Redis

1. 存储结构

Redis是Remote Dictionary Server远程字典服务器。以字典结构存储数据，允许其他应用通过TCP协议读写字典中的内容。

1. 准备
2. 启动和停止Redis

启动Redis有直接启动和通过初始化脚本启动，分别适用于开发环境和生产环境。

1. 直接启动：

redis-server 默认使用6379端口,通过—port 6380可以设置

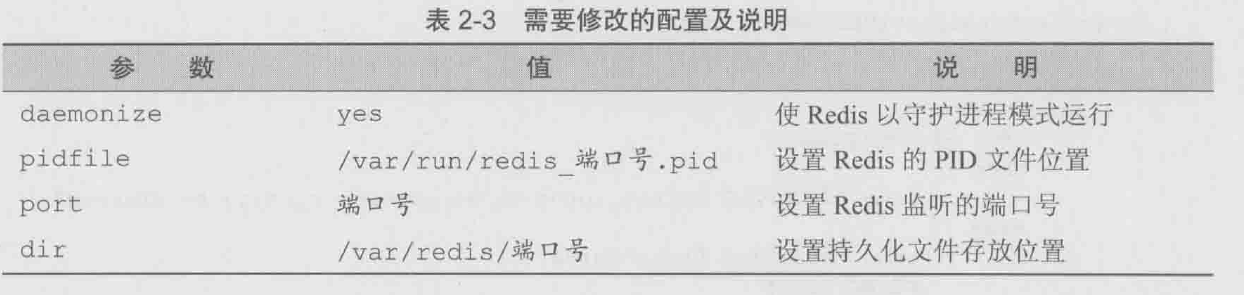
1. 通过初始化脚本启动Redis

在Redis源代码目录的utils文件夹中有一个名为redis\_init\_script的初始化脚本文件。我们需要配置Redis的运行方式和持久化文件、日志文件的存储位置等。

[1].配置初始化脚本文件。首先把初始化脚本复制到/etc/init.d目录中，文件名为redis\_**端口号**，端口号表示要让Redis监听的端口号，客户端通过该端口连接Redis。然后修改脚本第6行REDISPORT变量的值为同样的端口号。

[2].建立文件夹，/etc/redis用于存放redis配置文件，/var/redis/端口号用于存放redis持久化文件。

[3].修改配置文件，首先将配置文件模版复制到/etc/redis目录中，以端口号命名。然后安装如下修改：



这样就可以通过sudo update-rc.d redis\_端口号 defaults

1. 停止redis

redis-cli SHUTDOWN当redis收到SHUTDOWN的命令 后，会先断开客户端连接，然后根据配置进行持久化，最后完成退出。

1. Redis命令行客户端

通过redis-cli进入交互模式

命令返回值：状态回复、错误回复、整数回复、字符串回复、多行字符串回复。

1. 入门
2. 基本语句

SET bar 1 设置名为bar的键，然后通过KEY \*或者KEY bar可以查看对于的键。KEY会遍历所有的键，不建议使用。Redis不区分大小写，但是习惯用大写表示命令。

EXISTS key判断一个键是否存在，存在返回1，否则返回0.

DEL key删除键 返回值是删除键的个数。

TYPE key用来获取键值的数据类型。

1. 字符串类型

命令

赋值与取值：SET key、GET key

递增数字：INCR key

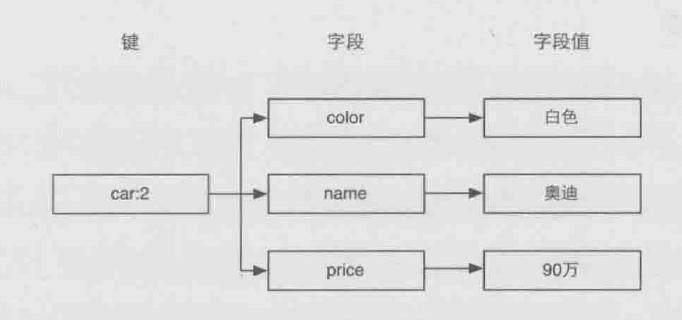
生成自增ID：redis对于每一类对象使用名为对象类型(复数形式):count的键（如users：count）来存储当前类型对象的数量。

加：INCRBY key num INCRBY bar 2

减：DECRBY key num

1. 散列类型：

散列适合存储对象，使用对象类别和ID构成键名，使用字段表示对象的属性，而字段值则存储属性值。



命令：

赋值与取值：HSET key field value

HMSET key field1 value1 field2 value2

HGET key field

HMGET key field[field…]

HGETALL key

HEXISTS kye field判断字段是否存在

HSETNX key field value 当字段不存在时赋值

HDEL key field 删除字段

1. 列表类型

命令：

LPUSH key value [value ...]向左边增加元素；

RPUSH key value[value ...]向右边增加元素

LPOP key 从列表左边弹出第一个元素

RPOP key 从列表右边弹出第一个元素

LINE key 获取列表中元素的个数

LRANGE key start stop 获取列表的片段

LREM key count value删除列表指定的值

1. 集合类型

命令：

SADD key member 增加元素

SREM key member 删除元素

SMEMBERS key 获取集合中所有的元素

SISMEMBER key member判断元素是否在集合中

SDIFF key1 key2… 执行集合差 key1 – key2

SINTER setA setB 执行交集运算

SUNION setA setB 执行并集运算

1. 有序集合

命令：

ZADD key score value向有序集合加入一个元素和该元素的分数，如果该元素已经存在则用新的分数代替原有的分数。

ZSCORE kye member获取元素的分数

ZRANGE key start stop按照分数从小到大的顺序返回。Start和stop都是索引。

ZRANGEBYSCORE key min max获取指定分数范围的元素。

1. 事务

Redis 事务可以一次执行多个命令， 并且带有以下两个重要的保证：

事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

一个事务从开始到执行会经历以下三个阶段：

1.开始事务。

2.命令入队。

3.执行事务。

命令：先以 MULTI 开始一个事务， 然后将多个命令入队到事务中， 最后由 EXEC 命令触发事务， 一并执行事务中的所有命令。

DISCARD 取消事务放弃事务块里的所有命令

EXEC 执行事务块你的命令

MULTI 标记一个事务开始

UNWATCH 取消监视

WATCH 监视一个或多个key

1. spring整合redis
2. 准备包jedis-2.1.0.jar，commons-pool-1.5.4.jar，spring-data-redis-1.0.0.RELEASE.jar
3. 编写redis.properties配置文件；

#最大分配的对象数

redis.pool.maxActive=1024

#最大能够保持idel状态的对象数

redis.pool.maxIdle=200

#当池内没有返回对象时，最大等待时间

redis.pool.maxWait=1000

#当调用borrow Object方法时，是否进行有效性检查

redis.pool.testOnBorrow=true

#IP （host）

redis.ip=127.0.0.1

#Port

redis.port=6379

1. 编写spring-redis.xml配置文件

低版本的配置如下：

<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">

<property name="maxActive" value="${redis.pool.maxActive}" />

<property name="maxIdle" value="${redis.pool.maxIdle}" />

<property name="maxWait" value="${redis.pool.maxWait}" />

<property name="testOnBorrow" value="${redis.pool.testOnBorrow}" />

</bean>

高版本的配置如下：

<beanid="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">

<property name="maxIdle" value="${redis.pool.maxIdle}"/>

<property name="maxTotal" value="${redis.pool.maxActive}"/>

<property name="maxWaitMillis" value="${redis.pool.maxWait}"/>

<property name="testOnBorrow" value="${redis.pool.testOnBorrow}"/>

<property name="testOnReturn" value="${redis.pool.testOnReturn}"/>

</bean>

 <!-- redis的连接池pool，不是必选项：timeout/password  -->

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

**<constructor-arg** index="0" ref="jedisPoolConfig"**/>**

**<constructor-arg** index="1" value="${redis.host}"**/>**

**<constructor-arg** index="2" value="${redis.port}" type="int"**/>**

**<constructor-arg** index="3" value="${redis.timeout}" type="int"**/>**

**<constructor-arg** index="4" value="${redis.password}"**/>**

**</bean>**

<bean id = "redisTemplate" **class**="org.nubia.framework.data.redis.JedisTemplate">

   <property name="jedisPoll" ref="jedisClient"/>

</bean>

Jedis基础

Jedis是redis的java版本的客户端实现

[一]. 入门使用：  
 下载Jedis的依赖包jedis-2.1.0.jar，然后将其添加到classpath下面。然后，即可进行编程：  
  1. 定义连接：Redis暂时不要设置登录密码

1. Jedis jedis = **new** Jedis("192.168.142.12");

2. 进行键值存储：

1. jedis.set("country", "China");

3. 获取value值：

1. String country = jedis.get("country");

4. 删除key：

1. jedis.del("country");

[二]. 使用连接池：  
  1. 添加依赖包commons-pool.jar，注意不要选择高版本，以免不必要的错误。  
  2. 配置属性文件：redis.properties

1. redis.host=192.168.142.12       #Redis服务器地址
2. redis.port=6379            #服务端口
3. redis.timeout=3000           #超时时间：单位ms
4. redis.password=nick123         #授权密码
6. redis.pool.maxActive=200        #最大连接数：能够同时建立的“最大链接个数”
7. redis.pool.maxIdle=20         #最大空闲数：空闲链接数大于maxIdle时，将进行回收
8. redis.pool.minIdle=5          #最小空闲数：低于minIdle时，将创建新的链接
9. redis.pool.maxWait=3000        #最大等待时间：单位ms
11. redis.pool.testOnBorrow=true    #使用连接时，检测连接是否成功
12. redis.pool.testOnReturn=true    #返回连接时，检测连接是否成功

3. 加载属性文件：redis.properties

1. ResourceBundle bundle = ResourceBundle.getBundle("redis");

4. 创建配置对象：

1. JedisPoolConfig config = **new** JedisPoolConfig();
2. String host = bundle.getString("redis.host");
3. ...
4. config.setMaxActive(Integer.valueOf(bundle.getString("redis.pool.maxActive")));
5. ...
6. config.setTestOnBorrow(Boolean.valueOf(bundle.getString("redis.pool.testOnBorrow")));
7. ...

  5. 创建Jedis连接池：

1. JedisPool pool = **new** JedisPool(config, host, port, timeout, password);

 [三]. 使用方式：     
 1. 从连接池获取Jedis对象：

1. Jedis jedis = pool.getResource();

2. 基本操作：

1. jedis.set("province", "shannxi");
2. String province = jedis.get("province");
3. jedis.del("province");

3. 将Jedis对象归还给连接池：

1. pool.returnResource(jedis);

**Jedis常见操作**

<一>. 使用list：  
 可以使用列表模拟队列(queue)、堆栈(stack)，并且支持双向的操作(L或者R)。  
  1. 右边入队：

1. jedis.rpush("userList", "James");

  2. 左边出队：右边出栈(rpop)，即为对堆栈的操作。

1. jedis.lpop("userList");

  3. 返回列表范围：从0开始，到最后一个(-1) [包含]

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

   Redis的TopN操作，即使用list完成：lrange  
 4. 删除：使用key

1. jedis.del("userList");

  5. 设置：位置1处为新值

1. jedis.lset("userList", 1, "Nick Xu");

  6. 返回长度：

1. Long size = jedis.llen("userList");

  7. 进行裁剪：包含

1. jedis.ltrim("userList", 1, 2);

 <二>. 使用set：和列表不同，集合中的元素是无序的，因此元素也不能重复。  
  1. 添加到set：可一次添加多个

1. jedis.sadd("fruit", "apple");
2. jedis.sadd("fruit", "pear", "watermelon");
3. jedis.sadd("fruit", "apple");  //no

  2. 遍历集合：

1. Set<String> fruit = jedis.smembers("fruit");

  3. 移除元素：remove

1. jedis.srem("fruit", "pear");

  4. 返回长度：

1. Long size = jedis.scard("fruit");

  5. 是否包含：

1. Boolean isMember = jedis.sismember("fruit", "pear");

  6. 集合的操作：包括集合的交运算(sinter)、差集(sdiff)、并集(sunion)

1. jedis.sadd("food", "bread", "milk");
2. Set<String> fruitFood = jedis.sunion("fruit", "food");

<三>. 使用sorted set：有序集合在集合的基础上，增加了一个用于排序的参数。  
 1. 有序集合：根据“第二个参数”进行排序。

1. jedis.zadd("user", 22, "James");

 2. 再次添加：元素相同时，更新为当前的权重。

1. jedis.zadd("user", 24, "James");

  3. zset的范围：找到从0到-1的所有元素。

1. Set<String> user = jedis.zrange("user", 0, -1);

  4. 说明：我们可能还有一个疑虑，集合是怎么做到有序的呢？  
   实际上，上述user的数据类型为java.util.LinkedHashSet  
     
<四>. 使用hash：  
 1. 存放数据：使用HashMap

1. Map<String, String>  capital = **new** HashMap<String, String>();
2. capital.put("shannxi", "xi'an");
3. ...
4. jedis.hmset("capital", capital);

   2. 获取数据：

1. List<String> cities = jedis.hmget("capital", "shannxi", "shanghai");

<五>. 其他操作：  
 1. 对key的操作：  
  @ 对key的模糊查询：

1. Set<String> keys = jedis.keys("\*");
2. Set<String> keys = jedis.keys("user.userid.\*");

  @ 删除key：

1. jedis.del("city");

  @ 是否存在：

1. Boolean isExists = jedis.exists("user.userid.14101");

  2. 失效时间：  
  @ expire：时间为5s

1. jedis.setex("user.userid.14101", 5, "James");

  @ 存活时间(ttl)：time to live

1. Long seconds = jedis.ttl("user.userid.14101");

  @ 去掉key的expire设置：不再有失效时间

1. jedis.persist("user.userid.14101");

 3. 自增的整型：  
  @ int类型采用string类型的方式存储：

1. jedis.set("amount", 100 + "");

  @ 递增或递减：incr()/decr()

1. jedis.incr("amount");

  @ 增加或减少：incrBy()/decrBy()

1. jedis.incrBy("amount", 20);

 4. 数据清空：  
  @ 清空当前db：

1. jedis.flushDB();

    @ 清空所有db：

1. jedis.flushAll();

  5. 事务支持：  
  @ 获取事务：

1. Transaction tx = jedis.multi();

  @ 批量操作：tx采用和jedis一致的API接口

1. **for**(**int** i = 0;i < 10;i ++) {
2. tx.set("key" + i, "value" + i);
3. System.out.println("--------key" + i);
4. Thread.sleep(1000);
5. }

  @ 执行事务：针对每一个操作，返回其执行的结果，成功即为Ok

1. List<Object> results = tx.exec();

使用Jedis实现客户端Sharding

<一>. 背景介绍：  
 1. sharding机制：即通常所说的“分片”，允许数据存放在不同的物理机器上，以适应数据量过大的场景，克服单台机器内存或者磁盘空间的限制。而这种“离散式”地存放，对客户端来说是透明的，对客户端来讲，完全看不到这种差别。  
 2. 常见的内存缓存中间件，比如Memcached支持服务器端Sharding，客户端根本无须关心具体的实现细节。而Redis并不支持服务器端分片，不过我们可以使用Jedis提供的API来实现客户端的分片，通过“一致性hash”算法，使得数据离散地存放在不同的服务器上面。  
 3. 对客户端来说，这是透明的，数据具体存在在那一台机器上面，对客户端来讲是无差别的。然后，通过不同机器上面的Redis控制台，我们还是可以看出数据的存储情况。  
 4. 优缺点：使用Sharding机制，克服了单台服务器的“内存限制”，但是不可避免的降低了数据的存储和访问效率。

<二>. 进行配置：  
  1. 增加节点：去掉之前的主从配置，作为用作Sharding的一台服务器，增加属性设置

1. redis.host2=192.168.142.34

  2. 连接池配置：使用shardedJedisPool

1. **<bean** id = "shardedJedisPool" class = "redis.clients.jedis.ShardedJedisPool"**>**
2. **<constructor-arg** index="0" ref="jedisPoolConfig"**/>**
3. **<constructor-arg** index="1"**>**
4. **<list>**
5. **<bean** class="redis.clients.jedis.JedisShardInfo"**>**
6. **<constructor-arg** index="0" value="${redis.host}"**/>**
7. **<constructor-arg** index="1" value="${redis.port}" type="int"**/>**
8. **<constructor-arg** index="2" value="${redis.timeout}" type="int"**/>**
9. **<property** name="password" value="${redis.password}"**/>**
10. **</bean>**
11. **<bean** class="redis.clients.jedis.JedisShardInfo"**>**
12. **<constructor-arg** index="0" value="${redis.host2}"**/>**
13. **<constructor-arg** index="1" value="${redis.port}" type="int"**/>**
14. **<constructor-arg** index="2" value="${redis.timeout}" type="int"**/>**
15. **<property** name="password" value="${redis.password}"**/>**
16. **</bean>**
17. **</list>**
18. **</constructor-arg>**
19. **</bean>**

 <三>. 使用API编程：  
  1. 获取shardedJedis：

1. ShardedJedisPool shardedPool = (ShardedJedisPool)context.getBean("shardedJedisPool");
2. ShardedJedis shardedJedis = shardedPool.getResource();
3. ...
4. shardedPool.returnResource(shardedJedis);

  2. 存储/访问/删除数据：

1. shardedJedis.set("president", "Obama");
2. String president = shardedJedis.get("president");
3. shardedJedis.del("president");