# 微服务

微服务架构成了当下的技术热点，实现微服务是要付出很大成本的，但也许是因为微服务的优点太过于吸引人，以至于大部分开发者都将它当成未来的发展趋势。

**微服务架构的演进过程**



## orm – 单一应用架构

 一个高内聚版本，所有功能部署在一起。数据访问框架（orm）成为关键。这个架构很少被人使用，几乎接近灭绝了吧。  
        优点：成本低，适合功能少又简单 缺点：很多，比如无法适应高流量，二次开发难，部署成本高

## mvc架构  － 垂直应用架构

当访问量渐渐增大，慢慢演化成用的很多的mvc架构。虽然还是所有的功能都是部署在同一个进程中，但是可以通过双机或者前置负载均衡来实现负载分流。这样应用也可以拆分成不同的几个应用，以提升性能和效率。  
        此时，mvc架构用于分离前后端逻辑。一方面，有一定的模块化。另一方面，加速和方便了开发。

## rpc架构-分布式服务架构

   当mvc垂直应用分成不同应用时，越来越多的情况下。不可避免的是应用a与应用b之间的交互。此时将核心和公共的 业务功能抽出来，作为单独的服务，并实现前后端逻辑分离。  
      此时则就需要提高业务的复用及整合的分布式rpc框架，例如dubbo等。

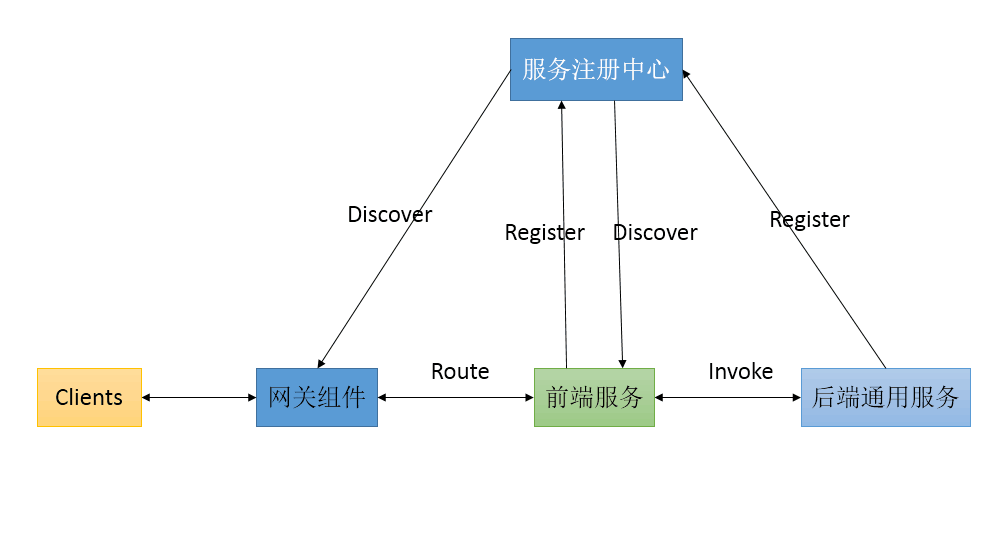
## soa架构 － 流动计算架构

   当rpc架构中的服务越来越多时，服务的生命周期的管控，容量的评估等各种问题会出现，使服务化成为瓶颈。需要增加一个调度中心来进行对服务管控，监督等。

## 微服务架构

 在soa的基础上，人们又提出了微服务架构，它就是将功能分散到各个离散的服务中然后实现对方案的解耦。服务更原子，自治更小，然后高密度部署服务。

**微服务的技术关注点**



# Dubbo

官网地址：<http://dubbo.io/>

## dubbo-admin

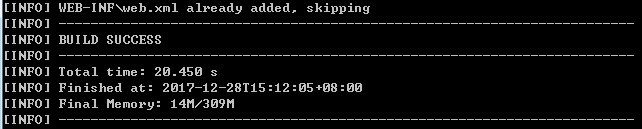
### 生成war包

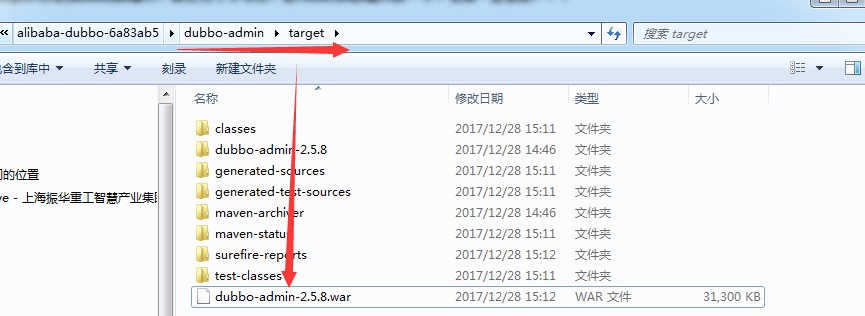
因为这里只需要编译Dubbo-Admin，所以打开Cmd,然后进入解压路径 E:\下载\dubbo-master\dubbo-master\dubbo-admin,有人把整个工程都编译了，其实没有必要，只需要编译dubbo-admin即可，有什么依赖的，maven会自动帮你解决。整个过程如下：

命令如下：mvn package -Dmaven.skip.test=true

这里要自己把maven的环境变量配置好，这里的-Dmaven.skip.test表示不打包测试包。然后结果如下，表示打包成功







### 安装dubbo-admin

（1）将war包放入tomcat的webapps目录下

（2）然后打开tomcat的conf目录下的server.xml文件，把启动端口改成8090，因为zookeeper默认使用的是8080，以免冲突.

（3）打开刚刚tomcat解压生成的dubbo.properties，我的在D:\07\_Architect\02\_document\apache-tomcat-8.5.8\webapps\dubbo-admin-2.5.8\WEB-INF下：

内容是：

dubbo.registry.address=zookeeper://127.0.0.1:2181

dubbo.admin.root.password=root

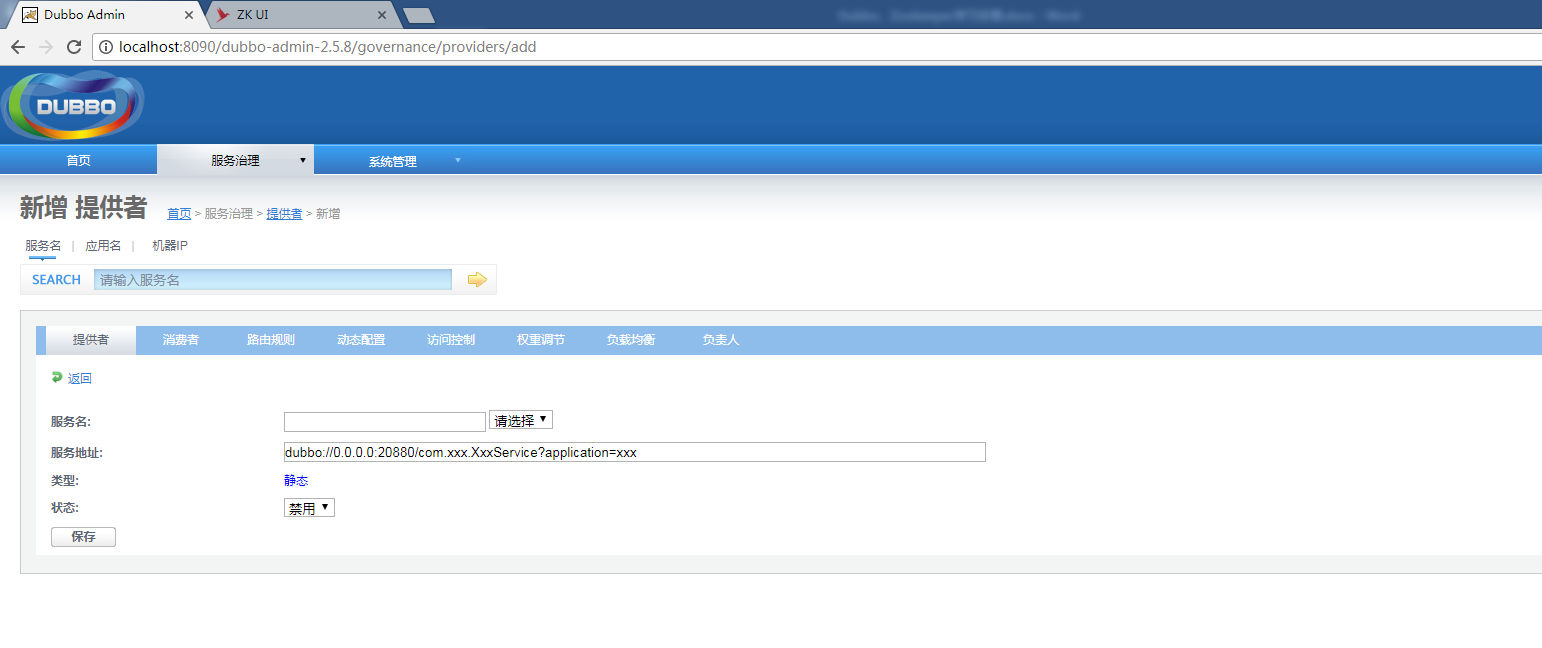
dubbo.admin.guest.password=guest

### 启动dubbo-admin

<http://localhost:8090/dubbo-admin-2.5.8>

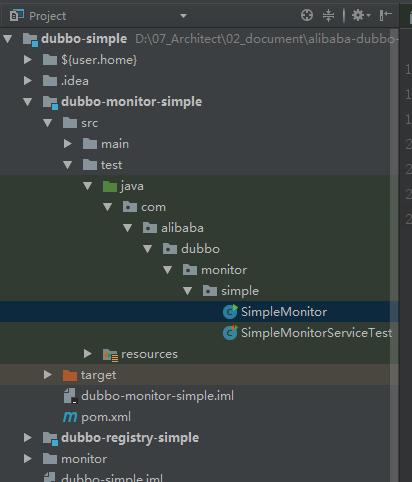
用户名：root

密码:root



## Dubbo-monitor

### 下载dubbo-monitor-simple项目



### 修改test目录下的dubbo.properties文件如下

|  |
| --- |
| dubbo.container=log4j,spring,registry,jetty dubbo.application.name=simple-monitor dubbo.application.owner= #dubbo.registry.address=multicast://224.5.6.7:1234 dubbo.registry.address=zookeeper://127.0.0.1:2181 #dubbo.registry.address=redis://127.0.0.1:6379 #dubbo.registry.address=dubbo://127.0.0.1:9090 dubbo.protocol.port=7070 dubbo.jetty.port=8080 dubbo.jetty.directory=D:/07\_Architect/02\_document/alibaba-dubbo-dubbo-2.5.8-58-g72cecba/alibaba-dubbo-72cecba/dubbo-simple/monitor dubbo.charts.directory =${dubbo.jetty.directory}/charts1 dubbo.statistics.directory=${dubbo.jetty.directory}/statistics #dubbo.log4j.file=logs/dubbo-demo-consumer.log #dubbo.log4j.level=WARN |

### 运行SimpleMonitor，并在浏览器中访问

http://localhost:8080/

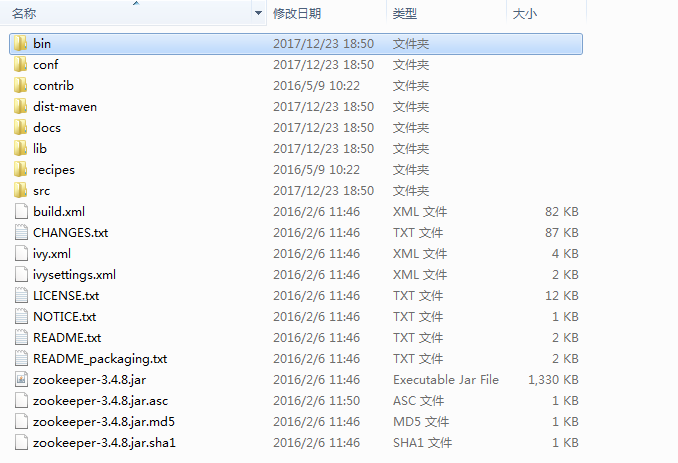
# Zookeeper

<https://zookeeper.apache.org/releases.html>

## 系统要求

ZooKeeper是用Java编写的，运行在Java环境上，因此，在部署zk的机器上需要安装Java运行环境。为了正常运行zk，我们需要JRE1.6或者以上的版本。   
 对于集群模式下的ZooKeeper部署，3个ZooKeeper服务进程是建议的最小进程数量，而且不同的服务进程建议部署在不同的物理机器上面，以减少机器宕机带来的风险，以实现ZooKeeper集群的高可用。   
 ZooKeeper对于机器的硬件配置没有太大的要求。例如，在Yahoo!内部，ZooKeeper部署的机器其配置通常如下：双核处理器，2GB内存，80GB硬盘。

## 目录



ZooKeeper软件的文件和目录

* bin目录   
  zk的可执行脚本目录，包括zk服务进程，zk客户端，等脚本。其中，.sh是Linux环境下的脚本，.cmd是Windows环境下的脚本。
* conf目录   
  配置文件目录。zoo\_sample.cfg为样例配置文件，需要修改为自己的名称，一般为zoo.cfg。log4j.properties为日志配置文件。
* lib   
  zk依赖的包。
* contrib目录   
  一些用于操作zk的工具包。
* recipes目录   
  zk某些用法的代码示例

## 单机模式

ZooKeeper的安装包括单机模式安装，以及集群模式安装。

单机模式较简单，是指只部署一个zk进程，客户端直接与该zk进程进行通信。

在开发测试环境下，通过来说没有较多的物理资源，因此我们常使用单机模式。当然在单台物理机上也可以部署集群模式，但这会增加单台物理机的资源消耗。故在开发环境中，我们一般使用单机模式。

但是要注意，生产环境下不可用单机模式，这是由于无论从系统可靠性还是读写性能，单机模式都不能满足生产的需求。

### 运行配置

上面提到，conf目录下提供了配置的样例zoo\_sample.cfg，要将zk运行起来，需要将其名称修改为zoo.cfg。

打开zoo.cfg，可以看到默认的一些配置。

（1）tickTime

时长单位为毫秒，为zk使用的基本时间度量单位。例如，1 \* tickTime是客户端与zk服务端的心跳时间，2 \* tickTime是客户端会话的超时时间。

tickTime的默认值为2000毫秒，更低的tickTime值可以更快地发现超时问题，但也会导致更高的网络流量（心跳消息）和更高的CPU使用率（会话的跟踪处理）。

（2）clientPort

zk服务进程监听的TCP端口，默认情况下，服务端会监听2181端口。

（3）dataDir

无默认配置，必须配置，用于配置存储快照文件的目录。如果没有配置dataLogDir，那么事务日志也会存储在此目录。

### 启动

在Windows环境下，直接双击zkServer.cmd即可。在Linux环境下，进入bin目录，执行命令

./zkServer.sh start

这个命令使得zk服务进程在后台进行。如果想在前台中运行以便查看服务器进程的输出日志，可以通过以下命令运行：

./zkServer.sh start-foreground

执行此命令，可以看到大量详细信息的输出，以便允许查看服务器发生了什么。

### 连接

如果是连接同一台主机上的zk进程，那么直接运行bin/目录下的zkCli.cmd（Windows环境下）或者zkCli.sh（Linux环境下），即可连接上zk。

直接执行zkCli.cmd或者zkCli.sh命令默认以主机号 127.0.0.1，端口号 2181 来连接zk，如果要连接不同机器上的zk，可以使用 -server 参数，例如：

bin/zkCli.sh -server 192.168.0.1:2181

## 集群模式

单机模式的zk进程虽然便于开发与测试，但并不适合在生产环境使用。在生产环境下，我们需要使用集群模式来对zk进行部署。

注意

在集群模式下，建议至少部署3个zk进程，或者部署奇数个zk进程。如果只部署2个zk进程，当其中一个zk进程挂掉后，剩下的一个进程并不能构成一个quorum的大多数。因此，部署2个进程甚至比单机模式更不可靠，因为2个进程其中一个不可用的可能性比一个进程不可用的可能性还大。

### 运行配置

在集群模式下，所有的zk进程可以使用相同的配置文件（是指各个zk进程部署在不同的机器上面），例如如下配置：

tickTime=2000

dataDir=/home/myname/zookeeper

clientPort=2181

initLimit=5

syncLimit=2

server.1=192.168.229.160:2888:3888

server.2=192.168.229.161:2888:3888

server.3=192.168.229.162:2888:3888

initLimit

ZooKeeper集群模式下包含多个zk进程，其中一个进程为leader，余下的进程为follower。

当follower最初与leader建立连接时，它们之间会传输相当多的数据，尤其是follower的数据落后leader很多。initLimit配置follower与leader之间建立连接后进行同步的最长时间。

syncLimit

配置follower和leader之间发送消息，请求和应答的最大时间长度。

tickTime

tickTime则是上述两个超时配置的基本单位，例如对于initLimit，其配置值为5，说明其超时时间为 2000ms \* 5 = 10秒。

server.id=host:port1:port2

其中id为一个数字，表示zk进程的id，这个id也是dataDir目录下myid文件的内容。

host是该zk进程所在的IP地址，port1表示follower和leader交换消息所使用的端口，port2表示选举leader所使用的端口。

dataDir

其配置的含义跟单机模式下的含义类似，不同的是集群模式下还有一个myid文件。myid文件的内容只有一行，且内容只能为1 - 255之间的数字，这个数字亦即上面介绍server.id中的id，表示zk进程的id。

注意

如果仅为了测试部署集群模式而在同一台机器上部署zk进程，server.id=host:port1:port2配置中的port参数必须不同。但是，为了减少机器宕机的风险，强烈建议在部署集群模式时，将zk进程部署不同的物理机器上面。

### 启动

假如我们打算在三台不同的机器 192.168.229.160，192.168.229.161，192.168.229.162上各部署一个zk进程，以构成一个zk集群。

三个zk进程均使用相同的 zoo.cfg 配置：

tickTime=2000

dataDir=/home/myname/zookeeper

clientPort=2181

initLimit=5

syncLimit=2

server.1=192.168.229.160:2888:3888

server.2=192.168.229.161:2888:3888

server.3=192.168.229.162:2888:3888

在三台机器dataDir目录（ /home/myname/zookeeper 目录）下，分别生成一个myid文件，其内容分别为1，2，3。然后分别在这三台机器上启动zk进程，这样我们便将zk集群启动了起来。

### 连接

可以使用以下命令来连接一个zk集群：

bin/zkCli.sh -server 192.168.229.160:2181,192.168.229.161:2181,192.168.229.162:2181

成功连接后，可以看到如下输出：

2016-06-28 19:29:18,074 [myid:] - INFO [main:ZooKeeper@438] - Initiating client connection, connectString=192.168.229.160:2181,192.168.229.161:2181,192.168.229.162:2181 sessionTimeout=30000 watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@770537e4

Welcome to ZooKeeper!

2016-06-28 19:29:18,146 [myid:] - INFO [main-SendThread(192.168.229.162:2181):ClientCnxn$SendThread@975] - Opening socket connection to server 192.168.229.162/192.168.229.162:2181. Will not attempt to authenticate using SASL (unknown error)

JLine support is enabled

2016-06-28 19:29:18,161 [myid:] - INFO [main-SendThread(192.168.229.162:2181):ClientCnxn$SendThread@852] - Socket connection established to 192.168.229.162/192.168.229.162:2181, initiating session

2016-06-28 19:29:18,199 [myid:] - INFO [main-SendThread(192.168.229.162:2181):ClientCnxn$SendThread@1235] - Session establishment complete on server 192.168.229.162/192.168.229.162:2181, sessionid = 0x3557c39d2810029, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null

[zk: 192.168.229.160:2181,192.168.229.161:2181,192.168.229.162:2181(CONNECTED) 0]

图2：客户端连接zk集群的输出日志

从日志输出可以看到，客户端连接的是192.168.229.162:2181进程（连接上哪台机器的zk进程是随机的），客户端已成功连接上zk集群。

## 参考资料

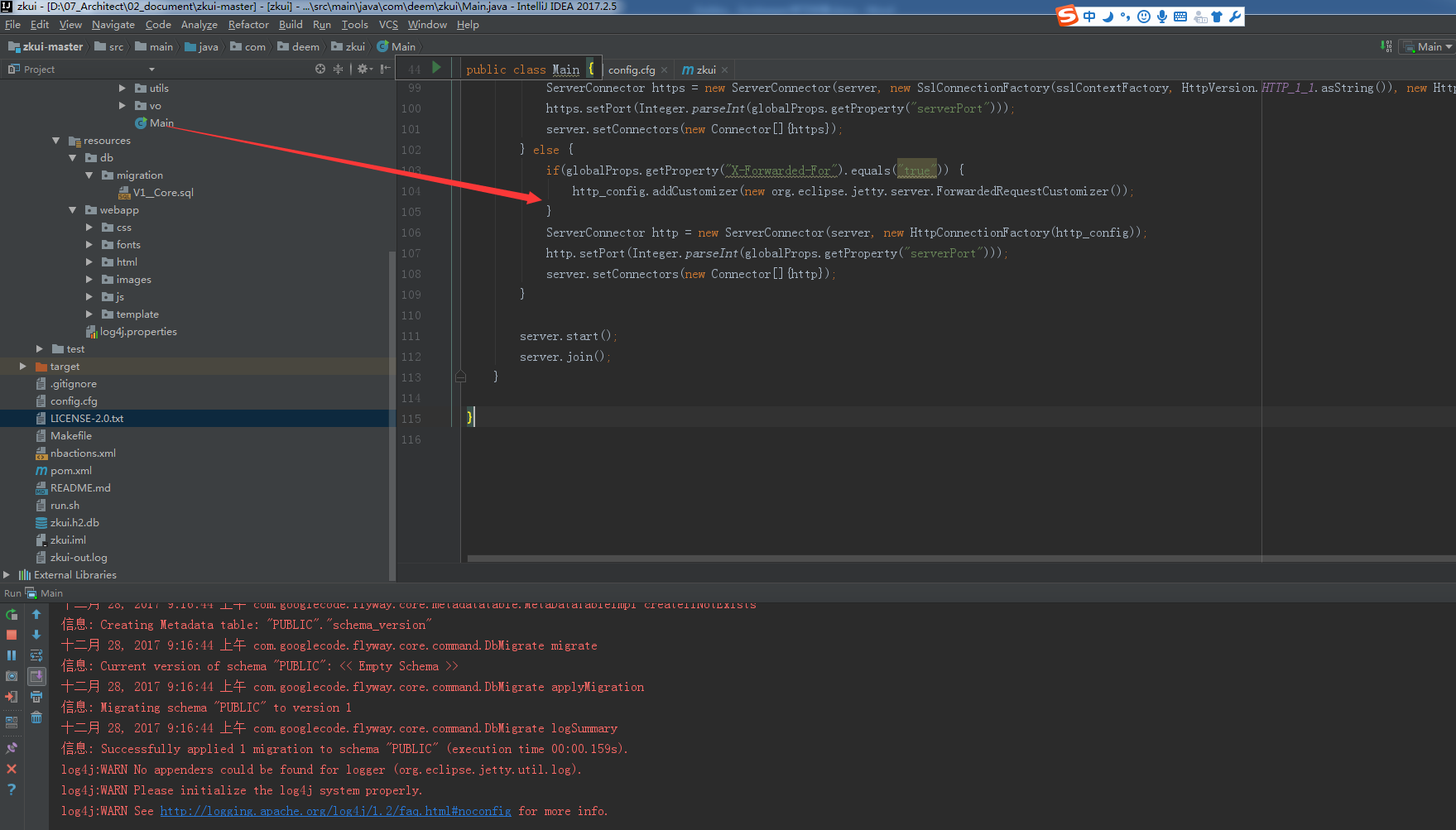
<http://blog.csdn.net/lihao21/article/details/51778255>

http://coolxing.iteye.com/blog/1871009 《Zookeeper 安装和配置》

<http://blog.csdn.net/kongxx/article/details/41015817> Zookeeper单机模式启动

# Zkui

A UI dashboard that allows CRUD operations on Zookeeper.



<http://localhost:9090/>

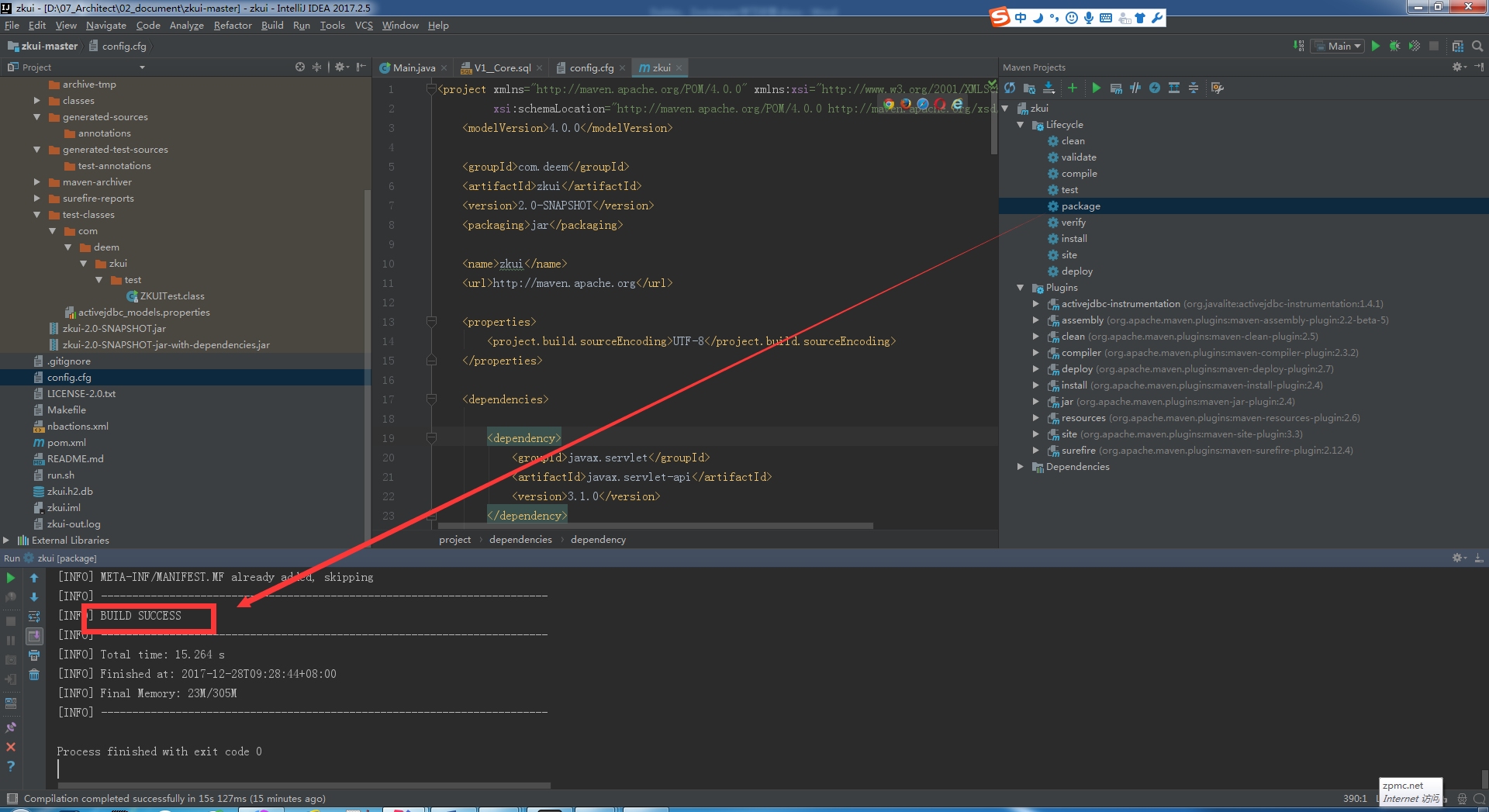
用户名：admin

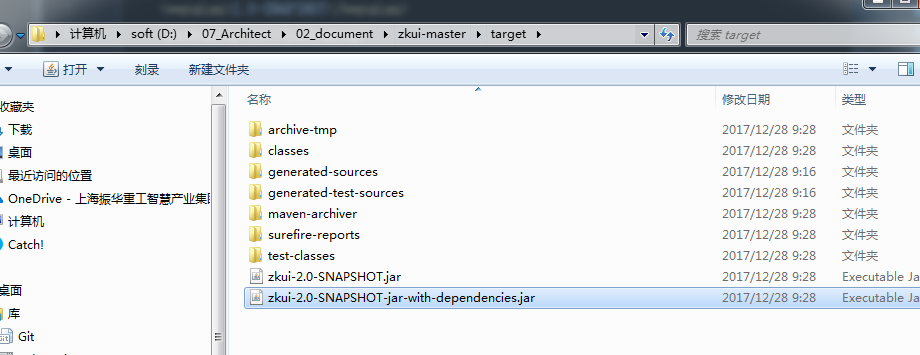
密码：manager

## 下载源码

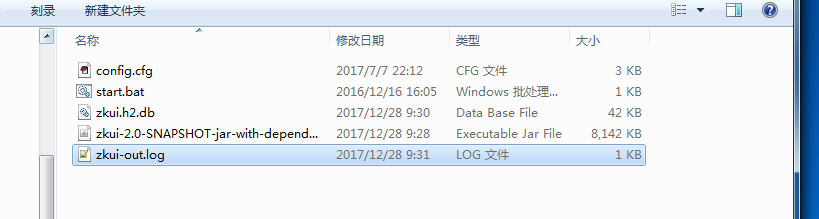
https://github.com/DeemOpen/zkui

## 打包成jar





## 制作启动工具



# Redis

## 简介

Redis是完全开源免费的，遵守BSD协议，是一个高性能的key-value数据库

BSD协议：开源协议是一个给于使用者很大自由的协议。可以自由的使用，修改源代码，也可以将修改后的代码作为开源或者专有软件再发布

Rdis与其他key-value缓存产品有以下三个特点：

（1）Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

（2）Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

（3）Redis支持数据的备份，即master-slave模式的数据备份

**Redis优势：**

（1）性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。

（2）丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

（3）原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。

（4）丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

## 安装

<https://github.com/MicrosoftArchive/redis/releases>

默认是在windows服务中启动服务，如果想卸载此服务CMD下执行：

sc delete Redis

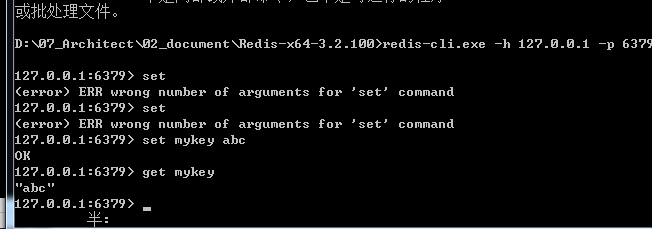
### 启动Server

打开cmd 切换到redis解压目录：运行redis-server.exe redis.windows.conf



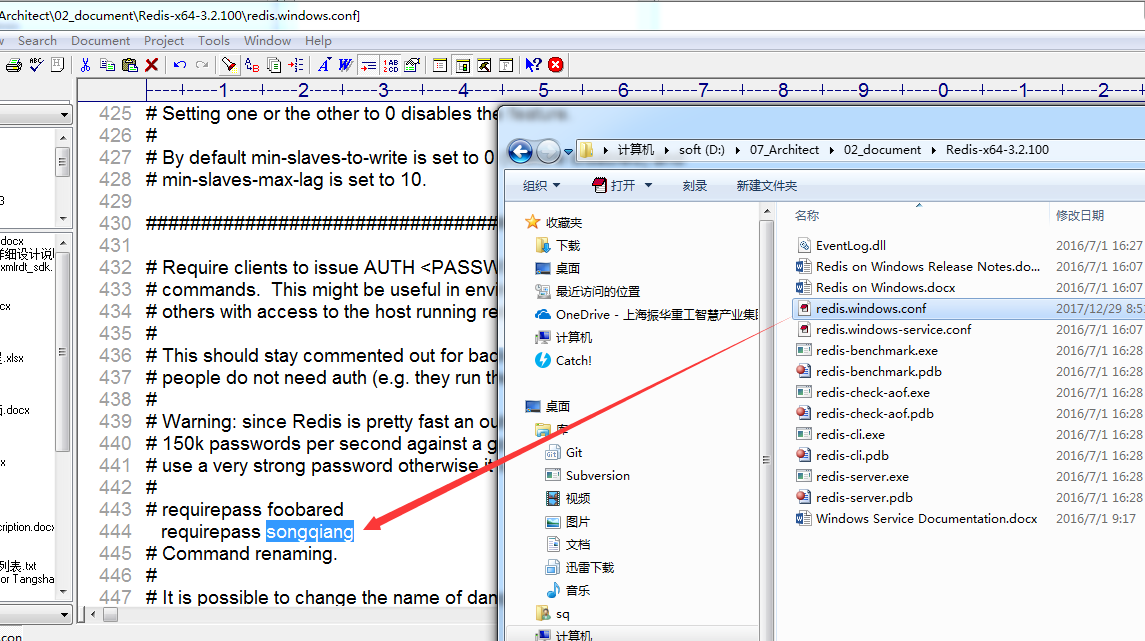
### 启动Client

Server端cmd 窗口不要关闭，切换到redis目录下运行 redis-cli.exe -h 127.0.0.1 -p 6379



### 设置密码

#### 通过配置文件修改



#### 通过命令行进行配置

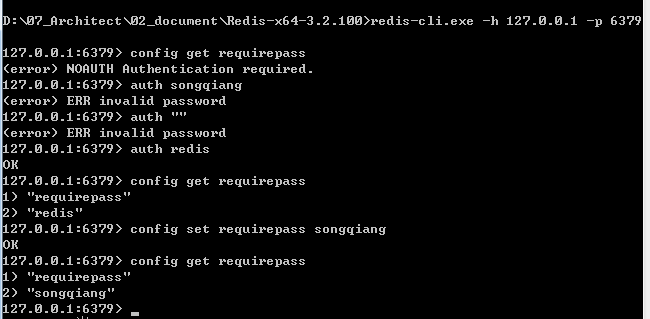
redis-cli.exe -h 127.0.0.1 -p 6379

config get requirepass

auth redis

config get rquirepass

config set requirepass songqiang

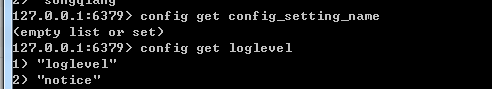


## 配置

redis 127.0.0.1:6379> CONFIG GET CONFIG\_SETTING\_NAME

redis 127.0.0.1:6379> CONFIG GET loglevel

redis 127.0.0.1:6379> CONFIG GET \*



## 数据类型

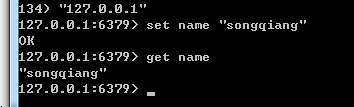
Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zset(sorted set：有序集合)。

### String

string是redis最基本的类型，你可以理解成与Memcached一模一样的类型，一个key对应一个value。

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

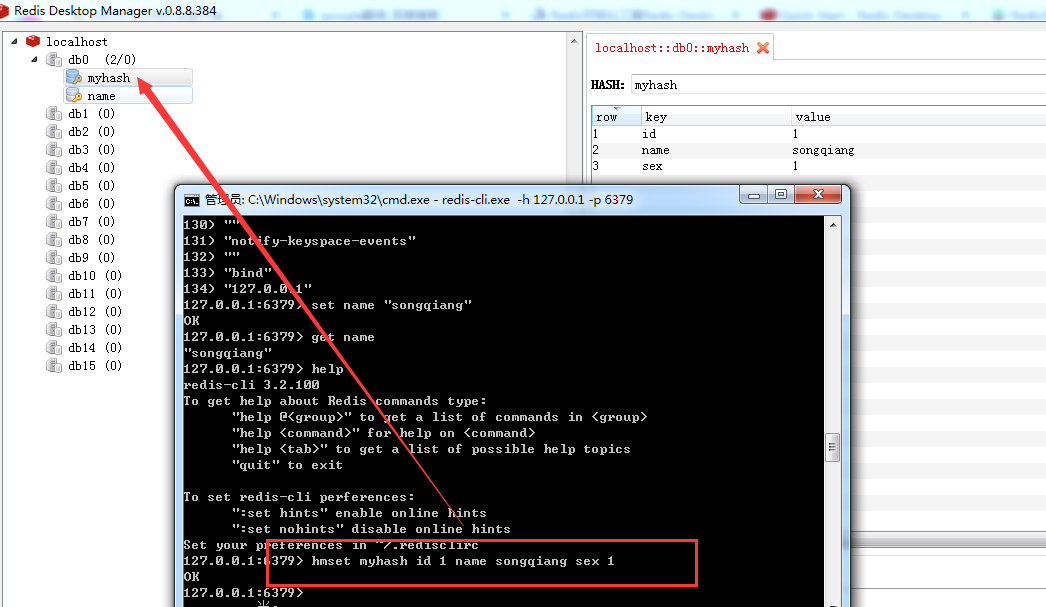
string类型是Redis最基本的数据类型，一个键最大能存储512MB。



### Hash（哈希）

Redis hash 是一个键值(key=>value)对集合。

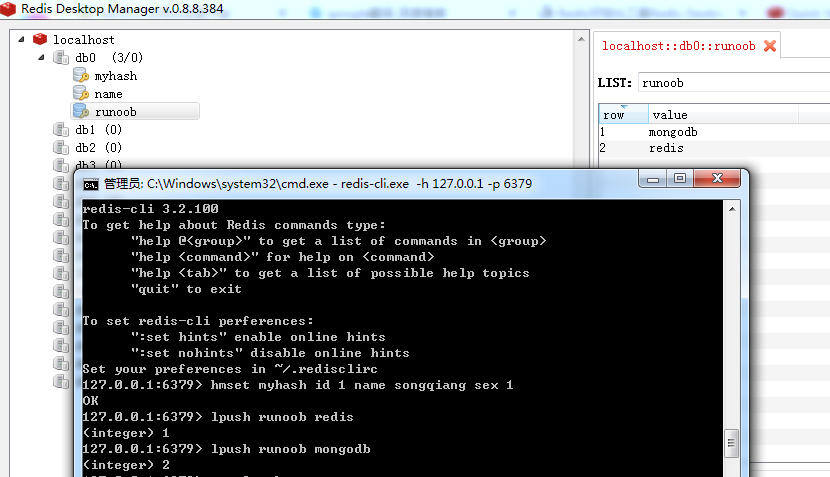
Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。



### List（列表）

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）。

命令：lpush runoob redis



### Set（集合）

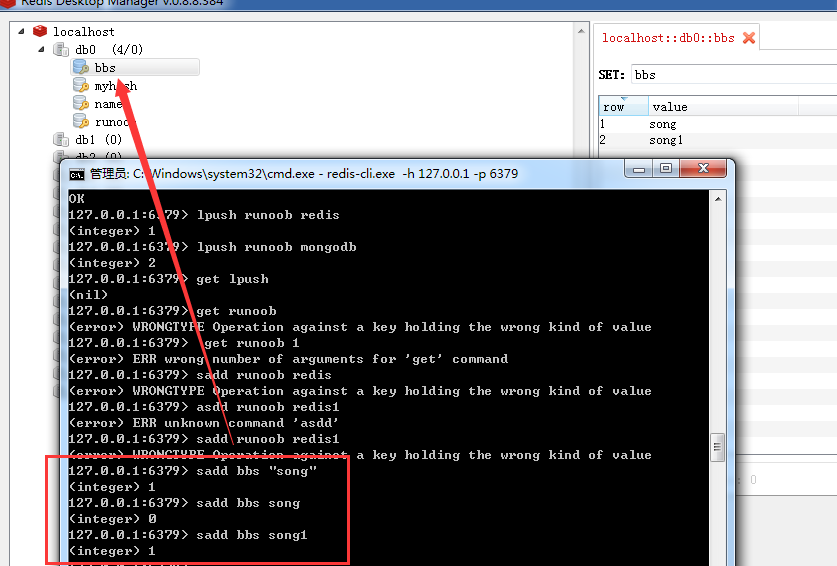
Redis的Set是string类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

**sadd 命令：**sadd key member 添加

smembers key 查看set集合

添加一个string元素到,key对应的set集合中，成功返回1,如果元素已经在集合中返回0,key对应的set不存在返回错误。



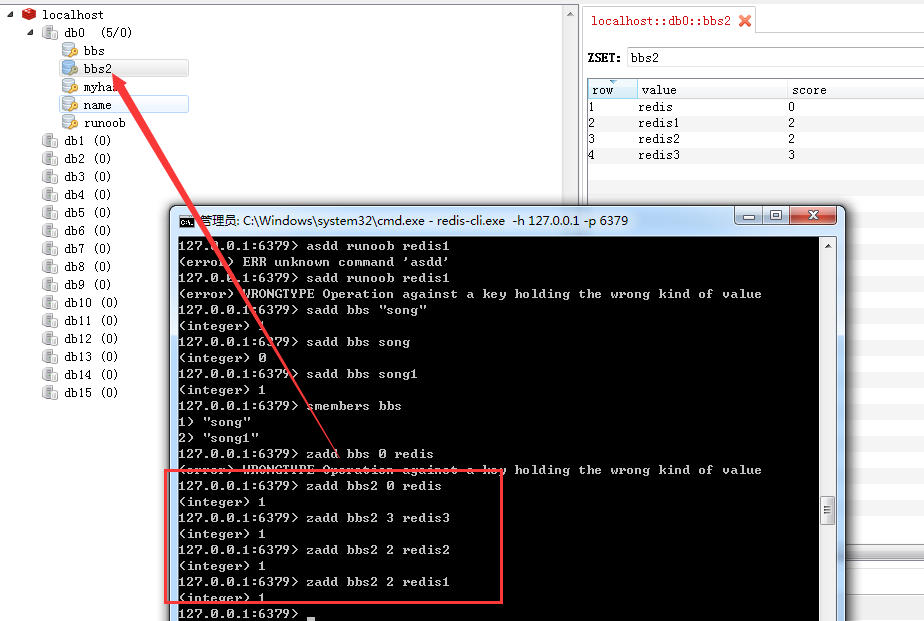
### ZSET（sorted set：有序集合）

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。

**命令：zadd key score member**

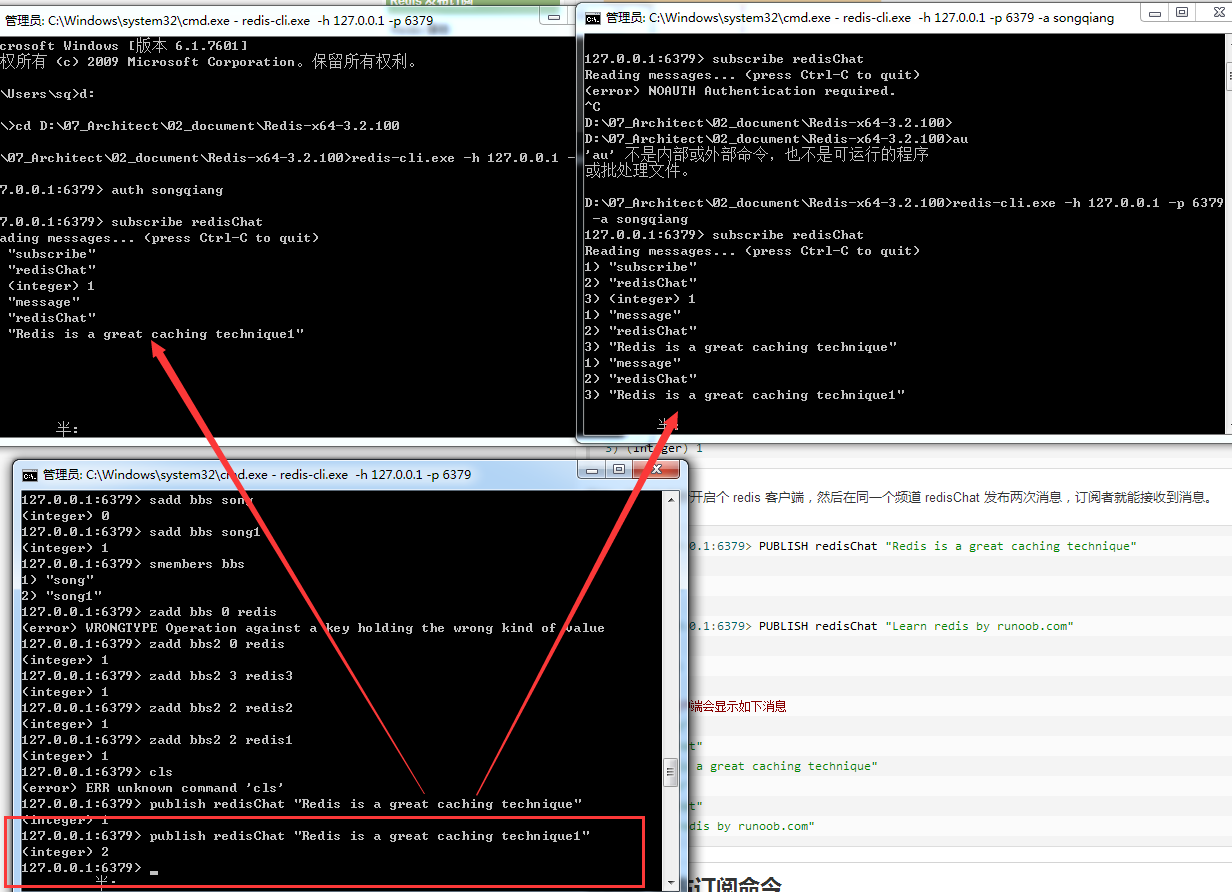


### pub/sub（发布/订阅）

Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。

命令：SUBSCRIBE redisChat 订阅者

PUBLISH redisChat "Redis is a great caching technique" 发布者



### 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：**

批量操作在发送 EXEC 命令前被放入队列缓存。

收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行。

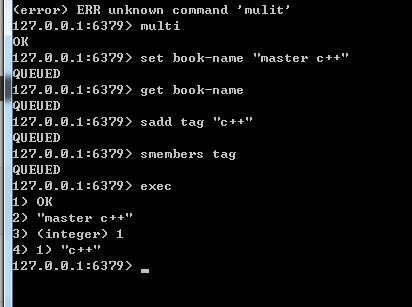
在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。

**一个事务从开始到执行会经历以下三个阶段：**

开始事务。

命令入队。

执行事务。



### 脚本

### 连接

Redis 连接命令主要是用于连接 redis 服务。

**命令：AUTH "password"**

**Ping**

### 服务器

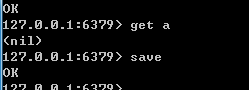
## Redis高级教程

### 数据备份与恢复

Redis **SAVE** 命令用于创建当前数据库的备份。

**命令：redis 127.0.0.1:6379> SAVE**

该命令将在 redis 安装目录中创建dump.rdb文件



恢复数据

如果需要恢复数据，只需将备份文件 (dump.rdb) 移动到 redis 安装目录并启动服务即可。获取 redis 目录可以使用 **CONFIG** 命令

命令：查看路径：CONFIG GET dir

### 安全

### 性能测试

## Java使用Redis

|  |
| --- |
| package com; import redis.clients.jedis.Jedis;  */\*\*  \* redistest  \*  \** ***@author*** *songqiang  \** ***@create*** *2017-12-29 23:04  \*\*/* public class RedisTest {  public static void main(String[] args){  Jedis jedis = new Jedis("localhost");  jedis.auth("redis");  System.*out*.println("连接成功");  //查看服务是否运行  System.*out*.println("服务正在运行: "+jedis.ping());  } } |

## 参考资料

<http://www.runoob.com/redis/redis-install.html> Redis菜鸟教程

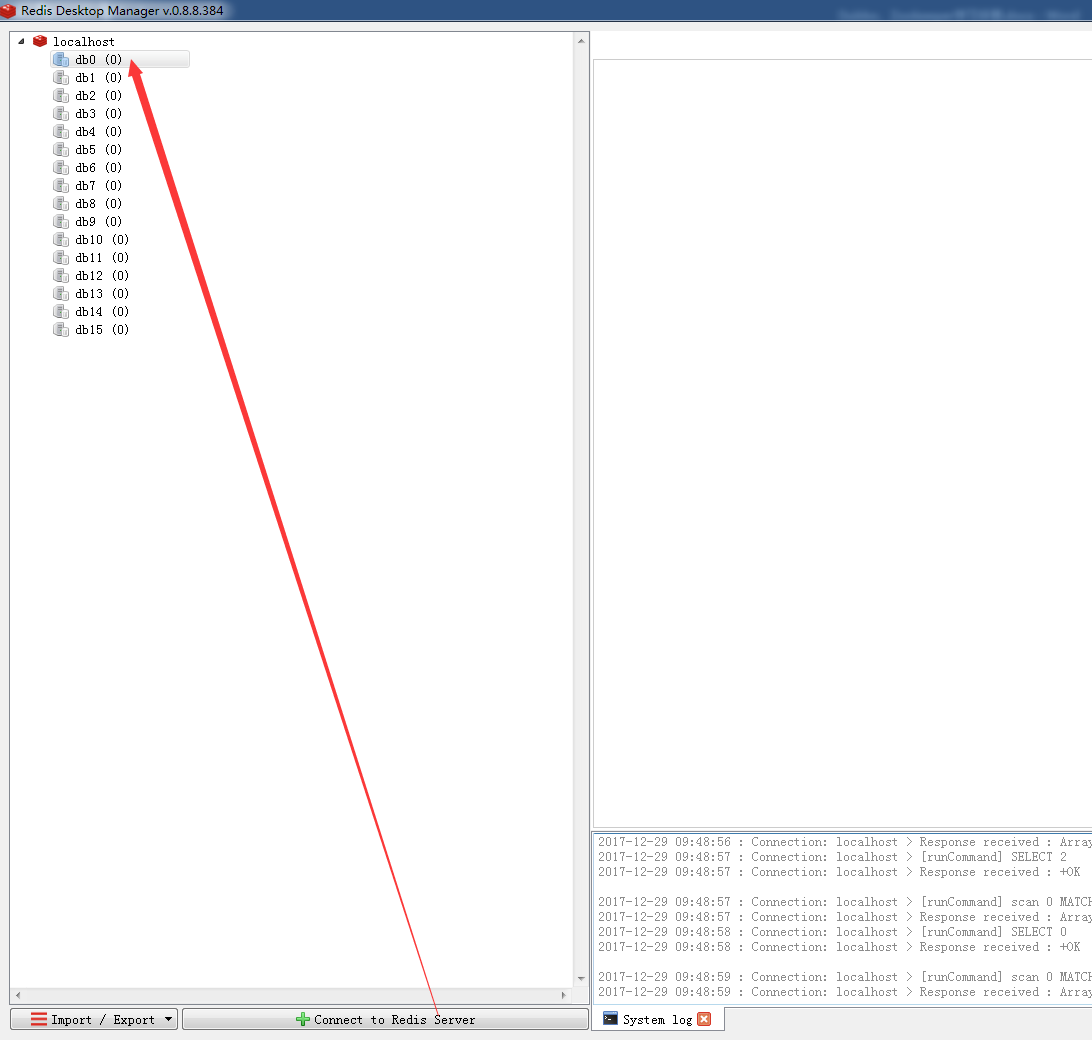
<http://doc.redisfans.com/set/sadd.html> Redis命令参考

# Redis可视化工具（Redis Desktop Manager）

源码下载地址：<https://github.com/uglide/RedisDesktopManager>

安装下载地址：<https://redisdesktop.com/download>

## Windows安装



# 服务降级及dubbo中的实现示例

## 介绍

经历过12306抢票的人应该经常会遇到这个问题：在抢票高峰的时候，明明票还有，但是查询出来的列表却是为空的（如果没票列表也应该会呈现）；等高峰过后再查询，列表又恢复正常。个人猜测应该是查询过程中出现了问题，要么超时，要么网络问题导致查询失败采用的服务降级处理。所以，最终呈现给用户的并不是内部系统出错之类的提示，而是一个空的列表。

**服务降级**，当服务器压力剧增时，根据当前业务情况及流量对一些服务和页面有策略的降级，以此缓解了服务器资源压力，以保证核心任务的正常运行，同时也保证了部分甚至大部分客户得到正确响应。

## 服务降级实施策略

（1）页面拒绝服务：页面提示由于服务繁忙此服务暂停。跳转到varnish或nginx的一个静态页面。

服务接口拒绝服务：无用户特定信息的页面能访问，提示服务器繁忙。页面内容也可在缓存（Varnish）或CDN内获取。

（2）延迟持久化：页面访问照常，但是涉及记录变更，会提示稍晚能看到结果，将数据记录到异步队列或log，服务恢复后执行。

（3）随机拒绝服务：服务接口随机拒绝服务，让用户重试，目前较少采用。因为用户体验不佳。

## 服务降级埋点

（1）消息中间件   
所有API调用请求可以使用消息中间件进行控制，这可以有效地处理请求流量的增长，避免服务压力过大，导致服务雪崩效应。缺点是，增加了服务处理和业务处理的复杂性。

（2）前端页面   
指定网址不可访问（NGINX等load balance来处理），处理简单。缺点是用户体验非常不好。

（3）底层数据层   
拒绝所有增删改动作，只允许查询。此时用户可以查看所有的查询处理页面和业务，但无法进行有更新修改的操作。

## 在dubbo平台上实现服务降级

dubbo开发中，可能由于服务没有启动或者网络不通，调用中会出现RpcException，也就是远程调用失败。如果是服务启动顺序的问题，可能加工check=”false”的配置可以得到很好的解决。但是，如果是服务宕掉或者并发数太高导致的RpcException该如何处理?

查看dubbo的官方文档，可以发现有个mock的配置，mock只在出现非业务异常(比如超时，网络异常等)时执行。mock的配置支持两种，一种为boolean值，默认的为false。如果配置为true，则缺省使用mock类名，即类名+Mock后缀；另外一种则是配置”return null”，可以很简单的忽略掉异常。

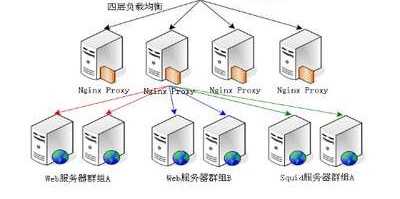
## 参考网站

<http://blog.csdn.net/whereismatrix/article/details/53354141>

# Nginx

## 反向代理

在计算机世界里，由于单个服务器的处理客户端（用户）请求能力有一个极限，当用户的接入请求蜂拥而入时，会造成服务器忙不过来的局面，可以使用多个服务器来共同分担成千上万的用户请求，这些服务器提供相同的服务，对于用户来说，根本感觉不到任何差别。



## 反向代理的实现

1）需要有一个负载均衡设备来分发用户请求，将用户请求分发到空闲的服务器上

2）服务器返回自己的服务到负载均衡设备

3）负载均衡将服务器的服务返回用户

以上的潜台词是：用户和负载均衡设备直接通信，**也意味着用户做服务器域名解析时，解析得到的IP其实是负载均衡的IP，而不是服务器的IP**，这样有一个好处是，当新加入/移走服务器时，仅仅需要修改负载均衡的服务器列表，而不会影响现有的服务。

谈完反向代理服务，再来谈谈终端用户常用的代理服务。

## 代理

1）用户希望代理服务器帮助自己，和要访问服务器通信，为了实现此目标，需要以下工作：

a) 用户IP报文的目的IP = 代理服务器IP  
b) 用户报文端口号 = 代理服务器监听端口号  
c) HTTP 消息里的URL要提供服务器的链接

2）代理服务器可以根据c)里的链接与服务器直接通信

3）服务器返回网页

4）代理服务器打包3）中的网页，返回用户。

## 代理服务器应用场景

**场景一**  
如果不采用代理，用户的IP、端口号直接暴露在Internet（尽管地址转换NAT），外部主机依然可以根据IP、端口号来开采主机安全漏洞，所以在企业网，一般都是采用代理服务器访问互联网。

**那有同学会有疑问，那代理服务器就没有安全漏洞吗？**  
相比千千万万的用户主机，代理服务器数量有限，修补安全漏洞更方便快捷。

**场景二**  
在一个家庭局域网，家长觉得外部的世界是洪水猛兽，为了不让小盆友们学坏，决定不让小盆友们访问一些网站，可小盆友们有强烈的逆反心理，侬越是不让我看，我越是想看，于是小盆友们使用了代理服务器，这些代理服务器将禁止访问的网页打包好，然后再转交给小盆友，仅此而已。