redis

**[1](#_Toc26031)** [概述 4](#_Toc26031)

**[2](#_Toc13156)** [安装 5](#_Toc13156)

**[2.1](#_Toc22006)** [windows 5](#_Toc22006)

[2.1.1 安装 5](#_Toc31644)

[2.1.2 配置环境变量 7](#_Toc4422)

**[2.2](#_Toc28772)** [linux下安装 7](#_Toc28772)

[2.2.1 安装 7](#_Toc5865)

[2.2.2 环境path 9](#_Toc11056)

**[3](#_Toc7305)** [C/S架构 9](#_Toc7305)

**[3.1](#_Toc11490)** [服务器端 10](#_Toc11490)

[3.1.1 开启 10](#_Toc16862)

[3.1.2 主要配置选项 11](#_Toc5956)

[3.1.3 管理进程 13](#_Toc26289)

[3.1.4 将启动脚本写在 rc.local(linux) 13](#_Toc9088)

**[3.2](#_Toc24063)** [客户端 14](#_Toc24063)

[3.2.1 命令行客户端 14](#_Toc17710)

[3.2.2 PHP程序 15](#_Toc20299)

**[4](#_Toc9962)** [数据(类型)操作 18](#_Toc9962)

**[4.1](#_Toc24409)** [实例化redis对象, 连接redis服务器 19](#_Toc24409)

**[4.2](#_Toc21656)** [通用操作 19](#_Toc21656)

[4.2.1 delete, 删除缓存项 20](#_Toc17385)

[4.2.2 expire, expireAt, 有效期设置, ttl有效期获取 20](#_Toc18618)

[4.2.3 exists 检测一个缓存项是否存在 21](#_Toc16943)

**[4.3](#_Toc32132)** [string字符串类型操作 21](#_Toc32132)

[4.3.1 set, 设置 21](#_Toc22039)

[4.3.2 get, 获取 22](#_Toc26521)

[4.3.3 incr, incrBy 22](#_Toc28820)

[4.3.4 decr, decrBy 22](#_Toc12174)

[4.3.5 append, 追加 23](#_Toc28048)

[4.3.6 strLen, 内容长度 23](#_Toc26791)

[4.3.7 getRange获取部分 23](#_Toc9785)

**[4.4](#_Toc13937)** [list 链表类型 24](#_Toc13937)

[4.4.1 lpush(), 将元素压入链表 24](#_Toc27314)

[4.4.2 linsert(), 在某个位置插入新元素 25](#_Toc8222)

[4.4.3 lset(), 设置某个元素的值 25](#_Toc18051)

[4.4.4 llen(), 获取列表元素个数 25](#_Toc1997)

[4.4.5 lindex(), 返回索引对应的元素 25](#_Toc19760)

[4.4.6 lrange(), 获取某个选定范围元素 26](#_Toc11570)

[4.4.7 lpop(), 从列表中, 左侧弹出元素 26](#_Toc22752)

[4.4.8 rpush(), rpop() 27](#_Toc7720)

[4.4.9 lremove, 移除元素 27](#_Toc13063)

[4.4.10 队列操作, FIFO, 先进先出 28](#_Toc19771)

**[4.5](#_Toc8028)** [set 集合类型 35](#_Toc8028)

[4.5.1 sadd, 添加元素到集合 36](#_Toc1330)

[4.5.2 spop, 弹出一个随机元素 37](#_Toc17331)

[4.5.3 sscan, 遍历集合 37](#_Toc27157)

[4.5.4 smembers, 获取所有的成员 37](#_Toc31599)

[4.5.5 scard, 获取基数 38](#_Toc31565)

[4.5.6 sunion, sdiff, sinter, 并集, 差集, 交集 38](#_Toc23458)

**[4.6](#_Toc26567)** [有序集sorted-set 39](#_Toc26567)

[4.6.1 zadd, 添加元素 40](#_Toc5401)

[4.6.2 zincrBy, 元素分值增减 40](#_Toc19532)

[4.6.3 zRange(), zRevRange() 40](#_Toc3049)

[4.6.4 zrangebyscore(), zrevrangebyscore() 41](#_Toc30052)

[4.6.5 zCard, 获取基数 元素个数 41](#_Toc30769)

[4.6.6 zscore, 获取元素的分值 41](#_Toc18811)

[4.6.7 zrem, zremrangebyrank, zremrangebyscore删除, 根据排行删除, 42](#_Toc16950)

**[4.7](#_Toc25903)** [hash-table, hash表 43](#_Toc25903)

[4.7.1 hset, 设置 44](#_Toc7163)

[4.7.2 hget, 获取 44](#_Toc736)

[4.7.3 hGetAll, 获取全部的元素 44](#_Toc11525)

[4.7.4 hexists, 判断元素是否存在 45](#_Toc20482)

[4.7.5 hdel, 删除某个元素 45](#_Toc2296)

[4.7.6 hlen, 获取长度 45](#_Toc9665)

**[4.8](#_Toc12311)** [HyperLog, 集合基准数 46](#_Toc12311)

[4.8.1 pfadd, 向集合中添加元素 46](#_Toc2033)

[4.8.2 pfcount, 统计基数 46](#_Toc4354)

**[4.9](#_Toc9319)** [bitmap, 位图 47](#_Toc9319)

[4.9.1 setbit, 设置位图的某一位 47](#_Toc32032)

[4.9.2 getbit, 获取位图的某一个位 48](#_Toc7722)

[4.9.3 get 测试 16进制 48](#_Toc19864)

**[5](#_Toc2849)** [持久化支持 49](#_Toc2849)

**[5.1](#_Toc9409)** [配置项: dbfilename 50](#_Toc9409)

**[5.2](#_Toc31)** [配置项: dir, 工作目录, rdb文件的存储位置 50](#_Toc31)

**[5.3](#_Toc22646)** [配置项: save, 持久化算法 51](#_Toc22646)

**[5.4](#_Toc9867)** [save命令 51](#_Toc9867)

**[5.5](#_Toc13237)** [还原 rdb 51](#_Toc13237)

**[5.6](#_Toc28648)** [aof 持久化 52](#_Toc28648)

[5.6.1 配置: appendonly, 是否开启AOF 52](#_Toc29288)

[5.6.2 配置项: appendfilename, AOF记录文件名称 52](#_Toc24335)

[5.6.3 appendonly.aof 53](#_Toc12367)

**[6](#_Toc23522)** [口令验证 54](#_Toc23522)

**[6.1](#_Toc3169)** [服务器配置文件配置密码 55](#_Toc3169)

**[6.2](#_Toc28077)** [客户端认证 55](#_Toc28077)

**[7](#_Toc3108)** [批处理(事务)支持 56](#_Toc3108)

**[7.1](#_Toc22941)** [multi, 开启批处理(开启事务) 57](#_Toc22941)

**[7.2](#_Toc26311)** [exec, 执行(提交) 57](#_Toc26311)

**[7.3](#_Toc19960)** [discard, 放弃(回滚) 57](#_Toc19960)

# 概述

又是一个内存缓存技术!

对比memcached, 有以下的几个典型的特征:

1, 数据类型的支持, redis内置了多种数据类型.(联想MySQL). 不同的数据类型带来了不同的业务逻辑操作上.

三个典型的使用不同的类型完成的案例:

字符串, 数据缓存(memcache能做的, redis能做)

队列, 并发排队.

有序集, 排行榜top类.

2, 持久化的支持. 内存中的数据, 可以被永久保存在磁盘上. (数据库的特性)

有些基于数据库的业务逻辑, 也可以用redis完成.

3, 是Key-value型的NOSQL

NOSQL: Not Only SQL. 非关系型数据库系统.

# 安装

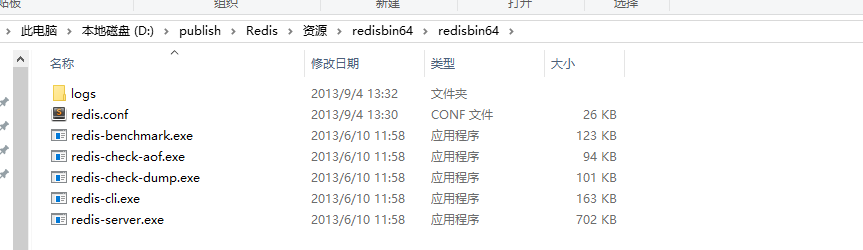
## windows

### 安装

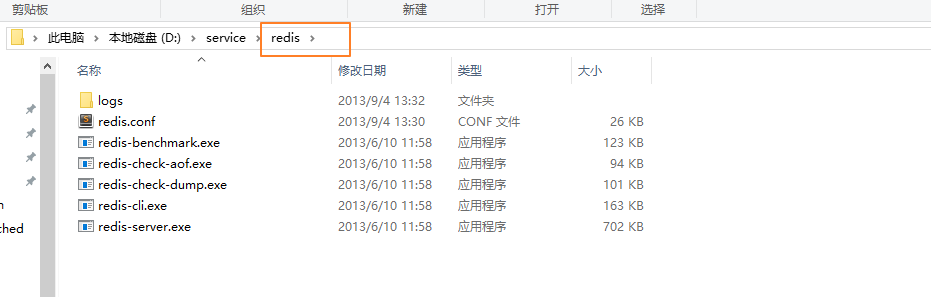
提供好的二进制执行程序



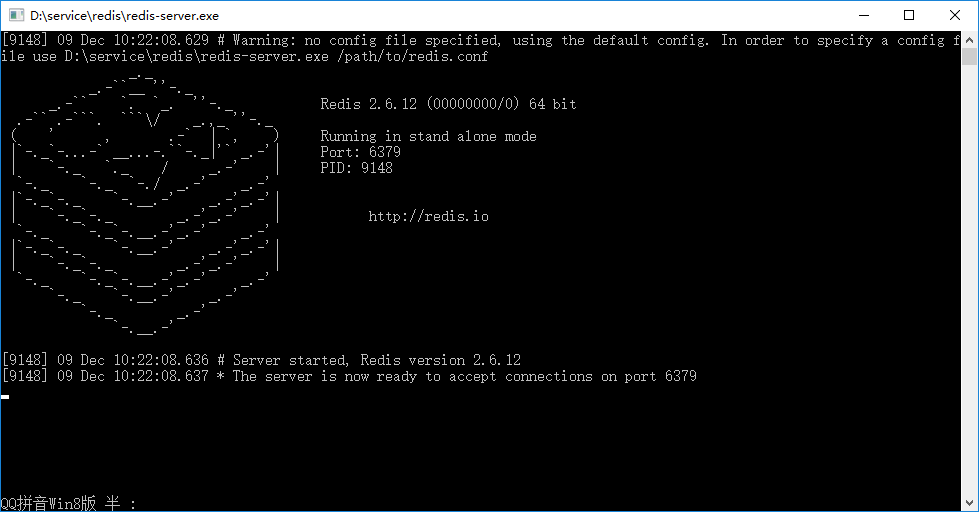
解压, 拷贝到指定的位置



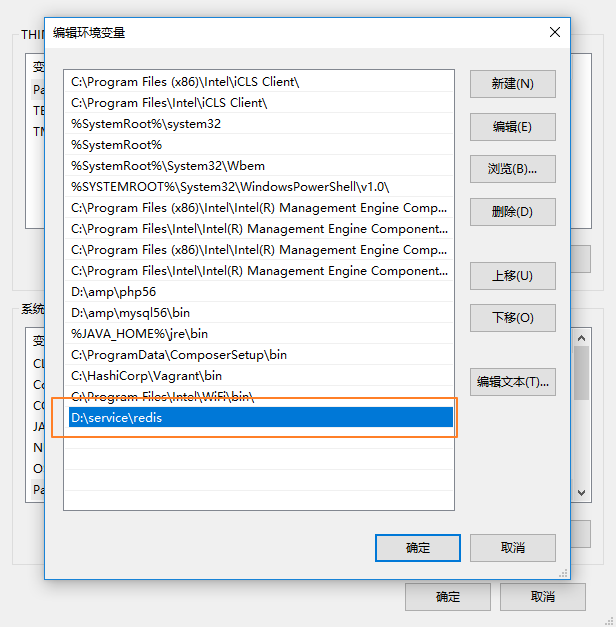
service/redis



可以双击redis-server.exe测试运行



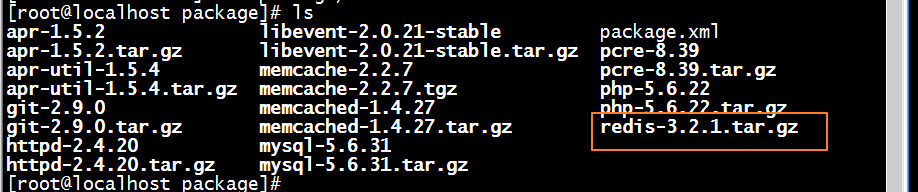
### 配置环境变量



## linux下安装

### 安装

获取源码, 上传的到linux服务器



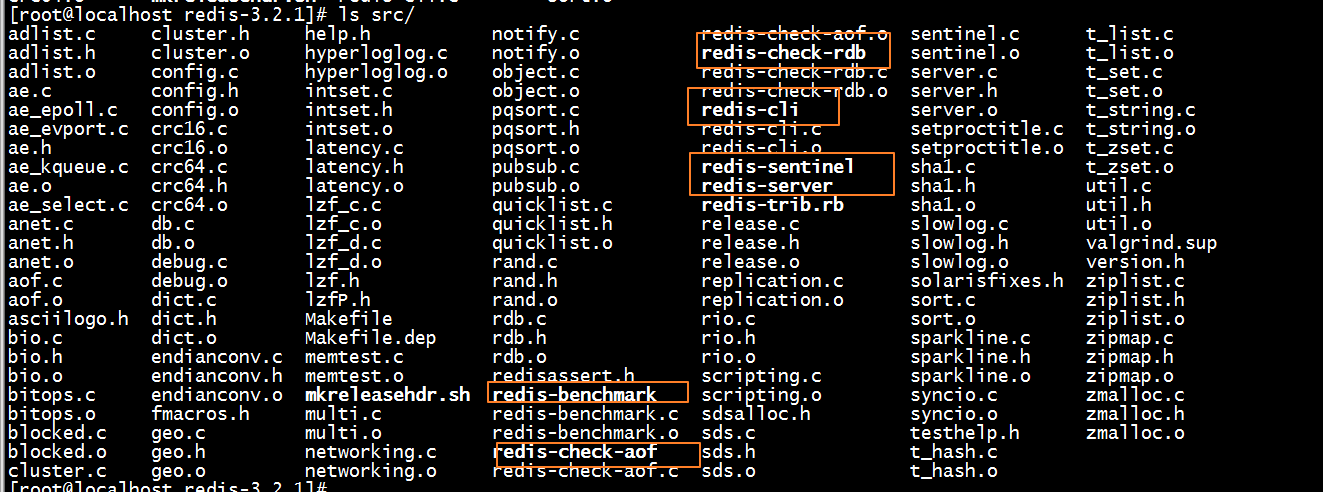
解压, 进入目录

make 编译生成即可(保证 gcc, gcc-c++)





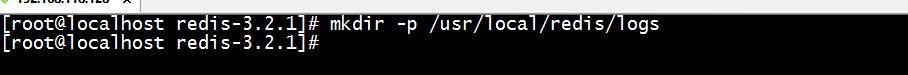
make之后, 生成需要的 执行程序.



按照需要整理到合适的目录中即可

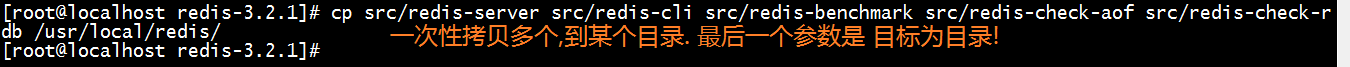
/usr/local/redis 需要手动创建

例如: 需要创建logs这些子目录(与windows保持一致)



将需要的文件移动过去

执行性程序



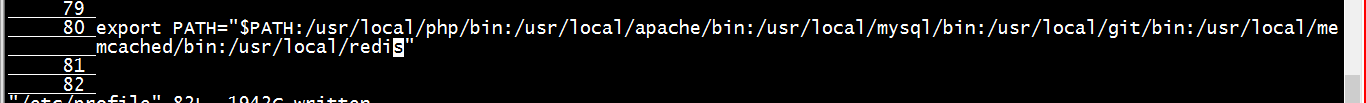
配置文件



结果



### 环境path





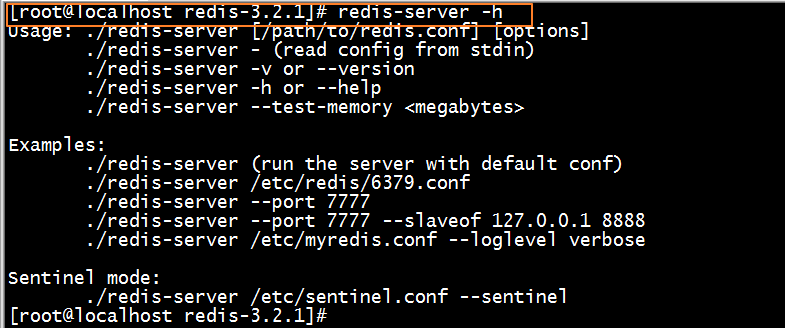
# C/S架构

## 服务器端

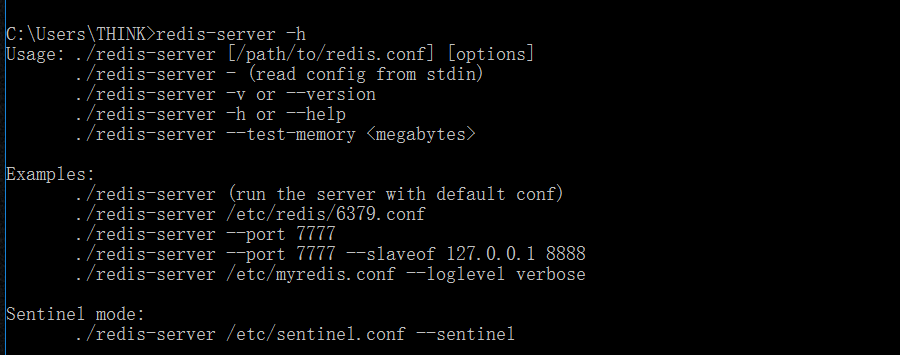
redis-server

提供的基于shell(CMD)的服务器端程序

linux:



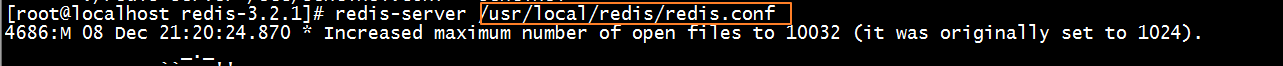
windows:



### 开启

典型的情况, 就是开启时, 指定配置文件

linux



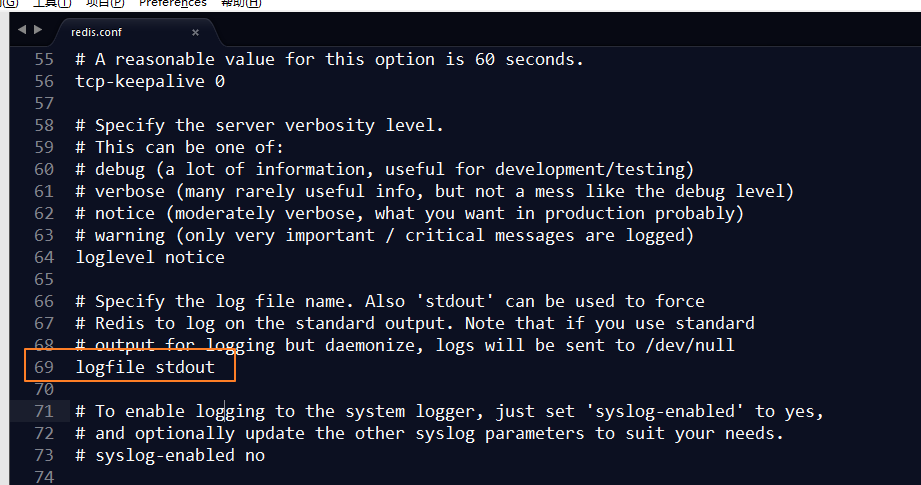


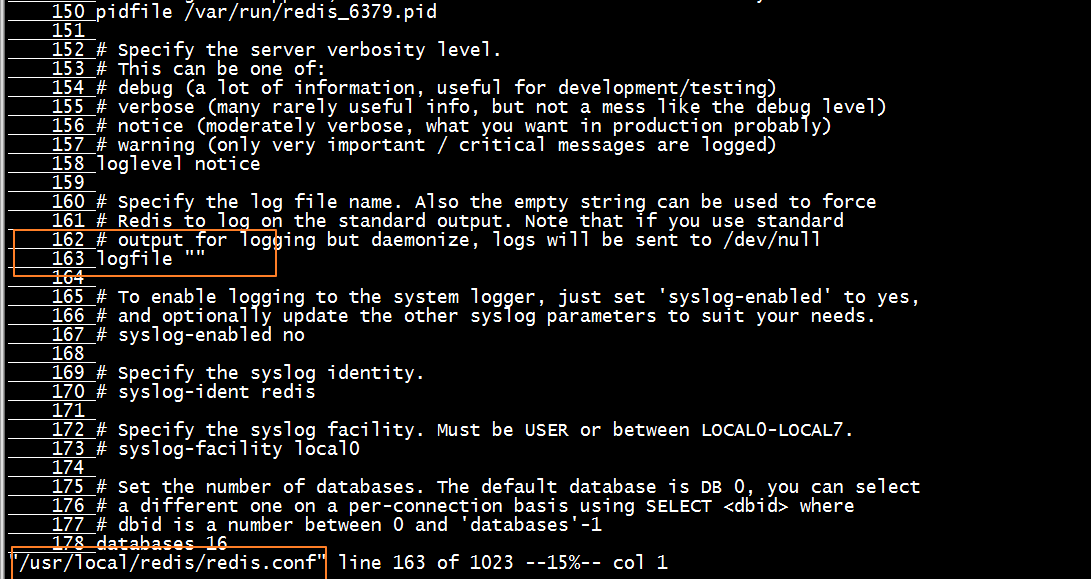
### 主要配置选项

#### logfile, 日志类型或位置

linux, 空字符串, 表示输出到标准输出

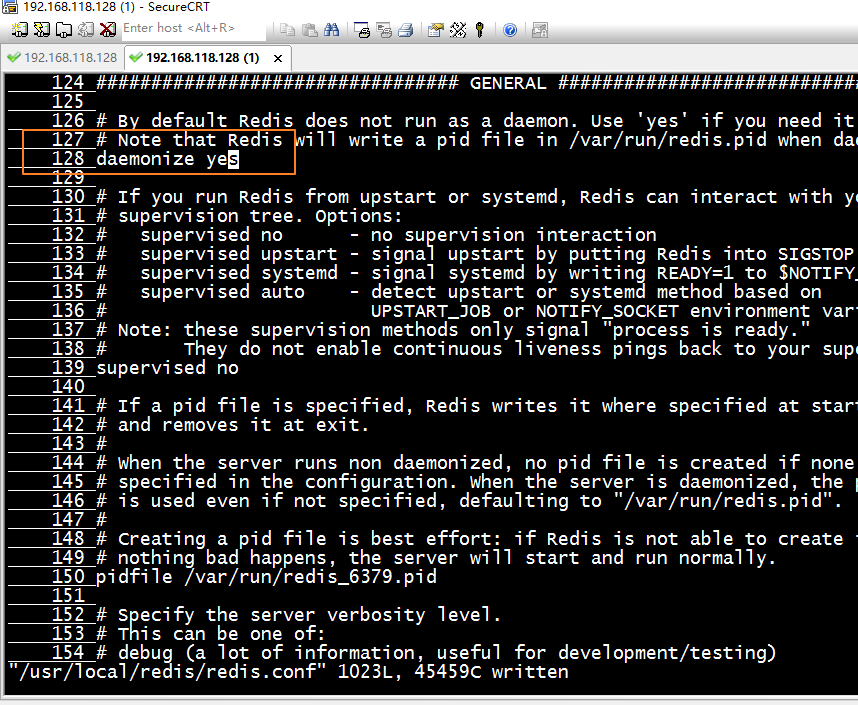
windows: stdout, 表示输出到标准输出





#### daemonize,

linux支持



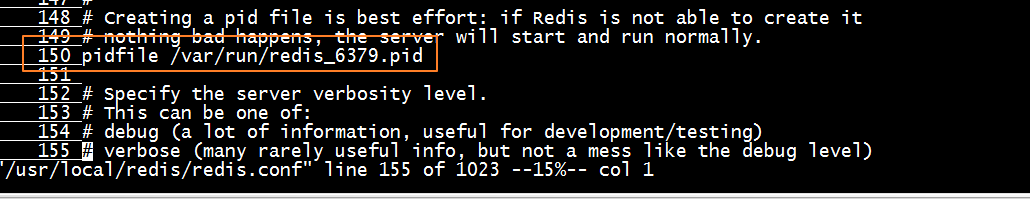
再开启服务,

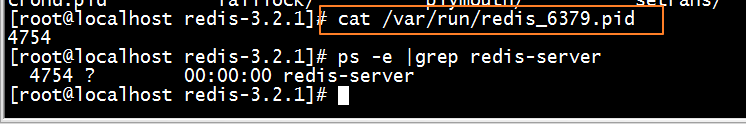


#### pidfile

进程ID文件, 用于存储当前的redis的主进程号!

用来管理redis进程, 例如, 重启, 退出等!

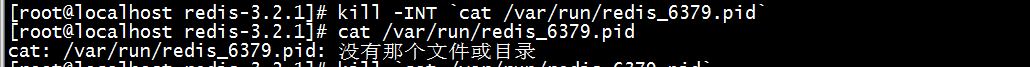




### 管理进程

linux, 下使用 kill, 向进程发送信号, 管理当前的redis-server进程

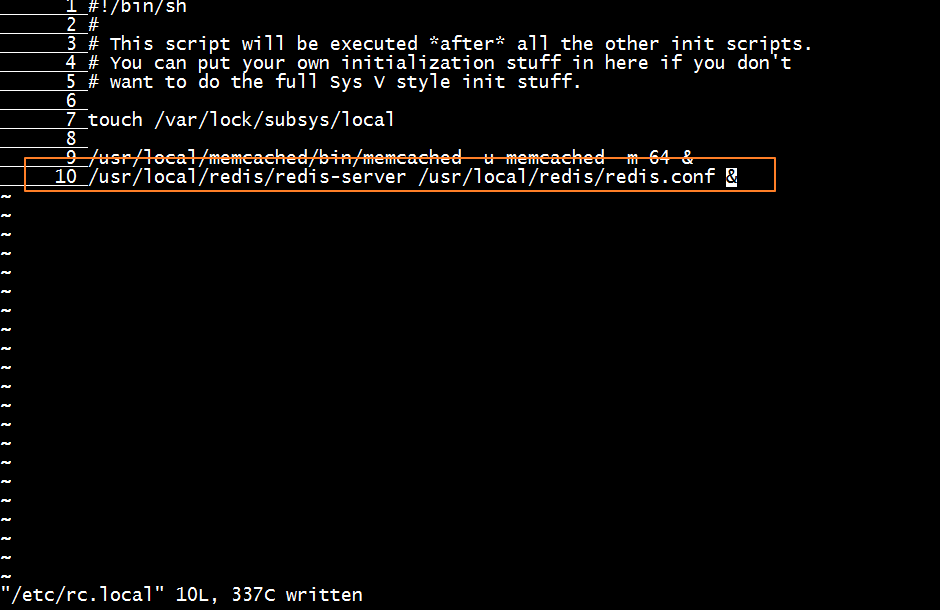
-INT关闭(默认的信号), 省略也是-INT



反引号包裹 命令, 会去执行该命令, 作为参数.

新获取进程号, 再发送信号!

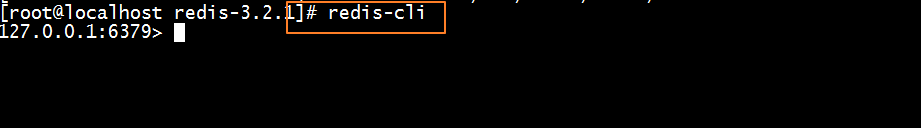
### 将启动脚本写在 rc.local(linux)



## 客户端

### 命令行客户端

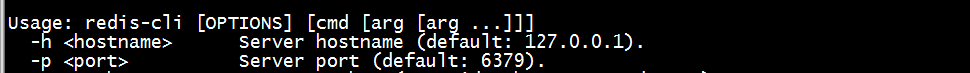
redis-cli

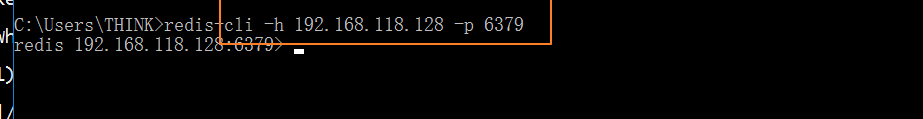


默认没有验证, 直接登录即可

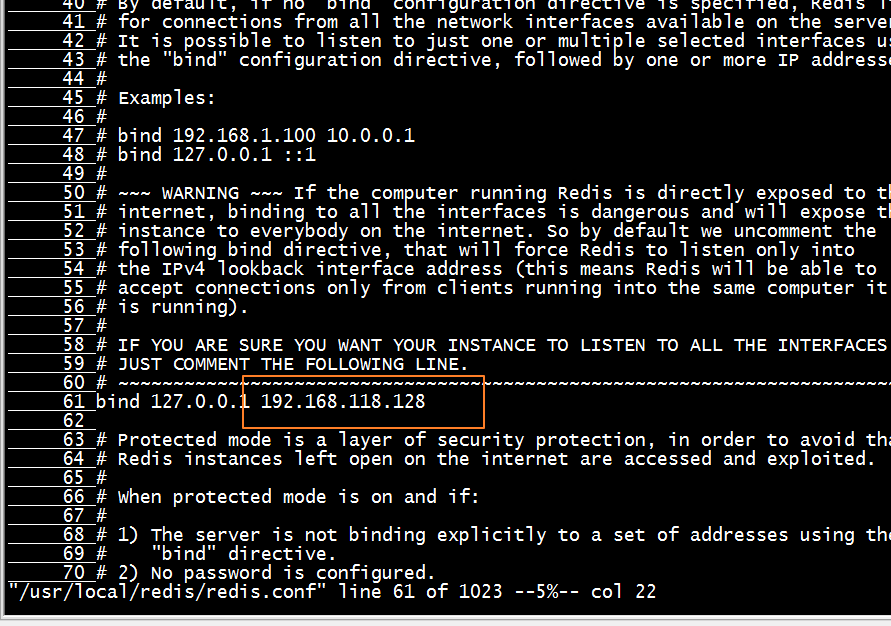
默认连接的服务器为: 127.0.0.1:6379

如果需要连接其他服务器, 需要指定参数





默认, redis的服务器, 仅仅允许本机连接, 需要配置配置项bind加入允监听的IP



主要用于调试!

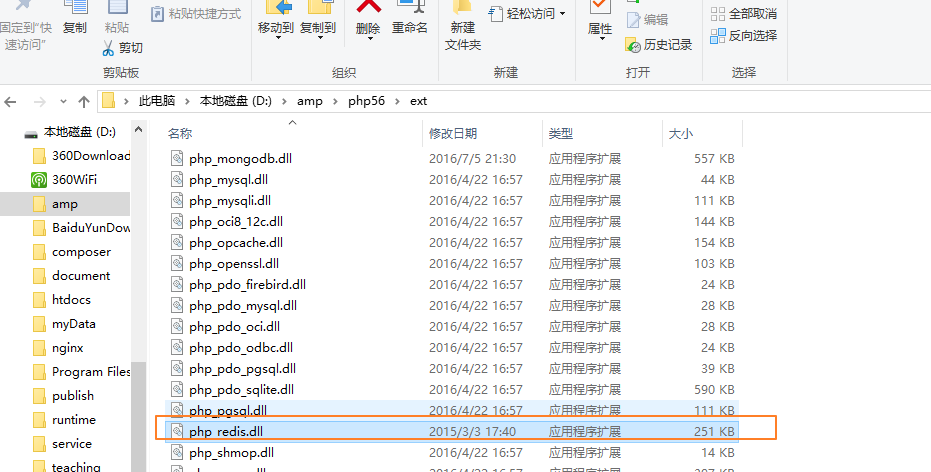
### PHP程序

安装可以操作reids的扩展

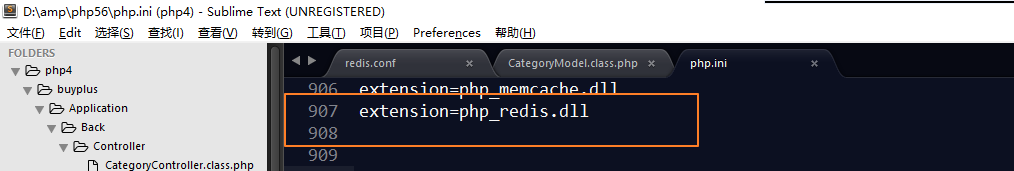
#### windows

获取到相应的dll, 放置到php/ext

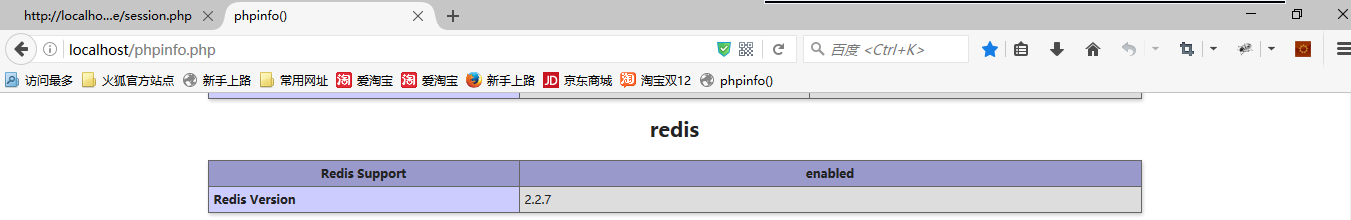




配置 php.ini

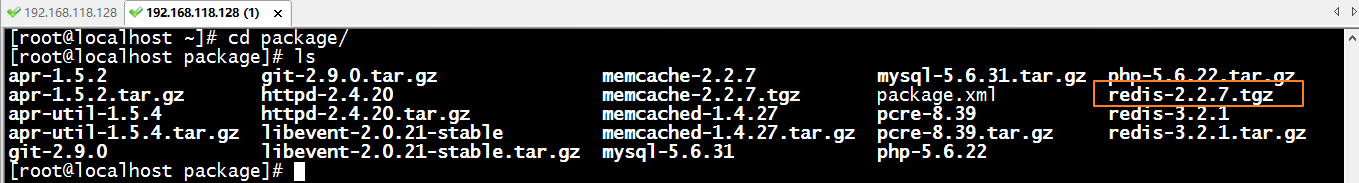


重启apache httpd



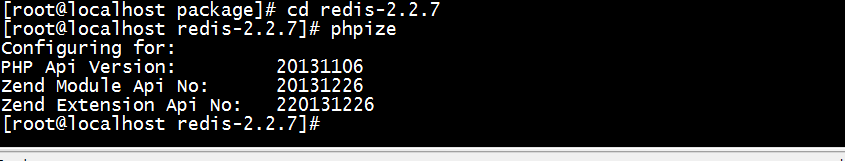
#### linux

扩展源码获取

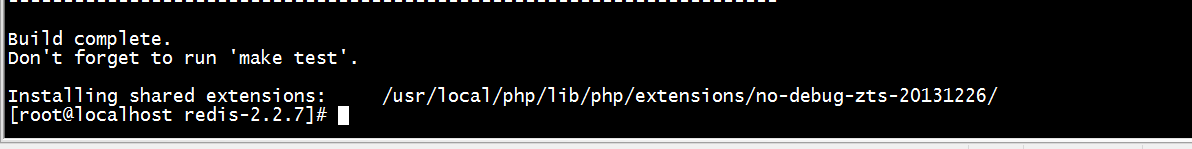


phpize

解压进入源码包, phpize

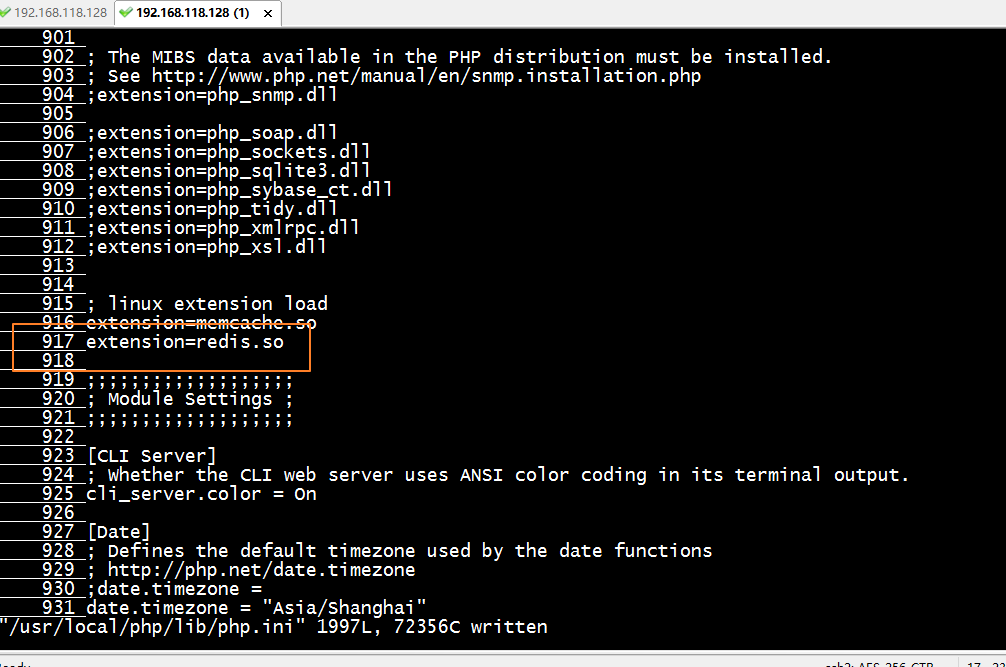


配置, 编译, 安装



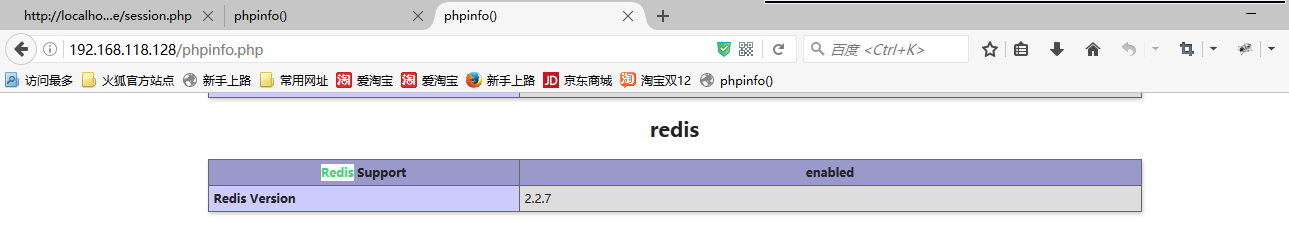


配置php.ini



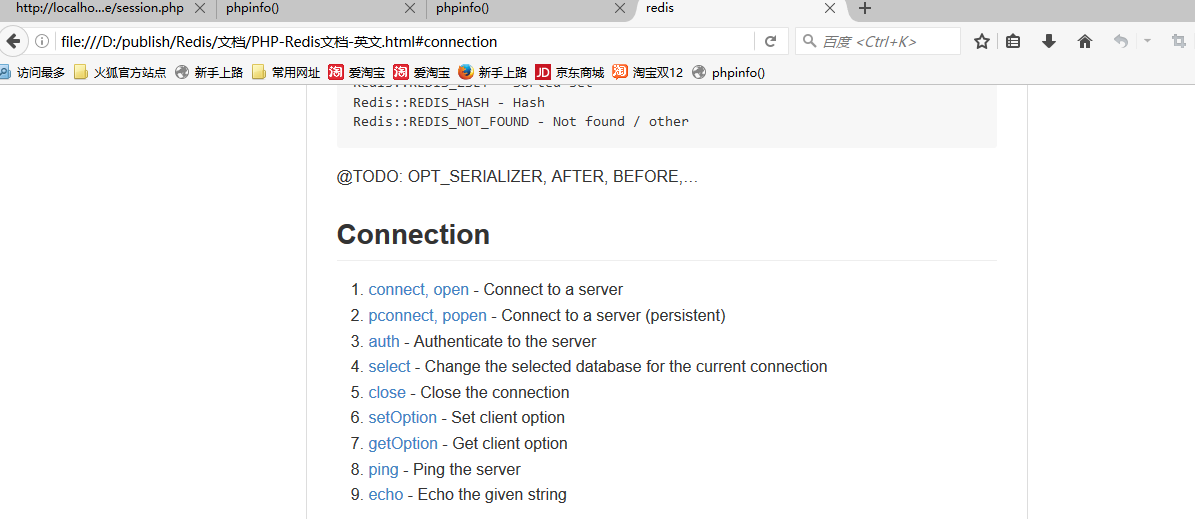
重启apache





# 数据(类型)操作

参考

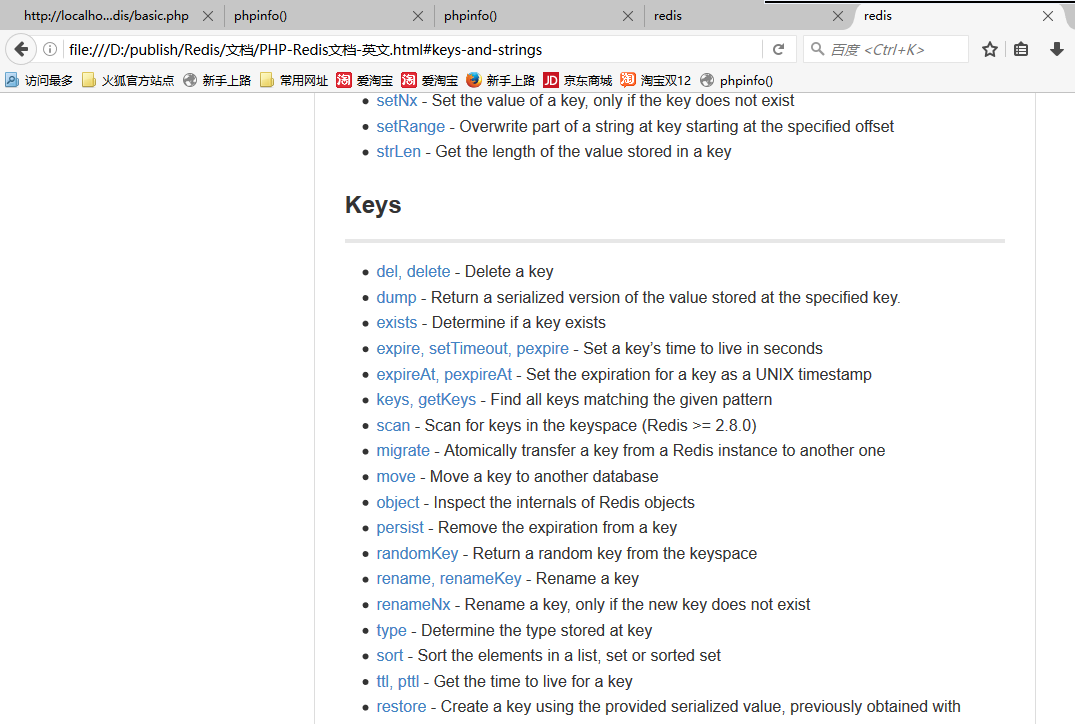


## 实例化redis对象, 连接redis服务器

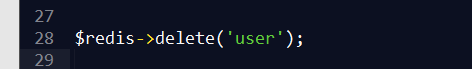


redis提供了大量的命令(方法). 根据不同的数据类型, 操作是不同!

## 通用操作



### delete, 删除缓存项



### expire, expireAt, 有效期设置, ttl有效期获取

需要在设置好了缓存项后, 再设置有效期.

expire, 时间周期

expireAt, 某个时间戳时间点,失效.



可以获取当前还剩余的有效期时间: ttl()方法. time to live

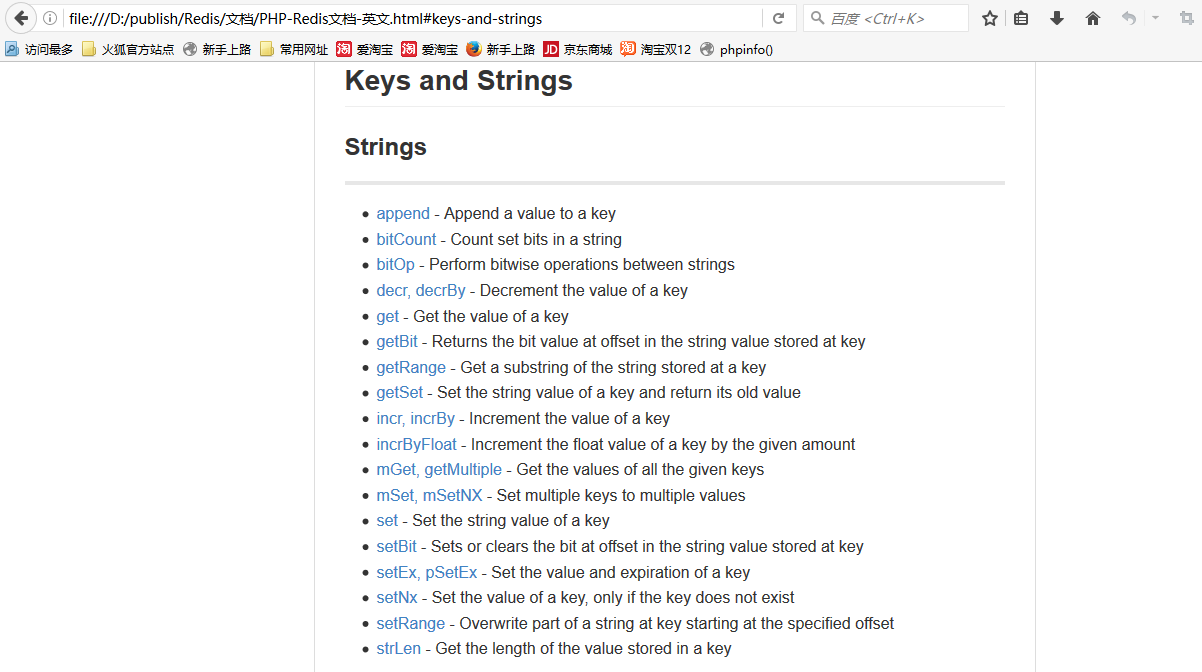


PS:时间可以精确到毫秒！使用p作为方法前缀!

pexpire, pexpireAt, pttl

### exists 检测一个缓存项是否存在

## string字符串类型操作



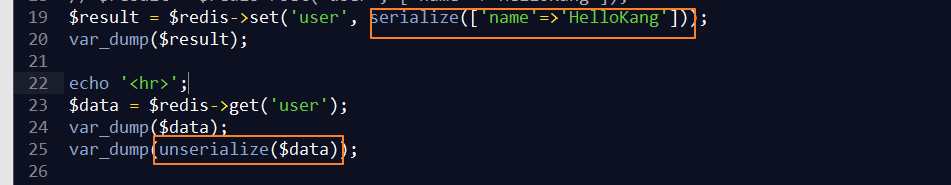
最常规, 最通用 的数据类型. 数据缓存时, 用到就是该数据类型. 字符串!

### set, 设置

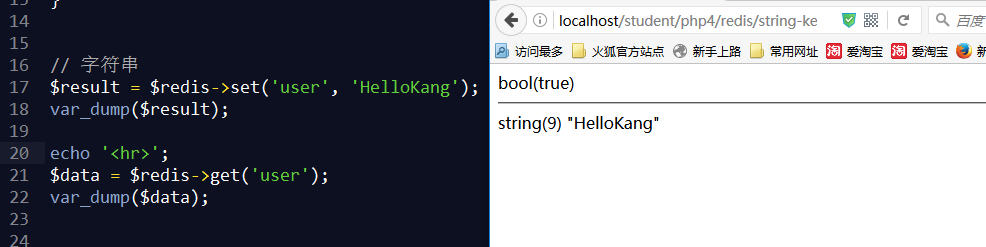
仅仅支持字符串操作. 不支持内置数据编码功能.

如果需要存储PHP的非字符串类型. 需要手动序列化, 在存储.

获取时, 反序列化!



### get, 获取

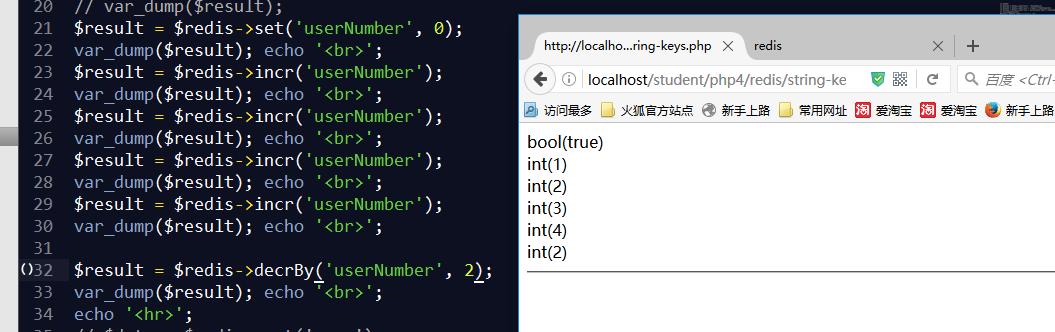


### incr, incrBy

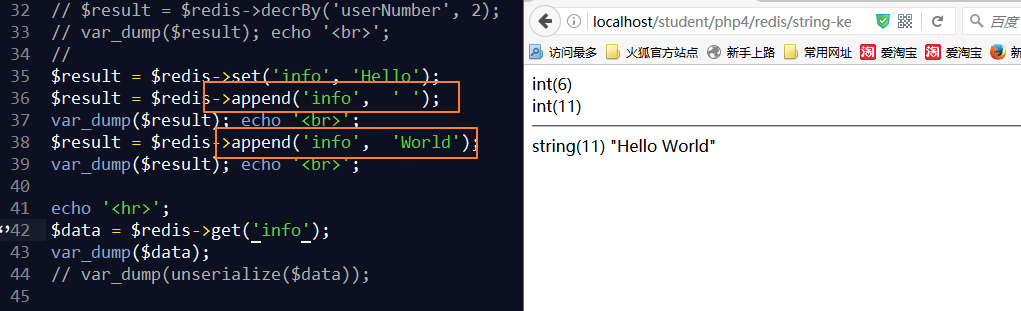
增1, 指定步长增加

### decr, decrBy

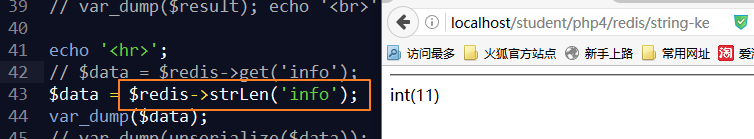
减1, 指定步长的减



### append, 追加



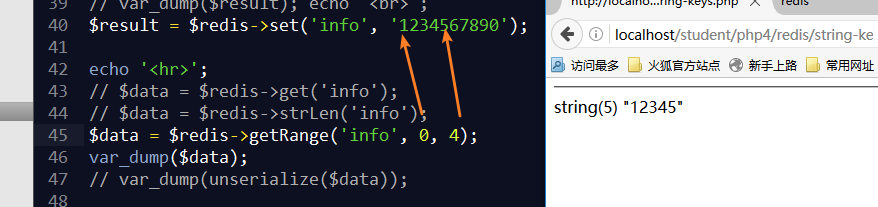
### strLen, 内容长度



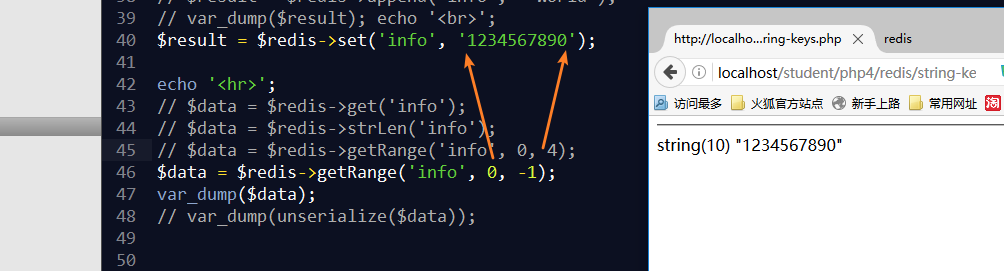
### getRange获取部分

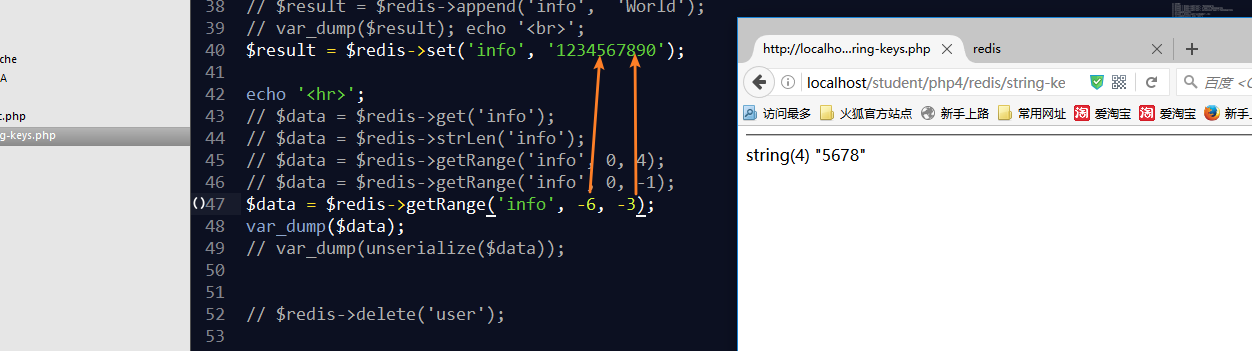
获取字符串的部分内容

利用 开始结束的索引位置, 进行返回部分



支持负数位置:





## list 链表类型

(队列是一种特殊的链表)

连续的线性存储空间, 就是链表(线性表)

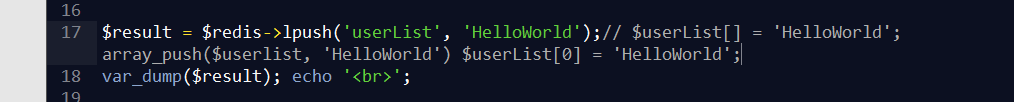
是一种复合数据类型. 一个链表, 可以存储多个值!

就类似于PHP: 索引数组.(下标为从0开始连续的逐一递增的数组).



将元素放入链表

### lpush(), 将元素压入链表

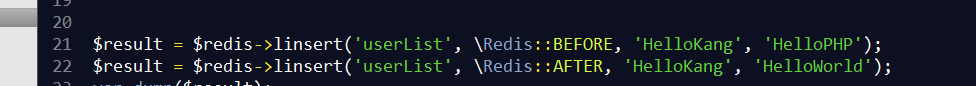


此时, 链表中, 存在一个元素.

### linsert(), 在某个位置插入新元素

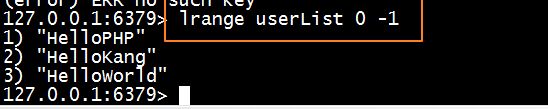
位置的判断, 是根据相对的参考元素判断.

当前链表中仅仅存在一个HelloKang的元素, 执行下面的插入:



结果是:

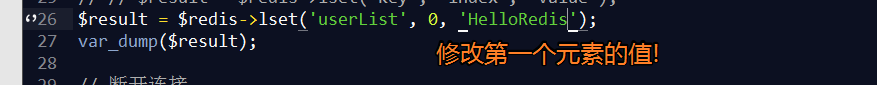
在HelloKang前后分别有 :HelloPHP 和HelloWorld



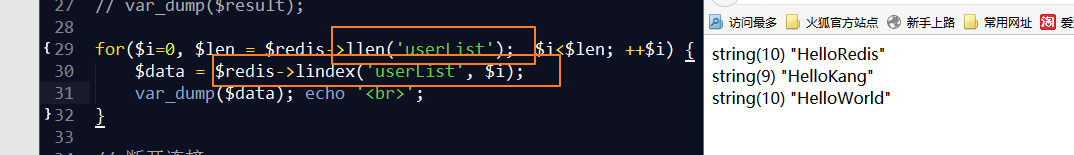
### lset(), 设置某个元素的值

某个元素使用索引下标进行确定：

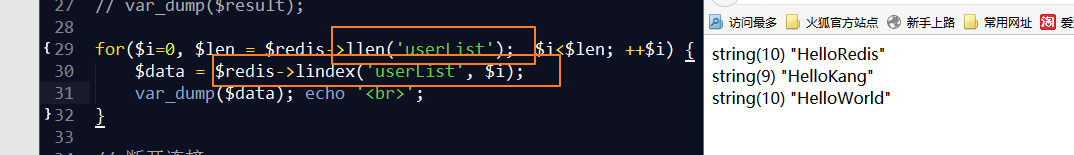
始终是从0开始, 到长度-1逐一递增的索引下标!



### llen(), 获取列表元素个数



### lindex(), 返回索引对应的元素

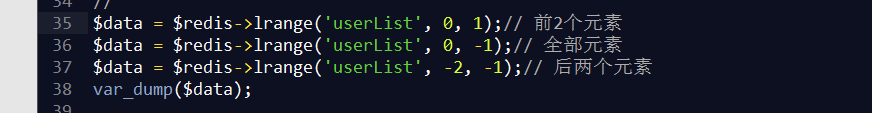


索引从0开始!

### lrange(), 获取某个选定范围元素

范围是利用, 起止下标确定.

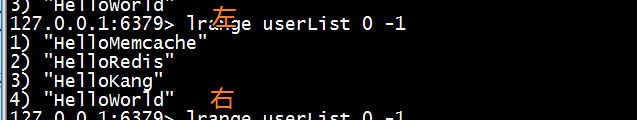
支持整数和负数. 负数表示从后向前第几个元素.



### lpop(), 从列表中, 左侧弹出元素

将数据从左侧弹出, 会获取弹出的元素, 数据元素在list中消失.

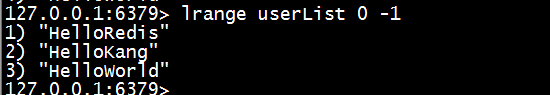
原始数据:



左弹出:


此时, list中的数据:



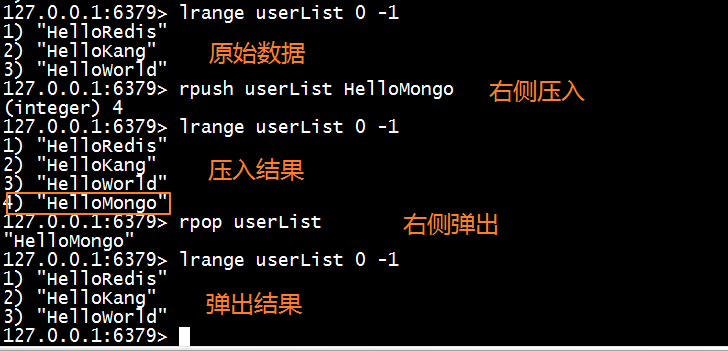
lpush()也是在左侧将数据压入 list中.

相对的.

### rpush(), rpop()

从list中的, 右侧, 压入或者弹出元素

测试:



### lremove, 移除元素

根据值, 移除元素

list中可能出现重复的元素, 需要通过count指定, 需要移除的元素的数量.

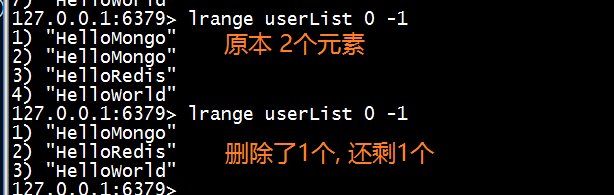
(key, value, count=全部)





删除固定数量:





### 队列操作, FIFO, 先进先出

链表操作中, 如果限制为 仅仅在一端压入(无论左右), 另一端弹出(无论左右), 满足这样的要求的链表, 就称之为队列queue!

例如, 在redis中:

lpush + rpop

rpush + lpop

队列在应用中, 主要解决, 并发的问题!



当请求到达服务器端应用程序时, 并不是马上处理. 而是将请求安置在队列中.

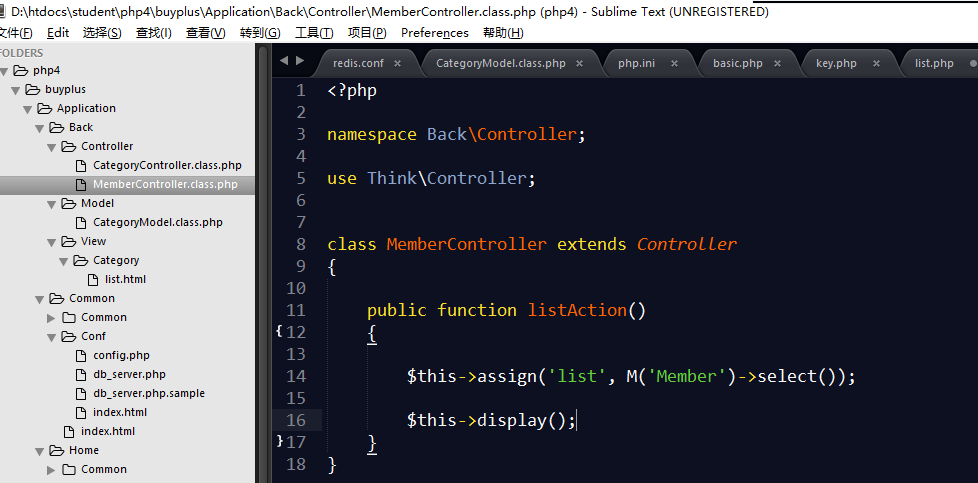
应用程序, 根据进入的先后顺序, 逐一处理请求.

当操作需要花费大量的资源时, 通常现将任务排序, 再处理.

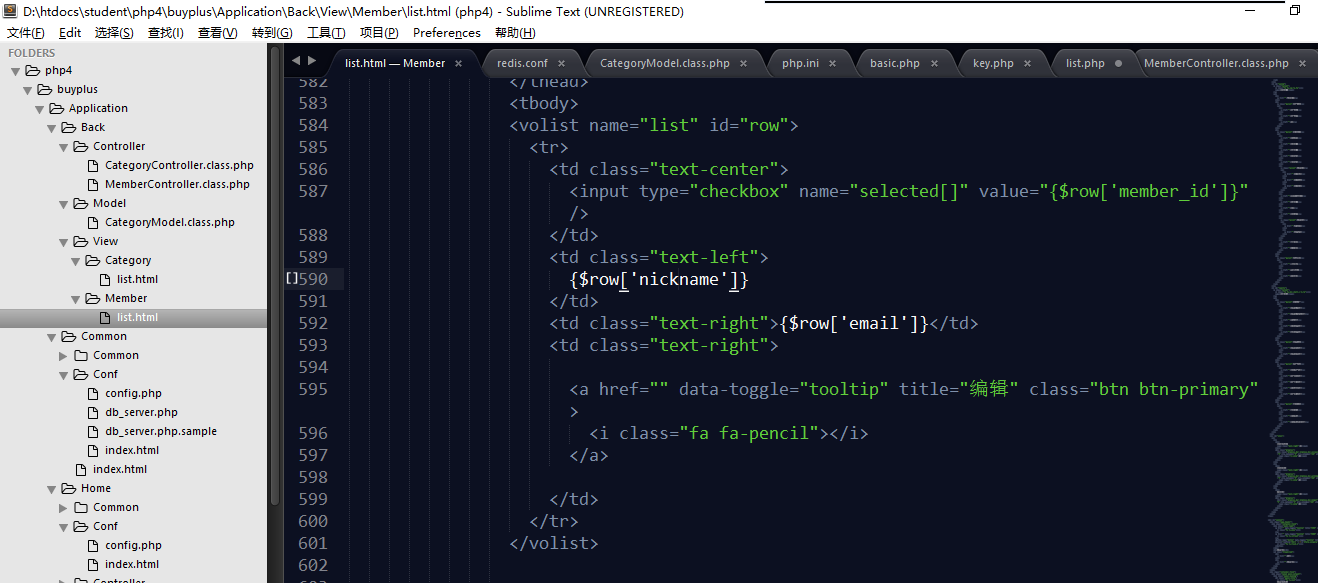
例如: 后台, 向会员发送邮件功能: 通常需要将需求, 加入邮件发送队列中. 另外的进程处理队列, 完成逐一的发送.

#### 会员列表, 提供多选发送邮件功能

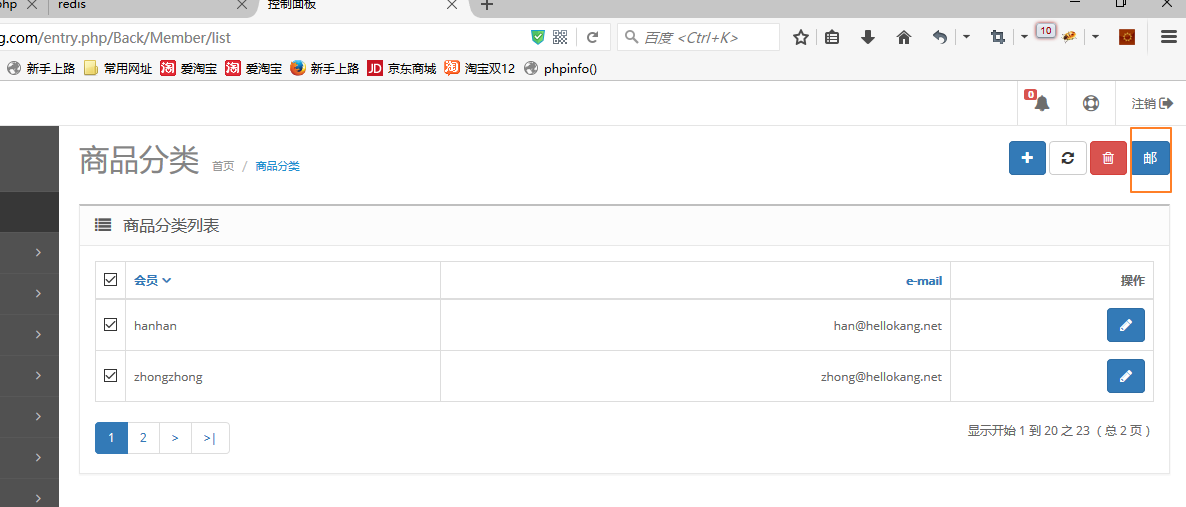
##### 控制器



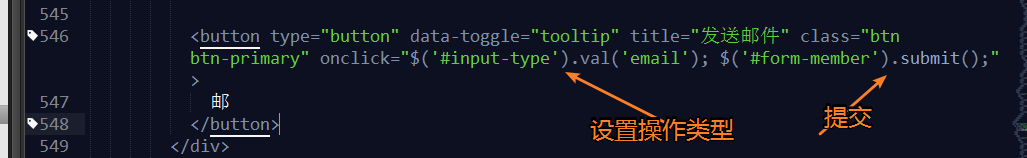
##### 视图-模板



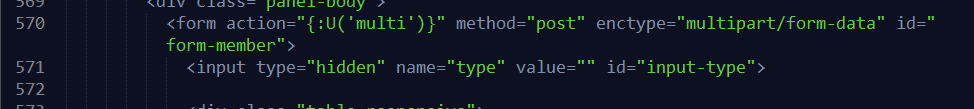
##### 增加发送邮件按钮



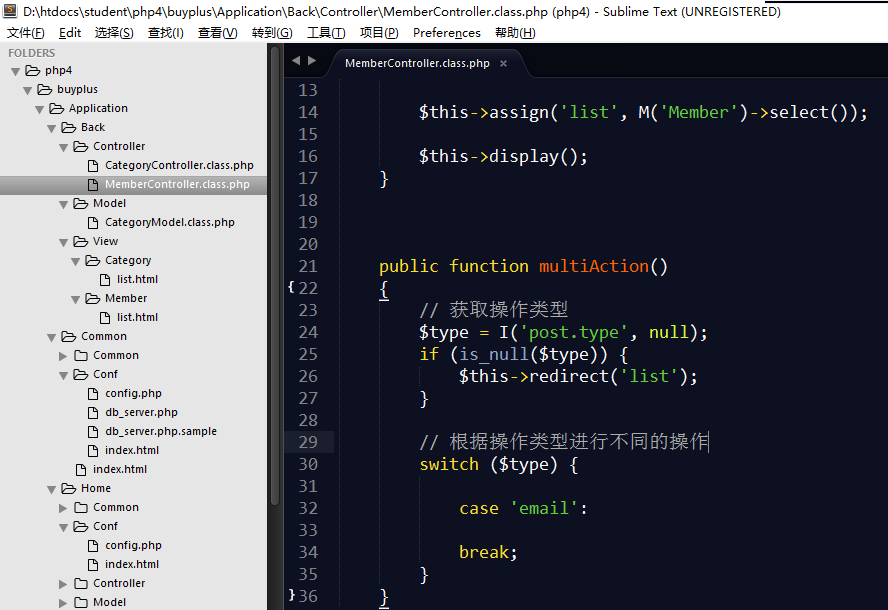
点击后, 设置操作类型, 同时提交表单:



使用表单的隐藏于, 传递当前的操作类型, 给multi动作:



##### 发送邮件的multi动作



每法送一封邮件, 时间是较长.

批量发送, 一次请求, 需要发送多封.

如果该动作, 等待发送完毕再响应, 浏览器用户, 等待时间 无限延长!

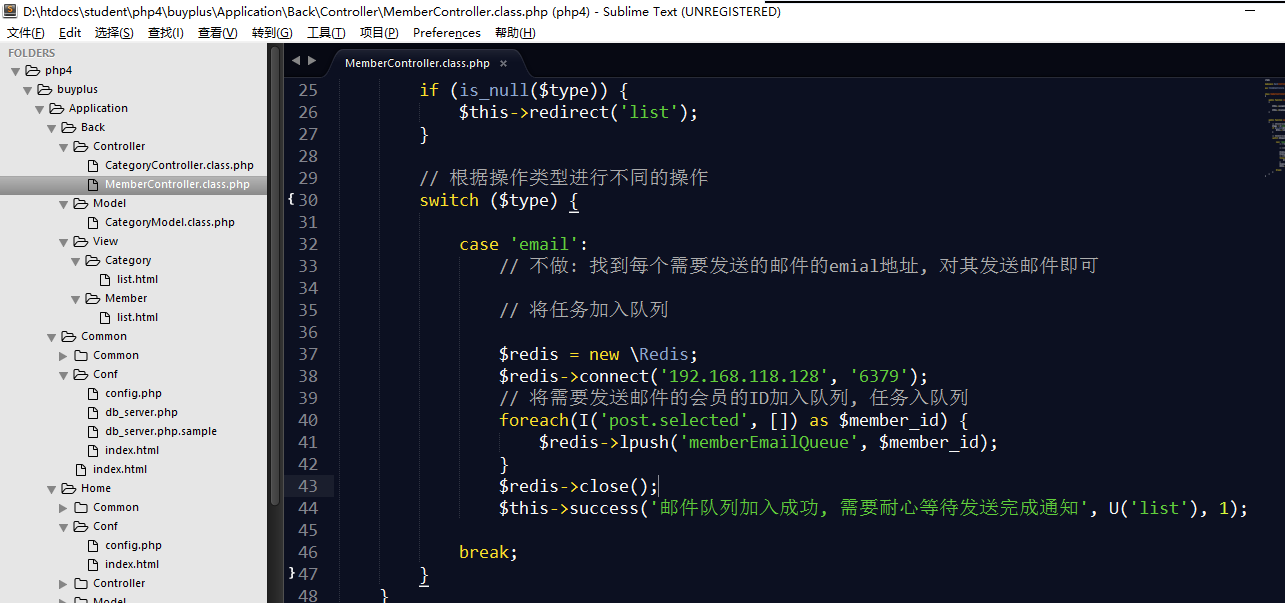
该操作就是典型的并发问题:

同一时刻请求多, 但是资源提供少.

为了避免, 浏览器用户处于长时间的阻塞等待状态.

我们的emial处理, 不是立即发送邮件, 而是将发送邮件的需求, 加入到队列中

实现的代码如下



邮件请求入队列成功

#### 处理队列

一个额外的动作, 专门处理队列

通常是一直执行的守护进程状态!

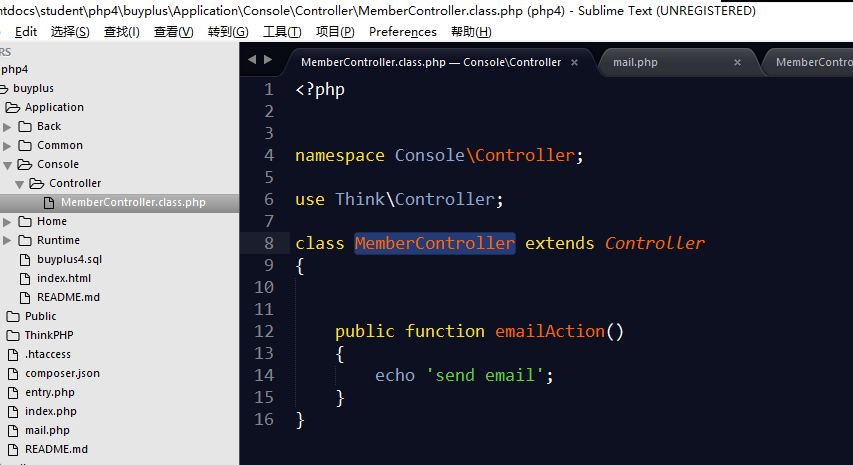
建立特殊的动作, 便于在CMD控制台上请求, 独立一个入口文件, 专门解析对应的动作

(控制台在处理时, 应该使用绝对路径)

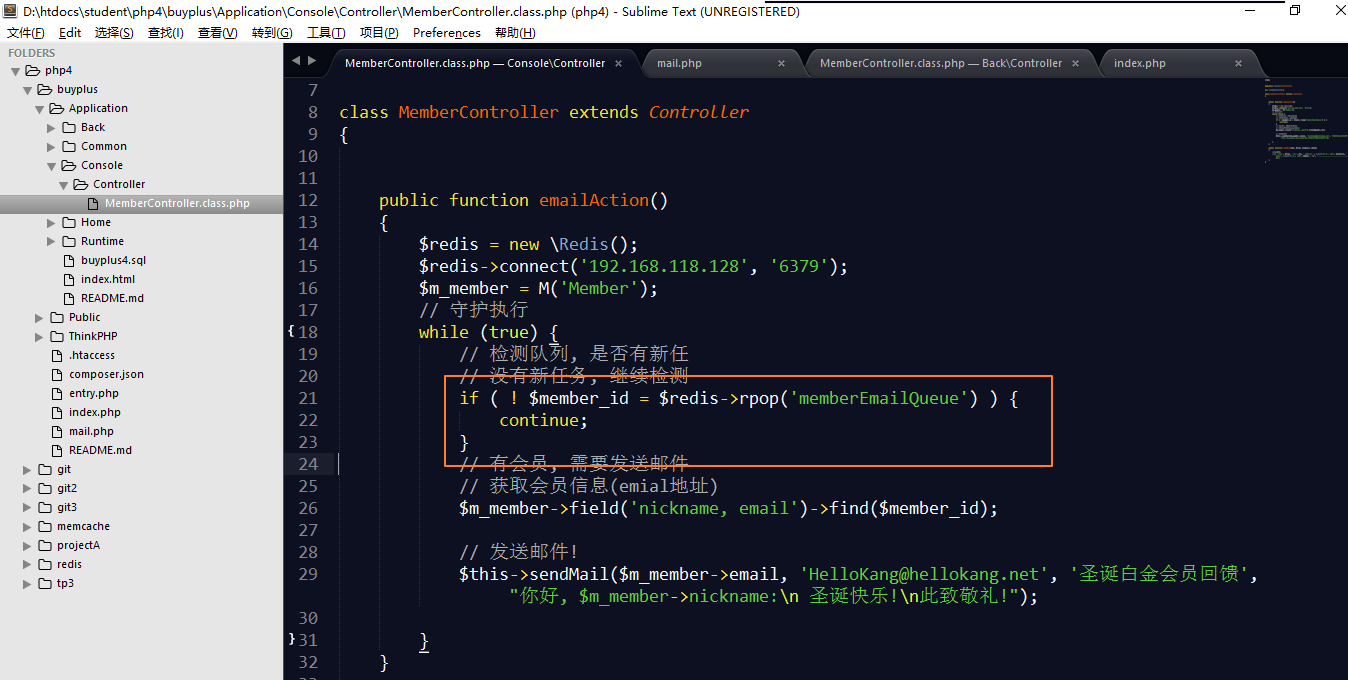
入口文件



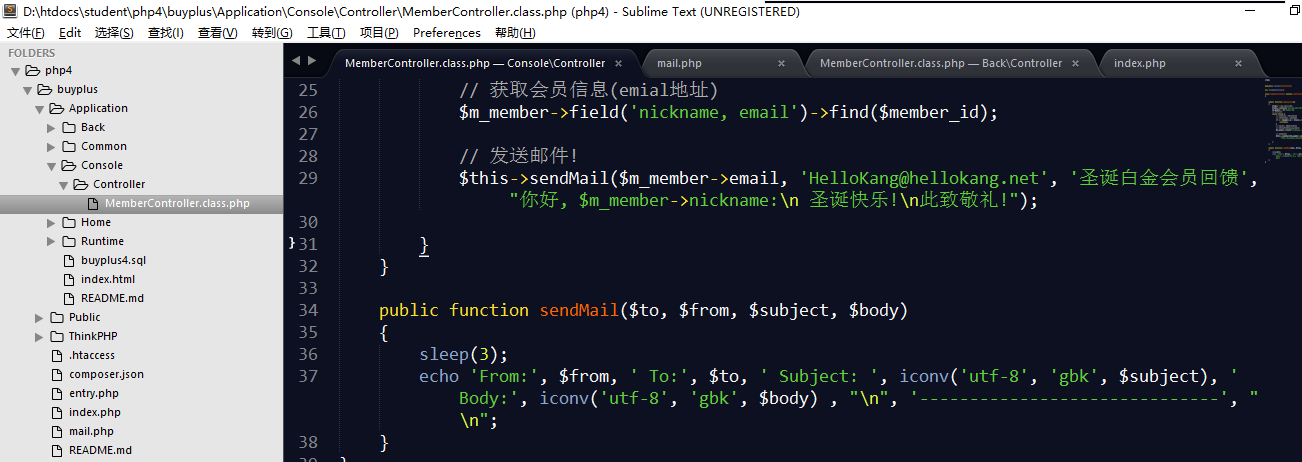
控制器动作:



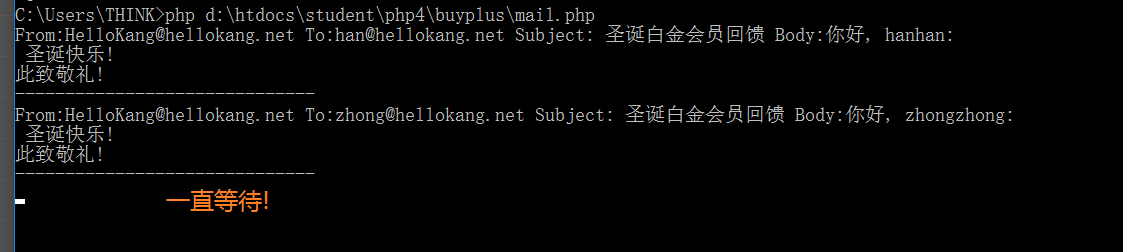
控制器动作, 守护代码实现:



模拟发送邮件:



执行的守护脚本



一旦加入了新任务, 则立即处理

## set 集合类型

也是一种复合类型.

一个缓存项, 可以存储多个数据!

集合数据的特征:

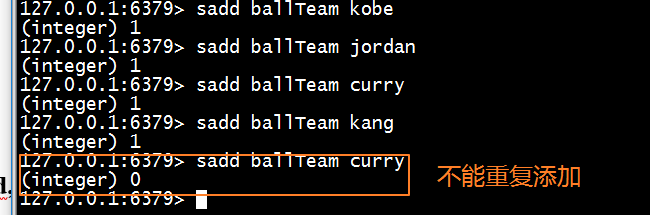
1元素不能重复, 保持唯一性

2,元素无序. 不能使用索引操作



操作 使用 命令行客户端演示.

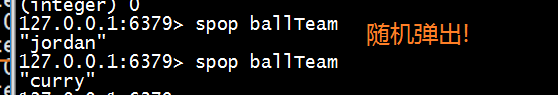
### sadd, 添加元素到集合



唯一性

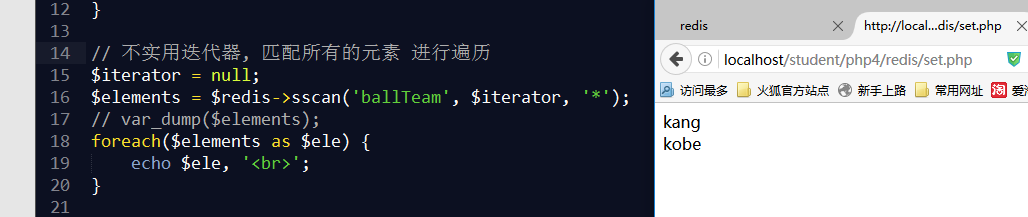
### spop, 弹出一个随机元素

无序性



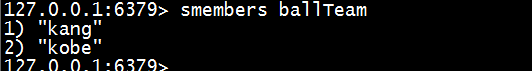
### sscan, 遍历集合

使用PHP程序演示



### smembers, 获取所有的成员

没有任何属性的sscan



### scard, 获取基数

基数, 在集合中, 指的是集合内的元素个数

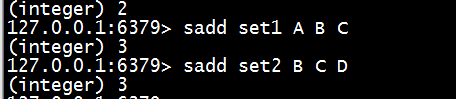


### sunion, sdiff, sinter, 并集, 差集, 交集

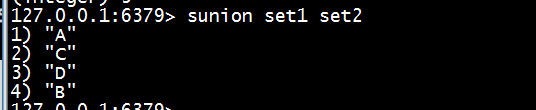
提供一些, 关于集合运算的操作

set1, A, B, C

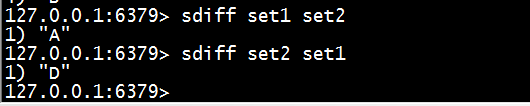
set2, B, C, D



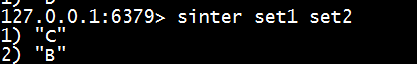
union



sdiff,



交集



## 有序集sorted-set

与集合相比, 每个元素多了一个属性, socre, 分值. 权重.

也称之为: 加分集合.

排序就是利用score完成!



处理元素时, 也要加上score的处理

### zadd, 添加元素

添加元素



### zincrBy, 元素分值增减



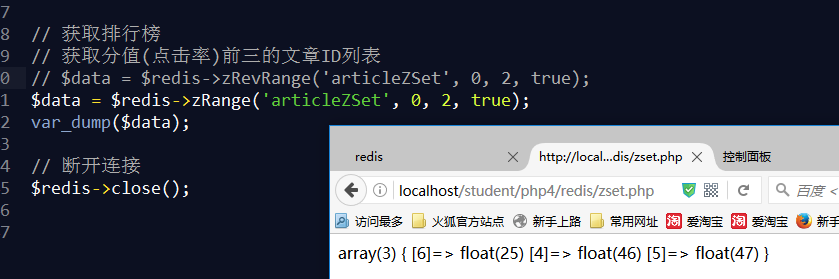
分值可以为负数, 表示递减!

### zRange(), zRevRange()

根据分值排序后的,升序和降序(rev)的列表获取



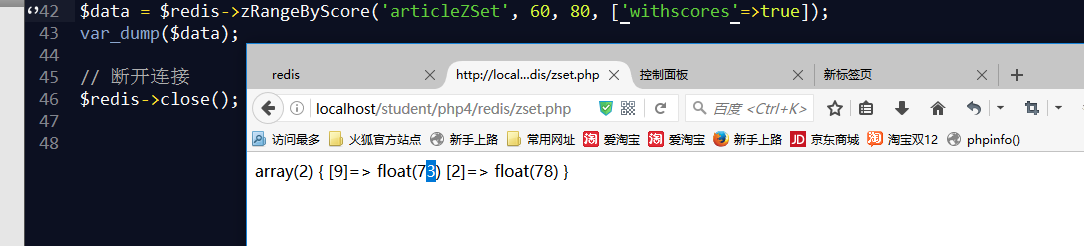
升序



### zrangebyscore(), zrevrangebyscore()

根据 分值过滤之后的列表

需要提供 分值区间!



### zCard, 获取基数 元素个数

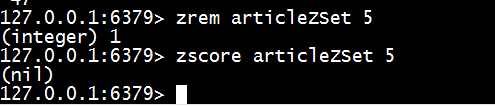


### zscore, 获取元素的分值

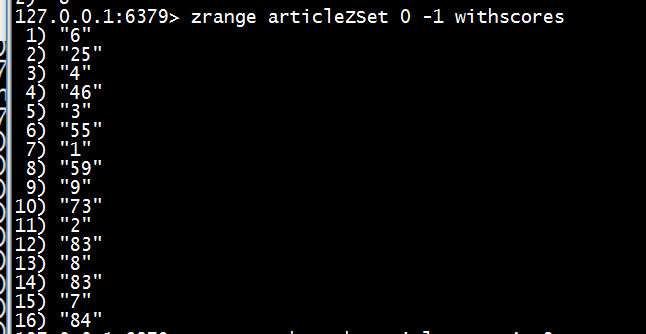


### zrem, zremrangebyrank, zremrangebyscore删除, 根据排行删除,

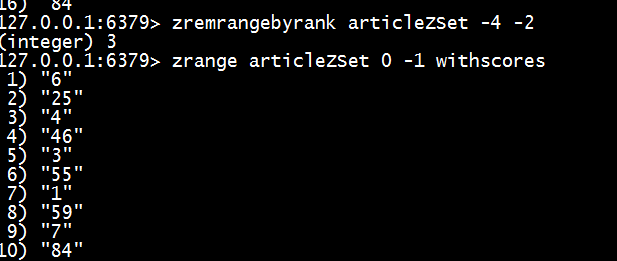
通过元素删除:



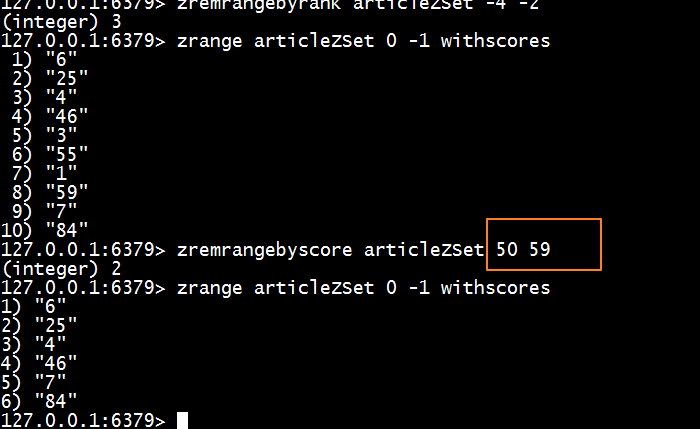
根据排行删除



结果



根据分数区间删除



## hash-table, hash表

PHP中的关联数组, 就是hash表!

一种复合数据结构. 采用Key=value型 进行存储数据结构.

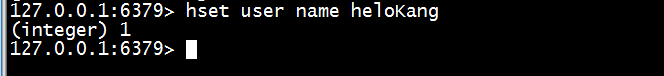
一个完整的数据, 例如, 一个用户的各个属性信息.

一篇文章的各个字段, 作者, 标题等.

实操时, 一般是, 一个用户信息, 序列化后存储到string中即可.

不需要一个个字段的存储在hashtable中!

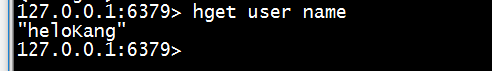
### hset, 设置



需要指定, 缓存项名, 下标字段, 值

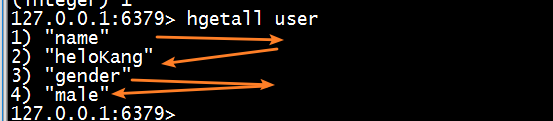
$user[‘name’] = ‘HeloKang’

### hget, 获取



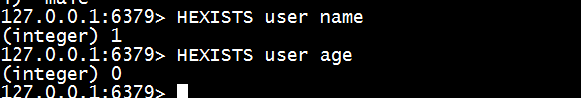
需要指定, 缓存项名, 下标字段.

### hGetAll, 获取全部的元素



print\_r($user)

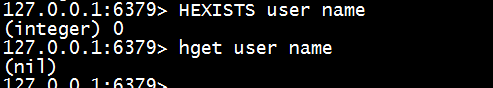
### hexists, 判断元素是否存在



isset($user[‘name’], $user[‘age’])

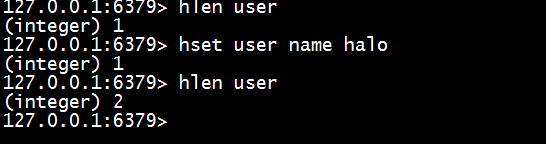
### hdel, 删除某个元素





unset($user[‘name’])

### hlen, 获取长度



count($user)

## HyperLog, 集合基准数

仅仅用来存储集合基数的. 而不是存储集合中的元素.

专门用来统计不同的元素的数量的.

不能获取元素, 仅仅可以统计数量!

实操中, 如果不在乎元素内容, 而仅仅在乎的数量, 则可以使用该类型.

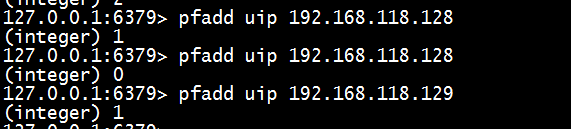
类型:

统计网站的 UIP(独立访客IP), US(独立会话session, 独立用户). 衡量业务逻辑量.

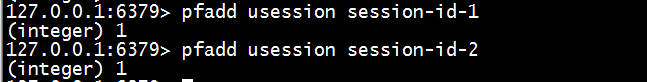
pv(page view), 衡量网站性能. 站点的吞吐量.

### pfadd, 向集合中添加元素

与: sadd()一致!

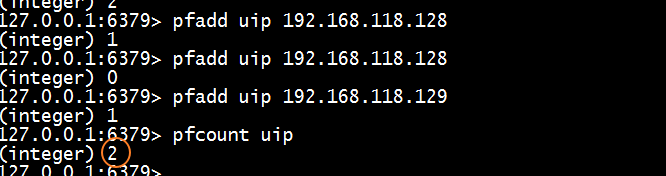


不能添加重复元素



### pfcount, 统计基数

scard()



## bitmap, 位图

一组01的组合.

‘01010100001010’

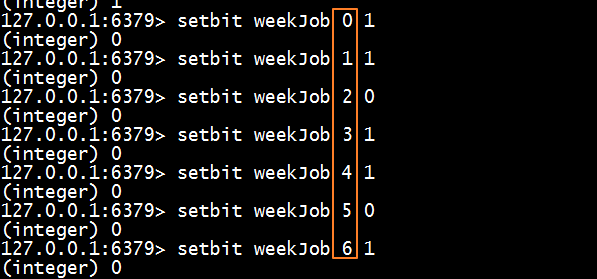
在redis的实现中: 就是一个, 在每个字符位, 仅仅是0或者1 的字符串.

由于仅仅需要存储0, 1, 所以可以转码后存储, 占用更少的空空间.

通过 字符串操作 get获取 位图的内容, 可以发现, 是16进制进行存储:

### setbit, 设置位图的某一位

需要指定某个位, 某个为的0或1的值



### getbit, 获取位图的某一个位



### get 测试 16进制



# 持久化支持

reids 默认支持支持化, 依据自己方式, 将缓存在内存中的数据, 存储在磁盘上!

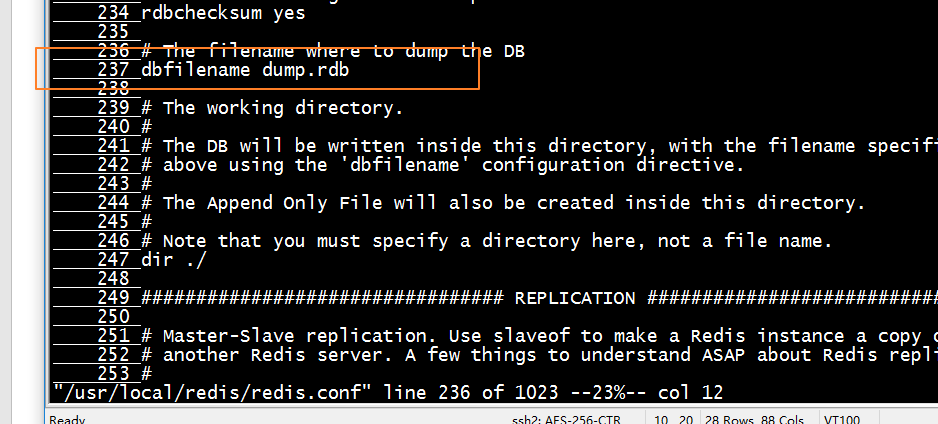
默认的文件: dump.rdb就是 redis持久化数据的存储文件.

如果在开启redis服务器时, 可以找到dump.rdb, 则将rdb中内容给, 还原到内存.

在客户端使用时, 数据体现 服务器重启, 但是数据不丢失 状态.

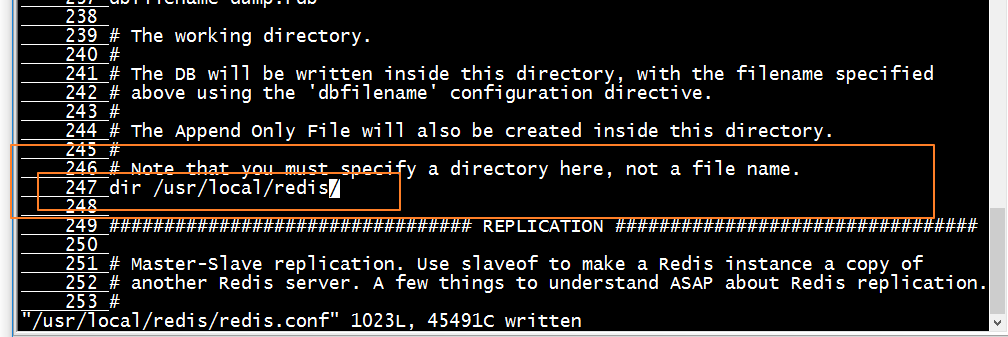
需要的配置

## 配置项: dbfilename



设置 存储的文件名

## 配置项: dir, 工作目录, rdb文件的存储位置

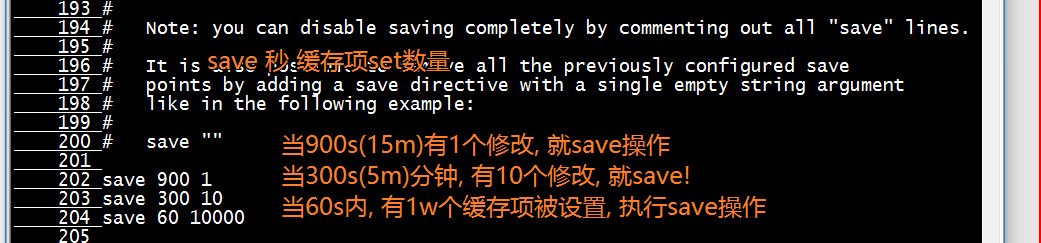


为了统一, 将目录改为: redis的安装目录.

默认在 当前下, 相对目录. 在哪运行redis-server, 哪就是当前目录!

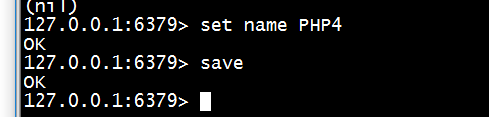
## 配置项: save, 持久化算法

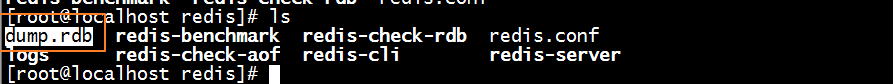
什么时间, 将内存同步到磁盘文件, rdb!



## save命令

客户端执行save, 立即持久化到rdb文件中:



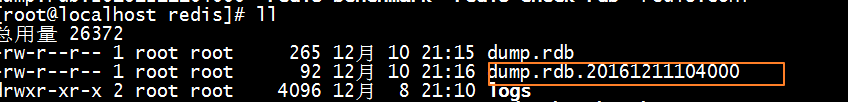


## 还原 rdb

将rdb文件, 拷贝到配置项dir指定的目录, 就会依据dump.rdb文件进行还原操作

redis的数据备份.

save命令后, 将生成好的dump.rdb, 保存在任意位置即可!



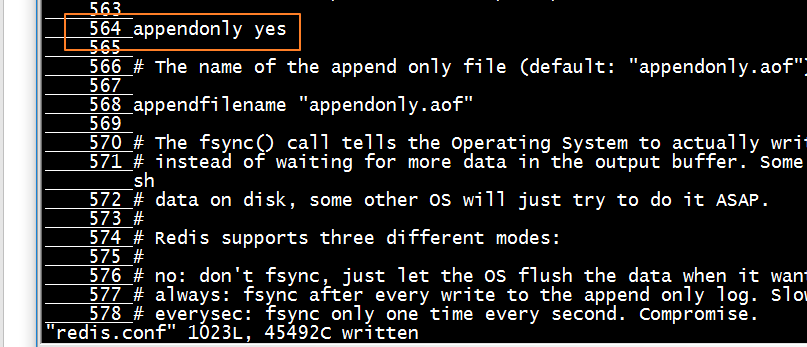
## aof 持久化

新版的redis,增加的持久化方式.

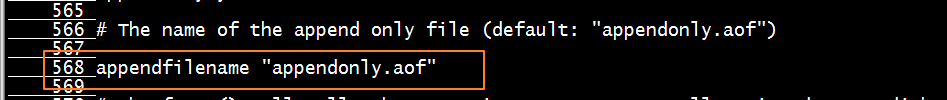
dump这种, 快照的持久化方式, 每次存储, 需要将内存中的全部缓存项, 无论之前的还是新增的, 都要重新存储到硬盘上! 好处, 内容完整! 坏处, 效率低!

新增的AOF, AppendOnly file 仅追加, 仅仅记录后来更新的缓存项.

### 配置: appendonly, 是否开启AOF



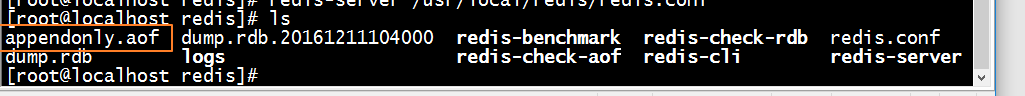
### 配置项: appendfilename, AOF记录文件名称



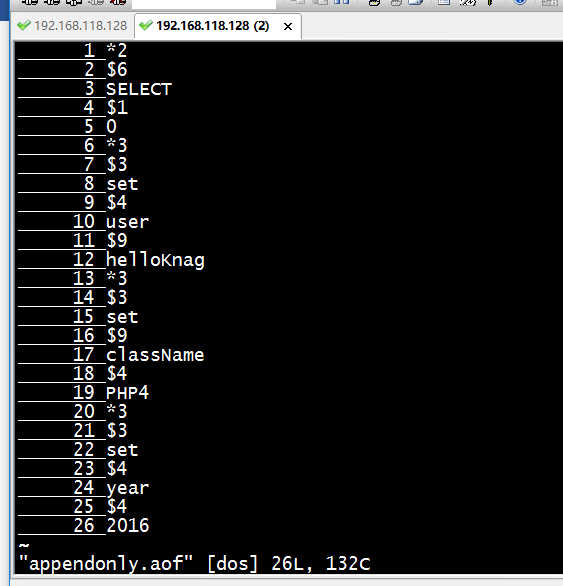
### appendonly.aof

重启 reids

执行了 set类的修改指令后, 就会生成AOF文件, 在dir目录中



查看:



服务器启动时, 还原时, 依次执行aof文件中记录的命令, 将修改重做!

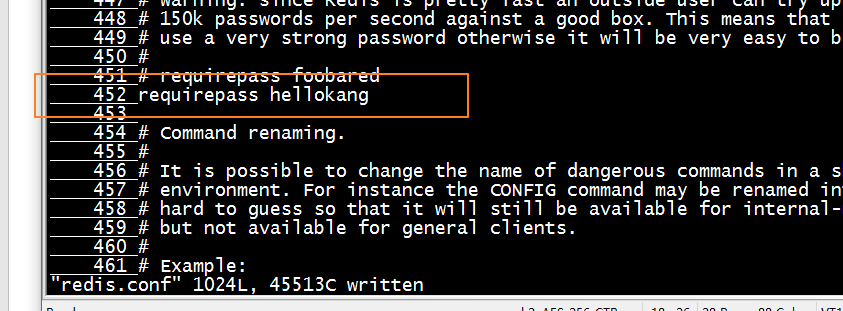
# 口令验证

redis, 为了速度, 默认默认没有开启密码口令验证.

为了安全 可以选择,开启.

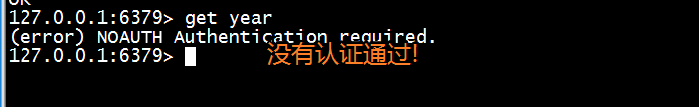
## 服务器配置文件配置密码

requirepass

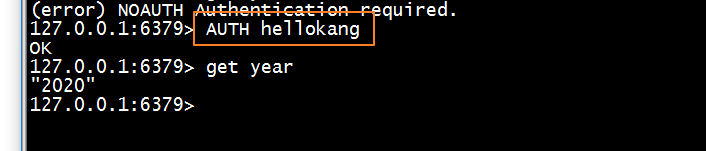


重启服务器端

## 客户端认证



开启客户端后, 执行auth即可:



没有用户, 没有权限, 仅仅支持口令!

对于PHP, 方法:

$redis->auth(password)

即可!

通常缓存服务器, 都在内部局域网. 不需要权限验证!

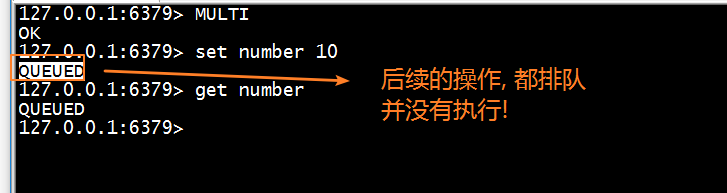
缓存服务器, 不在外网上的.

# 批处理(事务)支持

redis, 允许一次性执行多条命令! 就是批处理. redis自己称之为事务!

## multi, 开启批处理(开启事务)

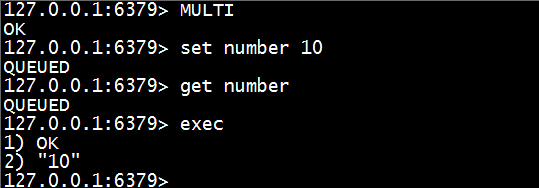
开启multi后, 后续的操作进行执行队列:



## exec, 执行(提交)

执行 刚刚的队列中的命令

队列中的命令依次执行即可!



## discard, 放弃(回滚)

当不需要执行队列中的命令时, 可以选择放弃:

