

Linux

Redis

[redis主从](#)
[redis配置认证密码](#)
[redis 过期策略](#)
[redis 禁用O\(n\) 复杂度方法](#)
[单节点redis分布式锁介绍](#)

命令介绍

[压测工具 wrk 使用](#)
[vim快捷键](#)
[链路测试工具mtr](#)
[系统监控工具 dstat 介绍](#)
[Linux之scp命令](#)
[htop 命令使用详解](#)
[crontab使用详解](#)
[crontab -r 清除后找回](#)
[logrotate 配置](#)

sphinx

[站内全文搜索引擎 Sphinx/coreseek 安装使用教程](#)
[sphinx/coreseek配置说明](#)
[sphinx 不关闭进程更新索引](#)
[sphinx增量索引配置](#)
[ubuntu 安装sphinx报错解决方法](#)

mac

[mac 上VMware安装VMware Tools选项显示灰色解决办法](#)
[brew国内加速](#)
[mac 下编译安装nginx](#)
[MAC上卸载AnyConnect后无法再重新安装问题解决](#)
[Mac 上软件推荐](#)
[Mac 上简易的ssh快捷菜单工具： shuttle](#)
[mac上charles 抓包手机端网络请求](#)
[Mac 软件卸载残留删除方法](#)
[命令窗口多开工具 xpanes](#)
[Mac上制作Linux U盘启动盘](#)

nginx

[Apache 与 Nginx 比较](#)
[Nginx 配置子目录项目](#)
[nginx+php使用open_basedir限制站点目录防止跨站](#)
[nginx log_format 配置](#)
[lnmp环境站点502与504错误分析](#)
[nginx杜绝HTTP伪请求头攻击](#)
[启用nginx status状态页详解](#)
[nginx配置auth_basic登录认证的方法](#)
[nginx 查看访问 IP 并封禁 IP 详解](#)
[nginx 工作原理](#)

[nginx 配置文件说明](#)

[nginx 禁止ip直接访问](#)

[去哪申请证书?](#)

git

[git 忽略已经跟踪的文件](#)

[git x分支强制覆盖master分支方法](#)

[git删除未跟踪文件](#)

[Gerrit代码Review入门实战](#)

[git 连接远程仓库方法](#)

[git修改远程仓库地址](#)

[将其他分支文件或提交合并到当前分支](#)

[git分支管理策略](#)

[一台电脑上不同的Git仓库账号使用不同的公私钥设置](#)

[搭建自己git服务](#)

[git 介绍](#)

shell

[shell脚本的静态检查工具shellcheck介绍](#)

[ssh 在本地执行远程主机命令](#)

[文件备份](#)

[crontab 备份](#)

[mysql 数据库备份](#)

ubuntu

[ubuntu 切换 sh 为 bash](#)

[切换国内镜像源，加速apt-get](#)

[ubuntu开启SSH服务远程登录](#)

[Ubuntu 18.04 单网卡多IP设置](#)

sentry

[搭建自己的 Sentry 服务](#)

[Sentry 实时异常信息监控系统介绍](#)

杂谈

[Linux服务器磁盘空间占满解决方法](#)

[linux 国内更新源](#)

[基于docker的php开发环境搭建（dnmp）](#)

[deployer 项目部署介绍](#)

[v2ray 搭建](#)

[Linux 系统磁盘挂载](#)

[创建自定义命令](#)

[Linux 网络常见报错及监控项](#)

Elasticsearch

[es配置](#)

[Elasticsearch 术语](#)

[ES介绍和安装](#)

[Linux 安装java环境](#)

Kibana

[Kibana 汉化方法](#)

[Kibana 简介、安装与配置](#)

[新购Linux服务器部署流程](#)

[新增用户&赋予sudo权限](#)

[配置指定命令sudo免密码使用](#)

[iptables 封禁ip](#)

[配置公私钥登录,禁止密码登录](#)

Mysql

[mysql 字段类型说明和推荐](#)

[MySQL 共享锁&排他锁](#)

[mysql日期和时间类型](#)

[mysql 事务特性以及隔离级别说明](#)

[高并发系统数据库架构设计](#)

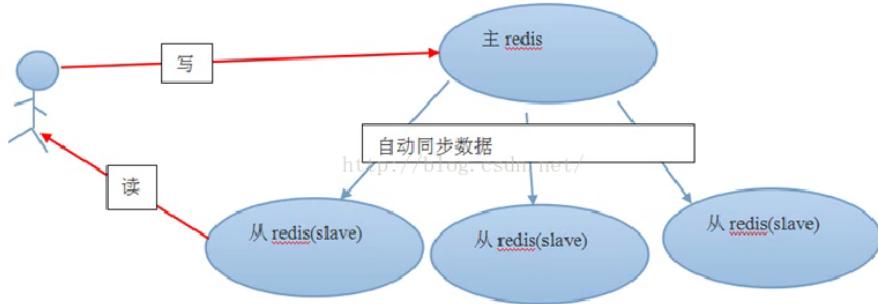
[MySql优化建议总结和注意事项](#)

[mysql定位和分析执行效率方法](#)

[count\(*\)与group by的统计问题](#)

[MySQL不能远程连接解决方法](#)

redis主从



slave默认是只读的，可以通过修改 slave-read-only 设置

如果你以前做过MySQL主从服务器的话，两相对比，你会发现Redis主从服务器不用做前期的数据同步，设置好了从服务器，简单启动就OK了。

当Slave启动后，会从Master进行一次冷启动数据同步，由Master触发BGSAVE生成RDB文件推送给Slave进行导入，导入完成后Master再将增量数据通过Redis Protocol同步给Slave。之后主从之间的数据便一直以Redis Protocol进行同步

redis主从主要用处：

- 备份
- 读写分离

redis配置认证密码

redis配置密码

1.通过配置文件进行配置

redis配置文件通常在redis.conf中，打开配置文件找到

```
1 #requirepass foobared
```

去掉行前的注释，并修改密码为所需的密码,保存文件

```
1 requirepass myredispassword
```

重启redis

这个时候尝试登录redis，发现可以登上，但是执行具体命令是提示操作不允许

```
1 redis 127.0.0.1:6379> keys *
2 (error) ERR operation not permitted
3 redis 127.0.0.1:6379> select 1
4 (error) ERR operation not permitted
```

尝试用密码登录并执行具体的命令看到可以成功执行

```
1 redis-cli -h 127.0.0.1 -p 6379 -a myredispassword
```

```
1 redis 127.0.0.1:6379> keys *
2 1) "myset"
3 2) "mysortset"
4 redis 127.0.0.1:6379> select 1
5 OK
6 redis 127.0.0.1:6379[1]> config get requirepass
7 1) "requirepass"
8 2) "myRedis"
```

2.通过命令行进行配置

```
1 redis 127.0.0.1:6379[1]> config set requirepass my_redis
2 OK
3 redis 127.0.0.1:6379[1]> config get requirepass
4 1) "requirepass"
5 2) "my_redis"
```

无需重启redis

使用第一步中配置文件中配置的老密码登录redis，会发现原来的密码已不可用，操作被拒绝

```
1 redis-cli -h 127.0.0.1 -p 6379 -a myRedis
2 redis 127.0.0.1:6379> config get requirepass
```

```
3 (error) ERR operation not permitted
```

使用修改后的密码登录redis，可以执行相应操作

```
1 redis-cli -h 127.0.0.1 -p 6379 -a my_redis
2 redis 127.0.0.1:6379> config get requirepass
3 1) "requirepass"
4 2) "my_redis"
```

尝试重启一下redis，用新配置的密码登录redis执行操作，发现新的密码失效，redis重新使用了配置文件中的密码

```
1 sudo service redis restart
2 Stopping redis-server: [OK]
3 Starting redis-server: [OK]
4 redis-cli -h 127.0.0.1 -p 6379 -a my_redis
5 redis 127.0.0.1:6379> config get requirepass
6 (error) ERR operation not permitted
7 redis-cli -h 127.0.0.1 -p 6379 -a myRedis
8 redis 127.0.0.1:6379> config get requirepass
9 1) "requirepass"
10 2) "myRedis"
```

除了在登录时通过 -a 参数制定密码外，还可以登录时不指定密码，而在执行操作前进行认证。

```
1 redis-cli -h 127.0.0.1 -p 6379
2 redis 127.0.0.1:6379> config get requirepass
3 (error) ERR operation not permitted
4 redis 127.0.0.1:6379> auth myRedispassword
5 OK
6 redis 127.0.0.1:6379> config get requirepass
7 1) "requirepass"
8 2) "myRedis"
```

3.master配置了密码， slave如何配置

若master配置了密码则slave也要配置相应的密码参数否则无法进行正常复制的。

slave中配置文件内找到如下行，移除注释，修改密码即可

```
1 #masterauth mstpassword
```

redis 过期策略

对于过期策略，一般有三种可能：

定时删除：在设置键的过期时间时，创建一个定时事件，当过期时间到达时，由事件处理器自动执行键的删除操作。

惰性删除：放任键过期不管，但是在每次从 dict 字典中取出键值时，要检查键是否过期，如果过期的话，就删除它，并返回空；如果没过期，就返回键值。

定期删除：每隔一段时间，对 expires 字典进行检查，删除里面的过期键。

定时删除

定时删除策略对内存是最友好的：因为它保证过期键会在第一时间被删除，过期键所消耗的内存会立即被释放。

这种策略的缺点是，它对 CPU 时间是最不友好的：因为删除操作可能会占用大量的 CPU 时间——在内存不紧张、但是 CPU 时间非常紧张的时候（比如说，进行交集计算或排序的时候），将 CPU 时间花在删除那些和当前任务无关的过期键上，这种做法毫无疑问会是低效的。

除此之外，目前 Redis 事件处理器对时间事件的实现方式——无序链表，查找一个时间复杂度为 $(O(N))$ ——并不适合用来处理大量时间事件。

惰性删除

惰性删除对 CPU 时间来说是最友好的：它只会在取出键时进行检查，这可以保证删除操作只会在非做不可的情况下进行——并且删除的目标仅限于当前处理的键，这个策略不会在删除其他无关的过期键上花费任何 CPU 时间。

惰性删除的缺点是，它对内存是最不友好的：如果一个键已经过期，而这个键又仍然保留在数据库中，那么 dict 字典和 expires 字典都需要继续保存这个键的信息，只要这个过期键不被删除，它占用的内存就不会被释放。

在使用惰性删除策略时，如果数据库中有非常多的过期键，但这些过期键又正好没有被访问的话，那么它们就永远也不会被删除（除非用户手动执行），这对于性能非常依赖于内存大小的 Redis 来说，肯定不是一个好消息。

举个例子，对于一些按时间点来更新的数据，比如日志（log），在某个时间点之后，对它们的访问就会大大减少，如果大量的这些过期数据积压在数据库里面，用户以为它们已经过期了（已经被删除了），但实际上这些键却没有真正的被删除（内存也没有被释放），那结果肯定是非常糟糕。

定期删除

从上面对定时删除和惰性删除的讨论来看，这两种删除方式在单一使用时都有明显的缺陷：定时删除占用太多 CPU 时间，惰性删除浪费太多内存。

定期删除是这两种策略的一种折中：

它每隔一段时间执行一次删除操作，并通过限制删除操作执行的时长和频率，籍此来减少删除操作对 CPU 时间的影响。

另一方面，通过定期删除过期键，它有效地减少了因惰性删除而带来的内存浪费。

Redis 使用的策略

Redis 使用的过期键删除策略是惰性删除加上定期删除，这两个策略相互配合，可以很好地在合理利用 CPU 时间和节约内存空间之间取得平衡。

redis 禁用O(n) 复杂度方法

某公司技术部发生2起本年度PO级特大事故，造成公司资金损失400万，原因如下：

由于工程师直接操作上线redis，执行：

```
1 keys * wxdb (此处省略) cf8*
```

这样的命令，导致redis锁住，导致CPU飙升，引起所有支付链路卡住，等十几秒结束后，所有的请求流量全部挤压到了rds数据库中，使数据库产生了雪崩效应，发生了数据库宕机事件。

redis开发规范中有一条铁律如下所示：

```
1 线上Redis禁止使用Keys正则匹配操作！
```

线上执行正则匹配操作，引起缓存雪崩，最终数据库宕机的原因：

1、redis是单线程的，其所有操作都是原子的，不会因并发产生数据异常；

2、使用高耗时的Redis命令是很危险的，会占用唯一的一个线程的大量处理时间，导致所有的请求都被拖慢。（例如时间复杂度为O(N)的KEYS命令，严格禁止在生产环境中使用）；

有上面两句作铺垫，原因就显而易见了！

运维人员进行keys *操作，该操作比较耗时，又因为redis是单线程的，所以redis被锁住；

此时QPS比较高，又来了几万个对redis的读写请求，因为redis被锁住，所以全部Hang在那；

因为太多线程Hang在那，CPU严重飙升，造成redis所在的服务器宕机；

所有的线程在redis那取不到数据，一瞬间全去数据库取数据，数据库就宕机了；

需要注意的是，同样危险的命令不仅有keys *，还有以下几组：

```
1 FLUSHALL 清空整个 Redis 服务器的数据(删除所有数据库的所有 key )。
2 FLUSHDB 清空当前数据库中的所有 key。
3 CONFIG SET 调整 Redis 服务器的配置(configuration)而无须重启。
```

因此，一个合格的redis运维或者开发，应该懂得如何禁用上面的命令。所以我一直觉得出现新闻中那种情况的原因，一般是人员的水平问题。

如何禁用redis命令

就是在redis.conf中，在SECURITY这一项中，我们新增以下命令：

```
1 rename-command FLUSHALL ""
2 rename-command FLUSHDB ""
3 rename-command KEYS ""
4 rename-command CONFIG ""
```

另外，对于FLUSHALL命令，需要设置配置文件中appendonly no，否则服务器是无法启动。

注意了，上面的这些命令可能有遗漏，大家可以查官方文档。除了Flushdb这类和redis安全隐患有关的命令意外，但凡发现时间复杂度为O(N)的命令，都要慎重，不要在生产上随便使用。例如hgetall、lrange、smembers、zrange、sinter等命令，它们并非不能使用，但这些命令的时间复杂度都为O(N)，使用这些命令需要明确N的值，否则也会出现缓存宕机。

删除大键

关于Redis大键(Key)，我们从[空间复杂性]和访问它的[时间复杂度]两个方面来定义大键。

- 1 1个大小200MB的String键(String Object最大512MB)；内存空间角度占用较大
- 2 1个包含100000000(1kw)个字段的Hash键，对应访问模式(如hgetall)时间复杂度高

因为内存空间复杂性处理耗时都非常小，测试 del 200MB String键耗时约1毫秒，而删除一个含有1kw个字段的Hash键，却会阻塞Redis进程数十秒。
若直接删除一个大键，也会面临上面的阻塞问题。

注意

redis的key设置过期时间，过期删除相当于del也会阻塞线程。所以大key不能直接设置过期时间删除

改良建议

业内建议使用scan命令来改良keys和SMEMBERS命令：

Redis2.8版本以后有了一个新命令scan，可以用来分批次扫描redis记录，这样肯定会导致整个查询消耗的总时间变大，但不会影响redis服务卡顿，影响服务使用。

具体使用，大家详情可以自己查阅下面这份文档：

<http://doc.redisfans.com/key/scan.html>

代码实例：

```
1 # php redis 扩展
2 $redis = new Redis();
3 $redis->connect(config('database.redis.default.host'), config('database.redis.default.port'));
4 $redis->auth(config('database.redis.default.password'));
5 $redis->select(config('database.redis.default.database'));
6 $redis->setOption(Redis::OPT_SCAN, Redis::SCAN_RETRY);
7 $it = NULL;
8 while($arr_keys = $redis->hScan($Key, $it)) {
9     foreach($arr_keys as $str_field => $str_value) {
10         //TODO
11     }
12 }
13
14 # laravel predis 驱动
15
16 $it = NULL;
17 while($arr_keys = Redis::hScan($Key, $it)) {
18     foreach($arr_keys[1] as $str_field => $str_value) {
```

```
19     //TODO
20
21     }
22     $it = $arr_keys[0];
23     if($arr_keys[0] == 0){
24         break;
25     }
26 }
27
28 $it = NULL;
29 while ($arr_keys = Redis::SScan('key', $it)) {
30     foreach ($arr_keys[1] as $str_field => $str_value) {
31         Redis::del($str_value);
32     }
33     $it = $arr_keys[0];
34     if($arr_keys[0] == 0){
35         break;
36     }
37 }
```

单节点redis分布式锁介绍

分布式锁

分布式锁是控制分布式系统或不同系统之间共同访问共享资源的一种锁实现，如果不同的系统或同一个系统的不同主机之间共享了某个资源时，往往需要互斥来防止彼此干扰来保证一致性。

分布式锁需要具备的特性

- 1 互斥性：在任意一个时刻，只有一个客户端可以获取锁。
- 2
- 3 无死锁：即使有一个客户端在持有锁的期间崩溃而没有主动解锁，也能保证后续客户端能加锁，加一个有效时间。
- 4
- 5 持锁人解锁：加锁和解锁必须是同一个客户端，客户端不能把别人加的锁给解了。
- 6
- 7 容错：只要大部分Redis节点都活着，客户端就可以获取和释放锁

分布式锁的实现方式

- 1 数据库
- 2 Memcached (add命令)
- 3 Redis (setnx命令)
- 4 Zookeeper (临时节点)

加解锁实例代码

```
1 /**
2  * 加锁
3  * @param $key      string 上锁key
4  * @param $value    string 锁对应的唯一值 uniqid() 或 UUID+threadId
5  * @param $lockExpire int 过期时间，单位毫秒
6  * @return int
7 */
8 function lock($key, $value, $lockExpire)
9 {
10     // key不能为空
11     if (empty($key)) {
12         throw new Exception('Key can not be empty string.', '1004');
13     }
14
15     # 加锁
16     # SET lock_key random_value NX PX 5000
17
18     $result = Redis::connection()->set($key, $value, 'NX', 'PX', $lockExpire);
19     return is_null($result) ? 0 : 1;
20 }
21
22 /**
23  * 解锁
```

```

24 * @param $key    string 上锁key
25 * @param $value   string 锁对应的唯一值
26 * @return bool
27 */
28 function unlock($key, $value)
29 {
30     $lua = "if redis.call('get', KEYS[1]) == ARGV[1]
31         then
32             return redis.call('del', KEYS[1])
33         else
34             return 0
35         end";
36     return (bool)Redis::connection()->eval($lua, 1, $key, $value);
37 }

```

加解锁过程注意点

- 1、加锁即在redis中设置(`set`)一个key,注意只有在该key不存在时设置，存在时不设置（存在即说明已经被加锁了），使用`NX`;不能用`setnx()`和`expire()`因为这样不具有原子性；
- 2、加锁设置key时，给key设置一个唯一不重复的`value`，避免解锁时解除了其他进程/线程设置的锁；
- 3、加锁时给key设置一个过期时间，用`PX`毫秒级，防止进程中断导致永远无法解锁；这个过期时间要设置的合理，大于代码运行时间，也不要太长；
- 4、解锁时用`lua脚本`，因为解锁需要判断key是否存在，且key中value是否相等再删除，要保证原子性操作才行，防止删除了其他进程的锁；

github 项目地址

[地址一](#)

[地址二](#)

单节点方式的缺点

加锁时只作用在一个Redis节点上，即使Redis通过sentinel保证高可用，如果这个master节点由于某些原因发生了主从切换，那么就会出现锁丢失的情况：

在Redis的master节点上拿到了锁；但是这个加锁的key还没有同步到slave节点；master故障，发生故障转移，slave节点升级为master节点；导致锁丢失。

分布式锁使用注意事项

- 1、加锁失败，即认为已有其他进程获得了锁，此时当前进程根据场景可以进行如下操作：接口返回错误码，请等待等；死循环获取锁，直到获取成功，每次循环之间加等待时间间隔；

压测工具 wrk 使用

wrk 项目地址:

<https://github.com/wg/wrk>

安装:

```
1 git clone https://github.com/wg/wrk.git  
2  
3 make
```

make之后，会在项目路径下生成可执行文件wrk，随后就可以用其进行HTTP压测了。可以把这个可执行文件拷贝到某个已在path中的路径，比如/usr/local/bin，这样就可以在任何路径直接使用wrk了。

使用:

```
1 wrk -t12 -c400 -d30s http://127.0.0.1:8080/index.html  
2  
3 说明:  
4 -c, --connections: total number of HTTP connections to keep open with  
5 each thread handling N = connections/threads 连接  
数  
6  
7 -d, --duration: duration of the test, e.g. 2s, 2m, 2h 请求持续时间  
8  
9 -t, --threads: total number of threads to use 线程数  
10  
11 -s, --script: LuaJIT script, see SCRIPTING  
12  
13 -H, --header: HTTP header to add to request, e.g. "User-Agent:  
wrk"  
14  
15 --latency: print detailed latency statistics  
16  
17 --timeout: record a timeout if a response is not received within  
this amount of time.
```

post请求

发送post请求时需要编写一个lua文件

http_post.lua:

```
1 wrk.method = "POST"  
2 wrk.body = '{"pushtoken":"f19Hs_5Vs","params":{"dpi":320,"device_category":"phone"}}'  
3 wrk.headers["Content-Type"] = "application/json"
```

post请求：

```
1 ./wrk -t1 -c400 -d200s --script=http_post.lua http://www.baidu.com
```

结果分析：

```
1 wrk -t8 -c200 -d30s --latency "http://www.bing.com"
2
3 Running 30s test @ http://www.bing.com (压测时间30s)
4 8 threads and 200 connections (共8个测试线程, 200个连接)
5 Thread Stats Avg Stdev Max +/- Stdev
6 (平均值) (标准差) (最大值) (正负一个标准差所占比例)
7 Latency 46.67ms 215.38ms 1.67s 95.59%
8 (延迟)
9 Req/Sec 7.91k 1.15k 10.26k 70.77%
10 (处理中的请求数)
11 Latency Distribution (延迟分布)
12 50% 2.93ms
13 75% 3.78ms
14 90% 4.73ms
15 99% 1.35s (99分位的延迟)
16 1790465 requests in 30.01s, 684.08MB read (30.01秒内共处理完成了17904
65个请求, 读取了684.08MB数据)
17 Requests/sec: 59658.29 (平均每秒处理完成59658.29个请求)
18 Transfer/sec: 22.79MB (平均每秒读取数据22.79MB)
```

注意：

wrk 使用的是 HTTP/1.1，缺省开启的是长连接，而 ab 使用的是 HTTP/1.0，缺省开启的是短链接。

用 wrk 测试短链接：

```
1 wrk -H "Connection: Close" -c 100 -d 10 http://domain/path
```

也就是说通过参数「H」传递一个自定义的 Connection 请求头来关闭长链接。

vim快捷键

```
1 gg:命令将光标移动到文档开头  
2  
3 G:命令将光标移动到文档末尾  
4  
5 0:跳转到行首  
6  
7 shift + e 或 $:跳转到行尾  
8  
9 dw:删除光标之后的单词剩余部分。  
10  
11 d$:删除光标之后的该行剩余部分。  
12  
13 dd:删除当前行。
```

链路测试工具mtr

mtr 命令行工具

mtr (My traceroute) 也是几乎所有 Linux 发行版本预装的网络测试工具。它把 ping 和 traceroute 的功能并入了同一个工具中，所以功能更强大。

mtr 默认发送 ICMP 数据包进行链路探测。可以通过 -u 参数来指定使用 UDP 数据包用于探测。

相对于 traceroute 只会做一次链路跟踪测试，mtr 会对链路上的相关节点做持续探测并给出相应的统计信息。所以，mtr 能避免节点波动对测试结果的影响，所以其测试结果更正确，建议优先使用。

用法说明：

```
1 mtr [-hvrcglspni46] [-help] [-version] [-report]           [-re
      port-cycles=COUNT] [-curses] [-gtk]                  [-raw] [-split] [-
      no-dns] [-address interface]           [-psize=bytes/-s bytes]
      [-interval=SECONDS] HOSTNAME [PACKETSIZE]
```

Mac 上安装使用

```
1 # 安装
2 brew install mtr
3
4 # 启用命令别名
5 vim ~/.profile
6 alias mtr="sudo /usr/local/sbin/mtr"
7
8 # 配置path变量
9 vim ~/.bash_profile
10 export PATH=$PATH:/usr/local/sbin
11 source ~/.profile
12
13 # 使其生效
14 source ~/.profile
15 source ~/.bash_profile
16
17 # 使用
18 mtr baidu.com
```

示例输出：

My traceroute [v0.85]								Mon Nov 19 23:40:46 2018
Host	Packets			Pings				quit
	Loss%	Snt	Last	Avg	Best	Wrst	StDev	
1. ???	0.0%	7	1.9	2.6	1.6	5.7	1.4	
2. 11.211.20.5	0.0%	7	1.1	1.1	1.1	1.2	0.0	
3. 11.211.20.106	0.0%	7	1.3	1.4	1.2	2.1	0.0	
4. 116.251.82.174	0.0%	7	1.3	3.6	1.3	17.2	6.0	
5. 203.12.205.249	0.0%	7	1.3	1.6	1.3	2.8	0.0	
6. 59.43.248.113	0.0%	7	1.3	1.6	1.3	2.8	0.0	
7. 59.43.246.225	0.0%	7	41.4	56.8	41.4	142.6	37.9	
8. 59.43.188.77	0.0%	7	41.6	42.8	41.5	50.0	3.1	
9. 59.43.131.253	0.0%	7	42.8	44.3	42.8	51.2	3.1	
10. 59.43.80.2	0.0%	7	60.2	45.1	42.5	60.2	6.7	
11. ???								
12. 218.30.25.122	20.0%	6	43.1	43.2	43.1	43.2	0.0	
13. ???								
14. 220.181.17.94	0.0%	6	43.9	43.5	43.3	43.9	0.0	
15. ???								
16. ???								
17. ???								
18. 220.181.57.216	0.0%	6	43.7	43.8	43.7	44.0	0.0	

常见可选参数说明：

- 1 -r 或 -report: 以报告模式显示输出。
- 2 -p 或 -split: 将每次追踪的结果分别列出来，而非如 -report 统计整个结果。
- 3 -s 或 -psize: 指定 ping 数据包的大小。
- 4 -n 或 -no-dns: 不对 IP 地址做域名反解析。
- 5 -a 或 -address: 设置发送数据包的 IP 地址。用于主机有多个 IP 时。
- 6 -4: 只使用 IPv4 协议。
- 7 -6: 只使用 IPv6 协议。
- 8 另外，也可以在 mtr 运行过程中，输入相应字母来快速切换模式，比如：
- 9
- 10 ? 或 h: 显示帮助菜单。
- 11 d: 切换显示模式。
- 12 n: 切换启用或禁用 DNS 域名解析。
- 13 u: 切换使用 ICMP 或 UDP 数据包进行探测。

返回结果说明：

默认配置下，返回结果中各数据列的说明：

- 1 第一列 (Host) : 节点 IP 地址和域名。如前面所示，按 n 键可以切换显示。
- 2 第二列 (Loss%) : 节点丢包率。
- 3 第三列 (Snt) : 每秒发送数据包数。默认值是 10，可以通过参数 -c 指定。
- 4 第四列 (Last) : 最近一次的探测延迟值。
- 5 第五、六、七列 (Avg、Best、Wrst) : 分别是探测延迟的平均值、最小值和最大值。
- 6 第八列 (StDev) : 标准偏差。越大说明相应节点越不稳定。

链路测试结果分析简要说明

My traceroute [v0.75]							
Host							
		Order of fields		Pings			
		Packets		Loss%	Snt	Last	Avg
				Best	Wrst	StDev	
1. ???				100.0%	88	2.1	2.4
2. 192.168.17.20	区域 A： 客户端本地网络			100.0%	88	112.7	162.7
3. 111.1.20.41				0.0%	88	1.4	3.0
4. 111.1.34.197				60.0%	88	116.3	70.4
5. 211.138.114.25				60.0%	88	112.1	77.8
6. 211.138.114.2	区域 B： 运营商骨干网络			40.0%	88	112.1	72.4
211.138.114.66				0.0%	88	113.3	73.5
211.138.128.134				0.0%	88	112.1	72.6
211.138.114.70				0.0%	88	112.1	73.4
7. 42.120.244.186	区域 C： 目标服务器本地网络			0.0%	88	119.1	2.6
42.120.244.194							
42.120.244.198							
42.120.244.190							
8. 42.120.244.246	区域 D： 链路负载均衡			0.0%	88	141.6	4.4
9. ???				0.0%	88	131.6	3.4
10. 223.5.5.5							

网络区域

正常情况下，从客户端到目标服务器的整个链路，会显著的包含如下区域：

客户端本地网络（本地局域网和本地网络提供商网络）：如前文链路测试结果示例图中的区域 A。如果该区域出现异常，如果是客户端本地网络相关节点出现异常，则需要对本地网络进行相应排查分析。否则，如果是本地网络提供商网络相关节点出现异常，则需要向当地运营商反馈问题。

运营商骨干网络：如前文链路测试结果示例图中的区域 B。如果该区域出现异常，可以根据异常节点 IP 查询归属运营商，然后直接或通过阿里云售后技术支持，向相应运营商反馈问题。

目标服务器本地网络（目标主机归属网络提供商网络）：如前文链路测试结果示例图中的区域 C。如果该区域出现异常，则需要向目标主机归属网络提供商反馈问题。

链路负载均衡

如前文链路测试结果示例图中的区域 D 所示。如果中间链路某些部分启用了链路负载均衡，则 mtr 只会对首尾节点进行编号和探测统计。中间节点只会显示相应的 IP 或域名信息。

结合Avg（平均值）和 StDev（标准偏差）综合判断

由于链路抖动或其它因素的影响，节点的 Best 和 Worst 值可能相差很大。而 Avg（平均值）统计了自链路测试以来所有探测的平均值，所以能更好的反应出相应节点的网络质量。

而 StDev（标准偏差值）越高，则说明数据包在相应节点的延时值越不相同（越离散）。所以，标准偏差值可用于协助判断 Avg 是否真实反映了相应节点的网络质量。例如，如果标准偏差很大，说明数据包的延迟是不确定的。可能某些数据包延迟很小（例如：25ms），而另一些延迟却很大（例如：350ms），但最终得到的平均延迟反而可能是正常的。所以，此时 Avg 并不能很好的反应出实际的网络质量情况。

综上，建议的分析标准是：

如果 StDev 很高，则同步观察相应节点的 Best 和 Wrst，来判断相应节点是否存在异常。

如果 StDev 不高，则通过 Avg 来判断相应节点是否存在异常。

注：上述 StDev “高”或者“不高”，并没有具体的时间范围标准。而需要根据同一节点其它列的延迟值大小来进行相对评估。比如，如果 Avg 为 30ms，那么，当 StDev 为 25ms，则认为是很高的偏差。而如果 Avg 为 325ms，则同样的 StDev (25ms)，反而认为是不高的偏差。

Loss%（丢包率）的判断

任一节点的 Loss%（丢包率）如果不为零，则说明这一跳网络可能存在故障。导致相应节点丢包的原因通常有两种：

运营商基于安全或性能需求，人为限制了节点的 ICMP 发送速率，导致丢包。

节点确实存在异常，导致丢包。

可以结合异常节点及其后续节点的丢包情况，来判定丢包原因：

如果随后节点均没有丢包，则通常说明异常节点丢包是由于运营商策略限制所致。可以忽略相关丢包。如前文链路测试结果示例图中的第 2 跳所示。

如果随后节点也出现丢包，则通常说明异常节点确实存在网络异常，导致丢包。如前文链路测试结果示例图中的第 5 跳所示。

另外，需要说明的是，前述两种情况可能同时发生。即相应节点既存在策略限速，又存在网络异常。对于这种情况，如果异常节点及其后续节点连续出现丢包，而且各节点的丢包率不同，则通常以最后几跳的丢包率为准。如前文链路测试结果示例图所示，在第 5、6、7 跳均出现了丢包。所以，最终丢包情况，以第 7 跳的 40% 作为参考。

关于延迟

延迟跳变

如果在某一跳之后延迟明显陡增，则通常判断该节点存在网络异常。如前文链路测试结果示例图所示，从第 5 跳之后的后续节点延迟明显陡增，则推断是第 5 跳节点出现了网络异常。

不过，高延迟并不一定完全意味着相应节点存在异常。如前文链路测试结果示例图所示，第 5 跳之后，虽然后续节点延迟明显陡增，但测试数据最终仍然正常到达了目的主机。所以，延迟大也有可能是在数据回包链路中引发的。所以，最好结合反向链路测试一并分析。

ICMP 限速导致延迟增加

ICMP 策略限速也可能会导致相应节点的延迟陡增，但后续节点通常会恢复正常。如前文链路测试结果示例图所示，第 3 跳有 100% 的丢包率，同时延迟也明显陡增。但随后节点的延迟马上恢复了正常。所以判断该节点的延迟陡增及丢包是由于策略限速所致。

转载自：https://help.aliyun.com/knowledge_detail/40573.html

系统监控工具 dstat 介绍

dstat:

多功能系统资源统计生成工具（versatile tool for generating system resource statistics）。在获取的信息上有点类似于top、free、iostat、vmstat等多个工具的合集，官方解释为vmstat、iostat、ifstat等工具的多功能替代品，且添加了许多额外的功能（Dstat is a versatile replacement for vmstat, iostat and ifstat. Dstat overcomes some of the limitations and adds some extra features.）；其结果可以保持到csv文件，使用脚本或第三方工具对性能进行分析利用（如通过监控平台监控，也可以保持到数据库）。在Centos 6.x系统上安装基本服务器即默认安装，而在其他操作系统可能需要手动安装。

安装

```
1 #centos
2 yum install dstat
3
4 # ubuntu
5 apt-get install dstat
```

使用

dstat的默认选项

与许多命令一样，dstat命令有默认选项，执行dstat命令不加任何参数，它默认会收集-cpu-、-disk-、-net-、-paging-、-system-的数据，一秒钟收集一次。默认输入 dstat 等于输入了dstat -cdngy 1或dstat -a 1。

dstat的常用选项：

dstat的用法如下：

```
1 dstat [-afv] [options..] [delay [count]]
```

使用 dstat -h查看全部选项，这里不逐一列举，下面简单介绍下常用选项

常用选项如下：

```
1 直接跟数字，表示#秒收集一次数据，默认为一秒；dstat 5表示5秒更新一次
2
3 -c,--cpu    统计CPU状态，包括 user, system, idle (空闲等待时间百分比) , wait
              (等待磁盘IO) , hardware interrupt (硬件中断) , software interrupt (软件中
              断) 等；
4
5 -d, --disk 统计磁盘读写状态
6
7 -D total,sda 统计指定磁盘或汇总信息
8
9 -l, --load 统计系统负载情况，包括1分钟、5分钟、15分钟平均值
10
11 -m, --mem 统计系统物理内存使用情况，包括used, buffers, cache, free
```

```
12
13 -s, --swap 统计swap已使用和剩余量
14
15 -n, --net 统计网络使用情况, 包括接收和发送数据
16
17 -N eth1,total 统计eth1接口汇总流量
18
19 -r, --io 统计I/O请求, 包括读写请求
20
21 -p, --proc 统计进程信息, 包括runnable、uninterruptible、new
22
23 -y, --sys 统计系统信息, 包括中断、上下文切换
24
25 -t 显示统计时间, 对分析历史数据非常有用
26
27 --fs 统计文件打开数和inodes数
28
29 以上这些就是最常用的选项, 而一般都组合使用, 个人比较常用的是:
30
31 dstat -cprndl 5
```

监测界面各参数含义

Procs

r:运行的和等待(CPU时间片)运行的进程数, 这个值也可以判断是否需要增加CPU(长期大于cpu核数)
b:处于不可中断状态的进程数, 常见的情况是由IO引起的

Memory

swpd: 切换到交换内存上的内存(默认以KB为单位)。如果 swpd 的值不为0, 或者还比较大, 比如超过100M了, 但是 si, so 的值长期为 0, 这种情况我们可以不用担心, 不会影响系统性能。

free: 空闲的物理内存

buff: 作为buffer cache的内存, 对块设备的读写进行缓冲

cache: 作为page cache的内存, 文件系统的cache。如果 cache 的值大的时候, 说明cache住的文件数多, 如果频繁访问到的文件都能被cache住, 那么磁盘的读IO bi 会非常小。

Swap

si: 交换内存使用, 由磁盘调入内存

so: 交换内存使用, 由内存调入磁盘

内存够用的时候, 这2个值都是0, 如果这2个值长期大于0时, 系统性能会受到影响。磁盘IO和CPU资源都会被消耗。

我发现有些朋友看到空闲内存(free)很少或接近于0时, 就认为内存不够用了, 实际上不能光看这一点的, 还要结合si,so, 如果free很少, 但是si,so也很少(大多时候是0), 那么不用担心, 系统性能这时不会受到影响的。

磁盘IO

bi: 从块设备读入的数据总量(读磁盘) (KB/s)

bo: 写入到块设备的数据总量(写磁盘) (KB/s)

注:随机磁盘读写的时候, 这2个值越大 (如超出1M), 能看到CPU在IO等待的值也会越大

System

in: 每秒产生的中断次数
cs: 每秒产生的上下文切换次数
上面这2个值越大，会看到由内核消耗的CPU时间会越多

Cpu

usr: 用户进程消耗的CPU时间百分比
us 的值比较高时，说明用户进程消耗的CPU时间多，但是如果长期超过50% 的使用，那么我们就该考虑优化程序算法或者进行加速了(比如 PHP/Perl)

sys: 内核进程消耗的CPU时间百分比
sys 的值高时，说明系统内核消耗的CPU资源多，这并不是良性表现，我们应该检查原因。

wai: IO等待消耗的CPU时间百分比
wa 的值高时，说明IO等待比较严重，这可能是由于磁盘大量作随机访问造成，也有可能是磁盘的带宽出现瓶颈(块操作)。
idl: CPU处在空闲状态时间百分比

dstat的高级用法

找出占用资源最高的进程和用户

--top-(io|bio|cpu|cputime|cputime-avg|mem) 通过这几个选项，可以看到具体是那个用户那个进程占用了相关系统资源，对系统调优非常有效。如查看当前占用I/O、cpu、内存等最高的进程信息可以使用

```
1 dstat --top-mem --top-io --top-cpu
```

图片

获取其他应用信息：

dstat除了可以获取系统关键信息外，还可以获取其他应用信息，如通过下列选项，可以获取到其他一些常用应用信息：

--postfix 显示postfix队列大小
--sendmail 显示sendmail队列大小
--ntp 显示ntp服务器时间
--nfs3 获取nfs客户端信息
--nfsd3 获取nfs服务器信息，不过nfs服务器版本需为第三版才可以，该选项还有更多用法，可以参考man帮助获取
--mysql5- (cmds|conn|io|keys) 获取mysql5相关信息

Linux之scp命令

Linux scp命令

Linux scp 命令用于 Linux 之间复制文件和目录,基于 ssh 登陆进行安全的远程文件拷贝。本地和远程都要支持scp命令才能进行文件拷贝。

scp 是加密的, rcp 是不加密的, scp 是 rcp 的加强版。

scp语法

```
1 scp [-1246BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file]
2 [-l limit] [-o ssh_option] [-P port] [-S program]
3 [[user@]host1:]file1 [...] [[user@]host2:]file2
```

简易写法:

```
1 scp [可选参数] file_source file_target
```

scp参数说明:

```
1 -1: 强制scp命令使用协议ssh1
2 -2: 强制scp命令使用协议ssh2
3 -4: 强制scp命令只使用IPv4寻址
4 -6: 强制scp命令只使用IPv6寻址
5 -B: 使用批处理模式 (传输过程中不询问传输口令或短语)
6 -C: 允许压缩。 (将-C标志传递给ssh, 从而打开压缩功能)
7 -p: 保留原文件的修改时间, 访问时间和访问权限。
8 -q: 不显示传输进度条。
9 -r: 递归复制整个目录。
10 -v: 详细方式显示输出。scp和ssh(1)会显示出整个过程的调试信息。这些信息用于调试连接, 验证和配置问题。
11 -c cipher: 以cipher将数据传输进行加密, 这个选项将直接传递给ssh。
12 -F ssh_config: 指定一个替代的ssh配置文件, 此参数直接传递给ssh。
13 -i identity_file: 从指定文件中读取传输时使用的密钥文件, 此参数直接传递给ssh。
14 -l limit: 限定用户所能使用的带宽, 以Kbit/s为单位。
15 -o ssh_option: 如果习惯于使用ssh_config(5)中的参数传递方式,
16 -P port: 注意是大写的P, port是指定数据传输用到的端口号
17 -S program: 指定加密传输时所使用的程序。此程序必须能够理解ssh(1)的选项。
```

scp实例

1、从本地复制到远程

```
1 scp local_file remote_username@remote_ip:remote_folder
2 或者
3 scp local_file remote_username@remote_ip:remote_file
4 或者
5 scp local_file remote_ip:remote_folder
```

```
6 或者
7 scp local_file remote_ip:remote_file
8
9 复制目录命令格式:
10
11 scp -r local_folder remote_username@remote_ip:remote_folder
12 或者
13 scp -r local_folder remote_ip:remote_folder
```

2、从远程复制到本地

从远程复制到本地，只要将从本地复制到远程的命令的后2个参数调换顺序即可，如下实例

```
1 scp root@www.runoob.com:/home/root/others/music /home/space/music/1.mp3
2 scp -r www.runoob.com:/home/root/others/ /home/space/music/
```

使用注意事项

1.如果远程服务器防火墙有为scp命令设置了指定的端口，我们需要使用 -P 参数来设置命令的端口号，命令格式如下：

```
1 #scp 命令使用端口号 4588
2 scp -P 4588 remote@www.runoob.com:/usr/local/sin.sh /home/administrat
or
```

2.使用scp命令要确保使用的用户具有可读取远程服务器相应文件的权限，否则scp命令是无法起作用的。

htop 命令使用详解

htop的使用简介

大家可能对top监控软件比较熟悉， htop算是top的增强版，相比top其有着很多自身的优势。如下：

- 1 两者相比起来， top比较繁琐
- 2 默认支持图形界面的鼠标操作
- 3 可以横向或纵向滚动浏览进程列表，以便看到所有的进程和完整的命令行
- 4 杀进程时不需要输入进程号等

安装

Htop的安装，既可以通过源码包编译安装，也可以配置好yum源后网络下载安装

```
1 # centos
2 yum install htop
3
4 # ubuntu
5 apt-get install htop
```

htop的使用

安装完成后，命令行中直接敲击htop命令，即可进入htop的界面

```
CPU[|||||] Tasks: 64, 234 thr; 2 running
Mem[|||||||||] Load average: 0.13 0.11 0.03
Swap[|||||] Uptime: 150 days(1), 13:03:08

PID USER PRI NI TIME COMMAND
891 root 20 0 19176 67688 3828 S 2.0 3.4 1h22:58 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
1247 root 20 0 19176 67688 3828 S 2.0 3.4 1h22:58 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
2343 gcloudapis1 20 0 19176 3112 5988 S 0.7 1.6 2d23:55 /usr/local/segis/segis_client/segis_10_59@AliyunRun
18694 root 0 -20 1588 3112 5988 S 0.7 1.6 15:34:05 /usr/local/segis/segis_client/segis_10_59@AliyunRun
18699 root 0 -20 1588 3112 5988 S 0.7 1.6 27:11:57 /usr/local/segis/segis_client/segis_10_59@AliyunRun
18698 root 0 -20 1588 3112 5988 S 0.7 1.6 3h29:38 /usr/local/segis/segis_client/segis_10_59@AliyunRun
901 root 20 0 19176 67688 3828 S 2.0 3.4 1h22:58 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
1271 root 20 0 19176 67688 3828 S 2.0 3.4 3h37:48 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
18767 root 0 -20 1588 3112 5988 S 0.7 1.6 1h47:58 /usr/local/segis/segis_client/segis_10_59@AliyunRun
16734 root 20 0 1730 46452 5268 S 0.7 2.3 2h49:52 /usr/bin/alicloud-salt-minion
5232 public 20 0 1240 47448 5268 S 0.7 1.6 1h49:52 /usr/bin/alicloud-salt-minion
6993 public 20 0 11740 47448 5268 S 0.7 2.4 31:25:48 PM v3.2.2: God Daemon (/home/public/.pm2)
17580 baokun 20 0 4376 5556 7988 S 0.0 0.4 4h1:35 /super-siner-proxy
28625 mysql 20 0 11588 2834 7184 S 0.0 14.2 2:38.78 /usr/local/mysql --basedir=/usr/local/mysql/var --datadir=/usr/local/mysql/var --plugin-dir=/usr/local/m
901 root 20 0 19176 67688 3828 S 2.0 3.4 1h22:58 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
9382 root 0 -20 44764 3776 2612 S 0.0 0.2 5:28:05 /usr/local/redis/bin/redis-server *:6379
1 root 20 0 37884 7008 S 0.0 0.2 1:49:42 /lib/systemd/systemd --system --deserialize 19
197 root 20 0 84580 2082 17988 S 0.0 0.2 8:41:12 /lib/systemd/systemd-journal
544 syslog 20 0 2580 47448 5268 S 0.0 0.2 1:49:42 /lib/systemd/systemd-journal
564 syslog 20 0 2580 4742 972 S 0.0 0.1 9:00:00 /usr/bin/rsyslog -n
565 syslog 20 0 2580 4742 972 S 0.0 0.1 9:03:52 /usr/bin/rsyslog -n
408 syslog 20 0 43240 1812 972 S 0.0 0.1 9:07:48 /usr/bin/rsyslog -n
585 gcloudapi 20 0 36264 3516 1448 S 0.0 0.2 9:11:11 /lib/systemd/systemd-logind
513 root 20 0 259H 888 S 0.0 0.0 3:08:08 /usr/lib/accountsservice/accounts-daemon
512 root 20 0 259H 888 S 0.0 0.0 3:09:02 /usr/lib/accountsservice/accounts-daemon
514 account 20 0 259H 888 S 0.0 0.0 3:09:02 /usr/lib/accountsservice/accounts-daemon
529 daemon 20 0 26848 888 6088 S 0.0 0.0 9:00:22 /usr/bin/atd -f
647 root 20 0 16088 866 S 0.0 0.0 9:00:45 /sbin/dhcclient -l -v -pf /run/dhcclient.eth0.pid -lf /var/lib/dhcp/dhcclient.eth0.leases -I -df /var/lib/dhcp/dhc
802 K 20 0 19176 67688 3828 S 2.0 3.4 1h22:58 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
829 root 20 0 19176 67688 3828 S 0.0 0.0 9:00:01 /sbin/agetty -noclear tt1 linux
848 root 20 0 65480 344 1768 S 0.0 0.2 9:00:45 /usr/bin/aliyun-service -d
849 root 20 0 65480 344 1768 S 0.0 0.2 9:00:54 /usr/bin/aliyun-service -d
864 root 20 0 65480 344 1768 S 0.0 0.2 9:01:03 /usr/bin/aliyun-service -d
865 root 20 0 65480 344 1768 S 0.0 0.2 1h22:39 /usr/bin/aliyun-service -d
866 root 20 0 65480 344 1768 S 0.0 0.2 1h23:39 /usr/bin/aliyun-service -d
838 root 20 0 65480 344 1768 S 0.0 0.2 1h31:28 /usr/bin/aliyun-service -d
862 root 20 0 19176 67688 3828 S 2.0 3.4 1h22:58 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
928 root 20 0 19176 67688 3828 S 0.0 3.4 1h23:31 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
936 root 20 0 19176 67688 3828 S 0.0 3.4 1h54:55 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
937 root 20 0 19176 67688 3828 S 0.0 3.4 1h54:56 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
946 root 20 0 19176 67688 3828 S 0.0 3.4 1h54:57 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
951 root 20 0 19176 67688 3828 S 0.0 3.4 1h54:58 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
952 root 20 0 19176 67688 3828 S 0.0 3.4 1h54:59 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
1131 root 20 0 19176 67688 3828 S 0.0 3.4 1h1:41 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
1132 root 20 0 19176 67688 3828 S 0.0 3.4 1h1:49 /usr/local/cloudmonitor/jre/bin/java -Djava.compiler:none -XX:-UseGCOverheadLimit -XX:NewRatio=1 -XX:SurvivorRa
F11:10 0 2580 47448 5268 S 0.0 0.0 7:09:59 /usr/bin/alicloud-salt-minion
```

各项从上至下分别说明如下：

左边部分从上至下，分别为，cpu、内存、交换分区的使用情况，右边部分为：Tasks为进程总数，当前运行的进程数、Load average为系统1分钟，5分钟，10分钟的平均负载情况、Uptime为系统运行的时间。

- 1 PID: 进行的标识号
- 2 USER: 运行此进程的用户
- 3 PRI: 进程的优先级
- 4 NI: 进程的优先级别值，默认的为0，可以进行调整

```
5 VIRT: 进程占用的虚拟内存值  
6 RES: 进程占用的物理内存值  
7 SHR: 进程占用的共享内存值  
8 S: 进程的运行状况, R表示正在运行、S表示休眠、等待唤醒、Z表示僵死状态  
9 %CPU: 该进程占用的CPU使用率  
10 %MEM: 该进程占用的物理内存和总内存的百分比  
11 TIME+: 该进程启动后占用的总的CPU时间  
12 COMMAND: 进程启动的启动命令名称
```

htop快捷键使用说明

F1: 显示帮助信息

```
htop 2.0.1 - (C) 2004-2016 Hisham Muhammad  
Released under the GNU GPL. See 'man' page for more info.  
  
CPU usage bar: [low-priority/normal/kernel/virtualiz          used/] [used/total]  
Memory bar:    [used/buffers/cache                      used/total]  
Swap bar:      [used                           used/total]  
Type and layout of header meters are configurable in the setup screen.  
  
Status: R: running; S: sleeping; T: traced/stopped; Z: zombie; D: disk sleep  
Arrows: scroll process list           Space: tag process  
Digits: incremental PID search      c: tag process and its children  
       F3 /: incremental name search   U: untag all processes  
       F4 \: incremental name filtering F9 k: kill process/tagged processes  
F5 t: tree view                   F7 ]: higher priority (root only)  
       p: toggle program path        F8 [: lower priority (+ nice)  
       u: show processes of a single user   a: set CPU affinity  
       H: hide/show user process threads e: show process environment  
       K: hide/show kernel threads     i: set IO priority  
       F: cursor follows process      l: list open files with lsof  
F6 + -: expand/collapse tree      s: trace syscalls with strace  
P M T: sort by CPU%, MEM% or TIME F2 S: setup  
       I: invert sort order         F1 h: show this help screen  
F6 >: select sort column         F10 q: quit  
Press any key to return.
```

```
1 h, ?, F1    查看htop使用说明  
2  
3 S, F2  htop 设定  
4  
5 /, F3    搜索进程  
6  
7 \, F4 增量进程过滤器  
8  
9 t, F5 显示树形结构  
10  
11 <, >, F6 选择排序方式  
12  
13 [, F7 可减少nice值可以提高对应进程的优先级  
14  
15 ], F8 可增加nice值, 降低对应进程的优先级  
16  
17 k, F9 可对进程传递信号  
18  
19 q, F10 结束htop  
20  
21 u 只显示一个给定的用户的过程  
22  
23 U 取消标记所有的进程  
24  
25 H 显示或隐藏用户线程  
26  
27 K 显示或隐藏内核线程  
28
```

```
29 F 跟踪进程
30
31 P 按CPU 使用排序
32
33 M 按内存使用排序
34
35 T 按Time+ 使用排序
36
37 l 显示进程打开的文件
38
39 I 倒转排序顺序
40
41 s 选择某进程, 按s:用strace追踪进程的系统调用
42
43 F2 Htop设定, 鼠标点击Setup或者按下F2 之后进入htop 设定的页面,
```

crontab使用详解

crontab 执行的命令文件等都要用绝对路径

文件位置：

位置一般在/var/spool/cron/下，如果你是root用户，那下面有个root文件，建议日常备份，避免误删除导致crontab 文件丢失；

常用命令：

```
1 server crond status    : 查看crontab运行状态
2 service crond restart  : 重启crontab服务 (每次修改后重启)
3 service crond stop     : 停止crontab服务
4 service crond start    : 启动crontab服务
5 crontab -e      : 编辑crontab命令
6 crontab -l      : 查看crontab命令
7 crontab -r      : 删除所有cron任务
8 crontab file [-u user] : 用指定的文件替代目前的crontab。
9 chkconfig -level 35 crond on  : 让crond在开机时运行，改变其运行级别
```

cron文件格式：

```
1 crontab文件的格式: M H D m d cmd。
2 基本格式：
3 * * * * * command
4 分 时 日 月 周 命令
5 M: 分钟 (0-59)。每分钟用*或者 */1表示
6 H: 小时 (0-23)。(0表示0点)
7 D: 天 (1-31)。
8 m: 月 (1-12)。
9 d: 一星期内的天 (0~6, 0为星期天)。
10 cmd: 要运行的程序，程序被送入sh执行，这个shell只有USER, HOME, SHELL这三个环境变量
11
12 说明：
13
14 crontab 是 用来让使用者在固定时间或固定间隔执行程序之用，换句话说，也就是类似使用者的时程表。
15 -u user 是指设定指定 user 的时程表，这个前提是用户必须要有其权限(比如说是 root)才能够指定他人的时程表。
16 如果不使用 -u user 的话，就是表示设定自己的时程表。
17
18 星号 (*) 可以用来代表所有有效的值。譬如，月份值中的星号意味着在满足其它制约条件后每月都执行该命令。
19 整数间的短线 (-) 指定一个整数范围。譬如，1-4意味着整数 1、2、3、4。
20 用逗号 (,) 隔开的一系列值指定一个列表。譬如，3, 4, 6, 8标明这四个指定的整数。
21 正斜线 (/) 可以用来指定间隔频率。在范围后加上 /<integer>意味着在范围内可以跳过 integer。譬如，0-59/2可以用来在分钟字段定义每两分钟。间隔频率值还可以和星号一起使用。例如，*/3的值可以用在月份字段中表示每三个月运行一次任务。
22 开头为井号 (#) 的行是注释，不会被处理。
```

```
1 时程表的格式如下：
2 f1 f2 f3 f4 f5 program
3 其中 f1 是表示分钟，f2 表示小时，f3 表示一个月份中的第几日，f4 表示月份，f5 表示一个星期中的第几天。program 表示要执行的程序。
```

- 4 当 f1 为 * 时表示每分钟都要执行 program, f2 为 * 时表示每小时都要执行程序, 其余类推
- 5 当 f1 为 a-b 时表示从第 a 分钟到第 b 分钟这段时间内要执行, f2 为 a-b 时表示从第 a 到第 b 小时都要执行, 其余类推
- 6 当 f1 为 */n 时表示每 n 分钟个时间间隔执行一次, f2 为 */n 表示每 n 小时个时间间隔执行一次, 其余类推
- 7 当 f1 为 a, b, c,... 时表示第 a, b, c,... 分钟要执行, f2 为 a, b, c,... 时表示第 a, b, c...个小时要执行, 其余类推
- 8 使用者也可以将所有的设定先存放在档案 file 中, 用 crontab file 的方式来设定时程表。

实例：

```

1 #每天早上7点执行一次 /bin/ls :
2 0 7 * * * /bin/ls
3
4 #在 12 月内, 每天的早上 6 点到 12 点中, 每隔3个小时执行一次 /usr/bin/backup :
5 0 6-12/3 * 12 * /usr/bin/backup
6
7 #周一到周五每天下午 5:00 寄一封信给 alex@domain.name :
8 0 17 * * 1-5 mail -s "hi" alex@domain.name < /tmp/maildata
9
10 #每月每天的0-23点每隔2个小时的20分执行 echo "haha"
11 20 0-23/2 * * * echo "haha"
12
13 #晚上11点到早上8点之间每两个小时, 和早上8点
14 0 23-7/2, 8 * * * date
15
16 #每月1、10、22日的4 : 45重启apache。
17 45 4 1,10,22 * * /usr/local/etc/rc.d/lighttpd restart
18
19 #每周六、周日的1 : 10重启apache。
20 10 1 * * 6,0 /usr/local/etc/rc.d/lighttpd restart
21
22 #注意单纯echo, 从屏幕上看不到任何输出, 因为cron把任何输出都email到root的信箱了。
23 #在末尾加上 >>文件名 会将输出写入文件
24 0 6 * * * echo "Good morning." >> /tmp/test.txt 2>&1
25
26 #每两个小时, 输出"Have a break now."到文件/tmp/test.txt 2>&1
27 0 */2 * * * echo "Have a break now." >> /tmp/test.txt 2>&1
28
29 #每分钟执行index.php文件, 输出信息不作处理, 不写入文件或发送邮件
30 * * * * /usr/bin/php /mnt/hgfs/webserver/test/index.php > /dev/nul
l 2>&1
31
32 #每分钟执行test.py文件, 输出信息写入到error.log文件
33 * * * * python /mnt/hgfs/python_work/study/test.py >> /mnt/hgfs
/webserver/test/error.log 2>&1

```

注意：

- 1 1、当程序在你所指定的时间执行后, 系统会寄一封信给你, 显示该程序执行的内容, 若是你不希望收到这样的信,
- 2 请在每一行空一格之后加上 >> /dev/null 2>&1 即可;
- 3 2、对于有输出或文件写入的执行命令, 若要输出可用: >> /tmp/test.txt 2>&1 或 > /tm
p/test.txt 2>&1;
- 4 前者可记录每次执行输出, 后者只记录最新执行输出, 替换前一次输出。所以在需要输出信息

时，最好用：

```
5    >> /tmp/test.txt 2>&1
6 3、若想按顺序调度多个任务，可以吧要调度的文件写入一个 shell脚本中（.sh文件），再用
   任务调度运行这个脚本
7 4、2>&1 表示不仅命令行正常的输出保存到app.log中，产生错误信息的输出也保存到app.lo
   g文件中；
```

crontab实现秒级执行任务：

用crontab+sleep实现以秒执行任务

```
1 crontab -e
2 * * * * * /bin/date >>/tmp/date.txt
3 * * * * * sleep 10; /bin/date >>/tmp/date.txt //暂停10秒后执行命令
4 * * * * * sleep 20; /bin/date >>/tmp/date.txt
5 * * * * * sleep 30; /bin/date >>/tmp/date.txt
6 * * * * * sleep 40; /bin/date >>/tmp/date.txt
7 * * * * * sleep 50; /bin/date >>/tmp/date.txt
```

实例：

```
1 #每隔10秒钟执行 index.php 文件
2 * * * * * /usr/bin/php /mnt/hgfs/webserver/test/index.php >> /dev/nul
   l 2>&1
3 * * * * * sleep 10; /usr/bin/php /mnt/hgfs/webserver/test/index.php >
   > /dev/null 2>&1
4 * * * * * sleep 20; /usr/bin/php /mnt/hgfs/webserver/test/index.php >
   > /dev/null 2>&1
5 * * * * * sleep 30; /usr/bin/php /mnt/hgfs/webserver/test/index.php >
   > /dev/null 2>&1
6 * * * * * sleep 40; /usr/bin/php /mnt/hgfs/webserver/test/index.php >
   > /dev/null 2>&1
7 * * * * * sleep 50;/usr/bin/php /mnt/hgfs/webserver/test/index.php >>
   /dev/null 2>&1
```

crontab -r 清除后找回

如果一旦执行`crontab -r`会清空当前用户下所有crontab配置命令。

复原方法

这里只讲述不存在备份，用日志的方法恢复crontab的方法。

系统：Linux（其他系统做法类似）

确定要恢复的日志存在

```
1 ll /var/log/cron*
```

【备注】这里的crontab日志是所有用户的日志

通过日志过滤需要的运行命令信息

我的Linux机器中crontab的日志信息主要如下：

```
1 Oct 9 03:28:01 Stor-Test CROND[7987]: (root) CMD (/home/scripts/check_alive.sh)
```

我们现在关键任务是将CMD后面的命令提取出来，并确定该条命令的执行周期

获取日志信息的命令

```
1 cat /var/log/cron | grep -i "whoami" | grep "CMD" | awk -F '(\'{print $3})' | awk -F ')' '{print $1}' | sort -u > cmd_tmp
```

解释：

grep -i "whoami"：是为了过滤指定用户的信息，whoami换成指定用户名称

grep "CMD"：是需要过滤命令行

awk -F '(\'{print \$3})'：是以'('为分隔符，提取第三个元素。这里结果

为“/home/scripts/check_alive.sh”

awk -F ')' '{print \$1}'：是以')'为分隔符，提取第一个元素。这里结果

为“/home/scripts/check_alive.sh”

这时，我们已经提取到自己所需要的命令了，但由于crontab定时触发，会有大量重复。后面需要进行去重

sort -u > cmd_tmp：去重后输出至cmd_tmp文件

后续根据提取出来的命令再去 /var/log/cron文件中确认一下时间间隔，按照指定的执行频率恢复

至此crontab恢复完毕

反思和总结

以后测试和发布内容之前一定要备份！！！为了防止类似事件再次发生，写一个自动化脚本是很有必要的。这里简单实现一个每天对crontab进行备份的脚本（备份最近7天的数据，每天定期删除7天前的数据）

```
1 #!/bin/bash
2 # 每天对crontab 进行备份，同时删除7天前的数据
3 DATE=$(date +%Y%m%d)
4 crontab -l > /home/work/bak/crontab_${DATE}.bak
5 find /home/work/bak/ -mtime +7 -name '*.bak' -exec rm -rf {} ;
```

复原原理

没有备份文件，crontab的复原方法

我们要恢复crontab，恢复的文件是哪个？

Linux下会根据用户的登录名，在/var/spool/cron下生成对应的文件。平时我们执行的crontab -e操作的文件时，编辑的文件就是/var/spool/cron/user_name

例如：root用户操作的是/var/spool/cron/root，其他用户同理

crontab的日志保存在哪里？

保存在/var/log/下

为什么/var/log/cron文件可以恢复crontab？

这里的/var/log/cron文件其实是crontab的日志，crontab执行的每个人物的时间、命令等详细信息都记录在这里，所以可以用这里的信息进行恢复。

其他方案

对于crontab其他比较安全的策略：

- 1、将/var/spool/cron/user_name文件添加到git定期缓存到本地。
- 2、使用第三方web调度管理系统，在web网页上操作，加入审核防止出错。
- 3、像php框架laravel一样将脚本执行频次放到一个入口代码中，相当于给所有要执行的crontab任务增加一个汇总入口页，crontab中只需要配置这个入口页脚本即可。代码用版本管理工具管理，不容易误删除。

logrotate 配置

logrotate 可以用于项目中生成的日志文件切割压缩归档，如可以在每天凌晨将前一天生成的日志压缩归档，并检查文件数，自动删除比较老的文件

以php和nginx 为例：

```
1 cd /etc/logrotate.d
2
3 vim php
4
5 /alidata/log/php/*.log {
6     daily
7     rotate 30
8     dateext
9     create
10    compress
11    sharedscripts
12    copytruncate
13
14 }
15
16 vim nginx
17
18 /alidata/log/nginx/access/*/*.log {
19     daily
20     rotate 30
21     dateext
22     create
23     compress
24     sharedscripts
25     copytruncate
26
27 }
```

配置完成后测试：

强制执行：

```
sudo /usr/sbin/logrotate -f /etc/logrotate.conf
```

1 若某文件logrotate失败，可以 sudo /usr/sbin/logrotate -f 单独的配置文件 -v
查看错误信息

强制以调试模式执行

```
sudo /usr/sbin/logrotate -d -f /etc/logrotate.conf
```

配置crontab 定时执行

```
2 0 * * * /etc/cron.daily/logrotate
```

重启crontab

```
sudo service cron restart
```

常用参数说明

```
1 常用参数
2 su wml wml
3
4 compress          通过gzip 压缩转储以后的日志
5
6 nocompress       不做gzip压缩处理
7
8 copytruncate     用于还在打开中的日志文件，把当前日志备份并截断；
                   是先拷贝再清空的方式，拷贝和清空之间有一个时间差，可能会丢失部分日志数据。
9
10 nocopytruncate 备份日志文件不过不截断
11
12 create mode owner group   轮转时指定创建新文件的属性，如create 0777 nob
                             ody nobody
13
14 nocreate         不建立新的日志文件
15
16 delaycompress   和compress 一起使用时，转储的日志文件到下一次转
                   储时才压缩
17
18 nodelaycompress 覆盖 delaycompress 选项，转储同时压缩。
19
20 missingok        如果日志丢失，不报错继续滚动下一个日志
21
22 errors address  转储时的错误信息发送到指定的Email 地址
23
24 ifempty          即使日志文件为空文件也做轮转，这个是logrotate的
                   缺省选项。
25
26 notifempty      当日志文件为空时，不进行轮转
27
28 mail address    把转储的日志文件发送到指定的E-mail 地址
29
30 nomail          转储时不发送日志文件
31
32 olddir directory 转储后的日志文件放入指定的目录，必须和当前日志文
                   件在同一个文件系统
33
34 noolddir         转储后的日志文件和当前日志文件放在同一个目录下
35
36 sharedscripts    运行postrotate脚本，作用是在所有日志都轮转后统
                   一执行一次脚本。如果没有配置这个，那么每个日志轮转后都会执行一次脚本
37
38 prerotate        在logrotate转储之前需要执行的指令，例如修改文件的属性等
                   动作；必须独立成行
39
40 postrotate       在logrotate转储之后需要执行的指令，例如重新启动（kill
                   -HUP）某个服务！必须独立成行
41
42 daily            指定转储周期为每天
43
44 weekly           指定转储周期为每周
```

```

45
46 monthly          指定转储周期为每月
47
48 rotate count    指定日志文件删除之前转储的次数, 0 指没有备份, 5 指保留5
49 个备份
50 dateext         使用当期日期作为命名格式
51
52 dateformat .%s   配合dateext使用, 紧跟在下一行出现, 定义文件切割后的文件
53 名, 必须配合dateext使用, 只支持 %Y %m %d %s 这四个参数
54 size(或minsize) log-size  当日志文件到达指定的大小时才转储, log-size能指定
55 bytes(缺省)及KB (sizek)或MB(sizeM).
56 当日志文件 >= log-size 的时候就转储。以下为合法格式: (其他格式的单位大小写没有试
57 过)
58 size = 5 或 size 5  (>= 5 个字节就转储)
59
60 size = 100k 或 size 100k
61
62 size = 100M 或 size 100M

```

Logrotate日志切割报错 文件不再同一个用户组下

分割日志时报错：

```

1 : error: skipping "/var/log/nginx/test.access.log" because parent
2 directory has insecure permissions (It's world writable or writable b
y
3 group which is not "root") Set "su" directive in config file to tell
4 logrotate which user/group should be used for rotation.

```

xx 文件所属用户

添加“su root xx”到/etc/logrotate.d/nginx文件中即可
如下；

```

1 /var/log/nginx/*.log {
2     su root public
3     daily
4     missingok
5     rotate 52
6     compress
7     delaycompress
8     notifempty
9     #ifempty
10    create 0640 www-data adm
11    sharedscripts
12    postrotate
13        [ ! -f /var/run/nginx.pid ] || kill -USR1 `cat /var/
run/nginx.pid`
14    endscript
15 }

```


站内全文搜索引擎 Sphinx/coreseek 安装使用教程

- 1 Sphinx是开源的搜索引擎，它支持英文的全文检索。所以如果单独搭建Sphinx，你就已经可以使用全文索引了。
- 2 但是往往我们要求的是中文索引，怎么做呢？国人提供了一个可供企业使用的，基于Sphinx的中文全文检索引擎。
- 3 也就是说Coreseek实际上的内核还是Sphinx。

- 1 sphinx可以通过设置为“一元切分模式”来支持搜索中文在实际使用中，搜索非中文的话，sphinx比coreseek要快；
- 2 搜索短中文字符串的话，开启了“一元切分模式”的sphinx比coreseek要快；
- 3 只有在搜索长中文串时，coreseek的分词优势才能显现，比sphinx要快

- 1 根据你的应用场景来选择用哪个，如果是索引英文、数字、字符较多的数据，就用源生sphinx；
- 2 如果是索引中文非常多非常长的数据，还是用coreseek吧。
- 3 Coreseek是基于Sphinx开发的一款软件，对Sphinx做了一些改动，在中文方面支持得比Sphinx好

安装注意

Coreseek发布了3.2.14版本和4.1版本等多个版本

3.2.14版本是2010年发布的，它是基于Sphinx0.9.9搜索引擎的。4.1版本是2011年发布的，它是基于Sphinx2.0.2以后的

本文章以安装 coreseek-4.1-beta 版本为例：

安装包下载地址：

```
1 http://dl.download.csdn.net/down11/20160530/de90462e2cf8350cbd4ad0f75  
8105c9d.gz?response-content-disposition=attachment%3Bfilename%3D%22co  
reseek-4.1-beta.tar.gz%22&OSSAccessKeyId=9q6nvzoJGowBj4q1&Expires=147  
4535438&Signature=8jae9H6dIfmd7zHRs72qcgi4Q1k%3D
```

3.2.14 稳定版（线上环境用此版本）

```
1 http://124.202.166.45/file3.data.weipan.cn/76885328/d22215f7ac79a8eff  
90a6eec0d7ccdc7f86048b6?ip=1490457292,118.144.20.162&ssig=z3aZVW70%2F  
Y&Expires=1490457892&KID=sae,l30zoo1wmz&fn=coreseek-3.2.14.tar.gz&ski  
prd=2&se_ip_debug=118.144.20.162&corp=2&from=1221134&wshc_tag=0&wsts_  
tag=58d68e1c&wsid_tag=75647862&wsiphost=ipdbm
```

第一步：安装升级autoconf

因为coreseek需要autoconf 2.64以上版本，因此需要升级autoconf，不然会报错

安装方法如下：

```
1 yum -y install glibc-common libtool autoconf automake mysql-devel exp  
at-devel
```

第二步：安装mmseg(coreseek所使用的词典)

```
1 tar -zxvf coreseek-4.1-beta.tar.gz
2 cd coreseek-4.1-beta
3 cd mmseg-4.1-beta/
4 ./bootstrap      #输出的warning信息可以忽略，如果出现error则需要解决
5 ./configure --prefix=/usr/local/mmseg3
6 make
7 make install
8
9 # 检测是否安装完成
10 mmseg
11 提示: -bash: mmseg: command not found
12
13 # 问题解决方法
14 ln -s /usr/local/mmseg3/bin/mmseg /bin/mmseg
15
16 # 再测试下
17 mmseg
18
19 # 显示以下内容表示安装成功
20 Coreseek COS(tm) MM Segment 1.0
21 Copyright By Coreseek.com All Right Reserved.
22 Usage: mmseg <option> <file>
23 -u <unidict>           Unigram Dictionary
24 -r                   Combine with -u, used a plain text build Unigram Dictionary, default Off
25 -b <Synonyms>          Synonyms Dictionary
26 -t <thesaurus>         Thesaurus Dictionary
27 -h                   print this help and exit
```

第三步：安装csft-4.1-beta

```
1 cd csft-4.1-beta/
2 sh buildconf.sh      #输出的warning信息可以忽略，如果出现error则需要解决
3 ./configure
4 --prefix=/usr/local/sphinx
5 --without-unixodbc
6 --with-mmseg
7 --with-mmseg-includes=/usr/local/mmseg3/include/mmseg/
8 --with-mmseg-libs=/usr/local/mmseg3/lib/
9 --with-mysql=/usr/local/mysql/
10
11 make
12 make install
13
14
15 #./configure --prefix=/usr/local/sphinx/ --with-mysql=/usr/local/mysql/ --enable-id64
16 make
17 make install
18 # 注意1：采用这种方式安装不支持中文分词。
19 # 注意2：--enable-id64 系统是64位，开启后Sphinx存储模块可以支持4-5G，不开启支持2G左右，不开启检索效率会高些，一般情况下建议不开启
20
21 # 错误：
```

```
22 collect2: ld 返回 1
23 make[2]: *** [indexer] 错误 1
24
25 # 修改configure文件中:
26 LIBS="$LIBS -L/usr/local/lib"
27 # 改为:
28 LIBS="$LIBS -liconv -L/usr/local/lib"
```

测试mmseg分词和coreseek搜索

```
1 cd testpack
2 cat var/test/test.xml #此时应该正确显示中文
3 /usr/local/mmseg3/bin/mmseg -d /usr/local/mmseg3/etc var/test/test.xml
4 /usr/local/sphinx/bin/indexer -c etc/csft.conf --all
5 /usr/local/sphinx/bin/search -c etc/csft.conf 网络搜索
6 #此时正确的应该返回
7 words:
8 1. '网络': 1 documents, 1 hits
9 2. '搜索': 2 documents, 5 hits
```

第四步：配置mmseg中文分词

目前这里不用操作，安装时已经帮我配置好了

```
1 cd /usr/local/mmseg3/
2
3 # 生成unigram.txt.uni
4 ./bin/mmseg -u /usr/local/mmseg/etc/unigram.txt
5 vim etc/mmseg.ini
6
7 [mmseg]
8 merge_number_and_ascii=0;      ;合并英文和数字 abc123/x
9 number_and_ascii_joint=;      ;定义可以连接英文和数字的字符
10 compress_space=1;           ;暂不支持
11 seperate_number_ascii=0;     ;将字母和数字打散
12
13 # 复制到sphinx/dict目录
14 cp etc/mmseg.ini /usr/local/sphinx/dict/mmseg.ini
15
16 # 复制到sphinx/dict目录
17 cp etc/unigram.txt /usr/local/sphinx/dict/uni.lib
```

第五步：配置sphinx

```
1 # 创建配置文件
2 mkdir -p /usr/local/sphinx/etc/conf.d/
3 vim /usr/local/sphinx/etc/conf.d/sphinx.conf
```

sphinx.conf 配置文件内容

```
1 #配置数据源，可以有多个
2 source src_blog
3 {
```

```

4   #数据库类型
5   type = mysql
6   #是否去掉html标签
7   strip_html = 0
8   sql_host = 127.0.0.1
9   sql_user = root
10  sql_pass = woshishui
11  sql_db = yphp_tutiantian
12  sql_port = 3006
13  #在执行sql_query前执行的sql命令，可以有多条
14  sql_query_pre= SET NAMES utf8
15  #全文检索要显示的内容，在这里尽可能不使用where或group by，将where与groupby
    的内容交给sphinx,
16  #由sphinx进行条件过滤与groupby效率会更高
17  #注意：select 出来的字段必须至少包括一个唯一主键(ARTICLESID)以及要全文检索
    的字段,
18  #你计划原本在where中要用到的字段也要select出来
19  #这里不用使用orderby
20  sql_query = SELECT * FROM tu_pic
21  #开头的表示一些属性字段，你原计划要用在where,orderby,groupby中的字段要在这
    里定义
22  #根据我们原先的SQL:
23  #select * from eht_articles where title like ? and catalogid=? And
    edituserid=? And addtime between ? and ? order by
    hits desc
24  #我们需要对catalogid,edituserid,addtime,hits进行属性定义(这四个字段也要
    在select的字段列表中),
25  #定义时不同的字段类型有不同的属性名称，具体可以见sphinx.conf.in中的说明
26  sql_attr_uint= pr
27  sql_attr_uint = big_id
28  sql_attr_uint = small_id
29  sql_attr_uint= uid
30  sql_attr_uint= is_del
31  sql_ranged_throttle = 0
32 }
33
34
35 # 在这里要注意，rt_field是检索字段，rt_attr_uint是返回字段
36 #定义索引名称
37 index sphinx_blog
38 {
39   #数据源名
40   source = src_blog
41   #索引记录存放目录
42   path = /usr/local/sphinx/var/data/sphinx_blog
43   docinfo = extern
44   mlock = 0
45   stopwords =
46   min_prefix_len = 0
47   min_infix_len = 0
48   morphology = none
49   min_word_len = 2
50   #字符集
51   charset_type = zh_cn.utf-8
52   #指明分词法读取词典文件的位置，当启用分词法时，为必填项。在使用LibMMSeg作为分
      词 库时，
53   #需要确保词典文件uni.lib在指定的目录下
54   charset_dictpath = /usr/local/mmseg3/etc/
55   charset_table = U+FF10..U+FF19->0..9, 0..9, U+FF41..U+FF5A->a..
      z, U+FF21..U+FF3A->a..z,A..Z->a..z, a..z, U+0149, U+017F, U+0138, U+

```

```

00DF, U+00FF, U+00C0..U+00D6->U+00E0..U+00F6, U+00E0..U+00F6, U+00D
8..U+00DE->U+00F8..U+00FE, U+00F8..U+00FE, U+0100->U+0101, U+0101, U+
0102->U+0103, U+0103, U+0104->U+0105, U+0105, U+0106->U+0107, U+010
7, U+0108->U+0109, U+0109, U+010A->U+010B, U+010B, U+010C->U+010D, U+
010D, U+010E->U+010F, U+010F, U+0110->U+0111, U+0111, U+0112->U+0113,
U+0113, U+0114->U+0115, U+0115, U+0116->U+0117, U+0117, U+0118->U+011
9, U+0119, U+011A->U+011B, U+011B, U+011C->U+011D, U+011D, U+011E->U+
011F, U+011F, U+0130->U+0131, U+0131, U+0132->U+0133, U+0133, U+0134
->U+0135, U+0135, U+0136->U+0137, U+0137, U+0139->U+013A, U+013A, U+0
13B->U+013C, U+013C, U+013D->U+013E, U+013E, U+013F->U+0140, U+0140,
U+0141->U+0142, U+0142, U+0143->U+0144, U+0144, U+0145->U+0146, U+01
46, U+0147->U+0148, U+0148, U+014A->U+014B, U+014B, U+014C->U+014D, U
+014D, U+014E->U+014F, U+014F, U+0150->U+0151, U+0151, U+0152->U+015
3, U+0153, U+0154->U+0155, U+0155, U+0156->U+0157, U+0157, U+0158->U+
0159, U+0159, U+015A->U+015B, U+015B, U+015C->U+015D, U+015D, U+015E-
>U+015F, U+015F, U+0160->U+0161, U+0161, U+0162->U+0163, U+0163, U+01
64->U+0165, U+0165, U+0166->U+0167, U+0167, U+0168->U+0169, U+0169, U
+016A->U+016B, U+016B, U+016C->U+016D, U+016D, U+016E->U+016F, U+01
6F, U+0170->U+0171, U+0171, U+0172->U+0173, U+0173, U+0174->U+0175, U
+0175, U+0176->U+0177, U+0177, U+0178->U+00FF, U+00FF, U+0179->U+017
A, U+017A, U+017B->U+017C, U+017C, U+017D->U+017E, U+017E, U+0410..U+
042F->U+0430..U+044F, U+0430..U+044F, U+05D0..U+05EA, U+0531..U+0556-
>U+0561..U+0586, U+0561..U+0587, U+0621..U+063A, U+01B9, U+01BF, U+06
40..U+064A, U+0660..U+0669, U+066E, U+066F, U+0671..U+06D3, U+06F0..
U+06FF, U+0904..U+0939, U+0958..U+095F, U+0960..U+0963, U+0966..U+096
F, U+097B..U+097F, U+0985..U+09B9, U+09CE, U+09DC..U+09E3, U+09E6..U+
09EF, U+0A05..U+0A39, U+0A59..U+0A5E, U+0A66..U+0A6F, U+0A85..U+0AB9,
U+0AE0..U+0AE3, U+0AE6..U+0AEF, U+0B05..U+0B39, U+0B5C..U+0B61, U+0B6
6..U+0B6F, U+0B71, U+0B85..U+0BB9, U+0BE6..U+0BF2, U+0C05..U+0C39, U+
0C66..U+0C6F, U+0C85..U+0CB9, U+0CDE..U+0CE3, U+0CE6..U+0CEF, U+0D0
5..U+0D39, U+0D60, U+0D61, U+0D66..U+0D6F, U+0D85..U+0DC6, U+1900..U+
1938, U+1946..U+194F, U+A800..U+A805, U+A807..U+A822, U+0386->U+03B1,
U+03AC->U+03B1, U+0388->U+03B5, U+03AD->U+03B5, U+0389->U+03B7, U+03A
E->U+03B7, U+038A->U+03B9, U+0390->U+03B9, U+03AA->U+03B9, U+03AF->U+
03B9, U+03CA->U+03B9, U+038C->U+03BF, U+03CC->U+03BF, U+038E->U+03C
5, U+03AB->U+03C5, U+03B0->U+03C5, U+03CB->U+03C5, U+03CD->U+03C5, U+
038F->U+03C9, U+03CE->U+03C9, U+03C2->U+03C3, U+0391..U+03A1->U+03B
1..U+03C1, U+03A3..U+03A9->U+03C3..U+03C9, U+03B1..U+03C1, U+03C3..U+
03C9, U+0E01..U+0E2E, U+0E30..U+0E3A, U+0E40..U+0E45, U+0E47, U+0E5
0..U+0E59, U+A000..U+A48F, U+4E00..U+9FBF, U+3400..U+4DBF, U+2000..U
+2A6DF, U+F900..U+FAFF, U+2F800..U+2FA1F, U+2E80..U+2EFF, U+2F00..U+2
FDF, U+3100..U+312F, U+31A0..U+31BF, U+3040..U+309F, U+30A0..U+30FF,
U+31F0..U+31FF, U+AC00..U+D7AF, U+1100..U+11FF, U+3130..U+318F, U+A0
00..U+A48F, U+A490..U+A4CF
56     html_strip = 0
57 }
58
59 #全局index定义
60 indexer
61 {
62     mem_limit = 256M
63 }
64
65 #sphinx守护进程配置
66 searchd
67 {
68     port = 9312
69     read_timeout = 5
70     max_children = 10
71     compat_sphinxql_magics = 0

```

```
72     pid_file = /usr/local/sphinx/var/log/searchd.pid
73     #全文检索日志
74     log = /usr/local/sphinx/var/log/searchd.log
75     #查询日志
76     query_log = /usr/local/sphinx/var/log/query_log.log
77     #最大匹配数，也就是查找的数据再多也只返回这里设置的1000条
78     max_matches = 1000
79     seamless_rotate = 1
80 }
```

建立sphinx 索引（若日常更新sphinx数据，只需执行以下1、2、3步即可，可写入sh文件执行）

```
1 # 结束所有索引
2 #pkill -9 search
3
4 1、#停止正在运行的searchd
5 /usr/local/sphinx/bin/searchd --config /usr/local/sphinx/etc/conf.d/
    sphinx.conf --stop
6
7 2、# 创建索引
8 # /usr/local/sphinx/bin/indexer --c --config /usr/local/sphinx/etc/c
    onf.d/sphinx.conf --all
9
10 #如果只想对某个数据源进行索引，则可以这样：
11 #/usr/local/sphinx/bin/indexer --c --config /usr/local/sphinx/etc/co
    nf.d/sphinx.conf 索引名称
12
13 3、# 启动索引
14 # /usr/local/sphinx/bin/searchd --config /usr/local/sphinx/etc/conf.
    d/sphinx.conf
15
16
17 # 错误解决
18 # 解决：复制MySQL客户端文件到所有用户可以放的目录去
19 cp /usr/local/mysql/lib/libmysqlclient.so.18 /usr/lib/
20
21 # 64位系统需要在创建一个软连接
22 ln -s /usr/local/mysql/lib/libmysqlclient.so.18 /usr/lib64
```

1 服务器所用MySQL在当时编译时并没有编译Sphinx扩展，而重新编译mysql并加入Sphinx暂时又无法实现（项目用到了多台服务器，在不影响现有业务的情况下不可能去重新编译MySQL的），所以采用的是程序通过API来外部调用Sphinx。Sphinx自带的API有PHP, Python, Ruby, Java等众多版本，所以基本也够用了，本人使用的编程语言是php所以下文的条用示例采用的是PHP版的API。

以下步骤不需要执行，原因如上：

第六步：安装sphinxclient

php模块依赖于libsphinxclient包

```
1 # 进入sphinx 源码包的api目录
2 cd coreseek-4.1/csft-4.1/api/libsphinxclient
3 ./configure --prefix=/usr/local/libsphinxclient
4 #若报一下错误
```

```
5 config.status: error: cannot find input file: Makefile.in #报错configure失败
6 #处理configure报错，运行下列指令再次编译：
7 aclocal
8 libtoolize --force
9 automake --add-missing
10 autoconf
11 autoheader
12 make clean
13
14 //重新configure编译
15 ./configure --prefix=/usr/local/libsphinxclient
16 make
17 make install
```

第七步：安装php的Sphinx扩展

```
1 # PHP7以前的版本使用下面扩展
2 wget http://pecl.php.net/get/sphinx-1.3.0.tgz
3
4 # PHP7 使用下面的扩展
5 wget http://git.php.net/?p=pecl/search_engine/sphinx.git;a=snapshot;
h=9a3d08c67af0cad216aa0d38d39be71362667738;sf=tgz
6
7 tar -zxvf sphinx-1.3.0.tgz
8 cd sphinx-1.3.0
9 /usr/local/php/bin/phpize ./configure --with-php-config=/usr/local/php/bin/php-config --with-sphinx=/usr/local/libsphinxclient/
10 make && make install
11 # 安装好后，在安装目录下etc目录下，有份测试数据和配置的样本
```

第八步：配置PHP支持Sphinx

编辑php.ini文件

```
1 vim /usr/local/php/etc/php.ini
2
3 # 修改以下内容
4 extension_dir = "/usr/local/php/lib/php/extensions/no-debug-non-zts-
20100525/"
5
6 # 增加扩展到php
7 extension = sphinx.so
8
9 # 重新启动php-fpm
10 /etc/init.d/php-fpm restart
```

查看Sphinx是否安装成功

新建PHP文件

```
1 vim index.php
2
3 # 写入一下内容：
4 <?php
5 echo phpinfo();
```

浏览器输入地址访问
看到sphinx表示扩展安装成功



第九步：php使用sphinx

```
1 <?php
2 // -----
3 // File name : test_coreseek.php
4 // Description : coreseek中文全文检索系统测试程序
5 // Requirement : PHP5 (http://www.php.net)
6 //
7 // Copyright(C), HonestQiao, 2011, All Rights Reserved.
8 //
9 // Author: HonestQiao (honestqiao@gmail.com)
10 //
11 // 最新使用文档, 请查看: http://www.coreseek.cn/products/products-instal
12 //
13 // -----
14 #若SphinxClient是php默认类, 则不需要引入sphinxapi.php
15 #sphinxapi.php文件在/coreseek-3.2.14/testpack/api/ 中
16 require ( "sphinxapi.php" );
17 $cl = new SphinxClient ();
18 $cl->SetServer ( '127.0.0.1', 9312 );
19 $cl->SetConnectTimeout ( 3 );
20 $cl->SetArrayResult ( true );
21 $cl->SetMatchMode ( SPH_MATCH_ANY );
22
23
24 $cl->setLimits(0,300);
25 $keyword = $_GET['keyword'];
26 // // 执行查询, 第一个参数查询的关键字, 第二个查询的索引名称, 多个索引名称以, 分开,
27 // // 也可以用*表示所有索引。
28 $res = $cl->Query ( $keyword, "sphinx_blog" );
29 print_r($res);
```

查询结果：

```
1 Array
2 (
3     [error] =>
4     [warning] =>
5     [status] => 0
6     [fields] => Array
7         (
8             [0] => search_word
9             [1] => actors
10            [2] => directors
11            [3] => wktag
12            [4] => pinyin
13            [5] => pinyin_first
14        )
15
```

```

16     [attrs] => Array
17         (
18             [xid] => 1
19         )
20
21     [matches] => Array
22         (
23             [0] => Array
24                 (
25                     #matches中的id就是指配置文件中sql_query SELECT语句中的
第一个字段
26                         [id] => 33514269
27                         [weight] => 1
28                         [attrs] => Array
29                             (
30                                 [xid] => 33514269
31                             )
32
33                     )
34
35                 )
36
37             [total] => 1
38             [total_found] => 1
39             [time] => 0.002
40             [words] => Array
41                 (
42                     [免先生] => Array
43                         (
44                             [docs] => 1
45                             [hits] => 1
46                         )
47
48                 )
49
50 )

```

Matches中就是查询的结果了，但是仿佛不是我们想要的数据，比如title, content字段的内容就没有查询出来，根据官方的说明是Sphinx并没有连接到MySQL去取数据，只是根据它自己的索引内容进行计算，因此如果想用Sphinx提供的API去取得我们想要的数据，还必须以查询的结果为依据，再次查询MySQL从而得到我们想要的数据。

sphinx/coreseek配置说明

sphinx 中定义index索引参数说明:

在index中有charset_type、charset_table参数，说明如下：

```
1 charset_type = utf-8 #####数据编码，只识别英文字符，搜索词只能是英文，对搜索结  
果没有影响  
2 #charset_type = zh_cn.utf-8 #####数据编码，识别中英文字符，搜索词可以是中文也  
可以是英文  
3  
4  
5 ##### charset_table 字符表，注意：如使用这种方式，则sphinx会对中文进行单字切  
分，  
6 ##### 即进行字索引，若要使用中文分词，必须使用其他分词插件如 coreseek, sfc  
7  
8 charset_table = U+FF10..U+FF19->0..9, 0..9, U+FF41..U+FF5A->a..z, U+  
FF21..U+FF3A->a..z,\n  
9 A..Z->a..z, a..z, U+0149, U+017F, U+0138, U+00DF, U+00FF, U+00C0..U+  
00D6->U+00E0..U+00F6,\n  
10 U+00E0..U+00F6, U+00D8..U+00DE->U+00F8..U+00FE, U+00F8..U+00FE, U+01  
00->U+0101, U+0101,\n  
11 U+0102->U+0103, U+0103, U+0104->U+0105, U+0105, U+0106->U+0107, U+01  
07, U+0108->U+0109,\n  
12 U+0109, U+010A->U+010B, U+010B, U+010C->U+010D, U+010D, U+010E->U+01  
0F, U+010F,\n  
13 U+0110->U+0111, U+0111, U+0112->U+0113, U+0113, U+0114->U+0115, U+01  
15, \n  
14 U+0116->U+0117, U+0117, U+0118->U+0119, U+0119, U+011A->U+011B, U+011  
B, U+011C->U+011D,\n  
15 U+011D, U+011E->U+011F, U+011F, U+0130->U+0131, U+0131, U+0132->U+013  
3, U+0133, \n  
16 U+0134->U+0135, U+0135, U+0136->U+0137, U+0137, U+0139->U+013A, U+013  
A, U+013B->U+013C, \n  
17 U+013C, U+013D->U+013E, U+013E, U+013F->U+0140, U+0140, U+0141->U+014  
2, U+0142, \n  
18 U+0143->U+0144, U+0144, U+0145->U+0146, U+0146, U+0147->U+0148, U+014  
8, U+014A->U+014B, \n  
19 U+014B, U+014C->U+014D, U+014D, U+014E->U+014F, U+014F, U+0150->U+015  
1, U+0151, \n  
20 U+0152->U+0153, U+0153, U+0154->U+0155, U+0155, U+0156->U+0157, U+015  
7, U+0158->U+0159,\n  
21 U+0159, U+015A->U+015B, U+015B, U+015C->U+015D, U+015D, U+015E->U+015  
F, U+015F, \n  
22 U+0160->U+0161, U+0161, U+0162->U+0163, U+0163, U+0164->U+0165, U+016  
5, U+0166->U+0167, \n  
23 U+0167, U+0168->U+0169, U+0169, U+016A->U+016B, U+016B, U+016C->U+016  
D, U+016D, \n  
24 U+016E->U+016F, U+016F, U+0170->U+0171, U+0171, U+0172->U+0173, U+017  
3, U+0174->U+0175,\n  
25 U+0175, U+0176->U+0177, U+0177, U+0178->U+00FF, U+00FF, U+0179->U+017  
A, U+017A, \n  
26 U+017B->U+017C, U+017C, U+017D->U+017E, U+017E, U+0410..U+042F->U+043  
0..U+044F, \n  
27 U+0430..U+044F, U+05D0..U+05EA, U+0531..U+0556->U+0561..U+0586, U+056  
1..U+0587, \n  
28 U+0621..U+063A, U+01B9, U+01BF, U+0640..U+064A, U+0660..U+0669, U+066
```

```

E, U+066F, \
29 U+0671..U+06D3, U+06F0..U+06FF, U+0904..U+0939, U+0958..U+095F, U+096
0..U+0963, \
30 U+0966..U+096F, U+097B..U+097F, U+0985..U+09B9, U+09CE, U+09DC..U+09E
3, U+09E6..U+09EF, \
31 U+0A05..U+0A39, U+0A59..U+0A5E, U+0A66..U+0A6F, U+0A85..U+0AB9, U+0AE
0..U+0AE3, \
32 U+0AE6..U+0AEF, U+0B05..U+0B39, U+0B5C..U+0B61, U+0B66..U+0B6F, U+0B7
1, U+0B85..U+0BB9, \
33 U+0BE6..U+0BF2, U+0C05..U+0C39, U+0C66..U+0C6F, U+0C85..U+0CB9, U+0CD
E..U+0CE3, \
34 U+0CE6..U+0CEF, U+0D05..U+0D39, U+0D60, U+0D61, U+0D66..U+0D6F, U+0D8
5..U+0DC6, \
35 U+1900..U+1938, U+1946..U+194F, U+A800..U+A805, U+A807..U+A822, U+038
6->U+03B1, \
36 U+03AC->U+03B1, U+0388->U+03B5, U+03AD->U+03B5, U+0389->U+03B7, U+03A
E->U+03B7, \
37 U+038A->U+03B9, U+0390->U+03B9, U+03AA->U+03B9, U+03AF->U+03B9, U+03C
A->U+03B9, \
38 U+038C->U+03BF, U+03CC->U+03BF, U+038E->U+03C5, U+03AB->U+03C5, U+03B
0->U+03C5, \
39 U+03CB->U+03C5, U+03CD->U+03C5, U+038F->U+03C9, U+03CE->U+03C9, U+03C
2->U+03C3, \
40 U+0391..U+03A1->U+03B1..U+03C1, U+03A3..U+03A9->U+03C3..U+03C9, U+03B
1..U+03C1, \
41 U+03C3..U+03C9, U+0E01..U+0E2E, U+0E30..U+0E3A, U+0E40..U+0E45, U+0E4
7, U+0E50..U+0E59, \
42 U+A000..U+A48F, U+4E00..U+9FBF, U+3400..U+4DBF, U+20000..U+2A6DF, U+F
900..U+FAFF, \
43 U+2F800..U+2FA1F, U+2E80..U+2EFF, U+2F00..U+2FDF, U+3100..U+312F, U+3
1A0..U+31BF, \
44 U+3040..U+309F, U+30A0..U+30FF, U+31F0..U+31FF, U+AC00..U+D7AF, U+110
0..U+11FF, \
45 U+3130..U+318F, U+A000..U+A48F, U+A490..U+A4CF
46
47 #在配置时不建议用 charset_table 参数, 若配置该参数, 搜索金牌, 他会将含有关键词
金、牌、金牌 的结果都搜出来, 但我只需要搜索金牌的结果, 而不需要这些结果。若去掉该
参数, 则只会搜索含有关键词'金牌'的结果, 只含有'金'或'牌'的结果不会搜出来

```

sphinx 不关闭进程更新索引

添加 '--rotate' 参数

```
1 #建立主索引
2 /coreseek/bin/indexer vall --rotate -c /coreseek/etc/vall.conf
3 /coreseek/bin/indexer cibnvip_index --rotate -c /coreseek/etc/vall.conf
4 /coreseek/bin/indexer valltips --rotate -c /coreseek/etc/vall.conf
```

```
1 #创建增量索引
2 /coreseek/bin/indexer delta --rotate -c /coreseek/etc/vall.conf
3 #合并索引
4 #将delta合并到main中
5 /coreseek/bin/indexer --merge vall delta --rotate -c /coreseek/etc/vall.conf
6 #增量索引
7 /coreseek/bin/indexer more --rotate -c /coreseek/etc/vall.conf
8 #将more合并到cibnvip_index中
9 /coreseek/bin/indexer --merge cibnvip_index more --rotate -c /coreseek/etc/vall.conf
```

1 --rotate参数可以在不停止searchd的情况下索引，不然的话会有类似如下的提示：

```
1 FATAL: failed to lock /usr/local/coreseek/var/data/hx_9enjoy_delta.spl:
2 Resource temporarily unavailable, will not index. Try --rotate option.
3 ERROR: index 'delta' is already locked; lock: failed to
4 lock /usr/local/coreseek/var/data/cncn_article_delta.spl:
5 Resource temporarily unavailable
```

sphinx增量索引配置

```
1 #源定义
2 source cibn
3 {
4     type          = mysql
5     sql_host      = localhost
6     sql_user      = root
7     sql_pass      = 111111
8     sql_db        = test
9     sql_port      = 3306
10
11    sql_sock    = /tmp/mysql.sock #修改路径
12
13    sql_query_pre   = SET NAMES utf8
14    sql_query_pre   = REPLACE INTO sph_counter SELECT 3, MAX(i
d) FROM cibn
15    sql_query      = SELECT id,id as xid, search_word FROM ci
bn
16    sql_attr_uint   = xid           #从SQL读取到的值必须为整数
17 }
18
19 #增量索引 源定义
20 source morecibn:cibn{
21     sql_query_pre   = SET NAMES utf8
22     sql_query_pre   =
23     sql_query      = SELECT id,id as xid, search_word FROM ci
bn WHERE id>(SELECT max_doc_id FROM sph_counter WHERE counter_id=3)
24     sql_query_post_index = REPLACE INTO sph_counter SELECT 3, MAX(i
d) FROM cibn
25 }
26
27 #index定义
28 index cibn_index
29 {
30     source          = cibn          #对应的source名称
31     path            = /cibn #请修改为实际使用的绝对路径, 如: /usr/local/cor
eseek/var/...
32     docinfo         = extern
33     mlock           = 0
34     morphology      = none
35     min_word_len    = 1
36     html_strip      = 0
37
38     #中文分词配置, 详情请查看: http://www.coreseek.cn/products-install/co
reseek_mmseg/
39     charset_dictpath = /mmseg/etc/ #BSD、Linux环境下设置, /符号结尾
40     #charset_dictpath = etc/           #Windows环境
下设置, /符号结尾, 最好给出绝对路径, 例如: C:/usr/local/coreseek/etc/...
41     charset_type     = zh_cn.utf-8
42     #min_prefix_len  = 2
43     min_infix_len    = 2
44     ngram_len        = 1
45 }
46
47 #增量索引 index定义
48 index morecibn:cibn_index{
```

```
49     source          = more
50     path            = /morecibn
51     docinfo         = extern
52     mlock           = 0
53     morphology      = none
54     min_word_len    = 1
55     html_strip      = 0
56
57     #中文分词配置, 详情请查看: http://www.coreseek.cn/products-install/co
      reseek_mmseg/
58     charset_dictpath = /mmseg/etc/ #BSD、Linux环境下设置, /符号结尾
59     charset_type     = zh_cn.utf-8
60     #min_prefix_len  = 2
61     min_infix_len    = 2
62     ngram_len        = 1
63 }
```

脚本：<http://blog.csdn.net/u013372487/article/details/78022951>

ubuntu 安装sphinx报错解决方法

错误一： coreseek无法生成configure文件

错误信息如下：

```
1 libstemmer_c/mkinc.mak:10: warning: source file 'runtime/utilities.c' is in a subdirectory,
2 libstemmer_c/mkinc.mak:10: but option 'subdir-objects' is disabled
3 libstemmer_c/Makefile.am:3: 'libstemmer_c/mkinc.mak' included from here
4 libstemmer_c/mkinc.mak:10: warning: source file 'libstemmer/libstemmer.c' is in a subdirectory,
5 libstemmer_c/mkinc.mak:10: but option 'subdir-objects' is disabled
6 libstemmer_c/Makefile.am:3: 'libstemmer_c/mkinc.mak' included from here
7 /usr/local/share/automake-1.14/am/library.am: warning: 'libstemmer.a': linking libraries using a non-POSIX
8 /usr/local/share/automake-1.14/am/library.am: archiver requires 'AM_PROG_AR' in 'configure.ac'
9 libstemmer_c/Makefile.am:2: while processing library 'libstemmer.a'
10 /usr/local/share/automake-1.14/am/library.am: warning: 'libsphinx.a': linking libraries using a non-POSIX
11 /usr/local/share/automake-1.14/am/library.am: archiver requires 'AM_PROG_AR' in 'configure.ac'
12 src/Makefile.am:14: while processing library 'libsphinx.a'
```

看起来非常多其实就是两个：

```
1 1、but option 'subdir-objects' is disabled
2 2、archiver requires 'AM_PROG_AR' in 'configure.ac'
```

解决方案：

1、在 csft-4.1/buildconf.sh 文件中，查找

```
1 && aclocal \
```

后加上

```
1 && automake --add-missing \
```

2、在 csft-4.1/configure.ac 文件中，查找：

```
1 AM_INIT_AUTOMAKE([-Wall -Werror foreign])
```

改为：

```
1 AM_INIT_AUTOMAKE([-Wall foreign])
```

查找：

```
1 AC_PROG_RANLIB
```

后面加上

```
1 AM_PROG_AR
```

3、最后，在 csft-4.1/src/sphinxexpr.cpp 文件中，替换所有：

```
1 T val = ExprEval ( this->m_pArg, tMatch );
```

成为：

```
1 T val = this->ExprEval ( this->m_pArg, tMatch );
```

错误二： make && make install出错

错误现象：执行 buildconf.sh 报错，无法生成configure文件

提示的主要错误为：

```
1 /setup/coreseek-3.2.14/csft-3.2.14/src/sphinx.cpp:20719: undefined reference to `libiconv_open'  
2 /setup/coreseek-3.2.14/csft-3.2.14/src/sphinx.cpp:20737: undefined reference to `libiconv'  
3 /setup/coreseek-3.2.14/csft-3.2.14/src/sphinx.cpp:20743: undefined reference to `libiconv_close'  
4 collect2: ld 返回 1  
5 make[2]: *** [indexer] 错误 1  
6 make[2]: Leaving directory `/setup/coreseek-3.2.14/csft-3.2.14/src'  
7 make[1]: *** [all] 错误 2  
8 make[1]: Leaving directory `/setup/coreseek-3.2.14/csft-3.2.14/src'  
9 make: *** [all-recurcive] 错误 1
```

解决方法

编辑：

```
1 ./src/Makefile文件
```

将

```
1 LIBS = -ldl -lm -lz -lexpat -L/usr/local/lib -lrt -lpthread
```

改成

```
1 LIBS = -ldl -lm -lz -lexpat -liconv -L/usr/local/lib -lrt -lpthread
```


mac 上VMware安装VMware Tools选项显示灰色 解决办法

解决办法如下：

1.关闭虚拟机；

2.在虚拟机设置分别设置CD/DVD、CD/DVD2和软盘为自动检测三个步骤；



3.再重启虚拟机，灰色字即点亮。

若设置后依然不行，将可移除设备中的软盘删除，重启试一下。

brew国内加速

Mac搭配homebrew简直舒爽啊，然而homebrew托管在github，对国内用户来说不仅频频被墙，而且速度也不理想。今天笔者就告诉大家国内用户顺畅访问homebrew的方法。

中科大的镜像

中科大镜像比较稳定，而且速度不错。官方网站：<http://mirrors.ustc.edu.cn/>。搜索brew，然后点击Help即可查看用法：

替换Homebrew默认源

参考地址：<https://lug.ustc.edu.cn/wiki/mirrors/help/brew.git>

```
1 # 替换brew.git:  
2 cd "$(brew --repo)"  
3 git remote set-url origin https://mirrors.ustc.edu.cn/brew.git  
4  
5 # 替换homebrew-core.git:  
6 cd "$(brew --repo)/Library/Taps/homebrew/homebrew-core"  
7 git remote set-url origin https://mirrors.ustc.edu.cn/homebrew-core.g  
it  
8  
9 brew update
```

替换Homebrew Bottles源

参考地址：<https://lug.ustc.edu.cn/wiki/mirrors/help/homebrew-bottles>

对于bash用户

```
1 echo 'export HOMEBREW_BOTTLE_DOMAIN=https://mirrors.ustc.edu.cn/homeb  
rew-bottles' >> ~/.bash_profile  
2 source ~/.bash_profile
```

对于zsh用户

```
1 echo 'export HOMEBREW_BOTTLE_DOMAIN=https://mirrors.ustc.edu.cn/homeb  
rew-bottles' >> ~/.zshrc  
2 source ~/.zshrc
```

清华大学镜像

参考：<https://mirror.tuna.tsinghua.edu.cn/help/homebrew/>

切换回官方源：

重置brew.git：

```
1 cd "$(brew --repo)"  
2 git remote set-url origin https://github.com/Homebrew/brew.git
```

重置homebrew-core.git：

```
1 cd "$(brew --repo)/Library/Taps/homebrew/homebrew-core"  
2 git remote set-url origin https://github.com/Homebrew/homebrew-core.g  
it
```

重置Homebrew Bottles源

注释掉bash配置文件里的有关Homebrew Bottles即可恢复官方源。 重启bash或让bash重读配置文件。

mac 下编译安装nginx

1、下载nginx源码

官网地址：<http://nginx.org/en/download.html>

选择 Stable version：最新稳定版本

下载后可以解压移动到/usr/local/bin目录下：

```
1 mv nginx-1.12.2.tar.gz /usr/local/bin
```

2、安装依赖包

nginx的安装依赖下面软件

OpenSSL

在[官网](https://www.openssl.org/source/)下到最新稳定版。<https://www.openssl.org/source/>

PCRE

下载地址：<https://ftp.pcre.org/pub/pcre/>

注意有两个大版本：8.x和10.x，nginx依赖的是8.x，所以选择 pcre-8.*的最高版本。

zlib

下载地址：<http://zlib.net/>

把以上库的解压目录也移动到/usr/local/bin（和 nginx 同目录）：

```
1 mv openssl-1.1.0g pcre-8.41 zlib-1.2.11 /usr/local/bin
```

3、编译安装

进入之前解压的 nginx 目录：

```
1 cd /usr/local/bin/nginx-1.18.0/
```

执行配置命令，几个依赖包的路径对就可以，官方文档提示要写到一行：

参考官方文档：<http://nginx.org/en/docs/configure.html>

```
1 ./configure --sbin-path=/usr/local/nginx/nginx \
2           --conf-path=/usr/local/nginx/nginx.conf \
3           --pid-path=/usr/local/nginx/nginx.pid \
4           --with-http_ssl_module \
5           --with-pcre=../pcre-8.44 \
6           --with-zlib=../zlib-1.2.11 \
7           --with-openssl=../openssl-1.1.1g
```

注意以上依赖包要改成自己的实际版本。

一阵 checking 无报错信息之后配置成功

然后就可以编译了：

```
1 make
```

一阵编译无报错信息之后安装：

```
1 sudo make install
```

4、使用

```
1 /usr/local/nginx/nginx -h
2
3 nginx version: nginx/1.18.0
4 Usage: nginx [-?hvTtQ] [-s signal] [-c filename] [-p prefix] [-g di
rectives]
5
6 Options:
7  -?, -h          : this help
8  -v              : show version and exit
9  -V              : show version and configure options then exit
10 -t              : test configuration and exit
11 -T              : test configuration, dump it and exit
12 -q              : suppress non-error messages during configuration t
esting
13 -s signal       : send signal to a master process: stop, quit, reope
n, reload
14 -p prefix       : set prefix path (default: /usr/local/nginx/)
15 -c filename     : set configuration file (default: /usr/local/nginx/
nginx.conf)
16 -g directives   : set global directives out of configuration file
```

5、配置简易操作命令

如果怕nginx命令操作比较麻烦，可以参考lnmp设计的操作脚本

```
1 sudo lnmp -h
2 +-----+
3 |      Manager for LNMP, Written by Licens   |
4 +-----+
5 |                  https://lnmp.org           |
6 +-----+
7 Usage: lnmp {start|stop|reload|restart|kill|status}
8 Usage: lnmp {nginx|mysql|mariadb|php-fpm|pureftpd} {start|stop|relo
ad|restart|kill|status}
9 Usage: lnmp vhost {add|list|del}
10 Usage: lnmp database {add|list|edit|del}
11 Usage: lnmp ftp {add|list|edit|del|show}
12 Usage: lnmp ssl add
13 Usage: lnmp {dnssl|dns} {cx|ali|cf|dp|he|gd|aws}
14 Usage: lnmp onlyssl {cx|ali|cf|dp|he|gd|aws}
```

或配置alias命令别名：

将以下代码放到 ~/.bash_profile 文件中

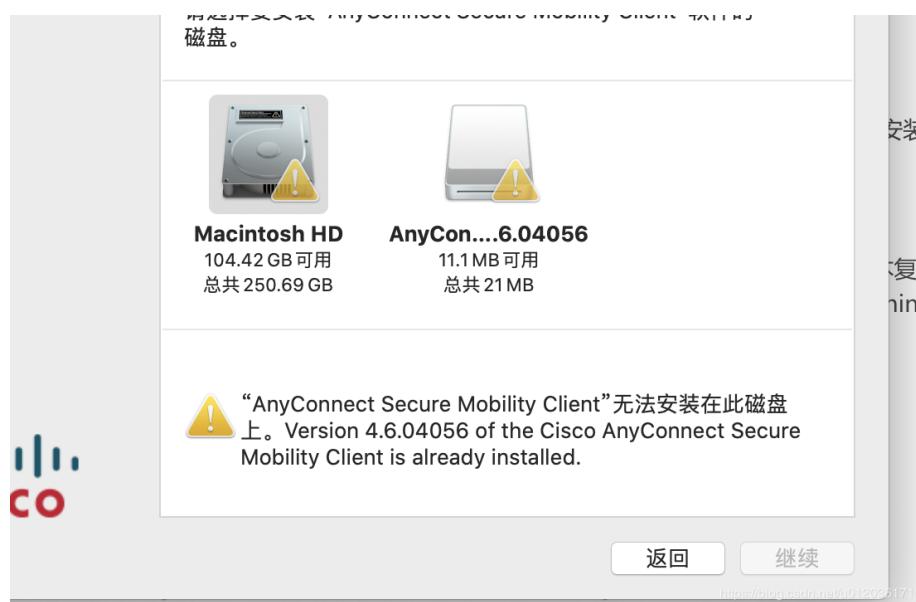
```
1 alias nginx restart='sudo /usr/local/nginx/nginx -s stop && sudo /us
r/local/nginx/nginx'
2 alias nginx start='sudo /usr/local/nginx/nginx'
3 alias nginx stop='sudo /usr/local/nginx/nginx -s stop'
```

source ~/.bash_profile

MAC上卸载AnyConnect后无法再重新安装问题 解决

升级了MAC的 Catalina 后我的AnyConnect有兼容性问题，于是使用APPCleaner 卸载了AnyConnect 并重新安装新包。但是卸载后重新安装后出现了

“AnyConnect Secure Mobility Client”无法安装在此磁盘上。Version 4.6.04056 of the Cisco AnyConnect Secure Mobility Client is already installed.
发现肯定是没卸载干净...



于是运行：

```
1 // 进入anyconnect 脚本目录
2 cd /opt/cisco/anyconnect/bin
3 // 需要sudo权限执行 脚本
4 sudo -s
5 // 所有卸载的脚本全都执行一遍
6 ./uninstall.sh
```

```
bash-3.2# ./uninstall.sh
Exiting Cisco AnyConnect Secure Mobility Client
Uninstalling Cisco AnyConnect ISE Posture Module...
Successfully removed Cisco AnyConnect ISE Posture Module from the system.
Uninstalling Cisco AnyConnect Posture Module...
Successfully removed Cisco AnyConnect Posture Module from the system.
Uninstalling Cisco AnyConnect Web Security Module...
Successfully removed Cisco AnyConnect Web Security Module from the system.
Uninstalling Cisco AnyConnect Umbrella Roaming Security Module...
Successfully removed Cisco AnyConnect Umbrella Roaming Security Module from the
system.
Uninstalling Cisco AnyConnect Network Visibility Module...
Successfully removed Cisco AnyConnect Network Visibility Module from the system.
Uninstalling Cisco AnyConnect Secure Mobility Client...
Successfully removed Cisco AnyConnect Secure Mobility Client from the system.
```

搞定，重新安装成功

若执行以上步骤后还是无法安装，提示 already installed 错误，应该是卸载残留问题，解决方法参考如下文章

Here is yuque doc card, click on the link to
view:<https://www.yuque.com/u316337/tforki/aen8yh>

参考文章：<https://blog.csdn.net/u012036171/article/details/102511221>

Mac 上软件推荐

1、 截图软件

[截图 介绍](#)

[Snip 介绍](#)

2、 软件卸载工具

[AppCleaner 下载地址](#)

3、 [CheatSheet](#) 展示应用快捷键 [下载地址](#)

4、 [PicGo](#) 跨平台图床工具 [官网](#)

5、 [itsycal](#) 日历工具

6、 [SourceTree](#) 版本管理工具

7、 [Postman](#) HTTP调试工具

8、 [Go2Shell](#) [下载地址](#) [下载地址2](#) 这是一款十分小巧的Finder插件，在Finder中，我们只要点击Go2Shell按钮，Go2Shell就会帮我们在命令行中打开当前目录。

9、 [Alfred](#) 系统全局搜索

10、 [Sequel Pro](#) MySql管理工具

11、 [Charles](#) 抓包工具

12、 [KeePassX](#) 密码管理工具

13、 [PopClip](#)，划词增强工具，包括但不限于（上百个官方扩展）：复制，粘贴，翻译，定制搜索（淘宝，知乎，google...）

破解版下载地址：

链接: <https://pan.baidu.com/s/1JyI9xjUKsKlzrXApAcwig> 提取码: hab8

14、 [iStat Menus](#)，菜单栏系统监控（内存，网速，磁盘，电池.....）

15、 [thefuck](#) 终端命令自动纠错功能

16、 [uTools](#) 一个极简、插件化、跨平台的现代桌面软件。通过自由选配丰富的插件，打造你得心应手的工具集合。加入应用插件，建立快捷键快速调起某个应用。

17、 [teamviewer](#) 远程控制、远程会议软件。

Mac 上简易的ssh快捷菜单工具： shuttle

官网地址: <http://fitztrev.github.io/shuttle/>

github 地址: <https://github.com/fitztrev/shuttle>



配置文件: `~/.shuttle.json`

实例:

```
1 {
2   "_comments": [
3     "Valid terminals include: 'Terminal.app' or 'iTerm'",
4     "In the editor value change 'default' to 'nano', 'vi', or another
      terminal based editor.",
5     "Hosts will also be read from your ~/.ssh/config or /etc/ssh_con
      fig file, if available",
6     "For more information on how to configure, please see http://fit
      ztrev.github.io/shuttle/"
7   ],
8   "editor": "default",
9   "launch_at_login": false,
10  "terminal": "Terminal.app",
11  "iTerm_version": "nightly",
12  "default_theme": "Homebrew",
13  "open_in": "new",
14  "show_ssh_config_hosts": false,
15  "ssh_config_ignore_hosts": [ ],
16  "ssh_config_ignore_keywords": [ ],
17  "hosts": [
18    {
19      "testarray": [
20        {"cmd": "ssh -i ~/id_rsa -o ServerAliveInterval=60 test@4
      0.68.190.133", "inTerminal": "tab", "name": "test1", "theme": "homebre
      w", "title": "title of tab"}, {
21          "cmd": "ssh -o ServerAliveInterval=60 test2@47.96.117.14
      ", "inTerminal": "tab", "name": "test2", "theme": "homebrew", "title":
      "test2"}, ]
22    ],
23  },
24  { "cmd": "ssh -o ServerAliveInterval=60 test@192.168.1.223",
      "inTerminal": "tab", "name": "桌下服务器", "theme": "homebrew", "titl
```

```
    e": "zhuoxia" },
25      { "cmd": "ssh -i '/home/xinjiapo/.pem' ubuntu@ec2.ap-southeast-1.co
mpute.amazonaws.com", "inTerminal": "tab", "name": "test", "them
e":"homebrew", "title": "test" }
26
27
28 ]
29 }
```

mac上charles 抓包手机端网络请求

配置手机端的http代理:

- 1 第一步：在mac的命令行下查看本机ip
- 2
- 3 第二步，配置手机的ip代理，iPhone的“设置”->“无线局域网”中，可以看到当前连接的wifi名，
- 4 通过点击右边的详情键，可以看到当前连接上的wifi的详细信息，包括IP地址，子网掩码等信息。
- 5 在其最底部有“HTTP代理”一项，我们将其切换成手动，然后填上Charles运行所在的电脑的IP，
- 6 以及charles proxy setting设置的端口
- 7
- 8 第三步：打开mac端自动发现代理，打开mac系统偏好设置 -> 网络 -> 高级 -> 代理 -> 勾选“自动发现代理”
- 9
- 10 第四步，到这里配置已经完成，就这么简单，然后我们可以访问微博客户端，
- 11 我们清晰的可以看到抓取的数据截图如下：

Mac 软件卸载残留删除方法

Mac卸载残留，以Cisco为例

Mac os系统的卸载大多时候打开应用程序，把要卸载的程序拖到废纸篓即可，如果某些软件出现问题，卸载过程中有残留，就会导致新的应用镜像无法正常安装，本文以Cisco-Anyconnect为例，解决卸载残留。

查看残留文件

```
1 $ pkgutil --pkgs|grep com.cisco
2
3 com.cisco.pkg.anyconnect.vpn
4 com.cisco.pkg.anyconnect.fireamp
5 com.cisco.pkg.anyconnect.dart
6 com.cisco.pkg.anyconnect.websecurity_v2
7 com.cisco.pkg.anyconnect.nvm_v2
8 com.cisco.pkg.anyconnect.umbrella
9 com.cisco.pkg.anyconnect.iseposture
10 com.cisco.pkg.anyconnect.posture
```

删除残留文件

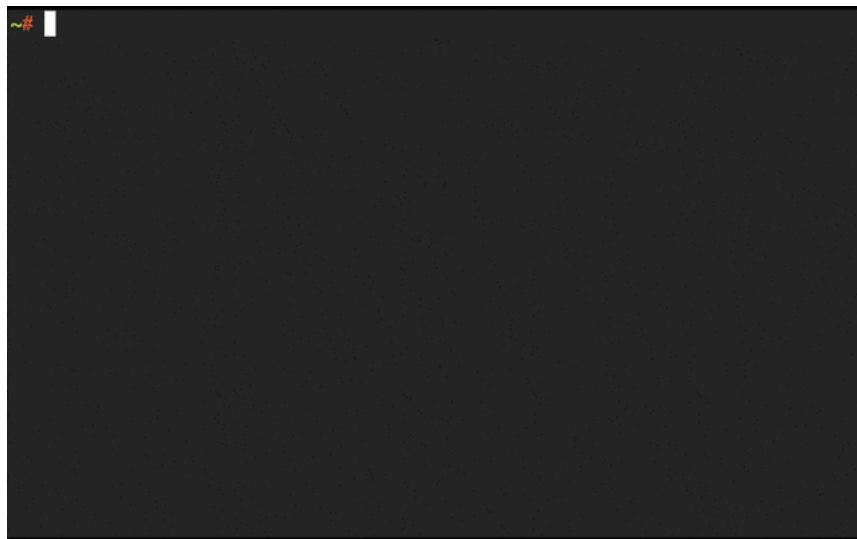
找到文件之后，再次使用pkgutil 将冲突的残留文件forget掉即可，如下命令：

```
1 $ sudo pkgutil --forget com.cisco.pkg.anyconnect.vpn
2
3 Forgot package 'com.cisco.pkg.anyconnect.vpn' on '/'.
```

依次forget掉残留文件即可

参考文章：https://blog.csdn.net/dinl_vin/article/details/89472755

命令窗口多开工具 xpanes



现已支持 ubuntu centos rhel mac ,详情见：

gitHub地址 : <https://github.com/greymd/tmux-xpanes>

使用实例：

```
1 # 屏幕拆分多屏，同时监控多个日志文件
2 xpanes -c "tail -f {}" ~/alidata/log/php_errors.log  ~/alidata/log/nginx_access.log
```

Mac上制作Linux U盘启动盘

一般我们都是在Windows上用UltraISO软碟通制作系统启动盘。在Mac上制作一个系统的启动盘的软件非常的稀少的，特别是制作非mac系统的软件。以下是在Mac上制作系统启动盘方法：

1、下载系统镜像到mac

2、把足够空间的U盘插入到Mac

3、打开命令终端，输入命令

```
1 $ diskutil list
```

```
/dev/disk0 (internal, physical):
#:          TYPE NAME                SIZE    IDENTIFIER
0: GUID_partition_scheme          *251.0 GB  disk0
1:      EFI EFI                  314.6 MB  disk0s1
2: Apple_APFS Container disk1   250.7 GB  disk0s2

/dev/disk1 (synthesized):
#:          TYPE NAME                SIZE    IDENTIFIER
0: APFS Container Scheme -         +250.7 GB  disk1
                           Physical Store disk0s2
1: APFS Volume Macintosh HD - 数据  171.2 GB  disk1s1
2: APFS Volume Preboot           81.0 MB   disk1s2
3: APFS Volume Recovery          526.6 MB  disk1s3
4: APFS Volume VM                9.7 GB    disk1s4
5: APFS Volume Macintosh HD      11.0 GB   disk1s5

/dev/disk2 (external, physical):
#:          TYPE NAME                SIZE    IDENTIFIER
0: FDisk_partition_scheme          *4.1 GB   disk2
1:      Apple_HFS wml2            4.1 GB    disk2s1

/dev/disk3 (disk image):
#:          TYPE NAME                SIZE    IDENTIFIER
0:          DAEMON Tools Lite       +12.3 MB  disk3
```

4、此时会得到U盘的设备名字 /dev/disk2 通过这个U盘存储容量可以知道。

5、接着输入卸载U盘命令

```
1 $ diskutil unmountDisk /dev/disk2
```

会提示卸载成功

6、卸载成功后，输入制作系统盘命令

```
1 sudo dd if={iso系统路径} of=/dev/disk2 bs=10m
2 # 实例：
3 sudo dd if=/Users/test/Desktop/ubuntu-19.10-desktop-amd64.iso of=/dev/disk2 bs=10m
```

其中{/dev/disk2}是设备名字，bs=10m是刻录的速度。回车输入管理员密码，等待执行完成即可刻录好一个系统盘。

Apache 与 Nginx 比较

当前基本Nginx是标配,首选Nginx

下面区别仅做参考吧：

Apache

Web服务器

特点

rewrite , 比nginx 的rewrite 强大；

模块超多，基本想到的都可以找到；

少bug , nginx 的bug 相对较多；

超稳定；

Apache在处理动态有优势，nginx比较适合跑静态

Nginx

Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，负载均衡服务器

工作原理：

Nginx会按需同时运行多个进程：一个主进程(master)和几个工作进程(worker)，配置了缓存时还会有缓存加载器进程(cache loader)和缓存管理器进程(cache manager)等。所有进程均是仅含有一个线程，并主要通过“共享内存”的机制实现进程间通信。主进程以root用户身份运行，而worker、cache loader和cache manager均应以非特权用户身份运行。nginx采用了模块化、事件驱动、异步、单线程及非阻塞的架构，并大量采用了多路复用及事件通知机制。在nginx中，连接请求由为数不多的 几个仅包含一个线程的进程worker以高效的回环(run-loop)机制进行处理，而每个worker可以并行处理数千个的并发连接及请求。

特点：

轻量级，同样起web 服务，比apache 占用更少的内存及资源；

并发能力比apache强，nginx 处理请求是异步非阻塞的，而apache 则是阻塞型的，在高并发下nginx 能保持低资源低消耗高性能；

Nginx 配置简洁，Apache 复杂；

最核心的区别在于apache是同步多进程模型，一个连接对应一个进程；nginx是异步的，多个连接（万级）可以对应一个进程；在Apache+PHP（prefork）模式下，如果PHP处理慢或者前端压力很大的情况下，很容易出现Apache进程数飙升，从而拒绝服务的现象。

nginx 处理静态文件好，静态处理性能比 apache 高三倍以上

总结：

一般来说，需要性能的web 服务，用nginx 。如果不需要性能只求稳定，那就apache 吧。

epoll(freebsd 上是 kqueue) 网络 IO 模型是 nginx 处理性能高的根本理由，但并不是所有的情况下都是 epoll 大获全胜的，如果本身提供静态服务的就只有寥寥几个文件，apache 的 select 模型或许比 epoll 更高性能。当然，这只是根据网络 IO 模型的原理作的一个假设，真正的应用还是需要实测了再说的。

更为通用的方案是，前端 nginx 抗并发，后端 apache 集群，（即把Nginx用作代理服务器，而把 Apache用作后台服务器）配合起来会更好。

Nginx 配置子目录项目

在项目中有时一个完整的项目需要整合在另外个项目中，作为一个子模块存在

如有两个项目 help 、 blog ,根目录分别为/alidata/www/help.abx.net, /alidata/www/blog.abx.net

若要用域名<http://test.abx.net/help>访问help, 用<http://test.abx.net/blog>访问blog项目，可参考如下配置

```
1 server
2 {
3     listen 80;
4     server_name test.abx.net;
5     index home.html index.htm index.php default.html default.htm
6     default.php;
7     root /alidata/www/test.abx.net/dist;
8
9     #error_page 404 /404.html;
10
11    location /help {
12        root /alidata/www/help.abx.net;
13        if (!-e $request_filename) {
14            rewrite ^(.*)$ /help/index.php$1 last;
15        }
16
17        location ~ \.php(\/.*)*$ {
18            fastcgi_split_path_info ^(.+?.php)(/.*)$;
19            fastcgi_pass unix:/tmp/php-cgi.sock;
20            fastcgi_index index.php;
21            fastcgi_param SCRIPT_FILENAME /alidata/www/help.abx.net$fastcgi_script_name;
22            include fastcgi_params;
23        }
24        access_log /alidata/log/nginx/access/help.abx.net.log;
25    }
26
27    location /blog {
28        root /alidata/www/blog.abx.net;
29        try_files $uri $uri/ /blog/index.php$args;
30        include enable-php.conf;
31        access_log /alidata/log/nginx/access/blog.abx.net.log;
32    }
33
34    location /api/ {
35        proxy_pass https://localhost:3010;
36    }
37
38    # Deny access to PHP files in specific directory
39    location ~ /(wp-content|uploads|wp-includes|images)/.*\.php$ {
40        deny all;
41    }
42
43    location ~ *.gif|jpg|jpeg|png|bmp|swf$ {
44        expires 30d;
45    }
```

```
44     location ~ \.(js|css)?$ {
45     {
46         expires      12h;
47     }
48
49     location ~ /.well-known {
50         allow all;
51     }
52     location ~ /\. {
53     {
54         deny all;
55     }
56
57     access_log  /alidata/log/nginx/access/test.abx.net.log;
58 }
```

nginx+php使用open_basedir限制站点目录防止跨站

方法1、在Nginx配置文件中配置

```
1 fastcgi_param PHP_ADMIN_VALUE "open_basedir=$document_root:/tmp/:/proc/:/其他想允许访问的目录/";
```

通常nginx的站点配置文件里用了include fastcgi.conf;, 这样的, 把这行加在fastcgi.conf里就OK了。如果某个站点需要单独设置额外的目录, 把上面的代码写在include fastcgi.conf;这行下面就OK了, 会把fastcgi.conf中的设置覆盖掉。
这种方式的设置需要重启nginx后生效。

方法2、在php.ini中加入配置:

```
1 [HOST=test.aaa.io]
2 open_basedir=/alidata/www/test.aaa.io/:/tmp/
3 或
4 [PATH=/alidata/www/test.aaa.io/public]
5 open_basedir=/alidata/www/test.aaa.io/:/tmp/
```

[HOST] 和 [PATH] 使用说明

[HOST=]

定义一组只在 host 主机上生效的php.ini指令。

```
1 [HOST=dev.site.com]
2 error_reporting = E_ALL
3 display_errors = On
```

[PATH=]

定义一组只从指定路径运行时生效的php.ini指令。

```
1 [PATH=/home/site/public/secure]
2 auto-prepend-file=security.php
```

这种方式的设置需要重启php-fpm后生效。

方法3、在网站根目录下创建.user.ini并写入:

open_basedir=/home/www/www.server110.com:/tmp/:/proc/

这种方式不需要重启nginx或php-fpm服务。安全起见应当取消掉.user.ini文件的写权限。

关于.user.ini文件的详细说明:

<http://php.net/manual/zh/configuration.file.per-user.php>

文章转自: <http://blog.chinaunix.net/uid-26729093-id-4780651.html>

nginx log_format 配置

nginx log_format用来设置nginx请求日志纪录格式

nginx.conf

```
1 # 添加$request_body参数
2 log_format log_format_name_1 '$remote_addr - $remote_user [$time_local]
1] "$request"
3     '$status $body_bytes_sent "$http_referer"
4     '"$http_user_agent" "$http_x_forwarded_for"
5     '"$request_body" $request_time $upstream_response_time';
```

./vhost/*.conf

```
1 # 注意一定要在后面带日志格式名称
2 access_log /alidata/log/nginx/*.access.log log_format_name_1;
```

参数说明

\$remote_addr #记录访问网站的客户端ip地址

\$remote_user #远程客户端用户名

\$time_local #记录访问时间与时区

\$request #用户的http请求起始行信息

\$status #http状态码，记录请求返回的状态码，例如：200、301、404等

\$body_bytes_sent #服务器发送给客户端的响应body字节数

\$http_referer #记录此次请求是从哪个连接访问过来的，可以根据该参数进行防盗链设置。

\$http_user_agent #记录客户端访问信息，例如：浏览器、手机客户端等

\$http_x_forwarded_for #当前端有代理服务器时，设置web节点记录客户端地址的配置，此参数生效的前提是代理服务器也要进行相关的x_forwarded_for设置

\$request_body post上传的数据

\$request_time 整个请求的总时间

\$upstream_response_time 请求过程中，upstream响应时间

Inmp环境站点502与504错误分析

状态代码解释

502 Bad Gateway: 作为网关或者代理工作的服务器尝试执行请求时，从上游服务器接收到无效的响应。

504 Gateway Time-out: 作为网关或者代理工作的服务器尝试执行请求时，未能及时从上游服务器（URI标识出的服务器，例如HTTP、FTP、LDAP）或者辅助服务器（例如DNS）收到响应。

502 Bad Gateway原因分析

将请求提交给网关如php-fpm执行，但是由于某些原因没有执行完毕导致php-fpm进程终止执行。说到这里，这个问题就很明了了，与网关服务如php-fpm的配置有关了。

php-fpm.conf配置文件中有两个参数就需要你考虑到，分别是max_children和request_terminate_timeout。

max_children最大子进程数，在高并发请求下，达到php-fpm最大响应数，后续的请求就会出现502错误的。可以通过netstat命令来查看当前连接数。

request_terminate_timeout设置单个请求的超时终止时间。还应该注意到php.ini中的max_execution_time参数。当请求终止时，也会出现502错误的。

当积累了大量的php请求，你重启php-fpm释放资源，但一两分钟不到，502又再次呈现，这是什么原因导致的呢？

这时还应该考虑到数据库，查看下数据库进程是否有大量的locked进程，数据库死锁导致超时，前端终止了继续请求，但是SQL语句还在等待释放锁，这时就要重启数据库服务了或kill掉死锁SQL进程了。

504 Gateway Time-out原因分析

504错误一般是与nginx.conf配置有关了。主要与以下几个参数有关：fastcgi_connect_timeout、fastcgi_send_timeout、fastcgi_read_timeout、fastcgi_buffer_size、fastcgi_buffers、fastcgi_busy_buffers_size、fastcgi_temp_file_write_size、fastcgi_intercept_errors。特别是前三个超时时间。如果fastcgi缓冲区太小会导致fastcgi进程被挂起从而演变为504错误。

小结

1. max_children
2. request_terminate_timeout、max_execution_time
3. 数据库
4. 网关服务是否启动如php-fpm

504错误主要查看nginx.conf关于网关如fastcgi的配置。

nginx杜绝HTTP伪请求头攻击

在查看nginx日志时，经常发现一种攻击方式，
不用GET，也没用POST，而是用了一个16进制伪码：\0x01作为请求method。
其目标是使伺服器溢出，并导致了大量400 bad request包外流。

所以应付此类攻击最好的方式就是直接掐断连接。

配置方式

下面代码放进http{}段内

```
1 map $request_method $ban_method{
2     default 1;
3     GET 0;
4     POST 0;
5 }
```

下面代码放进server{}段内

```
1 if ( $ban_method = 1 ) {
2     return 444;
3 }
```

444: Nginx上HTTP服务器扩展。 服务器不向客户端返回任何信息，并关闭连接（有助于恶意软件的威胁）。

map 说明

map 指令介绍：

map 指令是由 ngx_http_map_module 模块提供的， 默认情况下安装 nginx 都会安装该模块。

map 的主要作用是创建自定义变量，通过使用 nginx 的内置变量，去匹配某些特定规则，如果匹配成功则设置某个值给自定义变量。而这个自定义变量又可以作于他用。

直接看个例子理解起来比较清晰：

场景： 匹配请求 url 的参数，如果参数是 debug 则设置 \$foo = 1， 默认设置 \$foo = 0

```
1 map$args $foo {
2     default 0;
3     debug    1;
4 }
```

解释：

\$args 是nginx内置变量，就是获取的请求 url 的参数。如果\$args 匹配到 debug 那么 \$foo 的值会被设为 1，如果\$args 一个都匹配不到 \$foo 就是default 定义的值，在这里就是 0

map 语法

```
map $var1 $var2 {...}
```

map 指令的三个参数：

- 1 1、**default**： 指定源变量匹配不到任何表达式时将使用的默认值。当没有设置 **default**，将用一个空的字符串作为默认的结果。
- 2
- 3 2、**hostnames**： 允许用前缀或者后缀掩码指定域名作为源变量值。这个参数必须写在值映射列表的最前面。
- 4
- 5 3、**include**： 包含一个或多个含有映射值的文件。

#####在 Nginx 配置文件中的作用段： http{}，注意 map 不能写在 server{} 否则会报错

map 的 \$var1 为源变量，通常可以是 nginx 的内置变量，\$var2 是自定义变量。\$var2 的值取决于 \$var1 在对应表达式的匹配情况。如果一个都匹配不到则 \$var2 就是 default 对应的值。

一个正则表达式如果以“~”开头，表示这个正则表达式对大小写敏感。以“~*”开头，表示这个正则表达式对大小写不敏感。

```
1 map $http_user_agent $agent {  
2     default "";  
3     ~curl curl;  
4     ~*apachebench" ab;  
5 }
```

正则表达式里可以包含命名捕获和位置捕获，这些变量可以跟结果变量一起被其它指令使用。

```
1 map $uri $value {  
2     /abc                     /index.php;  
3     ^/teacher/(?<suffix>.*)$ /boy/;  
4     ~/fz(/.*)                  /index.php?fz=1;  
5 }
```

注意：不能在map块里面引用命名捕获或位置捕获变量。如`~^/qupeicom/(.*) /peiyin/$1;`这样会报错
nginx: [emerg] unknown variable

注意二：如果源变量值包含特殊字符如‘~’，则要以‘\’来转义。

```
1 map $http_referer $value {  
2     Mozilla      111;  
3     \~Mozilla    222;  
4 }
```

源变量匹配表达式对应的结果值可以是一个字符串也可以是另外一个变量。

```
1 map $http_referer $value {  
2     Mozilla      'chrom';  
3     \~safity     $http_user_agent;  
4 }
```

实例（一）

使用 map 来实现允许多个域名跨域访问的问题

如果是允许单域名跨域访问直接配置就行了，如下：

```
1 # 这些配置可以写在 http{} 或者 server{} 都是支持的。
2 add_header Access-Control-Allow-Origin "http://www.tutu.com";
3 add_header Access-Control-Allow-Methods "POST, GET, PUT, OPTIONS, DELETE";
4 add_header Access-Control-Max-Age "3600";
5 add_header Access-Control-Allow-Headers "Origin, X-Requested-With, Content-Type, Accept";
```

上面的配置只允许 <http://www.tutu.com> 跨域访问，如果要支持所有域名都可以跨域调用该站。把上面一行改成这样，不过不推荐这样做，因为不安全

```
1 add_header Access-Control-Allow-Origin "*";
```

如果不允许所有，但是又需要允许多个域名，那么就需要用到 map

```
1 map $http_origin $corsHost {
2     default 0;
3     "~http://www.haibakeji.com" http://www.haibakeji.com;
4     "~http://m.haibakeji.com" http://m.haibakeji.com;
5     "~http://wap.haibakeji.com" http://wap.haibakeji.com;
6 }
7 server
8 {
9     listen 80;
10    server_name www.haibakeji.com;
11    root /nginx;
12    location /
13    {
14        add_header Access-Control-Allow-Origin $corsHost;
15    }
16 }
```

实例（二）

使用源变量（通常是 nginx 内置变量）匹配一些规则，创建自定义变量，然后在页面输出。这通常在调试的时候非常有用

```
1 http {
2     map $uri $match {
3         ~^/hello/(.*) http://www.hello.com/;
4     }
5 }
6 server {
7     listen      8080;
8     server_name test.hello.com;
9
10    location /hello {
```

```
11         default_type text/plain;
12         echo uri: $uri;
13         echo match: $match;
14         echo capture: $1;
15         echo new: $match$1;
16     }
17 }
```

map 涉及的性能问题

大家可能会有一个问题，map 既然只能用在 http 段，这是全局的啊。这个设置会让访问所有虚拟主机的请求都要匹配并设置一遍变量的值，然而事实并非如此，对于没有用到相关变量的请求来说，并不会执行 map 操作。就没有性能上的损失。

匹配优先级问题

如果匹配到多个特定的变量，如掩码和正则同时匹配，那么会按照下面的顺序进行选择：

- 没有掩码的字符串
- 最长的带前缀的字符串，例如：“*.example.com”
- 最长的带后缀的字符串，例如：“mail.*”
- 按顺序第一个先匹配的正则表达式（在配置文件中体现的顺序）

启用nginx status状态页详解

nginx配置

在默认主机里面加上location或者你希望能访问到的主机里面。

```
1 server {
2     listen 80;
3     server_name 127.0.0.1;
4
5     #location /phpfpm_status {
6     #         fastcgi_pass unix:/tmp/php-cgi.sock;
7     #         include fastcgi_params;
8     #         fastcgi_param SCRIPT_FILENAME $fastcgi_script_name;
9     #}
10
11    location /nginx_status {
12        stub_status on;
13        access_log off;
14        #allow 127.0.0.1;
15        #deny all;
16    }
17 }
```

重启nginx使配置生效

```
1 # /etc/init.d/nginx restart
```

打开status页面

在浏览器打开 http://127.0.0.1/nginx_status

```
1 Active connections: 86
2 server accepts handled requests
3 269 269 565
4 Reading: 0 Writing: 11 Waiting: 75
```

nginx status详解

```
1 active connections - 活跃的连接数量
2 server accepts handled requests - 总共处理了269个连接，成功创建269次握手，总共处理了565个请求
3 reading - 读取客户端的连接数。
4 writing - 响应数据到客户端的数量
5 waiting - 开启 keep-alive 的情况下，这个值等于 active - (reading+writing)，意思就是 Nginx 已经处理完正在等候下一次请求指令的驻留连接。
```

注意

在nginx中可以开启Basic Auth登录认证，经常更换密码，这样就不会对外暴露 nginx_status 信息；

若nginx配置加到其他域名server中访问不成功，就单独配置出来了。

nginx配置auth_basic登录认证的方法

有时候我们通过nginx搭建了一台文件服务器，一般来讲是公开的，但我们又希望该服务器不让人看到，有人可能会搭建一个登录系统，但是太麻烦，也没太大必要，比较简单的方法是配置Basic Auth登录认证

纯后台的应用都可以加nginx basic auth提高安全性

方法步骤

1、安装htpasswd

htpasswd是Apache密码生成工具，Nginx支持auth_basic认证，因此我们可以将生成的密码用于Nginx中

Ubuntu:

```
1 sudo apt-get install apache2-utils
```

CentOS:

```
1 sudo yum -y install httpd-tools
```

参数如下：

```
1 -c 创建passwordfile.如果passwordfile 已经存在,那么它会重新写入并删去原有内容.
2 -n 不更新passwordfile, 直接显示密码
3 -m 使用MD5加密 (默认)
4 -d 使用CRYPT加密 (默认)
5 -p 使用普通文本格式的密码
6 -s 使用SHA加密
7 -b 命令行中一并输入用户名和密码而不是根据提示输入密码, 可以看见明文, 不需要交互
8 -D 删除指定的用户
```

2、生成密码

进入密码文件存储目录，如本次测试为 `/usr/local/nginx/conf/passwd`

```
1 cd /usr/local/nginx/conf/passwd
2 #生成密码
3 htpasswd -c ./passwordfile username
4 #执行上命令后会要求输入两次密码, ./passwordfile 是在当前目录下创建密码文件passwordfile , username即为需要设置的账号
```

3、载入配置

接下来在Nginx配置文件中（通常是server段内），加入如下两行，并重载Nginx（service nginx reload）即可生效。

```
1 server
2   {
3     listen 80 default_server reuseport;
4     server_name _;
5     index index.html index.htm index.php;
6     root /home/wwwroot/default;
7
8     auth_basic "请输入账号密码";    #这里是验证时的提示信息
9     auth_basic_user_file /usr/local/nginx/conf/passwd/ip_passwdfile;
10
11    include enable-php.conf;
12
13    location /nginx_status
14    {
15      stub_status on;
16      access_log off;
17    }
18
19    location ~ \.(gif|jpg|jpeg|png|bmp|swf)$
20    {
21      expires 30d;
22    }
23
24    location ~ \.(js|css)?$
25    {
26      expires 12h;
27    }
28
29    location ~ /.well-known {
30      allow all;
31    }
32
33    location ~ /.
34    {
35      deny all;
36    }
37
38    access_log /home/wwwlogs/access.log;
39 }
```

4、访问测试

再访问站点，提示需要输入用户名和密码才可以访问，此方法适合不宜公开的站点，或不想对其专门做账号登录系统的时候。这样可避免被弱口令扫描，无疑再上了一把锁。



注意

对于这种有HTTP Basic Authentication协议验证的页面，如果使用curl抓取的话，可以加上账号密码进行请求：

```
1 curl -u username:password URL
```

如果用wget下载，可以用：

```
1 wget --http-user= --http-passwd=passwd URL
```

nginx 查看访问 IP 并封禁 IP 详解

1、查找服务器所有访问者ip方法：

```
1 awk '{print $1}' nginx_access.log |sort |uniq -c|sort -n
```

nginx.access.log 为nginx访问日志文件所在路径

会到如下结果，前面是ip的访问次数，后面是ip，很明显我们需要把访问次数多的ip并且不是蜘蛛的ip屏蔽掉，如下面结果，

若 66.249.79.84 不为蜘蛛则需要屏蔽：

```
1      89 106.75.133.167
2      90 118.123.114.57
3      91 101.78.0.210
4      92 116.113.124.59
5      92 119.90.24.73
6      92 124.119.87.204
7      119 173.242.117.145
8    4320 66.249.79.84
```

2、屏蔽IP的方法：

在nginx的安装目录下面,新建屏蔽ip文件，命名为guolv_ip.conf，以后新增加屏蔽ip只需编辑这个文件即可。

加入如下内容并保存：

```
1 deny 66.249.79.84 ;
```

在nginx的配置文件nginx.conf中加入如下配置，可以放到http, server, location, limit_except语句块，需要注意相对路径，本例当中nginx.conf, guolv_ip.conf在同一个目录中。

```
1 include guolv_ip.conf;
```

保存nginx.conf文件，然后测试现在的nginx配置文件是否是合法的：

```
1 nginx -t
```

如果配置没有问题，就会输出：

```
1 the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok
2 configuration file /usr/local/nginx/conf/nginx.conf test is successful
```

如果配置有问题就需要检查下哪儿有语法问题，如果没有问题，需要执行下面命令，重载 nginx 配置文件：

```
1 service nginx reload
```

3、注意：

屏蔽ip的配置文件既可以屏蔽单个ip，也可以屏蔽ip段，或者只允许某个ip或者某个ip段访问。

```
1 //屏蔽单个ip访问
2
3 deny IP;
4
5 //允许单个ip访问
6
7 allow IP;
8
9 //屏蔽所有ip访问
10
11 deny all;
12
13 //允许所有ip访问
14
15 allow all;
16
17 //屏蔽整个段即从123.0.0.1到123.255.255.254访问的命令
18
19 deny 123.0.0.0/8
20
21 //屏蔽IP段即从123.45.0.1到123.45.255.254访问的命令
22
23 deny 124.45.0.0/16
24
25 //屏蔽IP段即从123.45.6.1到123.45.6.254访问的命令
26
27 deny 123.45.6.0/24
28
29 //如果你想实现这样的应用，除了几个IP外，其他全部拒绝，
30 //那需要你在guolv_ip.conf中这样写
31
32 allow 1.1.1.1;
33 allow 1.1.1.2;
34 deny all;
```

- 1 单独网站屏蔽IP的方法，把include guolv_ip.conf; 放到网址对应的在server{}语句块，
- 2 所有网站屏蔽IP的方法，把include guolv_ip.conf; 放到http {}语句块。

参考：<http://www.nginx.cn/2487.html>

nginx 工作原理

Nginx的模块与工作原理

Nginx由内核和模块组成，其中，内核的设计非常微小和简洁，完成的工作也非常简单，仅仅通过查找配置文件将客户端请求映射到一个location block（location是Nginx配置中的一个指令，用于URL匹配），而在location中所配置的每个指令将会启动不同的模块去完成相应的工作。

Nginx的模块从结构上分为核心模块、基础模块和第三方模块：

- 1 核心模块：HTTP模块、EVENT模块和MAIL模块
- 2
- 3 基础模块：HTTP Access模块、HTTP FastCGI模块、HTTP Proxy模块和HTTP Rewrite模块，
- 4
- 5 第三方模块：HTTP Upstream Request Hash模块、Notice模块和HTTP Access Key模块。

Nginx的模块从功能上分为如下三类。

- 1 Handlers（处理器模块）。此类模块直接处理请求，并进行输出内容和修改headers信息等操作。Handlers处理器模块一般只能有一个。
- 2
- 3 Filters（过滤器模块）。此类模块主要对其他处理器模块输出的内容进行修改操作，最后由Nginx输出。
- 4
- 5 Proxies（代理类模块）。此类模块是Nginx的HTTP Upstream之类的模块，这些模块主要与后端一些服务比如FastCGI等进行交互，
- 6 实现服务代理和负载均衡等功能。

图展示了Nginx模块常规的HTTP请求和响应的过程。

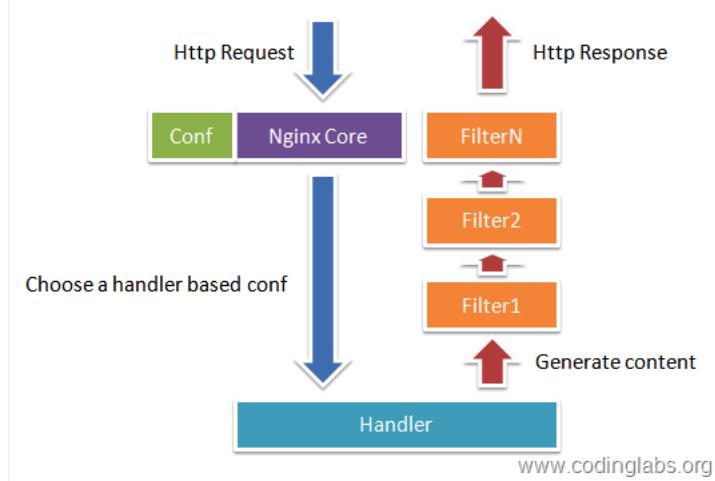


图1-1展示了Nginx模块常规的HTTP请求和响应的过程。

Nginx的进程模型

在工作方式上，Nginx分为单工作进程和多工作进程两种模式。在单工作进程模式下，除主进程外，还有一个工作进程，工作进程是单线程的；在多工作进程模式下，每个工作进程包含多个线程。Nginx默认为单工

作进程模式。

Nginx在启动后，会有一个master进程和多个worker进程。

master进程

主要用来管理worker进程，包含：接收来自外界的信号，向各worker进程发送信号，监控worker进程的运行状态，当worker进程退出后(异常情况下)，会自动重新启动新的worker进程。

master进程充当整个进程组与用户的交互接口，同时对进程进行监护。它不需要处理网络事件，不负责业务的执行，只会通过管理worker进程来实现重启服务、平滑升级、更换日志文件、配置文件实时生效等功能。

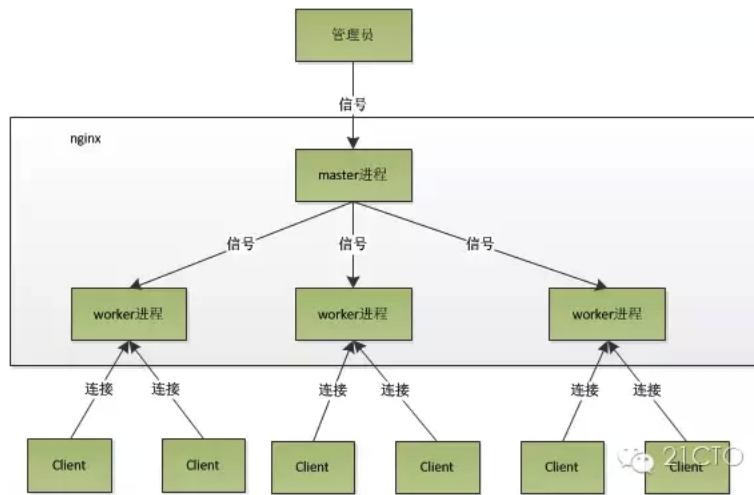
我们要控制nginx，只需要通过kill向master进程发送信号就行了。比如kill -HUP pid，则是告诉nginx，从容地重启nginx，我们一般用这个信号来重启nginx，或重新加载配置，因为是从容地重启，因此服务是不中断的。master进程在接收到HUP信号后是怎么做的呢？首先master进程在接到信号后，会先重新加载配置文件，然后再启动新的worker进程，并向所有老的worker进程发送信号，告诉他们可以光荣退休了。新的worker在启动后，就开始接收新的请求，而老的worker在收到来自master的信号后，就不再接收新的请求，并且在当前进程中的所有未处理完的请求处理完成后，再退出。当然，直接给master进程发送信号，这是比较老的操作方式，nginx在0.8版本之后，引入了一系列命令行参数，来方便我们管理。比如，./nginx -s reload，就是来重启nginx，./nginx -s stop，就是来停止nginx的运行。如何做到的呢？我们还是拿reload来说，我们看到，执行命令时，我们是启动一个新的nginx进程，而新的nginx进程在解析到reload参数后，就知道我们的目的是控制nginx来重新加载配置文件了，它会向master进程发送信号，然后接下来的动作，就和我们直接向master进程发送信号一样了。

worker进程：

而基本的网络事件，则是放在worker进程中来处理了。多个worker进程之间是对等的，他们同等竞争来自客户端的请求，各进程互相之间是独立的。一个请求，只可能在一个worker进程中处理，一个worker进程，不可能处理其它进程的请求。worker进程的个数是可以设置的，一般我们会设置与机器cpu核数一致，这里面的原因与nginx的进程模型以及事件处理模型是分不开的。

worker进程之间是平等的，每个进程，处理请求的机会也是一样的。当我们提供80端口的http服务时，一个连接请求过来，每个进程都有可能处理这个连接，怎么做到的呢？首先，每个worker进程都是从master进程fork过来，在master进程里面，先建立好需要listen的socket (listenfd) 之后，然后再fork出多个worker进程。所有worker进程的listenfd会在新连接到来时变得可读，为保证只有一个进程处理该连接，所有worker进程在注册listenfd读事件前抢accept_mutex，抢到互斥锁的那个进程注册listenfd读事件，在读事件里调用accept接受该连接。当一个worker进程在accept这个连接之后，就开始读取请求，解析请求，处理请求，产生数据后，再返回给客户端，最后才断开连接，这样一个完整的请求就是这样的了。我们可以看到，一个请求，完全由worker进程来处理，而且只在一个worker进程中处理。worker进程之间是平等的，每个进程，处理请求的机会也是一样的。当我们提供80端口的http服务时，一个连接请求过来，每个进程都有可能处理这个连接，怎么做到的呢？首先，每个worker进程都是从master进程fork过来，在master进程里面，先建立好需要listen的socket (listenfd) 之后，然后再fork出多个worker进程。所有worker进程的listenfd会在新连接到来时变得可读，为保证只有一个进程处理该连接，所有worker进程在注册listenfd读事件前抢accept_mutex，抢到互斥锁的那个进程注册listenfd读事件，在读事件里调用accept接受该连接。当一个worker进程在accept这个连接之后，就开始读取请求，解析请求，处理请求，产生数据后，再返回给客户端，最后才断开连接，这样一个完整的请求就是这样的了。我们可以看到，一个请求，完全由worker进程来处理，而且只在一个worker进程中处理。

nginx的进程模型，可以由下图来表示：



Nginx+FastCGI运行原理

1、什么是 FastCGI

FastCGI是一个可伸缩地、高速地在HTTP server和动态脚本语言间通信的接口。多数流行的HTTP server都支持FastCGI，包括Apache、Nginx和lighttpd等。同时，FastCGI也被许多脚本语言支持，其中就有PHP。

FastCGI是从CGI发展改进而来的。传统CGI接口方式的主要缺点是性能很差，因为每次HTTP服务器遇到动态程序时都需要重新启动脚本解析器来执行解析，然后将结果返回给HTTP服务器。这在处理高并发访问时几乎是不可用的。另外传统的CGI接口方式安全性也很差，现在已经很少使用了。

FastCGI接口方式采用C/S结构，可以将HTTP服务器和脚本解析服务器分开，同时在脚本解析服务器上启动一个或者多个脚本解析守护进程。当HTTP服务器每次遇到动态程序时，可以将其直接交付给FastCGI进程来执行，然后将得到的结果返回给浏览器。这种方式可以让HTTP服务器专一地处理静态请求或者将动态脚本服务器的结果返回给客户端，这在很大程度上提高了整个应用系统的性能。

2、Nginx+FastCGI运行原理

Nginx不支持对外部程序的直接调用或者解析，所有的外部程序（包括PHP）必须通过FastCGI接口来调用。FastCGI接口在Linux下是socket（这个socket可以是文件socket，也可以是ip socket）。

wrapper：为了调用CGI程序，还需要一个FastCGI的wrapper（wrapper可以理解为用于启动另一个程序的程序），这个wrapper绑定在某个固定socket上，如端口或者文件socket。当Nginx将CGI请求发送给这个socket的时候，通过FastCGI接口，wrapper接收到请求，然后Fork(派生)出一个新的线程，这个线程调用解释器或者外部程序处理脚本并读取返回数据；接着，wrapper再将返回的数据通过FastCGI接口，沿着固定的socket传递给Nginx；最后，Nginx将返回的数据（html页面或者图片）发送给客户端。这就是Nginx+FastCGI的整个运作过程，如图所示。

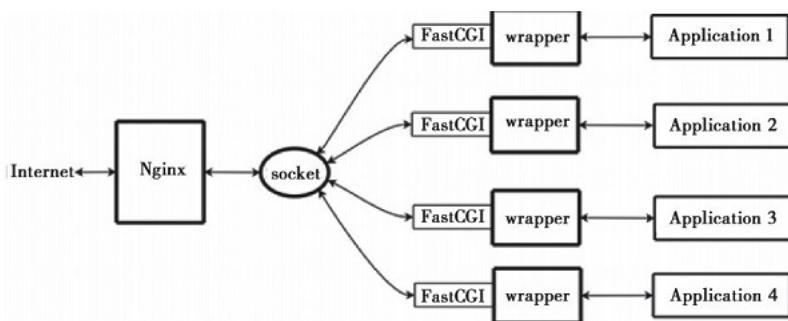


图 1-3 Nginx+FastCGI 运行过程

所以，我们首先需要一个wrapper，这个wrapper需要完成的工作：

通过调用fastcgi（库）的函数通过socket和nginx通信（读写socket是fastcgi内部实现的功能，对wrapper是非透明的）

调度thread，进行fork和kill

和application（php）进行通信

3、PHP-FPM

FastCGI接口方式在脚本解析服务器上启动一个或者多个守护进程对动态脚本进行解析，这些进程就是FastCGI进程管理器，或者称为FastCGI引擎。spawn-fcgi与PHP-FPM就是支持PHP的两个FastCGI进程管理器。因此HTTPServer完全解放出来，可以更好地进行响应和并发处理。

Nginx是个轻量级的HTTP server，必须借助第三方的FastCGI处理器才可以对PHP进行解析，因此其实这样看来nginx是非常灵活的，它可以和任何第三方提供解析的处理器实现连接从而实现对PHP的解析（在nginx.conf中很容易设置）。现在主要用PHP的FastCGI处理器PHP-FPM，它有如下优点：

由于它是作为PHP的patch补丁来开发的，安装的时候需要和php源码一起编译，也就是说编译到php core中了，因此在性能方面要优秀一些；

同时它在处理高并发方面也优于spawn-fcgi，至少不会自动重启fastcgi处理器。因此，推荐使用Nginx+PHP/PHP-FPM这个组合对PHP进行解析。

相对Spawn-FCGI，PHP-FPM在CPU和内存方面的控制都更胜一筹，而且前者很容易崩溃，必须用crontab进行监控，而PHP-FPM则没有这种烦恼。FastCGI的主要优点是把动态语言和HTTP Server分离开来，所以Nginx与PHP/PHP-FPM经常被部署在不同的服务器上，以分担前端Nginx服务器的压力，使Nginx专一处理静态请求和转发动态请求，而PHP/PHP-FPM服务器专一解析PHP动态请求。

4、Nginx+PHP-FPM

PHP-FPM是管理FastCGI的一个管理器，它作为PHP的插件存在，在安装PHP要想使用PHP-FPM时在老php的老版本（php5.3.3之前）就需要把PHP-FPM以补丁的形式安装到PHP中，而且PHP要与PHP-FPM版本一致，这是必须的）

PHP-FPM其实是PHP源代码的一个补丁，旨在将FastCGI进程管理整合进PHP包中。必须将它patch到你的PHP源代码中，在编译安装PHP后才可以使用。PHP5.3.3已经集成php-fpm了，不再是第三方的包了。PHP-FPM提供了更好的PHP进程管理方式，可以有效控制内存和进程、可以平滑重载PHP配置，比spawn-fcgi具有更多优点，所以被PHP官方收录了。在./configure的时候带--enable-fpm参数即可开启PHP-FPM。

fastcgi已经在php5.3.5的core中了，不必在configure时添加--enable-fastcgi了。老版本如php5.2的需要加此项。

当我们安装Nginx和PHP-FPM完后，配置信息：

PHP-FPM的默认配置php-fpm.conf：

```
1 listen_address 127.0.0.1:9000 #这个表示php的fastcgi进程监听的ip地址以及端口
2 start_servers
3 min_spare_servers
4 max_spare_servers
```

Nginx配置运行php：编辑nginx.conf加入如下语句：

```
1 location ~ \.php$ {
2     //指定了fastcgi进程侦听的端口，nginx就是通过这里与php交互的
3     root html; fastcgi_pass 127.0.0.1:9000;
4     fastcgi_index index.php;
5     include fastcgi_params;
```

```
6      fastcgi_param SCRIPT_FILENAME /usr/local/nginx/html$fastcgi_script_name;
7 }
```

Nginx通过location指令，将所有以php为后缀的文件都交给127.0.0.1:9000来处理，而这里的IP地址和端口就是FastCGI进程监听的IP地址和端口。

其整体工作流程：

- 1 1)、FastCGI进程管理器php-fpm自身初始化，启动主进程php-fpm和启动start_servers个CGI 子进程。
- 2 主进程php-fpm主要是管理fastcgi子进程，监听9000端口。fastcgi子进程等待来自Web Server的连接。
- 3 2)、当客户端请求到达Web Server Nginx时，Nginx通过location指令，将所有以php为后缀的文件都交给127.0.0.1:9000来处理，
- 4 即Nginx通过location指令，将所有以php为后缀的文件都交给127.0.0.1:9000来处理。
- 5 3) FastCGI进程管理器PHP-FPM选择并连接到一个子进程CGI解释器。Web server将CGI环境变量和标准输入发送到FastCGI子进程。
- 6 4)、FastCGI子进程完成处理后将标准输出和错误信息从同一连接返回Web Server。当FastCGI子进程关闭连接时，请求便告处理完成。
- 7 5)、FastCGI子进程接着等待并处理来自FastCGI进程管理器（运行在 WebServer中）的下一个连接。

nginx 配置文件说明

nginx.conf

```
1 user www www; # Nginx的worker进程运行用户以及用户组
2
3 worker_processes 4; # 启动进程数,通常设置成和cpu的数量相等
4 #worker_processes auto;
5
6 #以下参数指定了哪个cpu分配给哪个进程,一般来说不用特殊指定。如果一定要设的话,用0和
7 #1指定分配方式。
8 #这样设就是给1~4个进程分配单独的核来运行,出现第5个进程是就是随机分配了。
9 #worker_processes 4      #4核CPU
9 #worker_cpu_affinity 0001 0010 0100 1000
10
11 error_log /home/wwwlogs/nginx_error.log crit; # 定义全局错误日志定义类
    [debug|info|notice|warn|crit]
12
13 pid      /usr/local/nginx/logs/nginx.pid; # PID文件
14
15 #Specifies the value for maximum file descriptors that can be opened
    by this process.
16 # 一个nginx进程打开的最多文件描述符数目,理论值应该是最多打开文件数 (ulimit -n)
    与nginx进程数相除,
17 # 但是nginx分配请求并不是那么均匀,所以最好与ulimit -n的值保持一致。
18 worker_rlimit_nofile 51200;
19
20 # 工作模式及连接数上限
21 events
22 {
23     # use [ kqueue | rtsig | epoll | /dev/poll | select | poll
24 ];
25     #epoll模型是Linux 2.6以上版本内核中的高性能网络I/O模型,如果跑在FreeB
    SD上面,就用kqueue模型。
26     use epoll;
27     worker_connections 51200; # 单个后台worker process进程的最大并发
    连接数
28     #worker工作方式:串行(一定程度降低负载,但服务器吞吐量大时,关闭使用并行
    方式)
29     multi_accept on;
30 }
31 http
32 {
33     #文件扩展名与文件类型映射表
34     include mime.types;
35     #默认文件类型
36     default_type application/octet-stream;
37     # 服务器名字的hash表大小
38     server_names_hash_bucket_size 128;
39     #指定来自客户端请求头的headerbuffer大小
40     client_header_buffer_size 32k;
41     #指定客户端请求中较大的消息头的缓存最大数量和大小。
42     large_client_header_buffers 4 32k;
43     #客户端请求单个文件的最大字节数
44     client_max_body_size 50m;
45 }
```

```

46         sendfile    on; #开启高效传输模式。
47         #防止网络阻塞
48         tcp_nopush on;
49
50         keepalive_timeout 60; # 连接超时时间 , 单位是秒
51
52         tcp_nodelay on;
53
54         #FastCGI相关参数是为了改善网站的性能: 减少资源占用, 提高访问速度。
55         fastcgi_connect_timeout 300;
56         fastcgi_send_timeout 300;
57         fastcgi_read_timeout 300;
58         fastcgi_buffer_size 64k;
59         fastcgi_buffers 4 64k;
60         fastcgi_busy_buffers_size 128k;
61         fastcgi_temp_file_write_size 256k;
62
63         gzip on; # 开启gzip压缩
64         gzip_min_length 1k; #最小压缩文件大小
65         gzip_buffers 4 16k; #压缩缓冲区
66         gzip_http_version 1.1; #压缩版本 (默认1.1, 前端如果是squid2.5请使用
1.0)
67         gzip_comp_level 2; #压缩等级 1-9 等级越高, 压缩效果越好, 节约宽带, 但
CPU消耗大
68         gzip_types      text/plain application/javascript application/
n/x-javascript text/javascript text/css application/xml application/
xml+rss;
69         #压缩类型, 默认就已经包含text/html, 所以下面就不用再写了,
70         #写上去也不会有问题, 但是会有一个warn。
71         #前端缓存服务器缓存经过压缩的页面
72         gzip_vary on;
73         gzip_proxied   expired no-cache no-store private auth;
74         gzip_disable   "MSIE [1-6]\.";
75
76         #limit_conn_zone $binary_remote_addr zone=perip:10m;
77         ##If enable limit_conn_zone,add "limit_conn perip 10;" to se
rver section.
78
79         server_tokens off;
80         access_log off;
81         # 定义日志格式
82         log_format up_head '$remote_addr - $remote_user [$time_loca
l] "$request" '
83                         '$status $body_bytes_sent "$http_referer" '
84                         '"$http_user_agent" "$http_x_forwarded_for" "d
eviceid: $http_deviceid"'
85                         '"$request_body" $upstream_response_time
"$request_time"';
86
87
88         include vhost/*.conf;
89 }

```

vhost/test.conf

```

1 server {
2     listen 80;
3     listen 443 ssl;

```

```

4     server_name www.test.com;
5
6     ssl_certificate      /nginx/conf/https/www.test.com.crt;
7     ssl_certificate_key /nginx/conf/https/www.test.com.key;
8     root    /www/www.test.com;
9     index   index.html index.htm index.php;
10
11    location ~^/test.php {
12        default_type application/json;
13        return 404 '{404}';
14        access_log off;
15    }
16
17    location ~ ^/test/status(.*)$ {
18    }
19        default_type text/html;
20        access_log /log/nginx/access/dot.log;
21        return 200 '';
22    }
23
24    location / {
25        try_files $uri $uri/ /index.php?$query_string;
26    }
27    location ~\.\php$ {
28        # 如果用到 sock 则值参考 unix:/var/run/php/php7.0-fpm.sock
29        fastcgi_pass          unix:/tmp/php-cgi.sock;
30        fastcgi_index         index.php;
31        fastcgi_param         SCRIPT_FILENAME $document_root
32        fastcgi_script_name;
33        include               fastcgi_params;
34        access_log            /log/nginx/access/access.log up_head;
35    }
36
37    #静态文件
38    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$ {
39        expires      30d;
40    }
41
42    location ~ .*\.(js|css)?$ {
43    }
44        expires      12h;
45    }
46
47    location ~ /.well-known {
48        allow all;
49    }
50
51    location ~ /\.
52    {
53        deny all;
54    }
55
56    error_log /log/nginx/error/error.log;
57 }

```

定义日志格式参数说明：

```
1      #定义日志的格式。后面定义要输出的内容。
2      #1.$remote_addr 与$http_x_forwarded_for 用以记录客户端的ip地址;
3      #2.$remote_user : 用来记录客户端用户名;
4      #3.$time_local : 用来记录访问时间与时区;
5      #4.$request   : 用来记录请求的url与http协议;
6      #5.$status    : 用来记录请求状态;
7      #6.$body_bytes_sent : 记录发送给客户端文件主体内容大小;
8      #7.$http_referer : 用来记录从那个页面链接访问过来的;
9      #8.$http_user_agent : 记录客户端浏览器的相关信息
```

nginx 禁止ip直接访问

添加server

```
1 server
2 {
3     listen 80 default_server;
4     server_name _;
5     return 500;
6 }
7 或者
8 server {
9     listen 80 default_server;
10    server_name _;
11    rewrite ^(.*) http://www. exchangecn .net permanent;
12 }
```

去哪申请证书？

FreeSSL

一个申请免费HTTPS证书的网站

网址: <https://freessl.cn/>

git 忽略已经跟踪的文件

.gitignore

在git中要忽略某些文件，常用的就是在.gitignore 中进行修改，通常用在新建仓库时使用。

但文件一旦在版本库里已经存在，则.gitignore忽略文件失效；

即.gitignore 只能忽略从来没有加入到版本库中的文件。

git update-index --assume-unchanged

但是有时候会有这种需求，忽略已追踪文件的改动，例如，工程里已经有了a.xml，并且已经放到版本库，这时候我想修改下a.xml让其适配自己的环境（比如设置连自己的测试数据库），但是又不想不小心push到远程仓库。由于已经存在于版本库，因此.gitignore方式无效。

解决方法如下：

```
1 # 本地修改不提交到远程仓库
2
3 git update-index --assume-unchanged [file-path]
4
5 # 取消本地忽略
6
7 git update-index --no-assume-unchanged [file-path]
8
9 # 查看本地仓库哪些文件被加入忽略列表
10
11 git ls-files -v
```

git x分支强制覆盖master分支方法

1、删除本地master分支

2、将本地x分支名称改为master分支

3、强制推送本地master分支到远程

```
1 git push origin master --force
```

git删除未跟踪文件

git删除未跟踪文件

1、删除 untracked files

```
1 git clean -f
```

2、连 untracked 的目录也一起删掉

```
1 git clean -fd
```

3、连 .gitignore 中的 untrack 文件/目录也一起删掉 (慎用)

```
1 git clean -xfd
```

4、在用上述 git clean 前，墙裂建议加上 -n 参数来先看看会删掉哪些文件，防止重要文件被误删

```
1 git clean -nxfd  
2 git clean -nf  
3 git clean -nfd
```

Gerrit代码Review入门实战

代码审核（Code Review）是软件研发质量保障机制中非常重要的一环，但在实际项目执行过程中，却因为种种原因被Delay甚至是忽略。在实践中，给大家推荐一款免费、开放源代码的代码审查软件Gerrit。

1、Why Code Review

Code Review是什么

Code Review最直观的解释即看代码。常规的做法为自己看，有时代码逻辑问题可能自己看不出来，需要找同事一起看，在大家知识体系相对平均的情况下可能需要花钱专门的公司帮助查看。

Code Review需要看哪些？对于刚入职场或者刚接触到Coding的新人来说，代码风格是比较重要的一块。除此之外，编码规范及代码结构写法，框架和工具的选型，具体项目的业务逻辑，安全隐患，性能问题等都可以通过review的方式发现。Code Review从前往后大致分为结对编程，提交代码后，测试之前，发版之前，发版之后等几个阶段，越往后，Code Review的效果越差，修复的成本也越来越高。

为什么一定要做入库前Code Review

首先，代码审查的最大的功用是纯社会性的。如果你在编程，而且知道将会有同事检查你的代码，你编程态度就完全不一样了。你写出的代码将更加整洁，有更好的注释和程序结构。

其次，偷懒是人的天性，从节约成本的角度考虑，大家一般会选择在测试之前无限制的Delay Code Review。入库前做Code Review便是成本和效果之间最佳平衡点，它能及时发现问题，进行修改后确保代码质量。

最后，代码审查能传播知识。在很多开发团队里，经常每个人负责一个核心模块，每个人都只关注自己的模块。除非是同事的模块影响了自己的程序，他们从不相互交流。这种情况的后果是，每个模块只有一个人熟悉里面的代码。如果这个人体假或辞职了，其他人则束手无策。通过代码审查，至少会有两个人熟悉这些程序——作者，以及审查者。审查者并不能像程序的作者一样对程序十分了解，但至少他会熟悉程序的设计和架构，这是极其重要的。

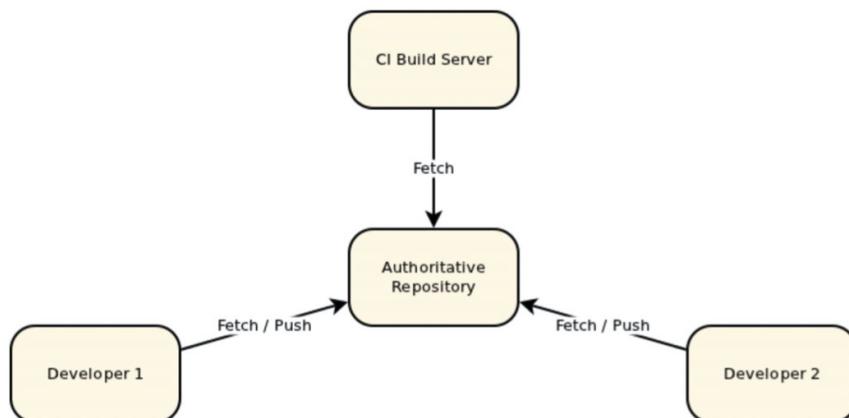
2、Gerrit（盖瑞特）简介

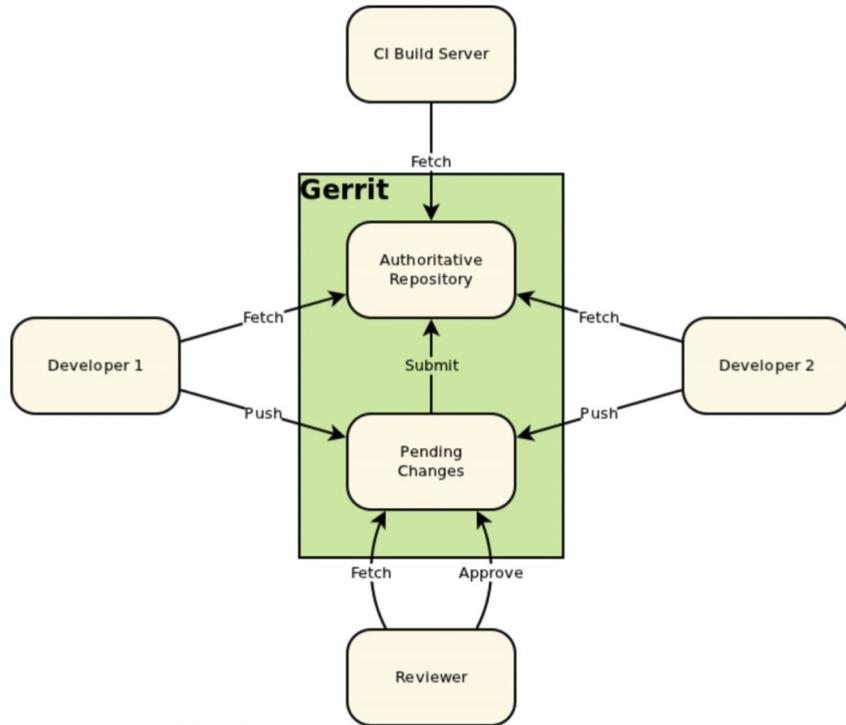
Gerrit，是一种开放[源代码](#)的基于web平台的代码评审工具，基于 Git 版本控制系统。它在传统的源码管理协作流程中强制性引入代码审核机制，通过人工代码审核和自动化代码验证过程，将不符合要求的代码屏蔽在代码库之外，确保核心代码多人校验、多人互备和自动化构建核验。

- | 作者是 Google 公司的 Shawn Pearce，最初是为了管理 Android 项目而产生的。
- | 它最初是由 Python 编写，在第二版后改用 Java 与 SQL。
- | 名字来源于一名荷兰设计师及建筑师 Gerrit Thomas Rietveld(赫里特·托马斯·里特费尔德)的名字。

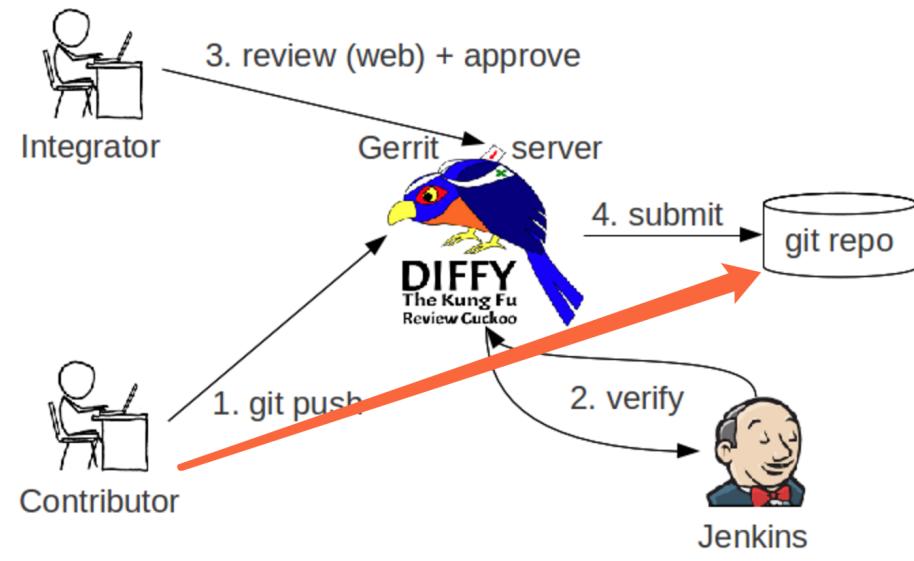
Gerrit工作流

git





或：



使用过git的同学，都知道，当我们git add --> git commit --> git push之后，你的代码会被直接提交到repo，也就是图中橘红色箭头指示的那样。

那么gerrit就是上图中的那只鸟，普通成员的代码是被先push到gerrit服务器上，然后由代码审核人员，就是左上角的integrator在web页面进行代码的审核(review)，可以单人审核，也可以邀请其他成员一同审核，当代码审核通过(approve)之后，这次代码才会被提交(submit)到代码仓库(repo)中去。

无论有新的代码提交待审核，代码审核通过或被拒绝，代码提交者(Contributor)和所有的相关代码审核人员(Integrator)都会收到邮件提醒。

gerrit还有自动测试的功能，和主线有冲突或者测试不通过的代码，是会被直接拒绝掉的，这个功能似乎就是右下角那个老头(Jenkins)的任务。

整个流程就是这样。在使用过程中，有两点需要特别注意下：

当进行commit时，必须要生成一个Change-ID，否则，push到gerrit服务器时，会收到一个错误提醒。
提交者不能直接把代码推到远程的master主线(或者其他远程分支)上去。这样就相当于越过了gerrit了。
gerrit必须依赖于一个refs/for/*的分支。

假如我们远程只有一个master主线，那么只有当你的代码被提交到refs/for/master分支时，gerrit才会知道，我收到了一个需要审核的代码推送，需要通知审核员来审核代码了。

当审核通过之后，gerrit会自动将这条分支合并到master主线上，然后邮件通知相关成员，master分支有更新，需要的成员再去pull就好了。而且这条refs/for/master分支，是透明的，也就是说普通成员其实是不需要知道这条线的，如果你正确配置了sourceTree，你也应该是看不到这条线的。

Gerrit适用性

几乎任何需要正式发布的项目都应当使用Gerrit来进行代码审查，如果Team中有新人，必须使用Gerrit确保代码质量。

Gerrit效果

The screenshot shows two main parts of the Gerrit interface:

- Search Results for 'status:open':** A table listing various code reviews. The columns include Subject, Status, Owner, Project, Branch, Updated, Size, CR, and V. The table shows multiple entries from different users (赵伟, liub, miaoyl, hujun) across various branches (develop, master, dev, branch-2.9.0-20160524_0937) and projects.
- Code Review for CrashStatController.java:** A detailed view of a specific code review. The code editor shows Java code for generating histograms. A green highlight covers several lines of aggregation logic, specifically the part where it builds a histogram for user counts and issue times. The code uses Elasticsearch's Aggregation framework.

整体上来说，个推使用的标准配置为Gerrit+Jenkins+Sonar，整个系统搭建完成后得到的效果为：100% Code Style问题避免入库，80% 设计问题避免入库，40% 逻辑错误避免入库，20% 安全隐患避免入库，100% 人员互备。

3、Gerrit入门实战

Gerrit部署和运行

JDK环境配置

java -jar gerrit-2.12.war init -d review_site

```

Generating SSH host key ... rsa(simple)... done

*** HTTP Daemon
*** 

Behind reverse proxy      [y/N]?
Use SSL (https://)          [y/N]?
Listen on address          [*]:
Listen on port              [8080]: 8381
Canonical URL              [http://getui:8381/]: http://121.199.40.127:8381/

*** Plugins
*** 

Installing plugins.
Install plugin singleusergroup version v2.12 [y/N]?
Install plugin commit-message-length-validator version v2.12 [y/N]?
Install plugin reviewnotes version v2.12 [y/N]?
Install plugin replication version v2.12 [y/N]?
Install plugin download-commands version v2.12 [y/N]?
Initializing plugins.
No plugins found with init steps.

Initialized /root/gerrit/review_site
Executing /root/gerrit/review_site/bin/gerrit.sh start
Starting Gerrit Code Review: OK
Waiting for server on 121.199.40.127:8381 ... OK
Opening http://121.199.40.127:8381/#/admin/projects/ ...FAILED
Open Gerrit with a JavaScript capable browser:
  http://121.199.40.127:8381/#/admin/projects/

```

Gerrit用户角色和权限

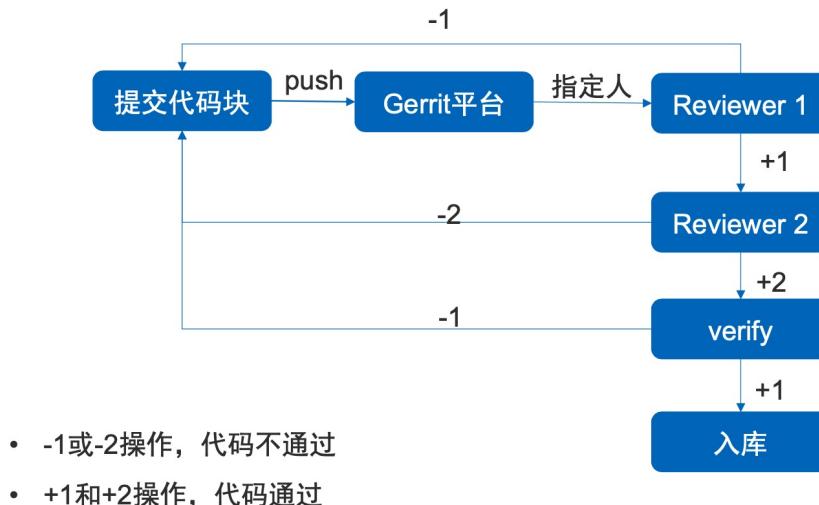
Gerrit用户角色全新可以由管理者进行配置。

Gerrit用户角色:

- 1 admin: 拥有最高权限
- 2 verify: 确认+1权限, 入库权限
- 3 reviewer: 代码评审权限, 包含+1/-1, +2/-2的权限
- 4 developer: push权限

第一个登录gerrit的网页的用户为admin管理员用户, 创建dev帐号、review帐号和verify帐号, 创建dev、review和verify用户组并添加相应用户。

Gerrit使用流程:



Gerrit相关配置

接下来还有很多内容需要设置一下。我们依次来看看吧。

profile

注意设置Username, 代码同步时需要用到。

初次登录时，Full Name 和 Email Address 字段都是空的，要设置一下，没有设置之前，右上角显示的用户名也是 admin

Preferences

Preferences 页面用于配置gerrit的web页面，我一般把时间格式改成习惯的24小时制，同时确保 Email Notification 处于开启状态，这个默认就是开启的；然后在 Show Change Number In Changes Tables 打上勾，这样能清楚地看到每个审核的编号，邮件里面显示的也是这个编号。

Watched Projects

这里有必要先说一下gerrit的两个默认项目：

我们点击左上方的菜单栏 Projects -> List，就能看到两个默认的项目 All-Projects 和 All-Users，这两个工程是两个基础的工程，我们新建的工程默认都是继承自 All-Projects 的权限。关于权限部分我们在后面的章节详细介绍。

所以在 Watched Projects 菜单中，就是当前用户要监听的项目，当这些项目发生变化时，你会收到邮件提醒，如果你选择了 All-Projects，那么就意味着你要监听所有的工程，因为所有工程都会默认继承自 All-Projects。

注意，后面的那些选项，勾选了某一项，就表示仅仅给当前项发送邮件提醒，保险期间，我们就全部勾上就好了。

Contact Information

The screenshot shows the Gerrit web interface under the 'Contact Information' tab. The left sidebar lists various settings: Profile, Preferences, Diff Preferences, Edit Preferences, Watched Projects, Contact Information (which is selected and highlighted with a red box), SSH Public Keys, HTTP Password, Identities, and Groups. The main content area shows fields for Username (admin), Full Name (xwlpeng), and Preferred Email (lpeng@microwu.com). A red box highlights the 'Register New Email ...' link. Below these fields is a 'Save Changes' button. At the bottom right of the main area, it says 'Powered by Gerrit Code Review (2.13.5) | Press "?" to view keyboard shortcuts'.

这里是当前账号的配置，你可以在这里把full-name填写完全，这样右上角也会同步更新你刚设置好的名称。

注意这里的Preferred Email有两种设置方法：

1. 如果你的邮件服务器配置完成了，可以点击Register New Email，你就会收到一封确认邮件，确认之后就能设置好了。

2. 通过SSH在命令行中进行配置，通过命令行进行配置是最方便快捷，也是最优先推荐的方法。

不过因为SSH命令行的方式需要配置SSH公钥，所以我们这里先留空，一会通过命令行来配置。

SSH Public Keys

The screenshot shows the Gerrit web interface under the 'SSH Public Keys' tab. The left sidebar lists: Profile, Preferences, Diff Preferences, Edit Preferences, Watched Projects, Contact Information (selected and highlighted with a red box), SSH Public Keys (highlighted with a red box), HTTP Password, Identities, and Groups. The main content area has a 'Status Algorithm Key Comment' table with one row: ssh-rsa AAAA3NzaC1yc2EAAAQABAAAQ...fBPCyp8RvK. Below this is a 'Add SSH Public Key' section with a 'How to Generate an SSH Key' link. A large red box highlights this entire section. At the bottom are 'Clear', 'Add', and 'Close' buttons. A 'Server Host Key' section is also visible at the bottom.

这里需要把你的公钥内容拷贝出来，然后粘贴到对话框中。

Groups

The screenshot shows the Gerrit web interface under the 'Groups' tab. The left sidebar lists: Profile, Preferences, Diff Preferences, Edit Preferences, Watched Projects, Contact Information, SSH Public Keys, HTTP Password, Identities, and Groups (selected and highlighted with a red box). The main content area shows a table with columns: Group Name, Description, and Visible To All. It contains three rows: Administrators (Gerrit Site Administrators), Anonymous Users, and Registered Users. A red box highlights the 'Administrators' row. At the bottom right, it says 'Powered by Gerrit Code Review (2.13.5) | Press "?" to view keyboard shortcuts'.

最后来看一下gerrit的分组。图片里面是gerrit默认的几个分组，我们需要知道的是 Administrator 就是管理员分组， Anonymous Users 指的是所有添加到gerrit数据库中的成员都默认加入的一个组。之后我们还可以建立新的分组，加入新的成员等等。

示例：

接下来我们来做一个演示，看看一个新的成员是如何被添加到gerrit服务器中，然后他们又是如何协同工作的。

这里一共涉及到两个角色，一个是管理员，一个是普通成员。

管理员设置SSH

在之前的文中我们提到过，gerrit自带的H2数据库就完全够用了，对成员的管理，邮件添加等操作，均可以通过SSH来完成。那第一步我们就来看一下管理员如何才能远程SSH到gerrit服务器。

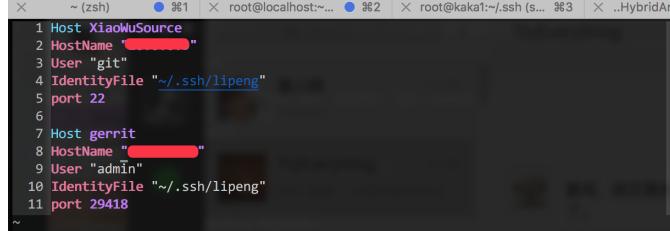
首先确保在之前，已经成功把你的公钥添加到了web页面账户中。

接下来，需要修改之前`~/.ssh/`文件夹下面的config文件，我们拿我的config文件做为示例，做个讲解。

我们还是先进入到`~/.ssh/`文件夹中

```
{17-01-18 11:27}MichaelLees-MacBook-Pro:~/ssh KaKa% ll  
total 40  
-rw-r--r-- 1 KaKa staff 173B Jan 13 14:55 config  
-rw-r--r-- 1 KaKa staff 4.8K Jan 16 15:42 known_hosts  
-r----- 1 KaKa staff 1.6K Jun 8 2016 lipeng  
-r----- 1 KaKa staff 402B Jan 13 14:33 lipeng.pub  
{17-01-18 11:27}MichaelLees-MacBook-Pro:~/ssh KaKa% vim lipeng.pub  
{17-01-18 13:32}MichaelLees-MacBook-Pro:~/ssh KaKa% vim config
```

然后查看一下config文件：`vim config`



```
1 Host XiaoWuSource  
2 HostName "████████"  
3 User "git"  
4 IdentityFile "~/.ssh/lipeng"  
5 port 22  
6  
7 Host gerrit  
8 HostName "████████"  
9 User "admin"  
10 IdentityFile "~/.ssh/lipeng"  
11 port 29418  
~
```

我们看到这里面有两个Host部分，我们重点来看第2个Host部分，这个是我们新建的，用于连接到gerrit服务器的配置。

照猫画虎，对于我们之前建立在`192.168.1.100`的gerrit服务器来说，你的Host配置可能如下：

1	Host gerrit
2	HostName "192.168.1.100"
3	User "admin"
4	IdentityFile "~/.ssh/id_rsa"
5	port 29418

User要和我们在gerrit服务器上注册的名称保持一致(不是full-name)，认证文件注意要和公钥对应的私钥文件，端口要填写gerrit服务的端口号，这里是默认的`29418`。

配置完config文件，我们就可以SSH到gerrit了，我们来尝试一下吧：

```
[17-01-18 13:52}MichaelLees-MacBook-Pro:~/ssh KaKa% ssh gerrit -l admin  
**** Welcome to Gerrit Code Review ****  
Hi xw.lipeng, you have successfully connected over SSH.  
Unfortunately, interactive shells are disabled.  
To clone a hosted Git repository, use:  
git clone ssh://admin@████████:29418/REPOSITORY_NAME.git  
Connection to ██████████ closed.
```

我们在管理员的机器上，输入`ssh gerrit -l admin`命令，就可以得到gerrit服务器的响应，只不过因为我们禁用了shell，所以连接很快断开了，没关系，这样证明做为管理员，已经可以通过命令行对gerrit服务器进行一系列的操作了。

添加普通成员

在管理员添加新的组员之前，我们需要先在普通成员的机器上生成ssh的公私钥，这里方便描述，我们把这个普通成员命名为test3。

在test3的电脑命令行中，生成利用`ssh-keygen`命令，生成公私钥。

```
[root@kaka1 ~]# cd ~/.ssh  
[root@kaka1 .ssh]# ll  
total 12  
-rw----- 1 root root 1675 Jan 16 09:02 id_rsa  
-rw-r--r-- 1 root root 392 Jan 16 09:02 id_rsa.pub  
-rw-r--r-- 1 root root 231 Jan 16 16:31 known_hosts  
[root@kaka1 .ssh]#
```

我们就使用默认的`id_rsa`命名好了。

接下来，test3成员需要把`id_rsa.pub`公钥发送给管理员，这样管理员才能正常把test3添加到gerrit用户组中。

我们假设管理员将test3的pub公钥放到了`~/home`目录下，也就是说，在管理员的电脑上，test3的公钥存放在`~/home/id_rsa.pub`文件，当然我们也可以重新把它命名为`test3.pub`，方便演示我这里就不做更名处理了。

接下来，管理员在命令行中输入如下的命令来完成添加普通成员的操作。注意：这个命令很强大很方便，可以一步到位地把成员的的名称，全名，邮箱以及ssh公钥认证全部设置好。

	\$ cat ~/home/id_rsa.pub ssh gerrit gerrit create-account --full-name test3
--	---

接下来我们来详细看一下这个命令：

- 符号把这个命令分成了两部分，第一部分的 `cat ~/home/id_rsa.pub` 表示把test3的公钥内容读入到输入流中
- `ssh gerrit`是我们之前在 `~/.ssh/config` 中配置好的gerrit服务器地址
- 又接着一个gerrit表示通过ssh中输入gerrit命令来进行相关操作
- `create-account` 表示要新建用户。注意，新建的用户名写在最后面，中间是其他参数
- `full-name` 就如同页面中的全名，我们这里命名为test3
- `email`表示该用户的email地址，我们填入 `test3@microwu.com`
- `ssh-key` – 注意，最后的 `-` 表示从输入流中读取ssh的公钥内容，也就是 `l` 符号之前我们读入的 test3用户公钥内容
- 最后面加上我们要create的用户名

这个命令执行完之后，管理员就把test3用户加入到了gerrit用户组中，并且设置了他的全称，邮件以及公钥文件，是不是一步到位，非常方便？？

这里我们回过头来，在管理员首次登陆web页面进行修改配置的时候，我们说过，管理员的邮箱可以通过命令行来设置，是的，同样通过ssh命令行：

```
1 $ ssh gerrit gerrit set-account --add-email admin@microwu.com admin
```

这个命令就表示为我们的 `admin` 用户添加email `admin@microwu.com`。执行完这个命令，再回到web界面上的用户设置界面，看看是不是管理员的email已经被设置好了？？

修改用户所在组

接下来我们看一下怎样修改test3用户所在的组吧。我们知道他已经处在 `Anonymous Users` 组中了，那我们想要新建一个组，就叫 `test_user` 吧，我们来看一下

Group Name	Description	Visible To All
Administrators	Gerrit Site Administrators	
Non-interactive Users	Users who perform batch actions on Gerrit	

我们在gerrit页面的顶部，点击 `People` → `list`，看一下默认的两个分组，`Administrator` 和 `Non-interactive Users`，这两个分组我们都能从字面上理解是什么意思。我们注意到 `Anonymous Users` 这个分组并没有显示在页面，因为它是匿名的嘛，所有的用户自动添加到这个分组中了。

选择 `Create New Group`，输入我们要添加的新分组 `test_user`

新的分组中，我们看到管理员的账号被自动添加了进来

我们在 `Add` 搜索栏中输入 `test`，就会自动显示出来管理员之前在命令行中创建的test3用户，看到 `full-name` 和 `email` 了吧，都已经添加完成了！

test3用户已经添加到了test_user分组中了。

创建第一个项目，配置权限管理

添加project，选择 `Inherit From All-Projects`，当然也可以自定义Parent Project。

The screenshot shows the Gitea interface with the 'Projects' tab selected. On the left, there's a 'Create Project' form with fields for 'Project Name' (set to 'getui-test'), 'Rights Inherit From' (set to 'All-Projects'), and a 'Browse' button. Below these are 'Parent Suggestion' and 'Project Name' fields, both set to 'All-Projects'. To the right is a sidebar table titled 'All projects' with a single row for 'All-Projects'.

添加Verified标签支持，这里修改All-Project项目的project.config，所有继承自All-Project的项目自动添加Verified 标签，也可针对项目自定义是否verify。

The screenshot shows the 'Project All-Projects' configuration page. Under 'Project Options', there are several settings like 'State' (Active), 'Submit Type' (Merge If Necessary), and 'Automatically resolve conflicts' (TRUE). On the right, a code editor displays a configuration file snippet for 'project.config' under the 'All-Projects' project:

```

0 *      value = +2 Looks good to me, approved
1 [*label "Verified"]
2 *      function = MaxWithBlock
3 *      value = -1 Fails
4 *      value = 0 No score
5 *      value = +1 Verified
6 *      defaultValue = 0

```

创建用户组

The screenshot shows the 'People' tab selected. A new group named 'gexintest-dev' is being created. In the 'Members' section, there is a table with columns 'Member' and 'Email Address'. Several user icons are listed, each with a checkbox next to it, indicating they can be added to the group. An 'Add' button is visible at the top right of the table.

添加相关用户权限

Project gixin-test

Edit

Rights Inherit From: All-Projects

History: (gitweb)

Reference: refs/*

Owner	ALLOW <input type="radio"/> project-verifier	<input type="checkbox"/> Exclusive
Read	ALLOW <input type="radio"/> Non-Interactive Users ALLOW <input type="radio"/> gexintest-dev ALLOW <input type="radio"/> gexintest-review ALLOW <input type="radio"/> gexintest-verify	<input type="checkbox"/> Exclusive
Abandon	ALLOW <input type="radio"/> gexintest-review	<input type="checkbox"/> Exclusive
Label Code-Review	-2 <input type="radio"/> +2 <input type="radio"/> gexintest-review	review 用户组 <input type="checkbox"/> Exclusive
Label Verified	-1 <input type="radio"/> +1 <input type="radio"/> Non-Interactive Users -1 <input type="radio"/> +1 <input type="radio"/> project-verifier	verify 用户组, 这里配置jenkins自动化用户 <input type="checkbox"/> Exclusive
Submit	ALLOW <input type="radio"/> gexintest-review	<input type="checkbox"/> Exclusive

Reference: refs/for/*

Push	ALLOW <input type="radio"/> gexintest-dev	develop 提交的for分支权限 <input type="checkbox"/> Exclusive
------	---	---

将代码库同步到本地 (SSH/Http)

HTTP 方式:

HTTP Password 密码在 账户 - -> Settings --> HTTP Password 处获取。

All My Projects People Plugins Documentation

List General Branches Access Dashboards Create New Project

Settings

- Profile
- Preferences
- Watched Projects
- Contact Information
- SSH Public Keys
- HTTP Password**
- Identities
- Groups

Username: lvgx

Password:

Generate Password | Clear Password

SSH方式:

添加SSH Public Key。

All My Projects People Plugins Documentation

List General Branches Access Dashboards Create New Project

Settings

- Profile
- Preferences
- Watched Projects
- Contact Information
- SSH Public Keys**
- HTTP Password
- Identities
- Groups

Status	Algorithm	Key	Comment
<input type="checkbox"/>	ssh-rsa	AAAAB3NzaC1yc2EAAAQABAAQ...rCmy1Lyw==	lvx@getui.com

Server Host Key

Fingerprint: SHA256:...
Ssh-Host-Name: ...

Entries:
Environnement hosts: ...
Environnement hosts: ...
Environnement hosts: ...

Clone代码到本地

All My **Projects** People Plugins Documentation

List General Branches Access Dashboards Create New Project

提供http和ssh两种clone方式,自行测试

Project gixin-test

务必选中clone with commit-msg hook

clone | clone with commit-msg hook | Anonymous HTTP | SSH | HTTP |

```
git clone http://lvgx@192.168.14.217:8281/gixin-test && scp -p -P 29418 lvgx@192.168.14.217:hooks/cc
```

Description

```
xz@xzdeMacBook-Pro:~$ git clone ssh://lvgx@192.168.14.217:29418/gixin-test &&
scp -p -P 29418 lvgx@192.168.14.217:hooks/commit-msg gixin-test/.git/hooks/
Cloning into 'gixin-test'...
remote: Counting objects: 69, done
remote: Finding sources: 100% (69/69)
remote: Total 69 (delta 14), reused 64 (delta 14)
Receiving objects: 100% (69/69), 7.51 KiB | 0 bytes/s, done.
Resolving deltas: 100% (14/14), done.
Checking connectivity... done.
commit-msg
100% 4362 4.3KB/s 00:00
```

git clone 后面的scp 是hook钩子,对git push提交进行监听, 监听到后进行拦截操作, 拦截后进行后续审核。

commit-msg ,提供自动写入change-id 至git log内功能

提交第一个change

All My Projects People Plugins Documentation

status:open

Search for status:open

Subject	Status	Owner	Project	Branch	Updated	Size	CR	V
update:gerrit test	Open	lgx	gixin-test	master	11:24 AM			

Subject: update:gerrit test
Status: Open
Owner: lgx
Project: gixin-test
Branch: master
Updated: 11:24 AM
Size: 0
CR: 0
V: 0

Code-Review: 2
Owner: lgx
Reviewers: fe-d
Project: gixin-test
Branch: master
Topic:
Strategy: Merge If Necessary
Updated: 2 minutes ago
Cherry Pick: Release: Abandon: Follow-Up

Code-Review: Verified +1 fe-d

File Path: A.xz_test
Comments Size: 1
+1 fe-d

```
xz@xzdeMacBook-Pro:~/gixin-test$ touch A.xz_test
xz@xzdeMacBook-Pro:~/gixin-test$ vi A.xz_test
xz@xzdeMacBook-Pro:~/gixin-test$ git add .
xz@xzdeMacBook-Pro:~/gixin-test$ git ci -m "update:gerrit test"
[origin/master] update:gerrit test
 1 file changed, 1 insertion(+)
 create mode 100644 A.xz_test
xz@xzdeMacBook-Pro:~/gixin-test$ git push origin HEAD:refs/for/master
Counting objects: 1, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 314 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1)
remote: Processing changes: new 1, refs: 1, done
remote:
remote: New Changes:
remote:   http://192.168.14.217:8281/3892 update:gerrit test
remote: /> ssh://lvgx@192.168.14.217:29418/gixin-test
remote:   http://lvgx@192.168.14.217:8281/3892 update:gerrit test
remote: /> ssh://lvgx@192.168.14.217:29418/gixin-test
```

Gerrit上进行代码审查, 确认入库

Verify:

工程里面接入了jenkins自动verify, 结果可在上图红框内展示verify结果。
review代码, 提交入库。

gixin-test / A.xz_test Patch Set 100644

Patch Set 1

Reply... **Code-Review: 2**

Owner: lgx
Reviewers: fe-d
Project: gixin-test
Branch: master
Topic:
Strategy: Merge If Necessary
Updated: 6 minutes ago
Cherry Pick: Release: Abandon: Follow-Up

Code-Review: Verified +1 fe-d

Submit

Code-Review: 2
Owner: lgx
Reviewers: fe-d
Project: gixin-test
Branch: master
Topic:
Strategy: Merge If Necessary
Updated: 17 seconds ago
Cherry Pick: Release: Abandon: Follow-Up

Code-Review: +2 lgx
Verified +1 fe-d

本地代码库更新, 获取最新入库代码

代码submit后通过git pull - - rebase 更新代码。

```

xz@xzdeMacBook-Pro:~/gixin-test$ git st
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
xz@xzdeMacBook-Pro:~/gixin-test$ git ls -2
a8d736e 2016-06-27 (HEAD -> master) update:gerrit test [lvgx]
080ea4d 2016-06-12 (origin/master) update:gerrit test 2 [daizi]
xz@xzdeMacBook-Pro:~/gixin-test$ git pull --rebase
From ssh://192.168.14.217:29418/gixin-test
  080ea4d..a8d736e master      -> origin/master
Current branch master is up to date.
xz@xzdeMacBook-Pro:~/gixin-test$ git st
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean

```

Gerrit入门实战-初级修补

如果所有代码提交均被打回，可以进行暴力回滚：git reset <commit>，接着重新提交Gerrit，再进行Gerrit审查入库。

Subject	Status	Owner	Project	Branch	Updated	Size	CR	V
update:reset test again	Open	lvgx	gixin-test	master	11:40 AM		<input checked="" type="checkbox"/>	<input type="checkbox"/>
update:reset test	Open	lvgx	gixin-test	master	11:38 AM		<input type="checkbox"/>	<input checked="" type="checkbox"/>

Reply...
 Owner lvgx
 Reviewers fe-ci ✘ lvgx ✘ Add...
 Project gixin-test
 Branch master
 Topic
 Strategy Merge if Necessary
 Updated 7 minutes ago
 Cherry Pick Rebase Abandon
 Follow-Up
 采用reset方式，需abandon本次commit
 Code-Review -2 lvgx
 Verified +1 fe-ci

Gerrit入门实战-高级修补

如果单个提交打回，则可交互式回滚：git rebase -i <commit>，修改指定commit点：git commit --amend，完成所有commit点处理：git rebase --continue，然后重新提交Gerrit，最后Gerrit审查入库。

Rebase前

Subject	Status	Owner	Project	Branch	Updated	Size	CR	V
update:rebase test	Open	lvgx	gixin-test	master	11:49 AM		<input checked="" type="checkbox"/>	<input type="checkbox"/>

Rebase后

Subject	Status	Owner	Project	Branch	Updated	Size	CR	V
update:rebase test, commit again	Open	lvgx	gixin-test	master	11:51 AM		<input checked="" type="checkbox"/>	<input type="checkbox"/>

rebase 在同一个点上修改，不会产生审核点，多个commit点同时存在是尤其有用。

```

xz@xzdeMacBook-Pro:~/gexin-test$ git rebase -i HEAD~2
Stopped at 88dd9da673487bf7d3... update:rebase test
You can amend the commit now, with
  git commit --amend
Once you are satisfied with your changes, run
  git rebase --continue
xz@xzdeMacBook-Pro:~/gexin-test$ git commit --amend
[detached HEAD 133045e] update: rebase test, commit again
Date: Mon Jun 27 11:49:39 2016 +0800
1 file changed, 2 insertions(+)
xz@xzdeMacBook-Pro:~/gexin-test$ git st
interactive rebase in progress; onto 88dd9da673487bf7d3
Last command done (2 commands done):
  pick 40d7979 update:reset test again
  edit 88dd9da673487bf7d3--rebase test
No commands remaining.
You are currently editing a commit while rebasing branch 'master' on 'a8d736e'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

nothing to commit, working directory clean
xz@xzdeMacBook-Pro:~/gexin-test$ git rebase --continue
Successfully rebased and updated refs/for/master.
xz@xzdeMacBook-Pro:~/gexin-test$ git push origin HEAD:refs/for/master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 334 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1)
remote: Processing changes: updated: 1, refs: 1, done
remote: (W) 133045e: no files changed, message updated
remote:
remote: Updated Changes:
remote: http://192.168.14.217:8281/3099 update:rebase test , commit again
remote:
To ssh://lvgx@192.168.14.217:29418/gexin-test
 * [new branch]      HEAD -> refs/for/master

```

Gerrit经验谈

第一，Git别名绑定，添加别名字段，通过git review master这样简单语法提交到master源端分支，可以省去很多工作。修改系统目录或者项目目下的.gitconfig文件，添加

```

[alias]
review = !sh -c 'git push origin HEAD:refs/for/$1' -

```

也可通过git config --global alias.review 命令修改

```

[xz@xzdeMacBook-Pro:~/gexin-test$ git push origin HEAD:refs/for/master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 334 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1)
remote: Processing changes: updated: 1, refs: 1, done
remote: (W) 133045e: no files changed, message updated
remote:
remote: Updated Changes:
remote: http://192.168.14.217:8281/3099 update:rebase test , commit again
remote:
To ssh://lvgx@192.168.14.217:29418/gexin-test
 * [new branch]      HEAD -> refs/for/master
[xz@xzdeMacBook-Pro:~/gexin-test$ git review master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 336 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1)
remote: Processing changes: updated: 1, refs: 1, done
remote: (W) 0e349fa: no files changed, message updated
remote:
remote: Updated Changes:
remote: http://192.168.14.217:8281/3099 update:rebase test , commit again,3
remote:
To ssh://lvgx@192.168.14.217:29418/gexin-test
 * [new branch]      HEAD -> refs/for/master

```

第二，工具只是一部分，更重要的是人与人当面的沟通交流，大家讨论一个好的解决方案，才能更好的解决问题。没有交流，工具也就失去了意义。

最后，关于review积压问题，要避免提交积压，代码审核过程要及时完成，避免 Code Review流于形式。从个推实际使用效果看，Gerrit在核心代码质量控制、知识传承、团队培养等方面都具备很高的实用价值，推荐给广大开发团队用。

git 连接远程仓库方法

方案一：本地创建项目根目录，然后与远程Git关联，之后的操作一样：

```
1 #创建新文件夹
2 mkdir xxx
3 #进入
4 cd xxx
5 #初始化Git仓库
6 git init
7 #提交改变到缓存
8 git commit -m 'description'
9 #本地git仓库关联GitHub仓库
10 git remote add origin git@github.com:han1202012/TabHost_Test.git
11 #提交到GitHub中
12 git push -u origin master
```

方案二：方案二就是不用关联Git仓库，直接从Git中克隆源码到本地，项目根目录也不用创建；

```
1 #从GitHub上克隆项目到本地
2 git clone git@github.com:han1202012/NDKHelloWorld.git #注意克隆的时候直接
   在仓库根目录即可，不用再创建项目根目录 ；
3 #添加文件
4 git add ./* # 将目录中所有文件添加;
5 #提交缓存
6 git commit -m '提交';
7 #提交到远程GitHub仓库
8 git push -u origin master
```

git修改远程仓库地址

方法有三种：

修改命令

```
1 git remote set-url origin [要改成的git url]
```

先删后加

```
1 git remote rm origin  
2 git remote add origin [要改成的git url]
```

直接修改config文件

```
1 vim .git/config  
2  
3 # 修改config文件中git远程仓库地址  
4  
5 [remote "origin"]  
6     url = [要改成的git url]
```

将其他分支文件或提交合并到当前分支

将其他分支文件合并到当前分支

如将dev分支x.php合并到master分支为例

```
1 1、将分支切换到master分支  
2 git checkout master  
3  
4 2、合并文件，将dev分支上 x.php文件追加补丁到dev分支上 x.php文件。你可以接受或者拒绝补丁内容。  
5 git checkout --patch dev x.php
```

将其他分支提交合并到当前分支

如将dev分支某次提交合并到master分支为例

```
1 1、将分支切换到master分支  
2 git checkout master  
3  
4 3、查看要合并的提交的commitID  
5 git log  
6  
7 2、合并提交,如要合并的commitID: 7fcb3defff  
8 git cherry-pick 7fcb3defff  
9  
10 4、推送到master远程  
11 git push
```

git分支管理策略

个人在项目中使用git分支管理策略介绍

主分支Master

首先，代码库应该有一个、且仅有一个主分支Master。项目的正式版本，都在这个主分支上发布。它是自动建立的，版本库初始化以后，默认就是在Master分支进行开发。

功能开发分支 feature/*

临时性分支

功能分支，它是为了开发某种特定功能，从master分支上面分出来的。开发完成后，并入release；

功能分支的名字，可以采用feature/w_*的形式命名。即 feature/开发者标识_开发的功能

- 1 基于master分支创建一个功能分支：
- 2 开发完成后，将功能分支合并到release分支：
- 3 删除feature分支：

版本预发布分支 release/*

永久分支，命名 release/版本号

预发布分支，它是指发布正式版本之前（即合并到Master分支之前），我们可能需要有一个预发布的版本进行测试。

预发布分支最初是从master分支上面分出来的，将开发完成的feature分支合并到release分支进行测试，测试完成后将release分支合并到master分支进行发布

- 1 创建一个预发布分支：
- 2 将各feature分支合并到release分支
- 3 对release分支进行测试
- 4 确认没有问题后，合并到master分支：

bug修复分支 fixbug/*

临时性分支，命名 fixbug/开发者标识_修改的功能，

软件正式发布以后，难免会出现bug。这时就需要创建一个分支，进行bug修补。

修补bug分支是从Master分支上面分出来的。修补结束以后，再合并进Master和release分支。它的命名，可以采用fixbug/*的形式。

- 1 基于master创建一个修补bug分支：
- 2
- 3 修补结束后，合并到master分支：
- 4
- 5 再合并到release分支：

6

7 最后，删除“修补bug分支”：

一台电脑上不同的Git仓库账号使用不同的公私钥设置

使用场景：

如我有多个git仓库账号，如2个bitbucket账号，一个github账号。我想让不同的git仓库账号使用不同的公私钥对，同时若多个bitbucket或多个github账号配置同一个公钥是不允许的，一个公钥只能配置在一个github或bitbucket账号中。要想解决如上问题，解决方法如下：

解决思路

生成多对私钥/公钥，注意密钥文件命名避免重复
git操作时，可以区分两个账户，推送到相应的仓库
设置不同的Host对应同一个HostName，但密钥不同
取消Git全局用户名/邮箱设置，为每个仓库单独设置用户名/邮箱

操作

1、创建多对公私钥

注意多对公私钥创建时，`-f` 和 `-C` 参数要不同。

创建方法如下：

```
1 ssh-keygen -t rsa -f ~/.ssh/id_rsa -C "xxx.yyyy@gmail.com"
2 解释几个参数：
3
4 -t 指定密钥类型，默认是 rsa，可以省略
5 -f 指定密钥文件存储文件名（注：~/.ssh/是密钥目录；id_rsa是密钥名，密钥名可以随意）
6 -C 设置注释文字，比如邮箱（1、这里的C是大写的 2、一定要关联你自己的GitHub的注册邮箱）
7 接着它会提示你输入两次密码（该密码是你push文件的时候要输入的密码，而不是GitHub管理者的密码），
8 你也可以不输入密码（推荐），直接按回车。那么push的时候就不需要输入密码，可以直接提交到GitHub上：
```

2、更改本地的SSH配置

用于区分账号，设置不同秘钥

在`~/.ssh`文件夹下新建config文件并编辑，配置Host（自定义账号域名）和HostName（实际git账号域名）、秘钥对应关系。

如果我们有一个bitbucket账号，一个github账号，设置方法如下

```
1 vim ~/.ssh/config
2
3 #第一个ssh密钥
4 Host github.com
5 HostName github.com
```

```
6 PreferredAuthentications publickey
7 User git
8 IdentityFile ~/.ssh/id_rsa_github
9 IdentitiesOnly yes
10
11 #第二个ssh密钥
12 Host bitbucket.org
13 HostName bitbucket.org
14 PreferredAuthentications publickey
15 User git
16 IdentityFile ~/.ssh/id_rsa_bitbucket
17 IdentitiesOnly yes
```

如果我们有两个bitbucket账号，设置方法如下

```
1 vim ~/.ssh/config
2
3 #第一个ssh密钥
4 Host git1.bitbucket.org
5 HostName bitbucket.org
6 PreferredAuthentications publickey
7 User git
8 IdentityFile ~/.ssh/id_rsa_bitbucket_1
9 IdentitiesOnly yes
10
11 #第二个ssh密钥
12 Host git2.bitbucket.org
13 HostName bitbucket.org
14 PreferredAuthentications publickey
15 User git
16 IdentityFile ~/.ssh/id_rsa_bitbucket_2
17 IdentitiesOnly yes
```

上面git1、git2 为自定义Host前缀。

以上配置文件参数说明：

```
1 # Host: 对识别的模式，进行配置对应的的主机名和ssh文件
2 # HostName: 登录主机的主机名
3 # PreferredAuthentications: 设置登录方式，publickey公钥，改成password则要输入密码
4 # IdentityFile: 私钥全路径名
5 # User: 登录名
6 # IdentitiesOnly 这个配置yes，表示只使用这里的key，防止使用默认的
```

3、将两个密钥对中的公钥分别加到两个Git账号配置下

4、克隆新的项目

以两个bitbucket账号为例：

一般情况下，我们是通过如下的方式克隆一个项目：

```
git clone git@bitbucket.org:your-account/your-prj.git
```

在我们有两个密钥的情况下（使用非默认密钥时），我们需要对这个语句中的域名部分做一下修改：

```
git clone git@xxx.bitbucket.org/your-prj.git
```

注意：这里的xxx是你更改本地的SSH配置第二步中的Host，比如的：

```
git clone git@git2.bitbucket.org:your-account/your-prj.git
```

这样就可以clone成功了。

若是一个github账号，一个bitbucket账号，那么git仓库地址不需要改变。

5、取消全局 用户名/邮箱设置，每个仓库单独设置

取消全局 用户名/邮箱 配置

```
1 git config --global --unset user.name  
2 git config --global --unset user.email
```

为每个仓库单独设置 用户名/邮箱

```
1 cd YourRepoPath  
2 git config user.name "You Name"  
3 git config user.email name@example.com
```

其实很简单，就是去掉--global参数就行了

参考文章：

<https://www.artjay.me/2019/more-ssh/>

搭建自己git服务

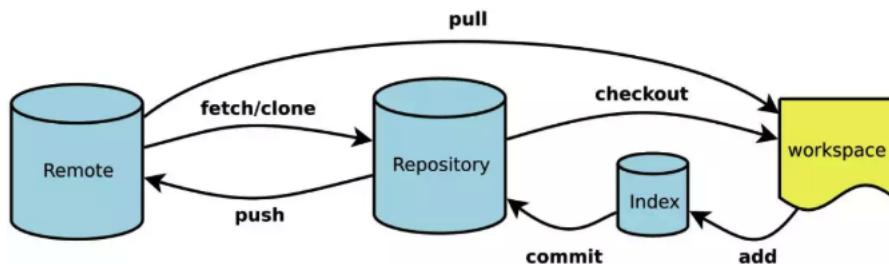
Gogs

一款极易搭建的自助 Git 服务 比gitlab更轻量级

站点: <https://gogs.io/>

git 介绍

Git 工作流程



以上包括一些简单而常用的命令，但是先不关心这些，先来了解下面这4个专有名词。

- 1 **Workspace:** 工作区
- 2
- 3 **Index / Stage:** 暂存区
- 4
- 5 **Repository:** 仓库区（或本地仓库）
- 6
- 7 **Remote:** 远程仓库

工作区

程序员进行开发改动的地方，是你当前看到的，也是最新的。

平常我们开发就是拷贝远程仓库中的一个分支，基于该分支进行开发。在开发过程中就是对工作区的操作。

暂存区

.git 目录下的 index 文件，暂存区会记录 git add 添加文件的相关信息（文件名、大小、timestamp...），不保存文件实体，通过 id 指向每个文件实体。可以使用 git status 查看暂存区的状态。暂存区标记了你当前工作区中，哪些内容是被 git 管理的。

当你完成某个需求或功能后需要提交到远程仓库，那么第一步就是通过 git add 先提交到暂存区，被 git 管理。

本地仓库

保存了对象被提交过的各个版本，比起工作区和暂存区的内容，它要更旧一些。

git commit 后同步 index 的目录树到本地仓库，方便从下一步通过 git push 同步本地仓库与远程仓库的同步。

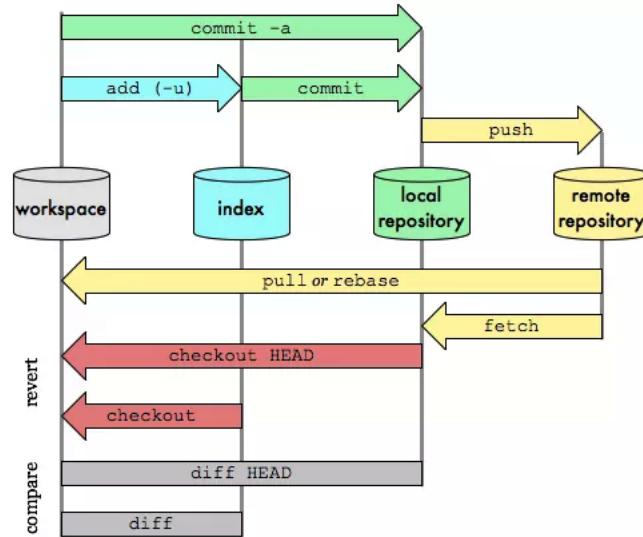
远程仓库

远程仓库的内容可能被分布在多个地点的处于协作关系的本地仓库修改，因此它可能与本地仓库同步，也可能不同步，但是它的内容是最旧的。

小结

- 1 任何对象都是在工作区中诞生和被修改；
- 2
- 3 任何修改都是从进入 index 区才开始被版本控制；
- 4
- 5 只有把修改提交到本地仓库，该修改才能在仓库中留下痕迹；
- 6
- 7 与协作者分享本地的修改，可以把它们 push 到远程仓库来共享。

下面这幅图更加直接阐述了四个区域之间的关系，可能有些命令不太清楚，没关系，下部分会详细介绍。

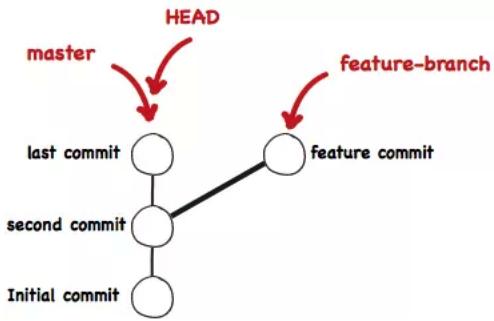


常用Git命令

Git 常用命令速查表		master : 默认开发分支	origin : 默认远程版本库	Head : 默认开发分支	Head^ : Head 的父提交
创建版本库	\$ git clone <url> \$ git init	#克隆远程版本库 #初始化本地版本库			
修改和提交	\$ git status \$ git diff \$ git add . \$ git add <file> \$ git mv <old> <new> \$ git rm <file> \$ git rm --cached <file> \$ git commit -m "commit message" \$ git commit --amend	#查看状态 #查看变更内容 #跟踪所有改动过的文件 #跟踪指定的文件 #文件改名 #删除文件 #停止跟踪文件但不删除 #提交所有更新过的文件 #修改最后一次提交			
查看提交历史	\$ git log \$ git log -p <file> \$ git blame <file>	#查看提交历史 #查看指定文件的提交历史 #以列表方式查看指定文件的提交历史			
撤消	\$ git reset --hard HEAD \$ git checkout HEAD <file> \$ git revert <commit>	#撤消工作目录中所有未提交文件的修改内容 #撤消指定的未提交文件的修改内容 #撤消指定的提交			
分支与标签	\$ git branch \$ git checkout <branch/tag> \$ git branch <new-branch> \$ git branch -d <branch> \$ git tag \$ git tag <tagname> \$ git tag -d <tagname>	#显示所有本地分支 #切换到指定分支或标签 #创建新分支 #删除本地分支 #列出所有本地标签 #基于最新提交创建标签 #删除标签			
合并与衍合	\$ git merge <branch> \$ git rebase <branch>	#合并指定分支到当前分支 #衍合指定分支到当前分支			
远程操作	\$ git remote -v \$ git remote show <remote> \$ git remote add <remote> <url> \$ git fetch <remote> \$ git pull <remote> <branch> \$ git push <remote> <branch> \$ git push <remote> :<branch/tag-name> \$ git push --tags	#查看远程版本库信息 #查看指定远程版本库信息 #添加远程版本库 #从远程库获取代码 #下载代码及快速合并 #上传代码及快速合并 #删除远程分支或标签 #上传所有标签			

网上找了个图，别人整理的一张图，很全很好，借来用下。下面详细解释一些常用命令。

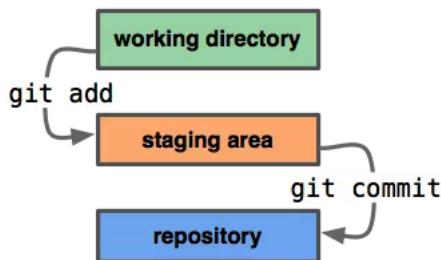
HEAD



在掌握具体命令前，先理解下HEAD。

HEAD，它始终指向当前所处分支的最新的提交点。你所处的分支变化了，或者产生了新的提交点，HEAD就会跟着改变。

add



add相关命令很简单，主要实现将工作区修改的内容提交到暂存区，交由git管理。

```

1 git add .    添加当前目录的所有文件到暂存区
2 git add [dir]  添加指定目录到暂存区，包括子目录
3 git add [file1] 添加指定文件到暂存区

```

commit

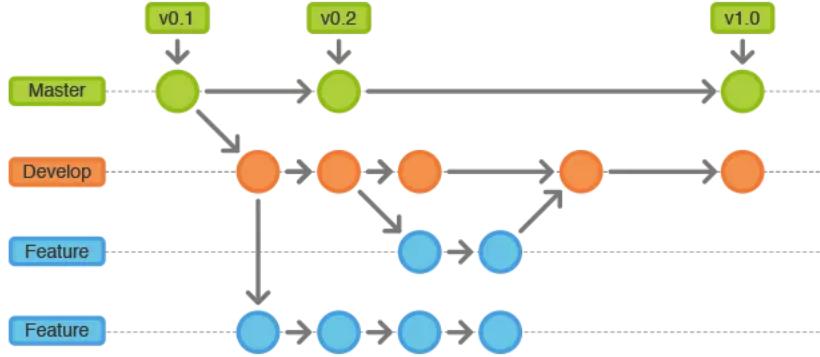
commit相关命令也很简单，主要实现将暂存区的内容提交到本地仓库，并使得当前分支的HEAD向后移动一个提交点。

```

1 git commit -m [message] 提交暂存区到本地仓库,message代表说明信息
2 git commit [file1] -m [message] 提交暂存区的指定文件到本地仓库
3 git commit --amend -m [message] 使用一次新的commit，替代上一次提交

```

branch



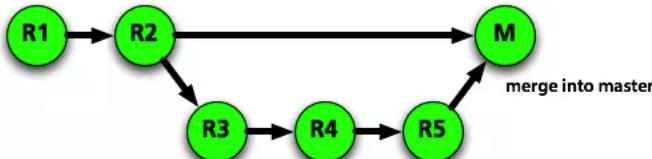
涉及到协作，自然会涉及到分支，关于分支，大概有展示分支，切换分支，创建分支，删除分支这四种操作。

```

1 git branch 列出所有本地分支
2 git branch -r 列出所有远程分支
3 git branch -a 列出所有本地分支和远程分支
4 git branch [branch-name] 新建一个分支，但依然停留在当前分支
5 git checkout -b [branch-name] 新建一个分支，并切换到该分支
6 git branch -track [branch] [remote-branch] 新建一个分支，与指定的远程分支建立追踪关系
7 git checkout [branch-name] 切换到指定分支，并更新工作区
8 git branch -d [branch-name] 删除分支
9 git branch -D [branch-name] 强制删除
10 git push origin -delete [branch-name] 删除远程分支

```

merge



merge命令把不同的分支合并起来。如上图，在实际开放中，我们可能从master分支中切出一个分支，然后进行开发完成需求，中间经过R3,R4,R5的commit记录，最后开发完成需要合入master中，这便用到了merge。

```

1 git fetch [remote] merge之前先拉一下远程仓库最新代码
2 git merge [branch] 合并指定分支到当前分支

```

一般在merge之后，会出现conflict，需要针对冲突情况，手动解除冲突。主要是因为两个用户修改了同一文件的同一块区域。

rebase

rebase又称为衍合，是合并的另外一种选择。

在开始阶段，我们处于new分支上，执行git rebase dev，那么new分支上新的commit都在master分支上重演一遍，最后checkout切换回到new分支。这一点与merge是一样的，合并前后所处的分支并没有改变。git rebase dev，通俗的解释就是new分支想站在dev的肩膀上继续下去。rebase也需要手动解决冲突。

rebase与merge的区别

现在我们有这样的两个分支，test和master，提交如下：

D—E test

/

A—B—C—F master

在master执行git merge test,然后会得到如下结果：

D——E

/ \

A—B—C—F—G test, master

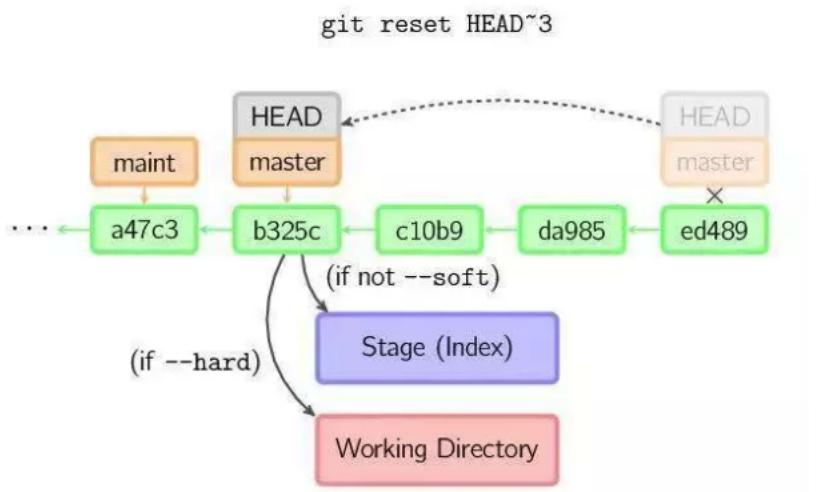
在master执行 git rebase test, 然后得到如下结果：

A—B—D—E—C'—F' test, master

可以看到，merge操作会生成一个新的节点，之前的提交分开显示。而rebase操作不会生成新的节点，是将两个分支融合成一个线性的提交。

如果你想要一个干净的，没有merge commit的线性历史树，那么你应该选择git rebase 如果你想保留完整的历史记录，并且想要避免重写commit history的风险，你应该选择使用git merge。

reset



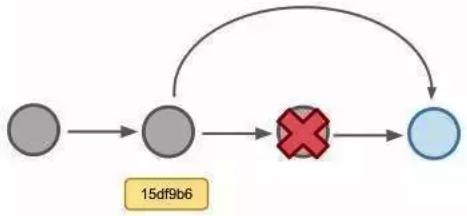
reset命令把当前分支指向另一个位置，并且相应的变动工作区和暂存区。

- | | | |
|---|--|-----------------------------|
| 1 | <code>git reset -soft [commit]</code> | 只改变提交点，暂存区和工作目录的内容都不改变 |
| 2 | <code>git reset -mixed [commit]</code> | 改变提交点，同时改变暂存区的内容 |
| 3 | <code>git reset -hard [commit]</code> | 暂存区、工作区的内容都会被修改到与提交点完全一致的状态 |
| 4 | <code>git reset -hard HEAD</code> | 让工作区回到上次提交时的状态 |

revert

Example

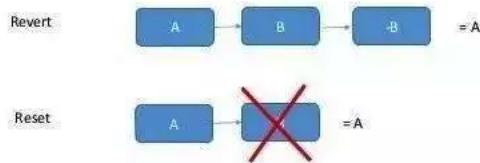
```
git commit -am "update readme"  
git revert 15df9b6
```



git revert用一个新提交来消除一个历史提交所做的任何修改。

revert与reset的区别

Reset versus Revert



git revert是用一次新的commit来回滚之前的commit, git reset是直接删除指定的commit。

在回滚这一操作上看, 效果差不多。但是在日后继续merge以前的老版本时有区别。因为git revert是用一次逆向的commit“中和”之前的提交, 因此日后合并老的branch时, 导致这部分改变不会再次出现, 减少冲突。但是git reset是直接删除指定的commit, 因而和老的branch再次merge时, 这些被回滚的commit应该还会被引入, 产生很多冲突。

git reset 是把HEAD向后移动了一下, 而git revert是HEAD继续前进, 只是新的commit的内容和要 revert的内容正好相反, 能够抵消要被revert的内容。

push

上传本地仓库分支到远程仓库分支, 实现同步。

```
1 git push [remote] [branch]    上传本地指定分支到远程仓库  
2 git push [remote] --force    强行推送当前分支到远程仓库, 即使有冲突  
3 git push [remote] --all     推送所有分支到远程仓库
```

其他命令

```
1 git status   显示有变更的文件  
2 git log     显示当前分支的版本历史  
3 git diff    显示暂存区和工作区的差异  
4 git diff HEAD  显示工作区与当前分支最新commit之间的差异
```

```
5 git cherry-pick [commit]    选择一个commit, 合并进当前分支
```

参考文章 <https://mp.weixin.qq.com/s/ FM-wcjd0WT84splbLuilaw>

shell脚本的静态检查工具shellcheck介绍

使用shellcheck 工具， shell脚本也是可以被静态检查的

(语法检查等，很多ide工具，也有检查功能)

shellcheck 命令的安装：

```
1 apt install shellcheck
```

写完shell脚本,记得用它检查一下,能给你点建议的.

```
1 # 检查指定shell脚本
2
3 shellcheck deployscript.sh
4
5 # 要检查现有项目的所有的脚本,
6
7 find your_project_folder -name "*.sh" | xargs -i shellcheck {}
```

ssh 在本地执行远程主机命令

ssh 本身支持在远程主机中运行命令的，语法就是

```
1 ssh user@host "command1; command2; command3; ...."
```

实例：

```
1 # 打压缩包并上传到跳板机的指定目录
2 function compress() {
3     tar -zcvf sc.tgz -C build . && scp -r sc.tgz root@xxx.xxx.xxx.xx
x:~/oss_download/demo/
4 }
5
6 # 通过 ssh 登录跳板机执行 3 - 7 步
7 # 注意在跳板机登录到目标服务器的时候需要 ssh -tt ,可以在远程机器上 ssh 到其他的
远程主机并执行。详细的解释通过 man ssh 查看
8 # mkdir -p 是如果目标机器不存在这个目录，就先创建这个目录，保证 cd 或者 tar 的时
候不会因为目标目录不存在而引起报错
9
10 function send() {
11     ssh root@xxx.xxx.xxx.xxx "scp -r oss_download/demo/sc.tgz alibab
a@${1}:~/ossdowload/data/demo/ ; ssh -tt alibaba@${1} 'mkdir -p ossd
ownload; cd ossdowload; mkdir -p data/demo/sc/${version}; tar -zvxf d
ata/demo/sc.tgz -C data/demo/sc/${version} && rm -rf data/demo/sc.
tgz && ./ceph_tmp.py'"
12 }
13
14 # 执行 compress 和 send 函数
15 function deploy() {
16     compress
17     send ${1}
18 }
19
20 # 发送到目标服务器
21 deploy 192.168.3.4
```

文件备份

文件使用git备份

```
1 0 4 * * * cd /alidata/www/img_bak/ && cp -R /alidata/www/mynav/share  
d/usr/uploads \  
2 /alidata/www/img_bak/mynav && /usr/bin/git add . && \  
3 /usr/bin/git commit -m 'img bak at'$(date +"%Y\%m\%d")' && \  
4 /usr/bin/git push >> /alidata/log/cron.log
```

crontab 备份

使用git备份

```
1 0 2 * * * cd /alidata/www/cron_bak/ && crontab -l > /alidata/www/cron_bak/crontab.sh \
2 && /usr/bin/git add . && /usr/bin/git commit -m 'cron bak at'$(date + "%Y%m%d") \
3 && /usr/bin/git push >> /alidata/log/cron.log 2>&1
```

mysql 数据库备份

备份数据库到git

例：

数据库备份文件目录为 /alidata/www/sqlbak/ 在该目录下创建git，指向远程git仓库

备份数据库名称为 myblog

备份shell如下：

```
1 0 1 * * * cd /alidata/www/sqlbak/ && /usr/local/mysql/bin/mysqldump  
myblog > \  
2 /alidata/www/sqlbak/db-myblog-$(date +"%Y\%m\%d").sql && git add .  
\  
3 && git commit -m 'db myblog bak at'$(date +"%Y\%m\%d") && git push
```

mysqldump免输入账号密码前提配置：

```
1 vim /etc/my.cnf  
2  
3 [mysqldump]  
4  
5 user=your_backup_user_name  
6  
7 password=your_backup_password
```

以上配置后使用mysqldump命令就不需要涉及用户名密码相关信息。

ubuntu 切换 sh 为 bash

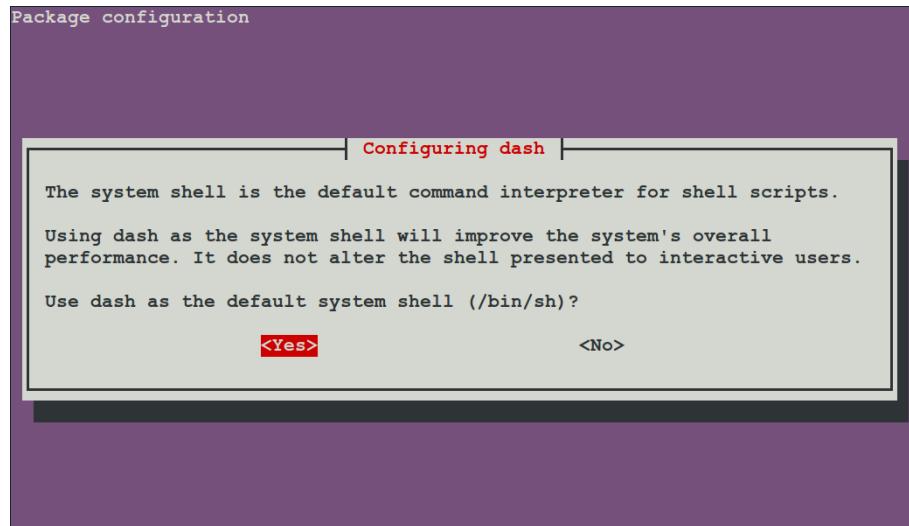
当前ubuntu默认的shell环境是dash,对于习惯在bash下操作的用户，可以通过如下操作切换为bash:

```
1 ls -l /bin/sh  
2  
3 sudo dpkg-reconfigure dash
```

然后选择 'NO' , 回车即可

然后执行以下命令查看是否生效

```
1 ll /bin/sh
```



切换国内镜像源，加速apt-get

使用apt-get命令安装包时，由于系统自带的下载源在国外服务器上，故下载速度较慢。若切换为国内源，将显著提升下载速度。下列是设置步骤：

1、查找适合自己系统的镜像源配置

以清华大学镜像源为例

官网地址：<https://mirrors.tuna.tsinghua.edu.cn/help/ubuntu/>

Ubuntu 的软件源配置文件是 `/etc/apt/sources.list`。将系统自带的该文件做个备份，将该文件替换为下面内容，即可使用 TUNA 的软件源镜像。

```
1 # 默认注释了源码镜像以提高 apt update 速度，如有需要可自行取消注释
2 deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main restricted
   universe multiverse
3 # deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main re
   stricted universe multiverse
4 deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates main
   restricted universe multiverse
5 # deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates
   main restricted universe multiverse
6 deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-backports mai
   n restricted universe multiverse
7 # deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-backpor
   ts main restricted universe multiverse
8 deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-security main
   restricted universe multiverse
9 # deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-securit
   y main restricted universe multiverse
10 # 预发布软件源，不建议启用
11 # deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-proposed ma
   in restricted universe multiverse
12 # deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-propose
   d main restricted universe multiverse
```

2、更新源，使配置生效

```
1 sudo apt-get update
```

ubuntu开启SSH服务远程登录

ubuntu默认是没有安装ssh服务的，所以为了可以远程登录ubuntu服务器，需要先在ubuntu服务器安装ssh服务。

SSH分客户端openssh-client和openssh-server

如果你只是想登陆别的机器的SSH只需要安装openssh-client (ubuntu有默认安装，如果没有则sudo apt-get install openssh-client) ，如果要使本机开放SSH服务就需要安装openssh-server。

查看当前的ubuntu是否安装了ssh-server服务。默认只安装ssh-client服务。

```
1 dpkg -l | grep ssh
```

安装ssh-server服务

```
1 sudo apt-get install openssh-server
```

确认ssh-server是否启动了：

```
1 ps -e | grep ssh
```

如果看到sshd那说明ssh-server已经启动了。

如果没有则可以这样启动：

```
1 sudo /etc/init.d/ssh start或sudo service ssh start
```

Ubuntu 18.04 单网卡多IP设置

Ubuntu 17.04 和18.04系统版本启用了新的网络工具netplan，对于命令行配置网络参数跟之前的版本有比较大的差别

之前的版本是在/etc/network/interfaces中修改
新版本使用/etc/netplan/*.yaml

查看系统版本命令

```
1 lsb_release -a
```

网卡配置详细举例

1. 使用vi 编辑配置网卡文件

```
vim /etc/netplan/*.yaml
```

```
1 # This file describes the network interfaces available on your system
2 # For more information, see netplan(5).
3 network:
4   version: 2
5   renderer: networkd
6   ethernets:
7     eno1: #配置的网卡名称
8       addresses: [ 192.168.0.1/27 ] #设置本机IP及掩码
9       gateway4: 192.168.0.254 #设置网关
10      nameservers:
11        addresses: [8.8.8.8] #设置DNS
```

若要添加IP: 192.168.0.2, 192.168.0.3, 192.168.0.4

修改上述内容为：

```
1 # This file describes the network interfaces available on your system
2 # For more information, see netplan(5).
3 network:
4   version: 2
5   renderer: networkd
6   ethernets:
7     eno1: #配置的网卡名称
8       addresses: [ 192.168.0.1/27, 192.168.0.2/28, 192.168.0.3/29, 1
9         92.168.0.4/30 ] #设置本机IP及掩码
10    gateway4: 192.168.0.254 #设置网关
11    nameservers:
12      addresses: [8.8.8.8] #设置DNS
```

2. 执行命令使网卡配置文件生效

```
1 netplan apply
```

实在不行重启服务器

搭建自己的 Sentry 服务

前文已经介绍了Sentry服务用途，一个自动化，实时的异常收集、监控、查看、提醒服务。

Sentry服务端原理

Sentry的服务端分为web、cron、worker这几块，我们的应用只是把错误信息上报给sentry的web端。

web处理后放入消息队列或Redis内存队列，worker从队列中消费数据进行处理。

主要处理这些数据的逻辑和队列的压力都在sentry服务端那边。所以实质上对我们系统性能的影响不大。

Sentry服务端安装

前文说到Sentry开源，且可以自己部署使用，有两种安装方式一个是 docker 一个是 python。

下面以mac系统docker为例，介绍一下Sentry的安装步骤：

1、安装docker

见以前文章，不再赘述，

注意安装环境要求：

```
[scode type="red"]  
至少2400mb 内存  
2个 CPU 内核  
[/scode]
```

2、正式搭建 sentry

做完了准备工作，就可以开始搭建 sentry 了。

从 GitHub 上面获取最新的 sentry

```
1 git clone https://github.com/getsentry/onpremise.git
```

获取到本地之后，就可以根据他的 README.md 开始着手搭建了，整个过程还是比较顺利的。

官方安装文档：<https://docs.sentry.io/server/installation/>

官方配置文档：<https://docs.sentry.io/server/config/>

根据上面文档，安装过程如下：

进入 clone 下来的 onpremise 目录依次执行

1、配置项目名称

```
1 .env文件  
2 COMPOSE_PROJECT_NAME=项目名称 # 注意这个项目不是sentry中各个项目的名称，和那个  
没关系
```

2、配置通知邮箱

```
1 sentry/config.yml文件
2
3 #####
4 # Mail Server #
5 #####
6
7 # mail.backend: 'smtp' # Use dummy if you want to disable email entirely
8 mail.host: 'smtp.qq.com'
9 mail.port: 25
10 mail.username: 'xxxxxxxx@qq.com'
11 mail.password: 'xxxxxxxxxx'
12 mail.use-tls: false
13 # The email address to send on behalf of
14 mail.from: 'xxxxxxxx@qq.com'
```

3、若想修改web访问端口，如本地安装php，因为php-fpm端口默认也是9000，可以修改本地与docker端口映射关系，如本地10000映射docker 9000，修改方法：

```
1 docker-compose.yml 文件
2
3 web:
4     << : *sentry_defaults
5     ports:
6         - '10000:9000/tcp'
```

4、运行安装脚本，创建容器

运行之前可以去官方配置文档中，看看是否还需要配置啥。

```
1 ./install.sh
2
3 构建镜像
4 执行过程中会让你输入邮箱、密码作为登录使用。
```

5、创建并启动容器

```
1 docker-compose up -d
```

6、网页访问

<http://localhost:9000>

对于在服务器搭建的用nginx代理一下访问接口。

```
1 server {
2     listen      80;
3     server_name yourdomain.com
4     location / {
5         proxy_pass  http://127.0.0.1:9000;
6     }
7 }
```

7、若要修改文件，修改完成后执行以下命令使配置生效

```
1 docker-compose build  
2 docker-compose restart
```

设置中文

搭建完成后可以将sentry设置成中文显示

The screenshot shows the Sentry account settings interface. On the left is a sidebar with links like '分配给我的', 'Bookmarked Issues', 'Activity', '统计', and '设置'. The main area is titled 'Account Details' under '账户'. It includes sections for 'ACCOUNT DETAILS' (Name: 409178623@qq.com), 'PREFERENCES' (Stacktrace Order: Default (let Sentry decide), Language: Simplified Chinese, Timezone: (UTC+0000) UTC), and 'AVATAR' (radio buttons for Use initials, Upload an image, or Use Gravatar). A red box highlights the 'Simplified Chinese' language selection. In the bottom right corner of the main area, there is a circular icon with the number '4' and a 'Save Avatar' button.

Sentry 实时异常信息监控系统介绍

简介

Sentry 是什么？中文翻译过来是 哨兵 的意思，从字面中可以知道『站岗、放哨、巡逻、稽查的士兵』，Sentry 是程序的 哨兵 ，它可以监控我们在生产环境中项目的运行状态，一旦某段代码运行报错，或者异常，会第一时间把报错的 路由，异常文件，请求方式 等一些非常详细的信息以消息或者邮件给我们，让我们第一时间知道：程序出错了，然后我们可以从 Sentry 给我们的详细的错误信息中瞬间找到我们需要处理的代码，在老板不知情的情况下悄悄把 Bug 修复调，你肯定不想等着老板来找你吧。

自动化，实时的异常 收集、监控、提醒服务。

你如果试用 [Sentry 官方](#) 提供给你的服务是需要收费的，不过可以免费试用。试用版有消息条数限制，注意不要超限。Sentry提供了开源版本，你也可以自己搭建 Sentry：自行搭建当然就不收费啦，这一点很赞。你可以查看 [文档](#) 自行安装，文档介绍的很详细。文档最下方有两种安装方式一个是 docker 一个是 python。一般推荐docker,这个方便很多。

解决什么问题

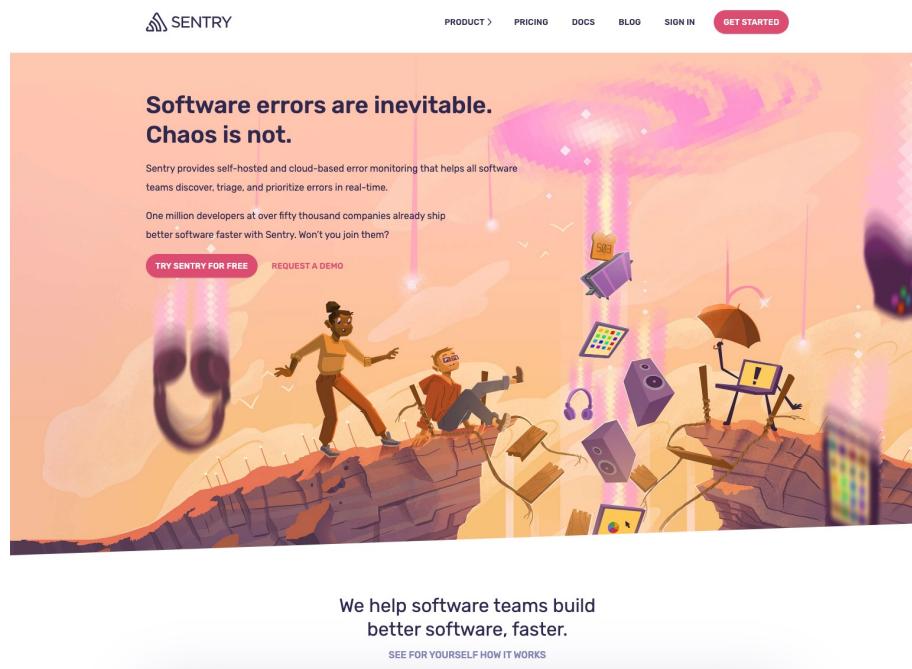
比如服务端开发人员异常日志太多不便于查找，或太多了懒得查找了，或干脆没有记录。

如服务端开发人员没有生产环境主机权限，无法去查看异常日志。

Sentry将异常信息汇总、过滤、通知、图形化、报表展示，便于开发人员发现问题，调试解决问题

Sentry 使用体验

这是 [Sentry 的官网](#)，当然你也可以自己部署Sentry服务，我们先不管来体验一把，注册一个账号



登录系统后我们要创建一个项目，这个项目要对应我们需要监控异常的项目。

Create a new Project
Projects allow you to scope events to a specific application in your organization. For example, you might have separate projects for your API server and frontend client.

Popular Browser Server Mobile Desktop All

Give your project a name Assign a Team
Project name #murachang Create Project

创建项目时要选择所属语言和框架，基本支持大部分的常见语言和框架。我们以php laravel 项目为例。

项目创建成功后他会提示你如何将异常监控引入你的代码中

CONFIGURE LARAVEL

This is a quick getting started guide. For in-depth instructions on integrating Sentry with Laravel, view our complete documentation.

Install the `sentry/sentry-laravel` package:

```
$ composer require sentry/sentry-laravel:1.6.1
```

If you're on Laravel 5.5 or later the package will be auto-discovered. Otherwise you will need to manually configure it in your `config/app.php`.

Add Sentry reporting to `App\Exceptions\Handler.php`:

```
public function report(Exception $exception)
{
    if ($app()->bound('sentry') && $this->shouldReport($exception)) {
        $app('sentry')->captureException($exception);
    }

    parent::report($exception);
}
```

Create the Sentry configuration file (`config/sentry.php`) with this command:

```
$ php artisan vendor:publish --provider="Sentry\Laravel\ServiceProvider"
```

Add your DSN to `.env`:

```
SENTRY_LARAVEL_DSN=https://e92f4e9b81524e89b7b57f64134c57f7@sentry.io/2568517
```

You can easily verify that Sentry is capturing errors in your Laravel application by creating a debug route that will throw an exception:

```
Route::get('/debug-sentry', function () {
    throw new Exception('My first Sentry error!');
});
```

Visiting this route will trigger an exception that will be captured by Sentry.

Got it! Take me to the Issue Stream.

文字版：

```

1 1、安装sentry/sentry-laravel包
2
3 $ composer require sentry/sentry-laravel:1.6.1
4
5 注意： If you're on Laravel 5.5 or later the package will be auto-discovered.
6 Otherwise you will need to manually configure it in your config/app.php.
7
8 我的测试laravel是6.x 版本，跳过了这一步
9
10 2、加入sentry代码到laravel异常监控中
11
12 public function report(Exception $exception)
13 {
14     if ($app()->bound('sentry') && $this->shouldReport($exception)) {
15         $app('sentry')->captureException($exception);
16     }
}
```

```

17
18     parent::report($exception);
19 }
20
21 除了如上所示加到异常监控中，我们还可以主动收集信息，在你任意想监控的地方可以通过主动
调用上面代码实施监控
22
23 3、创建Sentry配置文件
24
25 $ php artisan vendor:publish --provider="SentryLaravelServiceProvider"
26
27 4、将dsn加入到.env文件
28
29 这个dsn相当于远程sentry服务地址
30
31 SENTRY_LARAVEL_DSN=https://e92f4e9b81524e89b7qwef64134c57f7@sentry.i
o/1238517
32
33 3、测试
34
35 Route::get('/debug-sentry', function () {
36     throw new Exception('My first Sentry error!');
37 });
38
39 laravel项目中加入以上路由，在浏览器访问一下就会将异常信息推送到Sentry控制台中。

```

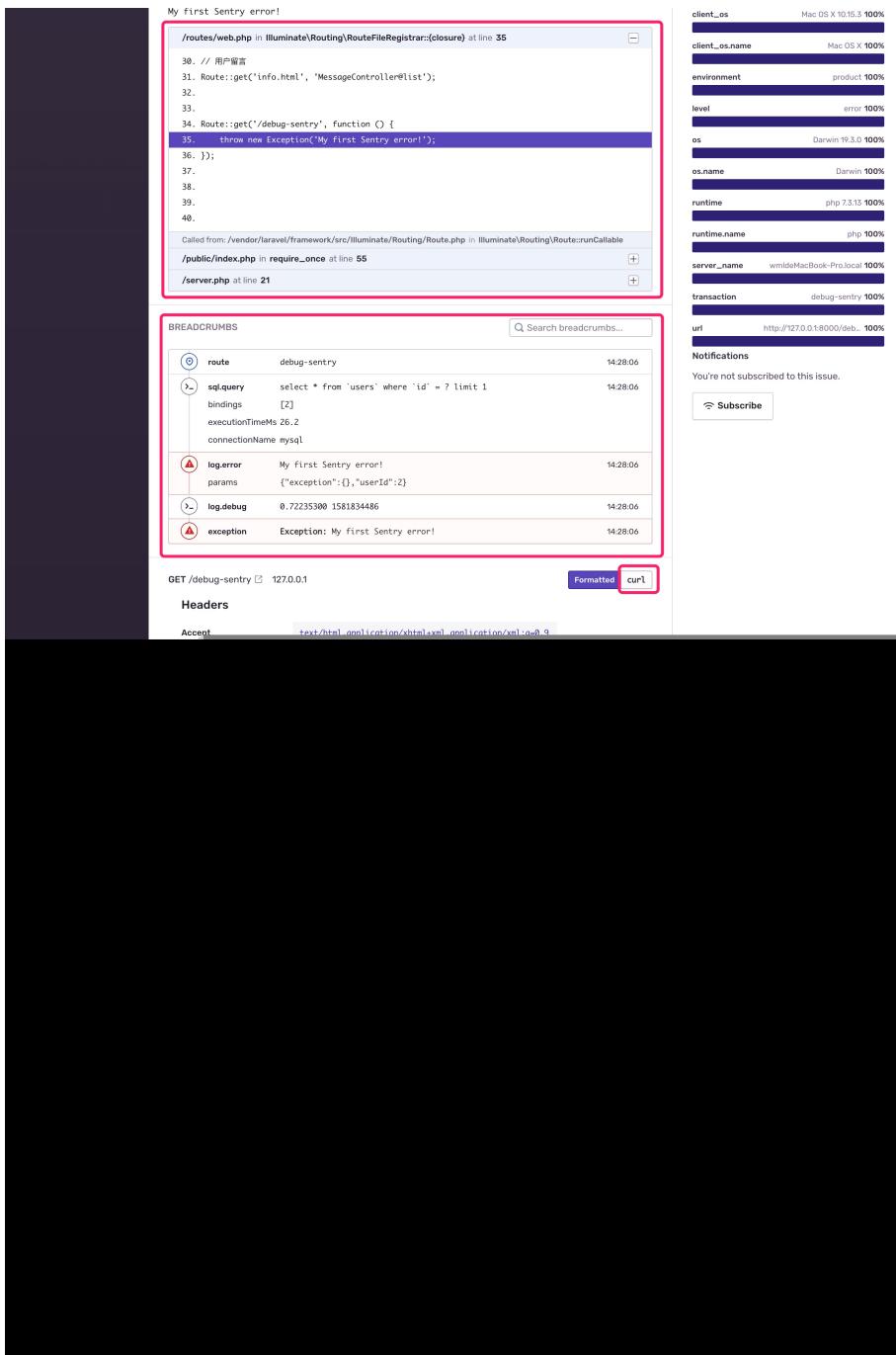
按照以上步骤修改代码后点击底部按钮进入异常监控台：

The screenshot shows the Sentry interface with a single error event listed under the 'Issues' tab. The event is titled 'My first Sentry error' and was created 'a few seconds ago'. The interface includes filters for 'All Environments' and 'Last 14 days', and tabs for 'GRAPH', 'EVENTS', 'USERS', and 'ASSIGNEE'.

请求一下laravel项目刚才我们添加的会抛出异常的路由，此时我们就看到了请求的异常信息，并且收到异常监控邮件。上图中我们可以看到异常出现次数、异常出现频率图

点击进入异常详情，如下所示：

The screenshot shows the detailed view of the 'LaravelTest-1' issue. It includes sections for 'Event' (ID: 40664f3da65e459c9a8a4bdca826c9bd, timestamp: Feb 16, 2020 6:28:06 AM UTC), 'Details' (including browser, PHP version, and OS), 'Tags' (listing various environment variables like browser, browser.name, client.os, etc.), and 'Logs' (with a red box highlighting the URL). The right side of the screen shows ownership rules, last seen activity, and linked issues.



此时我们可以看到非常详细的异常信息，包括异常内容、浏览器信息、PHP版本、访问者系统版本、访问路径、请求header、抛出异常用户环境百分比等大量信息。非常便于我们调试代码。

对于异常通知

Sentry在收到异常信息时，默认会给我们注册时填写的邮箱发送异常监控邮件。

除了邮件外Sentry也可以通过HipChat、钉钉等产品进行通知，需要时在研究一下，

End !

参考文章：<https://learnu.com/articles/4235/sentry-automation-exception-alert>

Linux服务器磁盘空间占满解决方法

解决服务器磁盘占满的思路:

查找磁盘磁盘占满的原因，系统中的大文件遍历。若这些文件是系统运行中生成的已不用日志文件可以手动删掉这些文件，若不能删除那就只能扩充磁盘了，或者备份到其他磁盘。

解决过程

查看磁盘占用情况

```
1 cd /
2 df -h #查看是哪个挂载目录满了，常常是根目录/占满
```

```
[root@alidata ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1       20G   9.6G   9.0G  52% /
tmpfs           1.9G     0   1.9G   0% /dev/shm
/dev/xvdb1      493G  278G  190G  60% /alidata
/dev/xvdc1      493G  291G  177G  63% /logs
```

如上图所示显示多个挂载的占用情况

假设我们认为 /dev/xvdc1 磁盘占用率太高，要清理，进行以下操作

查看该文件夹下文件情况

```
1 sudo du -h --max-depth=1 /logs/ | sort -hr | head -10
```

命令说明：

```
1 du #计算出单个文件或者文件夹的磁盘空间占用。
2 --max-depth # 展示的目录深度
3 sort -hr # -r由大到小排序    -h 使用易读性数字(例如： 2K 1G)
4 head -10 # 显示前10条记录
```

```
[root@alidata ~]# sudo du -h --max-depth=1 /logs/ | sort -hr | head -10
291G   /logs/
113G   /logs/backuplogs
93G    /logs/result
86G    /logs/orig
852K   /logs/log
72K    /logs/src
16K    /logs/lost+found
4.0K   /logs/tvjianshen
```

如上图所示，可以看到 /logs/backuplogs 目录大小为113G，若想看继续看/logs/backuplogs 目录下那个文件占用较大空间，可以继续执行

```
1 sudo du -h --max-depth=1 /logs/backuplogs | sort -hr | head -10
```

```
[root@... ~]# sudo du -h --max-depth=1 /logs/backuplogs | sort -hr | head -10
113G  /logs/backuplogs
107G  /logs/backuplogs/[REDACTED]
6.2G   /logs/backuplogs/[REDACTED]
411M   /logs/backuplogs/c/[REDACTED]
67M    /logs/backuplogs/tv
2.4M   /logs/backuplogs/sdk/[REDACTED]
280K   /logs/backuplogs/oassdk
188K   /logs/backuplogs/test
```

由此可一步步查看磁盘目录状态，最终确定是删除大文件还是扩充磁盘即可。

注意

对于项目中生成的日志文件最好用logrotated管理起来，防止占用太多磁盘空间

参考文章 <http://wangmaolin.net/index.php/linux/78.html>

linux 国内更新源

给自家的Ubuntu下载软件速度有点慢，毕竟是从国外下载软件，就想更换到国内比较好的更新源（就是这些软件所在的服务器），一般直接百度Ubuntu更新源就能出来一大堆，这时候最好是找和自己Ubuntu版本一致的更新源，我的Ubuntu版本是16.04，下面是我找到的一个比较好的更新源

中科大源选择器地址：

<https://mirrors.ustc.edu.cn/repogen/>

东北大学

```
1 deb-src http://mirror.neu.edu.cn/ubuntu/ xenial main restricted #Add
   ed by software-properties
2 deb http://mirror.neu.edu.cn/ubuntu/ xenial main restricted
3 deb-src http://mirror.neu.edu.cn/ubuntu/ xenial restricted multivers
   e universe
4 deb http://mirror.neu.edu.cn/ubuntu/ xenial-updates main restricted
5 deb-src http://mirror.neu.edu.cn/ubuntu/ xenial-updates main restric
   ted multiverse universe
6 deb http://mirror.neu.edu.cn/ubuntu/ xenial universe
7 deb http://mirror.neu.edu.cn/ubuntu/ xenial-updates universe
8 deb http://mirror.neu.edu.cn/ubuntu/ xenial multiverse
9 deb http://mirror.neu.edu.cn/ubuntu/ xenial-updates multiverse
10 deb http://mirror.neu.edu.cn/ubuntu/ xenial-backports main restric
    ed universe multiverse
11 deb-src http://mirror.neu.edu.cn/ubuntu/ xenial-backports main restr
    icted universe multiverse
12 deb http://archive.canonical.com/ubuntu xenial partner
13 deb-src http://archive.canonical.com/ubuntu xenial partner
14 deb http://mirror.neu.edu.cn/ubuntu/ xenial-security main restricted
15 deb-src http://mirror.neu.edu.cn/ubuntu/ xenial-security main restric
    ted multiverse universe
16 deb http://mirror.neu.edu.cn/ubuntu/ xenial-security universe
17 deb http://mirror.neu.edu.cn/ubuntu/ xenial-security multiverse
```

清华大学

```
1 # deb cdrom:[Ubuntu 16.04 LTS _Xenial Xerus_ - Release amd64 (201604
   20.1)]/ xenial main restricted
2 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial main restrict
   ed
3 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-updates main
   restricted
4 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial universe
5 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-updates unive
   rse
6 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial multiverse
7 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-updates multi
   verse
8 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-backports mai
   n restricted universe multiverse
9 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-security main
   restricted
10 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-security univ
```

```
      erse
11 deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-security mult
iverse
```

阿里云

```
1 # deb cdrom:[Ubuntu 16.04 LTS _Xenial Xerus_ - Release amd64 (201604
20.1)]/ xenial main restricted
2 deb-src http://archive.ubuntu.com/ubuntu xenial main restricted
3 deb http://mirrors.aliyun.com/ubuntu/ xenial main restricted
4 deb-src http://mirrors.aliyun.com/ubuntu/ xenial main restricted multiverse universe
5 deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main restricted
6 deb-src http://mirrors.aliyun.com/ubuntu/ xenial-updates main restricted multiverse universe
7 deb http://mirrors.aliyun.com/ubuntu/ xenial universe
8 deb http://mirrors.aliyun.com/ubuntu/ xenial-updates universe
9 deb http://mirrors.aliyun.com/ubuntu/ xenial multiverse
10 deb http://mirrors.aliyun.com/ubuntu/ xenial-updates multiverse
11 deb http://mirrors.aliyun.com/ubuntu/ xenial-backports main restricted universe multiverse
12 deb-src http://mirrors.aliyun.com/ubuntu/ xenial-backports main restricted universe multiverse
13 deb http://archive.canonical.com/ubuntu xenial partner
14 deb-src http://archive.canonical.com/ubuntu xenial partner
15 deb http://mirrors.aliyun.com/ubuntu/ xenial-security main restricted
16 deb-src http://mirrors.aliyun.com/ubuntu/ xenial-security main restricted multiverse universe
17 deb http://mirrors.aliyun.com/ubuntu/ xenial-security universe
18 deb http://mirrors.aliyun.com/ubuntu/ xenial-security multiverse
```

下面是更换步骤：

1、备份原来的更新源：

```
1 cp /etc/apt/sources.list /etc/apt/sources.list.backup
```

2、删除原来更新源：

```
1 rm -f /etc/apt/sources.list
```

3、编辑新的更新源：

```
1 vim /etc/apt/sources.list
2
3 选择上面其中的一个源写入改文件
```

4、让更新源生效

```
1 sudo apt-get update
```

5、安装软件

```
1 sudo apt-get install 软件名称、  
2  
3  
4 例如：  
5 sudo apt-get install vim      安装vim
```

基于docker的php开发环境搭建 (dnmp)

DNMP (Docker + Nginx + MySQL + PHP7/5 + Redis) 是一款基于docker的全功能的LNMP一键安装程序。

项目github地址: <https://github.com/wmlc/dnmp>

DNMP项目特点

1. 100% 开源
2. 100% 遵循Docker标准
3. 支持多版本PHP共存, 可任意切换 (PHP5.4、PHP5.6、PHP7.1、PHP7.2、PHP7.3)
4. 支持绑定任意多个域名
5. 支持HTTPS和HTTP/2
6. PHP源代码、MySQL数据、配置文件、日志文件都可在Host中直接修改查看
7. 内置完整PHP扩展安装命令
8. 默认支持pdo_mysql、mysqli、mbstring、gd、curl、opcache等常用热门扩展, 根据环境灵活配置
9. 可一键选配常用服务:
 - 多PHP版本: PHP5.4、PHP5.6、PHP7.1-7.3
 - Web服务: Nginx、Openresty
 - 数据库: MySQL5、MySQL8、Redis、memcached、MongoDB、ElasticSearch
 - 消息队列: RabbitMQ
 - 辅助工具: Kibana、Logstash、phpMyAdmin、phpRedisAdmin、AdminMongo
10. 实际项目中应用, 确保100%可用
11. 所有镜像源于[Docker官方仓库](#), 安全可靠
12. 一次配置, Windows、Linux、MacOs皆可用

1.目录结构

1 /	
2 --- data	数据库数据目录
3 --- esdata	ElasticSearch 数据目录
4 --- mongo	MongoDB 数据目录
5 --- mysql	MySQL8 数据目录
6 --- mysql5	MySQL5 数据目录
7 --- services	服务构建文件和配置文件目录
8 --- elasticsearch	ElasticSearch 配置文件目录
9 --- mysql	MySQL8 配置文件目录
10 --- mysql5	MySQL5 配置文件目录
11 --- nginx	Nginx 配置文件目录
12 --- php	PHP5.6 – PHP7.3 配置目录
13 --- php54	PHP5.4 配置目录
14 --- redis	Redis 配置目录
15 --- logs	日志目录
16 --- docker-compose.sample.yml	Docker 服务配置示例文件
17 --- env.sample	环境配置示例文件
18 --- www	PHP 代码目录

2.快速使用

1. 本地安装
 - git
 - Docker (系统需为Linux, Windows 10 Build 15063+, 或MacOS 10.12+, 且必须要64位)

- o docker-compose 1.7.0+

2. clone 项目:

```
1 $ git clone https://github.com/yeszao/dnmp.git
```

3. 如果不是 root 用户, 还需将当前用户加入 docker 用户组:

```
1 $ sudo gpasswd -a ${USER} docker
```

4. 拷贝并命名配置文件 (Windows系统请用 copy 命令), 启动:

```
1 $ cd dnmp                                # 进入项目目录
2 $ cp env.sample .env                         # 复制环境变量文件
3 $ cp docker-compose.sample.yml docker-compose.yml # 复制 docker-compose 配置文件。默认启动3个服务:
4                                         # Nginx、PHP7和MySQL
5                                         # MySQL。要开启更多其他服务, 如Redis、
6                                         # MongoDB, ElasticSearch等, 请删
7                                         # 除服务块前的注释
7 $ docker-compose up                          # 启动
```

5. 在浏览器中访问: http://localhost 或 https://localhost (自签名HTTPS演示)就能看到效果, PHP代码在文件 ./www/localhost/index.php。

3.PHP和扩展

3.1 切换Nginx使用的PHP版本

首先, 需要启动其他版本的PHP, 比如PHP5.4, 那就先在 docker-compose.yml 文件中删除PHP5.4前面的注释, 再启动PHP5.4容器。

PHP5.4启动后, 打开Nginx 配置, 修改 fastcgi_pass 的主机地址, 由 php 改为 php54, 如下:

```
1     fastcgi_pass    php:9000;
```

为:

```
1     fastcgi_pass    php54:9000;
```

其中 php 和 php54 是 docker-compose.yml 文件中服务器的名称。

最后, 重启 Nginx 生效。

```
1 $ docker exec -it nginx nginx -s reload
```

这里两个 nginx, 第一个是容器名, 第二个是容器中的 nginx 程序。

3.2 安装PHP扩展

PHP的很多功能都是通过扩展实现, 而安装扩展是一个略费时间的过程, 所以, 除PHP内置扩展外, 在 env.sample 文件中我们仅默认安装少量扩展, 如果要安装更多扩展, 请打开你的 .env 文件修改如下的PHP配置, 增加需要的PHP扩展:

```
1 PHP_EXTENSIONS pdo_mysql,opcache,redis      # PHP 要安装的扩展列表, 英文  
逗号隔开  
2 PHP54_EXTENSIONS opcache,redis             # PHP 5.4要安装的扩展列  
表, 英文逗号隔开
```

然后重新build PHP镜像。

```
1 docker-compose build php
```

可用的扩展请看同文件的 `env.sample` 注释块说明。

3.3 Host中使用php命令行 (php-cli)

1. 参考 `bash.alias.sample`示例文件, 将对应 php cli 函数拷贝到主机的 `~/.bashrc` 文件。
2. 让文件起效:

```
1 source ~/.bashrc
```

3. 然后就可以在主机中执行php命令了:

```
1 ~ php -v  
2 PHP 7.2.13 (cli) (built: Dec 21 2018 02:22:47) ( NTS )  
3 Copyright (c) 1997-2018 The PHP Group  
4 Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies  
5     with Zend OPcache v7.2.13, Copyright (c) 1999-2018, by Zend Techn  
ologies  
6     with Xdebug v2.6.1, Copyright (c) 2002-2018, by Derick Rethans
```

3.4 使用composer

方法1: 主机中使用composer命令

1. 确定composer缓存的路径。比如, 我的dnmp下载在 `~/dnmp` 目录, 那composer的缓存路径就是 `~/dnmp/data/composer`。
2. 参考 `bash.alias.sample`示例文件, 将对应 php composer 函数拷贝到主机的 `~/.bashrc` 文件。
这里需要注意的是, 示例文件中的 `~/dnmp/data/composer` 目录需是第一步确定的目录。
3. 让文件起效:

```
1 source ~/.bashrc
```

4. 在主机的任何目录下就能用composer了:

```
1 cd ~/dnmp/www/  
2 composer create-project yeszao/fastphp project --no-dev
```

5. (可选) 第一次使用 composer 会在 `~/dnmp/data/composer` 目录下生成一个 `config.json` 文件, 可以在这个文件中指定国内仓库, 例如:

```
1 {  
2     "config": {},  
3     "repositories": {  
4         "packagist": {  
5             "type": "composer",  
6             "url": "https://packagist.laravel-china.org"  
7         }  
8     }  
9 }
```

```
8      }
9 }
```

方法二：容器内使用composer命令

还有另外一种方式，就是进入容器，再执行`composer`命令，以PHP7容器为例：

```
1 docker exec -it php /bin/sh
2 cd /www/localhost
3 composer update
```

4.管理命令

4.1 服务器启动和构建命令

如需管理服务，请在命令后面加上服务器名称，例如：

```
1 $ docker-compose up                      # 创建并且启动所有容器
2 $ docker-compose up -d                     # 创建并且后台运行方式启动所有
     容器
3 $ docker-compose up nginx php mysql        # 创建并且启动nginx、php、m
     ysql的多个容器
4 $ docker-compose up -d nginx php  mysql    # 创建并且已后台运行的方式启动
     nginx、php、mysql容器
5
6
7 $ docker-compose start php                 # 启动服务
8 $ docker-compose stop php                  # 停止服务
9 $ docker-compose restart php               # 重启服务
10 $ docker-compose build php                # 构建或者重新构建服务
11
12 $ docker-compose rm php                 # 删除并且停止php容器
13 $ docker-compose down                   # 停止并删除容器，网络，图像和
     挂载卷
```

4.2 添加快捷命令

在开发的时候，我们可能经常使用`docker exec -it`进入到容器中，把常用的做成命令别名是个省事的方法。

首先，在主机中查看可用的容器：

```
1 $ docker ps                      # 查看所有运行中的容器
2 $ docker ps -a                    # 所有容器
```

输出的`NAMES`那一列就是容器的名称，如果使用默认配置，那么名称就是`nginx`、`php`、`php56`、`mysql`等。

然后，打开`~/.bashrc`或者`~/.zshrc`文件，加上：

```
1 alias dnginx='docker exec -it nginx /bin/sh'
```

```
2 alias dphp='docker exec -it php /bin/sh'
3 alias dphp56='docker exec -it php56 /bin/sh'
4 alias dphp54='docker exec -it php54 /bin/sh'
5 alias dmysql='docker exec -it mysql /bin/bash'
6 alias dredis='docker exec -it redis /bin/sh'
```

下次进入容器就非常快捷了，如进入php容器：

```
1 $ dphp
```

4.3 查看docker网络

```
1 ifconfig docker0
```

用于填写extra_hosts容器访问宿主机的hosts地址

5. 使用Log

Log文件生成的位置依赖于conf下各log配置的值。

5.1 Nginx日志

Nginx日志是我们用得最多的日志，所以我们单独放在根目录log下。

log会目录映射Nginx容器的/var/log/nginx目录，所以在Nginx配置文件中，需要输出log的位置，我们需要配置到/var/log/nginx目录，如：

```
1 error_log  /var/log/nginx/localhost.error.log  warn;
```

5.2 PHP-FPM日志

大部分情况下，PHP-FPM的日志都会输出到Nginx的日志中，所以不需要额外配置。

另外，建议直接在PHP中打开错误日志：

```
1 error_reporting(E_ALL);
2 ini_set('error_reporting', 'on');
3 ini_set('display_errors', 'on');
```

如果确实需要，可按一下步骤开启（在容器中）。

1. 进入容器，创建日志文件并修改权限：

```
1 $ docker exec -it php /bin/sh
2 $ mkdir /var/log/php
3 $ cd /var/log/php
4 $ touch php-fpm.error.log
5 $ chmod a+rw php-fpm.error.log
```

2. 主机上打开并修改PHP-FPM的配置文件`conf/php-fpm.conf`，找到如下一行，删除注释，并改值为：

```
1 php_admin_value[error_log] = /var/log/php/php-fpm.error.log
```

3. 重启PHP-FPM容器。

5.3 MySQL日志

因为MySQL容器中的MySQL使用的是`mysql`用户启动，它无法自行在`/var/log`下的增加日志文件。所以，我们把MySQL的日志放在与data一样的目录，即项目的`mysql`目录下，对应容器中的`/var/lib/mysql/`目录。

```
1 slow-query-log-file      = /var/lib/mysql/mysql.slow.log
2 log-error                 = /var/lib/mysql/mysql.error.log
```

以上是`mysql.conf`中的日志文件的配置。

6.数据库管理

本项目默认在`docker-compose.yml`中开启了用于MySQL在线管理的`phpMyAdmin`，以及用于redis在线管理的`phpRedisAdmin`，可以根据需要修改或删除。

6.1 phpMyAdmin

`phpMyAdmin`容器映射到主机的端口地址是：`8080`，所以主机上访问`phpMyAdmin`的地址是：

```
1 http://localhost:8080
```

MySQL连接信息：

- host: (本项目的MySQL容器网络)
- port: `3306`
- username: (手动在`phpmyadmin`界面输入)
- password: (手动在`phpmyadmin`界面输入)

6.2 phpRedisAdmin

`phpRedisAdmin`容器映射到主机的端口地址是：`8081`，所以主机上访问`phpMyAdmin`的地址是：

```
1 http://localhost:8081
```

Redis连接信息如下：

- host: (本项目的Redis容器网络)
- port: `6379`

7.在正式环境中安全使用

要在正式环境中使用，请：

1. 在php.ini中关闭XDebug调试
2. 增强MySQL数据库访问的安全策略
3. 增强redis访问的安全策略

8 常见问题

8.1 如何在PHP代码中使用curl?

参考这个issue: <https://github.com/yeszao/dnmp/issues/91>

8.2 Docker使用cron定时任务

[Docker使用cron定时任务](#)

8.3 Docker容器时间

容器时间在.env文件中配置TZ变量，所有支持的时区请看[时区列表·维基百科](#)或者[PHP所支持的时区列表·PHP官网](#)。

8.4 如何连接MySQL和Redis服务器

这要分两种情况，

第一种情况，在PHP代码中。

```
1 // 连接MySQL
2 $dbh = new PDO('mysql:host=mysql;dbname=mysql', 'root', '123456');
3
4 // 连接Redis
5 $redis = new Redis();
6 $redis->connect('redis', 6379);
```

因为容器与容器是expose端口联通的，而且在同一个networks下，所以连接的host参数直接用容器名称，port参数就是容器内部的端口。更多请参考[《docker-compose ports和expose的区别》](#)。

第二种情况，在主机中通过命令行或者Navicat等工具连接。主机要连接mysql和redis的话，要求容器必须经过ports把端口映射到主机了。以mysql为例，docker-compose.yml文件中有这样的ports配置：3306:3306，就是主机的3306和容器的3306端口形成了映射，所以我们这样连接：

```
1 $ mysql -h127.0.0.1 -uroot -p123456 -P3306
2 $ redis-cli -h127.0.0.1
```

这里host参数不能用localhost是因为它默认是通过sock文件与mysql通信，而容器与主机文件系统已经隔离，所以需要通过TCP方式连接，所以需要指定IP。

License

MIT

deployer 项目部署介绍

介绍

Deployer 是一个基于 SSH 协议的无侵入 web 项目部署工具，因为它不需要你在目标 服务器 上装什么服务之类的东西即可使用，它只需要在你的开发机，或者你的笔记本，就是发起部署动作的一方安装即可。

它的原理就是通过 SSH 到你的机器去创建目录，移动文件，执行指定的动作来完成项目的部署。

Deployer 不但可以部署php项目，也可以部署其他语言醒目

deployer 的优势

- 1 真正解放双手，一条命令完成部署。
- 2 进行部署的过程中，项目仍然能够正常访问。部署成功完成后才切到新的版本。
- 3 能十分方便地进行回滚。
- 4 丰富任务钩子和预置任务可灵活的组合完成各种任务，比如执行前端依赖的安装、构建等。

使用 deployer 的前提条件

- 1 本地机器（也就是你执行 dep 命令时所在的机器）能够 SSH 连接到目标机器
- 2 指定用户有登录目标机器并调整一些设置的权限
- 3 目标主机有拉取项目仓库的权限

使用说明

1、安装

```
1 curl -L0 https://deployer.org/deployer.phar
2 mv deployer.phar /usr/local/bin/dep
3 chmod +x /usr/local/bin/dep
```

2、生成部署脚本代码

切换到要部署的项目目录下，执行以下命令

```
1 dep init
```

此时会在改目录下生成 deploy.php文件

3、使用说明

未便于管理最好再创建文件deploy.config.php文件，提出公共配置，便于维护部署脚本。

实例：

以deploy部署go语言项目为例：

deploy.config.php

```
1 <?php
2
3 return [
4     'application' => 'miner-proxy', # 进程名称
5     'git' => 'git@bitbucket.org:miner/miner-proxy.git',
6     'branch' => 'master',
7     'deploy_path' => '/alidata/deploy/miner-proxy',
8     'servers' => [
9         'proxy0' => '192.134.45.45',
10        'proxy1' => '192.45.67.8',
11        'proxy2' => '193.168.45.67',
12    ],
13    'rsync' => [
14        'rsync -avz /alidata/deploy/miner-proxy/current/build/bin/*'
15        'public@{ip}:/alidata/service/miner-proxy/',
16        'rsync -avz /alidata/deploy/miner-proxy/current/build/superv
17        isord.conf public@{ip}:/alidata/service/miner-proxy/'
18    ]
19];
20];
```

deploy.php

```
1 <?php
2 namespace Deployer;
3
4 require 'recipe/common.php';
5 $config = require 'deploy.config.php';
6
7 // 项目名
8 set('application', $config['application']);
9
10 // 项目仓库地址
11 set('repository', $config['git']);
12
13 // [Optional] Allocate tty for git clone. Default value is false.
14 set('git_tty', true);
15
16 // Shared files/dirs between deploys
17 set('shared_files', []);
18 set('shared_dirs', []);
19
20 // Writable dirs by web server
21 set('writable_dirs', []);
22
23 set('keep_releases', 5);
24
25 $stage = $_SERVER['argv'][2];
26 if(!array_key_exists($stage, $config['servers'])){
27     exit('stage ' . $stage . ' 不存在');
28 }
29
30 # 目标主机配置
31 # 192.168.45.67 机器作为程序构建机器
```

```

32 # 不要用root用户，新创建一个普通用户用作部署，在部署之前设置好改账号的各个目录权限
33 host('192.168.45.67')
34     ->user('test')
35     ->set('branch', $config['branch']);
36     ->stage($stage)
37     ->set('deploy_path', $config['deploy_path']);
38
39 // Tasks
40 desc('Deploy super-miner-proxy project');
41 task('deploy', [
42     'deploy:info',
43     'deploy:prepare',
44     'deploy:lock',
45     'deploy:release',
46     'deploy:update_code',
47     'deploy:shared',
48     'deploy:writable',
49     'deploy:clear_paths',
50     'deploy:symlink',
51     'deploy:unlock',
52     'cleanup',
53     'success'
54 ]);
55
56 # 自定义任务
57 # 构建任务
58 task('make', function () use ($config, $stage) {
59     run('cd ' . $config['deploy_path'] . '/current && (export PATH=
$PATH:/usr/local/go/bin;make)');
60 });
61
62 # 打包同步任务，最好先压缩再同步
63 task('rsync', function () use ($config, $stage){
64     foreach ($config['rsync'] as $val){
65         $val = str_replace('{ip}', $config['servers'][$stage], $va
l);
66         run($val);
67     }
68 });
69
70 # 可执行程序同步完成后重启进程
71 # 对于需要sudo操作的命令，可以在指定机器上设置该命令免密
72 task('restart', function () use ($config, $stage) {
73     run('ssh test@' . $config['servers'][$stage] . ' "sudo /usr/bi
n/supervisorctl restart ' . $config['application'] . '"');
74 });
75
76
77 # 如果部署失败，自动解除部署锁定状态，以免影响下次执行
78 after('deploy:failed', 'deploy:unlock');
79 # 部署
80 after('deploy', 'make');
81 # 回退
82 after('rollback', 'make');
83 # 同步
84 after('make', 'rsync');
85 # 重启
86 after('rsync', 'restart');

```

注意

对于 `deploy:update_code` 使用 Git 下载新版本的代码。如果您使用的是 Git 2.0 版，并且 `git_cache` config 已打开，则此任务将使用先前发行版中的文件，因此仅下载更改的文件。否则重新 clone 代码。

执行部署命令

```
1 部署:  
2 dep deploy {stage} -vvv  
3  
4 回退:  
5 dep rollback {stage} -vvv
```

当你第一次成功部署的时候， Deployer 会自动帮你在服务器上生成一下文件：

```
1 releases 包含你部署项目的版本（默认保留 5 个版本）  
2  
3 shared 包含你部署项目的共享文件或目录（如： Laravel 的 Storage 目录、.env 文件等  
）  
4  
5 current 软连接到你当前发布的版本
```

官方文档地址

其他设置点可以通过以下网址查看

<https://deployer.org/docs/getting-started>

<https://github.com/DeployPHP/Deployer>

v2ray 搭建

推荐 v2ray 面板安装，便于管理。

面板： v2-ui

安装v2-ui面板会默认安装v2ray和控制web页面,根据安装成功后的提示使用即可。

GitHub地址：<https://github.com/sprov065/v2-ui>

Mac客户端使用推荐： v2rayU

github地址：<https://github.com/yanue/V2rayU>

以上工具使用说明见各自github介绍即可

Linux 系统磁盘挂载

随着业务发展，当我们服务器磁盘资源不足时，我们会选择挂接一块数据盘到Linux上，挂载步骤如下：

本示例以阿里云服务，挂接一块新的20 GiB数据盘（设备名为/dev/vdb）创建一个单分区，分区格式使用MBR，挂载的是ext4文件系统。

本文操作仅适用小于等于2 TiB的数据盘。

1. 运行`fdisk -l`命令查看实例上的数据盘。

[scode type="yellow"]说明：执行命令后，如果不存在/dev/vdb，表示您的实例没有数据盘。确认数据盘是否已挂载。[/scode]

2. 依次运行以下命令，创建一个单分区数据盘。

```
1 1. 运行`fdisk -u /dev/vdb`命令：分区数据盘。
2 2. 输入p：查看数据盘的分区情况。本示例中，数据盘没有分区。
3 3. 输入n：创建一个新分区。
4 4. 输入p：选择分区类型为主分区。
5
6 说明：本示例中创建一个单分区数据盘，所以只需要创建主分区。如果要创建四个以上分区，您应该创建至少一个扩展分区，即选择e (extended)。
7
8 5. 输入分区编号并按回车键。本示例中，仅创建一个分区，输入1。
9 6. 输入第一个可用的扇区编号：按回车键采用默认值2048。
10 7. 输入最后一个扇区编号。本示例中，仅创建一个分区，按回车键采用默认值。
11 8. 输入p：查看该数据盘的规划分区情况。
12 9. 输入w：开始分区，并在完成分区后退出。
```

3. 运行`fdisk -lu /dev/vdb`命令查看新分区。如果出现以下信息，表示新分区已创建完成。

```
1 [root@ecshost~ ]# fdisk -lu /dev/vdb
2
3 Disk /dev/vdb: 21.5 GB, 21474836480 bytes, 41943040 sectors
4 Units = sectors of 1 * 512 = 512 bytes
5 Sector size (logical/physical): 512 bytes / 512 bytes
6 I/O size (minimum/optimal): 512 bytes / 512 bytes
7 Disk label type: dos
8 Disk identifier: 0x3e60020e
9
10 Device Boot Start End Blocks Id System
11 /dev/vdb1 2048 41943039 20970496 83 Linux
```

4. 运行`mkfs.ext4 /dev/vdb1`命令在新分区上创建一个文件系统。

本示例中，创建一个ext4文件系统。您也可以根据自己的需要，选择创建其他文件系统。例如：如果您需要在Linux、Windows和Mac系统之间共享文件，可以运行`mkfs.vfat`命令创建VFAT文件系统。

创建文件系统所需时间取决于数据盘大小。

```
1 [root@ecshost~ ]# mkfs.ext4 /dev/vdb1
2
3 mke2fs 1.42.9 (28-Dec-2013)
4 Filesystem label=
5 OS type: Linux
6 Block size=4096 (log=2)
7 Fragment size=4096 (log=2)
8 Stride=0 blocks, Stripe width=0 blocks
9 1310720 inodes, 5242624 blocks
10 262131 blocks (5.00%) reserved for the super user
11 First data block=0
12 Maximum filesystem blocks=2153775104
13 160 block groups
14 32768 blocks per group, 32768 fragments per group
15 8192 inodes per group
16 Superblock backups stored on blocks:
17     32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 26
      54208,
18     4096000
19
20 Allocating group tables: done
21 Writing inode tables: done
22 Creating journal (32768 blocks): done
23 Writing superblocks and filesystem accounting information: done
```

5. 运行`cp /etc/fstab /etc/fstab.bak`命令备份`/etc/fstab`文件。

6. 运行`echo /dev/vdb1 /mnt ext4 defaults 0 0 >> /etc/fstab`命令向`/etc/fstab`写入新分区信息。

[scode type="yellow"]如要把数据盘单独挂载到某个文件夹，例如单独用来存放网页，则将命令中`/mnt`替换成所需的挂载点路径。[/scode]

[scode type="yellow"]Ubuntu 12.04系统不支持barrier，您需要运行`echo '/dev/vdb1 /mnt ext4 barrier=0 0 0' >> /etc/fstab`命令。[/scode]

7. 运行`cat /etc/fstab`命令查看`/etc/fstab`中的新分区信息。

```
1 ``
2 [root@ecshost~ ]# cat /etc/fstab
3 #
4 # /etc/fstab
5 # Created by anaconda on Wed Dec 12 07:53:08 2018
6 #
7 # Accessible filesystems, by reference, are maintained under '/dev/d
     isk'
8 # See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for mo
     re info
9 #
10 UUID=d67c3b17-255b-4687-be04-f29190d37396 / ext4 defaults 1 1
11 /dev/vdb1 /mnt ext4 defaults 0 0
12 ``
```

8. 运行`mount /dev/vdb1 /mnt`命令挂载文件系统。

9. 运行`df -h`命令查看目前磁盘空间和使用情况。

如果出现新建文件系统的信息，表示挂载成功，您不需要重启实例即可以使用新的文件系统。

```
1 [root@ecshost~ ]# df -h
2
3 Filesystem Size Used Avail Use% Mounted on
4 /dev/vda1 40G 1.6G 36G 5% /
5 devtmpfs 234M 0 234M 0% /dev
6 tmpfs 244M 0 244M 0% /dev/shm
7 tmpfs 244M 484K 244M 1% /run
8 tmpfs 244M 0 244M 0% /sys/fs/cgroup
9 tmpfs 49M 0 49M 0% /run/user/0
10 /dev/vdb1 20G 45M 19G 1% /mnt
```

创建自定义命令

如创建ll命令

步骤：

1、在home文件夹下创建.profile文件

```
1 vim ~/.profile
```

2、在.profile文件中写入

```
1 alias ll="ls -alh"
```

3、然后执行

```
1 source ~/.profile
```

原理：

mac 启动时会自动加载文件 `~/.bash.profile` 文件，并执行该文件下的命令：

在`~/.bash.profile`文件中添加

```
1 source ~/.profile
```

Linux 网络常见报错及监控项

查看服务器丢包

操作系统处理不过来，丢弃数据

有两种情况，一是网卡发现操作系统处理不过来，丢数据包，可以读取下面的文件：

```
1 $ cat /proc/net/dev  
2 每个网络接口一行统计数据，第 4 列 (errs) 是接收出错的数据包数量，第 5 列 (drop) 是  
接收不过来丢弃的数量。
```

第二部分是传统非 NAPI 接口实现的网卡驱动，每个 CPU 有一个队列，当在队列中缓存的数据包数量超过 net.core.netdev_max_backlog 时，网卡驱动程序会丢掉数据包，这个见下面的文件：

```
1 $ cat /proc/net/softnet_stat  
2 每个 CPU 有一行统计数据，第二列是对应 CPU 丢弃的数据包数量。
```

应用程序处理不过来，操作系统丢弃

内核中记录了两个计数器：

ListenOverflows：当 socket 的 listen queue 已满，当新增一个连接请求时，应用程序来不及处理；

ListenDrops：包含上面的情况，除此之外，当内存不够无法为新的连接分配 socket 相关的数据结构时，也会加 1，当然还有别的异常情况下会增加 1。

分别对应下面文件中的第 21 列 (ListenOverflows) 和第 22 列 (ListenDrops)，可以通过如下命令查看：

```
1 $ cat /proc/net/netstat | awk '/TcpExt/ { print $21,$22 }'  
2 如果使用 netstat 命令，有丢包时会看到 “times the listen queue of a socket o  
verflowed” 以及 “SYNs to LISTEN sockets ignored” 对应行前面的数字；如果值为  
0 则不会输出对应的行。
```

Out of memory, 内存不足

出现内存不足可能有两种情况：

有太多的 orphan sockets，通常对于一些前端的服务器经常会出现这种情况。

分配给 TCP 的内存确实较少，从而导致内存不足。

内存不足

这个比较好排查，只需要查看一下实际分配给 TCP 多少内存，现在用了多少内存即可。需要注意的是，通常的配置项使用的单位是 Bytes，在此用的是 Pages，通常为 4K。

先查看下给 TCP 分配了多少内存。

```
1 $ cat /proc/sys/net/ipv4/tcp_mem
2 183474 244633 366948
3 简单来说，三个值分别表示进入 无压力、压力模式、内存上限的值，当到达最后一个值的时候就
会报错。
```

接下来查看一下当前使用的内存。

```
1 $ cat /proc/net/sockstat
2 sockets: used 855
3 TCP: inuse 17 orphan 1 tw 0 alloc 19 mem 3
4 UDP: inuse 16 mem 10
5 UDPLITE: inuse 0
6 RAW: inuse 1
7 FRAG: inuse 0 memory 0
8 其中的 mem 表示使用了多少 Pages，如果相比 tcp_mem 的配置来说还很小，那么就有可能
是由于 orphan sockets 导致的。
```

orphan sockets

首先介绍一下什么是 orphan sockets，简单来说就是该 socket 不与任何一个文件描述符相关联。例如，当应用调用 close() 关闭一个链接时，此时该 socket 就成为了 orphan，但是该 sock 仍然会保留一段时间，直到最后根据 TCP 协议结束。

实际上 orphan socket 对于应用来说是无用的，因此内核希望尽可能减小 orphan 的数量。对于像 http 这样的短请求来说，出现 orphan 的概率会比较大。

对于系统允许的最大 orphan 数量，以及当前的 orphan 数量可以通过如下方式查看：

```
1 $ cat /proc/sys/net/ipv4/tcp_max_orphans
2 32768
3 $ cat /proc/net/sockstat
4 16-05-30 14:11
5 TCP: inuse 37 orphan 14 tw 8 alloc 39 mem 9
6 ... ...
```

你可能会发现，sockstat 中的 orphan 数量要远小于 tcp_max_orphans 的数目。

实际上，可以从代码中看到，实际会有个偏移量 shift，该值范围为 [0, 2]。

也就是说，在某些场景下会对 orphan 做些惩罚，将 orphan 的数量 2x 甚至 4x，这也解释了上述的问题。

如果是这样，那么就可以根据具体的情况，将 tcp_max_orphans 值适当调大。

es配置

es配置文件 /etc/elasticsearch/elasticsearch.yml

```
1 network.host: 0.0.0.0 # 允许外网访问, 若不允许可不改
2
3 discovery.seed_hosts: ["127.0.0.1", "[::1]"]
4
5 # 开启跨域访问支持, 默认为false
6 http.cors.enabled: true
7 # 跨域访问允许的域名地址, (允许所有域名)以上使用正则
8 http.cors.allow-origin: /*
```

Elasticsearch 术语

Elasticsearch几个重要的概念

集群

集群中的一个节点就是一个 Elasticsearch 进程，多个节点组成一个集群

一般每个节点都运行在不同的操作系统上，配置好集群相关参数后 Elasticsearch 会自动组成集群（节点发现方式也可以配置）

集群内部通过 Elasticsearch 的选主算法选出主节点，而集群外部则是可以通过任何节点进行操作，无主从节点之分（对外表现对等/去中心化，有利于客户端编程，例如故障重连）

索引

「索引」有两个意思：

1. 作为动词，它指的是把一个文档「保存」到 Elasticsearch 中的过程，索引一个文档后，我们就可以使用 Elasticsearch 搜索到这个文档
2. 作为名词，它是指保存文档的地方，相当于一个数据库概念中的「库」

为了方便理解，我们可以将 Elasticsearch 中的一些概念对应到我们熟悉的关系型数据库上

Elasticsearch	索引	类型	文档
DB	库	表	行

分片

Elasticsearch 是一个分布式系统，我们一开始就应该以集群的方式来使用它。

Elasticsearch 在保存索引时会选择适合的「主分片」（Primary Shard），把索引保存到其中

我们可以把分片理解为一块物理存储区域

分片的分法是固定的，而且是安装时候就必须要决定好的（默认是 5），后面就不能改变了

既然有主分片，那肯定是有「从」分片的，但 Elasticsearch 里称之为「副本分片」（Replica Shard）

副本分片主要有两个作用：

1. 高可用

某分片节点挂了的话可走其他副本分片节点，节点恢复后上面的分片数据可通过其他节点恢复

2. 负载均衡

Elasticsearch 会自动根据负载情况控制搜索路由，副本分片可以将负载均摊

范例

我们举一个简单的例子来总结上面阐述的内容

index	shard	prirep	state	docs	store	ip	node
index1	2	p	STARTED	1	2.7kb	10.180.120.58	es-58
index1	2	r	STARTED	1	2.7kb	10.180.120.59	es-59
index1	0	r	STARTED	0	87b	10.180.120.58	es-58
index1	0	p	STARTED	0	123b	10.180.120.59	es-59
index1	3	r	STARTED	0	87b	10.180.120.60	es-60
index1	3	p	STARTED	0	123b	10.180.120.59	es-59
index1	1	p	STARTED	0	123b	10.180.120.60	es-60
index1	1	r	STARTED	0	87b	10.180.120.59	es-59
index1	4	r	STARTED	0	87b	10.180.120.58	es-58
index1	4	p	STARTED	0	123b	10.180.120.60	es-60
index2	4	p	STARTED	0	123b	10.180.120.60	es-60
index2	4	r	STARTED	0	87b	10.180.120.59	es-59
index2	0	p	STARTED	0	123b	10.180.120.58	es-58
index2	0	r	STARTED	0	87b	10.180.120.60	es-60
index2	3	p	STARTED	0	123b	10.180.120.58	es-58
index2	3	r	STARTED	0	87b	10.180.120.60	es-60
index2	1	p	STARTED	0	123b	10.180.120.60	es-60
index2	1	r	STARTED	0	87b	10.180.120.59	es-59
index2	2	r	STARTED	1	2.7kb	10.180.120.58	es-58
index2	2	p	STARTED	1	2.7kb	10.180.120.59	es-59

该图是使用 Elasticsearch 的 RESTful 接口获取的，后面会介绍常用接口

上图中：

1. 3 个 Elasticsearch 节点（es-58/59/60）组成一个集群
2. 建立集群时使用默认的主分片数 5，shard0 ~ shard4
3. 该集群内有加入两个索引 index1、index2
4. 这两个索引中分别「索引」（保存）了两个文档
5. index1 索引中这个文档被 Elasticsearch 自动保存到了分片 2 中，主分片在 es-58 节点，副本分片在 es-59 节点

6. index2 索引中这个文档被 Elasticsearch 自动保存到了分片 2 中，主分片在 es-59 节点，副本分片在 es-58 节点

多租户

Elasticsearch 中的多租户简单的说就是通过多索引机制同时提供给多种业务使用，每种业务使用一个索引（关于多租户的详细定义与用途，可以参考[这里](#)）

前面我们提到过可以把索引理解为关系型数据库里的库，那多索引可以理解为一个数据库系统建立多个库给不同的业务使用

实际使用时，我们可以通过每个租户一个索引的方式将他们的数据进行隔离，并且每个索引是可以单独配置参数的（可对特定租户进行调优）

这在典型的多租户场景下非常有用：例如我们的一个多租户应用需要提供搜索支持，这时可以通过 Elasticsearch 根据租户建立索引，这样每个租户就可以在自己的索引下搜索相关内容了

RESTful

因为 RESTful，所以 Elasticsearch 接口非常清晰方便简单。

最关键的是 Elasticsearch 的 HTTP 接口不只是可以进行业务操作（索引/搜索），还可以进行配置，甚至是关闭 Elasticsearch 集群

下面我们介绍几个很常用的接口

接口	说明
/_cat/nodes?v	查看集群状态
/_cat/shards?v	查看分片状态
/\$	搜索

1. `v` 是 verbose 的意思，当使用这个参数，那么返回结果则更具可读性（有表头，有对齐）
2. `_cat` 是监测相关的 APIs。可以使用 `/_cat?help` 来获取所有接口
3. `${index}` 和 `${type}` 分别是具体的某一索引某一类型，是分层次的

Elasticsearch与数据库比较，理解各个概念：

Elasticsearch 中，索引是一个集合，相当于数据库（RDBMS，关系数据库管理系统）中的数据库

集合中的每个映射，相当于 RDBMS 中的表

而索引中的每个 JSON 对象，又相当于 RDBMS 中的数据行

所以，Elasticsearch，集合是一些包含了 JSON 对象的映射的集合

详细的对比如下

Elasticsearch	RDBMS
Index	Database
Shard	Shard
Mapping	Table
Field	Field
JSON Object	Tuple

术语

接下来的教程会涉及到很多术语，这些术语可能比较难懂，不过没关系，你不懂也可以继续，当不懂某个术语的时候，回来这篇文章看看就知道了

analysis 分析

分析是将文本（text）转化为查询词（term）的过程

比如这三种短语：FOO BAR, Foo-Bar, foo,bar 都有可能被分解成查询词 foo 与 bar

可以使用不同的分析器，这些查询词实际上将被存储在索引中

一次对 FoO:bAR 的全文查询（不是查询词查询）可能会被分析为为查询词 foo,bar，可以匹配上保存在索引中的查询词

这就是分析处理过程（包含了索引与搜索），它使得 es 可以进行全文查询

cluster (集群)

一个或多个拥有同一个集群名称的节点组成了一个集群

每个集群都会自动选出一个主节点，如果该主节点故障，则集群会自动选出新的主节点来替换故障节点

document (文档)

一个文档就是一个保存在 Elasticsearch 中的 JSON 文本，可以把它理解为关系型数据库表中的一行

每个文档都是保存在索引中的，拥有一种类型和 id

一个文档是一个 JSON 对象（一些语言中的 hash / hashmap / associative array）包含了 0 或多个字段（键值对）

原始的 JSON 文本在索引后将被保存在 `_source` 字段里，搜索完成后返回值中默认是包含该字段的

id

Id 是用于标识文档的，一个文档的索引/类型/id 必须是唯一的

文档 id 是自动生成的，如果显式不指定

field (字段)

一个文档包含了若干字段，或称之为键值对

字段的值可以是简单标量值，例如字符串、整型、日期，也可以是嵌套结构，例如数组或对象

一个字段类似于关系型数据库表中的一列

每个字段的映射都有一个字段类型（不要和文档类型搞混了），它描述了这个字段可以保存的值类型，例如整型、字符串、对象

映射还可以让我们定义一个字段的值如何进行分析

index (索引)

一个索引类似关系型数据库中的一个数据库，它可以映射为多种类型

一个索引就是逻辑上的一个命名空间，对应到 1 或多个主分片上，可以拥有 0 个或多个副本分片

mapping (映射)

一个映射类似于关系型数据库中的表

每个索引都存在一个映射，它定义了该索引中的每一种类型，以及索引相关的配置

映射可以显示定义，或者在文档被索引时自动创建

node (节点)

一个节点是集群中的一个 Elasticsearch 运行实例

测试环境，多个节点可以同时启在同一个服务器上，生产环境一般是一个服务器上一个节点

节点启动时将使用单播或者是组播来发现和自己配置的集群名称相同的集群，并尝试加入到该集群中

shard (分片)

一个分片就是一个 Lucene 实例，它是 Elasticsearch 管理的底层「工作单元」

一个索引是逻辑上的一个命名空间，指向主分片和副本分片

索引的主分片和副本分片数量必须明确指定好，在应用代码使用时只需要处理和索引的交互，不会涉及到和分片的交互

Elasticsearch 会在集群中的所有节点上设置好分片，但节点失效或加入新节点时会自动将移动节点分片

primary shard (主分片)

每个文档都会被保存在一个主分片上

当我们索引一个文档时，它将在一个主分片上进行索引，然后才放到该主分片的各副本分片上

默认情况下，一个索引有 5 个主分片

我们可以指定更少或更多的主分片来伸缩索引可处理的文档数

需要注意的是，一旦索引创建，就不能修改主分片个数

replica shard (副本分片)

每个主分片可以拥有 0 个或多个副本分片，一个副本分片是主分片的一份拷贝

这样做有两个主要原因：

1. 故障转移

当主分片失效时，一个副本分片会被提升为主分片

2. 提高性能

获取与搜索请求可以被主分片或副本分片处理

默认情况下，每个主分片都有一个副本分片，副本分片的数量可以动态调整

在同一个节点上，副本分片和其主分片不会同时运行

routing (路由)

当我们索引一个文档时，它将被保存在一个主分片上，分片的选择是通过路由值哈希得到的
默认情况下，路由值使用文档的 id，如果该文档指定了父文档，则路由值使用父文档 id
这是为了确保子文档和父文档被保存在相同的分片上
该值可以在索引时指定，也可以通过映射路由字段来指定

source field (源字段)

默认情况下，在获取和搜索请求返回值中的 `_source` 字段保存了源 JSON 文本
这使得我们可以直接在返回结果中访问源数据，而不需要根据 id 再发一次检索请求

索引的 JSON 字符串将完整返回，无论是否是一个合法的 JSON
该字段的内容也不会描述数据如何被索引

term (查询词)

一个查询词是一个被 Elasticsearch 索引的确切值
查询词 foo, Foo, FOO 是不同的
查询词可以使用查询词查询接口进行获取

text (文本)

文本（或称之为全文）是普通的、非结构化的文本，例如一句话。
默认情况下，文本将被分解为查询词，查询词将被保存在索引中
为能够进行全文搜索，文本字段在索引时将被分解为查询词，查询关键字在搜索时也将被分解为查询词，
通过对比查询词是否相同而完成全文搜索

type (类型)

一种类似于关系型数据库中的一张表的类型
每种类型都有若干字段，用于指定给该类型文档
映射定义了该文档中的每个字段如何进行分析

ES介绍和安装

Elasticsearch 是一个实时的分布式搜索分析引擎。它被用作全文检索、结构化搜索、分析以及这三个功能的组合。他建立在一个全文搜索引擎库 Apache Lucene™ 基础之上。 Elasticsearch 也是使用 Java 编写的，它的内部使用 Lucene 做索引与搜索，但是它的目的是使全文检索变得简单，通过隐藏 Lucene 的复杂性，取而代之的提供一套简单一致的 RESTful API。

Elasticsearch:

- 1 一个分布式的实时文档存储，每个字段 可以被索引与搜索
- 2 一个分布式实时分析搜索引擎
- 3 能胜任上百个服务节点的扩展，并支持 PB 级别的结构化或者非结构化数据

Elasticsearch 将所有的功能打包成一个单独的服务，这样你可以通过程序与它提供的简单的 RESTful API 进行通信，可以使用自己喜欢的编程语言充当 web 客户端，甚至可以使用命令行（去充当这个客户端）。

安装并运行

安装 Elasticsearch 之前，你需要先安装一个较新的版本的 Java，最好的选择是，你可以从 [oracle官网](#) 获得官方提供的最新版本的 Java。

安装方法参考：<https://www.yuque.com/u316337/ce14e5/goxi7c>

之后，你可以从 [elastic 的官网](#) 获取最新版本的 Elasticsearch。

提示

当你准备在生产环境安装 Elasticsearch 时，你可以在 官网下载地址 找到 Debian 或者 RPM 包，除此之外，你也可以使用官方支持的 Puppet module 或者 Chef cookbook。

按照下面的操作，在前台启动 Elasticsearch：

```
1 wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.6.2.tar.gz
2
3 tar -zvxf elasticsearch-6.6.2.tar.gz
4
5 cd elasticsearch-6.6.2
6
7 ./bin/elasticsearch
```

如果你想把 Elasticsearch 作为一个守护进程在后台运行，那么可以在后面添加参数 -d。

测试 Elasticsearch 是否启动成功，可以打开另一个终端，执行以下操作：

```
1 curl 'http://localhost:9200/?pretty'
```

你应该得到和下面类似的响应(response)：

```
1 {
2   "name" : "Tom Foster",
3   "cluster_name" : "elasticsearch",
4   "version" : {
5     "number" : "2.1.0",
6     "build_hash" : "72cd1f1a3eee09505e036106146dc1949dc5dc87",
7     "build_timestamp" : "2015-11-18T22:40:03Z",
8     "build_snapshot" : false,
9     "lucene_version" : "5.3.1"
10  },
11  "tagline" : "You Know, for Search"
12 }
```

这就意味着你现在已经启动并运行一个 Elasticsearch 节点了，你可以用它做实验了。单个 节点 可以作为一个运行中的 Elasticsearch 的实例。而一个 集群 是一组拥有相同 cluster.name 的节点，他们能一起工作并共享数据，还提供容错与可伸缩性。(当然，一个单独的节点也可以组成一个集群) 你可以在 elasticsearch.yml 配置文件中 修改 cluster.name ，该文件会在节点启动时加载(译者注：这个重启服务后才会生效)。

默认情况下，Elastic 只允许本机访问，如果需要远程访问，可以修改 Elastic 安装目录的 config/elasticsearch.yml 文件，去掉 network.host 的注释，将它的值改成 0.0.0.0，然后重新启动 Elastic。

```
1 network.host: 0.0.0.0
```

上面代码中，设成 0.0.0.0 让任何人都可以访问。线上服务不要这样设置，要设成具体的 IP。

停止 Elasticsearch

如果是开发环境，停止就直接使用 **CTRL + C** 组合键吧

然后会继续输出以下信息

```
1 [2020-03-07T10:57:47,907] [INFO ] [o.e.x.m.p.NativeController] [c5ce5ac
c533d] Native controller process has stopped - no new native processes can be started
2 [2020-03-07T10:57:47,911] [INFO ] [o.e.n.Node] [c5ce5acc
533d] stopping ...
3 [2020-03-07T10:57:47,939] [INFO ] [o.e.x.w.WatcherService] [c5ce5acc
533d] stopping watch service, reason [shutdown initiated]
4 [2020-03-07T10:57:47,952] [INFO ] [o.e.x.w.WatcherLifeCycleService] [c5
ce5acc533d] watcher has stopped and shutdown
5 [2020-03-07T10:57:48,348] [INFO ] [o.e.n.Node] [c5ce5acc
533d] stopped
6 [2020-03-07T10:57:48,350] [INFO ] [o.e.n.Node] [c5ce5acc
533d] closing ...
7 [2020-03-07T10:57:48,374] [INFO ] [o.e.n.Node] [c5ce5acc
533d] closed
```

centos系统， rpm包方式安装

```
1 rpm -ivh elasticsearch-7.6.1-x86_64.rpm
2
3 #配置环境变量
```

```
4 vi ~/.bashrc
5 然后在文件末尾添加以下语句
6
7 export PATH="$PATH:/usr/share/elasticsearch/bin"
8
9 # 启动es服务
10 elasticsearch
```

Linux 安装java环境

1、下载jdk安装包，以jdk13为例

下载地址：

<https://www.oracle.com/java/technologies/javase-jdk13-downloads.html>

以centos为例，下载rpm包

2、安装jdk

```
1 rpm -ivh jdk-13.0.2_linux-x64_bin.rpm
```

rpm安装方式默认安装在/usr/java目录下，不用配置环境变量

3、验证

```
1 java -version # 显示java版本号
```

Kibana 汉化方法

项目地址：

https://github.com/anbai-inc/Kibana_Hanization

汉化方法

The screenshot shows a GitHub repository page for 'Kibana 6.x - Future 汉化项目'. The repository has 4 commits from 'RedFree' over 10 days ago. The README.md file contains instructions for localization:

Kibana 6.x - Future 汉化项目

Kibana官方在6.x(具体版本忘了)的版本正式推出了资源汉化方法(虽然有很多资源还不能被汉化)，因此我废弃掉了之前暴力搜索替换的汉化方法。

老版本的汉化仍然可以参考此项目old文件下的汉化说明进行汉化，将来不再进行老版本汉化的维护(< 6.0)。

此项汉化资源会慢慢更新(业余时间汉化)，若某些页面还没有汉化，请耐心等待。

一、汉化方法

- 1、拷贝此项目中的translations文件夹到您的kibana目录下的src/legacy/core_plugins/kibana/目录。若您的kibana无此目录，那还是尝试使用此项目old目录下的汉化方法吧。
- 2、修改您的kibana配置文件kibana.yml中的配置项：i18n.locale: "zh_CN"
- 3、重启Kibana，汉化完成

二、说明：

此方式汉化完整度依赖于官方在源代码中提供的汉化资源，所以些地方没有汉化我也是没有办法滴。
有问题请邮件联系：redfree@anbai.com (#=@)

Kibana 简介、安装与配置

简介

Kibana 是一款开源的数据分析和可视化平台，它是 Elastic Stack 成员之一，设计用于和 Elasticsearch 协作。您可以使用 Kibana 对 Elasticsearch 索引中的数据进行搜索、查看、交互操作。您可以很方便的利用图表、表格及地图对数据进行多元化的分析和呈现。

Kibana 可以使大数据通俗易懂。它很简单，基于浏览器的界面便于您快速创建和分享动态数据仪表板来追踪 Elasticsearch 的实时数据变化。

安装

注意：Kibana 的版本需要和 Elasticsearch 的版本一致。这是官方支持的配置。

从 [elastic 的官网](#) 获取最新版本的 Kibana，和 Elasticsearch 的版本一致。

```
1 wget https://artifacts.elastic.co/downloads/kibana/kibana-6.6.2-darwi
n-x86_64.tar.gz
2
3 tar -zxf kibana-6.6.2-darwin-x86_64.tar.gz
4
5 cd kibana-6.6.2-darwin-x86_64
6
7 ./bin/kibana
```

目录说明

.tar.gz 整个包是独立的。默认情况下，所有的文件和目录都在 \$KIBANA_HOME — 解压包时创建的目录下。这样非常方便，因为您不需要创建任何目录来使用 Kibana，卸载 Kibana 就是简单地删除 \$KIBANA_HOME 目录。但还是建议修改一下配置文件和数据目录，这样就不会删除重要数据。

```
1 bin 二进制脚本，包括 kibana 启动 Kibana 服务和 kibana-plugin 安装插件。
2
3 config 配置文件，包括 kibana.yml 。
4
5 data Kibana 和其插件写入磁盘的数据文件位置。
6
7 optimize 编译过的源码。某些管理操作（如，插件安装）导致运行时重新编译源码。
8
9 plugins 插件文件位置。每一个插件都有一个单独的二级目录。
```

配置文件

Kibana 默认情况下从 \$KIBANA_HOME/config/kibana.yml 加载配置文件。

配置项说明：

```
1 server.port:
2 默认值：5601 Kibana 由后端服务器提供服务，该配置指定使用的端口号。
3
```

```
4 server.host:  
5 默认值: "localhost" 指定后端服务器的主机地址。  
6  
7 server.basePath:  
8 如果启用了代理, 指定 Kibana 的路径, 该配置项只影响 Kibana 生成的 URLs, 转发请求  
到 Kibana 时代理会移除基础路径值, 该配置项不能以斜杠 (/) 结尾。  
9  
10 server.maxPayloadBytes:  
11 默认值: 1048576 服务器请求的最大负载, 单位字节。  
12  
13 server.name:  
14 默认值: "您的主机名" Kibana 实例对外展示的名称。  
15  
16 server.defaultRoute:  
17 默认值: "/app/kibana" Kibana 的默认路径, 该配置项可改变 Kibana 的登录页面。  
18  
19 elasticsearch.url:  
20 默认值: "http://localhost:9200" 用来处理所有查询的 Elasticsearch 实例的 U  
RL 。  
21  
22 elasticsearch.preserveHost:  
23 默认值: true 该设置项的值为 true 时, Kibana 使用 server.host 设定的主机名,  
该设置项的值为 false 时, Kibana 使用主机的主机名来连接 Kibana 实例。  
24  
25 kibana.index:  
26 默认值: ".kibana" Kibana 使用 Elasticsearch 中的索引来存储保存的检索, 可视  
化控件以及仪表板。如果没有索引, Kibana 会创建一个新的索引。  
27  
28 kibana.defaultAppId:  
29 默认值: "discover" 默认加载的应用。  
30  
31 tilemap.url:  
32 Kibana 用来在 tile 地图可视化组件中展示地图服务的 URL。默认时, Kibana 从外部的  
元数据服务读取 url, 用户也可以覆盖该参数, 使用自己的 tile 地图服务。例如: "http  
s://tiles.elastic.co/v2/default/{z}/{x}/{y}.png?elastic_tile_servic  
e_tos=agree&my_app_name=kibana"  
33  
34 tilemap.options.minZoom:  
35 默认值: 1 最小缩放级别。  
36  
37 tilemap.options.maxZoom:  
38 默认值: 10 最大缩放级别。  
39  
40 tilemap.options.attribution:  
41 默认值: "© [Elastic Tile Service](https://www.elastic.co/elastic-til  
e-service)" 地图属性字符串。  
42  
43 tilemap.options.subdomains:  
44 服务使用的二级域名列表, 用 {s} 指定二级域名的 URL 地址。  
45  
46 elasticsearch.username: 和 elasticsearch.password:  
47 Elasticsearch 设置了基本的权限认证, 该配置项提供了用户名和密码, 用于 Kibana 启  
动时维护索引。Kibana 用户仍需要 Elasticsearch 由 Kibana 服务端代理的认证。  
48  
49 server.ssl.enabled  
50 默认值: "false" 对到浏览器端的请求启用 SSL, 设为 true 时, server.ssl.certifi  
cate 和 server.ssl.key 也要设置。  
51  
52 server.ssl.certificate: 和 server.ssl.key:  
53 PEM 格式 SSL 证书和 SSL 密钥文件的路径。
```

```
54
55 server.ssl.keyPassphrase
56 解密私钥的口令，该设置项可选，因为密钥可能没有加密。
57
58 server.ssl.certificateAuthorities
59 可信任 PEM 编码的证书文件路径列表。
60
61 server.ssl.supportedProtocols
62 默认值：TLSv1、TLSv1.1、TLSv1.2 版本支持的协议，有效的协议类型：TLSv1 、 TL
Sv1.1 、 TLSv1.2 。
63
64 server.ssl.cipherSuites
65 默认值：ECDHE-RSA-AES128-GCM-SHA256, ECDHE-ECDSA-AES128-GCM-SHA256,
ECDHE-RSA-AES256-GCM-SHA384, ECDHE-ECDSA-AES256-GCM-SHA384, DHE-RS
A-AES128-GCM-SHA256, ECDHE-RSA-AES128-SHA256, DHE-RSA-AES128-SHA25
6, ECDHE-RSA-AES256-SHA384, DHE-RSA-AES256-SHA384, ECDHE-RSA-AES256
-SHA256, DHE-RSA-AES256-SHA256, HIGH,!aNULL, !eNULL, !EXPORT, !DES,
!RC4, !MD5, !PSK, !SRP, !CAMELLIA. 具体格式和有效参数可通过[OpenSSL ciph
er list format documentation](https://www.openssl.org/docs/man1.0.2/apps/ciphers.html#CIPHER-LIST-FORMAT) 获得。
66
67 elasticsearch.ssl.certificate: 和 elasticsearch.ssl.key:
68 可选配置项，提供 PEM 格式 SSL 证书和密钥文件的路径。这些文件确保 Elasticsearch
后端使用同样的密钥文件。
69
70 elasticsearch.ssl.keyPassphrase
71 解密私钥的口令，该设置项可选，因为密钥可能没有加密。
72
73 elasticsearch.ssl.certificateAuthorities:
74 指定用于 Elasticsearch 实例的 PEM 证书文件路径。
75
76 elasticsearch.ssl.verificationMode:
77 默认值：full 控制证书的认证，可用的值有 none 、 certificate 、 full 。 full
执行主机名验证，certificate 不执行主机名验证。
78
79 elasticsearch.pingTimeout:
80 默认值：elasticsearch.requestTimeout setting 的值，等待 Elasticsearch
的响应时间。
81
82 elasticsearch.requestTimeout:
83 默认值：30000 等待后端或 Elasticsearch 的响应时间，单位微秒，该值必须为正整
数。
84
85 elasticsearch.requestHeadersWhitelist:
86 默认值：[ 'authorization' ] Kibana 客户端发送到 Elasticsearch 头体，发送
no 头体，设置该值为 []。
87
88 elasticsearch.customHeaders:
89 默认值：{} 发往 Elasticsearch 的头体和值，不管 elasticsearch.requestHead
ersWhitelist 如何配置，任何自定义的头体不会被客户端头体覆盖。
90
91 elasticsearch.shardTimeout:
92 默认值：0 Elasticsearch 等待分片响应时间，单位微秒，0 即禁用。
93
94 elasticsearch.startupTimeout:
95 默认值：5000 Kibana 启动时等待 Elasticsearch 的时间，单位微秒。
96
97 pid.file:
98 指定 Kibana 的进程 ID 文件的路径。
99
```

```
100 logging.dest:  
101 默认值: stdout 指定 Kibana 日志输出的文件。  
102  
103 logging.silent:  
104 默认值: false 该值设为 true 时, 禁止所有日志输出。  
105  
106 logging.quiet:  
107 默认值: false 该值设为 true 时, 禁止除错误信息除外的所有日志输出。  
108  
109 logging.verbose  
110 默认值: false 该值设为 true 时, 记下所有事件包括系统使用信息和所有请求的日志。  
111  
112 ops.interval  
113 默认值: 5000 设置系统和进程取样间隔, 单位微妙, 最小值100。  
114  
115 status.allowAnonymous  
116 默认值: false 如果启用了权限, 该项设置为 true 即允许所有非授权用户访问 Kibana 服务端 API 和状态页面。  
117  
118 cpu.cgroup.path.override  
119 如果挂载点跟 /proc/self/cgroup 不一致, 覆盖 cgroup cpu 路径。  
120  
121 cpuacct.cgroup.path.override  
122 如果挂载点跟 /proc/self/cgroup 不一致, 覆盖 cgroup cpuacct 路径。  
123  
124 console.enabled  
125 默认值: true 设为 false 来禁用控制台, 切换该值后服务端下次启动时会重新生成资源文件, 因此会导致页面服务有点延迟。  
126  
127 elasticsearch.tribe.url:  
128 Elasticsearch tribe 实例的 URL, 用于所有查询。  
129  
130 elasticsearch.tribe.username: 和 elasticsearch.tribe.password:  
131 Elasticsearch 设置了基本的权限认证, 该配置项提供了用户名和密码, 用于 Kibana 启动时维护索引。Kibana 用户仍需要 Elasticsearch 由 Kibana 服务端代理的认证。  
132  
133 elasticsearch.tribe.ssl.certificate: 和 elasticsearch.tribe.ssl.key:  
134 可选配置项, 提供 PEM 格式 SSL 证书和密钥文件的路径。这些文件确保 Elasticsearch 后端使用同样的密钥文件。  
135  
136 elasticsearch.tribe.ssl.keyPassphrase  
137 解密私钥的口令, 该设置项可选, 因为密钥可能没有加密。  
138  
139 elasticsearch.tribe.ssl.certificateAuthorities:  
140 指定用于 Elasticsearch tribe 实例的 PEM 证书文件路径。  
141  
142 elasticsearch.tribe.ssl.verifyMode:  
143 默认值: full 控制证书的认证, 可用的值有 none 、 certificate 、 full 。 full 执行主机名验证, certificate 不执行主机名验证。  
144  
145 elasticsearch.tribe.pingTimeout:  
146 默认值: elasticsearch.tribe.requestTimeout setting 的值, 等待 Elasticsearch 的响应时间。  
147  
148 elasticsearch.tribe.requestTimeout:  
149 Default: 30000 等待后端或 Elasticsearch 的响应时间, 单位微妙, 该值必须为正整数。  
150  
151 elasticsearch.tribe.requestHeadersWhitelist:  
152 默认值: [ 'authorization' ] Kibana 发往 Elasticsearch 的客户端头体, 发送
```

```
no 头体，设置该值为 []。
153
154 elasticsearch.tribe.customHeaders:
155 默认值: {} 发往 Elasticsearch的头体和值，不管 elasticsearch.tribe.reques
tHeadersWhitelist 如何配置，任何自定义的头体不会被客户端头体覆盖。
```

访问 Kibana

Kibana 是一个 web 应用，可以通过5601端口访问。只需要在浏览器中指定 Kibana 运行的机器，然后指定端口号即可。例如，`localhost:5601` 或者 `http://YOURDOMAIN.com:5601`。

当访问 Kibana 时，Discover 页默认会加载默认的索引模式。时间过滤器设置的时间为过去15分钟，查询设置为匹配所有 (*)。

如果看不到任何文档，试着把时间过滤器的范围调大。如果还是看不到任何结果，很可能是根本就 没有 任
何文档。

检查 Kibana 状态

您可以通过 `localhost:5601/status` 来访问 Kibana 的服务器状态页，状态页展示了服务器资源使
用情况和已安装插件列表。

新增用户&赋予sudo权限

新增用户

在root用户下

```
1 $ useradd -m xxx  
2  
3 $ passwd xxx
```

赋予sudo权限

- a、进入超级用户模式。也就是输入"su root",系统会让你输入超级用户密码，输入密码后就进入了超级用户模式。
- b、添加配置文件的写权限。也就是输入命令"chmod u+w /etc/sudoers"。
- c、编辑/etc/sudoers文件。也就是输入命令"vim /etc/sudoers",输入"i"进入编辑模式，找到这一行："root ALL=(ALL) ALL"在起下面添加"xxx ALL=(ALL) ALL"(这里的xxx是你的用户名)，然后保存（就是先摁一下Esc键，然后输入":wq"）退出。
- d、撤销文件的写权限。也就是输入命令"chmod u-w /etc/sudoers"。

此时 xxx用户即有了sudo权限

配置指定命令sudo免密码使用

```
## Allow root to run any commands anywhere
root    ALL=(ALL)      ALL

## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys  ALL=NEWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE, DRIVERS

## Allows people in group wheel to run all commands
# %wheel    ALL=(ALL)      ALL

## Same thing without a password
# %wheel    NOPASSWD:ALL
hadoop    ALL=(ALL)      NOPASSWD: ALL

## Allows members of the users group to mount and umount the
## cdrom as root
# %users   ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom
```

在已赋予用户sudo权限的基础上，配置用户sudo执行指定命令时不需要输入密码：

位置必须在`#includedir /etc/sudoers.d`后面，添加在其他地方不生效

```
1 sudo chmod u+w /etc/sudoers
2
3 sudo vim /etc/sudoers
4
5 # 赋予用户执行所有命令不需要输入密码,只需要下面一行
6 xxx ALL=(ALL) NOPASSWD:ALL
7
8 # 赋予用户执行某几个命令不需要密码,注意各个命令必写绝对路径,
9 # 此时只有指定的命令可以免密sudo执行,其他命令不再有sudo执行权限
10 xxx ALL= NOPASSWD: /usr/bin/lnmp php-fpm reload,b命令,c命令
11
12 # 若设置某些命令免密,其他命令需要密码,需要同时设置如下两行
13 test ALL=(ALL) ALL
14 test ALL=(ALL) NOPASSWD: /usr/bin/lnmp php-fpm reload
15
16 sudo chmod u-w /etc/sudoers
```

上面方式实现了不需要密码执行sudo命令，且在控制台手动执行时也不需要输入密码，但对于敏感命令这样也不好，还有一种方法是在运行命令时从输入自己获取密码

```
1 'echo "111111" | sudo -S rm -rf ./123.txt'
```

-S；从标准输入读取密码

这种方法将密码暴露在了代码中（也可以将密码放在一个公共位置，在代码中读取），但在控制台手动执行时是需要输入密码的。

两种方法可权衡选择。

iptables 封禁ip

iptables命令是Linux上常用的防火墙软件

查看所有规则

```
1 sudo iptables -L
```

iptables禁止指定ip访问

```
1 sudo iptables -A INPUT -s 54.36.143.97 -j DROP
```

封禁ip段

```
1 iptables -I INPUT -s 123.0.0.0/8 -j DROP      #封整个段即从123.0.0.1到12  
3.255.255.254的命令  
2 iptables -I INPUT -s 124.45.0.0/16 -j DROP      #封IP段即从123.45.0.1到12  
3.45.255.254的命令  
3 iptables -I INPUT -s 123.45.6.0/24 -j DROP      #封IP段即从123.45.6.1到12  
3.45.6.254的命令
```

iptables 解封ip

```
1 sudo iptables -L -n --line-number  
2  
3 上面命令会展示带编号的所有iptables规则，可执行下面命令对指定编号规则删除  
4  
5 sudo iptables -D INPUT num  
6  
7 比如要删除INPUT里序号为8的规则，执行：  
8  
9 sudo iptables -D INPUT 8
```

或直接

```
1 iptables -D INPUT -s ***.*.*.*.* -j DROP
```

注意以上设置的规则并不是永久有效，系统重启后会恢复原样。

永久有效设置

```
1 1.Ubuntu  
2 首先，保存现有的规则：  
3 iptables-save > /etc/iptables.rules  
4 然后新建一个bash脚本，并保存到 /etc/network/if-pre-up.d/目录下：
```

```
5 #!/bin/bash
6 iptables-restore < /etc/iptables.rules
7 这样，每次系统重启后iptables规则都会被自动加载。
8 ! 注意：不要尝试在.bashrc或者.profile中执行以上命令，因为用户通常不是root，而且
    这只能在登录时加载iptables规则。
9
10 2.CentOS, RedHat
11
12 yum install iptables-services
13 systemctl enable iptables.service //设置开机启动
14
15 # 保存iptables规则
16 service iptables save
17 # 重启iptables服务
18 service iptables stop
19 service iptables start
20 # 查看当前规则：
21 cat /etc/sysconfig/iptables
```

nginx 防止ab攻击

```
1 if ($http_user_agent ~* (ApacheBench)) {
2     return 200;
3 }
```

配置公私钥登录,禁止密码登录

* 禁止密码登录, 允许秘钥登录配置

编辑 SSH 配置文件:

```
1 #在root权限下执行  
2 vim /etc/ssh/sshd_config
```

找到一下几句

```
1 #RSAAuthentication yes  
2 #PubkeyAuthentication yes  
3 #AuthorizedKeysFile .ssh/authorized_keys
```

去掉上面3行前面的#

找到下面2行:

```
1 PasswordAuthentication yes  
2 PermitRootLogin yes  
3 修改为:  
4 PasswordAuthentication no  
5 PermitRootLogin no
```

保存后重启SSH服务。

```
1 service sshd restart
```

* 为用户配置公私钥

以 a 登录 b为例

```
1 1、a&b机器验证ssh是否安装成功: ssh -version  
2 若无安装, 先安装ssh  
3  
4 2、在要a机器生成公私钥: ssh-keygen -t rsa  
5 此时在/home/当前用户/.ssh下会生成公私钥文件  
6 公钥 id_rsa.pub  
7 私钥 id_rsa  
8  
9 3、将公钥配置到b机器中  
10 /home/要登录用户/.ssh/authorized_keys  
11  
12 4、修改b机器文件权限:  
13   authorized_keys 文件必须是600权限(也就是-rw---)或者644  
14   .ssh目录必须是700权限(也就是drwx--)  
15   /home/work目录 必须是 755权限 即drwxr-xr-x  
16
```

17 5、测试是否联通

18 第一次测试，会提示一个确认，选择yes回车，结果不用输入密码，登录成功！

mysql 字段类型说明和推荐

char (M) 和varchar (M) 的区别:

char的长度是不可变的，而varchar的长度是可变的；

char(M)定义的列的长度为固定的，M取值可以为0~255之间；

varchar(M)定义的列的长度为可变长，M取值可以为0~65535之间；

定义一个char[10]和varchar[10],如果存进去的是‘abcd’,那么char所占的长度依然为10，除了字符串‘abcd’外，后面跟六个空格，而varchar就立马把长度变为4了。

存数据时char类型数据后面的空格会被删除，varchar原样存储；取数据的时候，char类型尾部自动添加的空格会被删除，而varchar尾部的空格仍然保留；

char的存取速度还是要比varchar要快得多，因为其长度固定，方便程序的存储与查找；但是char也为此付出的是空间的代价，因为其长度固定，所以难免会有多余的空格占位符占据空间，可谓是以空间换取时间效率，而varchar是以空间效率为首位的；

char的存储方式是，对英文字母（ASCII）占用1个字节，对一个汉字占用两个字节；而varchar的存储方式是，对每个英文字母占用2个字节，汉字也占用2个字节，两者的存储数据都非unicode的字符数据；

何时该用char, 何时该用varchar:

varchar比char节省空间，在效率上比char会稍微差一些，即要想获得效率，就必须牺牲一定的空间，这也就是我们在数据库设计上常说的‘以空间换效率’；

varchar虽然比char节省空间，但是如果一个varchar列经常被修改，而且每次被修改的数据的长度不同，这会引起‘行迁移’(Row Migration)现象(如之前给分配的存储空间不足了)，而这造成多余的I/O，是数据库设计和调整中要尽力避免的，在这种情况下用char代替varchar会更好一些。而且varchar每次存储都要有额外的计算，得到长度等工作，如果一个非常频繁改变的，那就要有很多的精力用于计算，而这些对于char来说是不需要的。

char定长，一般用于固定长度的表单提交数据存储；例如：身份证号，手机号，电话，密码等；

固定长度的。比如使用uuid作为主键，身份证号，手机号，电话，密码等，用char应该更合适。因为他固定长度，varchar动态根据长度的特性就消失了，而且还要占个长度信息。

存储很短的信息，比如门牌号码101, 201.....这样很短的信息应该用char，因为varchar还要占个byte用于存储信息长度，本来打算节约存储的现在得不偿失。

从空间上考虑，varchar更好，从效率上考虑，char更好。这其中的选择就需要我们根据情况自己考量。

基本在使用时首先考虑varchar,若char更好则用char;

like 只能用于字符串格式

注意sql语句中like只能用于字符串格式字段，若要用于int|date等类型字段需要在sql语句中将字段数据转为char|varchar等字符串类型；

若int字段有索引，用like查询的话会使索引失效；

若某字段中存储的数据内容为数字，但业务场景中需要like查询等，则可以把该字段设置为char||varchar格式，如手机号虽然为11位数字，但最好用char(11),不要用bigint，即避免了32位系统导致的无法存储问

题，又方便模糊查询。

用int表示时间戳字段

存储时间戳时int类型具有更大的优势(时间戳 或 HmdHis 格式)

1、仅占用4个字节，资源占用少，但只能表示到2038年，可到时候改成bigint

2、一般逻辑中经常用到时间范围查询，大小比较等，int类型在建立索引后，查询速度更快。只是在处理简单的数字

3、条件范围搜索可以使用使用between

结论：适合需要进行大量时间范围查询的数据表

mysql 字段类型使用推荐

char

手机号 | 电话 | uuid | 身份证号 | 密码 | 门牌号

int

datetime类型时间保存为时间戳 或 HmdHis 格式

MySQL 共享锁&排他锁

共享锁(lock in share mode)

允许不同事务之间共享加锁读取，但不允许其它事务修改或者加入排他锁；如果有修改必须等待一个事务提交完成，才可以执行，容易出现死锁

共享锁事务之间的读取

session1:

```
1 start transaction;
2 select * from test where id = 1 lock in share mode;
```

session2:

```
1 start transaction;
2 select * from test where id = 1 lock in share mode;
```

此时session1和session2都可以正常获取结果，那么再加入session3 排他锁读取尝试

session3:

```
1 start transaction;
2 select * from test where id = 1 for update;
```

在session3中则无法获取数据，直到超时或其它事物commit

```
1 Lock wait timeout exceeded; try restarting transaction
```

共享锁之间的更新

当session1执行了修改语句：

session1:

```
1 update test set name = 'kkkk' where id = 1;
```

可以很多获取执行结果。

当session2再次执行修改id=1的语句时：

session2:

```
1 update test set name = 'zzz' where id = 1;
```

就会出现死锁或者锁超时，错误如下：

```
1 Deadlock found when trying to get lock; try restarting transaction
```

或者：

```
1 Lock wait timeout exceeded; try restarting transaction
```

必须等到session1完成commit动作后，session2才会正常执行，如果此时多个session并发执行，可想而知出现死锁的几率将会大增。

session3则更不可能

结论：

mysql共享锁(lock in share mode)

- 1 允许其它事务也增加共享锁读取
- 2 不允许其它事物增加排他锁(for update)
- 3 当事务同时增加共享锁时候，事务的更新必须等待先执行的事务commit后才行，
4 如果同时并发太大可能很容易造成死锁
5 共享锁，事务都加，都能读。修改是惟一的，必须等待前一个事务commit，才可

排他锁(for update)

当一个事物加入排他锁后，不允许其他事务加共享锁或者排它锁读取，更加不允许其他事务修改加锁的行。

排他锁不同事务之间的读取

同样以不同的session来举例

session1:

```
1 start transaction;
2 select * from test where id = 1 for update;
```

session2:

```
1 start transaction;
2 select * from test where id = 1 for update;
```

当session1执行完成后，再次执行session2，此时session2也会卡住，无法立刻获取查询的数据。直到出现超时

```
1 Lock wait timeout exceeded; try restarting transaction
```

或session1 commit才会执行

那么再使用session3 加入共享锁尝试

```
1 select * from test where id = 1 lock in share mode;
```

结果也是如此，和session2一样，超时或等待session1 commit

```
1 Lock wait timeout exceeded; try restarting transaction
```

排他锁事务之间的修改

当在session1中执行update语句：

```
1 update test set name = 123 where id = 1;
```

可以正常获取结果

```
1 Query OK, 1 row affected (0.00 sec)
2 Rows matched: 1  Changed: 1  Warnings: 0
```

此时在session2中执行修改

```
1 update test set name = 's2' where id = 1;
```

则会卡住直接超时或session1 commit,才会正常吐出结果

session3也很明显和session2一样的结果，这里就不多赘述

总结

- 1 事务之间不允许其它排他锁或共享锁读取，修改更不可能
- 2 一次只能有一个排他锁执行commit之后，其它事务才可执行
- 3 不允许其它事务增加共享或排他锁读取。修改是唯一的，必须等待前一个事务commit，才可

转自：<https://laravel-china.org/index.php/articles/12800/lock-in-share-mode-mysql-shared-lock-exclusive-lock-for-update>

mysql日期和时间类型

mysql有5种表示时间值的日期和时间类型，
分别为、DATE, TIME, YEAR, DATETIME, TIMESTAMP。

注意：

TIMESTAMP类型有专有的自动更新特性

类型	大小(字节)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2037 年某时	YYYYMMDD HHMMSS	混合日期和时间值, 时间戳

TIMESTAMP时间戳在创建的时候可以有多重不同的特性，如：

```
1 1. 在创建新记录和修改现有记录的时候都对这个数据列刷新:  
2  
3 TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
4  
5 2. 在创建新记录的时候把这个字段设置为当前时间, 但以后修改时, 不再刷新它:  
6  
7 TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
8  
9 3. 在创建新记录的时候把这个字段设置为0, 以后修改时刷新它:  
10  
11 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
12  
13 4. 在创建新记录的时候把这个字段设置为给定值, 以后修改时刷新它:  
14  
15 TIMESTAMP DEFAULT 'yyyy-mm-dd hh:mm:ss' ON UPDATE CURRENT_TIMESTAMP
```

TIMESTAMP列类型：

TIMESTAMP值可以从1970的某时的开始一直到2037年，精度为一秒，其值作为数字显示。

TIMESTAMP值显示尺寸的格式如下表所示：

1	+-----+	+-----+
2	列类型	显示格式
3	TIMESTAMP(14)	YYYYMMDDHHMMSS
4	TIMESTAMP(12)	YYMMDDHHMMSS
5	TIMESTAMP(10)	YYMMDDHHMM
6	TIMESTAMP(8)	YYYYMMDD
7	TIMESTAMP(6)	YYMMDD
8	TIMESTAMP(4)	YYMM
9	TIMESTAMP(2)	YY
10	+-----+	+-----+

1
2 “完整”TIMESTAMP格式是14位，但TIMESTAMP列也可以用更短的显示尺寸，创造最常见的显示尺寸是6、8、12、和14。
3 你可以在创建表时指定一个任意的显示尺寸，但是定义列长为0或比14大均会被强制定义为列长1

4。

4 列长在从1~13范围的奇数值尺寸均被强制为下一个更大的偶数。

mysql 事务特性以及隔离级别说明

事务：

事务就是针对数据库的一组操作，它可以由一条或者多条SQL语句组成，同一个事务的操作具备同步的特点，如果其中有一条语句不能执行的话，那么所有的语句都不会执行，也就是说，事务中的语句要么都执行，要么都不执行。

事务特性

事务操作具有严格的定义，它必须满足ACID：

ACID，指数据库事务正确执行的四个基本要素的缩写。包含：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。

原子性： 原子性是指事务是一个不可再分割的工作单位，事务中的操作要么都发生，要么都不发生。

一致性： 一致性是指在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。这是说数据库事务不能破坏关系数据的完整性以及业务逻辑上的一致性。

即事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

隔离性： 隔离性是指并发的事务是相互隔离的。即一个事务内部的操作及正在操作的数据必须封锁起来，不被企图进行修改的事务看到。

持久性： 持久性是指在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。即使出现了任何事故比如断电等，事务一旦提交，则持久化保存在数据库中。

隔离性说明：

隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

不考虑事务的隔离性会发生什么问题？

脏读： 脏读就是一个事务读取到了另一个事务还未提交的数据，另一个事务中数据可能进行了回滚，此时A事务读取的数据可能和数据库中数据是不一致的，此时认为数据是脏数据，读取脏数据过程叫做脏读。

不可重复读： 两次读取的数据不一致（表现在数据更新，数据内容不一致，update），当事务A第一次读取事务后，事务B对事务A读取的数据进行修改，事务A中再次读取的数据和之前读取的数据不一致，此过程称为不可重复读。

虚读（幻读）： 两次读取的数据一致（表现在数据新增或删除，数据量不一致，insert & delete），事务A按照特定条件查询出结果，事务B新增了一条符合条件的数据。事务A中查询的数据和数据库中的数据不一致的，事务A好像出现了幻觉，这种情况称为幻读。主要针对的操作是新增或删除。

丢失更新： 两个事务对同一条记录进行操作，后提交的事务，将先提交的事务的修改的数据覆盖了

为了防止出现脏读、不可重复读、幻读等情况，我们就需要根据我们的实际需求来设置数据库的隔离级别。

事务的隔离级别

隔离级别	含义
Serializable	可避免脏读、不可重复读、虚读情况的发生。（串行化）
Repeatable read	可避免脏读、不可重复读情况的发生。（可重复读）不可以避免虚读
Read committed	可避免脏读情况发生（读已提交）
Read uncommitted	最低级别，以上情况均无法保证。（读未提交）

如何使用这些隔离级别，那就需要根据业务的实际情况来进行判断了。

读未提交 (Read Uncommitted)

读未提交，顾名思义，就是可以读到未提交的内容。

因此，在这种隔离级别下，查询是不会加锁的，也由于查询的不加锁，所以这种隔离级别的一致性是最差的，可能会产生“脏读”、“不可重复读”、“幻读”。

如无特殊情况，基本是不会使用这种隔离级别的。

读提交 (Read Committed)

读提交，顾名思义，就是只能读到已经提交了的内容。

这是各种系统中最常用的一种隔离级别，也是SQL Server和Oracle的默认隔离级别。这种隔离级别能够有效的避免脏读，但除非在查询中显示的加锁，如：

```
1 select * from T where ID=2 lock in share mode;
2
3 select * from T where ID=2 for update;
```

不然，普通的查询是不会加锁的。

那为什么“读提交”同“读未提交”一样，都没有查询加锁，但是却能够避免脏读呢？

这就要说道另一个机制“快照 (snapshot) ”，而这种既能保证一致性又不加锁的读也被称为“快照读 (Snapshot Read) ”

假设没有“快照读”，那么当一个更新的事务没有提交时，另一个对更新数据进行查询的事务会因为无法查询而被阻塞，这种情况下，并发能力就相当的差。

而“快照读”就可以完成高并发的查询，不过，“读提交”只能避免“脏读”，并不能避免“不可重复读”和“幻读”。

可重复读 (Repeated Read)

可重复读，顾名思义，就是专门针对“不可重复读”这种情况而制定的隔离级别，自然，它就可以有效的避免“不可重复读”。而它也是MySQL的默认隔离级别。

在这个级别下，普通的查询同样是使用的“快照读”，但是，和“读提交”不同的是，当事务启动时，就不允许进行“修改操作 (Update) ”了，而“不可重复读”恰恰是因为两次读取之间进行了数据的修改，因

此，“可重复读”能够有效的避免“不可重复读”，但却避免不了“幻读”，因为幻读是由于“插入或者删除操作（Insert or Delete）”而产生的。

串行化（Serializable）

这是数据库最高的隔离级别，这种级别下，事务“串行化顺序执行”，也就是一个一个排队执行。

这种级别下，“脏读”、“不可重复读”、“幻读”都可以被避免，但是执行效率奇差，性能开销也最大，所以基本没人会用。

隔离级别的设置与查询：

1、设置隔离级别：

设置隔离级别分为设置全局的隔离级别与设置当前的隔离级别

全局设置，已存在的session不会生效，以后的新session会生效（以读未提交举例）：

```
1 set global transaction isolation level read uncommitted;
```

单独设置当前连接：

```
1 set session transaction isolation level read uncommitted;
```

2、在MySQL数据库中查看当前事务的隔离级别：

```
1 select @tx_isolation;
```

高并发系统数据库架构设计

在WEB网站的规模从小到大不断扩展的过程中，数据库的访问压力也不断的增加，数据库的架构也需要动态扩展，在数据库的扩展过程基本上包含如下几步，每一个扩展都可以比上一步骤的部署方式的性能得到数量级的提升。

WEB应用和数据库部署在同一台服务器上

一般的小规模的网站采用这种方式，用户量、数据量、并发访问量都比较小，否则单台服务器无法承受，并且在遇到性能瓶颈的时候升级硬件所需要的费用非常高昂，在访问量增加的时候，应用程序和数据库都来抢占有限的系统资源，很快就又会遇到性能问题。

WEB应用和数据库部署在各自独立的服务器上

web应用和数据库分部部署，WEB应用服务器和数据库服务器各司其职，在系统访问量增加的时候可以分别升级应用服务器和数据库服务器，这种部署方式是一般小规模网站的典型部署方式。在将应用程序进行性能优化并且使用数据库对象缓存策略的情况下，可以承载较大的访问量，比如2000用户，200个并发，百万级别的数据量。

数据库服务器采用集群方式部署（比如Oracle的一个数据库多个实例的情况）

数据库集群方式能承担的负载是比较大的，数据库物理介质为一个磁盘阵列，多个数据库实例以虚拟IP方式向外部应用服务器提供数据库连接服务。这种部署方式基本上可以满足绝大多数的常见WEB应用，但是还是不能满足大用户量、高负载、数据库读写访问非常频繁的应用。

数据库采用主从部署方式

在面向大众用户的博客、论坛、交友、CMS等系统中，有上百万的用户，有上千万的数据量，存在众多的数据库查询操作，也有较多的数据库写操作，并且在多数情况下都是读操作远大于写操作的。在这个时候，假如能将数据库的读写操作分离的话，对于系统来讲是一个很大的提高啦。数据库的主从部署方式就走到我们面前啦。

主从复制：

几乎所有的主流数据库都支持复制，这是进行数据库简单扩展的基本手段。下面以Mysql为例来说明，它支持主从复制，配置也并不复杂，只需要开启主服务器上的二进制日志以及在主服务器和从服务器上分别进行简单的配置和授权。Mysql的主从复制是一句主服务器的二进制日志文件进行的，主服务器日志中记录的操作会在从服务器上重放，从而实现复制，所以主服务器必须开启二进制日志，自动记录所有对于主数据库的更新操作，从服务器再定时到主服务器取得二进制日志文件进行重放则完成了数据的复制。主从复制也用于自动备份。

读写分离：

为保证数据库数据的一致性，我们要求所有对于数据库的更新操作都是针对主数据库的，但是读操作是可以针对从数据库来进行。大多数站点的数据库读操作比写操作更加密集，而且查询条件相对复杂，数据库的大部分性能消耗在查询操作上了。

主从复制数据是异步完成的，这就导致主从数据库中的数据有一定的延迟，在读写分离的设计中必须要考虑这一点。以博客为例，用户登录后发表了一篇文章，他需要马上看到自己的文章，但是对于其它用户来讲可以允许延迟一段时间（1分钟/5分钟/30分钟），不会造成什么问题。这时对于当前用户就需要读主数据库，对于其他访问量更大的外部用户就可以读从数据库。

数据库反向代理：

在读写分离的方式使用主从部署方式的数据库的时候，会遇到一个问题，一个主数据库对应多台从服务器，对于写操作是针对主数据库的，数据库个数是唯一的，但是对于从服务器的读操作就需要使用适当的算法来分配请求啦，尤其对于多个从服务器的配置不一样的时候甚至需要读操作按照权重来分配。

对于上述问题可以使用数据库方向代理来实现。就像WEB方向代理服务器一样，MysqI Proxy同样可以在SQL语句转发到后端的MysqI服务器之前对它进行修改。

数据库垂直分割

主从部署数据库中，当写操作占了主数据库的CPU消耗的50%以上的时候，我们再增加从服务器的意义就不是很大了，因为所有的从服务器的写操作也将占到CPU消耗的50%以上，一台从服务器提供出来查询的资源非常有限。数据库就需要重新架构了，我们需要采用数据库垂直分区技术啦。

最简单的垂直分区方式是将原来的数据库中独立的业务进行分拆（被分拆出来的部分与其它部分不需要进行Join连接查询操作），比如WEB站点的BLOG和论坛，是相对独立的，与其它的数据的关联性不是很强，这时可以将原来的数据库拆分为一个BLog库，一个论坛库，以及剩余的表所组成的库。这三个库再各自进行主从数据库方式部署，这样整个数据库的压力就分担啦。

另外查询扩展性也是采用数据库分区最主要的原因之一。将一个大的数据库分成多个小的数据库可以提高查询的性能，因为每个数据库分区拥有自己的一小部分数据。假设您想扫描1亿条记录，对一个单一分区的数据库来讲，该扫描操作需要数据库管理器独立扫描一亿条记录，如果您将数据库系统做成50个分区，并将这1亿条记录平均分配到这50个分区上，那么每个数据库分区的数据库管理器将只扫描200万记录。

数据库水平分割

在数据库的垂直分区之后，假如我们的BLOG库又再次无法承担写操作的时候，我们又该怎么办呢？数据库垂直分区这种扩展方式又无能为力了，我们需要的是水平分区。

水平分区意味着我们可以将同一个数据库表中的记录通过特定的算法进行分离，分别保存在不同的数据库表中，从而可以部署在不同的数据库服务器上。很多的大规模的站点基本上都是主从复制+垂直分区+水平分区这样的架构。水平分区并不依赖什么特定的技术，完全是逻辑层面的规划，需要的是经验和业务的细分。

如何分区呢？对于大型的WEB站点来说，必须分区，并且对于分区我们没有选择的余地，对于那些频繁访问导致站点接近崩溃的热点数据，我们必须分区。

在对数据分区的时候，我们必须要存在一个分区索引字段，比如USER_ID，它必须和所有的记录都存在关系，是分区数据库中的核心表的主键，在其它表中作为外键，并且在使用主键的时候，该主键不能是自增长的，必须是业务主键才可以。

余数分区：

我们可以将User_ID%10后的值为依据存入到不同的分区数据库中，该算法简单高效，但是在分区数据库个数有变动的时候，整个系统的数据需要重新分布。

范围分区：

我们可以将User_ID的范围进行分区，比如1-100000范围为一个分区数据库，100001-200000范围为一个分区数据库，该算法在分区数据库个数有变动的时候，系统非常有利于扩展，但是容易导致不同分区之间的压力不同，比如老用户的分区数据库的压力很大，但是新用户的分区数据库的压力偏小。

映射关系分区：

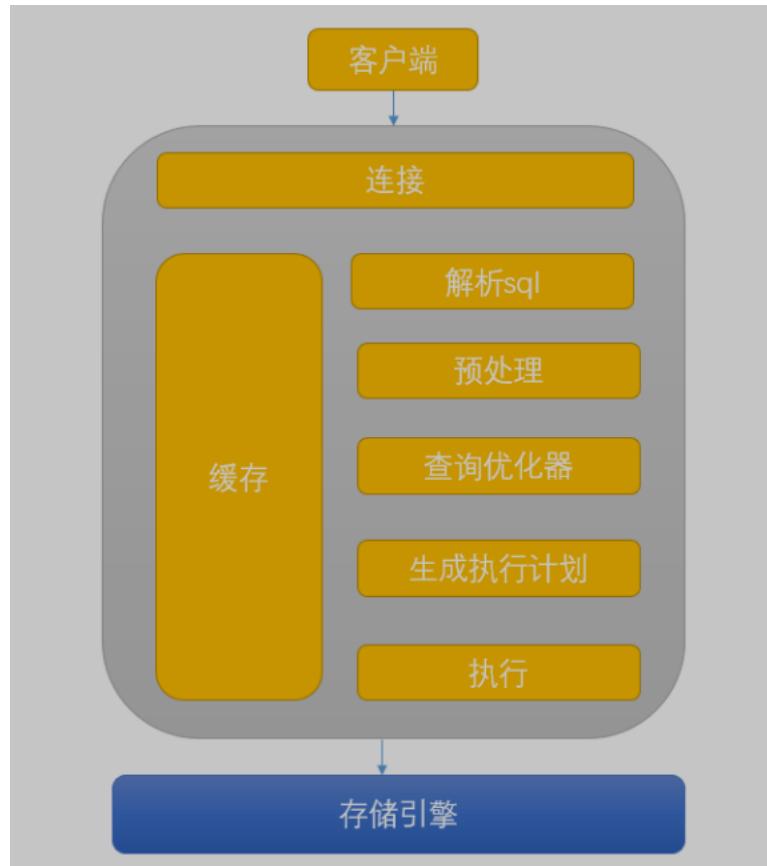
将对分区索引字段的每个可能的结果创建一个分区映射关系，这个映射关系非常庞大，需要将它们写入数据库中。比如当应用程序需要知道User_id为10的用户的BLOG内容在那个分区时，它必须查询数据库获取答案，当然，我们可以使用缓存来提高性能。

这种方式详细保存了每一个记录的分区对应关系，所以各个分区有非常强的可伸缩性，可以灵活的控制，并且将数据库从一个分区迁移到另一个分区也很简单，也可以使各个分区通过灵活的动态调节来保持压力的分布平衡。

转自：<http://www.cnblogs.com/lihaozy/archive/2013/08/02/3231776.html>

MySQL优化建议总结和注意事项

MySQL逻辑架构：



有三层结构：

第一层：客户端通过连接服务，将要执行的SQL指令传输过来

第二层（分为两种方式）

方式一：服务器解析并优化SQL，生成最终的执行计划并执行

方式二：服务器从缓存中获取查询结果

第三层：存储引擎，负责数据的储存和提取

优化建议总结：

- 对于查询缓存

查询缓存的空间不要设置的太大

原因：因为查询缓存是靠一个全局锁操作保护的，如果查询缓存配置的内存比较大且里面存放了大量的查询结果，当查询缓存失效的时候，会长时间的持有这个全局锁。因为查询缓存的命中检测操作以及缓存失效检测也都依赖这个全局锁，所以可能会导致系统僵死的情况。对于写密集型的应用，直接禁用查询缓存。

原因：如果一个表被更改了，那么使用那个表的查询缓存将不再有效，并且从缓冲区中移出。频繁移除缓存会消耗大量时间

mysql缓存相关知识如 缓存配置， 使用监控， 优化等有兴趣可自己查一下， 包括很多内容

- 对于数据类型

尽量使用对应的数据类型。比如，不要用字符串类型保存时间，用整型保存IP。

选择更小的数据类型。能用TinyInt不用Int。

- 对于具体查询

避免查询无关的列，如使用Select * 返回所有的列。

从数据库里读出越多的数据，那么查询就会变得越慢。并且，如果你的数据库服务器和WEB服务器是两台独立的

服务器的话，这还会增加网络传输的负载。所以，你应该养成一个需要什么就取什么的习惯。

- 避免查询无关的行

切分查询。将一个对服务器压力较大的任务，分解到一个较长的时间中，并分多次执行。如要删除一万条数据，可以分10次执行，每次执行完成后暂停一段时间，再继续执行。过程中可以释放服务器资源给其他任务。

分解关联查询。将多表关联查询的一次查询，分解成对单表的多次查询。可以减少锁竞争，查询本身的查询效率也比较高。因为MySQL的连接和断开都是轻量级的操作，不会由于查询拆分为多次，造成效率问题。

注意count的操作只能统计不为null的列，所以统计总的行数使用count (*)。

group by 按照标识列分组效率高，分组结果不宜出行分组列之外的列。

关联查询延迟关联，可以根据查询条件先缩小各自要查询的范围，再关联。

Limit分页优化。可以根据索引覆盖扫描，再根据索引列关联自身查询其他列。

原理：

利用表的覆盖索引来加速分页查询。

索引查询的语句中如果只包含了那个索引列（覆盖索引），那么这种情况会查询很快。

```
SELECT * FROM product a JOIN (select id from product limit 866613, 20) b ON a.ID = b.id
```

- Union查询默认去重，如果不是业务必须，建议使用效率更高的Union All

只查询一条数据时，加 limit 1

当你查询表的有些时候，你已经知道结果只会有一条结果，在这种情况下，加上 LIMIT 1

可以增加性能。这样MySQL数据库引擎会在找到一条数据后停止搜索，而不是继续往后查找下一条符合记录的数据。

- 千万不要 ORDER BY RAND()

想打乱返回的数据行？随机挑一个数据？这样使用只让你的数据库的性能呈指数级的下降。这里的问题是：MySQL会不得不去执行RAND()函数（很耗CPU时间），而且这是为了每一行记录去记行，然后再对其排序。就算是你用了Limit 1也无济于事（因为要排序）

- 我们应该为数据库里的每张表都设置一个ID做为主键，而且最好的是一个INT型的（推荐使用UNSIGNED），并设置上自动增加的AUTO_INCREMENT标志。

就算是你 users 表有一个主键叫“email”的字段，你也别让它成为主键。使用 VARCHAR 类型来当主键会使用得性能下降。另外，在你的程序中使用表的ID来构造你的数据结构。

- 选择正确的存储引擎

在 MySQL 中常用的有两个存储引擎 MyISAM 和 InnoDB，每个引擎都有利有弊。

MyISAM 适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要update一个字段，整个表都会被锁起来，而别的进程，就算是读进程都无法操作直到读操作完成。

InnoDB 的趋势会是一个非常复杂的存储引擎，对于一些小的应用，它会比 MyISAM 还慢。但是它支持“行锁”，于是在写操作比较多的时候，会更优秀。并且，他还支持更多的高级应用，比如：事务。InnoDB的性能更加平衡。

老版本中mysql默认搜索引擎是MyISAM，现在已经都改为innodb。

- 使用INNER JOIN 而不是WHERE来创建连接(多表查询)

这个笛卡尔连接问题，若量表各有100数据，where 产生 $100 \times 100 = 10000$ 条数据，再根据where条件过滤

而 INNER JOIN 只产生 100 条结果

- mysql插入

一条SQL语句插入多条数据

```
1 INSERT INTO `insert_table` (`datetime`, `uid`, `content`, `type`)
2 VALUES ('0', 'userid_0', 'content_0', 0), ('1', 'userid_1', 'content_1', 1)
```

```
1', 1);
```

合并后日志量（MySQL的binlog和innodb的事务日志）减少了，降低日志刷盘的数据量和频率，从而提高效率，同时也能减少SQL语句解析的次数，减少网络传输的IO。

- 数据有序插入

数据有序的插入是指插入记录在主键上是有序排列，dateid:

```
1 INSERT INTO `insert_table` (`dateid`, `uid`, `content`, `type`) VALUE
  S ('0', 'userid_0', 'content_0', 0);
2
3
4 INSERT INTO `insert_table` (`dateid`, `uid`, `content`, `type`) VALUE
  S ('1', 'userid_1', 'content_1', 1);
5
6
7 INSERT INTO `insert_table` (`dateid`, `uid`, `content`, `type`) VALUE
  S ('2', 'userid_2', 'content_2', 2);
```

由于数据库插入时，需要维护索引数据，无序的记录会增大维护索引的成本。我们可以参照innodb使用的B+tree索引，如果每次插入记录都在索引的最后面，索引的定位效率很高，并且对索引调整较小；如果插入的记录在索引中间，需要B+tree进行分裂合并等处理，会消耗比较多计算资源，并且插入记录的索引定位效率会下降，数据量较大时会有频繁的磁盘操作。建表时最好以自增id为主键就解决了这个问题。

- 主从同步，读写分离

MySQL主从同步的作用：

- 1、可以作为一种备份机制，相当于热备份（在从备份，避免备份期间影响主服务器服务）
- 2、可以用来做读写分离，均衡数据库负载（主写从读）
- 3、当主服务器出现问题时，可以切换到从服务器。

- 分库分表

对表进行水平划分

如果一个表的记录数太多了，比如上千万条，而且需要经常检索，那么我们就有必要化整为零了。如果我拆成100个表，那么每个表只有10万条记录。当然这需要数据在逻辑上可以划分。一个好的划分依据，有利于程序的简单实现，也可以充分利用水平分表的优势。比如系统界面上只提供按月查询的功能，那么把表按月拆分成12个，每个查询只查询一个表就够了。

- 对表进行垂直划分

有些表记录数并不多，可能也就2、3万条，但是字段却很长，表占用空间很大，检索表时需要执行大量I/O，严重降低了性能。这个时候需要把大的字段拆分到另一个表，并且该表与原表是一对一的关系。

（降低了每张表的文件大小）实例：文章信息表就可以拆分成两张表，一张记录文章信息如作者、栏目、发表日期、关键词等，另一张表记录文章内容，摘要等。利用id进行对应。因为在大多数情况下只需要搜索第一张表即可，只有在文章展示页才需要访问两张表，如此可以提高搜索性能！

- 水平分库分表

水平分库分表与上面讲到的水平分表的思想相同，唯一不同的就是将这些拆分出来的表保存在不同的数据中。

某种意义上来说，有些系统中使用的“冷热数据分离”（将一些使用较少的历史数据迁移到其他的数据库中。而在业务功能上，通常默认只提供热点数据的查询），也是类似的实践。在高并发和海量数据的场景下，分库分表能够有效缓解单机和单库的性能瓶颈和压力，突破IO、连接数、硬件资源的瓶颈。

- 垂直分库

垂直分库基本的思路就是按照业务模块来划分出不同的数据库，而不是像早期一样将所有的数据表都放到同一个数据库中。

**count() 优化：

1 有时候某些业务场景并不需要完全精确的COUNT值，可以用近似值来代替，EXPLAIN出来的行数就是一个不错的近似值，而且执行EXPLAIN并不需要真正地去执行查询，所以成本非常低。

2 实例：

3

```
4 [SQL]select count(name) from aaa;
5 受影响的行: 0
6 时间: 2.957s
7
8 结果:
9 428396
10
11 [SQL]EXPLAIN select count(name) from aaa;
12 受影响的行: 0
13 时间: 0.038s
14
15 结果:
16 1 SIMPLE aaa index      name_idx    767    413996 Using index
```

1 通常来说，执行COUNT()都需要扫描大量的行才能获取到精确的数据，因此很难优化，MySQL层面还能做得也就只有覆盖索引了。
2
3 如果不还能解决问题，只有从架构层面解决了，比如添加汇总表，或者使用redis这样的外部缓存系统。

关联查询优化：

1 在大数据场景下，表与表之间通过一个冗余字段来关联，要比直接使用JOIN有更好的性能。
如果确实需要使用关联查询的情况下，需要特别注意的是：
2
3 1、确保ON和USING字句中的列上有索引。在创建索引的时候就要考虑到关联的顺序。当表A和表B用列c关联的时候，如果优化器关联的顺序是A、B，那么就不需要在A表的对应列上创建索引。
4
5 2、没有用到的索引会带来额外的负担，一般来说，除非有其他理由，只需要在关联顺序中的第二张表的相应列上创建索引。
6
7 3、确保任何的GROUP BY和ORDER BY中的表达式只涉及到一个表中的列，这样MySQL才有可能使用索引来优化。

union 优化：

1 MySQL处理UNION的策略是先创建临时表，然后再把各个查询结果插入到临时表中，最后再来做查询。因此很多优化策略在UNION查询中都没有办法很好的时候。经常需要手动将WHERE、LIMIT、ORDER BY等字句“下推”到各个子查询中，以便优化器可以充分利用这些条件先优化。
2
3 除非确实需要服务器去重，否则就一定要使用UNION ALL，如果没有ALL关键字，MySQL会给临时表加上DISTINCT选项，这会导致整个临时表的数据做唯一性检查，这样做的代价非常高。当然即使使用ALL关键字，MySQL总是将结果放入临时表，然后再读出，再返回给客户端。虽然很多时候没有这个必要，比如有时候可以直接把每个子查询的结果返回给客户端。

大表 alter table：

大表ALTER TABLE非常耗时，MySQL执行大部分修改表结构操作的方法是用新的结构创建一个张空表，从旧表中查出所有的数据插入新表，然后再删除旧表。

尤其当内存不足而表又很大，而且还有很大索引的情况下，耗时更久。当然有一些奇淫技巧可以解决这个问题，有兴趣可自行查阅。

比如：大表更改默认值使用alter table不重建表，直接修改.frm

在MySQL中执行很大部分的修改动作，都需要重建一个表，然后把数据放进去，最后删除旧的表！有时候要是有索引的列上进行大批且频繁的表的时候会导致系统的性能严重下降，这里可以在修改SQL上做部分调整，减轻相关的构建结构带来的系统压力问题！

例如 在修改一个表的默认值为8的时候，常规做法为：

```
(1) : alter table modes modify column dept tinyint(3) not null default 8;
```

这里通过一些show status分析出，每次都要做大量的句柄的读取和插入操作，类似于从新构建了一张新表的样式，这样在一些大表，上千万的数据量会出现瓶颈问题！

这里我们需要灵活知道mysql的相关默认值实际是放在相关的表结构.frm文件中；我们可以不经过数据层，可以直接调整！上述的modify column会导致相关的表进行拷贝操作，不利于系统的正常稳定运行，可以使用下面方式；

```
(2) : alter table modes alter column dept set default 8;
```

这里只是更改了相关的frm文件，没有改动表，因此速度很快的即可完成；

总结：在此我们可以注意到相关的alter table后面跟不同形式的命令，可以对数据产生了不同程度的影响，这里还有一个change column操作也是不一样的！

表命名问题

1、数据库表名创建时使用小写英文字母加下划线的形式，不要出现大写字母，防止大小写敏感问题（数据库、字段创建最好也遵守这一规则，保持统一）

mysql定位和分析执行效率方法

```
1
2 1. explain(mysql), 查看sql语句执行计划;
3
4 2. 启用slow query log记录慢查询;
5
6 3. 通常还要看数据库设计是否合理, 需求是否合理等。
```

count(*)与group by的统计问题

sql统计示例：

统计 group by 后每个分组的个数：

```
1 SELECT COUNT(*) FROM search_word GROUP BY word;
```

那么得到的结果是：

```
1 COUNT(*)
2 1
3 10
4 1
5 2
6 1
7 1
8 1
9 15
```

统计 group by 后分组个数

```
1 SELECT COUNT(DISTINCT word) FROM search_word;
```

结果：

```
1 COUNT(DISTINCT word)
2 346
```

或

```
1 SELECT COUNT(*) FROM (SELECT COUNT(*) FROM search_word GROUP BY word)
t;
```

结果：

```
1 COUNT(*)
2 346
```

MySQL不能远程连接解决方法

mysql不能远程连接主要从以下3个方面着手解决：

1、如果是云服务器检查其安全组规则，查看一下是否放行了3306端口。

2、在mysql库user表中添加一个用户 主机为 % 的任意主机（也可以编辑已有的用户），如果要指定IP远程访问，创建一个 主机 为你指定IP的用户即可。

```
1 # 创建并授权可远程访问root用户
2 GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '111111' WITH
   GRANT OPTION;
3
4 # 刷新MySQL的系统权限相关表，使设置生效
5 FLUSH PRIVILEGES;
```

3、检查iptables，删除DROP 3306端口的规则。参考如下文章：

Here is yuque doc card, click on the link to
view:<https://www.yuque.com/u316337/tforki/woqd87>