# Redis

## Tomcat服务器集群后续

### 操作tomcat命令

1. java -jar 8091.war & java -jar 8092.war & java -jar 8093.war &
2. **ps -ef | grep 关键字**

管道:将上一步获取结果,当做下一步的参数

1. 杀死进程

kill PID 普通杀进程 如果权限不足无法杀死

kill -9 PID号 强制杀死进程

kill -15 PID号 等待进程执行完毕之后杀死

### 修改Nginx配置

server {

listen 80;

server\_name manage.jt.com;

location / {

#代理服务器路径

proxy\_pass http://localhost:8091;

#proxy\_pass http://jtWindows;

#proxy\_pass http://jtLinux;

#proxy\_connect\_timeout 2;

#proxy\_read\_timeout 2;

#proxy\_send\_timeout 2;

}

}

### 修改数据库链接

spring:

datasource:

#引入druid数据源

type: com.alibaba.druid.pool.DruidDataSource

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

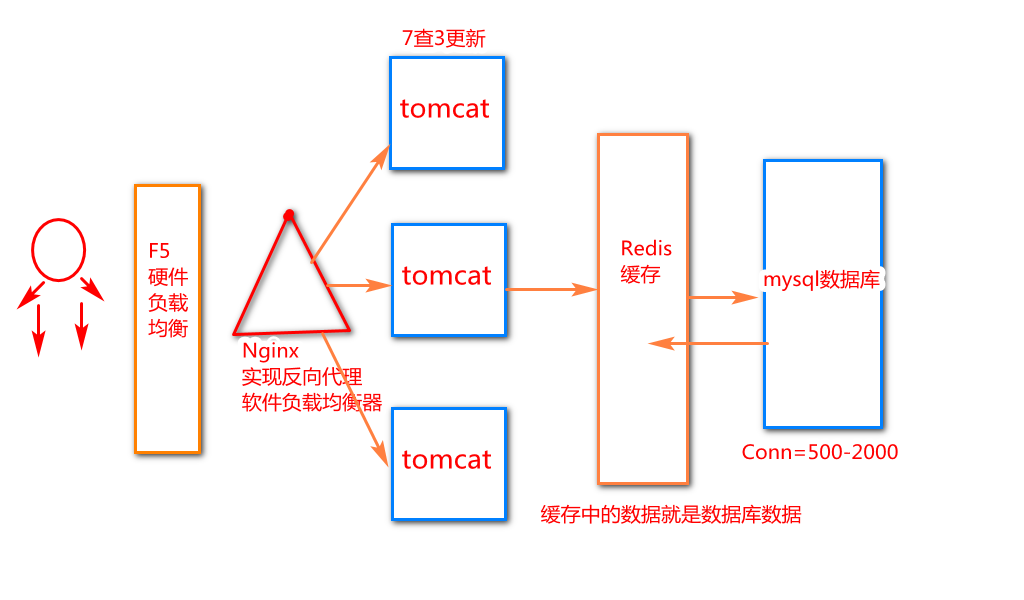
url: jdbc:mysql://**localhost:3306**/jtdb?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true

username: root

password: root

## 京淘项目架构优化

### 引入缓存策略



## 缓存实现方案

### 缓存需要考虑问题

1. 开发语言使用C语言.
2. 使用K-V保存缓存数据. [] list link map
3. 缓存一般运行在内存中.
4. 实现缓存数据的持久化.
5. 使用内存优化算法.控制内存使用大小

LRU算法:最近最少使用的数据提前删除.

LFU算法:按照使用次数将最少使用的数据删除.

## Redis

### Redis介绍

Redis是一个开源（BSD许可），内存存储的数据结构服务器，可用作**数据库**，**高速缓存**和**消息队列代理**。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)。内置复制、[Lua脚本](https://www.redis.net.cn/tutorial/3516.html)、LRU收回、[事务](https://www.redis.net.cn/tutorial/3515.html)以及不同级别磁盘持久化功能，同时通过Redis Sentinel提供**高可用**，通过**Redis Cluster**提供自动[分区](https://www.redis.net.cn/tutorial/3524.html)。

读速度: 11.2万/秒

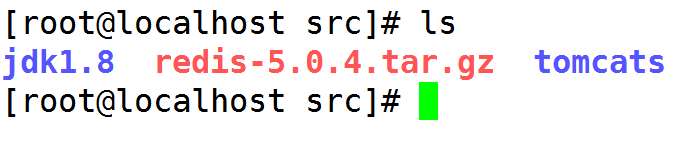
写速度: 8.6万/秒

平均读写:10万/秒

### Redis安装

#### 安装准备

1. 上传Redis安装包.



1. 解压文件

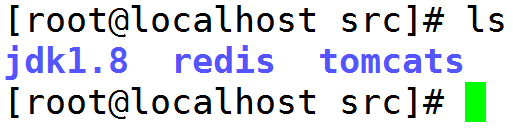
tar -xvf redis-5.0.4.tar.gz

1. 删除安装文件

rm -f redis-5.0.4.tar.gz

1. 修改文件名称

mv redis-5.0.4 redis

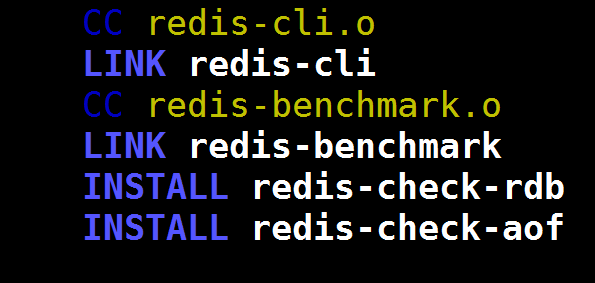


### Redis命令安装

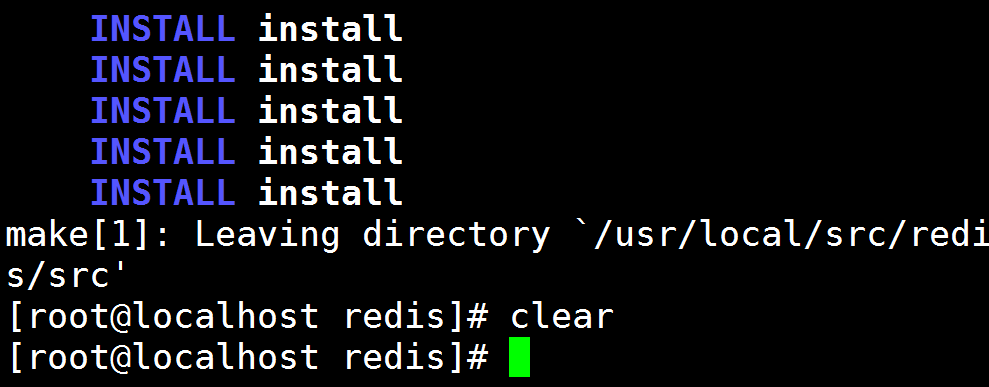
要求:在redis根目录中执行下列命令

命令:

1. make 编译程序



1. make install 安装redis

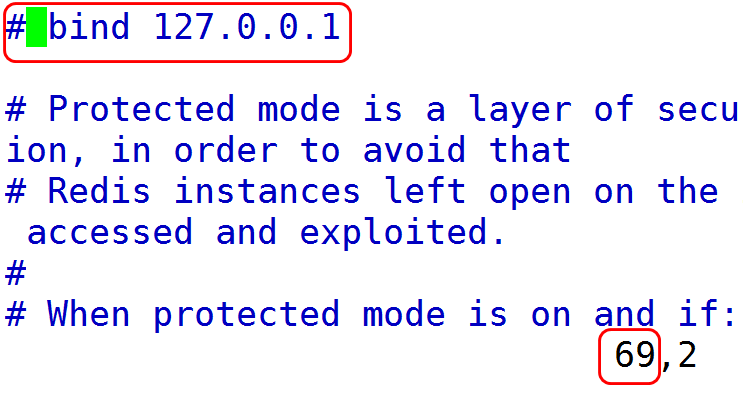


### 需要配置文件

说明:redis.conf是redis最为重要的配置文件.

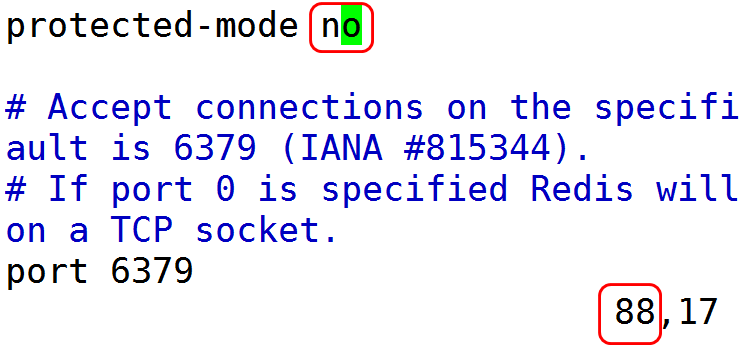
命令:vim redis.conf

#### 去除IP绑定

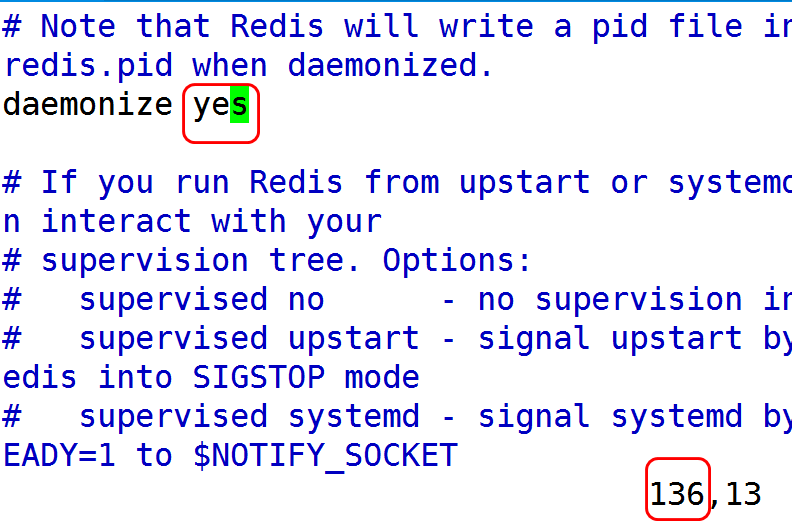


#### 关闭保护模式

保护模式默认是开启的.



#### 开启后台启动



### 启动命令

1.启动 redis-server redis.conf

2.停止 redis-cli -p 6379 shutdown

ps -ef |grep redis 检索服务

kill -9 PID号 杀死进程

3.客户端 redis-cli -p 6379

退出客户端 exit

说明:一般默认的6379 可以省略

## 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值的长度 | 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  -2没有该数据  -1 该数据一直存在 |
| 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集合是双端循环链表,分别可以从左右两个方向插入数据.List集合中的数据不能当做缓存使用.其中的数据都会被消费

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

push 压栈 pop 弹栈

|  |  |  |
| --- | --- | --- |
| 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入门案例

### 引入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>

### String 类型操作

**public** **class** TestRedis {

/\*\*

\* 1.Spring整合redis入门案例

\*/

@Test

**public** **void** testRedis1() {

String host = "192.168.182.129";

**int** port = 6379;

Jedis jedis = **new** Jedis(host, port);

jedis.set("1903","1903班下午好");

System.***out***.println(jedis.get("1903"));

//设定数据超时时间

jedis.expire("1903", 20);

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("1903key还能存活:"

+jedis.ttl("1903"));

}

/\*\*

\* 2.简化操作数据超时用法

\*/

@Test

**public** **void** testRedis2() {

Jedis jedis = **new** Jedis("192.168.182.129",6379);

jedis.setex("abc",100,"英文字母");

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

}

/\*\*

\* 3.锁机制用法

\* 实际用法: 保证set数据时如果这个key已经存在

\* 不允许修改.

\* 业务场景.

\* 小明: set("jimian","8点")

\* 小张: set("jimian","5点")

\*/

@Test

**public** **void** testRedis3() {

Jedis jedis = **new** Jedis("192.168.182.129",6379);

//jedis.set("yue", "8点xxxx地点");

//jedis.set("yue", "5点xxxxx地点"); //更新操作

Long flag1 = jedis.setnx("yue", "8点xxxx地点");

jedis.del("yue");

Long flag2 = jedis.setnx("yue", "5点xxxx地点");

System.***out***.println(flag1+":::"+flag2);

System.***out***.println("小丽约会时间:"+jedis.get("yue"));

}

/\*\*

\* 死锁

\* 1.setnx("yue","今晚8点") //加锁

\* 2.jedis.del("yue"); //减锁

\* 3.setnx("yue","今晚9点半") //加锁

\* 避免死锁:添加key的超时时间

\* 锁机制优化

\*/

@Test

**public** **void** testRedis4() {

Jedis jedis = **new** Jedis("192.168.182.129",6379);

String result1 = jedis.set("yue", "今晚8点", "NX", "EX",20);

//int a = 1/0;

//jedis.del("yue");

String result2 = jedis.set("yue", "今晚5点", "NX", "EX", 20);

System.***out***.println(result1);

System.***out***.println(result2);

}

}

### 操作Hash类型

/\*\*

\* 2.hash在工作中出场率低

\*/

@Test

**public** **void** testHash1() {

Jedis jedis = **new** Jedis("192.168.182.129",6379);

jedis.hset("user", "id", "120");

jedis.hset("user", "name", "测试数据");

jedis.hset("user", "age", "19");

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

}

### 操作List集合

/\*\*

\* 3.list集合

\*/

@Test

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

Jedis jedis = **new** Jedis("192.168.182.129",6379);

//1.当做队列

//jedis.lpush("list","1,2,3,4,5");//注意",号"

jedis.lpush("list","1","2","3","4");

System.***out***.println

("获取数据:"+jedis.rpop("list"));

}

### 事务控制

/\*\*

\* 4.测试事务控制

\*/

@Test

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

Jedis jedis = **new** Jedis("192.168.182.129",6379);

Transaction transaction = jedis.multi(); //1.开启事务

**try** {

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

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

**int** a =1/0; //模拟报错

transaction.exec(); //提交事务

} **catch** (Exception e) {

e.printStackTrace();

transaction.discard();

}

}

## SpringBoot整合Redis

### 编辑properties文件

#配置redis的节点信息

redis.host=192.168.182.129

redis.port=6379

### 编辑配置类

@Configuration //标识我是一个配置类

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

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

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

**private** String host;

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

**private** Integer port;

/\*\*

\* 回顾:

\* 1.xml配置文件 添加bean标签 (远古时期)

\* 2.配置类的形式

\* 配置:

\* 将jedis对象交给spring容器管理

\*

\* 利用properties配置文件为属性动态赋值.

\*

\*/

@Bean //<bean id="jedis" class="包.jedis">

**public** Jedis jedis() {

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

}

}

## 对象与JSON转化

### 对象转化代码

**public** **class** TestObjectToJSON {

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

//将对象转化为JSON串

@Test

**public** **void** toJSON() {

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

itemDesc.setItemId(1000L)

.setItemDesc("我是一个测试用例!!!")

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

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

**try** {

String json =

mapper.writeValueAsString(itemDesc);

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

//~~~~~~~~~~~~~~~~~~~~~~~

//将json转化为对象类型 反射机制

ItemDesc itemDesc2 = mapper.readValue(json,ItemDesc.**class**);

System.***out***.println(itemDesc2);

} **catch** (IOException e) {

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

e.printStackTrace();

}

}

}

### 定义对象转化工具API

在工具代理jt-common中添加工具API调用

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

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

//1.将对象转化为JSON

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

String result = **null**;

**try** {

result = ***mapper***.writeValueAsString(target);

} **catch** (JsonProcessingException e) {

e.printStackTrace();

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

}

**return** result;

}

//2.将json转化为对象

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

T t = **null**;

**try** {

t = ***mapper***.readValue(json, targetClass);

} **catch** (IOException e) {

e.printStackTrace();

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

}

**return** t;

}

}