



开发记录

卷一

Dolphin

Copyright © 2017 Xiaoqiang Jiang

EDITED BY XIAOQIANG JIANG

[HTTP://JIANGXIAOQIANG.GITHUB.COM/](http://jiangxiaoqiang.github.com/)

All Rights Reserved.

Version 01:41, March 26, 2018

Contents

| I | Tool | |
|------------|-------------------------------|----------|
| 1 | Widgets | 3 |
| 1.1 | ssh(Secure Shell) | 3 |
| 1.1.1 | 安全 | 3 |
| 1.1.2 | ssh 连接慢 (Connect Slow) | 4 |
| 1.1.3 | ssh 连接自动断开 | 5 |
| 1.1.4 | Permission denied (publickey) | 5 |
| 1.1.5 | Session 时间 | 6 |
| 1.1.6 | 代理转发 | 7 |
| 1.1.7 | ssh 查看日志 | 7 |
| 1.2 | VisualVM | 8 |
| 1.3 | Tool Set | 8 |
| 1.3.1 | ECS(Elastic Compute Service) | 8 |
| 1.3.2 | shadowsocks | 8 |
| 1.3.3 | youtube-dl | 9 |
| 1.3.4 | Jenkins | 9 |
| 1.3.5 | OpenVPN | 11 |
| 1.3.6 | fastDFS | 16 |

| | | |
|------------|----------------------------|-----------|
| 2 | Gradle | 17 |
| 2.1 | 基础 | 17 |
| 2.1.1 | 引用本地文件 | 18 |
| 2.1.2 | Gradle Properties 支持 | 18 |
| 2.1.3 | 执行流程 | 19 |
| 2.1.4 | 默认 JVM | 19 |
| 2.1.5 | Repositories | 19 |
| 2.1.6 | 常见问题 | 21 |
| 2.2 | Gradle 对象 | 23 |
| 2.2.1 | 属性 (Properties) | 23 |
| 2.2.2 | Task | 24 |

II

DB

| | | |
|------------|---|-----------|
| 3 | MariaDB | 29 |
| 3.1 | 常用操作 | 29 |
| 3.1.1 | 导入导出 | 29 |
| 3.2 | 常见问题 | 30 |
| 3.2.1 | mariadb 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES) | 30 |
| | Bibliography | 33 |
| | Books | 33 |
| | Articles | 33 |

这里记录的是一些比较杂乱的笔记，绝大多数文字皆来源于网络，不是自己的原创，这里没有高深的算法，没有宏伟的技术及系统架构，只是一些平时工作中遇到的一些问题，和解决问题的思路以及所采用的方案。由于平时工作时还没有遇到前人没有遇到过的问题需要自己发明方去解决(其实真的遇到估计也是没辙)，所以绝大部分内容是为了避免再遇到同样的问题时，又需要到处去搜寻，索性将之记录下来，以便于下次可以快刀斩乱麻，迅速解决问题。

1 Widgets 3

- 1.1 ssh(Secure Shell)
- 1.2 VisualVM
- 1.3 Tool Set

2 Gradle 17

- 2.1 基础
- 2.2 Gradle 对象

1. Widgets

1.1 ssh(Secure Shell)

关闭不活动的 ssh 会话, 使用 w 命令来识别出不活动或者是空闲的 ssh 会话, 使用 pstree 命令来获取空闲会话的 PID, 就是使用 kill 命令来关闭会话了。

1.1.1 安全

修改默认端口 (Change Default Port)

修改默认端口无法阻止专业的攻击, 因为修改 ssh 默认端口后, 使用 nmap 工具一样可以识别出来 OpenSSH 服务. 但是修改默认端口为随机端口后有一点好处就是可以避免被脚本自动扫描到来尝试登录 (爆破).

免密登陆 (Login Without Password)

SSH 服务如果在公网上, 非常容易受到攻击, 特别是弱口令扫描, 字典扫描. 保护服务器免受攻击, 可以使用 SSH 密钥, 禁用口令认证, 如果不能做到这一点, 务必使用强壮的密码. 还可以设置登陆 IP 白名单. 更改服务器 ssh 端口 (基本上没有效果, 调整了之后可以使用 nmap 轻松扫描到). 使用 snort、ossec 等开源的入侵检测设备保护服务器. 免密登录需要注意的是, .ssh 文件夹下的 authorize_key 文件的权限需要是 600. 而 .ssh 文件夹的权限需要是 700. 调整/etc/ssh/sshd_config 配置文件:

```
1 RSAAuthentication yes
2 PubkeyAuthentication yes
```

```
3 AuthorizedKeysFile      .ssh/authorized_keys
```

拷贝公钥到服务器:

```
1 ssh -p 22 pi@192.168.31.25 'mkdir -p .ssh && cat >> .  
    ssh/authorized_keys' < ~/.ssh/id_rsa.pub  
2 # 如果本地没有生成过 ssh key  
3 # 使用如下命令生成 ssh key  
4 ssh-keygen -t rsa -C "a@gmail.com"
```

调整文件夹权限:

```
1 chmod 700 ~/.ssh  
2 chmod 600 ~/.ssh/authorized_keys
```

在 Mac OS X 中, 有时自动登陆需要反复输入密码, 解决问题的方法是可以配置 Serria 记住密码:

```
1 usekeychain yes
```

有时在 Ubuntu 下使用 ssh 也需要输入密码 (Enter passphrase for key id_rsa), 此时也可以使用 keychain:

```
1 #安装 keychain  
2 sudo apt install keychain  
3 /usr/bin/keychain ~/.ssh/id_rsa
```

keychain 会起一个 ssh-agent, 后来登录的人通过设置环境变量使用同一个 ssh-agent. 添加的只是在当前 shell 下生效, 如果需要每次都生效.

1.1.2 ssh 连接慢 (Connect Slow)

在使用 SSH 时, 每次连接建立都相当的慢啊, 要 20 秒以上才能够登录上去, 严重影响工作效率。在服务端的/etc/ssh/sshd_config 文件中, 修改配置, 将 GSSAPI-Authentication 默认设置为关闭即可, 配置文件修改后需要重新启动 sshd 守护进程配置才能够生效:

```
1 GSSAPIAuthentication no  
2 # 重启 sshd 守护进程  
3 sudo service sshd restart
```

GSSAPI (Generic Security Services Application Programming Interface) 是一套类似 Kerberos 5 的通用网络安全系统接口。该接口是对各种不同的客户端服务器安全机制的封装，以消除安全接口的不同，降低编程难度。但该接口在目标机器无域名解析时会有问题。使用 `strace` 查看后发现，ssh 在验证完 key 之后，进行 authentication gssapi-with-mic，此时先去连接 DNS 服务器，在这之后会进行其他操作。连接慢也可以关闭 DNS 反向解析，在 linux 中，默认就是开启了 SSH 的反向 DNS 解析，这个会消耗大量时间，因此需要关闭 [1]。

```
1 UseDNS=no
```

1.1.3 ssh 连接自动断开

ssh 连接总是自动断开，客户端配置：

```
1 TCPKeepAlive=yes
2 #客户端主动向服务端请求响应的间隔
3 ServerAliveInterval 60
```

1.1.4 Permission denied (publickey)

在 ssh 登陆机器时，提示 Permission denied (publickey)。可以尝试的方法，在连接命令中加上 -vvv 参数，观察详细的调试输出：

```
1 ssh -p 2222 -vvv hldev@10.0.0.22
```

设置文件夹对应的权限：

```
1 sudo chmod 700 .ssh
2 sudo chmod 600 .ssh/authorized_keys
```

登陆服务器观察相应的日志输出：

```
1 tail -f /var/log/auth.log
```

是不是服务器关闭了密码登陆？只能使用公钥认证登陆。后面检查确实如此，服务器端为了安全考虑，关闭了基于密码登录的方式，但是需要登录的主机并没有将自己的公钥拷贝到服务器上。所以服务器直接提示了 Permission denied (publickey)，解决的办法就是在服务器端暂时开启密码登录，在 `/etc/ssh/sshd_config` 中调整配置：

```
1 #允许使用基于密钥认证的方式登陆
2 PubkeyAuthentication yes
```

1.1.5 Session 时间

在使用 ssh 的过程中，经常会遇到一会儿没有操作就自动断开了，不是非常方便。

ClientAliveInterval

修改/etc/ssh/sshd_config 配置文件 ClientAliveInterval 300（默认为 0），参数的意思是每 5 分钟，服务器向客户端发一个消息，用于保持连接，使用 service sshd reload 让其修改后生效。如果发现还是有问题，可以试着把 300 设置小一点，例如 60。

ClientAliveCountMax

另外，至于 ClientAliveCountMax，使用默认值 3 即可。ClientAliveCountMax 表示服务器发出请求后客户端没有响应的次数达到一定值，就自动断开。

ControlPersist 4h

在./ssh/config 中添加一行：

```
1 ControlPersist 4h
```

When used in conjunction with ControlMaster, specifies that the master connection should remain open in the background (waiting for future client connections) after the initial client connection has been closed. If set to no, then the master connection will not be placed into the background, and will close as soon as the initial client connection is closed. If set to yes or 0, then the master connection will remain in the background indefinitely (until killed or closed via a mechanism such as the “ssh -O exit”). If set to a time in seconds, or a time in any of the formats documented in sshd_config, then the backgrounded master connection will automatically terminate after it has remained idle (with no client connections) for the specified time¹. 现在你每次通过 SSH 与服务器建立连接之后，这条连接将被保持 4 个小时，即使在你退出服务器之后，这条连接依然可以重用，因此，在你下一次（4 小时之内）登录服务器时，你会发现连接以闪电般的速度建立完成，这个选项对于通过 scp 拷贝多个文件提速尤其明显，因为你不在需要为每个文件做单独的认证了。

¹http://man.openbsd.org/ssh_config.5

1.1.6 代理转发

本地转发 Local Forward

将本地机(客户机)的某个端口转发到远端指定机器的指定端口. 工作原理是这样的, 本地机器上分配了一个 socket 侦听 port 端口, 一旦这个端口上有了连接, 该连接就经过安全通道转发出去, 同时远程主机和 host 的 hostport 端口建立连接. 可以在配置文件中指定端口的转发. 只有 root 才能转发特权端口. 应用实例可以参看??。

```
1 # Mac OS X 端口转发
2 /usr/bin/ssh -g -L 2222:10.10.30.1:22222 127.0.0.1
3 # 接口服务器设置代理转发
4 ssh -g -L 7805:192.168.250.100:7805 10.10.1.32
```

有时在本地转发会遇到一些问题, 比如 Connection Refused. 首先要确定本地要运行有 ssh 服务端, 使用如下命令启动 sshd:

```
1 # Mac OS X 启动 sshd
2 sudo /usr/bin/sshd
3 # Ubuntu 启动 sshd
4 sudo /etc/init.d/ssh
```

启动 SSHD 的时候系统提示:Could not load host key: /etc/ssh/ssh_ed25519_key. 新版的 opensshd 中添加了 Ed25519 做签名验证, 而之前系统里没这个算法的证书, 所以办法也很简单新生成下证书即可。

```
1 sudo ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key
2 sudo ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key
3 sudo ssh-keygen -t ecdsa -f /etc/ssh/
    ssh_host_ecdsa_key
4 ssh-keygen -t ed25519 -f /etc/ssh/
    ssh_host_ED25519_key
```

1.1.7 ssh 查看日志

如果需要查看 ssh 日志, 可在/etc/ssh/sshd_config 配置日志输出:

```
1 SysLogFacility LOCAL7
```

Facility: 设施, 是 rsyslog 引入的概念, 从功能或者程序上对日志进行分类, 并由专门的工作负责记录相对应的信息, 分类有:

auth(授权), authpriv, cron(定时任务), daemon(守护进程), kern(内核), lpr, mail(邮件相关), mark, news, security(安全), syslog, user, uucp, local0, local7。

多线程, 多协议 (udp/tcp/ssl/tls/relp); mysql, pgsql, oracle 等多种关系数据中, 强大的过滤器, 可实现过滤系统信息中的任意部分。自定义输出格式。适用于企业级别日志记录需求。ssh 配置文件配置后, 在 /etc/rsyslog.conf 文件中作如下配置:

```
1 LOCAL7.*    /var/log/sshd.log
```

1.2 VisualVM

使用 nmap 扫描 1099 端口, 看 jstatd 是否生效。

1.3 Tool Set

1.3.1 ECS(Elastic Compute Service)

配置内网 ECS 端口映射规则, 在云服务器 ECS→ 网络和安全→ 安全组→ 配置规则→ 添加安全组规则。

1.3.2 shadowsocks

启动 ss 服务器:

```
1 #可以查看启动日志
2 ssserver -c /home/ec2-user/shadowsocks.json
3 #后台启动
4 ssserver -c /home/ec2-user/shadowsocks.json -d start
```

Shadowsocks 客户端操作:

```
1 sudo apt-get install python-pip
2 sudo apt-get install python-setuptools m2crypto
3 #安装 Shadowsocks(Ubuntu/Fedora)
4 pip install shadowsocks
5 #前台启动
6 #可以看到实时的日志输出
```



```

7 #关闭终端后代理断开
8 sslocal -c /etc/shadowsocks/shadowsocks.json
9 #后台启动
10 sslocal -c /etc/shadowsocks/shadowsocks.json -d start

```

1.3.3 youtube-dl

YouTube 视频一般是不能下载的,但是是国内访问 YouTube 比较慢,经常卡顿,所以可以使用 youtube-dl 工具下载 YouTube 视频:

```

1 #下载默认的视频格式
2 youtube-dl https://www.youtube.com/watch?v=
   SnHxKQiXrFU
3 #查看所有视频格式
4 youtube-dl -F https://www.youtube.com/watch?v=
   SnHxKQiXrFU
5 #下载指定清晰度的视频
6 #137 为指定视频格式的编码 format code
7 youtube-dl -f 137 https://www.youtube.com/watch?v=
   SnHxKQiXrFU

```

查看 YouTube 所有格式视频输出效果如图所示:

```

[youtube] eq3KiAH4IBI: Downloading webpage
[youtube] eq3KiAH4IBI: Downloading video info webpage
[youtube] eq3KiAH4IBI: Extracting video information
[info] Available formats for eq3KiAH4IBI:

```

| format code | extension | resolution | note |
|-------------|-----------|--------------|---|
| 249 | webm | audio only | DASH audio 55k , opus @ 50k, 21.82MiB |
| 171 | webm | audio only | DASH audio 80k , vorbis@128k, 31.52MiB |
| 250 | webm | audio only | DASH audio 90k , opus @ 70k, 30.13MiB |
| 140 | m4a | audio only | DASH audio 99k , m4a_dash container, mp4a.40.2@128k, 40.92MiB |
| 251 | webm | audio only | DASH audio 126k , opus @160k, 44.80MiB |
| 278 | webm | 256x144 144p | 64k , webm container, vp9, 13fps, video only, 16.01MiB |
| 242 | webm | 426x240 240p | 114k , vp9, 25fps, video only, 18.26MiB |
| 160 | mp4 | 256x144 144p | 126k , avc1.42c00c, 13fps, video only, 47.61MiB |
| 243 | webm | 640x360 360p | 218k , vp9, 25fps, video only, 36.78MiB |
| 134 | mp4 | 640x360 360p | 248k , avc1.4d401e, 25fps, video only, 49.41MiB |

Figure 1.1: 查看所有格式视频

1.3.4 Jenkins

启动 Jenkins 提示 Job for jenkins.service failed because the control process exited with error code. See "systemctl status jenkins.service" and "journalctl -xe" for details. 多

半是由于 Java 未安装或者安装后未指定 Java 路径。打开文件：

```
1 #配置文件中指定 Java 路径
2 vim /etc/rc.d/init.d/jenkins
3 #启动 Jenkins
4 service jenkins start
```

配置文件中指定 Java 路径如图所示。



```
1 # Search usable Java as /usr/bin/java might
2 # see http://www.nabble.com/guinea-pigs-wan
3 candidates="21:41:03 INFO connecting ww
4 /etc/alternatives/java INFO connecting ba
5 /usr/lib/jvm/java-1.8.0/bin/java connecting ss
6 /usr/lib/jvm/jre-1.8.0/bin/java connecting ss
7 /usr/lib/jvm/java-1.7.0/bin/java connecting ww
8 /usr/lib/jvm/jre-1.7.0/bin/java connecting ss
9 /usr/bin/java 21:41:03 INFO connecting ww
10 /opt/dolphin/jdk1.8.0_161/bin/java connecting ww
11 2018-03-21 21:41:20 INFO connecting be
12 for candidate in $candidates connecting be
```

Figure 1.2: 指定 Jenkins 的 Java 路径

修改默认的端口号：

```
1 vim /etc/sysconfig/jenkins
2 service jenkins restart
```

默认密码路径：

```
1 #密码默认路径
2 vim /var/lib/jenkins/secrets/initialAdminPassword
3 vim {$home}/.jenkins/secrets/initialAdminPassword
```

忘记登录密码

忘记 Jenkins 登录密码，可以先禁用登录认证，进入 Jenkins 修改密码后，使用新密码登录，打开配置文件：

```
1 vim /var/lib/jenkins/jenkins.config
```

删除与登录认证相关的配置：

```
1 <useSecurity>true</useSecurity>
```

```
2 <authorizationStrategy class="hudson.security.  
    FullControlOnceLoggedInAuthorizationStrategy"  
    >  
3 <denyAnonymousReadAccess>true</  
    denyAnonymousReadAccess>  
4 </authorizationStrategy>  
5 <securityRealm class="hudson.security.  
    HudsonPrivateSecurityRealm">  
6 <disableSignup>true</disableSignup>  
7 <enableCaptcha>false</enableCaptcha>  
8 </securityRealm>
```

1. 重启 Jenkins 服务；
2. 进入首页 > “系统管理” > “Configure Global Security”；
3. 勾选 “启用安全”；
4. 点选 “Jenkins 专用户数据库”，并点击 “保存”；
5. 重新点击首页 > “系统管理”，发现此时出现 “管理用户”；
6. 点击进入展示 “用户列表”；
7. 点击右侧进入修改密码页面，修改后即可重新登录。

Jenkins 中 Gradle 版本

Jenkins 中使用 gradlew 时，Gradle 版本始终是某一个，但是 build.gradle 中和 Jenkins 中指定的却是另一个版本。

```
1 #查看 gradlew 版本  
2 ./gradlew -v  
3 #更新版本  
4 gradle wrapper
```

Can't retrieve the Gradle executable

Can't retrieve the Gradle executable.

1.3.5 OpenVPN

OpenVPN 从 2001 年开始开发，使用的是 C 语言。此处使用的 OpenVPN 版本是 2.4.1。如果使用 Mac 下的 brew 工具安装，则 OpenVPN 目录在：/usr/local/Cellar/openvpn/2.4.1，OpenVPN 的配置文件在：/usr/local/etc/openvpn。目前 OpenVPN

能在 Solaris、Linux、OpenBSD、FreeBSD、NetBSD、Mac OS X 与 Microsoft Windows 以及 Android 和 iOS 上运行，并包含了许多安全性的功能。此处的服务器使用的是 CentOS 7.3，客户端包含 Fedora 24、Ubuntu 14.04、Ubuntu 16.04、Windows 7、Windows 10。在 OpenVPN 网络中查看存活的主机：

```
1 nmap -A -T4 10.0.0.*
```

安装 (Install)

安装基础包：

```
1 sudo yum -y install openssl openssl-devel lzo openvpn  
    easy-rsa --allowerasing  
2 #手动安装  
3 wget -c https://swupdate.openvpn.org/community/  
    releases/openvpn-2.4.1.tar.gz  
4 tar -zxvf openvpn-2.4.1.tar.gz  
5 ./configure  
6 make && make install
```

LZO 是致力于解压速度的一种数据压缩算法，LZO 是 Lempel-Ziv-Oberhumer 的缩写。这个算法是无损算法，参考实现程序是线程安全的。实现它的一个自由软件工具是 lzop。最初的库是用 ANSI C 编写、并且遵从 GNU 通用公共许可证发布的。现在 LZO 有用于 Perl、Python 以及 Java 的各种版本。代码版权的所有者是 Markus F. X. J. Oberhumer。如果没有安装 easyrsa 工具，使用如下命令安装：

```
1 #下载 easy-rsa 源码  
2 wget -c https://github.com/OpenVPN/easy-rsa/archive/  
    master.zip  
3 #目录/etc/openvpn/easy-rsa/easyrsa3 下拷贝默认配置  
4 cp var.example vars
```

调整配置

修改证书生成的配置文件：

```
1 set_var EASYRSA_REQ_COUNTRY      "CN"  
2 set_var EASYRSA_REQ_PROVINCE     "Chongqing"  
3 set_var EASYRSA_REQ_CITY         "Chongqing"
```

```
4 set_var EASYRSA_REQ_ORG "Three people"
5 set_var EASYRSA_REQ_EMAIL "jiangtingqiang@gmail
  .com"
6 set_var EASYRSA_REQ_OU "My Organizational"
7
8 set_var EASYRSA_KEY_SIZE 2048
```

生成根证书

生成 OpenVPN 的根证书:

```
1 #清除之前生成的证书, 重新生成;
2 ./easyrsa clean-all
3 #生成 root 根证书
4 #证书路径:/etc/openvpn/easy-rsa/easyrsa3/pki/ca.crt
5 ./easyrsa build-ca
```

生成服务器端证书

生成服务器端证书命令如下:

```
1 # 指定服务端证书的文件名为 server, 可以任意改动
2 ./easyrsa build-server-full server
```

生成客户端证书

生成客户端证书端步骤如下:

```
1 ./easyrsa build-client-full jiangxiaoqiang
```

PKI: Public Key Infrastructure 公钥基础设施。生成请求:

```
1 ./easyrsa gen-req dolphinfedora
```

输入 PEM 验证码。PEM - Privacy Enhanced Mail, 打开看文本格式, 以"—BEGIN..." 开头, "—END..." 结尾, 内容是 BASE64 编码。Apache 和 *NIX 服务器偏向于使用这种编码格式。签约:

```
1 #切换到服务端生成 rsa 的目录
2 #导入 req
```

```
3 ./easyrsa import-req ~/client/easyrsa/easy-rsa-master
    /easyrsa3/pki/reqs/dolphinfedora.req
    dolphinfedora
4 #用户签约，根据提示输入服务端的 ca 密码
5 ./easyrsa sign client dolphinfedora
```

PKI: Public Key Infrastructure 公钥基础设施。输入 PEM 验证码。PEM - Privacy Enhanced Mail, 打开看文本格式, 以"—BEGIN..." 开头, "—END..." 结尾, 内容是 BASE64 编码. 查看 PEM 格式证书的信息:openssl x509 -in certificate.pem -text -noout。Apache 和 *NIX 服务器偏向于使用这种编码格式. 服务端生成的文件有:

| 文件名称 | 说明 (Purpose) | 位置 |
|--------------------|--------------------------------|--------------------------|
| ca.crt | 根证书 (Root CA certificate) 件 | Server+All Clients |
| reqs/server.req | | |
| reqs/dolphin.req | | |
| private/ca.key | 根证书私钥 (Root CA key) | key signing machine only |
| private/server.key | | |
| issued/server.crt | 服务器证书 Server Certificate | server only |
| issued/dolphin.crt | | |
| dh.pem | Diffie Hellman parameters | server only |

客户端生成的文件有:

| 序号 | 名称 |
|----|---------------------------|
| | private/dolphinclient.key |
| | reqs/dolphinclient.req |

拷贝出客户端证书文件:

```
1 cp easyrsa/easy-rsa-master/easyrsa3/pki/ca.crt ~/
    dolphinfedora/
2 cp easyrsa/easy-rsa-master/easyrsa3/pki/issued/
    dolphinfedora.crt ~/dolphinfedora/
3 cp ~/client/easyrsa/easy-rsa-master/easyrsa3/pki/
    private/dolphinfedora.key ~/dolphinfedora/
```

启动 OpenVPN:

```
1 sudo openvpn server.conf
```

```

2 # Mac 下启动 OpenVPN
3 sudo /usr/local/Cellar/openvpn/2.4.1/sbin/openvpn /
    usr/local/etc/openvpn/client.conf
4 # 需要以后台交互方式启动时
5 screen sudo openvpn client.conf

```

客户端配置如下:

```

1 client          #指定当前 VPN 是客户端
2 dev tun         #必须与服务器端的保持一致
3 proto udp       #必须与服务器端的保持一致
4 #指定连接的远程服务器的实际 IP 地址和端口号
5 remote 192.168.1.106 1194
6 #断线自动重新连接
7 #在网络不稳定的情况下（例如：笔记本电脑无线网络）非常
    有用
8 resolv-retry infinite
9 nobind          #不绑定特定的本地端口号
10 persist-key
11 persist-tun
12 ca ca.crt       #指定 CA 证书的文件路径
13 cert client1.crt #指定当前客户端的证书文件路径
14 key client1.key  #指定当前客户端的私钥文件路径
15 ns-cert-type server #指定采用服务器校验方式
16 #如果服务器设置了防御 DoS 等攻击的 ta.key
17 #则必须每个客户端开启；如果未设置，则注释掉这一行；
18 tls-auth ta.key 1
19 comp-lzo        #与服务器保持一致
20 #指定日志文件的记录详细级别，可选 0-9，等级越高日志内
    容越详细
21 verb 3

```

配置 ns-cert-type(Netscape Cert Type) 指定为 server 主要是防止中间人攻击 (Man-in-the-Middle Attack)。在服务端做如下配置:

```

1 nsCertType server

```

生成 Diffie Hellman 参数

生成 Diffie Hellman 参数命令如下:

```
1 ./easymrsa gen-dh
```

初始化配置文件

初始化配置文件如下:

```
1 #拷贝示例配置到配置目录
2 cp /usr/share/doc/openvpn-2.4.4/sample/sample-config-
   files/server.conf /etc/openvpn
```

拷贝 server.crt/ca.cert 等文件到配置目录.

生成 ta.key

使用如下命令生成 ta.key:

```
1 openvpn --genkey --secret ta.key
```

启动服务端:

```
1 openvpn --config /etc/openvpn/server.conf
```

拷贝客户端文件

将服务器生成的客户端文件拷贝到客户端:

1.3.6 fastDFS

启动 tracker server:

```
1 /usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf
```

启动 stored server:

```
1 /usr/bin/fdfs_storaged /etc/fdfs/storage.conf
```


2. Gradle

一般在 Mac 下 Gradle 的安装路径为: /usr/local/Cellar/gradle/3.2.1/libexec, 在 Mac 的 IntelliJ Idea 指定 Gradle 路径时, 即用此路径。

```
1 ${PROGRAM_DEVELOP_PATH_BACKEND}/gradlew -p ${  
    PROGRAM_DEVELOP_PATH_BACKEND}/cc-web-boot -x  
    test build --info
```

info 参数会输出详细的 log 信息, 便于问题定位。

2.1 基础

Gradle 使用的 Groovy 语言, Groovy 语言字符串的表示形式很多, 有一些细微的区别:

```
1 def a = '单引号形式'  
2 def b = "双引号形式, ${v}"  
3 def c = '''三个单引号形式  
4 支持多行  
5 不支持占位符'''  
6 def d = """三个双引号形式  
7 支持多行  
8 支持占位符, ${v}"""
```

```
9 def e = /反斜杠形式
10 支持多行
11 支持占位符, ${v}/
12 def f = $/美刀反斜杠形式
13 支持多行
14 支持占位符, ${v}/$
```

建议使用'和"这两种形式。单引号''中的内容严格对应 Java 中的 String，不对 \$ 符号进行转义。双引号""的内容则和脚本语言的处理有点像，如果字符中有 \$ 号的话，则它会 \$ 表达式先求值。三个引号'''xxx''' 中的字符串支持随意换行。

2.1.1 引用本地文件

有时候项目依赖的可能不是公布在网络上的一个公共库，可能只是一个闭源的 jar 包，只是一个本地文件而已。Gradle 中引用本地文件写法如下：

```
1 def ccDataLibs = fileTree(dir: "$rootDir/cc-data/lib"
    , include: '*.jar')
2
3 dependencies {
4     compile ccDataLibs
5 }
```

dir 表示文件所在目录，include 表示包含哪些文件。

2.1.2 Gradle Properties 支持

Gradle 支持三种 Properties，这三种 Properties 的作用域和初始化阶段都不一样：

System Properties： 可通过 gradle.properties 文件，环境变量或命令行-D 参数设置，可在 setting.gradle 或 build.gradle 中动态修改，在 setting.gradle 中的修改对 buildscript block 可见；所有工程可见，不建议在 build.gradle 中修改。多子工程项目中，子工程的 gradle.properties 会被忽略掉，只有 root 工程的 gradle.properties 有效。

Project Properties： 可通过 gradle.properties 文件，环境变量或命令行-P 参数设置，优先级是：可在 build.gradle 中动态修改，但引用不存在的 project properties 会立即抛错，动态修改过的 project properties 在 buildscript block 中不可见。

Project ext properties： 可在项目的 build.gradle 中声明和使用，本工程和子工程可见，不能在 setting.gradle 中访问，如果有多处设置，加载次序如下（注意：gradle 2.0 是这样的，1.10 1.12 有 bug），后面的覆盖前面的设置。

- from gradle.properties located in project build dir.
- from gradle.properties located in gradle user home.
- from system properties, e.g. when -Dsome.property is used in the command line.
- setting.gradle
- build.gradle

2.1.3 执行流程

There is a one-to-one relationship between a Project and a "build.gradle" file. During build initialisation, Gradle assembles a Project object for each project which is to participate in the build, as follows:

Create a Settings instance for the build. Evaluate the "settings.gradle" script, if present, against the Settings object to configure it. Use the configured Settings object to create the hierarchy of Project instances. Finally, evaluate each Project by executing its "build.gradle" file, if present, against the project. The projects are evaluated in breadth-wise order, such that a project is evaluated before its child projects. This order can be overridden by calling `evaluationDependsOnChildren()` or by adding an explicit evaluation dependency using `evaluationDependsOn(String)`.

2.1.4 默认 JVM

在 IntelliJ Idea 中首次导入 Gradle 项目时，可能提示未设置 Gradle JVM，此时在项目初始选择页面选择 `Configure > Project Defaults > Project Structure`，在项目 Structure 中设置 JVM 即可。

2.1.5 Repositories

Gradle 默认每次编译或者刷新 (Refresh) 都会 Resolve Dependencies，每次都会去进行网络请求。可以通过 Wireshark 抓包来验证是否有网络请求以及请求的具体地址，从图中可以看出 Gradle Resolve 时的请求地址，如果没有调整请求地址，Gradle 会使用默认的 Maven 仓库的地址或 JCenter 地址，众所周知，访问 Maven 仓库非常慢，导致 Gradle 一直卡在 Resolve Dependencies 阶段，具体如图 2.1 所示。

如果访问 JCenter 很慢，光是这个步骤说不定就要十几分钟。解决方法是在 Project Preferences 的 Gradle 那里启用 Offline work。注意这样之后当你添加了新的 dependencies，它是不会去自动下载的，也就是说报错。需要进入项目目录运行 `gradlew` 手动同步，或是临时在 Preferences 那里禁用 Offline work。也可以设置国内的源，在 home 目录 gradle 文件夹中新建 `init.gradle` 文件，并在文件中输入如下内容，指定默认的国内源的地址，加速依赖包下载速度：

| http | | | | | |
|---------|-----------------|----------------|-----------------|----------|------------------|
| No. | Time | Source | Destination | Protocol | Length |
| 2040... | 1396.5092843... | 192.168.31.201 | 124.203.224.145 | HTTP | 490 GET /moviets |
| 2052... | 1399.5583254... | 192.168.31.201 | 124.203.224.145 | HTTP | 490 GET /moviets |
| 2060... | 1402.0761734... | 192.168.31.201 | 124.203.224.145 | HTTP | 490 GET /moviets |
| 2067... | 1403.8083968... | 192.168.31.201 | 124.203.224.145 | HTTP | 490 GET /moviets |

▶ Frame 13: 394 bytes on wire (3152 bits), 394 bytes captured (3152 bits) on interface 0
 ▶ Ethernet II, Src: HonHaiPr_f7:58:66 (70:18:8b:f7:58:66), Dst: XiaomiCo_53:92:b4 (64:09:8c:53:92:b4)
 ▶ Internet Protocol Version 4, Src: 192.168.31.201, Dst: 114.215.66.82
 ▶ Transmission Control Protocol, Src Port: 55530, Dst Port: 8078, Seq: 1, Ack: 1, Len: 340
 ▼ Hypertext Transfer Protocol
 ▶ HEAD /org/springframework/boot/spring-boot-gradle-plugin/1.5.6.RELEASE/spring-boot-gra
 Cache-Control: max-age=0\r\n
 Host: repox.gttn.com:8078\r\n
 Connection: Keep-Alive\r\n
 User-Agent: Gradle/3.2.1 (Linux;4.8.14-200.fc24.x86_64;amd64) (Oracle Corporation;1.8.
 Accept-Encoding: gzip,deflate\r\n
 \r\n
 [Full request URI: http://repor.gttn.com:8078/org/springframework/boot/spring-boot-gra
 [HTTP request 1/43]
 [Response in frame: 17]
 [Next request in frame: 19]

Figure 2.1: Wireshark 查看 Gradle Resolve 请求

```

1 allprojects{
2   repositories {
3     def ALIYUN_REPOSITORY_URL = 'http://maven.aliyun.com/
      nexus/content/groups/public'
4     def ALIYUN_JCENTER_URL = 'http://maven.aliyun.com/
      nexus/content/repositories/jcenter'
5     all { ArtifactRepository repo ->
6       if(repo instanceof MavenArtifactRepository){
7         def url = repo.url.toString()
8         if (url.startsWith('https://repo1.maven.org/maven2'))
9           {
10            project.logger.lifecycle "Repository ${repo.url}
              replaced by $ALIYUN_REPOSITORY_URL."
11          }
12        if (url.startsWith('https://jcenter.bintray.com/')) {
13          project.logger.lifecycle "Repository ${repo.url}
              replaced by $ALIYUN_JCENTER_URL."
14          remove repo
15        }
16      }
17    }
18    maven {

```

```
19 url ALIYUN_REPOSITORY_URL
20 url ALIYUN_JCENTER_URL
21 }
22 }
23 }
```

在 Gradle 构建标本 build.gradle 里，经常会看到如下脚本：

```
1 repositories {
2     maven {
3         url 'http://www.eveoh.nl/files/maven2'
4     }
5     maven {
6         url 'http://repox.gttn.com:8078'
7     }
8     mavenCentral()
9     jcenter()
10    maven {
11        url 'http://repo.spring.io/plugins-release'
12    }
13 }
```

总的来说，只有两个标准的 Android library 文件服务器：JCenter 和 Maven Central。起初，Android Studio 选择 Maven Central 作为默认仓库。如果你使用老版本的 Android Studio 创建一个新项目，mavenCentral() 会自动的定义在 build.gradle 中。但是 Maven Central 的最大问题是对开发者不够友好。上传 library 异常困难。上传上去的开发者都是某种程度的极客。同时还因为诸如安全方面的其他原因，Android Studio 团队决定把默认的仓库替换成 JCenter。正如你看到的，一旦使用最新版本的 Android Studio 创建一个项目，JCenter() 自动被定义，而不是 mavenCentral()。mavenCentral() 表示依赖是从 Central Maven 2 仓库中获取的，库的地址是<https://repo1.maven.org/maven2>。JCenter 表示依赖是从 Bintary' s JCenter Maven 仓库中获取的，仓库的地址是<https://jcenter.bintray.com>，Bintray 是一家提供全球企业软件开发包托管的商业公司。

2.1.6 常见问题

could not find method compile()

gradle 项目配置中引用 java 插件。

Could not find or load main class org.gradle.wrapper.GradleWrapperMain

运行 gradlew 脚本时, 出现错误: Could not find or load main class org.gradle.wrapper.GradleWrapperMain。安装 gradle 后运行 gradle wrapper 命令修复。

Gradle 找不到 Tasks

有时在使用 Gradle 命令编译时找不到相应任务 (Tasks)。比如在部署脚本在 project/script/deploy 目录下, 当在此目录下运行脚本时, 会提示在项目目录 deploy 下找不到任务。命令如下:

```
1 ${PROGRAM_DEVELOP_PATH_BACKEND}/gradlew copyTask -x  
   test
```

此时就需要指定项目的路径, 将命令调整为:

```
1 ${PROGRAM_DEVELOP_PATH_BACKEND}/gradlew -p ${  
   PROGRAM_DEVELOP_PATH_BACKEND} copyTask -x  
   test
```

p 参数指定项目的路径。在不同目录下面执行 gradle tasks 命令, 可以看到不同的任务, 当在 /project/script/deploy 目录下执行 gradle tasks 命令时, 是看不到 copyTask 命令的, 但是在项目的根目录下执行是可以看到 copyTasks 任务, 如图2.2所示。

```
Verification tasks  
-----  
check - Runs all checks.  
test - Runs the unit tests.  
  
Other tasks  
-----  
copyTask  
wrapper - Generates gradlew[.bat] scripts
```

Figure 2.2: Gradle 查看 Tasks

2.2 Gradle 对象

2.2.1 属性 (Properties)

Extra Properties

extra 属性一般用于定义常量，All extra properties¹ must be defined through the "ext" namespace. Once an extra property has been defined, it is available directly on the owning object (in the below case the Project, Task, and sub-projects respectively) and can be read and updated. Only the initial declaration that needs to be done via the namespace.

```
1 buildscript {  
2   ext {  
3     springBootVersion = '1.4.5.RELEASE'  
4     jacksonVersion = '2.8.7'  
5     springfoxVersion = '2.6.1'  
6     poiVersion = "3.14"  
7     aspectjVersion = '1.7.4'  
8   }  
9 }
```

Reading extra properties is done through the "ext" or through the owning object.

```
ext.isSnapshot = version.endsWith("-SNAPSHOT") if (isSnapshot) // do snapshot  
stuff
```

定义目标 jar 包名

有时在构建后需要定义目标 jar 包的名称。

```
1 project(":web"){  
2   description = "web"  
3   jar{  
4     baseName = "dolphin-web"  
5   }  
6 }
```

编译打包后的 jar 包名称即为 dolphin-web.jar。

¹<https://docs.gradle.org/current/javadoc/org/gradle/api/Project.html#extraproperties>

2.2.2 Task

对于 build.gradle 配置文件，当运行 Gradle <Task> 时，Gradle 会为我们创建一个 Project 的对象，来映射 build.gradle 中的内容。其中呢，对于不属于任何 Task 范畴的代码，Gradle 会创建一个 Script 类的对象，来执行这些代码；对于 Task 的定义，Gradle 会创建 Task 对象，并将它作为 project 的属性存在 (实际上是通过 getTaskName 完成的)。

拷贝文件

在项目中需要根据不同的用途生成不同的文件，比如 api 需要单独打包，平台需要单独打包，但是 api 和平台的代码都是一套代码，只是配置不同罢了。此时需要 Gradle 在构建之后根据不同的用途生成不同的包名称，使用 copy 功能来实现。首先定义项目的动态属性：

```
1 buildscript {
2     ext {
3         projectVersion = '1.1.11'
4     }
5 }
```

然后在 task 中使用此属性即可。

```
1 task copyfile(){
2     println(projectVersion)
3 }
```

一个简单的拷贝任务，在生成完毕文件后，目的文件拷贝一个接口部署包副本，接口与应用程序单独部署，并重新命名。

```
1 def ccCommonBuildScript = file("$rootDir/gradle/
   common.gradle")
2 apply from: ccCommonBuildScript
3 /**
4  * 复制一份接口的部署包
5  * 接口与平台单独部署，避免互相影响
6  */
7 task copyTask(type: Copy) {
8     from 'build/libs/credit-system-web-boot-' +
       version + '.jar'
```



```
9      into 'build/libs/api'
10      rename 'credit-system-web-boot-' + version + '.
        jar','credit-system-web-api-' + version + '.
        jar'
11  }
12
13  /**
14   * 复制一份接口的部署包
15   * 接口与平台单独部署，避免互相影响
16   * 指定路径
17   */
18  task copyTask(type: Copy) {
19      from "$projectDir/build/libs/credit-system-web-
        boot-${version}.jar"
20      into "$projectDir/build/libs/api"
21      rename "credit-system-web-boot-${version}.jar","
        credit-system-web-api-${version}.jar"
22  }
```

在这里，version 变量是来自另外一个 gradle 文件。version 变量在 common.gradle 中定义，在 build.gradle 文件中使用。

| | | |
|----------|----------------------|-----------|
| 3 | MariaDB | 29 |
| 3.1 | 常用操作 | |
| 3.2 | 常见问题 | |

| | | |
|--|---------------------------|-----------|
| | Bibliography | 33 |
|--|---------------------------|-----------|

Books

Articles

3. MariaDB

3.1 常用操作

在 CentOS 下启动 MariaDB:

```
1 systemctl start mariadb.service
```

3.1.1 导入导出

导出整个库结构和数据:

```
1 mysqldump -h localhost -uroot -p123456 database >  
   dump.sql
```

启动 MariaDB 时注意是否带了 `--skip-networking` 参数，否则会出现数据库启动了，但是端口没有监听的情况。

允许远程登录

首先以 root 用户登陆 MariaDB 服务器:

```
1 --允许用户名为'dolphin'的用户从任意ip以密码为123456访  
   问所有数据库  
2 grant all PRIVILEGES on *.* to dolphin@'%' identified  
   by '123456';
```

```
3 --使其生效
4 flush privileges;
```

修改数据库密码

修改数据库密码如下：

```
1 #登录 SQL
2 mysql -uroot -p
3 use mysql;
4 UPDATE user SET password=password('newpassword')
      WHERE user='root';
5 flush privileges;
6 exit;
```

3.2 常见问题

3.2.1 mariadb 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)

在使用命令登录时, 出现错误. 依次执行如下语句即可:

```
1 #停止 MariaDB
2 systemctl stop mariadb.service
3 #登录
4 mysqld_safe --user=mysql --skip-grant-tables --skip-
      networking &
5 mysql -u root mysql
6 #设置新密码
7 UPDATE user SET Password=PASSWORD('123123') where
      USER='root';
8 FLUSH PRIVILEGES;
9 quit
10 #启动 MariaDB
11 systemctl start mariadb.service
```

修改后, 登录数据库:

```
1 mysql -u root -p123123
```




Bibliography

Books

- [1] 林沛满. *Wireshark 网络分析的艺术*. 1st ed. 人民邮电出版社, 2016, pp. 3–6.
ISBN: 978-7-115-41021-4.

Articles

