conf

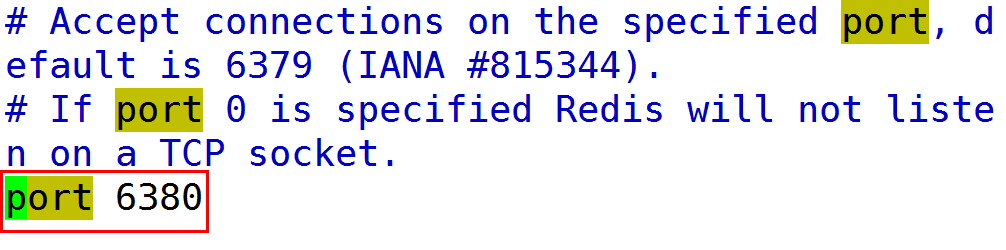


### 修改端口号

可以使用:/port快速定位数据



将6380.conf中端口号改为6380.



同理 修改6381.conf文件

### 启动多台redis

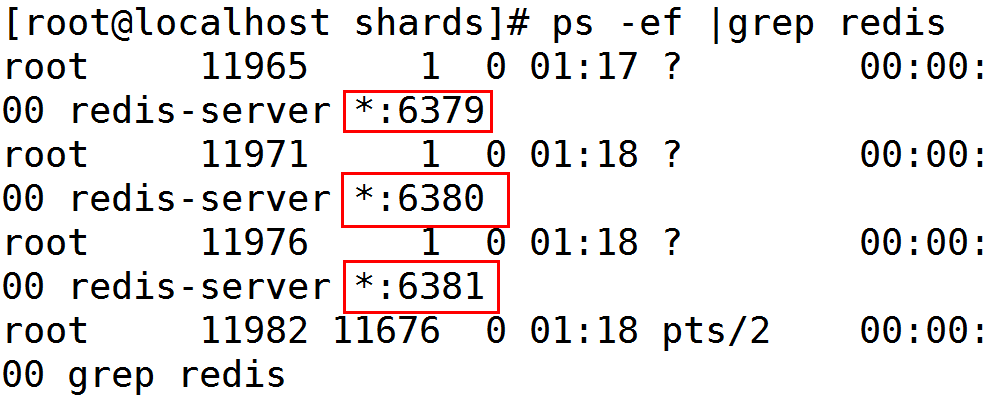
1.启动redis

redis-server redis-6379.conf

redis-server redis-6380.conf

redis-server redis-6381.conf

2.检查redis服务



## Redis分片测试

### 入门案例

/\*\*

\* 操作时需要将多台redis当做1台使用.

\*/

@Test

**public** **void** testShards() {

List<JedisShardInfo> shards =

**new** ArrayList<JedisShardInfo>();

JedisShardInfo info1 =

**new** JedisShardInfo("192.168.175.129",6379);

JedisShardInfo info2 =

**new** JedisShardInfo("192.168.175.129",6380);

JedisShardInfo info3 =

**new** JedisShardInfo("192.168.175.129",6381);

shards.add(info1);

shards.add(info2);

shards.add(info3);

//操作分片 redis对象工具类

ShardedJedis shardedJedis =

**new** ShardedJedis(shards);

shardedJedis.set("1902","1902班");

System.***out***.println(shardedJedis.get("1902"));

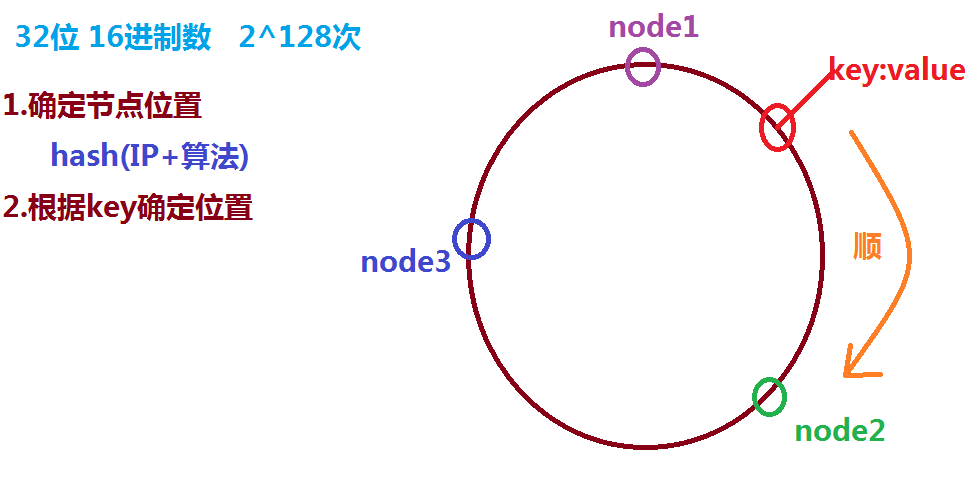
}

## Hash一致性算法

### 概念说明

概念:同一个字符串hash值是一致的.

### Hash一致性说明



### Hash一致性—特性

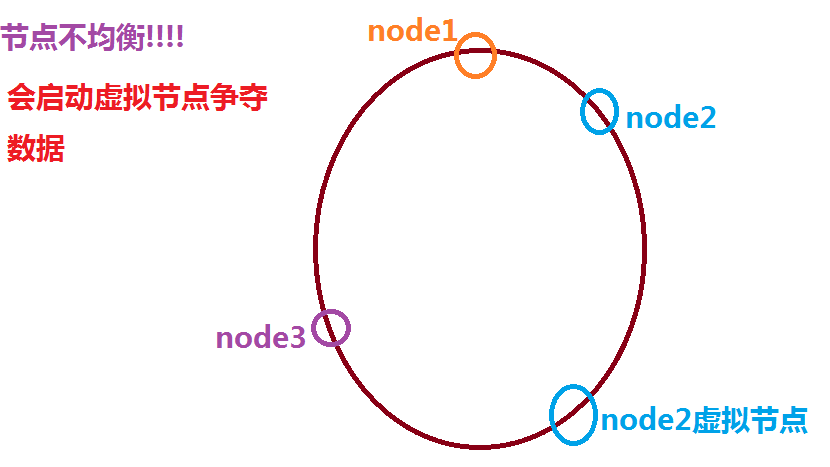
#### 问题说明

因为所有节点都是通过ip地址加算法计算获取的,则可能会出现节点分配不均的问题.导致数据丢失.

#### 均衡性

说明:均衡性要求节点中的数据尽可能的平均.

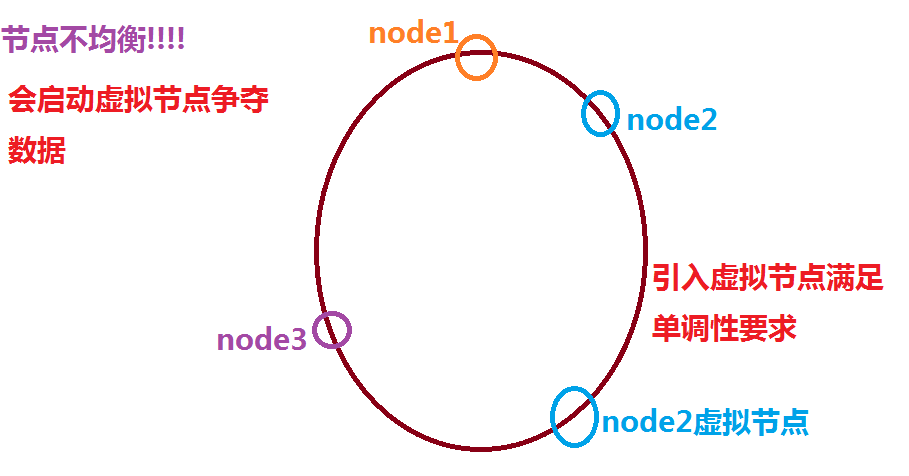
措施:引入虚拟节点概念



#### 单调性

说明:当节点**新增**时,能够实现数据的自动的迁移.

补充说明:如果节点一旦丢失,则导致内存丢失则整个分片无法使用.



#### 分散性

概念:由于分布式原因,导致系统不能获取全部的内存空间.导致一个key有多个位置.

#### 负载

概念:由于分布式原因,系统不能获取全部的内存地址.导致同一个位置保存多个数据