统信 uos 运维工程师

编写：培训认证部　　版本：0.20. 0916

统信软件
UNIONTECH

统信软件技术有限公司

第 1 章 Apache2 服务

1. 1 Apache2 简介

Apache2 服务的历史和工作原理

Apache 介绍

Apache 是世界使用排名第一的 Web 服务器软件。它可以运行在几乎所有广泛使用的计 算机平台上。

Apache 源于 NCSAhttpd 服务器，经过多次修改，成为世界上最流行的 Web 服务器软件 之一。Apache 取自"a patchy server"的读音，意思是充满补丁的服务器，因为它是自由软 件，所以不断有人来为它开发新的功能、新的特性、修改原来的缺陷。Apache 的特点是简 单、速度快、性能稳定，并可做代理服务器来使用。

HTTP 协议简介

HTTP 协议，全称 HyperText Transfer Protocol,中文名为超文本传输协议，是互联网 中最常用的一种网络协议。HTTP 的重要应用之一是 WWW 服务。设计 HTTP 协议最初目的就是 提供一种发布和接收 HTML （一种页面标记语言）页面的方法（请求返回）。

HTTP 协议是互联网上常用的通信协议之一。它有很多的应用，但最流行的就是用于 Web 浏 览器和 Web 服务器之间的通信，即 WWW 应用或称 Web 应用。

WWW,全称 World Wide Web,常称为 Web,中文译为“万维网”。它是目前互联网上最 受用户欢迎的信息服务形式。HTTP 协议的 WWW 服务应用的默认端口为 80 （端口的概念）， 另外的一个加密的 WWW 服务应用 https 的默认端口为 443,主要用于网银，支付等和钱相关 的业务。当今，HTTP 服务，WWW 服务，Web 服务三者的概念已经混淆了，都是指当下最常见 的网站服务应用。

Apache2 的模块类型

检查启用的 **Apache** 模块

Apache 基于模块化的理念而构建,这样就可以让 web 管理员添加不同的模块来扩展主要的 功能及增强性能。列出或查看已经启用的 Apache 服务器启动了哪些模块，可以在运行运行 如下命令来确认：

```
$ apache2ctl -t -D DUMP_MODULES 或者
$ apache2ctl -M
```

常见的 **Apache** 模块有：

- mod_ssl -提供了 HTTPS 功能。
- mod_rewrite -可以用正则表达式匹配 url 样式，并且使用.htaccess 技巧来进行 透时转发，或者提供 HTTP 状态码回应。
- mod_security -用于保护 Apache 免于暴力破解或者 DDoS 攻击。
- mod_status -用于监测 Apache 的负载及页面统计。

使能模块（加载模块）：只有加载的模块才能起作用

a2enmod〈模块名〉

已安装的 apache2 模块可以在/etc/apache2/mods_available 找到

模块列表

mod_actions	基于媒体类型或请求方法，为执行 CGI 脚本而提供
mod_alias	提供从文件系统的不同部分到文档树的映射和 URL 重定向
mod_asis	发送自己包含 HTTP 头内容的文件
mod_auth_basic	使用基本认证
mod_auth_digest	使用 MD5 摘要认证(更安全，但是只有最新的浏览器才支持)
mod_authn_alias	基于实际认证支持者创建扩展的认证支持者，并为它起一个别名以便于 引用
mod_authn_anon	提供匿名用户认证支持
mod_authn_dbd	使用 SQL 数据库为认证提供支持
mod_authn_dbm	使用 DBM 数据库为认证提供支持
mod_authn_default	在未正确配置认证模块的情况下简单拒绝一切认证信息
mod_authn_file	使用纯文本文件为认证提供支持
mod_authnz_ldap	允许使用一个 LDAP 目录存储用户名和密码数据库来执行基本认证和授 权
mod_authz_dbm	使用 DBM 数据库文件为组提供授权支持
mod_authz_default	在未正确配置授权支持模块的情况下简单拒绝一切授权请求
mod_authz_groupfile	使用纯文本文件为组提供授权支持
mod_authz_host	供基于主机名、IP 地址、请求特征的访问控制
mod_authz_owner	基于文件的所有者进行授权
mod_authz_user	基于每个用户提供授权支持
mod_autoindex	自动对目录中的内容生成列表，类似于"Is"或"dir"命令
mod_cache	基于 URI 键的内容动态缓冲(内存或磁盘)
mod_cern_meta	允许 Apache 使用 CERN httpd 元文件，从而可以在发送文件时对头进行 修改
mod_cgi	在非线程型 MPM(prefork)上提供对 CGI 脚本执行的支持
mod_cgid	在线程型 MPM(worker) 上用一个外部 CGI 守护进程执行 CGI 脚本
mod_charset_lite	允许对页面进行字符集转换

mod_dav	允许 Apache 提供 DAV 协议支持
mod_dav_fs	为 mod.dav 访问服务器上的文件系统提供支持
mod_dav_lock	为 mod.dav 锁定服务器上的文件提供支持
mod_dbd	管理 SQL 数据库连接，为需要数据库功能的模块提供支持
mod_deflate	压缩发送给客户端的内容
mod_dir	指定目录索引文件以及为目录提供"尾斜杠"重定向
mod_disk_cache	基于磁盘的缓冲管理器
mod_dumpio	将所有 I/O 操作转储到错误日志中
mod_echo	一个很简单的协议演示模块
mod_env	允许 Apache 修改或清除传送到 CGI 脚本和 SSI 页面的环境变量
mod_example	一个很简单的 Apache 模块 API 演示模块
mod_expires	允许通过配置文件控制 HTTP 的"Expires:"和"Cache-Control:"头内容
mod_ext_filter	使用外部程序作为过滤器
mod_file_cache	提供文件描述符缓存支持，从而提高 Apache 性能
mod_filter	根据上下文实际情况对输出过滤器进行动态配置
mod_headers	允许通过配置文件控制任意的 HTTP 请求和应答头信息
mod_ident	实现 RFC1413 规定的 ident 查找
mod_imagemap	处理服务器端图像映射
mod_include	实现服务端包含文档（SSI）处理
mod_info	生成 Apache 配置情况的 Web 页面
mod_isapi	仅限于在 Windows 平台上实现 ISAPI 扩展
mod_ldap	为其它 LDAP 模块提供 LDAP 连接池和结果缓冲服务
mod_log_config	允许记录日志和定制日志文件格式
mod_log_forensic	实现"对比日志”，即在请求被处理之前和处理完成之后进行两次记录
mod_logio	对每个请求的输入/输出字节数以及 HTTP 头进行日志记录
mod_mem_cache	基于内存的缓冲管理器
mod_mime	根据文件扩展名决定应答的行为（处理器/过滤器）和内容（MIME 类型/语言/字符集/编码）

mod_mime_magic	通过读取部分文件内容自动猜测文件的 MIME 类型

mod_negotiation	提供内容协商支持
mod_nw_ssl	仅限于在 NetWare 平台上实现 SSL 加密支持
mod_proxy	提供 HTTP/1. 1 的代理/网关功能支持
mod_proxy_ajp	mod_proxy 的扩展，提供 Apache JServ Protocol 支持
mod_proxy_balancer	mod_proxy 的扩展，提供负载平衡支持
mod_proxy_connect	mod_proxy 的扩展，提供对处理 HTTP CONNECT 方法的支持
mod_proxy_ftp	mod_proxy 的 FTP 支持模块
mod_proxy_http	mod_proxy 的 HTTP 支持模块
mod_rewrite	一个基于一定规则的实时重写 URL 请求的引擎
mod_setenvif	根据客户端请求头字段设置环境变量
mod_so	允许运行时加载 DSO 模块
mod_speling	自动纠正 URL 中的拼写错误
mod_ssl	使用安全套接字层(SSL)和传输层安全(TLS)协议实现高强度加密传输
mod_status	生成描述服务器状态的 Web 页面
mod_suexec	使用与调用 web 服务器的用户不同的用户身份来运行 CGI 和 SSI 程序
mod_unique_id	为每个请求生成唯一的标识以便跟踪
mod_userdir	允许用户从自己的主目录中提供页面(使用"厂 username")
mod_usertrack	使用 Session 跟踪用户(会发送很多 Cookie),以记录用户的点击流
mod_version	提供基于版本的配置段支持
mod_vhost_alias	提供大批量虚拟主机的动态配置支持
libmysql	mysql 模块
php5apache2_2. dll	Php 模块

web 服务器端的工作流程

(1) 客户端发送请求

客户端(通过浏览器)和 Web 服务器建立 TCP 连接，连接建立以后，向 Web 服务器发出 访问请求(如 get) 0 根据 HTTP 协议，该请求中包含了客户端的 IP 地址、浏览器的类型和 请求的 URL 等一系列信息。

(2) 服务器解析请求

Web 服务器对请求按照 HTTP 协议进行解码来确定进一步的动作，设计的内容有三要点： 方法(GET)、文档(/sample. html)＞和浏览器使用的协议(HTTP/1. 1)其中方法告诉服务

器应完动的动作，GET 方法的含义很明显是：服务器应定位、读取文件并将它返回给客户。

Web 服务器软件现在就知道了，它应该找到文件/sample, html,并使用 HTTP/1. 1 协议将内 存返回给客户。信息是经过与请求到来相同的连接发出的，所以服务器不需要定们客户或创 建新的连接。

（3）读取其它信息（非必须步骤）

Web 服务器根据需要去读取请求的其它部分。在 HTTP/1. 1 下，客户还应给服务器提 供关于它的一些信息。元信息（metainformation）可用来描述浏览器及其能力，以使服务 器能据此确定如何返回应答。

（4）完成请求的动作

若现在没有错误出现，WWW 服务器将执行请求所要求的动作。要获取（GET） 一个文档， web 服务器在其文档树中搜索请求的文件（/sample. html）。这是由服务器机器上作为操作 系统一部分的文件系统完成的。若文件能找到并可正常读取，则服务器将把它返回给客户。 如果成功：文件被发送出去。

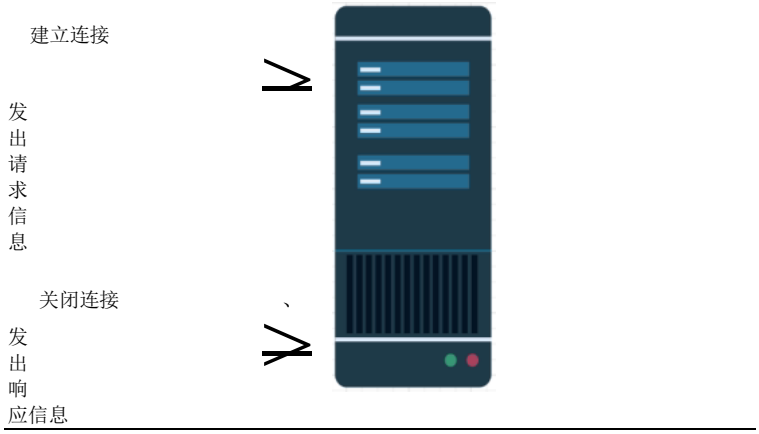
首先，web 服务器发送一个状态码及一些描述信息。既然文件已经找到，则发送状态码 200,表示一切都 OK ,文档随后发出，因为发送的信息是 HTML 文档，所以 Content-type 取 值为 text/htmlo 文档长为 1024 个字节，所以 Content-type 取 1024。服务器软件的标识 及文件的时间属性信息也被包含在头域中。

如果失败：返回错误指示。

如果请求的文件没有找到或找到但无法读取，测请求无法满足。这时将返回不同于 200 的状态码。最常见的问题是请求中的文件名拼写有误，所以服务器无法找到该文件。这种情 况下，服务器将发送一个状态码—404 给客户。

（5）关闭文件和网络连接，结束会话。

当文件已被发邮或错误已发出后，web 服务器结束整个会话。它关闭打开的的被请求文件， 关闭网络端口从而结束网络连接。有关的其它工作则是由客户端来完成的，包括接收数据， 并以用户可读的方式呈现出来。这些与服务器无关。



1.2 安装 Apache2 软件

配置文件解析

apache 各文件存放路径

- 默认 Web 目录： /var/ww/html/
- 配置目录： ,/etc/apache2/
- 全局配置文件： ,/etc/apache2/apache2. conf
- 端口配置文件： /etc/apache2/ports. conf
- 虚拟主机配置文件： ,/etc/apache2/sites-enabled/000-default.conf
- Apache2 环境变量设置： envvars
- mods-available： 这个目录包含模块和模块配置文件。不是所有的模块都有配置文件。
- mods-enabled :持有/etc/apache2/mods-available 目录下文件的链接，当该目录下 有一个模块文件和其配置文件，那么 Apache 重启后该模块将生效。
- sites-available :这个目录包含 Apache 虚拟主机的配置文件。虚拟主机允许 Apache 配置多个站点并为每个站点配置不同的参数。后面下面配置的时候会配置 80 端口的 http 重定向为 443 的 httpso
- sites-enabled： 持有/etc/apache2/sites-available 目录下文件的链接。当 Apache 重启后，该目录中包含的站点将会被激活

apache 配置文件详解：

- <VirtualHost *:自定义端口>
- ServerName www. uos. com #在 ServerName 后加上你的网站名称
- ServerAlias ftp. uos. com mail. uos. com #如果你想多个网站名称都取得相同的网站，可以加在 ServerAlias 后加上其他网站别名。别名间以空格隔开
- ServerAdmin uos@pxb. com #在 ServerAdmin 后加上网站管理员的电邮地址，方便别人有问题是可以联络网站管理员

#定义“/”目录区域的开始 #在 DocumentRoot 后加上存放网站内容的目录路
#控制选项允许使用软链接
#不允许隐含控制文件中的覆盖配置
#访问控制策略的应用顺序
#禁止任何人访问此区域
#定义“/”目录区域的结束

HTTP 相应状态码

HTTP 状态码

* DocumentRoot /var/ww/html 径（用户的个人目录）

- </VirtualHost>
- <Directory />
- Options FollowSymLinks
- AllowOverride None
- Order allow, deny
- Allow from all
- </Directory>

当浏览者访问一个网页时，浏览者的浏览器会向网页所在服务器发出请求。当浏览器接 收并显示网页前，此网页所在的服务器会返回一个包含 HTTP 状态码的信息头(server hea der)用以响应浏览器的请求。HTTP 状态码的英文为 HTTP Status Code。

下面是常见的 HTTP 状态码：

- 200 -请求成功
- 301 -资源（网页等）被永久转移到其它 URL
- 404 -请求的资源（网页等）不存在
- 500 -内部服务器错误

HTTP 状态码分类

HTTP 状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型，后两个数字没有分类的作用。HTTP 状态码共分为 5 种类型：

HTTP 状态码分类

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误， 清求包含语法错误或无法完成请求
5**	服务器错误， 服务器在处理请求的过程中发生了错误

HTTP 状态码列表： HTTP 状态码列表

状态码	状态码英文名称	中文描述
100	Continue	继续。客户端应继续其请求
101	Switching Protocols	切换协议。服务器根据客户端的请求切换协议。 只能切换到更高级的协议，例如，切换到 HTTP 的新版本协议
200	OK	请求成功。一般用于 GET 与 POST 请求
201	Created	己创建。成功请求并创建了新的资源

202	Accepted	已接受。已经接受请求，但未处理完成
203	Non-Authoritative Information	非授权信息。请求成功。但返回的 meta 信息不在原始的服务器，而是一个副本
204	No Content	无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
205	Reset Content	重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
206	Partial Content	部分内容。服务器成功处理了部分 GET 请求
300	Multiple Choices	多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301	Moved Permanently	永久移动。请求的资源已被永久的移动到新 URI,返回信息会包括新 的 URI,浏览器会自动定向到新 URL 今后任何新的请求都应使用新的 URI 代替
302	Found	临时移动。与 301 类似。但资源只是临时被移动。客户端应继续使用原有 URI
303	See Other	查看其它地址。与 301 类似。使用 GET 和 POST 请求查看
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
305	Use Proxy	使用代理。所请求的资源必须通过代理访问
306	Unused	已经被废弃的 HTTP 状态码
307	Temporary Redirect	临时重定向。与 302 类似。使用 GET 请求重定向
400	Bad Request	客户端请求的语法错误，服务器无法理解

401	Unauthorized	请求要求用户的身份认证
402	Payment Required	保留，将来使用
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置〃您所请求的资源无法找到"的个性页面
405	Method Not Allowed	客户端请求中的方法被禁止
406	Not Acceptable	服务器无法根据客户端请求的内容特性完成请求
407	Proxy Authentication Required	请求要求代理的身份认证，与 401 类似，但请求者应当使用代理进行授权
408	Request Time-out	服务器等待客户端发送的请求时间过长，超时
409	Conflict	服务器完成客户端的 PUT 请求时可能返回此代码，服务器处理请求时发生了冲突
410	Gone	客户端请求的资源已经不存在。410 不同于 404,如果资源以前有现在被永久删除了可使用 410 代码，网站设计人员可通过 301 代码指定资源的新位置
411	Length Required	服务器无法处理客户端发送的不带 Content-Length 的请求信息
412	Precondition Failed	客户端请求信息的先决条件错误
413	Request Entity Too Large	由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个 Retry-After 的响应信息
414	Request-URI Too Large	请求的 URI 过长（URI 通常为网址），服务器无法处理
415	Unsupported Media Type	服务器无法处理请求附带的媒体格式

416	Requested range not satisfiable	客户端请求的范围无效
417	Expectation Failed	服务器无法满足 Expect 的请求头信息
500	Internal Server Error	服务器内部错误，无法完成请求
501	Not Implemented	服务器不支持请求的功能，无法完成请求
502	Bad Gateway	作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应
503	Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的 Retry-After 头信息中
504	Gateway Time-out	充当网关或代理的服务器，未及时从远端服务器获取请求
505	HTTP Version not supported	服务器不支持请求的 HTTP 协议的版本，无法完成处理

基本配置和维护

安装 apache2

apt install -y apache2 启动 Apache 服务：

systemctl start apache2

systemctl enable apache2 启动完成后　　#设置开机启动

打开浏览器，验证：

http://192. 168. 200. 201

1. 3 虚拟 web 主机项目实战

1 .虚拟主机概念和类型介绍

(1) 虚拟主机的概念

所谓虚拟主机，在 Web 服务里就是一个独立的网站站点(www.baidu.org),这个站点对应 独立的域名(也可能是 IP 或端口)，具有独立的程序及资源目录，可以独立地对外提供服 务供用户访问

(2) 虚拟主机类型

常见的虚拟主机类型有如下几种

- 基于域名的虚拟主机

所谓基于域名的虚拟主机，意思就是通过不同的域名区分不同的虚拟主机，基于域名的虚拟 主机是企业应用最广的虚拟主机类型，几乎所有对外提供服务的网站都是使用基于域名的虚 拟主机，例如：www. apache, org

- 基于端 I I 的虚拟主机

同理，所谓基于端口的虚拟主机，意思就是通过不同的端口来区分不同的虚拟主机，此类虚 拟主机对应的企业应用主要为公司内部的网站，例如：一些不希望直接对外提供用户访问的 网站后台等，访问基于端口的虚拟主机地址里要带有端口，例如：http://www. baidu. com:80

- 基于 IP 的虚拟主机

同理，所谓基于 IP 的虚拟主机，意思就是通过不同的 IP 区分不同的虚拟主机，此类虚拟主 机对应的企业应用非常少见，一般不同业务需要使用多 IP 的场景都会在负载均衡器上进行 VIP 绑定，而不是在 Web 上通过绑定 IP 区分不同的虚拟机。

2. 配置基于端口的虚拟主机 在 uos server1 上

创建网页目录

```
mkdir /var/www/8899 准备访 I'可页面
echo 8899 > /var/www/8899/index, html 修改配置文件
```

vim /etc/apache2/sites-enabled/vhosts. conf Listen 8899 #更改监听的端口号

<VirtualHost *:8899>

ServerName 192. 168. 200. 201

DocumentRoot /var/ww/8899

</VirtualHost> 重启 apache 服务

```
systemctl restart apache2
```

打开浏览器访问验证：http: 〃 192. 168. 200. 201:8899 页面输出 8899 则表示成功

3. 基于域名的 web 虚拟主机 首先我们创建测试页面：

```
mkdir /var/www/uosl                    #创建网页目录
mkdir /var/www/uos2                      #创建网页目录
echo "uosl" > /var/www/uosl/index. html                  #准备访问页面
echo "tios2" > /var/www/uos2/index. html              #准备访问页面 配置修改：
```

这里扁用 grep 过滤命令来生成基础的 apache 主配置文件，然后根据生成的初始配置进行修 改，使其成为所需的形式，具体步骤为：

```
cd /etc/apache2/sites-enabled/
```

```
egrep -v 000-default.conf >vhosts.conf #去除带#和空行，重定向
```

或者直接新创建配置文件，然后编辑。

```
vim vhosts.conf #修改配置文件如下 <VirtualHost *:80>
    ServerName www.uosl.com #网站名，第一个网站 www.uosl.com
    DocumentRoot /var/ww/uosl #网站目录
</VirtualHost> <VirtualHost *:80>
    ServerName www.uos2.com #第二个网站 www.uos2.com
    DocumentRoot /var/ww/uos2
</VirtualHost> 重启 apache2 服务
```

```
systemctl restart apache2
```

在 uos2 或宿主机 I：面，访问验证：

说明：

通过 IP 地址来访问的话，apache 并不知道你想要访问哪个站点，因此，他默认你是要 访问他配置文件里的第一个站点，也就是 www.uosl.com

通过修改 hosts 映射我们可以访问不同的站点。

修改 hosts 映射文件

```
vim /etc/hosts #添加主机名映射
```

```
192.168.200.201 www.uosl.com www.uos2.com
```

使用 curl 工具验证（如没有可 apt install curl）

```
curl http://www.uosl.com
```

输出：uosl

```
curl http://www.uos2.com
```

输出：uos2

4.基于 IP 的虚拟主机

添加一块网长并设置 IP 地址为 192.168.100.202/24 （假设网 R 设备名为 ens36）

```
nmcli device show
nmcli connection add type ethernet con-name ens36 ifname ens36
nmcli connection modify ens36 ipv4.method manual ipv4.addresses
192.168.200.202/24 connection.autoconnect yes
nmcli connection up ens36 修改配置文件
cd /etc/apache2/sites-enabled/
vim vhosts.conf #修改配置文件如下
<VirtualHost 192.168.200.201:80>
    ServerName www.uosl.com
    DocumentRoot /var/ww/uosl </VirtualHost>
<VirtualHost 192.168.100.202:80> ServerName www.uos2.com DocumentRoot /var/ww/uos2
</VirtualHost> 重启服务
```

```
systemctl restart apache2
```

在 uos2 或宿主机 I：面验证

```
vim /etc/hosts #修改主机名映射
```

```
192. 168. 200. 201      www. uosl. com
192. 168. 200. 202      www. uos2. com
```

访问验证

curl http://192.168.200.201 输出：uosl

curl http://192. 168. 200. 202 输出：uos2

1.4 虚拟 web 主机项目实战

1. alias 别名

web 网站别名配置是被经常使用的一个特性。这个功能实际上是为站点 URI 定义一个路径映射关系

2. 还原 uosl 到基于域名的虚拟主机环境

```
mkdir /alias                    #目录的位置可以随意指定、根下也是可以的、但是不建议 echo alias > /alias/index. html #创建访问页面
cd /etc/apache2/sites-enabled/
vim vhosts. conf                #修改配置文件
<VirtualHost *:80>
     ServerName www. uosl. com
     DocumentRoot /var/ww/uos
          Alias /net /alias #添加 alias 别名
</VirtualHost>
<Directory /alias>                    #设置允许访问目录
  AllowOverride none
  Require all granted
</Directory>
```

上述别名的配置，就是说当你基于你的站点访问 ww.uosl.com/net 目录下的文件时， 会直接从服务器 /alias 目录下访问对应的文件。

```
systemctl restart apache2           #重启 apache 月艮务
```

3. 客户端访问验证（用浏览器访问）

firefox http:〃192. 168. 200. 201/net

当页面显示 alias 的时候，就说明已经成功了

1. 5 动态网站

1.动态网站概述

最早的 Web 服务器简单地响应浏览器发来的 HTTP 请求，并将存储在服务器上的 HTML 文件返回给浏览器，也就是静态 htmlo 事物总是不断发展，网站也越来越复杂，所以出现 动态技术。但是服务器并不能

直接运行 php，asp 这样的文件，自己不能做，外包给别人吧，但是要与第三做个约定，我给你什么，然后你给我什么，就是握把请求参数发送给你，然后 我接收你的处理结果给客户端。那这个约定就是 common gateway interface,简称 cgi

2．创建不同类型的脚本

在名为 L10S1 的机器上面

```
mkdir /var/www/cgi-bin #创建脚本目录
vim /var/www/cgi-bin/shell, sh              #创建 shell 脚本
#!/bin/bash
echo -en “Content-Type: text/html: charset=UTF-8\n\n〃;
date +%c
chmod +x /var/ww/cgi-bin/shell. sh          #为脚本添加执行权限

vim /var/www/cgi-bin/perl, pl               #创建 perl 脚本
#!/usr/bin/perl
print Content-Type: text/html; charset=UTF-8\n\n'z:
$now=localtime ();
print 〃$now\n〃;
chmod +x /var/ww/cgi-bin/perl. pl

apt-get install libapache2 - mod - wsgi             # 安装 wsgi

vim /var/www/cgi-bin/python. py             #这里写一个 python 的脚本
import time                                 #调用 time 模块(注：复制时需去掉注释)
def application (environ, start_response):
    response_body = 'UNIX EPOCH time is now: %s\n' % time. time() status = ' 200 OK'
    response_headers = [(* Content-Type*, 'text/plain'),
                        ('Content-Length5, T'),
                        C Content-Length', str(len(response_body)))]
    start_response(status, response_headers)
    return [response body]
```

3.修改配置文件添加 alias

```
vim /etc/apache2/sites-enabled/vhosts. conf
<VirtualHost *:80>
      ServerName www. uosl. com
     DocumentRoot /var/ww/uosl
ScriptAlias /jiaoben/ 〃/var/www/cgi-bin/"
WSGIScriptAlias /python /var/www/cgi-bin </VirtualHost>

cp /etc/apache2/mods-available/cgi. * /etc/apache2/mods-enabled/ systemctl restart apache2
```

4. 客户端验证访问

```
curl http://www. uosl. com/jiaoben/perl.pl
Wed Jul 22 13:11:17 2020
curl http://www. uosl. com/jiaoben/shell, sh
Wed Jul 22 13:11:20 2020
curl http://www. uosl. com/python/python. py
UNIX EPOCH time is now: 1595394831. 07
```

1. 6 部署 Discuz!论坛

1. LAMP

LAMP 是指一组通常一起使用来运行动态网站或者服务器的自由软件名称首字母缩写:

Linux,操作系统

Apache,网页服务器

MariaDB 或 MySQL,数据库管理系统（或者数据库服务器）

PHP> Perl 或 Python,脚本语言

2.安装测试 LAMP 环境

```
apt install -y php-fpm            php-mysql mariadb-server mariadb-cllent php-fpm
libapache2-mod-php php-mysql          #安装环境
systemctl restart php7. 3-fpm              #启动 php
systemctl enable php7. 3-fpm               #PHP 加入开机启动
systemctl restart mariadb                  #启动 mariadb
systemctl enable mariadb                   #mariadb 加入开机启动


echo'ᶻ<?php phpinfo0 : ?> ″       >>/var/www/html/index, php                    #创建 php 测试页
```

客户端访问验证

```
curl http://192. 168. 200. 201/index. php
```

3. 部署 Discuz!

论坛下载地址(链接:**https://pan.baidu.eom/s/lS8Sv2JZRKlYrOLa4tEfoPA** 提取码:**ujdn**) 将下载好的 Discuz_X3. 4. zip 拷贝到虚拟机 uosl 中

```
sep /root/Discuz_X3. 4. zip root@192. 168. 200. 237: /var/ww/html/

apt install php-xml #discuz 函数需要 PHP 支持 XML
cd /var/ww/html
unzip Discuz_X3. 4. zip #解压
cp -r upload/* .                  #拷贝到网页目录
```

```
chown www-data: www-data -R /var/ww/htm 1 / #递归更改目录权限 vim /etc/apache2/sites-
enabled/vhosts. conf
<VirtualHost *:80>
    ServerName 192. 168. 200. 201
    DocumentRoot /var/ww/html
</VirtualHost>
```

数据库

```
mysqladmin -u root password ⁵ 123456'          #为数据库设置密码
vim /etc/mysql/mariadb. conf, d/50-server. enf #bind-address = 127. 0. 0. 1     将此行注释
systemctl restart mariadb.service
```

```
systemctl restart apache2
```

客户端访问配置

```
firefox http://192. 168. 200. 201/index. php
```

请勿在页面内输入任何敏感信息（密码或银行卡信息等），否则可能会被攻击者盗用

Discuz：安装向导

DiSCUZ!X3.4 统体中 Z UTF8 — 20191201

z 安装数据库

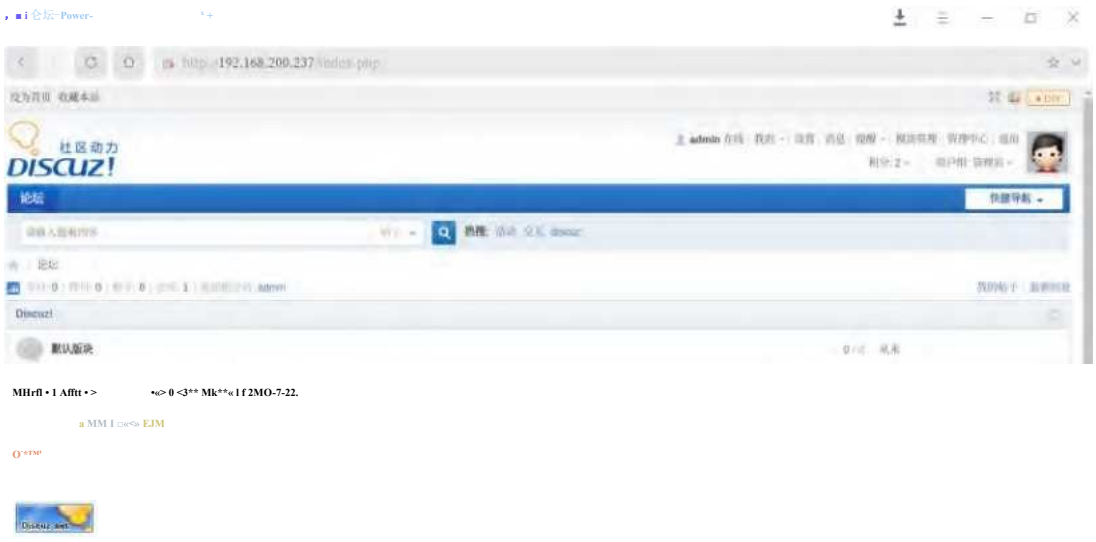
0 怜台安技环垣 . _c 吸胃炫行环境 -»VtlKS

信息

	ShW	收耕侔販务钳地 M.一般为 k»moSi
MI 時：	H	
hWffffFI₁	|123456	
Mtntita		同一 8tJK，iS 行多个论后狀，谊修枚
签就侣 m		
Emad:	adm0admin.com	用干力«S5 倍快 1 告
	|aorrwi	
冒用晚密，：	H-	
«H«a	tj	Z3
fiffla Email:	adnwiffaomm com	

Ccpyngtn C2001-2020. Ifencem Cloud.

登陆成功



MHrfl • 1 Affit • > •«> 0 <3** Mk**« 1 f 2MO-7-22.

a MM I □«<» EJM

O°*™

接下来就可以设置其他的 r 哦!

第 2 章 NGINX 服务

2.1 简介

1、 nginx 介绍

Nginx （engine x）是一个高性能的 HTTP 和反向代理 web 服务器，同时也提供了 IMAP/POP3/SMTP 服务反向代理。Nginx 是由伊戈尔-赛索耶夫为俄罗斯访问量第二的 Rambler.ru 站点（俄文：Рамблер）开发的，第一个公开版本 0. 1. 0 发布于 2004 年 10 月 4 日。其将源代码以类 BSD 许可证的形式发布，因它的稳定性、丰富的功能集、示例 配置文件和低系统资源的消耗而闻名。

其特点是占有内存少，并发能力强，事实上 nginx 的并发能力在同类型的网页服务器中 表现较好，中国大陆使用 nginx 网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

2、 Nginx 的重要特性

- 支持高并发：能支持几万并发连接（特别是静态小文件业务环境）
- 资源消耗少：在 3 万并发连接下，开启 10 个 Nginx 线程消耗的内存不到 200MB

- 可以做 HTTP 反向代理及加速缓存，即负载均衡功能，内置对 RS 节点服务器健康检查功 能，这相当于专业的 Haproxy 软件或 LVS 的功能
- 具备 Squid 等专业缓存软件等的缓存功能。
- 支持异步网络 I/O 事件模型 epoll (linux2. 6+)。

3、 Nginx 软件的主要企业功能应用

1) 作为 Web 服务软件
2) 反向代理或负载均衡服务
3) 前端业务数据缓存服务

4、 Nginx 和 apache 对比

- Nginx 同 Apache 一样都是一种 Web 服务器。基于 REST 架构风格，以统一资源描述 符(Uniform Resources Identifier) URI 或者统一资源定位符(Uniform Resources Locator) URL 作为沟通依据，通过 HTTP 协议提供各种网络服务。
- 然而，这些服务器在设计之初受到当时环境的局限，例如当时的用户规模，网络带宽， 产品特点等局限并且各自的定位和发展都不尽相同。这也使得各个 Web 服务器有着各 自鲜明的特点。
- Apache 的发展时期很长，而 FL 是毫无争议的世界第一大服务器。它有着很多优点：稳 定、开源、跨平台等等。

•它出现的时间太长了，它兴起的年代，互联网产业远远比不上现在。所以它被设计为一 个重量级的。

- 它不支持高并发的服务器。在 Apache 上运行数以万计的并发访问，会导致服务器消耗 大量内存。
- 操作系统对其进行进程或线程间的切换也消耗了大量的 CPU 资源，导致 HTTP 请求的 平均响应速度降低。
- 这些都决定了 Apache 不可能成为高性能 Web 服务器，轻量级高并发服务器 Nginx 就 应运而生了。

2. 2 nginx 安装部署

安装

```
apt install nginx
nginx -v                    #查看 nginx 版本
systemctl start nginx
systemctl enable nginx
```

注：启动后，在浏览器输入 IP 地址，即可看到 nginx 的欢迎页面。至此 nginx 安装成功 **nginx** 各文件存放路径

- /usr/sbin/nginx　　#主程序
- /etc/nginx　　#存放配置文件
- /usr/share/nginx　　#存放静态文件

- /var/log/nginx #存放日志

源码编译安装

```
wget http://nginx.org/download/nginx-1.14.2.tar.gz#下载 nginx 源码包 tar -xf nginx-1.14.2.tar.gz
cd nginx-1.14.2
./configure 一help
./configure --prefix=/usr/local/nginx --without-http_rewrite_module --without -http_gzip_module
#指定安装路径#避免 Nginx 依赖于其他库（常用指定用户组开启加密）
make && make install #编译并安装
/usr/local/nginx/sbin/nginx 启动
netstat -anptu |grep nginx
```

tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN
13597/nginx: master

2.3 nginx 内核优化

内核参数调整：linux 内核参数调整有两种方式

- 方法一：修改/proc T 内核参数文件内容，不能使用编辑器来修改内核参数文件，理由 是由于内核随时可能更改这些文件中的任意一个，另外，这些内核参数文件都是虚拟文 件，实际中不存在，因此不能使用编辑器进行编辑，而是使用 echo 命令，然后从命令 行将输出重定向至/proc 下所选定的文件中。
  如：将 timeout_timewait 参数设置为 30 秒：

```
echo 30 > /proc/sys/net/ipv4/tcp_fin_timeout
```

注：参数修改后立即生效，但是重启系统后，该参数又恢复成默认值。因此，想永久更改内 核参数，需要修改/etc/sysctl.conf 文件

- 方法二.修改/etc/sysctl.conf 文件。检查 sysctl.conf 文件，如果已经包含需要修 改的参数，则修改该参数的值，如果没有需要修改的参数，在 **sysctl.conf** 文件中添加 参数。如：

```
net.ipv4.tcp_f in_timeout=30
```

注：保存退出后，可以重启机器使参数生效，如果想使参数马上生效，也可以执行如下命令：sysctl -p

更改系统内核参数，使 **nginx** 支持更多并发请求

- net.ipv4.ip_nonlocal_bind = 1 #允许非本地 IP 地址 socket 监听
- net.ipv4.ip_forward = 1 #开启 IPv4 转发
- net.ipv4.tcp_timestamps = 0 #是否开启数据包时间戳
- net.ipv4.tcp_tw_reuse = 0 #端口复用
- net.ipv4.tcp_tw_recycle = 0 #快速回收 TIME_WAIT 状态，用于大量 TIME_OUT 场景
- fs.file-max = 1000000 #表示单个进程较大可以打开的句柄数

- net. ipv4. tcp_tw_reuse = 1 #参数设置为 1 ,表示允许将 TIME_WAIT 状态的 socket 重新用于新的 TCP 链接，这对于服务器来说意义重大，因为总有大量 TIME_WAIT 状态的链接存在
- net. ipv4. tcp_keepalive_time = 600 #当 keepalive 启动时，TCP 发送 keepalive 消息的频度；默认是 2 小时，将其设置为 10 分钟，可更快的清理无效链接
- net. ipv4. tcp_fin_timeout = 30 #当服务器主动关闭链接时，socket 保持在 FIN_WAIT_2 状态的较大时间
- net. ipv4. tcp_max_tw_buckets 二 5000 #表示操作系统允许 TIME_WAIT 套接字数量的较大值，如超过此值，TIME_WAIT 套接字将立刻被清除并打印警告信息，默认为 8000,过 多的 TIME_WAIT 套接字会使 Web 服务器变慢

net. ipv4. ip_local_port_range = 1024 65000 口的取值范围#定义 UDP 和 TCP 链接的本地端

net. ipv4. tcp_rmem 二 10240 87380 12582912 #定义了 TCP 接受 socket 请求缓存的内存最小值、默认值、较大值

net. ipv4. tcp_wmem 二 10240 87380 12582912 #定义 TCP 发送缓存的最小值、默认值、较大值

net. core. netdev_max_backlog = 8096 #当网卡接收数据包的速度大于内核处理速度时，会有一个列队保存这些数据包。这个参数表示该列队的较大值

- net. core. rmem_default = 6291456 R 表示内核套接字接受缓存区默认大小 #表
- net. core. wmem_default = 6291456 示内核套接字发送缓存区默认大小 #表示内核
- net. core. rmem_max = 12582912 套接字接受缓存区较大大小
- net. core. wmem_max = 12582912 #表示内核套接字发送缓存区较大大小
- 以上四个参数，需要根据业务逻辑和实际的硬件成本来综合考虑
- net. ipv4. tcp_syncookies = 1 #与性能无关。用于解决 TCP 的 SYN 攻击
- net. ipv4. tcp_max_syn_backlog = 8192 #这个参数表示 TCP 三次握手建立阶段接受 SYN 请求列队的较大长度，默认 1024,将其设置的大一些可使出现 Nginx 繁忙来不及 accept 新连接时，Linux 不至于丢失客户端发起链接请求
- net. ipv4. tcp_tw_recycle = 1 #这个参数用于设置启用 timewait 快速回收
- net. core. somaxconn=262114 #选项默认值是 128,这个参数用于调节系统同时发起 的 TCP 连接数，在高并发的请求中，默认的值可能会导致链接超时或者重传，因此需要 结合高并发请求数来调节此值
- net. ipv4. tcp_max_orphans=262114 #选项用于设定系统中最多有多少个 TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤立链接将立即被复位并输出警告信息。这个限制指示为了防止简单的 DOS 攻击，不用过分依靠这个限制甚至认为 的减小这个值，更多的情况是增加这个值。

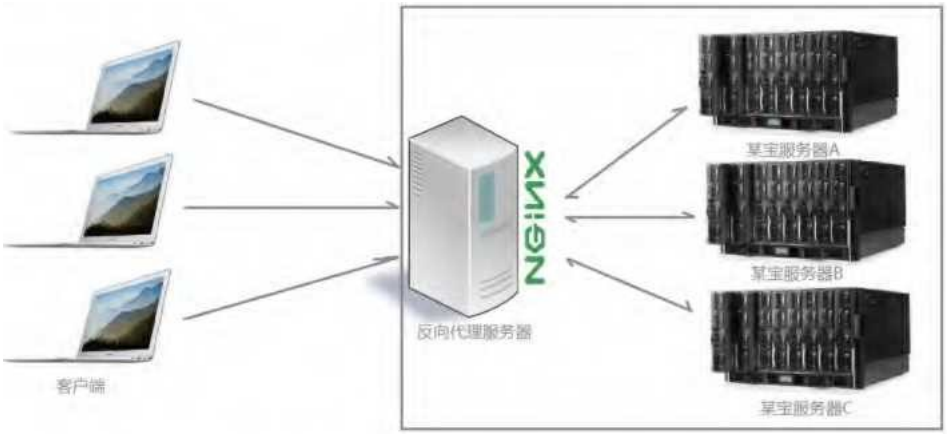
2. 4 nginx 反向代理与负载均衡应用

(1) nginx 反向代理

说到代理，首先我们要明确一个概念，所谓代理就是一个代表、一个渠道；

此时就设计到两个角色，一个是被代理角色，一个是目标角色，被代理角色通过这个代 理访问目标角色完成一些任务的过程称为代理操作过程；如同生活中的专卖店~客人到 adidas 专卖店买了一双鞋，这个专卖店就是代理，被代理角色就是 adidasT 家，目标角色

就是用户



在 UOS1 (192.168.200.201)中：

```
#   apt install nginx #安装 nginx 服务
```

进入/etc/nginx,编辑主配置文件 nginx. conf:

#vim /etc/nginx/nginx. conf

这一行屏蔽：

include /etc/nginx/sites-enabled/*; #默认 nginx 读取此文件配置 在 http{}T 配置

```
    server(
        listen 80;
        location / (
            proxy_pass http://192. 168. 200. 202:80/;
            #发向 80 端口的请求将被转发至 uos2 这个地址
            pr ox y_ s e t_he ad er Host $host;
            #Host 头域指定请求资源的 Intenet 主机和端 I 丨号。这里可选，这样配置后 用户浏览器中不会显示端口
            pr oxy_se t_he ad er X-Forwarded-For $proxy_add_x_forwarded_for;
            #后端 web 服务器可根据头域 X-Forwarded-For 获取请求用户的真实 IP }
        } 重启并设置开机启动
#   systemctl restart nginx #重启 nginx 服务
#   systemctl enable nginx
```

在 uos2 (192. 168. 200.202)中：

```
apt install nginx #安装 nginx 服务 echo 〃nginx2〃 > /var/www/html/index, html systemctl start nginx 在 uosl (192. 168.200.201)上测试:
```

http://192. 168. 200. 202   #自动跳转到 uos2 的 nginx 服务

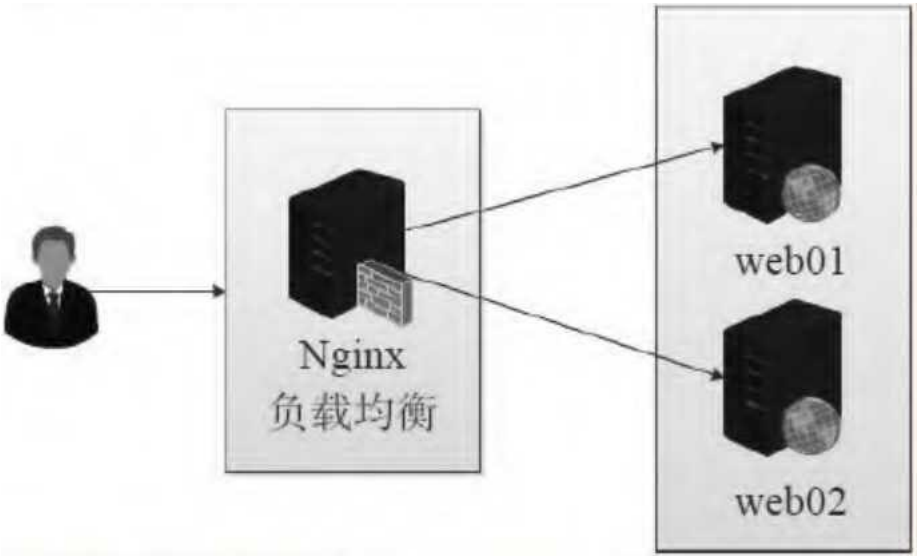
(2) nginx 负载均衡

I、可题：我们已经明确了所谓代理服务器的概念，那么接下来，nginx 扮演了反向代理服务器 的角色，

它是以依据什么样的规则进行请求分发的呢？不用的项目应用场景，分发的规则是 否可以控制呢？

负载均衡：

这里提到的客户端发送的、nginx 反向代理服务器接收到的请求数量，就是我们说的负载量 请求数量按照一定的规则进行分发到不同的服务器处理的规则，就是一种均衡规则 所以'将服务器接收到的请求按照规则分发的过程，称为负载均衡。



Nginx 的 **upstream** 分配模式

• weight 轮询(默认)

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器 down 掉，能自动剔除 upstream uosserver (server 192. 168. 200. 201; server 192. 168. 200. 202;

) • weight

指定轮询几率，weight 和访问比率成正比，用于后端服务器性能不均的情况。

```
upstream uosserver (
server 192. 168. 200. 201 weight=3;
server 192. 168. 200. 202 weight=7;
```

• ip_hash 每个请求按照访问 ip 的 hash 结果分配，这样每个访客固定访问一个后端服务器，可以解决 session 的问题。

```
upstream uosserver ( ip_hash;
server 192. 168. 200. 201:80;
server 192. 168. 200. 202:80;
}
```

fair(第三方)

按后端服务器的响应时间来分配请求，响应时间短的优先分配。

在 UOS1 (192. 168. 200.201)中：

```
#   apt install nginx          #安装 nginx 服务
```

进入/etc/nginx,编辑主配置文件 nginx. conf:

vim /etc/nginx/nginx. conf

这一行屏蔽：

ttinclude /etc/nginx/sites-enabled/*; #默认 nginx 读取此文件配置 在 http{}下配置

```
upstream uosserver {
        server 192. 168. 200. 202;
        server 192. 168. 200. 203;
    }

    server {
        listen 80;
        server_name 192.168.200. 201;
        location / {
            root /var/ww/html;
            index index, html:
            proxy_pass http://uosserver:
```

#重启 nginx 服务

```
systemctl restart nginx systemctl enable nginx
```

在 uos2 (192.168.200.202)、uos3 (192.168.200.203)中： apt-get install nginx #安装 nginx 服务 echo 〃tios2〃 > /var/ww/htnl/index. html #uos2 中执行 echo 〃tios3〃 > /var/ww/htnl/index. html #uos3 中执行 systemctl restart nginx #重启 nginx 服务

systemctl enable nginx

打开浏览器访问 http://192. 168. 200. 201,验证负载均衡的轮询效果

2. 5 配置 HTTPS

安装 openssl

```
# apt install openssl          #确保安装 openssl 命令
```

生成 server 密钥 key

```
•   openssl genrsa -des3 -out server, key 2048          #生成秘钥 key
```

注：

- 会有两次要求输入密码，输入同一个即可
- 输入密码，获得了一个 server, key 文件.

- 以后使用此文件（通过 openssl 提供的命令或 API）可能经常回要求输入密码，如果想去 除输入密码的步骤可以使用以下命令：
- openssl rsa -in server, key -out server, key 创建服务器证书的中靖文件 **server, csr,** 运行：

openssl req -new -key server, key -out server, csr

注：

其中 Country Name 填 CN, Common Name 填主机名也可以不填，如果不填浏览器会认为不安 全.（例如你以后的 url 为 https://abcd/xxxx....这里就可以填 abed）,其他的都可以不填.

创建 **CA** 证书：

openssl req -new -x509 -key server, key -out ca. ert -days 3650 此时，你可以得到一个 **ca. ert** 的证书，这个证书用来给自己的证书签名.

创建自当前日期起有效期为期十年的服务器证书 **server, ert：**

```
# openssl x509 -req -days 3650 -in server, csr -CA ca. ert -CAkey server, key -
CAcreateserial -out server, ert
```

Is 当前的文件夹，可以看到一共生成了 5 个文件：

ca. ert ca.srl server, ert server.csr server, key 注：其中,server, ert 和 server, key 就是你的 nginx 需要的证书文件.

- 单向认证只是客户端认证服务端，
- 双向认证就是相互都要认证（双向认证这种,在浏览器一般都体现为需要证书）

配置 **nginx：**

进入/etc/nginx,编辑主配置文件 nginx. conf:

- vim /etc/nginx/nginx. conf

server （

```
    listen 443;
        server_name localhost;
        ssl                     on;
        ssl_certificate /root/server. crt; #配置证书位置 ssl_certificate_key /root/server,
        key; #配置秘钥位置 #ssl_client_certificate ca. crt; #双向认证
        #ssl_verify_c 1 ient on; #双向认证

        ssl_session_timeout 5m;
        ssl_protocols SSLv2 SSLv3 TLSv1;
        ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
        ssl_prefer_server_ciphers on;
```

#重启 **nginx**

```
systemctl restart nginx 验证
```

https ://192.168. 200. 201 #nginx 的 https 功能可以使用了

2. 6 nginx 虚拟主机

(1)基于域名：

进入/etc/nginx,编辑主配置文件 nginx. conf:

vim /etc/nginx/nginx. conf

这一行屏蔽：

#include /etc/nginx/sites-enabled/*; #默认 nginx 读取此文件配置 在 http{}下配置

```
server (
        listen 80;
        server_name www. uosl. com; #设置域名为 www. uosl. com

        location / (
            index index.html:
            root /var/ww/html/uosl/:



server (
        listen 80;
        server_name www. uos2. com; #设置域名为 www. uos2. com

        location / (

            index index, html:
            root /var/ww/html/uos2/;
```

说明：

- listen 为监听的端口，本例中监听 80 端口
- server_name 即指定的虚拟主机名
- location /表示匹配这个主机名下的所有请求
- index 指主页的文件名
- root 为网站根目录在系统中的实际位置

- vim /etc/hosts #更改 hosts 文件

192. 168. 200. 201 www. uosl. com

192. 168. 200. 201 www. uos2. com 配置 root 目录和 index 文件

```
mkdir /var/ww/html/uosl /var/www/html/uos2 #创建 web 页面目录 cd /var/www/html/uos 1 &&
echo 〃tiosl〃 > index, html
cd /var/www/html/uos2 && echo 〃tios2〃 > index, html #重启 nginx 服务
```

```
systemctl restart nginx
```

访问 www. uosl. com 验证

访问 www. uos2. com 验证

(2)基于端口：

进入/etc/nginx,编辑主配置文件 nginx. conf:

vi /etc/nginx/nginx. conf

这一行屏蔽：

#include /etc/nginx/sites-enabled/*; #默认 nginx 读取此文件配置 在 http{}下配置

```
server (
        listen 80;
        server_name www. uosl. com; #设置域名为 www. uosl. com

        location / (
             index index.html:
             root /var/ww/html/uosl/:



server (
        listen 80;
        server name www. uos2. com; #设置域名为 www. uos2. com
        location / (
             index index.html:
             root /var/ww/html/uos2/;



server (
        listen 8080;
        server_name www. uos2. com; #设置域名为 www. uos2. com location / (
             index index.html:
             root /var/ww/html/uos3/;
```

vi /etc/hosts #更改 hosts 文件

www. uosl. com 192.168. 200. 201

www. uos2. com 192. 168. 200. 201 配置 **root** 目录和 **index**

```
mkdir /var/www/html/uosl /var/www/html/uos2 /var/www/html/uos3 #创建 web 页面目 录
cd /var/ww/html/uosl && echo 〃uosl〃 > index, html
cd /var/ww/html/uos2 && echo 〃tios2〃 > index, html
cd /var/ww/html/uos3 && echo 〃uos3〃 > index, html #重启 nginx 服务
```

systemctl restart nginx

访问 www. uos2. com:8080 验证

(3)基于 IP：

在 uosl 匕增加一个 IP 地址

```
nmcli connection modify ens33 ipv4. method manual +ipv4.addresses
"192. 168. 200. 101/24
nmcli connection down ens33
nmcli connection up ens33 进入/etc/nginx,编辑主配置文件 nginx. conf:
```

```
# vim /etc/nginx/nginx. conf
这一行屏蔽：
#include /etc/nginx/sites-enabled/*; #默认 nginx 读取此文件配置 在 http{}下配置
server (
        listen 80;

        server_name 192. 168. 200. 201; #设置 IP 为 192.168. 200. 201 location / (
            index index.html:
            root /var/ww/html/uosl/;



server (
        listen 80;

        server_name 192. 168. 200. 202; #设置 IP 为 192. 168. 100. 101 location / (
            index index.html:
            root /var/ww/html/uos2/;
```

配置 root 目录和 index

```
mkdir /var/www/html/uos 1 /var/ww/html/uos2 #创建 web 页面目录 cd /var/ww/html/uosl &&
echo 〃uosl〃 > index, html
cd /var/ww/html/uos2 && echo 〃tios2〃 > index, html #重启 nginx 服务
systemctl restart nginx 访|»J 192. 168. 200. 201 和 192. 168. 200.101 验证
```

2. 7 rewrite 功能模块

重写 URL 是非常有用的一个功能，提高网站访问能力，而口在你改变了自己的网站结构后，无需客户端进行任何修改，并且提高网站安全性。

格式：

```
rewrite <regex> <replacement> [flag]:
关键字 正则              替代内容          flag 标记
```

- 关键字：其中关键字 error_log 不能改变
- 正则：正则表达式语句进行规则匹配
- 替代内容：将正则匹配的内容替换成 replacement

- flag 标记：rewrite 支持的 flag 标记

flag 标记说明：

- last #本条规则匹配完成后，继续向下匹配新的 1 ocation URI 规则
- break #本条规则匹配完成即终止，不再匹配后面的任何规则
- redirect #返回 302 临时重定向，浏览器地址会显示跳转后的 URL 地址
- permanent #返回 301 永久重定向，浏览器地址栏会显示跳转后的 URL 地址

(1) 内置变量

```
$request                    # HTTP 请求行，如"GET / HTTP/1. 1"
$request_method             # HTTP 请求方法,如"GET"、"POST"
$request_uri                # HTTP 请求 URI,带查询参数
$request_filename           # HTTP 请求文件，如〃/path/to/index. html"
$content_type               # HTTP 报头”Content-Type”字段
$content_length             # HTTP 报头"Content-Length"字段
$http_cookie                # HTTP 报头”Cookie”字段
$ http_user_agent           # HTTP 报头”User-Agent"字段
$server_protocol            # 服务器 HTTP 版本，如"HTTP/1. 0”、“HTTP/1. 1”
$server_name                #服务器 Host 名
$server_addr                #服务器 IP 地址
$server_port                #服务器 Port 号
$remote_addr                #客户端 IP 地址
$remote_port                #客户端 Port 号
$remote_user                #客户端 User 名（认证）
$scheme $host               #请求的协议类型,如"http”、"https
$uri $args                  #请求的虚拟主机
$query_string               #请求的 URI,不带参数
$document_uri               #请求的参数
$document_root              #请求的参数，同$args
$status                     #请求的 URI,不带参数，同$uri
                            #请求的根目录
                            # HTTP 响应码，如 200
```

(2) 匹配规则

```
rewrite 写在 server、location、if 字段中 #正
    Z
                则匹配
•   *           #正则匹配（忽略大小写）
•   广          #正则取反
• !~*           #正则取反（忽略大小写）
•   -f 和! -f   #判断是否为文件
•   -d 和! -d   #判断是否为目录
. -e 和! -e     #判断文件是否存在
•   -x 和! -x   #判断文件是否可执行
•   set         #设置变量
```

- last　　#完成 rewrite
- break　　#终止匹配
- return　　#返回状态码
- redirect　　#返回 302 临时重定向
- permanent　　#返回 301 永久重定向

(3) 重写实例

实例：访问 uos. com 网站自动跳转到 www. uos. com

在 uos1 (192. 168. 200.101)中：

```
#  apt install nginx        #安装 nginx 服务
```

进入/etc/nginx,编辑主配置文件 nginx. conf:

```
#  vim /etc/nginx/nginx. conf server (
     listen 80;
     server_name uos. com;
     rewrite "/(.*) http://www. uos. com/$1 permanent: )
server (
     listen 80;
     server_name www. uos. com;
     location / (
          root /var/ww/html/uos:
          index index, html;
```

目录与 index

```
#  mkdir /var/www/html/uos
#  echo "uos" > /var/ww/html/uos/index, html 配置域名解析
```

```
# vim /etc/hosts #编辑 hosts 文件
192. 168. 200. 201 www. uos. com 重启 nginx
```

```
# systemctl restart nginx 访问 www. uos. com 验证
```

实例：将 url 靖求的文件重定向到 uos/目录中去寻找

在 uosl (192. 168. 200.201)中：

```
#  apt install nginx      #安装 nginx 服务
```

进入/etc/nginx,编辑主配置文件 nginx. conf:

```
#  vim /etc/nginx/nginx. conf server(
     listen 80;
     server_name www. uos. com;
     root /var/ww/html;
     index index, html:
     if ($request_uri 广’ ″/uos/’)
     (
          rewrite /(. *) /uos/$l redirect;
```

配置 root 目录与 index

```
#  mkdir /var/www/html/uos
#  echo 〃tios〃 > /var/ww/html/index, html
#  echo 〃uosl〃 > /var/www/html/uos/index. html # vim /etc/hosts #编辑 hosts 文件
192. 168. 200. 201 www. uos. com
```

重启 nginx

```
# systemctl restart nginx
```

访问 www. uos. com 自动跳转到 www. uos. com/uos

实例：配置防盗链避免别的网站引用 www.uos. com 不想被引用的图片等文件

盗链：ref er er 是记录打开一个页面之前记录是从哪个页面跳转过来的，如果别人只链接了 自己网站图片或某个单独的资源，而不是网站的页面，这就是盗链，ref er er 就是之前的那 个网站域名

在 UOS1 (192. 168. 200.201)中：

```
#  apt install nginx      #安装 nginx 服务
```

进入/etc/nginx,编辑主配置文件 nginx. conf:

```
#  vim /etc/nginx/nginx. conf
server(
     listen 80;
     server_name www. uos. com;
```

```
    root /var/ww/html;
    location. +. (jpg I jpeg|gif css png js|rar|zip flv) $ (
        valid_referers none blocked server_names uos. com; if ($invalid_referer)
        (
            return 403;
```

说明：

- valid_referers：白名单
- invalid_referer：无效的（未在白名单中定义的）
- none：允许 ref er er 为空（也就是允许直接访问，未从其他站点跳转的请求）
- blocked：允许来源地址不含 http/https

```
#  systemctl restart nginx
#  mkdir /var/www/html/uos
#  vim /etc/hosts #编辑 hosts 文件
192. 168. 200. 201 www. uos. com
```

下载一张图片命名为 abc. jpg 并存放在/var/www/html/下

在 server2 (192. 168.200.202)中：

```
#  apt install nginx #安装 nginx 服务
#  systemctl restart nginx
#  systemctl enable nginx
n mv /var/ww/html/index, nginx-debian. html /var/ww/html/index, html
#  vim /var/www/html/index. html      #修改网页文件，加入 serverl 中的图片
hello, world<img src='"http://www. uos.com/abc. jpg'" width=〃14〃 height=〃18〃 alt=W 件
"title=〃 附件”〉
#  vim /etc/hosts #编辑 hosts 文件
192. 168. 200. 201 www. uos. com
```

访|M WWW. uos. com 图片无法查看，防盗链生效。

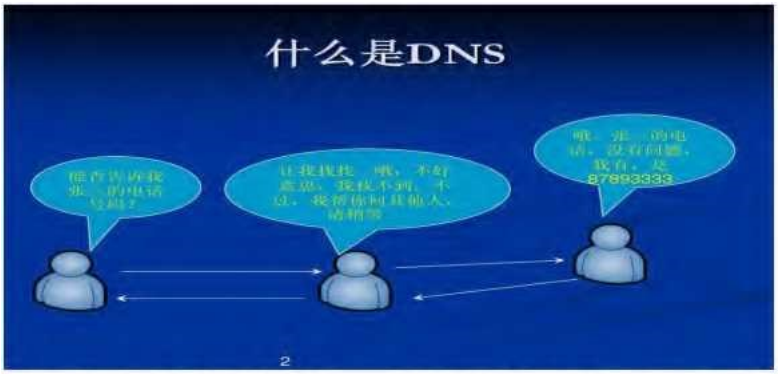
第 3 章 DNS 域名服务

3．1 DNS 服务概述

DNS 服务的介绍

DNS 服务器概述

整个 Internet 大家庭中连接「数以亿计的服务器、个人主机，其中大部的网站、邮件服务 等服务器都使用了域名形式的地址,如 ww. google, com、mail. 163. com 等。很显然这种地 址形式要比使用 64.233. 189. 147、202. 108. 33. 74 的 IP 地址形式更加直观,更加容易被用 户记住。



FQDN 格式（完整域名格式）：在常见域名后添加“.”（根域）。例如：www.baidu.com.

DNS 系统在网络中的作川就是维护着一个地址数据库，其中记录 r 各种主机域名与 IP 地址 的对应关系，以便为客户程序提供正向或反向的地址查询服务，即正向解析与反向解析。

*正向解析：根据域名查 IP 地址，是 DNS 服务最常用的基本功能

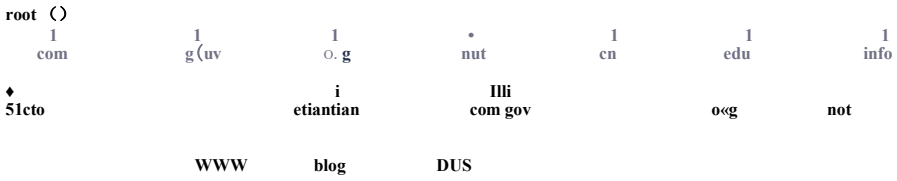
*反向解析：根据 IP 含域名，不是很常用，应用于例如反垃圾邮件的验证等

DNS 域名结构和工作原理

DNS 的结构与工作原理

每台 DNS 服务器都负责管理一个有限范围（一个或几个域）内的主机域名和 IP 地址的对 应关系，这些特定的 DNS 域或 IP 地址段称为“zone”（区域）。

DNS 系统的架构类似于一颗倒挂着的树（和 Linux 系统同录结构类似），它的顶点也是根，只不过这个根是用点（.）来表示的，不是目录的根斜线（/） 0



DNS 服务器的常见分类

根据地址解析的方向不同,DNS 区域相应的分为正向区域（包含域名到 IP 地址的解析记录） 和反向区域（包含 IP 地址到域名的解析记录）o

根据管理的区域地址数据的来源不同，DNS 系统可以分为不同的类型：

（1） 主域名服务器

维护某一个特定 DNS 区域的地址数据席，对其中的解析记录具有自主控制权，是指定区域 中唯一存在的权威服务器、官方服务器。构建主域名服务器时，需要自行建立所有负责区域 的地址数据文件。

（2） 从域名服务器

与主域名服务器提供完全相同的 DNS 解析服务，通常用于 DNS 服务器的热备份。对客户机 来说，无论使用主域名服务器还是从域名服务器，查询结果都是一样的。

以上所述主、从域名服务器的角色，只是针对某一特定的 DNS 区域来说的。例如，同一台 DNS 服务器，可以是“.chinaunix. net”区域的主域名服务器，同时也可以是“.cublog.cn”区 域的从域名服务器。

（3） 缓存域名服务器

只提供域名解析结果的缓存功能，同的在于提高数据查询速度和效率，但是没有自己控制的 区域地址数据。构建缓存域名服务器时，必须设置根域或指定其他 DNS 服务器作为解析来 源。

DNS 服务的查询方式和解析流程

DNS 服务器的查询模式

（1） DNS 服务器递归查询

当客户机向 DNS 服务器发出解析请求，DNS 服务器从服务器木地的高速缓存中查询出 结果，反馈给客户机，此过程称为递归查询，即 Client-Servero

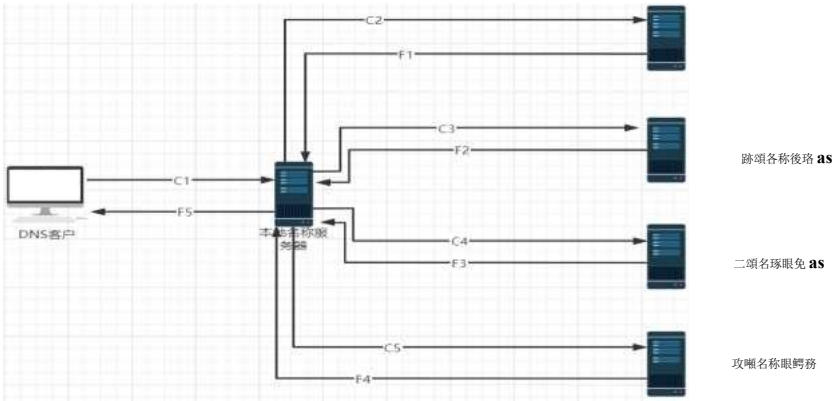
(2)DNS 服务器迭代查询

当客户机向 DNS 服务器发出解析请求，DNS 服务器从服务器木地的唯高速缓存中查询 发现没有结果，此时该 DNS 服务器以 DNS 客户的身份向其他 DNS 服务器发送解析请求或向 根域名服务器发送请求，此过程称为迭代查询，即 Server-Servero

DNS 解析流程

例：客户端解析 www. uos. com

- 1.客户端查询自己的缓存（包含 hosts 中的记录），如果没有将查询请求发送给 ,/etc/resolv. conf 中列出的 DNS 服务器；
- 2.如果 DNS 服务器对于请求的信息具有权威性，会将（权威答案）发送到客户端。
- 3.否则（不具有权威性），如果 DNS 服务器在其缓存中有请求信息，则将（非权威答 案）发送到客户端
- 4.如果缓存中没有该查询信息，DNS 服务器将搜索权威 DNS 服务器以查找信息：
  - a.从根区域开始，按照 DNS 层次结构向下搜索，直至对于信息具有权威的名称服 务器，为客户端获取答案，DNS 服务器将信息传递给客户端，并在自己的缓存中保 留一个副本，以备以后查找。
  - b.转发到其它 DNS 服务器



部署 DNS 服务器软件 bind9

安装 bind9

在 uos serverl 上

```
apt install bind9 #安装 DNS 软件 bind9
```

BIND 安装以后，会自动增加一个名为 named 的系统服务

DNS 服务的端口

- TCP 53 端口
- UDP 53 端口

DNS 服务关联的配置文件参数详解

配置文件列表：

```
Is -1 /etc/bind
```

. bind, keys
. db. 0
. db. 255
. db. empty
. db. 127 本地反向区域数据库，用于将 ip 解析为对应的域名
. db. local 本地正向区域数据库，用于将域名解析为对应的 IP 地址
. named.conf 主配置文件，通过 include 关键字加载其他三个配置文件
. named, conf, default-zones 默认区域
. 用于定义解析域，也可以直接在 named. conf 中直接划定解
named, conf, local 析域
. named, conf, options 配置文件，全局选项配置
. rndc. key
. zones. rfc1918

其中，/etc/bind9/named. conf 是 Bind 的主配置文件，不过他并不包含 DNS 数据。查看 /etc/bind9/named. conf 文件可以发现，主配置文件里面使用了 include 关键字来加载其它 3 个配置文件。

Named. conf. options 文件解析：

```
options {
    directory "/var/cache/bind" ;      # zone 文件的默认路径
    // forwarders { #这里设置的是主 DNS 的 IP 地址
    // 0.0. 0.0;          #当本地缓存中没有对应的解析时，将客户机的查询转发到哪些
DNS 服务器，可以添加多个 DNS 服务器的地址。本地测试不设置
    // };
    dnssec-validation auto; # 开启 DNS SEC 功能，DNS 安全扩展 listen-on-v6 { any; }; #定义 bind 的监听 IP 地址(IPv6)
};
```

3. 2 Bind9 配置与应用

添加 DNS 区域配置

编辑 named. conf. local 添加

```
#   cp /etc/bind/named. conf, local /etc/bind/named. conf, local, bak #备份
#   vim /etc/bind/named. conf, local
```

```
zone "uos. com" (            #定义 DNS 的 zone
     type master;            #定义此区为主服务器
     file "/etc/bind/db. uos. com";      #指定具体存放 DNS 记录的文件
);
zone "168. 192. in-addr. arpa" (         #定义一个 IP 为 192. 168. 200. *反向域区
      type master;
      file "/etc/bind/db. 192. 168. 200〃;
}；  type 类型有三种
```

- master:表示定义的是主域名服务器
- slave :表示定义的是辅助域名服务器
- hint:表示是互联网中根域名服务器

dns 正向解析区域文件的配置

- cp /etc/bind/db. local /etc/bind/db. uos. com
- vim /etc/bind/db. uos. com

```
;BIND data file for local loopback interface
9
$TTL 604800                              #正确解析数据的缓存生存周期
@ IN SOA                         uos. com. root. uos. com.(
域名          认证授权（主、从服务器            域名        邮箱地址
                    2       ; Serial            #序号
               604800       ; Refresh           #更新间隔时间
                86400       ; Retry       #更新失败再次尝试的间隔时间
              2419200       ; Expire            #若一直失败，尝试多久后放弃
                                604800 )      ; Negative Cache TTL        #查询失败的 dns 缓存
9
@   IN NS uos.
@   IN A      127.0.0. 1
@ IN AAAA : : 1
www IN A 192. 168. 200. 201          #主机记录
```

```
Lail       IN~~CNAME www
```

dns 反向解析区域文件的配置

```
#   cp /etc/bind/db.127 /etc/bind/db. 192. 168. 200
#   vim /etc/bind/db. 192. 168. 200 ;BIND reverse data file for local loopback interface
```

```
$TTL    604800
@ IN    SOA   uos. root. uos.  com.(
                              1 ;Serial
                 604800        ; Refresh
                  86400        ; Retry
                2419200        ; Expire
                  604800 );      Negative Cache TTL

*       NS    localhost.
@ IN
1.0.0 IN      PTR localhost.
201     IN    PTR www. uos. com.
```

注意

- 声明域的时候已经有了，192. 168. 200 所以我们只需要输入 201 既代表 192. 168. 200. 201
- PTR 记录域名后的点，即 FQDN 记录

记录类型

- A：定义了一条 A 记录，即主机名到 IP 地址的对应记录
- MX 定义了一邮件记录
- CNAME：定义了对应主机的一个别名
- PTR：表示反向记录
- NS: 域名服务器记录

测试配置文件

- named-checkzone db. uos. com db.127
- named-checkzone db.192.168.200 db. 127

重启服务以重载配置

- systemctl restart bind9

部署主机域名缓存机制

DNS 缓存服务器作用

为了增加访问效率，计算机有域名缓存机制，当访问过某个网站并得到其 IP 后，会将其域 名和 IP 缓存下

来，下一次访问的时候，就不需要再请求域名服务器获取 IP,直接使用缓存 中的 IP,提高了响应的速度。当然缓存是有有效时间的，当过了有效时间后，再次请求网 站，还是需要先请求域名解析。
缓存 DNS 服务器并不在本地数据库保存任何资源记录，它仅仅缓存本地局域网内客户端的查 询结果，从而起到加速查询请求和节省网络带宽的作用。

在 uos server2 上配置
确认本机的网络地址，主机映射，默认 DNS 服务器地址
```
# apt install bind9
```

配置

```
# vim /etc/bind/named. conf, options options (
          directory "/var/cache/bind"; forwarders (
                    192. 168.200.201;
           } ;
          dnssec-validation auto;  listen-on-v6 ( any; );
} ;
```

重启

```
# systemctl restart bind9
```

配置客户端网卡 DNS 项

DNS 客户端的配置文件在/etc/resolv. conf:

- Generated by NetworkManager

nameserver 114. 114. 114. 114
很明显，系统在使用了 NetworkManager 作为网络配置服务以后，这个文件已经是自动生成 的了，不要直接修改，我们应该去修改网卡的 IPv4 里的 DNS 项目。

- 通过 nmcli 命令
- 通过 nutui 字符图形界面
- 通过图形界面里的系统管理=》网络设置

详情请奏考《第二部分第三章 3.2 网络管理》

3.3 项目实战

1. 根据企业需求完成多个域名的解析服务

参考本章所有内容，完成 uos. com 域名的配置

要求：

- 配置 A 记录

ww. uos. com 192.168. 200. 201

exam. uos. com 192.168. 200. 202

mail. uos. com 192.168. 200. 203

- 配置 CNAME 记录

www. uos. cn 指向 www. uos. com

•配置客户端，能够正常解析到所配的域名

2. 根据公司要求，配置主机 DNS 缓存功能

要求：

- 配置缓存功能，使 forwards 指向 114. 114.114.114
- 配置客户端，验证即可解析到 www. uos. com,也能正常访问 www. baidu. com

第 4 章 DHCP 服务

4. 1 DHCP 服务概述

DHCP 服务的介绍

DHCP （动态主机配置协议）是一个局域网的网络协议。指的是由服务器控制一段 IP 地址范 围的分配，客户机登录服务器时就可以自动获得服务器分配的 IP 地址和子网掩码、网关、 DNSo

DHCP 服务工作原理和安装部署

工作原理

原理： 动态主机设置协定（DHCP）是一种使网络管理员能够集中管理和自动分配 IP 网络地址的通信协议。在 IP 网络中，每个连接 Internet 的设备都需要分配 唯一的 IP 地址。DHCP 使网络管理员能从中心结点监控和分配 IP 地址。当某台 计算机移到网络中的其它位置时，能自动收到新的 IP 地址。

DHCP 使用了租约的概念，或称为计算机 IP 地址的有效期。租用时间是不定的， 主要取决于用户在某地联接 Internet 需要多久，这对于教育行业和其它用户频 繁改变的环境是很实用的。通过较短的租期，DHCP 能够在一个计算机比可用 IP 地址多的环境中动态地重新配置网络。DHCP 支持为计算机分配静态地址，如需 要永久性 IP 地址的 Web 服务器。

DHCP 自动分配 IP 地址的过程：

DHCP 统一使用两个 IANA 分配的端口作为 BOOTP：服务器端使用 67/udp,客户端 使用 68/udpo

DHCP 运行分为四个基本过程，分别为发现阶段、提供 IP、选择 IP 租约和确认 IP 租约。


```
# apt install isc-dhcp-server #安装 dhcp
```

#查看同标主机的网 k 名称

I# ifconfTˆ

DHCP 服务关联的配置文件详解 配置文件详解 /etc/dhcp/dhcpd. conf

option domain-name-servers 202.106.0.20, 202.106.148.1;

DNS 曜务器 IP （北京网通） —全局设置

log-facility local?;日志记录配置 **subnet 192.168.2.0 netmask 255.255.255.0** （

range 192.168.2.100 192.168.2.254;® 网段的 **IPWlt** 池范围 **option routers 192.168.2.1;**路由问壬

default-lease-time 21600;默 **U** 租约 **6** 小时
max-lease-time 43200;昂七租约 **12** 小时

host test （

hardware ethernet 00:0C:29:80:16:3E;某客户机网卡 **MACi** 也址 **fixed-address 192.168.2.188;**指定分配的 **IPi** 蜘

4. 2 DHCP 基础应用

DHCP 服务器端的基本配置

首先

- 此实验建议在单独网络环境进行、关闭虚拟网络的 dhcp 服务，以免影响实验效果，避 免同一网络上有 2 台 dhcp server 提供服务，造成混乱。
- 确定监听网卡的设备名
- 确定需要 dhcp server 分配的 IP 地址段、掩码、网关、DNS

在 uos serverl k

```
# vim /etc/default/isc-dhcp-server
```

```
IXTERFACESv4="ens33                                    #将监听端口修改为对应网 1<
```

```
# vim /etc/dhcp/dhcpd. conf                            #主配置文件
  subnet 192. 168. 200. 0 netmask 255. 255. 255. 0 （ #subnet 后跟子网网段,netmask 后跟 子网掩码
  range 192. 168.200.201 192. 168.200.250;             #分配的 IP 地址起止范围
  opt ion domain-name-servers 202. 106. 0. 20; #DNS 服务器地址（多个地址用”,〃隔开）

  option routers 192. 168. 200. 2;                     #默认网关
  default-lease-time 600;                              #最小租约 600 秒
```

```
max-lease-time 7200;                                    #最大租约 36000 秒
option broadcast-address 192. 168. 200. 255;        #分发广播地址
```

重启

```
# systemctl restart isc-dhcp-server
```

验证是否运行

```
# netstat -uap Igrep :67 # 查看 dhcp 是否正常运行 udp 67
```

DHCP 客户端的基本配置

- 客户端网卡配置选择自动分配即可；
- 默认情况下，UOS 在安装完成后的就是自动分配状态，无需配置。
- 通过以下命令或在图形界面点击网卡配置，可以使客户端立即发出自动配置清求: 打开 uos server2

客户端验证（先 down 掉网卡再重启网卡服务）

```
#   nmcli connection down 有线连接
#   nmcli connection up 有线连接
#   ifconfig
```

根据 MAC 地址分配固定 IP

1. 找到 uos2 客户端 MAC 地址

```
# ifconfig
```

2.在 uosl 上

```
# vim /etc/dhcp/dhcpd. conf
host test (       #主机声明
  hardware ethernet 00:0c :29: a8 :cc: 8f;          #客户机网卡 MAC 地址
  fixed-address 192.168. 200. 202;                  #制定分配的 IP 地址
}
# systemctl restart isc-dhcp-server
```

uos2 客户端验证

```
n nmcli connection down 有线连接
#   nmcli connection up 有线连接
#   ifconfig
```

得到的 IP 地址是 192. 168. 200. 202

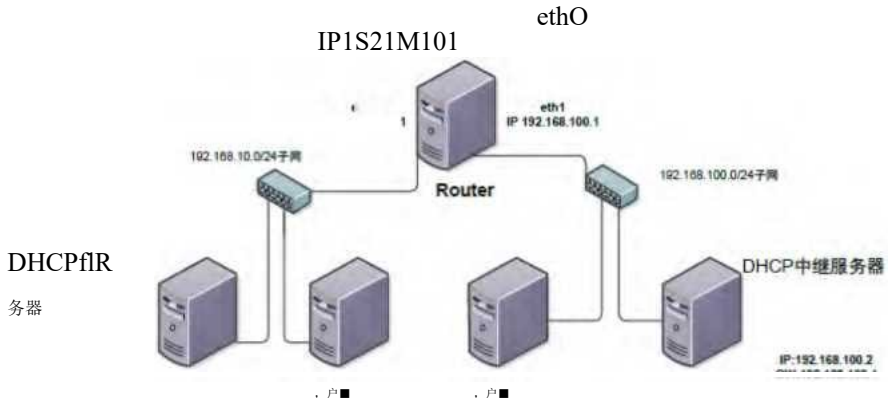
4.4 DHCP 高级应用

DHCP Relay 的功能实现

DHCP 服务器和终端不在同一个网络中，终端无法通过广播到达 DHCP 服务器，必须采用 DHCP 中继到达 DHCP 服务器

用 DHCP Relay 代理可以去掉在每个物理的网段都要有 DHCP 服务器的必要，它可以传递消息 到不在同一个物理子网的 DHCP 服务器，也可以将服务器的消息传回给不在同一个物理子网 的 DHCP 客户机。

DHCP Relay （DHCP 中继），可以实现在不同子网和物理网段之间处理和转发 dhcp 信息的 功能



DHCPflR
务器

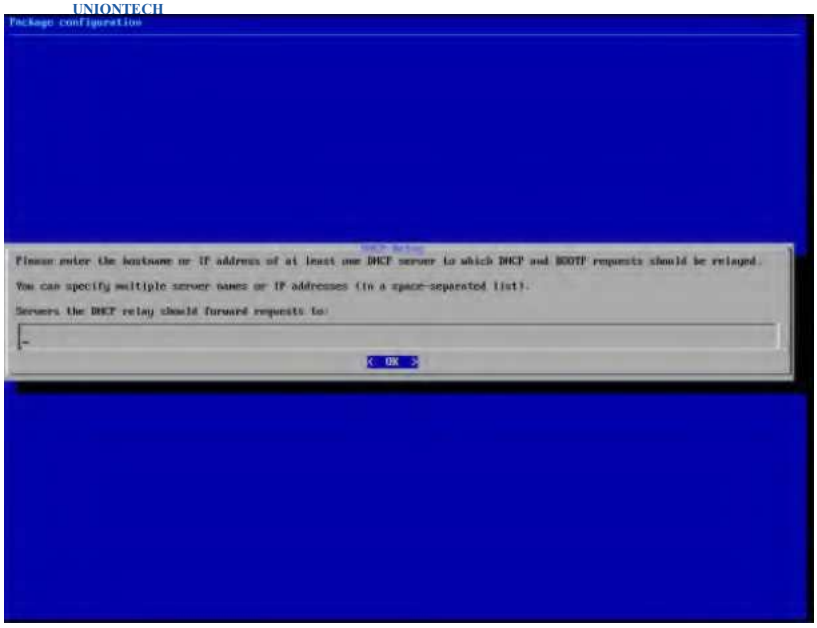
安装配置：

首先，做 relay server 的服务器需要安装两块网卡，分别连接在 dhcp server 所在子网和 需要中继到的子网，并配置好相应的 IP 地址。

安装 relay 软件包：

```
apt install isc-dhcp-relay
```

安装过程中会提示配置 dhcp server 的 ip、监听的网卡名称、启动选项三项配置



配置：

vi /etc/default/isc-dhcp-relay

I tefdelt% for itc rclau iallscript
- Mmrxrd by zeU/lnit .I^ isc-dhcp-relay
- iBtailed at /etczde(Mlt/isc-4hcp-relay hy the Maintainer scripts

- TMB IS a HKIX shell gMmt

- UlMt Kcrucrs shoiiM the IHCF rel«u req.mU *Uft*

SEMKS^"19Z.It>6 i zae"

- On Mhat Interfaces should the DHCP relay (dhrelaiyl serve tMCF ra(uests?

- itioiKi upUmas Uiel are passe*! ta the IHCP reid^ d^enun? WIIORS- —

配置好 dhcp server 的 IP 和要监听的网 F 设备名即可

重启服务：

```
systemctl restart isc-dhcp-relay
```

检查服务状态和监听端 I I：

```
systemctl status isc-dhcp-relay ss -nlup grep :67
```

DHCP 服务器缓存的管理方式

UOS 的 DHCP 服务端如果修改了配置，只要重启服务端 DHCP 进程即可生效，客户端会自动 同步配置，如果遇到特殊情况，可以通过 nmcli 命令或图形界面或拔插网絡等多种方式使配 置生效。

服务器端的缓存文件是/var/lib/dhcpd. leases,特殊情况如需清除缓存，可以: echo "" > /var/lib/dhcp/dhcpd. leases

4.3 项目实战

1. 基于同网段 DHCP 服务的配置

要求：

根据本章所学内容，搭建 DHCP 服务，自动为客户端主机分配指定网段的 IP、掩码、网关、 DNS 四项网络基本参数；

2. 根据企业需求完成分配的 IP 与 MAC 地址绑定的需求

要求：

- 收集客户端网卡的 MAC 地址；
- 为每一个 mac 地址的网卡固定分配 IP 地址；

3. 根据企业复杂的网络环境完成 DHCP 中继的搭建

要求：

设有两个内网段 192.168. 1. 0/24、172. 16. 1. 0/24,

- 在 192. 168. 1. 0/24 网段搭建一台 dhcp server 192. 168. 1. 200,配置为 2 个网段的客 户机分配 IP 等网络参数；
- 搭建一台具有 2 块网卡的 dhcp relay 服务器，为 172.16. 1. 0/24 网段做 dhcp 中继；

4. 定期对 DHCP 服务器缓存的数据进行清理

要求：

- 使用 crontab,每天早上 2： 00 清理一次 dhcp 缓存数据；

第 5 章高阶网络配置

5. 1 链路聚合简介

链路聚合介绍

链路聚合(英语：Link Aggregation)是一个计算机网络术语，指将多个物理端口汇 聚在一起，形成一个逻辑端口，以实现出/入流量吞吐量在各成员端口的负荷分担，交换机 根据用户配置的端口负荷分担策略决定网络封包从哪个成员端口发送到对端的交换机。当交 换机检测到其中一个成员端口的链路发生故障时，就停止在此端口上发送封包，并根据负荷 分担策略在剩下的链路中重新计算报文的发送端口，故障端口恢复后再次担任收发端口。链 路聚合在增加链路带宽、实现链路传输弹性和工程冗余等方面是一项很重

要的技术。

Linux 配置链路聚合的两种方式

网卡的链路聚合一般常用的有"bond 〃和〃 team”两种模式，“ bond”模式最多可以添加两块网 卡，〃teanT 模式最多可以添加八块网卡。

5.2 项目实战

1、实验-网卡链路聚合

(1) bonding 一共有 7 种工作模式

0： (balance-rr) Round-robin policy:(平衡轮询策略)：传输数据包顺序是依次传输， 直到最后一个传输完毕，此模式提供负载平衡和容错能力。

1： (active-backup) Active-backup policy:(活动备份策略)：只有一个设备处于活动 状态。一个宕掉另一个马上由备份转换为主设备。mac 地址是外部可见得。此模式提供了 容错能力。

2： (balance-xor) XOR policy:(平衡策略)：传输根据 [(源 MAC 地址 xor 目标 MAC 地 址)mod 设备数量] 的布尔值选择传输设备。此模式提供负载平衡和容错能力。

3： (broadcast) Broadcast policy:(广播策略)：将所有数据包传输给所有设备。此 模式提供了容错能力。

4： (802. 3ad) IEEE 802. 3ad Dynamic link aggregation. IEEE 802. 3ad 动态链接聚 合：创建共享相同的速度和双工设置的聚合组。此模式提供了容错能力。每个设备需要基于 驱动的重新获取速度和全双工支持；如果使用交换机，交换机也需启用 802. 3ad 模式。

5： (balance-tlb) Adaptive transmit load balancing(适配器传输负载均衡)：通道 绑定不需要专用的交换机支持。发出的流量根据当前负载分给每一个设备。由当前设备处理 接收，如果接受的设备传不通就用另一个设备接管当前设备正在处理的 mac 地址。

6： (balance-alb) Adaptive load balancing:(适配器负载均衡)：包括 mode5,由 ARP 协商完成接收的负载。bonding 驱动程序截获 ARP 在本地系统发送出的请求，用其中之一 的硬件地址覆盖从属设备的原地址。就像是在服务器上不同的人使用不同的硬件地址一样。

实例：

在 uos1 添加 2 块新的网 R

```
nmcli device show | grep DEVICE          #发现新增 2 块网卡 ens33 和 ens37 将原有网卡规则删除掉，以防影响后续实验 nmcli connection delete ens33
```

(2)新建 bond

```
nmcli connection add type bond con-name bond0 mode active-backup ipv4. addresses 192.168.200.201/24〃     #表示添加一个 bond,名称为 bond。，工作模式为主备，IP 为〃192. 168. 200. 201″。
nmcli connection add con-name ens33 ifname ens33 type bond-slave master bond0 #将 ens33 网卡连接添加到这个 bond 中。
nmcli connection add con-name ens37 ifname ens37 type bond-slave master bond0 #将 ens37 网卡连接添加到这个 bond 中。
nmcli  connection up ens33 #启动 bond-slave ens33
nmcli  connection up ens37 #启动 bond-slave ens37
nmcli  connection up bond0 #启动 bond0
ip a   #查看 ens33 和 ens37 网卡 mac 地址相同
cat
/proc/net/bonding/nm-bond #查看 bond 已生效
```

(3)故障测试

```
nmcli device disconnect ens33 cat /proc/net/bonding/nm-bond    #禁掉当前网卡
nmcli device connect ens33 cat /proc/net/bonding/nm-bond       #查看当前的活动网卡为 ens37 #重新添加 ens33 网卡
                                                               #ens33 网卡已成为备用网卡
```

(4)删除 bond 模式的链路聚合

```
nmcli connection delete bond0 nmcli connection delete ens33 nmcli connection delete ens37
systemctl restart NetworkManager
```

2、实验-桥接

还原虚拟机到 ok 状态

```
nmcli connection delete ens33
nmcli connection add type bridge con-name uosbr ifname uosbr
nmcli connection modify uosbr ipv4. method manual ipv4. addresses 192. 168. 200. 201/24 ipv4. gateway 192. 168. 200. 201 ipv4. dns 192. 168. 200. 201
nmcli connection add type bridge-slave con-name uosbrslavel ifname ens33 master uosbr
nmcli connection up uosbr
systemctl restart NetworkManager
nmcli connection show          #如果出现重名的 uosbr 配置文件，需要删除错误的 uuid,再 nmcli connection up uosbr 和 systemctl restart NetworkManager
ping -I uosbr 192.168. 100. 202      #-I 表示从哪块网卡发出 ping 包
```

3、实验-删除虚拟桥接网卡 virbr0 方法 1

```
apt-get remove -y libvirt-* reboot
nmcli connection show
                                    #虚拟桥接网卡 virbr0 已删除
```

方法 2

```
virsh net-list                      #查看虚拟网络设备
virsh net-destroy default             #删除名为 default 的设备
virsh net-undefine default            #删除配置文件中的相关信息
systemctl restart libvirtd
nmcli connection show               #虚拟桥接网卡 virbr0 已删除
```

4、实验-添加 ipv6 地址

```
nmcli connection show          #查看配置 ipv6 的网卡名称，这里为 ens33
nmcli connection modify ens33 ipv6. method manual ipv6. addresses 2001::1/64 ipv6. gateway
2001::1 ipv6. dns 2001::1
nmcli connection down ens33 ; nmcli connection up ens33
ping6 2001::1
```

第 6 章时钟同步服务

6. 1 Chrony 服务概述

时间同步服务器的作用

时间和空间是宇宙的两大法则，准确的时间对于计算机系统来说至关重要，时间服务器的作 用就是对时；

原因是：

- 每台 pc 或服务器的时间晶振发生器都有误差，过一段时间就会发生时间不准的现象;
- 很多服务、集群、交易系统都需要精准的时间来进行服务，否则就会出现故障； 互联网上有很多的公共时间服务器，如:

Internet 空间服务器 pool. ntp. org 或

0. pool. ntp. org

1. pool. ntp. org

2. pool. ntp. org

3. pool. ntp. org 以亚洲为例： asia. pool. ntp. org 或

0. asia. pool. ntp. org

1. asia. pool. ntp. org

2. asia. pool. ntp. org

3. asia. pool. ntp. org 以中国为例：

cn. pool. ntp. org 或

0. cn. pool. ntp. org

1. cn. pool. ntp. org

2. cn. pool. ntp. org

3. cn. pool. ntp. org 企业

ntp. aliyun. com

Chrony 的介绍与工作原理

NTP 是网络时间协议(Network Time Protocol),它是用来同步网络中各个计算机的时间的 协议。

- Chrony 是新一代的时间同步服务，用来代替旧的 ntpd 服务;
- Chrony 是一个开源的自由软件，它能帮助你保持系统时钟与时钟服务器(NTP)同步， 因此让你的时间保持精确。它由两个程序组成，分别是 chronyd 和 chronyc。
- chronyd 是一个后台运行的守护进程，用于调整内核中运行的系统时钟和时钟服务器同 步。它确定计算机增减时间的比率，并对此进行补偿。
- chronyc 提供了一个用户界面，用于监控性能并进行多样化的配置。它可以在 chronyd 实例控制的计算机上工作，也可以在一台不同的远程计算机

Chrony 服务部署和基本配置

安装

```
apt install chrony
```

修改配置文件：

vi /etc/chrony/chrony. conf

添加：

```
server cn. pool. ntp. org
```

表示这台服务器向 cn. pool. ntp. org 来中请时间同步。

重启：

```
systemctl restart chrony
```

服务的配置文件详解

/etc/chrony/chrony. conf 配置项说明：

- server：指明时间服务器地址；
- allow NETADD/NETKL4SK 允许那些客户端来同步
- allow all：；允许所有客户端主机
- deny NETADDR/NETMASK
- deny all：拒绝所有客户端；
- b indcmdaddres s:命令管理接口监听的地址；
- local stratum 10：即使自己未能通过网络时间服务器同步到时间，也允许将本地时 间作为标准时间授时给其它客户端。

6. 2 Chrony 配置与应用

Chrony 服务端的配置与应用

在 uos serverl 11

安装服务

```
apt install chrony
```

修改配置文件以下内容 vim /etc/chrony/chrony. conf

```
pool 2. debian. pool. ntp. org iburst     #将原先的 ntp 时间同步配置注释掉
allow 192.168. 200. 0/24                  #允许哪些客户端来同步主机的时间
local stratum 10                          #增加，本机不同步任何主机时间，本机作为时间源
```

重启验证：

```
systemctl restart chronyd                 #重启服务
systemctl enable chronyd                  #开机启动
netstat -antulp grep chronyd              #查看时间服务器是否允许

timedatectl                               #显示系统当前日期和时间
```

Chrony 客户端的配置与应用

安装服务

```
apt install chrony
```

修改配置文件以下内容 vim /etc/chrony/chrony. conf

```
#pool 2. debian. pool. ntp. org          #注释此行
server 192. 168. 200. 201 iburst         #将时间服务器指向我们自建的服务器，burst 表示当此 NTP 服务器不可用时，向它发送一系列的并发包进行检测
```

重启验证：

```
systemctl restart chronyd ss -anptu I grep chronyd
```

timedatectl 检查同步结果

```
      Local time: H 2020-08-27 13:35:00 CST Universal time:四 2020-08-27 05:35:00 UTC
          RTC time:四 2020-08-27 13:32:28
         Time zone: Asia/Shanghai (CST, +0800)
System clock synchronized: no
           NTP service: active
       RTC in local TZ: yes

Warning: The system is configured to read the RTC time in the local time zone.
       This mode cannot be fully supported. It will create various problems with time
       zone changes and daylight saving time adjustments. The RTC time is never updated,
       it relies on external facilities to maintain it. If at all possible, use RTC in
       UTC by calling
         * timedatectl set-local-rtc 0'.
```

chronyc sources - v #查看时间同步源

```
MS Name/IP address         Stratum Poll Reach LastRx Last sample
 * 192. 168. 200. 201   106     37     35 +2035ns[+8891ns] +/- 157us
```

chronyc sourcestats - v #查看时间同步源状态

Xame/IP Address	NP	NR	Span	Frequency	Freq Skew	Offset	Std Dev
192. 168. 200. 201	4	4	194	+0. 000	12.437	+0ns	50us

时间设置工具 timedatectl 的运用

- 查看当前时间/日期/时区：timedatectl 或者 timedatectl status
- 查看所有可用时区：timedatectl list-timezones
- 设置时区：timedatectl set-timezone "时区信息"
- 设置 UTC： timedatectl set-timezone UTC
- 设置时间：timedatectl set-time HH:MM:SS
- 设置日期：timedatectl set-time YYYY-MM-DD
- 设置日期时间：timedatectl set-time "YYYY-MM-DD HH:MM:SS"
- 设置硬件时钟为本地时间：timedatectl set-local-rtc 1
- 设置硬件时钟为 UTC 时间：timedatectl set-local-rtc 0
- 启动 NTP 时间同步（启用 NTP 服务或者 Chrony 服务）：timedatectl set-ntp true
- 禁用 NTP 时间同步：timedatectl set-ntp false

Chronyc 命令和参数详解

- accheck -检查 NTP 访问是否对特定主机可用
- activity -该命令会显示有多少 NTP 源在线/离线
- add server -手动添加一台新的 NTP 服务器。
- clients -在客户端报告已访问到服务器
- delete -手动移除 NTP 服务器或对等服务器
- settime -手动设置守护进程时间
- tracking -显示系统时间信息

6.3 项目实战

根据企业服务器的运行需求，实现服务器时间同步

解决 lintix 系统时间与 BIOS 不一致的问题

timedatectl set-local-rtc 1 一一 adjust-system-clock

或者使用旧的命令 ntpdate：

```
sudo apt-get install ntpdate
sudo ntpdate timel. cloud, tencent. com
sudo hwclock 一一 systohc
```

第 7 章防火墙

7. 1 防火墙概述

防火墙技术的基本概念

防火墙技术是通过有机结合各类用于安全管理与筛选的软件和硬件设备，帮助计算机网 络于其内、外网之间构建一道相对隔绝的保护屏障，以保护用户资料与信息安全性的一种技 术。

防火墙技术的功能主要在于及时发现并处理计算机网络运行时可能存在的安全风险、数 据传输等问题，其中处理措施包括隔离与保护，同时可对计算机网络安全当中的各项操作实 施记录与检测，以确保计算机网络运行的安全性，保障用户资料与信息的完整性，为用户提 供更好、更安全的计算机网络使用体验。

数据包过滤的概念和机制

Netfilter 与 iptables

Linux 真正的防火墙安全框架是 netfilter ，它位于 Linux 的内核空间。
iptables 本质是一个命令行工具，位于用户空间，它相当于一个代理，将用户的安全 设定执行到对应的安全框架 netfilter 中。

- Iptables 是属于网络层的防火墙，但是并不正真意义上是防火墙，因为 iptables/netfilter 是一个组件，iptables 只是负责编写规则并提交给 netfilter 做 执行的规则生成器。
- Netfiter 是在 linux 内核中 TCP/IP 协议栈中工作的一个框架，从软件的角度来将是在 TCP/IP 协议栈中做了五个钩子函数，这五个钩子函数可非常准确的执行 iptables 所编 写的规则并实现规则中相关的拦截和放行。
- 规则的功能分别为 raw, mangle, nat, filter 四种，这些规则统称为表；
- 而钩子函数分为 PREROUTING、INPUT> FORWARD、OUTPUT、POSTOUTING 五种，这些钩子 函数统称为链

钩子函数是指：

PREROUTING：马上就要到本机时，简称为路由前
INPUT：到达本机内部的报文必经之路
FORWARD：由本机转发的报文必经之路
OUTPUT：由本机发出的报文的必经之路
POSTROUTING：马上就要离开本机，简称为路由后

规则的功能是指：

filter:过滤，定义是否允许通过防火墙
nat:地址转换，用于转换源地址和源端口或目标地址和目标端口 mangle:用于修改报文首部某些特性但不修改 IP raw:目标是为 nat 表上启用的连接追踪功能

表和链的对应关系：

filter:INPUT, FORWARD, OUTPUT
nat:PREROUTING （SNAT） , POSTROUTING （DNAT） , OUTPUT
mangle:PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING
raw:PREROUTING, OUTPUT

数据包经过防火墙的流程

1、 当主机收到一个数据包后，数据包先在内核空间中处理，若发现目的地址是自身， 则传到用户空间中交给对应的应用程序处理；若发现目的不是自身，则会将包丢弃或进行转 发。

2、 iptables 实现防火墙功能的原理是：在数据包经过内核的过程中有五处关键地方， 分别是 PREROUTING、INPUT> OUTPUT> FORWARD> POSTROUTING,称为钩子函数，iptables 可以在这 5 处地方写规则，对经过的数据包进行处理，规则一般的定义为“如果数据包头 符合这样的条件，就这样处理数据包”。

3> iptables 中定义有 5 条链，说白了就是上面说的 5 个钩子函数，因为每个钩子函数 中可以定义多条规则，每当数据包到达一个钩子函数时，iptables 就会从钩子函数中第一 条规则开始检查，看该数据包是否满足规则所定义的条件。如果满足，系统就会根据该条规 则所定义的方法处理该数据包；否则 iptables 将继续检查下一条规则，如果该数据包不符 合钩子函数中任一条规则，iptables 就会根据该函数预先定义的默认策略来处理数据包

4、iptables 中定义有表，分别表示提供的功能，有 filter 表（实现包过滤）、nat 表（实现网络地址转换）、mangle 表（实现包修改）、raw 表（实现数据跟踪），这些表具 有一定的优先级：raw —>mangle—>nat—>filter

ufw 防火墙的概述

UFW 全称为 Uncomplicated Firewall,是 Ubuntu 系统上默认的防火墙组件，为了轻量化配 置 iptables 而开发的一款工具。UFW 提供一个非常友好的界面用于创建基于 IPV4, IPV6 的防火墙规则。

Gufw 图形化管理工具

Gufw 是 ufw 的图形化前端

常见问题

如何用系统自动启动 Gufw?

你并不需要它。当你在 Gufw 中做了所有的变化后，设置仍然在适当的位置，直到下一次改 变。

为什么 Gufw 默认禁用？

默认情况下，防火墙不打开外部进入本机的端口。这是从安全角度出发做出的规则设定。

有些规则是 Gufw 自己添加的？

当你编辑或导入一个或一组规则，Gufw 会添加相应的规则，并且包括添加 ipv4 和 ipv6 规 则。

什么是允许，否认，拒绝和限制？

- 允许： 将允许流量。allow
- 否认： 会否认流量。reject
- 拒绝： 将否认流量，并通知其已被拒绝。deny 会否认流量，如果一个 IP 尝试几个连接的话。
- 限制：

我在监听报告里看到了什么？

显示了 TCP 的处于监听状态的和 UDP 的处于打开状态的实时系统上的端口。

更多括助信息参见官方文档：

help, ubuntu, com/community/Gufw

7. 2 UFW 基本语法

防火墙 ufw 安装和部署

安装：

apt install ufw 安装 Gufw:

```
apt install gufw
```

启用 ufw:

```
ufw enable
```

禁用 ufw

```
ufw disable
```

ufw 防火墙的基本规则精讲

检查 UFW 的状态：

ufw status verbose 或者 ufw status

输出应如下所示：

//没开启(disable)是这个样子的
Status: inactive

〃开启(enable)后是这样子的
Status: active
//如果你添加了防火墙规则下面这里就会显示
Logging: on (low)　　〃有效正在登录：(低)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip　　//默认值：拒绝(传入),允许(传出)新的个人资料：跳过用户

To	Action	From
22	ALLOW IN	Anywhere

请注意，默认情况下，拒绝将应用于传入。

允许
#　ufw allow〈端口〉/〈可选：协议〉
示例：允许在端口 53 上传入 tcp 和 udp 数据包
#　ufw allow 53 示例：允许端口 53 上的传入 tcp 数据包

```
# ufw allow 53/tcp
```

示例：允许端口 53 上的传入 udp 数据包

```
# ufw allow 53/udp
```

拒绝

ufw deny〈端口〉/〈可选：协议〉 示例：拒绝端口 53 上的 tcp 和 udp 数据包

ufw deny 53 示例：拒绝端口 53 上的传入 tcp 数据包

```
# ufw deny 53/tcp 示例：拒绝端口 53 上的传入 udp 数据包
# ufw deny 53/udp
```

使用服务名

您也可以按服务名称允许或拒绝，因为 ufw 从/ etc / services 中读取 要查看获取服务 表：

按服务名称允许

ufw allow〈服务名称〉

示例：按名称允许 ssh

ufw allow ssh

按服务名称拒绝

ufw deny〈服务名称〉 示例：按名称拒绝 ssh

ufw deny ssh

删除现有规则

要删除规则，只需在原始规则前面加上 delete。例如，如果原始规则是:

ufw deny 80/tcp

使用它删除它：

ufw delete deny 80/tcp

关联的配置文件详解

ufw 相关的文件和文件夹有：

- /etc /ufw/：里面是一些 ufw 的环境设定文件，如 before, rules^ after, rules> sysctl. conf^ ufw. conf,及 for ip6 的 before6. rule 及 after6. ruleso 这些文件 一般按照默认的设置进行就 Oko
- /var/lib/ufw/user. rules 这个文件中是我们设置的一些防火墙规则，打开大概就能看 明白，有时我们可以直接修改这个文件，不用使用命令来设定。修改后记得 ufw reload 重启 ufw 使得新规则生效。

注意：

若开启 ufw 之 后，**/etc/ufw/sysctl. conf** 会覆盖默认的**/etc/sysctl. conf** 文件，若 你原来的/etc/sysctl. conf 做了修 改，启动 ufw 后，若/etc/ufw/sysctl. conf 中有新赋值， 则会覆盖

/etc/sysctl. conf 的，否则还以/etc /sysctl. conf 为准。当然你可以通过修改 /etc/default/ufw 中的 “IPT_SYSCTL= ”条目来设置使用哪个 sysctrl. conf.

安装后，ufw 默认不启动，使用 ufw enable 启动它

默认策略：

- 进入数据拒绝
- 转发拒绝
- 发出数据允许。
- 默认策略跟踪进入’转发的新连接。
- 除此外还增加了下列默认规则集：
  - -DROP packets with RHO headers

    丢弃含 RHO 头的数据
  - -DROP INVALID packets

    丢弃无效数据
  - -ACCEPT certain icmp packets (INPUT and FORWARD):

    允许 icmp 的输入和转发包

ufw 防火墙的转发规则

语法格式：

路由［删除］［插入第行］允许’拒绝’拒绝并提示’限制［数据进入’发出［网络接口］］［记 录’全记录］［协议**］［来自**［端口 **］］［指向**［端口**］

ufw [--dry-run] route [delete] [insert NUM] [prepend] allow I deny I reject|limit [in|out on INTERFACE] [log|log-all] [proto PROTOCOL] [from ADDRESS [port PORT | app APPNAME]] [to ADDRESS [port PORT | app APP NAME]] [comment COMMENT]

示例：

允 许来自 192. 168. 0. 0-192. 168. 255. 255 通 过 eth0 网 卡收入 的数据 Ft 指向 10. 0. 0. 0-10. 255. 255. 255 通过 ethl 网卡发出的数据经本机路由

ufw route allow in on eth0 out ethl to 10. 0. 0. 0/8 from 192. 168. 0. 0/16

7.3 UFW 高级语法

高级 ufw 防火墙规则编写

允许访问

本节说明如何允许特定访问。

通过特定 IP 允许

ufw alllow from〈IP 地址〉

示例：允许来自 192.168.200.10 的数据包：

ufw allow from 192.168.200.10

(1) 允许子网

您可以使用网络掩码：

ufw allow from 192. 168. 200. 0/24

(2) 通过特定的端口和 IP 地址允许

ufw allow from <target> to <destination> port <port number>

ufw 允许从〈目标>到〈目标>端口〈端口号〉

示例：允许所有协议的 IP 地址 192. 168. 200. 20 访问端口 22

ufw allow from 192. 168. 200. 20 to any port 22

(3) 允许通过特定的端口，IP 地址和协议

ufw allow from <target> to〈destination〉port <port number> proto <protocol name>

ufw 允许从〈目标>到〈目标〉端口〈端口号〉协议〈协议名称〉

例如：允许 IP 地址 192.168.200.20 使用 TCP 访问端口 22

ufw allow from 192. 168. 200. 20 to any port 22 proto tcp

(4) 启用或禁用 PING 反馈

注意：使用隐秘的安全性对于现代解密器脚本可能几乎没有实际好处。默认情况下，UFW 允 许 ping 请求。您可能会发现希望保留（icmp） ping 请求以诊断网络问题。

为了禁用 ping （icmp）请求，您需要编辑/etc/ufw/before. rules 并删除以下几行：

#vim /etc/ufw/before. rules

```
# ok icmp codes for INPUT
-A ufw-before-input    -p  icmp —icmp-type    destination-unreachable -j ACCEPT time-
-A ufw-before-input    -p  icmp —icmp-type    exceeded -j ACCEPT
-A ufw-before-input    -p  icmp —icmp-type    parameter-problem -j ACCEPT
-A ufw-before-input -p     icmp —icmp-type    echo-request -j ACCEPT
```

或将“接受”更改为“丢弃”

```
# ok icmp codes for INPUT
-A ufw-before-input -p     icmp —icmp-type    destination-unreachable -j DROP
-A ufw-before-input -p     icmp —icmp-type    time-exceeded -j DROP
-A ufw-before-input    -p  icmp —icmp-type    parameter-problem -j DROP
-A ufw-before-input -p     icmp —icmp-type    echo-request -j DROP
```

拒绝访问

(1)通过特定 IP 拒绝

ufw deny from <ip address>

ufw 从〈IP 地址〉拒绝

示例：要阻止来自 192. 168. 200. 20 的数据包:

ufw deny from 192.168. 200. 20 通过特定的端口和 IP 地址拒绝

ufw deny from <ip address> to <protocol> port <port number>

ufw 拒绝从〈IP 地址〉到〈协议〉端口〈端口号〉

示例：对于所有协议,拒绝 IP 地址 192.168.200.20 访问端口 22

ufw deny from 1192. 168. 200. 20 to any port 22

(2) 使用编号规则

带有参考编号的上市规则

您可以使用编号的状态来显示规则的顺序和 ID 号：

ufw status numbered

编辑编号规则

删除编号规则

然后，您可以使用数字删除规则。这将删除第一个规则，并且规则将向上移动以填充列表。

ufw delete 1

插入编号规则

ufw insert 1 allow from <ip address>

应用程序的集成管理

应用程序集成管理

ufw 能从/etc/ufw/applications. d.中读取应用程序清单。你可以使用命令查看:

ufw app list

*大家可以使用应用程序名字来增加规则。比如

```
# ufw allow〈程序名字〉
# ufw allow CUPS
# ufw allow from 192. 168. 0. 0/16 to any app〈程序名字〉
```

注意，端口号已经被程序名所对应的策略所包括，不要再重新列举端口号。

*查看程序名所对应的策略内容，命令：

```
# ufw app into〈程序名字〉
```

注意：程序名字是清单上有的才行。程序名字改用用 all,可以看全部策略。

*如果你编辑或者增加了程序清单，你可使用此命令更新防火墙：

ufw app update〈程序名字〉

程序名字改用用 all,则更新整个清单。

*更新清单同时增加规则可以使用如下命令：

```
# ufw app update - add-new〈程序名字〉 注意：update - add-new 参数的行为由此命令配置：
# ufw app default skip|allow|deny
```

默认是 skip,也就是没有设定。

警告：如果程序规则设定为 default allow ,将会引起很大的风险。请三思而后行！

ufw 防火墙规则优化

随着时间的推移，在我们使用系统的过程中，因为日常需要是在不断变化的，我们先后添加 了很多条防火墙的规则，而这些规则可能是相互矛盾的，这就需要我们常常检查和优化这些 规则。

比如：

装完系统，我们添加了： ufw allow ssh

一个月后，又添加了： ufw allow from 192. 168. 1. 0/24 to any port 22 proto tcp

二个月后，又添加了： ufw allow from 172. 16. . 0. 0/24 to any port 22 proto tcp

检查一下当前的状态:

ufw status numbered

Status: active

	To	Action	From
[1]	10. 0. 0. 0/8 on ethl	ALLOW FWD	192. 168. 0. 0/16 on eth0
[2]	22/tcp	ALLOW IN	192. 168. 1.0/24
[3]	22/tcp	ALLOW IN	Anywhere
[4]	22/tcp	ALLOW IN	172. 16. 0. 0/24
[5]	22/tcp (v6)	ALLOW IN	Anywhere (v6)

发现有 4 条关于 ssh 的规则,而我其实只想让 192.168. 1. 0/24 和 172.16. 0. 0/24 网段的 IP 可以连接我的计算机，多余的 3 和 5 号应该删除：

ufw delete 5

ufw delete 3

注意执行顺厅，规则序号会随观则数量和顺厅变化

入侵检测及防护

这里我们以查询系统登陆日志+使用 ufw 拒绝多次尝试登陆 IP 的方式，档住黑客入侵的企图:

首先，找出多次尝试登陆系统的 IP： grep “Failed password″ /var/log/auth.log grep -Po ″from \K\d+\. \d+. \d+. \d+〃|uniq -c I tee 1 如果最次错误登陆次数大于 10 次，我们就把这个 IP 加入 ufw 的 deny 规则： cat 1|awk ’$1〉 10{print ″ufw deny ”$2} ‘ |sh 我们可以把这些命令规则加入到 root 的 crontab 里，周期性的检测和执行，这样就完成了 一个简单的入侵检测策略。

解释日志条目

Ufw 支持许多日志级别。默认是低级(low),用户也可以自己指定：

命令基本语法：

ufw logging on I off|low medium high full

> * off 就是关闭日志
> * low 记录与默认策略冲突的封装数据包(记录速度被限制)o 记录与规则符合的数据

vim /etc/samba/smb. conf

> * medium 记录与默认策略冲突的数据包（包括被规则允许的）、无效数据包、所有新 连接。记录速度被限制。
> * high 同 medium,只是没有记录速度限制。附加记录所有数据包（有记录速度限制）。
> * full 与 high 等同，只是取消记录速度限制。

注意：

- medium 级别及更上级会记录许多内容，有可能短时间内撑爆你的硬盘。特别是用在服 务器一类的机器上。
- on 与 off 只是起开关作用，不代表级别。

UFW 日志分析

Ufw 日志会记录到系统的/var/log/kern. log 和/var/log/messages 文件中 SPT 和 DPT 值以及 SRC 和 DST 值通常是分析防火墙日志时要关注的值。

口志字段说明：

Feb 4 23:33:37 hostname kernel: [3529.289825] [UFW BLOCK] IN=eth0 OUT= MAC=00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd SR0444. 333.222. I11 DST=111.222.333.444 LEN=103 TOS=0x00 PREC=0x00 TTL=52 ID=0 DF PROTO=UDP SPT=53 DPT=36427 LEN=83

Date #日期，观察日期和时间是个好习惯。如果情况不正常或缺少时间段，则攻击者可能会弄乱您的日志。

Hostname #主机名

Uptime #正常运行时间 自启动以来的时间（以秒为单位）。

Logged Event #1 已录的事件的简短描述；例如[UFW BLOCK]

IN #如果设置，则该事件是传入事件。

OUT #如果设置，则该事件为传出事件

MAC #按照以太网 II 标头中的顺厅，这提供了 14 字节的目标 MAC,源 MAC 和 EtherType 字段组合。有关更多信息，请参见以太网帧和 EtherType。

SRC 这表示最初发送数据包的源 IPo 一些 IP 可通过 Internet 路由，某些 IP 仅通过 LAN 进行通信，而某些 IP 仅路由回到源计算机。有关更多信息，请参见 IP 地址。

DST 这表示目的 IP,意在接收数据包。您可以使用 whois, net 或 cli whois 确定 IP 地址的所有者。

LEN #这表示包的长度

TOS #我相信这是指 IPv4 标头的 TOS 字段。有关更多信息，请参见 IPv4 优先级字段的 TCP 处理。

PREC #我相信这是指 IPv4 标头的“优先级”字段。

TTL #这表示数据包的“生存时间”。基本上，每个数据包只会在死亡和消失之前通过给定数量的路由器反弹。如果在 TTL 过期之前仍未找到其目的地，则数据包将蒸发。 此字段可防止丢失的数据包永远阻塞互联网。有关更多信息，请参见生存时间。

ID 可能是 ufw 的内部 ID 系统，也可能是操作系统的 ID。

PROTO 这表示数据包的协议 TCP 或 UDPo

SPT 这表明了来源。我相信这是 SRC IP 通过其发送 IP 数据包的端口。有关更多信息，请参见 TCP 和 UDP 端口号列表。

DPT 这指示目标端口。我相信这是 SRC IP 将其 IP 数据包发送到的端口，期望该端口上正在运行服务。

WINDOW #窗口，这表明发送方愿意接收的数据包大小。

RES 该位保留供将来使用，并且始终设置为 0。基本上，与日志读取目的无关。

SYN URGP SYN 表示此连接需要三向握手，这是 TCP 连接的典型代表。URGP 指示紧急指针字段是否相关。0 表示不是。防火墙日志读取并不重要。

7.4 项目实战

1. 根据企业网站需求完成特定端口 /IP 地址/协议的放行

对于企业的网站安全，通常

要求：

- 允许来自公司办公网的 ssh 和 VPN 的正常连接；
- 允许来自外部的正常 http 和 https 的访问；
- 允许网站内部网的访问；
- 拒绝其它一切访问；

2. 根据开启的防火墙日志进行分析和优化

根据上节日志字段的说明，我们就可以通过编写 shell 脚本命令来筛选、分析日志，从而找 出可能的防火墙漏洞，优化防火墙的规则。

比如，我们想找出所有连接过 ssh 22 端口的所有 IP,可以执行：

```
grep -ri ufw /var/log/messages|grep DPT=22|grep -Po 〃. *SRC=\K\d+\. \d+. \d+. \d+〃
```

第 8 章 NFS 服务

8. 1 NFS 服务概述

NFS 服务的介绍和工作流程

NFS 的简介

NFS 是 Network File System 的缩写，中文意思是网络文件系统。它的主要功能是通过网络 （一般是局域网）让不同的主机系统之间可以共享文件或目录。
NFS 客户端通过挂载（mount）的方式将 NFS 服务器端共享的数据目录挂载到本地系统中。 从客户端本地看，NFS 服务器端共享的目录就好像是客户端自己的目录一样。

什么是 RPC

虽然 NFS 可以在网络中进行文件共享，但 NFS 在设计时并没有提供数据传输的功能。需要借 助 RPC （Remote Procedure Calls,远程过程调用）。RPC 定义了一种进程间通过网络进行交 互通信的机制，它允许客户端进程通过网络向远程服务进程请求服务，而不需要了解服务器 底层的通信协议详细信息。
RPC 和 NFS 的关系：NFS 是一个文件系统，而 RPC 是负责负责信息的传输。
NFS 依赖 RPC 与外部通信

命令：

```
rpcinfo -p 192. 168. 100. 1
```

可以查询指定主机的 RPC 服务信息

服务端需要安装的软件包：

nfs-kernel-server NFS 服务端
rpcbind 支持安全 NFS RPC 服务的连接

NFS 服务端守护进程

- nfsd： 它是基本的 NFS 守护进程，主要功能是管理客户端是否能够登录服务器；
- rpc.mountd：它是 RPC 安装守护进程，主要功能是管理 NFS 的文件系统。当客户端顺利 通过 nfsd 登录 NFS 服务器后，在使用 NFS 服务所提供的文件前，还必须通过文件使用 权限的验证。它会读取 NFS 的配置文件/etc/exports 来对比客户端权限。
- rpcbind：主要功能是进行端口映射工作。当客户端尝试连接并使用 RPC 服务器提供的 服务（如 NFS 服务）时，rpcbind 会将所管理的与服务对应的端口提供给客户端，从而 使客户可以通过该端口向服务器请求服务。

vim /etc/samba/smb. conf

8. 2 NFS 配置与应用

NFS 服务端配置

安装

apt -y install rpcbind nfs-kernel-server

新建测试用的三个目录以及文件

mkdir /uosl /uos2 /uos3

touch /uosl/file{1.. 9}

touch /uos2/file(10. . 19}

touch /uos3/file(100.. 109)

增加一个用户

```
useradd -u 1001 -m nfsuser #nfsuser 对应 uid 为 1001
chmod o+w /uosl
```

配置共享目录

vim /etc/exports

```
/uosl *(rw, anonuid=1001, anongid=1001, sync,no_wdelay)
/uos2 192. 168. 200. 0/24(ro)
/uos3 192. 168. 200. 2(rw, no_root_squash, sync, no_wdelay)
```

说明：/usrl 所有远程连接用户获得 nfsuser 身份，共享/uosl 目录 Ai sr2:以只读方式挂载
/usr3 不压制 root 权限

重启 nfs 相关服务

```
systemctl restart nfs-kernel-server. service
systemctl restart rpcbind
```

NFS 共享配置文件详解

/etc/exports 是 NFS 共享配置文件

指定共享给哪些主机：

- 指定 ip 地址的主机： 192. 168. *. *
- 指定子网中的所有主机：192. 168. *. */24 192. 168. *. */255. 255. 255. 0
- 指定域名的主机： nfs. uos. com
- 指定域中的所有主机： *. uos. com
- 所有主机： *

选项：用来设置输出目录的访问权限、用户映射等。NFS 主要有 3 类选项：

访 I、可权限选项

- 设置输出目录只读：ro
- 设置输出目录读写：rw

用户映射选项

- all_squash：　　将远程访问的所有普通用户及所属组都映射为匿名用户或用户组（nfsnobody）；
- no_all_squash：　与 all_squash 取反（默认设置）；
- root_squash：　　将 root 用户及所属组都映射为匿名用户或用户组（默认设置）；

vim /etc/samba/smb. conf

- no_root_squash： 与 rootsquash 取反；
- anonuid^xxx： 将远程访问的所有用户都映射为匿名用户，并指定该用户为本地用户

- anongid=xxx： 将远程访问的所有用户组都映射为匿名用户组账户，并指定该匿名用 户组账户为本地用户组账户（GID=xxx）；（UID=xxx）；

其它选项

- secure： 限制客户端只能从小于 1024 的 tcp/ip 端口连接 nfs 服务器（默认设置）；
- insecure： 允许客户端从大于 1024 的 tcp/ip 端口连接服务器；
- sync： 将数据同步写入内存缓冲区与磁盘中，效率低，但可以保证数据的一致性；
- async： 将数据先保存在内存缓冲区中，必要时才写入磁盘；
- wdelay： 检查是否有相关的写操作，如果有则将这些写操作一起执行，这样可以提高效率 （默认设置）；
- no_wdelay： 若有写操作则立即执行，应与 sync 配合使用；
- subtree： 若输出目录是一个子目录，则 nfs 服务器将检查其父目录的权限（默认设置）；
- no_subtree： 即使输出目录是一个子目录，nfs 服务器也不检查其父目录的权限，这样可以提高效率;

NFS 的普通挂载方式

在 uos server2 I：面安装 NFS 客户端软件 apt -y install nfs-common

检视 uos server 1 I：共享的冃录：

showmount -e 192.168. 200. 201

输出

```
Export list for 192.168.200. 201:
/sharel *
/share2 192.168.200.0/24
/share3 192.168.200.2
```

新建挂载冃录

mkdir /mnt/nfsl /mnt/nfs2 /mnt/nfs3 挂载 uos server 1 共享的目录：

mount 192. 168. 200. 201:/uosl /mnt/nfsl

mount 192. 168. 200. 201:/uos2 /mnt/nfs2

mount 192. 168. 200. 201:/uos3 /mnt/nfs3

在共享的目录中试建几个新文件

touch /mnt/nfsl/uostest	#返回 uosl 上查看文件属主为 nfsuser
touch /mnt/nfs2/uostest	#不能写
touch /mnt/nfs3/uostest	#返回 uosl 上查看文件属主为 root

NFS 的自动挂载与 fstab 开机挂载

卸载已 mount 的同录

umount /mnt/nfsl

编辑 fstab

vim /etc/fstab

192. 168. 200. 201:/uosl /mnt/nfsl nfs defaults, _rnetdev 0 0

重新挂载

mount -a

autofs 的用法详解

安装 autofs

```
apt install -y autofs systemctl start autofs
```

配置 autofs

vim /etc/auto. master ,/mnt/nfs2 /etc/auto. nfs2 cp /etc/auto. misc /etc/auto. nfs2 vim /etc/auto. nfs2 nfs2 -fstype=nfs 192. 168. 200. 201:/share2 重启 autofs 服务

```
systemctl restart autofs. service
```

验证

df -h

cd /mnt/nfs2

Is -1

NFS 服务的权限控制

在共享目录配置文件/etc/exports 中，以下几项与权限有关：

- ro：设置输出目录只读
- rw:设置输出目录读写
- no_root_squash：登入 NFS 主机使用分享目录的使用者，如果是 root 的话，那么对 于这个分享的目录来说，他就具有 root 的权限！这个项目『极不安全』，不建议使用！
- root_squash：在登入 NFS 主机使用分享之目录的使用者如果是 root 时，那么这个使 用者曲权限将被压缩成为匿名使用者，通常他的 UID 与 GID 都会变成 nobody 那个系 统账号的身份；

- all_squash： 将远程访问的所有普通用户及所属组都映射为匿名用户或用户组 (nfsnobody)；
- no_all_squash： 与 all_squash 取反(默认设置)；
- anonuidˆxxx： 将远程场问的所有用户都映射为匿名用户，并指定该用户为本地用户 (UID=xxx)；
- anongid=xxx： 将远程访问的所有用户组都映射为匿名用户组账户，并指定该匿名用

户组账户为本地用户组账户(GID=xxx)；

具有示例可参见前述《NFS 服务端配置》内容

8.3 项目实战

1 .为满足公司资源共享的需求，部署 NFS 服务器

安装：

```
apt -y install rpcbind nfs-kernel-server
```

2.根据企业架构，优化 NFS 服务器，实现不同员工访问 NFS 服务器的权限不同

新建测试用的三个目录以及文件

```
mkdir /uosl /uos2 /uos3
touch /uosl/file{1.. 9)
touch /uos2/file(10. .19)
touch /uos3/file(100.. 109)
```

增加一个用户

```
useradd -u 1001 -m nfsuser #nfsuser 对应 uid 为 1001
chmod o+w /uosl
```

配置共享目录

```
vim /etc/exports
/uosl *(rw, anonuid=1001, anongid=1001, sync, no_wdelay)          #所有远程连接用户获得 nfsuser 身份，共享/uosl 目录
/uos2 192. 168. 200. 0/24(ro)
/uos3 192. 168. 200. 2 (rw, no_root_squash, sync, no_wdelay) #不压制 root 权限
```

重启 nfs 相关服务

```
systemctl restart nfs-kernel-server. service
systemctl restart rpcbind
```

第 9 章 SAMBA 服务

9.1 SAMBA 的介绍

Samba 是在 Linux 和 UNIX 系统上实现 SMB 协议的一个免费软件，由服务器及客户端程 序构成。SMB（Server Messages Block,信息服务块）是一种在局域网上共享文件和打印机 的一种通信协议，它为局域网内的不同计算机之间提供文件及打印机等资源的共享服务。SMB 协议是客户机/服务器型协议，客户机通过该协议可以访问服务器上的共享文件系统、打印 机及其他资源。通过设置“NetBIOS over TCP/IP”使得 Samba 不但能与局域网络主机分享 资源，还能与全世界的电脑分享资源。

在早期网络世界当中，档案数据在不同主机之间的传输大多是使用 FTP 这个好用的服务 器软件来进行传送。不过，使用 FTP 传输档案却有个小小的问题，那就您无法直接修改主 机上面的档案数据！也就是说您想要更改 Linux 主机上的某个档案时，必需要由 Server 端 将该档案下载到 Client 端后才能修改，也因此该档案在 Server 与 Client 端都会存在。 这个时候，万一如果有一天您修改了某个档案，却忘记将数据上传回主机，那么等过了一阵 子之后，如何知道那个档案才是最新的？！

既然有这样的问题，可不可以在 Client 端的机器上面直接取用 Server 上面的档案， 如果可以在 Client 端直接进行 Server 端档案的存取，那么在 Client 端就不需要存在该 档案数据，也就是说，只要有 Server 上面的档案资料存在就可以！有没有这样的档案系统 (File System)很高兴的是，NetworkFile System,

NFS 就是这样的档案系统之一！我只 要在 Client 端将 Server 所提供分享的目录挂载进来，那么在 Client 的机器上面就可以 直接取用 Server ±的档案数据，而 FL,该数据就像 Client 端上面的 partition 一般！而 除了可以让 Unix Like 的机器互相分享档案的 NFS 服务器之外。

在微软(Microsoft)上面也有类似的档案系统，那就是 Commonlnternet File System, CIFS 这个咚咚啦！ CIFS 最简单的想法就是目前常见的『网上邻居』。Windows 系统的计算 机可以透过桌面上『网上邻居 J 来分享别人所提供的档案数据。不过，NFS 仅能让 Unix 机 器沟通，CIFS 只能让 Windows 机器沟通。伤脑筋，那么有没有让 Windows 与 Unix-Like 这两个不同的平台相互分享档案数据的档案系统？

1991 年一个名叫 Andrew Tridgwell 的大学生就有这样的困扰，他手上有三部机器，分 别是跑 DOS 的个人计算机、DEC 公司的 Digital Unix 系统以及 Sun 的 Unix 系统。在当 时，DEC 公司有发展出一套称为 PATWORKS 的软件，这套软件可以用来分享 DEC 的 Unix 与个人计算机的 DOS 这两个操作系统的档案数据，可惜让 Tridgwell 觉得较困扰的是,Sun 的 Unix 无法藉由这个软件来达到数据分享的目的。这个时候 Tridgwell 就想说：『咦！ 既然这两部系统可以相互沟通，没道理 Sun 就必需这么苦命吧？可不可以将这两部系统的运作原理找出来，然后让 Sun 这部机器也能够分享档案数据呢？』，为了解决这样的的问 题，这老兄就自行写了个 program 去侦测当 DOS 与 DEC 的 Unix 系统在进行数据分享传 送时所使用到的通讯协议信息，然后将这些重要的信息撷取下来，并口基于上述所找到的通 讯协议而开发出 ServerMessage Block (SMB)这个档案系统，而就是这套 SMB 软件能够让 Unix 与 DOS 互相的分享数据！(注：再次的给他强调一次，在 Unix Like 上面可以分享 档案数据的 file system 是 NFS,那么在 Windows 上面使用的『网络邻居」所使用的档案系统则称为 Common Internet File System, CIFS)

因此 Tridgwell 就去申请了 SMBServer (Server Message Block 的简写)这个名字 来做为他撰写的这个软件的商标，可惜的是，因为 SMB 是没有意义的文字，因此没有办法 达成注册。既然如此的话，那么能不能在字典里面找到相关的字词可以做为商标来注册呢？ 翻了老半天，呵呵！这个 SAMBA 刚好含有 SMB ，又是热情有劲的拉丁舞蹈的名称，不如就 用这个名字来做为商标好了。如此，这成为我们今天所使用的 SAMBA 的名称由来。

uos ----windows 之间共享

UOS 使用 2 个进程

- smb ip 之间的通信用 smb (tcp)

  smb tcp 139 445

- nmb 主机名之间的通信用 nmb (netbios 协议)

  nmb udp 137 138

9.2 SAMBA 配置实战

配置项详解

Samba 的配置文件 /etc/samba/smb. conf

• hosts　　allow = 192.168.200. EXCEPT 192.168.200. 202

hosts　deny = 192.168. 100. 0/24 #主机黑名单

同时出现，　allow 生效

• workgroup = MYGROUP 工作组名

server string = Samba Server Version %v

#对方看到的共享信息，出于防止泄漏版本号的安全考虑，最好改掉

log file = /var/log/samba/log. %m

眦 in 代表客户端 ip,多个客户端连接生成多个以客户端 ip 结尾的日志

max log size = 50　　#日志大小 50K

security = user　　#共享的模式

#share 不用输入用户名/密码，匿名可访问

#user 服务器用口令文件进行验证，客户端连接需要提供用户名和口令

#server 网络中配置专门的服务器认证

#domain 使用微软的 DC 认证

实验-匿名模式

安装 samba

```
#  apt install -y samba
#  systemctl start smbd
#  systemctl status smbd
```

建立共享冃录与权限

```
#  mkdir /uosl
tt mkdir /uos2
tt mkdir /uos3
tt touch /uosl/file{l. . 9)
tt touch /uos2/file{ll.. 19)
tt touch /uos3/file{lll. . 119)
#  chmod o+rwx /uosl #生产中尽量采用 setfacl 赋权更安全
#  chmod o+rwx /uos2
#  chmod o+rwx /uos3
```

配置共享 uosl R 录

```
# vim /etc/samba/smb. conf [uosl]
    comment = uos
```

```
path = /uosl
browseable = yes
public = yes
#writable = yes
```

重启 **smbd**

```
# systemctl restart smbd
```

客户端测试访问

```
#  apt install -y samba-common smbclient
n smbclient -L 192. 168. 200. 201
#  smbclient //192.168. 200. 201/tiosl #无需输入密码即可进入
```

```
smb: \> Is
smb: \> get filel #put 上传
```

注：如需写入需要在 uosl 的配置 I 二打开 writable = yes,并设置共享目录/uosl 有 o+w 权 限

批量下载

```
smb: \>prompt
smb: \>mget file*
```

windows 客户端

设置虚拟机通过虚拟网络与真机连接

```
\\192. 168.200.202
net use * /del 清除缓存
```

实验-用户模式

在 **uosl I：**

```
# useradd -s /sbin/nologin userl
# useradd -s /sbin/nologin user2
# useradd -s /sbin/nologin user3
# smbpasswd -a userl
# smbpasswd -a user2
# smbpasswd -a user3 禁用 samba 用户 disable 允许
          -d          samba 用户 enable 删除
          -e          samba 用户 delete
          -x
```

查看用户列表

```
# pdbedit -L                    #提取/var/lib/samba/private/passdb. tdb 数据库信息，查看用户列表
```

配置

```
# vim /etc/samba/smb. conf security = user # 更改
```

```
path = /uosl
browseable = yes
public = yes
#writable = yes



 [uosl]
```

```
    comment = uosl
    path = /uosl
    public = no
    browseable = yes
    writable = yes

    [uos2J
    comment = uos2
    path = /uos2
    public = no
    browseable = yes
    writable = no
    write list = userl

    [uos3]
    comment = uos3
    path = /uos3
    public = no
    browseable = yes
    writable = no              #默认不可写
    write list = userl           #只有 userl 授权写入
    valid users = userl     #只有 userl 授权登陆
```

重启 SMBD

```
# systemctl restart smbd
```

说明：

- #优先级 valid users>writable>write list
- #writable=yes 表示所有用户都有写的权限
- #write list 生效时必须 writable=no
- #write list 和 valid users 可采用 userl @shichangbu +shichangbu 的格式，中间用 空格分开

在 uos2 上面，测试共享连接

```
n smbclient -L //192. 168. 200. 201

•   smbclient -U userl //192. 168. 200. 201/uosl #输入 userl 密码可进入
```

- 用 userl 登录 share 1-3 均可写入
- 用 user2 登录 shtiosl 可写入，uos2 可登陆不可写入，uos3 不可登录

排错：

如果搭建了 samba 服务，无法写入

- 检测配置文件 writable write list

• 检测文件系统是否有写的权限 ugo setfacl • 防火墙

实验-普通挂载

在 uos2 I：

```
apt -y install cifs-utils
mkdir /mnt/uos3/
mount -t cifs -o username=user3 //192.168. 200. 201/uos3/ /mnt/uos3/            〃输入密码
可挂载
Is -1 /mnt/uos3/
umount /mnt/uos3/
```

vim /etc/fstab

//192.168.200.201/uos3 /mnt/uos3 cifs credentials=/etc/file 0 0

配置验证

# vim /etc/file
user=user3
password=user3

注：或者可以不使用 file 文件做验证，直接把帐号密码写在 fstab 里
//192. 168.200.201/uos3 /mnt/uos3 cifs username=user3, password=user3 0 0 普通挂载 su 到其他用户可以直接使用挂载资源

实验-multiuser 方式挂载一个用户

在 **uosl I:**

```
  pdbedit -L      #查看哪些用户可以挂载                                              —
```

vim /etc/samba/smb. conf

```
    [uosl]
    comment = uosl
    path = /uosl
    public = no
    browseable = yes writable = no write list = userl valid users = userl user2
```

重启 **smbd**

systemctl restart smbd

在 **uos2 I：** useradd userl #本地必须有和服务器对应的用户 useradd user2

useradd user3 在 **root** 用户下执行挂载 mkdir /mnt/uosl
mount -o multiuser,user=userl, sec=ntlmssp //192.168. 200. 201/uosl /mnt/uosl 注：输入密码可挂载，也加入 password=userl 直接挂载

以 userl 身份

```
su - userl
ll /mnt/uosl        #无权限
cifscreds add 192. 168. 200. 201 #输入 uosl _h userl 的密码，从 samba 服务器获取认证
ll /mnt/uosl        #获得 uosl ± userl 的权限，可登陆，可写入
```

以 uer2 身份

```
su - user2
ll /mnt/uosl        #无权限
cifscreds add 192. 168. 200. 201    #输入 uosl _t user2 的密码，从 samba 服务器获取认证
ll /mnt/uosl        #获得 uosl ±user2 的权限，可登陆，不可写入
```

以 user3 身份

```
su - user3
ll /mnt/uosl        #无权限
cifscreds add 192. 168. 200. 201    #输入 uosl _t user3 的密码，从 samba 服务器获取认证
ll /mnt/uosl        #获得 uosl ±user3 的权限，不可登陆
```

注：在 root 用户下 cifscreds clearall 可清除获得的认证，su 到其他用户需重新获取认 证

多用户各自有配置文件的情况，可以建立用户文件，一次挂载多个用户，
vim /uos. txt username=userl password=userl username=user2 password=user2 mount -o multiuser,credentials=/uos. txt, sec=ntlmssp //192.168.200.201/uosl /mnt/uosl

自动挂载
vi /etc/fstab
//192.168.200.201/uosl /mnt/uosl cifs defaults, multiuser, credentials=/uos. txt, sec=ntlmssp 0 0 mount -a

实验-单独用户配置文件

[Golble]

config file = /etc/samba/smb. conf.%U

注：增加%U 代表用户名，以后有对应配置文件的用户将直接跳转读取自己的配置文件

删 J 除前面配置的所有文件夹设置信息，否则所有用户都能读写这些文件夹,造成安全性降低 cp /etc/samba/smb. conf /etc/samba/smb. conf. user3
vim /etc/samba/smb. conf. user3
删除 config file = /etc/samba/smb. conf. %U

[uos3]

comment = uos3 path = /uos3 public = no browseable = yes writable = yes

重启 SMBD

```
# systemctl restart smbd uos2 上

smbclient -U user3 //192. 168. 200. 201/uos3 注：由于 user3 的配置文件中没有 uosl 和 uos2 目
```

vim /etc/samba/smb. conf

录，以 user3 不能登录 uosl 和 uos2

第 10 章 FTP 服务

10.1 FTP 服务概述

FTP 原理

FTP 是 File Transfer Protocol （文件传输协议）的英文简称，用于 Internet ±的控制文 件的双向传输。基于不同的操作系统有不同的 FTP 应用程序，分为服务端和客户端，而所有 这些应用程序都遵守同一种协议以传输文件。在 FTP 的使用当中，用户经常遇到两个概念: “下载〃(Download)和"上传"(Upload)。"下载〃文件就是从远程主机拷贝文件至自己的计 算机上；”上传〃文件就是将文件从自己的计算机中拷贝至远程主机上。

FTP 的工作模式

Port 模式（主动模式）FTP 客户端首先和服务器的 TCP 21 端口建立连接，用来发送命令， 客户端需要接收数据的时候在这个通道上发送 PORT 命令。PORT 命令包含了客户端用什么端 口接收数据。在传送数据的时候，服务器端通过自己的 TCP 20 端口连接至客户端的指定端 口发送数据。FTP server 必须和客户端建立一个新的连接用来传送数据。

Passive 模式（被动模式）建立控制通道和 Standard 模式类似，但建立连接后发送 Pasv 命令。服务器收到 Pasv 命令后，打开一个临时端口（端口号大于 1023 小于 65535）并且通 知客户端在这个端口上传送数据的请求，客户端连接 FTP 服务器此端口，然后 FTP 服务器将 通过这个端口传送数据。

很多防火墙在设置的时候都是不允许接受外部发起的连接的，所以许多位于防火墙后或内 网的 **FTP** 服务器不支持 **PASV** 模式，因为客户端无法穿过防火墙打开 FTP 服务器的高端端口 ； 而许多内网的客户端不能用 PORT 模式登陆 FTP 服务器，因为从服务器的 TCP 20 无法和内部 网络的客户端建立一个新的连接，造成无法工作。

主动 **FTP** 对 **FTP** 服务器的管理有利，但对客户端的管理不利。因为 FTP 服务器企图与客户 端的高位随机端口建立连接，而这个端口很有可能被客户端的防火墙阻塞掉。

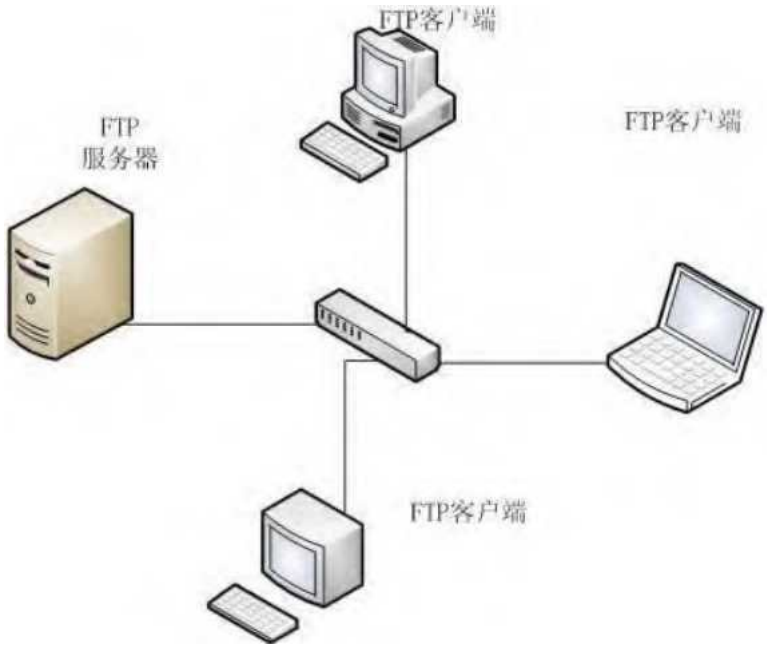
被动 **FTP** 对 **FTP** 客户端的管理有利，但对服务器端的管理不利。因为客户端要与服务器端 建立两个连接，其中一个连到一个高位随机端口，而这个端口很有可能被服务器端的防火墙 阻塞掉。

常用的 FTP 服务器软件介绍

vsftpd 是"very secure FTP daemon"的缩写，是一个完全免费的、开发源代码的 ftp 服务 器软件，安全性是它的一个最大的特点。

vsftpd 可以运行在诸如 Linux、BSD、Solaris、HP-UNIX 等系统上面，支持很多其他的 FTP 服务器所不支持的特征。比如：非常高的安全性需求、带宽限制、良好的可伸缩性、可创建 虚拟用户、支持 IPv6、速率高等。是一款在 Linux 发行版中最受推崇的 FTP 服务器程序。 特点是小巧轻快，安全易用。

在开源操作系统中常用的 FTPD 套件主要还有 ProFTPD、PureFTPd 和 wuftpd 等



安装启动 VSFTP

在 uos server!上

安装

```
apt -y install vsftpd
```

启动

```
systemctl start vsftpd
```

设置开机时启动

```
systemctl enable vsftpd
```

查验端 I I

```
ss -nltp grep 21
```

验证客户端登陆:

使用浏览器打开 ftp://＜serverl IP＞,以验证匿名用户可以登陆

VSFTP 的配置文件详细说明

配置文件：/etc/vsftpd. conf

vsftpd. conf 的形式非常简单，每一行即为一项设定。若是空白行或是开头为#的一行，将 会被忽略。内容的格式只有一种，如下所示

option=value

但要注意：等号两边不能加空白。

默认配置：

- 1＞允许匿名用户和本地用户登陆。

  anonymous_enable=YES local_enable=YES
- 2)匿名用户使用的登陆名为ftp或anonymous，口令为空;匿名用户不能离开匿名用户 家目录/var/ftp，FL 只能下载不能上传。
- 3＞本地用户的登录名为本地用户名，口令为此本地用户的口令；本地用户可以在自己家 目录中进行读写操作；木地用户可以离开自家目录切换至有权限访问的其他目录，并在 权限允许的情况下进行上传/下载。

  write_enable=YES
- 4＞写在文件/etc/vsftpd. ftpusers 中的本地用户禁止登陆。

匿名用户(anonymous)设置选项

- anonymous_enable=YES/NO (YES)
  控制是否允许匿名用户登入，YES 为允许匿名登入，NO 为不允许。默认值为 YES。
- write_enable=YES/NO (YES)
  是否允许登陆用户有写权限。属于全局设置，默认值为 YES。
- no_anon_password=YES/NO (NO)
  若是启动这项功能，则使用匿名登入时，不会询问密码。默认值为 NO。
- ftp_username=ftp
  定义匿名登入的使用者名称。默认值为 ftp。
- anon_root=/var/ftp
  使用匿名登入时，所登入的目录。默认值为/var/ftpo 注意 ftp 目录不能是 777 的权限 属性，即匿名用户的家目录不能有 777 的权限。
- anon_upload_enable=YES/NO (NO)
  如果设为 YES,则允许匿名登入者有上传文件(非目录)的权限，只有在 write_enabl e=YES 时，此项才有效。当然，匿名用户必须要有对上层目录的写入权。默认值为 NO
- anon_wor 1 d_readable_only=YES/NO (YES)
  如果设为 YES,则允许匿名登入者下载可阅读的档案(可以下载到本机阅读，不能直接 在 FTP 服务器中打开阅读)。默认值为 YES。
- anon_mkdir_write_enable=YES/NO (NO)
  如果设为 YES,则允许匿名登入者有新增目录的权限，只有在 write_enable=YES 时，

此项才有效。当然，匿名用户必须要有对上层目录的写入权。默认值为 NO。

- anon_other_write_enable =YES/NO (NO)
  如果设为 YES,则允许匿名登入者更多于上传或者建立目录之外的权限，譬如删除或者 重命名。(如果 anon_upload_enable=NO,则匿名用户不能上传文件，但可以删除或者 重命名己经存在的文件；如果 anon_mkdir_write_enable=NO,则匿名用户不能上传或 者新建文件夹，但可以删除或者重命名己经存在的文件夹。)默认值为 NO。
- chown_uploads=YES/NO (NO)
  设置是否改变匿名用户上传文件(非目录)的属主。默认值为 NO。
- chown_username=username
  设置匿名用户上传文件(非目录)的属主名。建议不要设置为 root。
- anon_umask=077
  设置匿名登入者新增或上传档案时的 innask 值。默认值为 077,则新建档案的对应权限 为 700。
- deny_email_enable=YES/NO (NO)
  若是启动这项功能，则必须提供一个档案/etc/vsftpd/banner_emaIls,内容为 email addresso 若是使用匿名登入，则会要求输入 email address,若输入的 email addres s 在此档案内，则不允许进入。默认值为 NO。
- banned_email_file=/etc/vsftpd/banner_emaIls
  此文件用来输入 email address,只有在 deny_email_enable=YES 时，才会使用到此档 案。若是使用匿名登入，则会要求输入 email address,若输入的 email address 在此 档案内，则不允许进入。

本地用户设置选项

- local_enable=YES/NO (YES)
  控制是否允许本地用户登入，YES 为允许本地用户登入，NO 为不允许。默认值为 YES。
- local_root=/home/username
  当本地用户登入时，将被更换到定义的目录下。默认值为各用户的家目录。
- write_enable=YES/NO (YES)
  是否允许登陆用户有写权限。属于全局设置，默认值为 YES。
- local_umask=022
  本地用户新增档案时的 innask 值。默认值为 077。
- f ile_open_mode=0755
  本地用户上传档案后的档案权限，与 chmod 所使用的数值相同。默认值为 0666。

欢迎语设置

- dirmessage_enable=YES/NO (YES)
  如果启动这个选项，那么使用者第一次进入一个目录时，会检查该目录下是否有.mess age 这个档案，如果有，则会出现此档案的内容，通常这个档案会放置欢迎话语，或是 对该目录的说明。默认值为开启。
- message_file=. message
  设置目录消息文件，可将要显示的信息写入该文件。默认值为.messageo
- banner_file=/etc/vsftpd/banner
  当使用者登入时，会显示此设定所在的档案内容，通常为欢迎话语或是说明。默认值为 无。如果欢迎

信息较多，则使用该配置项。

- ftpd_banner=Welcome to BOB’s FTP server

  这里用来定义欢迎话语的字符串，banner_file 是档案的形式，而 ftpd_banner 则是字 符串的形式。预设为无。

控制用户是否允许切换到上级目录(chroot)

在默认配置下，本地用户登入 FTP 后可以使用 cd 命令切换到其他目录，这样会对系统带来 安全隐患。可以通过以下三条配置文件来控制用户切换目录。

- chroot_list_enable=YES/NO (NO)

  设置是否启用 chroot_list_file 配置项指定的用户列表文件。默认值为 NO。

- chroot_list_file=/etc/vsftpd. chroot_list

  用于指定用户列表文件，该文件用于控制哪些用户可以切换到用户家目录的上级目录。

- chroot_local_user=YES/NO (NO)

  用于指定用户列表文件中的用户是否允许切换到上级目录。默认值为 NO。

- 通过搭配能实现以下几种效果：

  ① 当 chroot_list_enable=YES, chroot_local_user=YES 时，在/etc/vsftpd. chroot_li st 文件中列出的用户，可以切换到其他目录；未在文件中列出的用户，不能切换到其他 目录。

  ② 当 chroot_list_enable=YES, chroot_local_user=NO 时，在/etc/vsftpd. chroot_lis t 文件中列出的用户，不能切换到其他目录；未在文件中列出的用户，可以切换到其他目 录。

  ③ 当 chroot_list_enable=NO» chroot_local_user=YES 时，所有的用户均不能切换到其 他目录。

  ④ 当 chroot_list_enable=NO» chroot_local_user=NO 时，所有的用户均可以切换到其他 目录。

数据传输模式设置

FTP 在传输数据时，可以使用二进制方式，也可以使用 ASCII 模式来上传或下载数据。

- ascii_upload_enable=YES/NO (NO)

  设置是否启用 ASCII 模式上传数据。默认值为 NO。

- ascii_download_enable=YES/NO (NO)

  设置是否启用 ASCII 模式下载数据。默认值为 NO。

访问控制设置

两种控制方式：一种控制主机访问，另一种控制用户访问。

①控制主机访问：

- tcp_wrappers=YES/NO (YES)

  设置 vsftpd 是否与 tcp wrapper 相结合来进行主机的访问控制。默认值为 YES。如果 启用,则 vsftpd 服务器会检查/etc/hosts. allow 和/etc/hosts. deny 中的设置,来决

  定请求连接的主机，是否允许访问该 FTP 服务器。这两个文件可以起到简易的防火墙功 能。

  比如：若要仅允许 192. 168. 0. 1—192. 168. 0. 254 的用户可以连接 FTP 服务器，则在/e tc/hosts. allow 文件中添加以下内容：

```
vsftpd: 192. 168. 0. :allow
all:all :deny
```

②控制用户访问：

对于用户的访问控制可以通过/etc 目录下的 vsftpd. user_list 和 ftpusers 文件来实现。

- userlist_file=/etc/vsftpd.user_list
  控制用户访问 FTP 的文件，里面写着用户名称。一个用户名称一行。
- userlist_enable=YES/NO (NO)
  是否启用 vsftpd. user_list 文件。
- userlist_deny=YES/NO (YES)
  决定 vsftpd. user_list 文件中的用户是否能够访问 FTP 服务器。若设置为 YES,则 vs ftpd. user_list 文件中的用户不允许访问 FTP,若设置为 NO,则只有 vsftpd. user_li st 文件中的用户才能访问 FTPo
- /etc/vsftpd/ftpusers 文件专门用于定义不允许访问 FTP 服务器的用户列表(注意：如 果 userlist_enable=YES, us er l i s t_deny =NO,此时如果在 vsftpd. user_list 和 ftpuse rs 中都有某个用户时,那么这个用户是不能够访问 FTP 的,即 ftpusers 的优先级要高)。默认情况下 vsftpd. user_list 和 ftpusers,这两个文件已经预设置了一些不允许访问 FTP 服务器的系统内部账户。如果系统没有这两个文件，那么新建这两个文件，将用户 添加进去即可。

访问速率设置

- anon_max_rate=0
  设置匿名登入者使用的最大传输速度，单位为 B/s, 0 表示不限制速度。默认值为 0。
- loc al_max_rat e=0
  本地用户使用的最大传输速度，单位为 B/s, 0 表示不限制速度。预设值为 0。

超时时间设置

- accept_timeout=60
  设置建立 FTP 连接的超时时间，单位为秒。默认值为 60。
- connect_t imeout=60
  PORT 方式下建立数据连接的超时时间，单位为秒。默认值为 60。
- data_connection_timeout=120
  设置建立 FTP 数据连接的超时时间，单位为秒。默认值为 120。
- idle_session_timeout=300
  设置多长时间不对 FTP 服务器进行任何操作，则断开该 FTP 连接，单位为秒。默认值为 300 o

日志文件设置

- xferlog_enable= YES/NO (YES)
  是否启用上传/下载日志记录。如果启用，则上传与下载的信息将被完整纪录在 xferlo g_file 所定义的档案中。预设为开启。
- xferlog_file=/var/log/vsftpd. log
  设置日志文件名和路径，默认值为/var/log/vsftpd. logo
- xferlog_std_format=YES/NO (NO)
  如果启用，则日志文件将会写成 xferlog 的标准格式，如同 wu-ftpd 一般。默认值为 关闭。

- log_ftp_protocol=YES|NO (NO)
  如果启用此选项，所有的 FTP 请求和响应都会被记录到日志中，默认日志文件在/var/ log/vsftpd.logo 启用此选项时，xferlog_std_format 不能被激活。这个选项有助于 调试。默认值为 NO。

定义用户配置文件

在 vsftpd 中，可以通过定义用户配置文件来实现不同的用户使用不同的配置。

- user_config_dir=/etc/vsftpd/userconf
  设置用户配置文件所在的目录。当设置了该配置项后，用户登陆服务器后，系统就会到 ,/etc/vsftpd/userconf 目录下，读取与当前用户名相同的文件，并根据文件中的配置 命令，对当前用户进行更进一步的配置。
  例如：定义 user_config_dir=/etc/vsftpd/userconf» II 主机上有使用者 testl, test 2,那么我们就在 user_config_dir 的目录新增文件名为 testl 和 test2 两个文件。若 是 testl 登入，则会读取 user_config_dir 下的 testl 这个档案内的设定。默认值为 无。利用用户配置文件，可以实现对不同用户进行访问速度的控制，在各用户配置文件 中定义 1 ocal_max_rate=XX,即可。

FTP 的工作方式与端 II 设置

FTP 有两种工作方式：PORT FTP (主动模式)和 PASV FTP (被动模式)

- listen_port=21
  设置 FTP 服务器建立连接所监听的端口，默认值为 21。
- conne ct_from_port_2 O=YES/NO
  指定 FTP 使用 20 端口进行数据传输，默认值为 YESo
- f tp_dat a_port =2 0
  设置在 PORT 方式下，FTP 数据连接使用的端口，默认值为 20。
- pasv_enable=YES/NO (YES)
  若设置为 YES,则使用 PASVI 作模式；若设置为 NO,则使用 PORT 模式。默认值为 YES, 即使用 PASV 工作模式。
- pasv_max_port=0
  在 PASV 工作模式下，数据连接可以使用的端口范围的最大端口，0 表示任意端口。默 认值为 0。
- pasv_min_port=0
  在 PASV 工作模式下，数据连接可以使用的端口范围的最小端口，0 表示任意端口。默 认值为 0。

与连接相关的设置

- listen=YES/NO (YES)
  设置 vsftpd 服务器是否以 standalone 模式运行。以 standalone 模式运行是一种较好 的方式，此时 listen 必须设置为 YES,此为默认值。建议不要更改，有很多与服务器 运行相关的配置命令，需要在此模式下才有效。若设置为 NO,则 vsftpd 不是以独立的 服务运行，要受到 xinetd 服务的管控，功能上会受到限制。
- max_clients=0
  设置 vsftpd 允许的最大连接数，默认值为 0,表示不受限制。若设置为 100 时，则同 时允许有 100 个连接，超出的将被拒绝。只有在 standalone 模式运行才有效。
- max_per_ip=0
  设置每个 IP 允许与 FTP 服务器同时建立连接的数目。默认值为 0,表示不受限制。只 有在 standalone 模式运行才有效。

- listen_address=IP 地址
  设置 FTP 服务器在指定的 IP 地址上侦听用户的 FTP 请求。若不设置，则对服务器绑定 的所有 IP 地址进行侦听。只有在 standalone 模式运行才有效。
- setproctitle_enable=YES/NO (NO)
  设置每个与 FTP 服务器的连接，是否以不同的进程表现出来。默认值为 NO,此时使用 p s aux |grep ftp 只会有一个 vsftpd 的进程。若设置为 YES,则每个连接都会有一个 v sftpd 的进程。

虚拟用户设置

- 虚拟用户使用 PAM 认证方式。
- pam_service_name=vsftpd
  设置 PAM 使用的名称，默认值为/etc/pam. d/vsftpdo
- guest_enable= YES/NO (NO)
  启用虚拟用户。默认值为 NO。
- guest_username=ftp
  这里用来映射虚拟用户。默认值为 ftp。
- virtual_use_local_privs=YES/NO (NO)
  当该参数激活(YES)时，虚拟用户使用与本地用户相同的权限。当此参数关闭(NO) 时，虚拟用户使用与匿名用户相同的权限。默认情况下此参数是关闭的(NO) o

其他设置

- text_userdb_names= YES/NO (NO)
  设置在执行 Is -la 之类的命令时，是显示 UID、GID 还是显示出具体的用户名和组名。 默认值为 NO,即以 UID 和 GID 方式显示。若希望显示用户名和组名，则设置为 YES。
- ls_recurse_enable=YES/NO (NO)
  若是启用此功能，则允许登入者使用 Is -R（可以查看当前目录下子目录中的文件）这个指令。默认值为 NO。
- hide_ids=YES/NO (NO)
  如果启用此功能，所有档案的拥有者与群组都为 ftp,也就是使用者登入使用 Is -al 之类的指令，所看到的档案拥有者跟群组均为 ftp。默认值为关闭。
- download_enable=YES/NO (YES)
  如果设置为 NO,所有的文件都不能下载到本地，文件夹不受影响。默认值为 YES。

10.2 FTP 配置与应用

用户访问方式-匿名 FTP 的基本配置

在 uosl I：

安装

```
apt install -y vsftpd
```

修改配置文件允许匿名访问 vim /etc/vsftpd. conf anonymous_enable=YES 重启 vsftpd 服务 systemctl restart vsftpd 创建测试文件 touch /srv/ftp/test, txt 匿名用户登录上来的位置，使用默认位置：/srv/ftp

在 uos2 I：

登录 ftp

ftp 192. 168.200.201

Name (192. 168. 200. 201:root) : ftp

Password: 〃直接回车

ftp\> Is

200 PORT command successful. Consider using PASV.

150 Here comes the directory listing.

-rw-r\—r\— 1 0 0 0 Jun 08 21:42 test, txt #可以看到测试文件 ftp\> get test, txt #下载文件到根下

匿名用户上传

在 uos1 上面

配置

vim /etc/vsftpd. conf

anonymous_enable=YES wr i t e_e na ble=YES anon_upload_enable=YES anon_mkdir_write_enable=YES 重启 vsftpd systemctl restart vsftpd 创建冃录并修改权限 mkdir /srv/ftp/pub #创建上传目录 chmod 777 /srv/ftp/pub #更改权限 chown ftp:ftp /srv/ftp/pub/

在 uos2 I：

创建上传文件

touch 1.sh

登陆 ftp

ftp 192. 168.200.201

Name (192. 168. 200. 201:root) : ftp

Password:

ftp\> cd pub

ftp\> put 1. sh #上传文件

调用 pam 安全模块检测

配置系统本地用户访问

在 uosl I：

创建测试用户

useradd -m -s /bin/bash userl useradd -m -s /bin/bash user2 echo \〃userl:123456\〃 \|chpasswd echo \〃user2:123456、〃 \|chpasswd 修改配置文件以下选项 vim /etc/vsftpd. conf

anonymous_enable=NO

#切记将以下配置文件内容注释掉 #chroot_local_user=YES #chroot_list_enable=YES

#chroot_list_file=/etc/vsftpd/chroot_list 重启

systemctl restart vsftpd

在 UOS2 I：

登录测试

ftp 192.168.200.201

Name (192. 168. 200. 201:root): userl

Password:

ftp\> put 1.sh

ftp\> Is

修改配置文件，使用 pam 认证

vim /etc/vsftpd. conf

pam_service_name=ftp

说明：

- uos 操作系统本地用户需要更改成 ftp
- 虚拟用户需要改成 vsftpd（详见后面章节《配置 FTP 的虚拟用户登录》）

FTP 服务访问控制用户登录

限制用户登录

在 uosl I：

配置

vim /etc/vsftpd. conf

userlist_enable=YES

userlist_deny=NO

#为村 0 时为白名单，即只有/etc/vsftpd. user_list 中的用户可以登录;

#YES 时为黑名单，/etc/vsftpd. user_list 中的用户禁止登录 user2 加入名单并重启

```
echo user2 » /etc/vsftpd. user_list systemctl restart vsftpd
```

#效果是 userl 将不能登录，只有 user2 可以登陆

登录测试

ftp 192. 168.200.201

注：假如换成

```
vim /etc/vsftpd/vsftpd. conf u ser l i s t_deny=YES # 修改 systemctl restart vsftpd #客户端只有 userl 可以登录
```

vim /etc/samba/smb. conf

锁定用户登录目录

在 UOS1 I：

添加木地用户并设置密码

```
useradd uosftpl -g uosftp -d /home/uosftp useradd uosftp2 -g uosftp -d /home/uosftp passwd uosftpl
passwd uosftp2
```

设置家闩录权限

chmod 777 -R /home/uosftp/ftp 修改登陆 shell 为不能登陆 usermod -s /sbin/nologin uosftpl usermod -s /sbin/nologin uosftp2 建立限制名单文件

touch /etc/vsftpd. chroot_list

修改配置

vim /etc/vsftpd. conf chroot_local_user=YES chroot_list_enable=YES chroot_list_file=/etc/vsftpd. chroot_list 将 uosftpl 加入限制列表并重启 echo uosftpl \> /etc/vsftpd. chroot_list systemctl restart vsftpd

登录测试

ftp 192. 168.200.201

使用 uosftp2 登录将被锁定目录，uosftpl 不被锁定

配置 FTP 的虚拟用户登录

虚拟用户

建立虚拟用户对应的系统用户 vmftp useradd -m -s /bin/false vmftp

在/home/vmftp 目 录下建立 3 个 子目录 mkdir vmftp {1.. 3}

创建虚拟账户和密码的文木文件，奇数行为账户名、偶数行为密码

vim /etc/loguser. txt

vmftp1

vmftpl

vmftp2

vmftp2

vmftp3

vmftp3

生成数据库

```
apt install db5. 3-util
db5. 3_load -T -t hash -f /etc/loguser. txt /etc/vsftpd_login. db #生成虚拟用 户数据库
    chmod 600 /etc/vsftpd login.db                                  #增强安全级别
```

修改 pam 认证

vim /etc/pam. d/vsftpd

注释掉第二行：# auth required pam_listfile. so item=user sense=deny fi le=/etc/ftpusers onerr=succeed

增加以下内容：

```
    auth sufficient pam_userdb. so db=/etc/vsftpd_login 后缀                    #注意没有.db
    account sufficient pam_userdb. so db=/etc/vsftpd_login 后缀
                                                                                #注意没有.db
```

编辑 ftp 配置文件

vim /etc/vsftpd. conf

确保含有以下配置，注意 guest 相关配置项后面不能有多余空格，否则容报错 listen=YES

```
#listen_ipv6=YES guest_enable=YES gue st_username=vmftp #指向刚刚创建的系统用户
user_config_dir=/etc/vsftpd_user_conf pam_service_name=vsftpd
local_enable=YES secure chroot dir=/var/run/vsftpd/empty
```

创建虚拟川户的配置文件

```
mkdir /etc/vsftpd_user_conf
```

```
echo "local_root=/home/vmftp/vmftpl" > /etc/vsftpd_user_conf/vmftpl echo "
local_root=/home/vmftp/vmftp2" > /etc/vsftpd_user_conf/vmftp2 echo "local
root=/home/vmftp/vmftp3" > /etc/vsftpd user conf/vmftp3
```

重启

```
systemctl restart vsftpd
```

去掉写权限

```
chmod a~w /home/vmftp chmod a~w /home/vmftp/*
```

登录测试

```
ftp 192. 168.200.201
```

在客户端分别用 3 个账号登录，进入 3 个不同的目录

FTP 服务器的优化设置

一：进程类别优化：

- 1： listen=YES/NO 设置独立进程控制 vsftpd

二：登录和访问控制选项优化：

- 1： anonymous_enable=YES/NO 允许/禁止匿名用户登陆
- 2： banned_email_file=/etc/vsftpd/vsftpd. banned_emails 允许/禁止邮件的使用的 存放路径和目录

配合使用：deny_email_enable=YES/NO 允许/禁止匿名用户使用邮件作为密码

- 3: banner_file=/etc/vsftp/banner_file 即可 在文件 banner_file 添加欢迎词
- 4： cmds_allowed=HELP, DIR, QUIT, ! 列出被允许使用的 FTP 命令
- 5： ftpd_banner=we l come to ftp server 一样 和第三条一样是欢迎词屏幕，方法不
- 6： local_enable=YES/NO 允许/禁止本地用户登陆
- 7: pam_service_name=vsftpd 使用 pam 模块进行 ftp 客户端的验证
- 8： userlist_deny=YES/NO 访问 ftp 服务器 禁止/允许文件列表 user_list 的用户

配合使用：userlist_file=/etc/vsftpd/user_list 用户列表文件

配合使用：us er l i st_enabl e=YES/NO 激活/失效第 8 条的功能

- 9： tcp_wrappers=YES/NO 启用/不启用 tcp_wrappers 控制服务访问的功能

二、 匿名用户选项的优化：

- 1： anon_mkdir_write_enable=YES/NO 允许/禁止匿名用户创建目录、删除文件
- 2： anon_root=/path/to/file 设置匿名用户的根冃录，默认是 /var/ftp/ 你可以修改这个默认路径
- 3： anon_upload_enable=YES/NO 允许/禁止匿名用户上传
- 4: anon_world_readable_only=YES/NO 禁止/允许匿名用户浏览目录和下载
- 5： ftp_username=anonftpuser 把匿名用户绑定到系统用户名
- 6： no_anon_password=YES/NO 不需要/需要匿名用户的登录密码

四、 本地用户选项的优化：

- 1： chmod_enable=YES/NO 允许/禁止本地用户修改文件权限
- 2： chroot_list_enable=YES/NO 启用/不启用禁锢本地用户在家目录
- 3： chroot_list_file=/path/to/file 建立禁锢用户列表文件，一行一个用户
- 4： guest_enable=YES/NO 激活/不激活虚拟用户

- 5： guest_username=系统实体用户 把虚拟用户绑定在某个实体用户上
- 6： local_root=/path/to/file 指定或修改本地用户的根目录
- 7： local_umask=具体权位数字 设置本地用户新建文件的权限
- 8： user_config_dir=/path/to/file 激活虚拟用户的的主配置文件

五：同录选项的优化：

- 1： text_userdb_names=YES/NO 启用/禁用用户的名称取代用户的 UID

六：文件传输选项优化：

- 1： chown_uploads=YES/NO 启用/禁用修改匿名用户上传文件的权限
  配合使用：chown_username=账户 指定匿名用户上传文件的所有者
- 2： write_enable =YES/NO 启用/禁止用户的写权限
- 3： max_clients=数字 设置 FTP 服务器同一时刻最大的连接数
- 4： max_per_ip=数字 设置每 ip 的最大连接数

七：网络选项的优化：

- 1： anon_max_rat e=数字 设置匿名用户最大的下载速率
  设置本地用户最大的下载速率

10.3 项目实战

1. 安装 VSFTP

要求：

- 根据本节所学内容，安装并启动 VSftp
- 使用浏览器连接 VSftp

2. 基于网络存储上搭建 FTP 服务器

要求：

- 准备两台安装好 UOS 系统的虚拟机;
- 在 server1 上配置 NFS 服务，共享/share 目录
- 在 server2 上挂载 server1 的/share 目录到/mnt
- 在 server2 上配置 vsftpd 服务，将挂截的共享目录设置为 ftp 的根目录 上建立用户 uosftp,密码 uos@2020
- 在 server2
  - 在/mnt 目录下建立 sa、sb、sc、sd 四个子目录，设置权限为：
    sa\sb: root:root 750
    sc\sd: uosftp:uosftp 750
- 在 server1 上使用匿名用户和 uosftp 分别登陆 ftp://server2,查验这四个目录的上 传下载权限;

3. 根据企业需求限制不同的用户访问

要求：

- 在 server2 ±建立四个系统用户：uosftpl、uosftp2ˆ uosftp3ˆ uosftp4
- 使用 userlist 白名单方式,限制 uosftpl、uosftp2 可以登陆,而 uosftp3> uosftp4 不能登陆

4. 根据企业需求创建不同的虚拟用户访问 FTP 服务器

要求：

- 在 server2 上配置 4 个虚拟用户,uosl> uos2、uos3> uos4
- 登陆后进入不同的虚拟目录

第 11 章 ISCSI 存储服务

11.1 网络存储结构

网络存储的结构划分

网络存储技术(Network Storage Technologies)网络存储技术是基于数据存储的一种通 用网络术语。

三种传统的存储方式：DAS、SAN、NAS

- 直连式存储(DAS: Direct Attached Storage)
- 网络连接式存储(NAS： Network Attached Storage)
- 存储网络(SAN： Storage Area Network)

三种存储类型：块存储、文件存储、对象存储

块存储和文件存储是我们比较熟悉的两种主流的存储类型，而对象存储(Object-based Storage)是一种新的网络存储架构，基于对象存储技术的设备就是对象存储设备 (Object-based Storage Device)简称 OSD。

分布式存储
分布式存储系统，是将数据分散存储在多台独立的设备上。传统的网络存储系统采用集中的 存储服务器存放所有数据，存储服务器成为系统性能的瓶颈，也是可靠性和安全性的焦点， 不能满足大规模存储应用的需要。分布式网络存储系统采用可扩展的系统结构，利用多台存 储服务器分担存储负荷，利用位置服务器定位存储信息，它不但提高了系统的可靠性、可用 性和存取效率，还易于扩展。

直连式存储 DAS 的概念精讲

DAS (Direct Attach STorage)：
是直接连接于主机服务器的一种储存方式，每一台主机服务器有独立的储存设备，每台主机 服务器的储存设备无法互通，需要跨主机存取资料时，必须经过相对复杂的设定，若主机服 务器分属不同的操作系统，要存取彼此的资料，更是复杂，有些系统甚至不能存取。通常用 在单一网络环境下旦数据交换量不大，性能要求不高的环境下，可以说是一种应用较为早的 技术实现。

优点：价格便宜、速度快
缺点：不易扩展、备份困难、不易共享

网络连接式存储 NAS 的概念精讲

NAS （Network Attached Storage）：

是一套网絡储存设备，通常是直接连在网络上并提供资料存取服务，一套 NAS 储存设备就 如同一个提供数据文件服务的系统，特点是性价比高。例如教育、政府、企业等数据存储应 用。

它采用 NFS 或 CIFS 命令集访问数据，以文件为传输协议，通过 TCP/IP 实现网络化存储，可 扩展性好、价格便宜、用户易管理，如目前在集群计算中应用较多的 NFS 文件系统，但由于 NAS 的协议开销高、带宽低、延迟大，不利于在高性能集群中应用。

典型设备：FTP、NFS 服务器

优点

（1） 造价低：随便一台机器就可以，另外普通的以太网就可以，根本不需要专用的 SAN 网 络，所以造价低

（2） 方便文件共享

缺点

（1）读写速率低，传输速率慢：以太网，上传下载速度较慢，另外所有读写都要 1 台服务 器里面的硬盘来承受，相比起磁盘阵列动不动就十几上百块硬盘同时读写，速率慢了许多。

存储区域网络 SAN 的概念精讲

SAN （Storage Area Network）：

是一种用高速（光纤）网络联接专业主机服务器的一种储存方式，此系统会位于主机群的后 端，它使用高速 I/O 联结方式，如 SCSI, ESCON 及 Fibre- Channelso

一般而言，SAN 应用在对网络速度要求高、对数据的可靠性和安全性要求高、对数据共享的 性能要求高的应用环境中，特点是代价高，性能好。例如电信、银行的大数据量关键应用。 它采用 SCSI 块 I/O 的命令集，通过在磁盘或 FC （Fiber Channel）级的数据访问提供高性 能的随机 I/O 和数据吞吐率，它具有高带宽、低延迟的优势，在高性能计算中占有一席之地，但是由于 SAN 系统的价格较高，且可扩展性较差，已不能满足成千上万个 CPU 规模的系统。

优点：速度快、可靠性高、安全性高

缺点：造价高

11.2 iSCSI 概述与部署

ISCSI 概述 **iSCSI：** (Internet Small Computer System Interface),即 Internet 小型计算机系统接口。

iSCSI 技术是一种由 IBM 公司研究开发的，是一个供硬件设备使用的可以在 IP 协议的上层 运行的 SCSI 指令集，这种指令集合可以实现在 **IP** 网络上运行 **SCSI** 协议，使其能够在诸如 高速千兆以太网上进行路由选择。iSCSI 技术是一种新储存技术，该技术是将现有 SCSI 接 口与以太网络(Ethernet)技术结合，使服务器可与使用 IP 网络的储存装置互相交换资料。

iSCSI 是一种基于 TCP/IP 的协议，用来建立和管理 IP 存储块设备、主机和客户机等之间 的相互连接，并创建存储区域网络(SAN) o SAN 使得 SCSI 协议应用于高速数据传输网络 成为可能，这种传输以数据块级别(block-level)在多个数据存储网络间进行。

ISCSI 属于 SAN 的一种，它走的协议是 bai 通 du 过 IP 网络，将 SCSI 块数据转换成网络封 包。虽然 ISCSI 与 NAS 一样，都是通过 IP 网络来传输数据，但是在传输数据的方式协议上 它还是属于 FC-SANo 一个是文件级别的(NAS)，一个是 block 级别的(iSCSI)。

SCSI 结构基于客户/服务器模式，其通常应用环境是：设备互相靠近，并口这些设备由 SCSI 总线连接。iSCSI 的主要功能是在 TCP/IP 网络上的主机系统(启动器 initiator)和存储 设备(目标器 target)之间进行大量数据的封装和可靠传输过程。此外，iSCSI 提供了在 IP 网络封装 SCSI 命令，FL 运行在 TCP 上。

iSCSI 技术优点和成本优势

早期的企业使用的服务器若有大容星磁盘的需求时，通常是透过 SCSI 来串接 SCSI 磁盘， 因此服务器上面必须要加装 SCSI 卡，而口这个 SCSI 是专属于该服务器的。后来这个外接式 的 SCSI 设备被 SAN 的架构所取代，在 SAN 的标准架构下，虽然有很多的服务器可以对同一 个 SAN 进行存取的动作，不过为了速度需求，通常使用的是光纤通道。但是光纤通道很贵， 不但设备贵，服务器上面也要有光纤卡，很麻烦，所以光纤的 SAN 在中小企业很难普及。

随着网络技术的发展，尤其是以 IP 封包为基础的 LAN 技术已经很成熟，再加上以太网路的 速度越来越快，所以就有厂商将 SAN 的连接方式改为利用 IP 技术来处理。然后再透过一些 标准的设定，最后就得到 Internet SCSI (iSCSI)这个的产生！ iSCSI 主要是透过 TCP/IP 的技术，将储存设备端透过 iSCSI target (iSCSI 目标端)功能，做成可以提供磁盘的服务 器端，再透过 iSCSI initiator (iSCSI 初始化用户)功能，做成能够挂载使用 iSCSI target 的用户端，如此便能透过 iSCSI 设置来进行磁盘的应用了。

硬件成本低：构建 iSCSI 存储网络，除了存储设备外，交换机、线缆、接口卡都是标准的以 太网配件，价格相对来说比较低廉。同时，iSCSI 还可以在现有的网络上直接安装，并不需 要更改企业的网络体系，这样可以最大程度地节约投入。

操作简单，维护方便：对 iSCSI 存储网络的管理，实际上就是对以太网设备的管理，只需花 费少量的资金去培训 iSCSI 存储网络管理员。当 iSCSI 存储网络出现故障时，问题定位及解 决也会因为以太网的普及而变得容易。

扩充性强：对于已经构建的 iSCSI 存储网络来说，增加 iSCSI 存储设备和服务器都将变得简 单且无需改变网络的体系结构。

带宽和性能：iSCSI 存储网絡的访问带宽依赖以太网带宽。随着千兆以太网的普及和万兆以 太网的应用，iSCSI 存储网络会达到甚至超过 FC (FiberChannel,光纤通道)存储网络的带 宽和性能。突破距高限制：iSCSI 存储网络使用的是以太网，因而在服务器和存储设备的空 间布局上的限制就会少了很多，甚至可以跨越地区和国家。

iSCSI 的架构组成部分

SCSI 使用 Client/server 模型。**Initiator** 为应用客户端，而 **Target** 包括设备 服务器端 和队列管理两部分。

ISCSI 的储存设备称为 iSCSITarget (或称 iSCSI Target Device),例如 iSCSI 磁盘阵列 柜、iSCSI 磁带柜等。

ISCSI 卡称为 ISCSI HBA (Host Bus Adapter)。ISCSI 允许使用一般 Ethernet NIC 卡，若 使用一般 GbE 卡，则还需要搭配软件才能让 GbE 卡收发 ISCSI 协议，此软件称为 ISCSI Initiator,事实上 ISCSI HBA 的角色也等同于 ISCSI Initiator。

概念

- IQN： iSCSI 限定名称，全球唯一名称，用于以强制命名格式来识别启动器和目标。
  IQN 格式如下：
  iqn. YYYY-MM. com. reversed, domain[: optionalstring]
  iqn：表示此名称使用域为标识符。
  YYYY-W：表示拥有域名的年月时间。
  com. reversed, domain:拥有此 iSCSI 组织的逆向域名
  :optionalstring：以冒号为前缀的可选字符串，全球唯一，由域所有者分配，其中可 包含冒号为分割符的组织边界
- Backstores：后端存储，支持四种后端存储方式。
- TPG：共享存储组，某个特定 iSCSI 目标的集合，它包含了 acl/lun/portals

- ACL：访问权限控制列表，一种使用节点 IQN（通常是启动器名称）来验证启动器的访问 权限的访问限制
- LUN：逻辑单元号，带有编号的块设备，连接到闩标且通过闩标来使用。可以有一个或 多个 LUN 连接到单个闩标，但通常一个目标提供一个 LUN
- portals 入口：目标或启动器上用于建立的 IP 地址和端口。默认端口： 3260 如图:

发现 ISCSI 存储设备的原理

在介绍设备发现之前，我们先看一下在 Linux 操作系统中 iSCSI 的整体软件架构 c 整个 iSCSI 启动器（客户端）的软件架构图包括用户态和内核态两部分的内容。其中用户态主要负责管 理，包括 iscsiadm 命令行工具和 iscsid 守护进程。而内核态包括如图中绿色的 4 个内核模 块，这些内核模块与 SCSI 子系统结合起来形成了整个功能。

对于一个管理命令，通常是 iscsiadm 命令行通过木地套接字发送给 iscsid 守护进程，而该 守护进程经过处理后通过 netlink 发送到内核模块进行处理。

本文我们需要知道的是在 iSCSI 中启动器和目标器之间是通过 session 维护两者之间的关 系，而每个 session 中可能不止有一条连接。但只要建立连接之后，在 Linux 操作系统中 就可以看到磁盘设备（当然，前提是存储端已经做过相关配置）。对于 session 中的连接数 量，设计为可以在同一 session 中有多条连接的冃的是为了可以实现物理链路的冗余，这样 可以保证某条链路故障的情况下可以通过其它链路继续提供服务。

说『半天，我们当前只需要知道在客户端（启动器）和服务端（目标器）之间建立 session 之后，也就是在目标器登录之后，就可以在启动器端看到磁盘，这个也就是设备发现的过 程。

名词解释：

- 服务端：即 I 涌说到的冃标器、存储端、target
- 客户端：即上面说到的启动器、initaitor〉iscsid

Targetcli 即服务端的命令行管理工具

- targetcli 是 Linux-IO Target 的用户态的管理配置工具。用户可以使用 yum 或 apt-get 直接从各大发行版的官方仓库安装，对于较老的 linux 版本需要自己编译源码安装。
- targetcli 提供一个类似 shell 的界面,各种 Target、TPG、LUN〉backstore 对象则被 组织成目录树的形式，用户可以用 Is、cd 命令来浏览目录树,用 create、delete 命令

来创建和删除各种对象。浏览配置，就是浏览目录树，创建删除各种对象，就是在目录 树的各级节点中创建新的节点。

- 同时 targetcli 还提供了常见的 shell 命令辅助编辑功能，例如 TAB 智能补全、上下键 切换命令历史、Ctrl + R 搜索命令历史。凡是有不熟悉的命令，都可以用 help 命令查 询使用说明。

Iscsiadm 即客户端的管理工具

- iscsiadm 命令允许在主机上管理 iSCSI （Internet SCSI）启动器。
- iscsiadm 作为一组子命令实现，许多子命令都具有其自己的选项，并在其各自的部分 中进行说明。“选项”部分介绍了不与特定子命令关联的选项。

11.3 iSCSI 配置与应用

基于磁盘方式建立 iSCSI 存储

在 **uosl I:**

（1）添加硬盘

新添加一块儿硬盘/dev/sdb

分出 sdbl 分区

- fdisk -l
- fdisk /dev/sdb

n p 1 回车 +10G w

（2）安装

- apt install -y targetcli-fb

（3）快速创建共享设备块

- targetcli #进入交互界面

- 步骤 1：建立一个块存储
  注意：名称 smydisk 可自定义），/dev/sdbl 为上面新建的分区名称。

```
/> /backstores/block create mydisk /dev/sdbl
```

- 步骤 2：配置 ISCSITarget 命名
  注意：命名在同一子网内确保是唯一的，命名格式为：iqn. yyyy-mm.〈主机名反写〉：自 定义名称（自定义名称内不能有下划线）

```
/> /iscsi create iqn. 2020-06. com. uos-uos
```

- 步骤 3：创建 ACL 允许 ISCSI 客户机连接
  注意：iqn. 2020-06. com. uos-client 为客户机 ISCSI 名称。

```
/〉/iscsi/iqn. 2020-06. com, uos-uos/tpgl/acls create iqn. 2020-06. com, uos-client
```

- 步骤 4：创建 lun （target 块设备的逻辑单元）

```
/> /iscsi/iqn. 2020-06. com. uos-uos/tpgl/luns create /backstores/block/mydisk
```

- 步骤 5：保存退出

saveconfig

exit

查看端 I I

```
# netstat  -antup   grep 3260
```

tcp 0 0 0.0.0.0:3260 0. 0. 0. 0:* LISTEN

在uos2 上

步骤 1：安装 open-iscsi

```
#  apt-get install open-iscsi
```

启动 iscsi 服务

systemctl restart iscsi #刷新 iqn 标识

systemctl start iscsid #启动 iscsi 服务

设置开机启动服务

```
#  systemctl enable iscsid
```

步骤 2：配置 ISCSIinitiator 名称

注：此处 InitiatorName 必须与服务端配置的 ACL 允许 ISCSI 客户机连接的名称一致。

vim /etc/iscsi/initiatorname, iscsi

```
#InitiatorName=iqn. 1993-08. org. debian:01:e7ae330e87d
Ini t i at orName=i qn. 2020-06. com. uos-client
```

步骤 3：发现 iSCSI 存储

man iscsiadm #搜索 example 查看示例语法

mode 有：discovery、node、session、iface。一般就用前两个 mode。

discovery：发现某服务器是否有 target 输出，以及输出了哪些 target。发现 target 后会生成 target 数据库 discoverydbo

type：有三种 type(sendtargets, SLP, iSNS)，一般只会用到 sendtargets portal：指定要发现 target 的 IP 和端口，不指定端口时使用默认的 3260

```
#  iscsiadm --mode discoverydb —type sendtargets --portal 192.168.200.201 — discover
```

步骤 4：连接 ISCSI 设备

node：管理跟某 target 的关联关系。在 discovery 发现了 target 后，是否要跟 target 建立关系，是否要删除已有的关系或者解除已有的关系等。删除关联关系不仅会解除关联， 还会删除发现 target 后生成的 discoverydb。

```
#  iscsiadm --mode node —targetname iqn. 2020-04. com. uos-uos --portal 192.
168.200.201:3260
--login
```

查看系统磁盘信息

fdisk -l

步骤 5：设置在开机后自动登录 Target

用 vi 编辑配置文件

```
# vim /etc/iscsi/iscsid. conf
```

```
#node. startup = automatic 去掉前面的注释符# node, startup = manual 在前面加上注释符#
```

步骤 6：设置开机挂载网络磁盘

```
# fdisk /dev/sdb            # 分出 /dev/sdbl
# mkfs. ext4 /dev/sdbl

# mkdir /mnt/mydiskl              #创建挂在目录
# blkid /dev/sdbl                 #查询 uuid
```

```
# vim /etc/fstab
```

```
UUID=5d70249d-e305-4d9e-821e-3378938fad35 /mydiski ext4
    defaults, _netdev 0 0
```

#不加_netdev 开机会卡住，因为当我们挂载网络存储设备时由于/dev/sdb 是一块网络存储 设备，而 iSCSI 协议是基于 TCP/IP 网络传输数据的，因此必须在/etc/fstab 配置文件中添 加上_netdev 参数，表示当系统联网后再进行挂载操作，以免系统开机时间过长或开机失败。 mount -a#测试挂截

reboot #重启测试自动挂载

基于文件方式建立 iSCSI 存储

在 uosl I:

创建 2G 大小的文件

```
# dd if=/dev/zero of=/iscsifile bs=lM count=2048
```

配置 target 共享 LUN 块

```
targetcli          #进入交互界面
```

```
,/backstores/fileio create mydisk2 /iscsifile
iscsi/ create iqn. 2020-04. com. uosfile
iscsi/iqn. 2020-04. com. uosfile/tpgl/acls create iqn. 2020-04. com. uosfile:client
iscsi/iqn. 2020-04. com. uosfile/tpgl/luns create /backstores/fileio/mydisk2 saveconfig
exit
```

在 uos2 I:

#配置 ISCSIinitiator 名称

```
# vim /etc/iscsi/initiatorname, iscsi
```

```
InitiatorName=iqn. 2020-04. com. uos-client
```

```
InitiatorName=iqn. 2020-04. com. uosfile:client
```

vim /etc/iscsi/iscsid.conf

node, session, timeo. replacement_timeout = 5 #修改，链路重试 5 秒不能连接则宣告断开 重启服务

systemctl restart iscsi
systemctl restart iscsid
systemctl enable iscsid

发现与登陆，以建立连接

```
# iscsiadm --mode discoverydb —type sendtargets --portal 192.168.200.201 —discover
# iscsiadm --mode node —name iqn. 2020-06. com. uosfile --portal
192. 168.200.201:3260 —login
```

设置在开机后自动登录 **Target**

用 vi 编辑配置文件
vim /etc/iscsi/iscsid. conf

```
#node. startup = automatic 去掉前面的注释符#

node, startup 二 manual 在前面加上注释符#
```

查询是否连接 I：了 **iscsi** 硬盘

fdisk -l

建立分区和文件系统
fdisk /dev/sdc #分区/dev/sdcl

mkfs. ext4 /dev/sdcl

设置开机挂截

mkdir /mnt/mydisk2

blkid #找出/dev/sdcl 的 UUID

vim /etc/fstab

```
UUID=' c56a71c0-d17c-4ff2-80f6-8be43b1663ea' ' /mnt/mydisk2 ext4 defaults, _netdev 0
0
# mount -a 或 reboot #重启自动挂载
```

查看 **session** 连接是否正常
iscsiadm -m session -P[123] #查看 iscsi 信息，，大写 P, session：会话管理。

iSCSI 存储的访问控制管理

Target 服务端配置：在 uosl I：

配置验证用户名和密码

targetcii />cd /iscsi/iqn. 2020-06. com. uos-uos/tpgl/acls/iqn. 2020-06. com. uos-client />set auth userid=test

```
/>set auth password 二 test
/>cd /
/>ls
/>exit
```

Iscsi 客户端配置：在uos2 h

配置客户端 ACL 名

vi /etc/iscsi/initiatorname. iscsi

InitiatorName=<iqn.... >

注意：这个 iqn 为 target 服务端配置的 ACL,应唯一， ---------- 对应客户端

配置用于登陆服务端的帐号密码

vi /etc/iscsi/iscsid.conf

```
node, session, auth. authmethod = CHAP      #去掉注释
node, session, auth. username #为存储服务端 set auth userid=username 配置的 username,
node, session, auth. password` password #为存储月艮务器立崙 set auth password=password 配置的 passwordo
```

11.4 项目实战

1.根据企业需求完成 iSCSI 存储的部署

要求：

- 在 serverl 添加 3 块硬盘，分别为 sdc/sdd/sde,容量 5G

在 serverl 安装 targetcli 服务端，配置三个 lun,分别对应添加的三块硬盘 在 serverl 为三个 lun 分别配置访问用户 testl/test2/test3，密码 pwdl/pwd2/pwd3

在 server2 安装 initator 在客户端，配置挂载 testl 客

server3 安装 initator 在户端，配置挂载 test2 客户

server4 安装 initator 端,配置挂载 test3

2.根据需求完成 iSCSI 存储设备的扩容

Iscsi 块存储的扩容，要求搭建基础的磁盘是支持在线扩容的，比如 LVM 磁盘，步骤如下:

- 先对基础 LVM 磁盘扩容
- 在 target 服务端重建 lun
- 在 initator 客户端重新发现磁盘

第 12 章数据库服务

12. 1 数据库原理概述

数据库原理概述

什么是数据库

- 数据库是“按照数据结构来组织、存储和管理数据的仓库”。是一个长期存储在计算机 内的、有组织的、可共享的、统一管理的大量数据的集合。
- 数据库是以一定方式储存在一起、能与多个用户共享、具有尽可能小的冗余度、与应用 程序彼此独立的数据集合，可视为电子化］的文件柜一一存储电子文件的处所，用户可 以对文件中的数据进行新增、查询、更新、删除等操作

为什么需要数据库
•文件的安全性问题
•文件不利于查询和对数据的管理
•文件不利于存放海量数据
•文件在程序中控制不方便

数据库作用：业务数据存储和业务逻辑运算

数据库存储结构：
（1） 物理存储结构：数据文件、控制文件、重做日志文件
（2） 逻辑存储结构：表空间、段、区、数据块

数据库模式：
（1）关系型数据库
⑵非关系型数据库

数据库的演进方向：分布式、开源，从单机数据库、集群数据库到如今云分布数 据库、AI-Native 数据库

常用数据库
- Oracle
- SQL Server
- mysql
- DB2
- postgresql

国内数据库厂商分类：
(1) 传统数据库：武汉达梦、南大通用、人大金仓、神州通用

(2) 云数据库：阿里 Ocean Base、腾讯云、百度云、华为 Gauss DB

(3) 开源数据库：瀚高科技、优炫软件、巨杉数据库、星环科技

MariaDB 基础介绍

mariadb 数据库简介

- 自甲骨文公司收购 MySQL 后，其在商业数据库与开源数据库领域市场的占有份额都跃居 第一，这样的格局引起了业内很多的人士的担忧，因为商业数据库的老大有可能将 MySQL 闭源。
- 为了避免 Oracle 将 MySQL 闭源，而无开源的类 MySQL 数据库可用，MySQL 社区采用分 支的方式来避开这个风险。MariaDB 数据库就这样诞生了，MariaDB 是一个向后兼容， 可能在以后替代 MySQL 的数据库产品，其官方地址为：https://mariadb.org/。
- mariadb 和 mysql 几乎是一样的。首先，mariadb 就是由 mysql 的创始人负责维护的。 而 mariadb 就是 mysql 创始人女儿的名字。使用方法和 mysql 相同

mariadb 与 **mysql** 的区别：

- MariaDB 不仅仅是 Mysql 的一个替代品，MariaDB 包括的一些新特性使它优于 MySQL
- MariaDB 跟 MySQL 在绝大多数方面是兼容的，对于开发者来说，几乎感觉不到任何不同。 目前 MariaDB 是发展最快的 MySQL 分支版本，新版本发布速度已经超过了 Oracle 官方 的 MySQL 版本。

  MariaDB 是一个采用 Aria 存储引擎的 MySQL 分支版本，这个项目的更多的代码都改编 于 MySQL 6. 0
- 通过全面测试发现，MariaDB 的查询效率提升了 3%-15%,平均提升了 8%,而口没有任 何异常发生；以 qp 为单位，吞吐量提升了 2%-10%

MariaDB 虽然被视为 MySQL 数据库的替代品，但它在扩展功能、存储引擎以及一些新的功能 改进方面都强过 MySQLo 而口从 MySQL 迁移到 MariaDB 也是非常简单的：

- 1、数据和表定义文件(.frm)是二进制兼容的
- 2、所有客户端 API、协议和结构都是完全一致的
- 3、所有文件名、二进制、路径、端口等都是一致的
- 4、所有的 MySQL 连接器，比如 PHP、Perl、Python> Java、.NET、MyODBC、Ruby 以及 MySQL C connector 等在 MariaDB 中都保持不变
- 5、mysql-client 包在 MariaDB 服务器中也能够正常运行
- 6、共享的客户端库与 MySQL 也是二进制兼容的

安装 **mariadb** 数据库

```
apt install mariadb-server mariadb-client #安装 mariadb 服务端系呈序 systemctl restart mariadb.
service                                   #启动程庁
systemctl enable mariadb. service #设为开机自启动
ss -nltp grep 3306                #mysql 服务默认的端口号是 tcp 3306
```

MariaDB 高级应用介绍

Mariadb 高级应用范例如

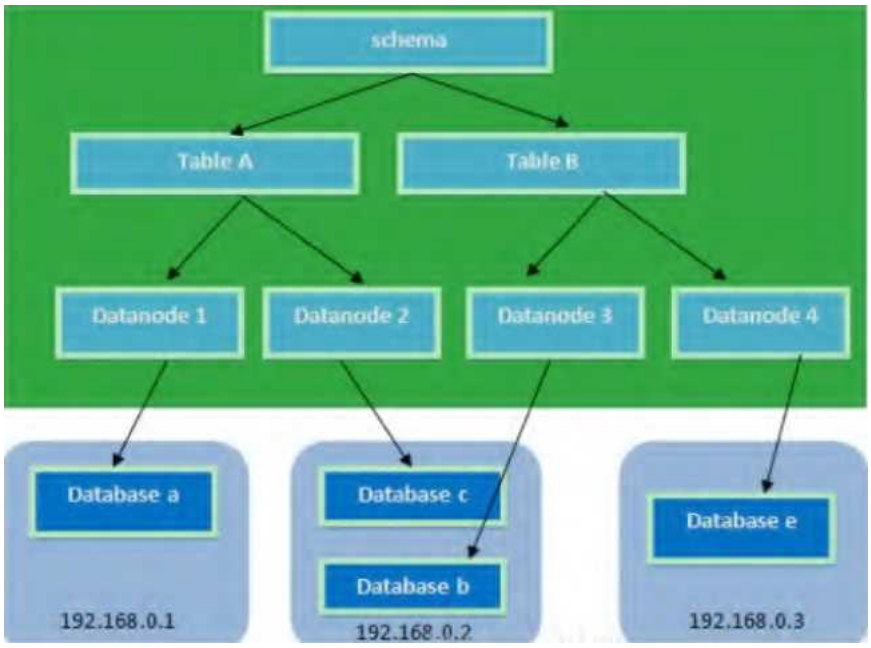
- 主从复制
- 读写分离
- 数据分片

•集群应用

Mycat 是一个新颖的数据库中间件产品，相当于在 mysql 集群前加一个前端路由，实现数据 库的分片存储技术，使 mysql 成为一个分布式的集群数据库，满足数据库数据大量存储，提 高了查询性能

数据库分片指：

通过某种特定的条件，将我们存放在一个数据库中的数据分散存放在不同的多个数据库（主 机）中，这样来达到分散单台设备的负载，根据切片规则，可分为以下两种切片模式



12.2 MariaDB 基础应用

Mariadb 数据库连接方式及初始化

连接数据库

Mysql 命令行工具是 MySQL 官方提供的连接工具，用户可以通过 mysql 连接到 mysqld 士进 行一系列的 SQL 操作。

mysql I 具有两种模式：交互模式和命令行模式。交互模式指令需要连接到 mysql 服务器下 达，命令行模式通过特定(-e)参数读取 shell 命令行传递的指令到服务器。

命令格式：

```
mysql [OPTIONS] [database]
        -h, 一 help                    显示帮助信息
        -e, 一一 execute=name          执行指定命令，命令行模式
        -u, ——username                 指定连接的用户
        -h, --host=name                指定主机名
        _P, --password                 指定密码
        _P, ——port                     指定端口
```

连接数据库示例：

(1)使用 root 账号连接服务器，无密码登录

```
  mysql -u root (2)使用 root 账号连接服务器，使用密码登录，-h 指定登陆主机的 IP 或名字，注意未指 定-h 选项时，默认登陆本机的数据库

  mysql -u root -p’ passwd’ -h <ip>
```

一些常用的查询命令示例:

```
select database ();          #查看当前进入的数据库
select user ();              #查看当前登录用户
show databases;          井查看数据库
use mysql                #进入 mysql 数据库
show tables:                 #查看此数据库的表，test 没表，可以看其他数据库的
desc user;               井查看表结果
[show variables like ’ innodb%’ ; #查看环境变量 有时命令输错停在〉界面出不来，一般是少些了’或者；补全即可退出，实在不行可以 CTRL+D 在重新登录
```

重新初始化 mysql

Mariadb 在安装完毕后，就自动初始化好了基础的数据库，目录位于：/var/lib/mysql Mysql 的配置文件：/etc/mariadb. cnf,为兼容 mysql,原来的配置文件 my. cnf 是它的一 个软链接

我们也可以通过命令 mysql_secure_installation,重新初始化 mysql: mysql_secure_installation
Enter current password for root (enter for none) : #输入当前 root 的密码，密码为空 直接回车即可
Change the root password? [Y/n] #是否修改 root 的密码
Remove anonymous users? [Y/n] #是否删除匿名用户
Disallow root login remotely? [Y/n] #是否不允许 root 远程登录
Remove test database and access to it? [Y/n] #是否删除测试数据库 test
Reload privilege tables now? [Y/n] #是否重新加载授权信息

重启

```
systemctl restart mariadb
```

Mariadb 数据库权限管理

权限设置中，登陆位置的配置说明

mariadb 的用户登录限制包括用户名和登录位置两部分。早期版本%就可以代表所有的连接， 后期版本 localhost 表示本地登录，%表示 tcp/ip 的远程登录的所有 ip。%也可以写具体主 机或网段，比如 192.168. 200.10> 192. 168. 100.%或者 192.168. 200. 0/255. 255. 255. 0,其 他格式不识别。
如果登录主机符合多个连接条件，默认连接第一个身份，设置权限的时候需要注意

建立新用户、设定密码与登陆位置示例：

在 mysql 的命令行状态下：

- select user () ; #查看当前登录用户
- use mysql
  select host, user, password from user; #查看数据库所有登录范围、用户和密码，4 个 root 不是同一个用户，可以单独设置密码
- create user uos@'localhost' identified by '123456' ; #建立可以本地登录的用户

uos 并设置密码 123456

- create user uos@'%' identified by 123456' ; #建立可以远程登录的用户 uos 并设置密码 123456
- create user test@'localhost' identified by '123456' ; #建立可以本地登录的用户 test 并设置密码 123456
- create user test 伊 192.168.200.10' identified by '123456'; # 建立 可以从 192. 168. 200. 10 主机登录的用户 test 并设置密码 123456
- grant all on *. * to uos@' localhost' ; #授权本地登录的用户 uos 对所有数据库的所有表有所有权限
- grant select on scott. * to test@'localhost' ; # 授权本地登录的用户 test 可以读取 scott 数据库的所有表
- grant all on *. * to uostest@'%' identified by ' 123456' ; #可以建立用户、授权、设置密码一起做
- flush privileges; #刷新授权表
- show privileges; #查看所有权限
- show grants for uos@' localhost' ; #查看用户权限
- show grants for test@' localhost' ;
- revoke select on scott. * from test@'192.168. 200. 10' ; #收回权限

修改新密码（需输入 root 原始密码） • 使用 mysqladmin 修改

```
mysqladmin -u root -p password '123456'
```

• 或进入数据库，在 mysql 命令行中修改

mysql -uroot -p

```
set password 二 password (' uos');    #对当前用户设置密码，立刻生效

set password for uos@Tocalhost'=password('123456') ; #对任意用户设置密码

use mysql                                  #使用 mysql 这个内置数据库
update user set password=password(' uos[5]) where user='root' and host- localhost[5] ;
    #修改密码后需要 flush privileges;刷新权限表，或重启服务 mariadb 忘记 root 密码怎么办?
```

- vim /etc/mysql/mariadb. conf, d/50-server. cnf
  [mysqld]
  skip-grant-tables #在[mysqld]下添加此字段，进入但用户模式
- systemctl restart mariadb. service
- 输入 mysql 命令无需密码直接进入数据库

```
use mysql
```

```
update user set password=password('Lios') where user='root* :
```

从 5. 5. 7 版本中 mysql 数据库中就开始引入 plugin 这项配置，用来进行用户密码验证 update user set plugin='' where user='root* ;

- 停止 mariadb 服务，删除配置文件的 skip-grant-tables 选项，重启 mariadb 服务，使 用新密码进入

数据库字符集修改

查看 Linux 的字符集用 locale 命令

注：客户端的字符集要和数据库的字符集一致，不一致有可能乱码

进入 mysql 命令行， status 查看 mariadb 属性

```
status
Server  characterset:       utf8mb4
Db      characterset:       utf8mb4
Client  characterset:       Utf8mb4
Conn.   characterset:       Utf8mb4
```

设置 mariadb server 的默认字符集：

vim /etc/mysql/mariadb. conf, d/50-server. cnf character-set-server = utf8

#collation-server = utf8

重启 mariadb systemctl restart mariadb.service

再验证字符集

mysql -uroot -p

```
MariaDB [(none)]〉status          #重启后变更为新字符集
Server characterset: utf8
```

查看数据库的字符集

```
create database scott; drop database
scott: source /scott. sql show create      #删除数据库
database scott; show create table emp;  #使用脚本重建数据库
                                           #在更改字符集之后建立，所以是字符集是 utf8
```

更改 scott 数据库和表的字符集

```
alter database scott charset utf8: alter table emp charset utf8;
```

数据库 DDL, DML 和 DCL 语言操作

SQL 语言可以理解成向服务端提问的语言，SQL 语言按照功能可以分为几种子语言：

- 数据定义语言(DDL):创建和管理数据库，包括创建修改表，定义索引，管理约束条 件 o DDL 操作对象为数据库内部的对象。
- 数据操纵语言(DML):查询和更新数据库中的数据，用于添加，删除，更新，查询。DML 的操作对象是表的内部数据。而不会涉及到表的定义，结构的修改。

- 数据控制语言(DCL):对用户的权限控制

ru 建立 scott 数据库快速建立脚本

SCOTT 是数据库内部的一个示例用户，缺省口令为 tiger,下面有表 emp, dept 等，这些表 和表间的关系演示了关系型数据库的一些基本原理

vim /scott. sql

```
create database scott;

use scott
create table dept (
    -部门编号
    deptno int unsigned auto_increment primary key,
    —部门名称
    dname        v ar char (15)     ,
    -部门所在位置
    loc          varchar(50)
)engine = InnoDB;

create table emp(
    -雇员编号
    empno              int unsigned auto_increment primary key,
    -雇员姓名
    ename              varchar (15)       ,
    -雇员职位
    job                varchar(10)        ,
    -雇员对应的领导的编号
    mgr                int unsigned       ,
    -雇员的雇佣日期
    hiredate           date               ,
    -雇员的基木工资
    sal                decimal (7,2)      ,
    --奖金
    comm               decimal(7,2)       ,
    -所在部门
    deptno             int unsigned       ,
    foreign key (deptno) references dept(deptno)
)engine = innodb;

create table salgrade (
    一工资等级
```

```
    grade int unsigned -此等级的最低工资 losal     int unsigned
    -此等级的最高工资 hisal int unsigned
)engine=innodb;


INSERT INTO dept VALUES (10,' ACCOUNTING',' NEW YORK');
INSERT INTO dept VALUES (20,' RESEARCH',' DALLAS');
INSERT INTO dept VALUES (30,' SALES',' CHICAGO');
INSERT INTO dept VALUES (40,' OPERATIONS', ' BOSTON');

INSERT INTO emp VALUES (7369,' SMITH' ,' CLERK', 7902, ' 1980T2T7', 800, NULL, 20);
INSERT INTO emp VALUES (7499,' ALLEN',' SALESMAN', 7698,' 1981-2-20 ', 1600, 300, 30)
INSERT INTO emp VALUES (7521, 'WARD',' SALESMAN', 7698, ' 1981-2-22', 1250, 500, 30);
INSERT INTO emp VALUES (7566, 'JONES',' MANAGER', 7839,' 1981-4-2', 2975, NULL, 20);
INSERT INTO emp VALUES (7654,' MARTIN',' SALESMAN', 7698,' 1981-9-28 ', 1250, 1400, 30);
INSERT INTO emp VALUES (7698,' BLAKE', ' MANAGER', 7839, ' 1981-5T', 2850, NULL, 30);
INSERT INTO emp VALUES (7782,' CLARK', ' MANAGER', 7839, ' 1981-6-9', 2450, NULL,
10);
INSERT INTO emp VALUES (7788,' SCOTT',' ANALYST', 7566,' 87-7-13', 3000, NULL, 20);
INSERT INTO emp VALUES (7839,' KING*,' PRESIDENT', NULL,' 1981T1T7', 5000, NULL, 10);
INSERT INTO emp VALUES (7844,' TURNER', ' SALESMAN' , 7698, ' 1981-9-8', 1500, 0,
30);
INSERT INTO emp VALUES (7876,' ADAMS',' CLERK', 7788, ' 87-7-13', 1100, NULL, 20);
INSERT INTO emp VALUES (7900, 'JAMES',' CLERK', 7698,' 1981-12-3 ', 950, NULL, 30);
INSERT INTO emp VALUES (7902,' FORD',' ANALYST', 7566,' 1981T2-3', 3000, NULL, 20);

INSERT INTO salgrade VALUES (1, 700, 1200);
INSERT INTO salgrade VALUES (2, 1201, 1400);
INSERT INTO salgrade VALUES (3, 1401,2000);
INSERT INTO salgrade VALUES (4, 2001,3000);
INSERT INTO salgrade VALUES (5, 3001,9999);
create table bonus(
-雇员姓名
ename        varchar(10),
一雇员职位
job          varchar (9),
一雇员工资
sal          decimal (7, 2),
一雇员资金
comm         decimal (7, 2)
)engine=innodb;
```

调用 sql 脚本

```
mysql -uroot -p system Is -1 /root source /scott. sql
```

F2J 查数据

```
select * from emp;
select ename, sal from emp;
select ename, (sal+200)*3 as bonus from emp;                    #支持算数表达式加减乘数和括号等
select ename, sal, comm, sal+ifnull (comm, 0) as income from emp;          #空值参与算术运算为空值，建议用 ifntill 函数转换
select distinct deptno from emp;                                #用 distinct 去除重复结果
select ename, sal, deptno from emp where deptno=30;
select * from emp where ename like '_______ TT';                #_代表任意单个字符 眦代表任意字符
select * from emp where ename like '%LL%';
select ename, sal, deptno from emp where deptno=30 and sal>2000; #与 select ename, sal, deptno from emp where deptno=30 or sal>2000; #或 select ename, sal, deptno from emp where not sal>2000; #非
select * from emp order by sal;                                 #默认升序，降序加 desc
select empno, ename, sal, deptno from emp order by deptno, sal de sc:       #多列排序
select cone at (ename,' V s sal is *, sal) from emp;            #支持多种函数，不一一举 r
select count(*) from emp;
select deptno,sum(sal), min(sal),max(sal), avg(sal) from emp group by deptno; select deptno,avg(sal) from emp group by deptno having avg(sal)<2000;
select ename, dname from emp, dept:                             #笛卡尔乘积
select ename, dname from emp, dept where emp. deptno=dept. deptno;
select ename, sal, grade from emp join salgrade on sal between losal and hisal; select y. ename yuangong, j. ename jingli from emp y,emp j where y.mgr=j. empno; select ename, dname, sal, grade from emp, dept, salgrade where emp. deptno=dept. deptno and emp. sal between salgrade. losal and salgrade.hisal:
select ename, sal from emp where sal=(select max (sal) from emp):   #子查询找到工资最高薪
select ename from emp where empno not in (select mgr from emp where mgr is not null);
    #in 结果集不能有空值，否则结果为空
```

F3J 创建表

创建数据库

```
MariaDB [ (none)]> create database uosdata;
```

创建表并建立字段

格式：create table 数据表名(字段名字段类型)；

```
MariaDB [uos]> create table uostable(id int, name varchar(10), mail varchar(30));
```

- int 类型表示正常大小的整数（数字数据类型）。
- char 类型（字符串数据类型）表示包含指定长度的空格的右侧带有固定长度的字符串。 M 表示字符的列长度，取值范围为 0-255,缺省值为 1。
- varchar 类型（字符串数据类型）表示一个可变长度字符串，M 范围（最大列长度）为 0 到 65535 o

「4」修改表 create table uosl like uostable; #没有数据，只是复制了表的结构 insert into uosl select * from uostable; #复制数据，表不存在无法复制

```
create table uos2 as select * from uostable; #新建表，并复制整个表结构+数据
```

查看表结构

格式：desc 数据表名;

```
MariaDB [uos]> desc uos:
```

向数据表插入数据

格式：insert into 数据表名(id, name, mail) values (1,' uosl'uosl\r@uos. com');

insert into uostable(id, name, mail) values(1,' uosl'uosl@uos. com');

insert into uostable values (2,' uos2',' uos2 旧 uos. com'); #加入全列数据可以不写列名

insert into uostable(id,name) values(3, 'uos3'); **insert into uostable values (4,' uos4',' uos4@uos. com'), (5,' uos5',' uos5@uos. com'), (6,' uos6',' uos6@uos. com')** #不是全列必须写列名

更新表

```
update uostable set name='test' where id=2;
```

修改表结构

```
alter table uostable add newlist varchar(20):          #增加 newlist 列
alter table uostable drop newlist;                     #删除 newlist 列
alter table uostable add firstlist varchar(20) first; 第一列   #增加 firstlist 列到
alter table uostable add afterid varchar(30) after id; 列后面   #增加 afterid 列到 id
```

F5J 删除表

```
delete from uostable where id=4; #要加上 where 约束,没有 where 删除整个表 delete from uosl;
#DML 操作，清除表数据，保留表结果
truncate uos2; #DDL 操作，清除表数据，保留表结果，更彻底，降低高水位线 drop table uosl; #清除表结构和数据
```

F6J 外部表导入导出

建立测试用外部表（即以逗号分隔各列的 CSV 格式） vim /uos. txt

1, uosl, uosl@uos. com

2, uos2, uos2@uos. com

3, uos3, uos3@uos. com

4, uos4, uos4@uos. com

将外部表导入数据库

```
create database uosdatabase:
use uosdatabase
create table uostable(id int(4), name varchar(10), email varchar(20)):
```

```
load data infile '/uos. txt' into table uostable fields terminated by ',' lines terminated by
' \n ;
```

将数据库导出为外部表

```
select * from uostable into outfile '/var/lib/mysql/uosdatabase/uostable. txt' fields
terminated by ',' lines terminated by ' \n';
```

通过外部表导入 scott 数据库

准备数据：分别建立 scott 数据库的 3 个 CSV 表文件 vim /scott. emp. txt

7369,' SMITH',' CLERK', 7902, 800, 200, 20

7499,' ALLEN',' SALES', 7698,1600, 300, 30

7521,' WARD', ' SALES', 7698, 1250, 500, 30

7566, 'JONES',' MANAG', 7839, 2975, 100, 20 vim /scott. dept, txt

10, ' ACCOUNTING',' NEW YORK'

20, ' RESEARCH',' DALLAS'

30,' SALES',' CHICAGO'

40, 'OPERATIONS', ' BOSTON' vim /scott. salgrade. txt

1,700,1200

2, 1201, 1400

3, 1401, 2000

建立 scott 库和二个表，并导入数据

```
create database scott;
use scott
create table emp (empno int (4), ename v ar char (10), job varchar (9), mgr int (4)default
null, sal int(7), comm int(7) default null, deptno int(2)):
load data infile '/scott. emp. txt' into table emp fields terminated by ',' lines terminated
by ' \n ;
create table dept(deptno int(2), dname varchar(14), loc varchar(13));
load data infile '/scott.dept, txt' into table dept fields terminated by ' ,' lines
terminated by ' \n ;
create table salgrade (grade int,losal int,hisal int);
load data infile ' /scott. salgrade. txt' into table salgrade fields terminated by ',' lines
terminated by ' \n ;
```

将 scott 数据库导出为外部表

```
use scott
select * from emp into outfile '/var/lib/mysql/scott/scott.emp.txt' fields terminated by ','
lines terminated by ' \n';
```

数据库管理程序 mysqladmin 详解

mysqladmin 是一个执行 mysqld 管理操作的客户端程序。它可以用来检查服务器的配置和当 前状态、创建和删除数据库等。

因此使用 mysqladmin 前就确保所连接的 mysqld 服务端进程正常运行和连接

mysqladmin 工具的使用格式：

mysqladmin [option] command [command option] command .....

option 选项：

- -c number 自动运行次数统计，必须和-i 一起使用
- -i number 间隔多长时间重复执行

示例：每隔两秒查看一次服务器的状态，总共重复 5 次。

mysqladmin -uroot -p -i 2 -c 5 status

- -h, --host=name Connect to host.连接的主机名或 iP
- -p, —password[=name]登录密码，如果不写于参数后，则会提示输入
- -P, --port=# Port number to use for connection.指定数据库端口
- -s, --silent Silently exit if one can't connect to server.
- -S, --socket=name Socket file to use for connection. 指定 socket file
- -i, --sleeps# Execute commands again and again with a sleep between. 间隔一 段时间执行一次
- -u, 一use rename User for login if not current user.登录数据库用户名
- -v, —verbose Write more information.写更多的信息
- -V, —version Output version information and exit. 显示版本

mysqladmin 的相关命令示例:

mysqladmin password uosl23 mysqladmin -uroot -puosl23 password uos mysqladmin -uroot -puosl23 status mysqladmin -uroot -puosl23 -i 1 status	#〈=设置密码，前文用过的。 商=修改密码，前文用过的。 #〈二=查看状态，相当于 show statuso #<==每秒查看一次状态。
mysqladmin -uroot -puosl23 extended-status	#<=等同 show global status； o
mysqladmin -uroot -puosl23 flush-logs mysqladmin -uroot -puosl23 processlist mysqladmin -uroot -puosl23 processlist -i mysqladmin -uroot -p’uos’ shutdown mysqladmin -uroot -p’uos’ variables	#<=切割日志。 飯=查看执行的 SQL 语句信息。 1 飯=每秒查看一次执行的 SQL 语句。 筋=关闭 mysql 服务，前文用过的。#<=相当于 show variableso

Mariadb 数据库备份与还原

在 linux 环境备份数据库

```
mysqldump  -u root -p scott    > /scott. dump                          #备份数据库
mysqldump  -u root -p scott    emp > /scott. emp. dump                      #备份数据库中的表
mysqldump  -u root -p scott    dept salgrade > /scott. dept+salgrade. dump          #备份数据库中的多个表
```

在 linux 环境还原数据库

```
>drop database scott;
#如果报错，删除/var/lib/mysql/scott/scott. emp. txt 文件
>create database uos;                                  #不建立数据库，无法导入整库，随意命名
mysql -u root -p uos < /scott. dump                        #还原数据库
>drop database uos;
>create database uos;                                  #重建数据库用于测试还原表
mysql -u root -p uos < /scott. emp. dump                        #还原数据库中的表
mysql -u root -p uos < /scott. dept+salgrade. dump                    #还原数据库中的多个表
```

Mariadb 数据库索引

什么是索引

索引是一种特殊的文件（innoDB 数据表上的索引是表空间的一个组成部分），它们包含着对 数据表里所有记录的引用指针。

更通俗的说，数据库索引好比是一本书前面的目录，在查找内容之前可以先在目录中查找索 引位置，以此快速定位查询数据。

对于索引，会保存在额外的文件中。

索引的类型

1、 普通索引
2、 唯一性索引
3、 主键索引（主索引）
4、 复合索引
5、 全文索引

创建普通索引

创建表时添加索引

语法：

```
create table 表名（
    列定义
    index 索引名称（字段）
    index 索引名称（字段）
）
```

注：可以使用 key 关键字，也可以使用 index 关键字。索引名称，可以加也可以不加，不加 使用字段名作为索引名。

```
MariaDB [book]> desc books;
```

示例：

```
MariaDB [book]> create table demo(id int(4), name varchar(20), pw( varchar(20), index(pwd));
Query OK, 0 rows affected (0. 08 sec)

MariaDB [book]> create table demo1(id int(4), name varchar(20), pw( varchar(20), key(pwd));
Query OK, 0 rows affected (0. 01 sec)

MariaDB [book]> create table demo2(id int(4), name varchar(20), pwd varchar(20), ke：
index_pwd(pwd)):          #为索引加上名称
Query OK, 0 rows affected (0. 02 sec)
```

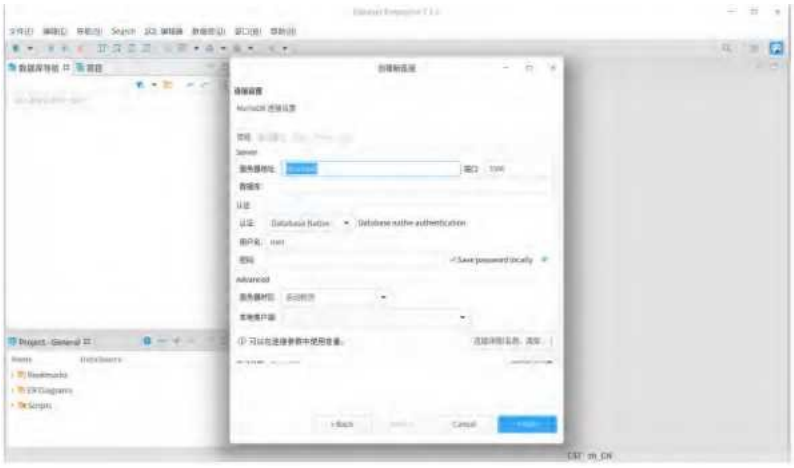
数据库图形化管理工具

在统信 UOS 的应用商店里搜索 mysql,可以找到两款 mysql 数据库的工具：

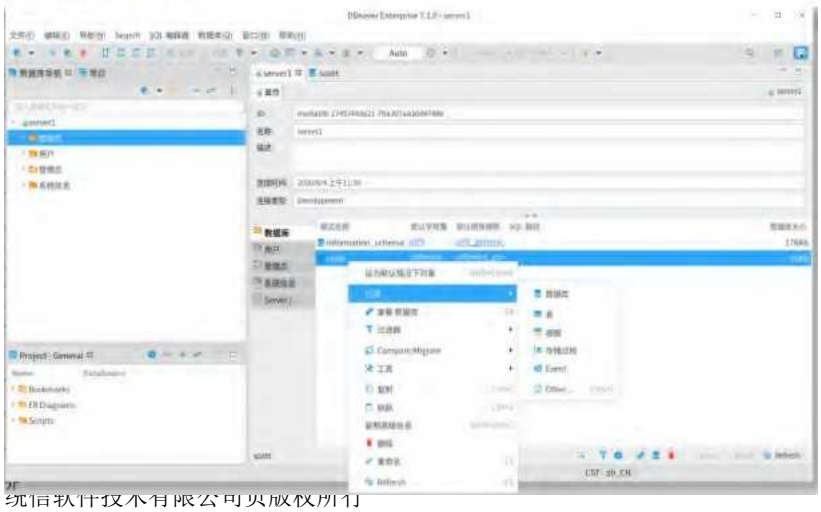
- Mysql workbench: oracle 官方出的，只支持原版 mysql 连接；
- Dbeaver：支持多种数据库，不过商店里的是（EE）企业版，收费，需要免费的社区版（CC）的话，可以去官网下载；

使用 GUI 图形化的工具，我们可以方便的建立、修改数据库、表，查询、导入导出数据以及 其它管理操作：

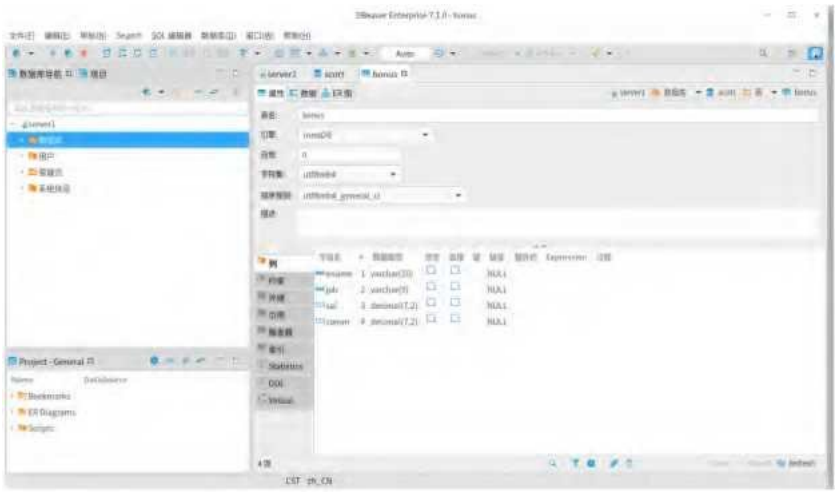
先建立连接：



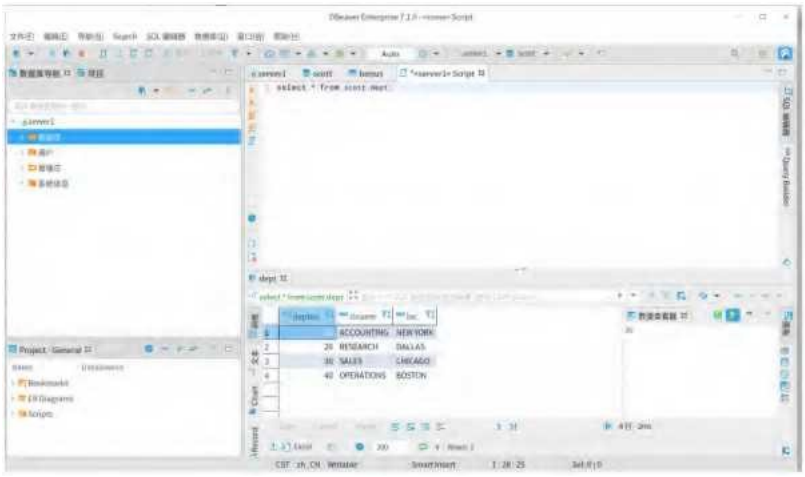
创建数据库和表：



修改字段和索引



编辑 SQL 查询数据:



12.3 项目实战

1. 根据企业需求完成数据库多实例部署

要求：

- 在 server1 上配置 mariadb,建立 uosdb1,导入 scott 示例库的的三个表及数据
- 在 server2 上配置 mariadb,建立 uosdb2,导入 scott 示例库的的三个表及数据
- 编写 crontab it-划任务，每天早上 2： 00 做一次数据库的全备份并压缩（xz），保留 30 天,将备份交差存放在异机 I:（server1 的数据库备份存在 server2 土，server2 同 理）

2. 安全性考虑完成数据库的权限划分

要求：

- 在 serverl 上，建立三个 mariadb 用户 dbuserl/dbuer2/dbuser3
- 在 serverl 上，建立三个数据库 uosdbl/uosdb2/uosdb3
- 一一对应授权：
  / dbuserl 只可访问 uosdbl
  / dbuser2 只可访问 uosdb2
  / dbuser3 只可访问 uosdb3

3.根据企业需求完成数据库和表的管理

要求：

- 在 serverl 上，配置 mariadb,并配置 root 可以连接
- 下载安装 dbeaver,图形客户端连接 serverl 上的 mariadb 服务
- 建立 scott 数据库，以及三个表、字段（参见＜＜4.1. 2 Mariadb 基础应用-数据库 DDL, DML 和 DCL 语言操作＞＞）

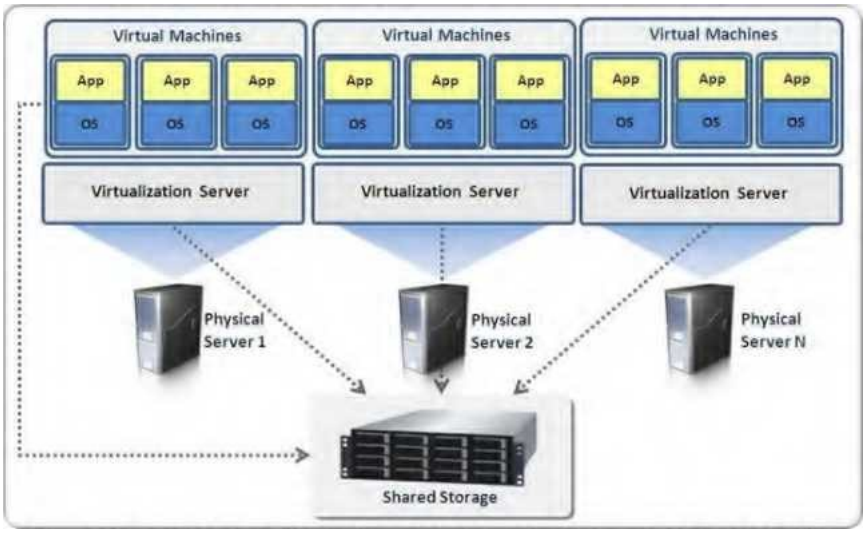
第 13 章 KVM 虚拟机管理

13. 1 虚拟化技术概述

虚拟化技术的历史背景

1、什么是虚拟化？

在计算机技术中，虚拟化（技术）或虚拟技术（英语：Virtualization）是一种资源管理技 术，是将计算机的各种实体资源（CPU、内存、磁盘空间、网络诂配器等），予以抽象、转 换后呈现出来并可供分区、组合为一个或多个电脑配置环境



由此，打破实体结构同的不可切割的障碍，使用户可以比原本的配置更好的方式来应用这些 电脑硬件资源。这些资源的新虚拟部分是不受现有资源的架设方式，地域或物理配置所限制。

2、为什么要用虚拟化

- 同一台物理机运行多个不同版本应用软件
- 硬件依赖性较低和便于数据迁移

3、虚拟化技术的优势

- 1. 降低运营成本
- 2. 提高应用兼容性
- 3. 加速应用部署
- 4. 提高服务可用性
- 5. 提升资源利用率
- 6. 动态调度资源
- 7. 降低能源消耗

虚拟化技术的演进与分类

虚拟化技术的演进:

1、虚拟化(Virtualization)技术最早出现在 60 年代的 IBM 大型机系统，在 70 年代的 System370 系列中逐渐流行起来，这些机器通过一种叫虚拟机监控器(VirtualMachine Monitor, VMM)的程序在物理硬件之上生成许多可以运行独立操作系统软件的虚拟机 (VirtualMachine)实例。

2、虚拟化技术发展到 2000 年左右，开始惠及到了 x86 架构。在此之前，虚拟化技术在 x86 架构上进展十分缓慢，其原因主要可以归结为两点：其一是早期的 x86 处理器的性能不足； 其二是 X86 架构本身不适合进行虚拟化，不过这个障碍已经由英特尔、AMD 修改 x86 处理器 的指令集得到解决；

3> 2005 年底英特尔发布了业内首个面向 X86 服务器的辅助虚拟化技术 Intel-VT 及相关的 处理器产品，从而拉开了 X86 架构虚拟化技术应用的新时代大幕；硬件辅助虚拟化技术缩小 了软件性能的差异，X86 虚拟化的开始进入爆发期。

4、随着硬件辅助虚拟化、10 直通等技术的发展，虚拟化软件功能逐渐向硬件板卡卸载成为 趋势，各厂商的虚拟化技术架构开始趋同，功能开始标准化。

5、当技术趋同，功能趋同后，业界比拼的是：谁最快的做出来、谁做的性能更好、谁的服 务最佳；归根到底比拼的是投入和对硬件的理解，华为对硬件的理解是所有虚拟化厂商中最 深的.

虚拟化技术主要分为以下几个大类:

- 1、平台虚拟化(Platform Virtualization),针对计算机和操作系统的虚拟化。
- 2、资源虚拟化(ResourceVirtualization),针对特定的系统资源的虚拟化，比如内 存、存储、网络资源等。
- 3、应用程序虚拟化(ApplicationVirtualization),包括仿真、模拟、解释技术等。

我们通常所说的虚拟化主要是指平台虚拟化技术，通过使用控制程序(ControlProgram,也 被称为 VirtualMachine Monitor 或 Hypervisor),隐藏特定计算平台的实际物理特性，为 用户提供抽象的、统一的、模拟的计算环境(称为虚拟机)。虚拟机中运行的操作系统被称 为客户机操作系统(GuestOS),运行虚拟机监控器的操作系统被称为主机操作系统(HostOS)， 当然某些虚拟机监控器可以脱离操作系统直接运行在硬件之上(如 VWARE 的 ESX 产品)o 运行虚拟机的真实系统我们称之为主机系统。

平台虚拟化技术义可以细分为如下几个子类:

1、全虚拟化(Full Virtualization)全虚拟化是指虚拟机模拟了完整的底层硬件，包括处 理器、物理内存、时钟、外设等，使得为原始硬件设计的操作系统或其它系统软件完全不做 任何修改就可以在虚拟机中运行。操作系统与真实硬件之间的交互可以看成是通过一个预先 规定的硬件接口进行的。全虚拟化 VMM 以完整模拟硬件的方式提供全部接口(同时还必须 模拟特权指令的执行过程)。举例而言，x86 体系结构中，对于操作系统切换进程页表的操 作，真实硬件通过提供一个特权 CR3 寄存器来实现该接口，操作系统只需执行〃inov pgtable,%%cr3〞汇编指令即可。全虚拟化 VMM 必须完整地模拟该接口执行的全过程。如果 硬件不提供虚拟化的特殊支持，那么这个模拟过程将会十分复杂：一般而言，VMM 必须运行 在最高优先级来完全控制主机系统，而 GuestOS 需要降级运行，从而不能执行特权操作。
当 GuestOS 执行前面的特权汇编指令时，主机系统产生异常(GeneralProtection Exception),执行控制权重新从 GuestOS 转到 VMM 手中。VMM 事先分配一个变量作为影 子 CR3 寄存器给 GuestOS,将 pgtable 代表的客户机物理地址(GuestPhysical Address) 填入影子 CR3 寄存器，然后 VMM 还需要 pgtable 翻译成主机物理地址(HostPhysical Address)并填入物理 CR3 寄存器，最后返回到 GuestOS 中。随后 VMM 还将处理复杂的 GuestOS ***缺页异常***(PageFault)。比较著名的全虚拟化 **VMM** 有 **MicrosoftVirtual PC、VMwareWorkstation> SunVirtual Box、ParallelsDesktop for Mac** 和 **QEMUo**

2、超虚拟化(Paravirtualization)这是一种修改 GuestOS 部分访问特权状态的代码以便 直接与 VMM 交互的技术。在超虚拟化虚拟机中，部分硬件接口以软件的形式提供给客户机 操作系统，这可以通过 Hypercall(VMM 提供给 GuestOS 的直接调用，与系统调用类似) 的方式来提供。例如,GuestOS 把切换页表的代码修改为调用 Hypercall 来直接完成修改影 子 CR3 寄存器和翻译地址的工作。由于不需要产生额外的异常和模拟部分硬件执行流程， 超虚拟化可以大幅度提高性能，比较著名的 VMM 有 Denali、Xen。

3、硬件辅助虚拟化(Hardware-Ass is ted Virtualization)硬件辅助虚拟化是指借助硬件 (主要是主机处理器)的支持来实现高效的全虚拟化。例如有了 Intel-VT 技术的支持， GuestOS 和 VMM 的执行环境自动地完全隔离开来，GuestOS 有自己的“全套寄存器”，可 以直接运行在最高级别。因此在上面的例子中,GuestOS 能够执行修改页表的汇编指令。 Intel-VT 和 AMD-V 是目前 x86 体系结构上可用的两种硬件辅助虚拟化技术。

4、部分虚拟化(Partial Virtualization) VMM 只模拟部分底层硬件，因此客户机操作系 统不做修改是无法在虚拟机中运行的，其它程序可能也需要进行修改。在历史上，部分虚拟 化是通往全虚拟化道路上的重要里程碑，最早出现在第一代的分时系统 CTSS 和 IBMM44/44X 实验性的分页系统中。

5、操作系统级虚拟化(Operating System Level Virtualization)在传统操作系统中，所 有用户的进程本质上是在同一个操作系统的实例中运行，因此内核或应用程序的缺陷可能影 响到其它进程。操作系统级虚拟化是一种在服务器操作系统中使用的轻量级的虚拟化技术， 内核通过创建多个虚拟的操作系统实例(内核和库)来隔离不同的进程，不同实例中的进程 完全不了解对方的存在。

KVM 虚拟化环境部署和管理方法

KVM 简介

KVM — 全称是基于内核的虚拟机(Kernel-based Virtual Machine)

是一个开源软件，基于内核的虚拟化技术，实际是嵌入系统的一个虚拟化模块，通过 优化内核来使用虚拟技术，该内核模块使得 Linux 变成了一个 Hypervisor,虚拟机使用 Linux 自身的调度器进行管理。

KVM 是基于虚拟化扩展(Intel VT 或者 AMD-V)的 X86 硬件的开源的 Linux 原生的 全虚拟化解决方案。KVM 中，虚拟机被实现为常规的 Linux 进程，由标准 Linux 调度程序 进行调度；虚机的每个虚拟 CPU 被实现为一个常规的 Linux 进程。这使得 KVM 能够使用 Linux 内核的已有功能。但是，KVM 本身不执行任何硬件模拟，需要客户空间程序通过 /dev/kvm 接口设置一个客户机虚拟服务器的地址空间，向它提供模拟的 I/O,并将它的视 频显示映射回宿主的显示屏。目前这个应用程序是 QEMUo

QEMU 是一套由 Fabrice Bel lard 所编写的模拟处理器的自由软件。它与 Bochs, PearPC 近 似，但其具有某些后两者所不具备的特性，如高速度及跨平台的特性。经由 kqemu 这个开源 的加速器，QEMU 能模拟至接近真实电脑的速度。

libvirt 是管理虚拟机和其他虚拟化功能的软件，比如存储管理，网络管理的软件集合。它 包括一个 API 库，一个守护程序(libvirtd)和一个命令行工具(virsh) ： libvirt 本身 构建于一种抽象的概念之上。它为受支持的虚拟机监控程序实现的常用功能提供通用的 APIo libvirt 的主要目标是为各种虚拟化工具提供一套方便、可靠的编程接口，用一种单 一的方式管理多种不同的虚拟化提供方式。

注意：我们通常所说和使用的 KVM 虚拟机，实际上是这三个软件的结合

安装 kvm、qemu、libvirtd：

查看和打开 CPU 虚拟化支持：

```
egrep 〃(svm|vmx)“ /proc/cpuinfo                    #查看 cpu 是否支持虚拟化
```

注：

- vmx 对应 inter 的 cpu
- svm 对应 amd 的 cpu
- 需在主板 bios 打开虚拟化选项，多数计算机默认处于打开状态

查看内核 kvm 支持

```
lsmod|grep kvm 安装所需软件包
```

apt-get install libvirt0 libvirt-daemon qemu virt-manager bridge-utils libvirt-clients python-libvirt qemu-efi uml-utilities virtinst qemu-system #安装 KVM 和相关虚拟化工具

启动 **libvirtd**

```
systemctl restart libvirtd. service              #启动 libvirtd
systemctl status libvirtd. service #查看启动状态
```

两种管理虚拟机的方法：

- virt-manager 是图形化方式，相对直观。
- virsh 是命令行方式

13.2 KVM 配置与应用

virsh 命令行管理工具详解

virsh （虚拟 shell），基于命令行的管理工具，可以实现简单的资源管理。支持交互模式 系正

```
# virsh  shutdown uosl start                常关闭 VID 虚拟机
# virsh  uosl destroy uosl                  #启动 kvm 虚拟机
# virsh  list list 一 all                   #强制关闭 kvm 虚拟机
# virsh  suspend uosl resume                #显示本地活动虚拟机
# virsh  uosl dominfo uosl                  #查看所有虚拟机
# virsh  undefine uosl 一旦关               #挂起 kvm 虚拟机
# virsh  闭就会消失) dumpxml                #恢复被挂起虚拟机
# virsh  uosl                               #查看指定虚拟机的配置摘要信息
# virsh                                     #删除 kvm 虚拟机（如果虚拟机处于
tuning,
# virsh                                     #显示虚拟机的当前配置文件
# virsh define uos2. xml                    #通过配置文件定义一个虚拟机（这
个虚拟机还不是活动的）
```

- # virsh autostart uosl ,/etc/libvirt/qemu/autostart/) #虚拟机设为自动启动（生成
- # virsh autostart --disable uosl #取消自动启动
- # virsh edit uosl #编辑配置文件（一般是在刚定义完虚拟机之后）

```
# virsh  setmem uosl 512000                 #给虚拟机设置内存大小
# virsh  setvcpus uosl 4                    #给虚拟机设置 cpu 个数
```

qemu-img 是一个功能强制磁盘镜像管理工具

qemu-img —help 包括以下功能

- check 检查完整性 创建镜像 提交更改 比较
- create
- commit 转换
- compare 获得信息 映射
- convert 快照管理
- info
- map
- snapshot
- rebase 在已有的镜像的基础上创建新的镜像

- resize 调整大小
- amend 修订镜像格式选

KVM 虚拟机的 xml 配置文件说明

xml 配置文件也就是通常所说的虚拟机的描述文件，主要用来定义一个虚拟机的名称、UUID、 CPU、内存、虚拟磁盘、网卡等各种参数设置

/etc/libvirt/qemu/ #配置文件默认路径

uos. xml 配置文件说明：

```
<domain type='kvm'>
<name>uos</name>                    #虚拟机的名字
<uuid>16e9cb49-8b05-4b34-9325-      ttuuid 值
3bcfl69dd32e</uuid>
<memory unit='KiB' >4194304</memory>    #虚拟机的最大内存
<currentMemory                          #虚拟机当前的内存
unit='KiB' >4194304</currentMemory>     #该虚拟机的 cpu 数
〈type arch='x86_64' machine='pc-i440fx-3. 1'>hvm〈/type> #hvm 表示全虚拟化
<boot dev='hd'/〉                    #hd 表示从硬盘启动
</os>
ooooooooooo
<emulator>/usr/bin/qemu-system-x86_64</emulator>    #二进制模拟器设备的完整路径
〈disk type='file' device='disk'〉           #disk 是用来描述磁盘的主要容器
<driver name='qemu' type='qcow2,/>
<source file-/var/1 ib/1 ibvirt/images/kvm/uos. qcow2'/〉        #指定磁盘上文件的绝对路径
<target dev=' hda' bus=' ide' />
<address type='drive' controller^ 0* bus='O' target='O' unit='O'/〉  ooooooooooooo
<interface type='direct             #桥接设备
<mac address=, 52:54:00:0c:ae: 41' /〉       甘 MAC 地址
<source dev='enp2s0, mode='bridge' />
<model type='el000, />
<address type='pci' domain='0x0000' bus=, 0x00* slot='0x03* function='0x0' /> </interface>
<source network='brO/>
```

KVM 虚拟机重命名技术

生成新的 UUID 字申，并复制备用

```
uuidgen
```

76d5dc2c-5eef-4e30-8b6c-e58851814f84

virsh edit uos #编辑虚拟机配置

```
<domain type=' kvm' >
<name>uos2</name>                    # 新名称
```

<uuid>76d5dc2c-5eef-4e30-8b6c-e58851814f84</uuid> #新 UUID 值

virsh list --all #确认导入的新配置

```
 Id Name State
----------------------------------------------------

 -uosl shut off
 - uos2 shut off                        #新名称的虚拟机
```

virsh undefine uosl #删除 I 日名称的虚拟机

virsh list --all #确认结果Id Name State

-uos2 shut off

KVM 虚拟机的复制技术

生成新 UUID,并复制备用

uuidgen 7b292992-a86e-4386-951f-a577365cc721

导出配置

```
virsh dumpxml uos2>/etc/libvirt/qemu/uos3. xml
```

修改导出后的配置

vim /etc/libvirt/qemu/uos3. xml

<domain type='kvm'>

< name >uos 3< / name> # 新名称

<uuid>3d7f2197-8dad-473c-99af-0570cd02ae5c</uuid> # 新 UUID 值

<memory unit=' KiB' >4194304</memory> #内存大小，单位 KB

<currentMemory unit- KiB' >4194304</currentMemory> #虚拟机分配的内存大小，单位 KB

<vcpu placement」static* >2</vcpu> #虚拟 CPU 核数

<disk type」file' device」disk'>

<driver name='qemu' type」qcow2' />

<source file=' /var/lib/libvirt/images/uos3. qcow2' /> #新磁盘路径

<target dev=' hda' bus=' ide' />

<address type='drive' controller=' 0' bus=' 0' target- 0' unit=' 0' />

</disk>

<interface type=' network* > #虚拟机网络连接方式

<mac address」 52:54:00:f5:9c:44 '/> #修改 MAC 地址

<source network」private】' /> <model type- el000' /> #当前主机网桥的名称 bus- 0x00' slot- 0x03' function- 0x0' />

<address type='pci' domain」0x0000'

确认结果

```
virsh list –all
```

定义新虚拟机

```
virsh define uos3. xml
```

```
 Id Name State

 -uos2 shut off
 -uos3 shut off #新虚拟机
```

虚拟机快照建立的方式

快照（raw格式的磁盘无法创建快照）

```
#qemu-img snapshot -l /kvm/img/test12. qcow2 #查看磁盘快照
virsh snapshot-list test12                    #查看快照
virsh snapshot-create test12                  #生成快照
virsh snapshot-create-as test12 snap1 virsh   #自定义快照名 #快照恢复虚拟机
snapshot-revert test12 snap1 virsh snapshot-delete #删除指定快照
test12 snapname virsh snapshot-current test12
```

13.3 项目实战

1. 使用命令行工具创建虚拟机

用法：virt-install --name NAME --memory MB STORAGE INSTALL [options] 用途：从指定安装源创建新虚拟机。

示例

创建镜像文件

```
 qemu-img create -f qcow2 -o size=20G /data/kvm/uos-server3. qcow2 通过cdrom镜相引导安装系统:
 virt-install ~n uos~server3 –vcpus 2 -r 4096 -c /home/yangh t/ISO/uniontechos-desktop-20-professional-1021_amd64. iso --disk / data/kvm/uos-server3. qcow2, bus=virtio, size=20 --graphics vnc, listen=0. 0. 0. 0 –network bridge=virbr0, model=virtio
```

2. 利用已有的虚拟机配置文件创建新的虚拟机

把 uos2 当做模板机，提取 uos2 的虚拟机磁盘、xml 配置文件

```
qemu-img info /var/lib/libvirt/images/uos2. qcow2
```

输出：

```
image: /var/lib/libvirt/images/uos. qcow2 file
format: qcow2
virtual size: 64G (68719476736 bytes) disk          #虚拟机磁盘容量 #在陪划服务器占用容量
size: 8. 1G
cluster_size: 65536
Format specific information:
compat: 1. 1
lazy refcounts: true
refcount bits: 16
corrupt: false
```

复用 uos2 模板机的磁盘数据

```
qemu-img create -f qcow2 -b /var/lib/libvirt/images/uos2. qcow2 /var/lib/libvirt/images/uos4. qcow2
```

复制配置

```
cd /etc/1 ibvirt/qemU/"
cp uos2. xml uos4. xml
```

修改配置文件

```
vim uos4. xml
<name>uos4</name>                                   #新虚拟机名称
<uuid>da2478f0-abf0-llea-a912-6f46e76df6bl</uuid> #薪 UUID 值 〈disk type='file' device='disk'>                                  #文件类型
<driver name='qemu' type='qcow2' />                 #磁盘类型
<source file=, /var/lib/libvirt/images/uos4. qcow2, /> #新虚拟卷位置 <target dev=, hda' bus=' ide' />
<address type='drive' controller^ 0* bus='O' target='O' unit='O'/〉 </disk>
</controller>
<interface type='network'〉                          #虚拟机的网络接口类型
#删除 MAC 地址
<source network='network】'/〉                       #虚拟机的网卡的源网络名称
<model type='elOOO' />
```

根据配置文件定义虚拟机

```
virsh define /etc/libvirt/qemu/uos4. xml
```

查看所有虚拟主机 virsh list 一 all Id Name State

-uos2 shut off
-uos4 shut off

开启 uos 虚拟机

```
virsh start uos4
```

3. 使用命令行工具完成磁盘设备的热插拔

virsh at tach-disk （添加磁盘设备）

用法：

attach-disk <domain> <source> <target> [--targetbus <string>] [--driver <string>] [—subdriver <string>] [--iothread <string>] [--cache <string>] [--io <string>] [--type <string>] [--mode <string>] [--sourcetype <string>] [--serial <string>] [--wwn <string>] [--alias <string>] [--rawio] [--address <string>] [--multifunction] [--print-xml] [--persistent] [--config] [--live] [--current]

示例：

1、 用 qemu-img 创建一块 100G 的 qcow2 硬盘 SOURCEFILE=/data/kvm/uos-serverl-diskl. qcow2 qemu-img create -f qcow2 $SOURCEFILE 20G

2、附加磁盘到指定虚拟机

```
DOMAIN=uos-serverl
TARGET 二 sdb
virsh attach-disk --domain $DOMAIN —source $SOURCEFILE 一 target $TARGET
-subdriver qcow2 --config --live
```

SDOMAIN：想要挂载数据盘文件的虚拟机名

$ TARGET： 一般为 vdb, vdc...

—subdriver：这一项是必须的，如果不加的话，虚拟机不知道镜像文件的 格式，挂载就会失败。

Virsh detach-disk （删除磁盘设备）

用法：

detach-disk <domain> <target> [一 persistent] [--config] [一 live] [--current] [一 print-xml]

示例：

```
virsh detach-disk UOSserverl sdb
```

输出：

Disk detached successfully

第 14 章 Docker 容器

14. 1 Docker 概述

容器技术的产生背景和工作原理

产生背景

过去一款产品的开发和上线需要两套环境，分别是开发环境和上线运行环境

在开发一一运维的过程中环境配置非常麻烦口经常出问题，比如要发布一个 war 项目，需要 配置的环境很多包括项目自带环境安装，服务器配置应用环境等等并且不能跨平台进行，这 给不论是开发还是运维都带了很大的压力：开发人员说，我的程序在我的电脑里运行很正常， 而运维人员说，你的程序根本运行不了。

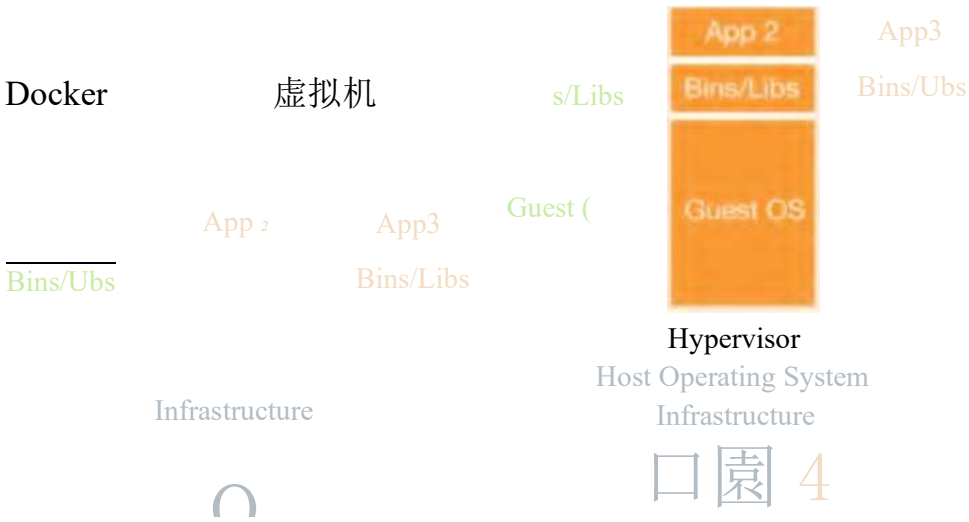
容器技术概述

容器是一个允许我们在资源隔离的过程中，运行应用程序和其依赖项的、轻量的、操 作系统级别的虚拟化技术，运行应用程序所需的所有必要组件都打包为单个镜像，这个镜像 是可以重复使用的。当镜像运行时，它是运行在独立的环境中，并不会和其他的应用共享主 机操作系统的内存，CPU 或磁盘。这保证了容器内的进程不会影响到容器外的任何进程。

容器与虚拟机之间的区别

虚拟机通常包括整个操作系统和应用程序。还需要与他们一起运行的虚拟机管理程序来 控制虚拟机。因为它们包括操作系统，因此它们的大小是几千兆字节（1 千兆字节=1GB） o 使用虚拟机的一个缺点是它们需要几分钟的时间才能启动操作系统，和初始化它们托管的应 用程序。

容器则是轻量级的，大部分是兆字节（1 兆字节=1MB）大小的。容器的性能与虚 拟机相比较，容器性能更好，可以立即启动。



显示易见，上图清晰的说明 docker 与虚拟机（Hypervisor）之间的根本区别:

- Docker 是在限定的/bins/libs 运行环境中运行应用;
- 虚拟机是比 docker 多了 Gue st OS 层，即完全虚拟了一台服务器;

Linux Container 的基本架构

通过 LXC （Linux 容器，Linux Container）来进行进程隔离

容器相当于你运行了一个接近于裸机的虚拟机。这项技术始于 2008 年，LXC 的大部分功能来自于 Solaris 容器（又叫做 Solaries Zones）以及之前的 FreeBSD jails 技术。

LXC 并不是创建一个成熟的虚拟机，而是创建 r 一个拥有自己进程程和网络空间的虚拟环 境，使用命名空间来强制进程隔离并利用内核的控制组（cgroups）功能，该功能可以限制， 计算和隔离一个或多个进程的 CPU,内存，磁盘 I/O 和网络使用情况。您可以将这种用户 空间框架想像成是 chroot 的高级形式。

chroot 是一个改变当前运行进程以及其子进程的根目录的操作° 一个运行在这种环境的程 序无法访问根目录外的文件和命令。

注意： LXC 使用命名空间来强制进程隔离，同时利用内核的控制组来计算以及限制一个或多 个进程的 CPU,内存，磁盘 I / O 和网络使用。

但容器充竟是什么？

简短的答案是容器将软件应用程序与操作系统分离，为用户提供干净 FL 最小的 Linux 环境， 与此同时在一个或多个隔离的“容器”中运行其他所有内容。

容器的目的是启动一组有限数量的应用程序或服务（通常称为微服务），并使它们在独立的 沙盒环境中运行。

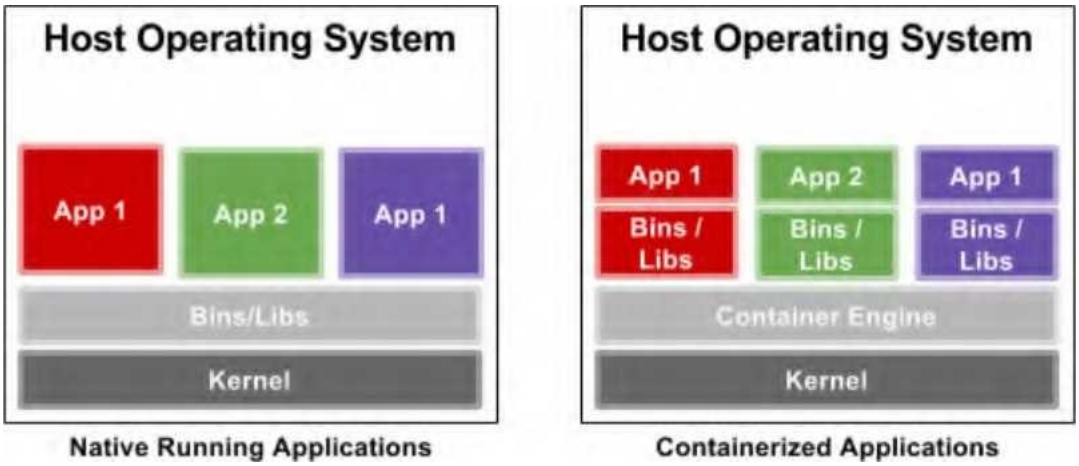


图 1 对比在传统环境以及容器环境运行的应用

防止互相影响

这种隔离可防止在给定容器内运行的进程监视或影响在另一个容器中运行的进程。此外，这 些集装箱化服务不会影响或干扰主机。能够将分散在多个物理服务器上的许多服务合并为一 个的想法是数据中心选择采用该技术的众多原因之一。

容器有以下几个特点：

- 安全性：容器里可以运行网络服务，这可以限制安全漏洞或违规行为造成的损害。那些 成功利用那个容器的一个或多个应用的安全漏洞的入侵者将会被限制在只能在那个容 器中做一些操作。
- 隔离性：容器允许在同一物理机器 11 部署一个或多个应用程序，即使这些应用程序必须 在不同的域下运行，每个域都需要独占访问其各自的资源。例如，通过将每个容器关联 的不同 IP 地址，在不同容器中运行的多个应用程序可以绑定到同一物理网络接匚 I。
- 虚拟化和透明性：容器为系统提供虚拟化环境，这个环境可以隐藏或限制系统底层的物 理设备或系统配置的可见性。容器背后的一般原则是避免更改运行应用程序的环境，但 解决安全性或隔离问题除外。

容器的特性

- 容器技术已经成为应用程序分装和交付的核心技术
- 容器技术内核组成（Cgroups-资源管理 NameSpace-进程隔离）
- 由于是在物理机器 I：实施隔离、启动一个容器、可以像启动一个进程一样快速

Docker 技术的基础特性和应用场景

docker 简介

- docker 的中文意思是码头搬运工
- Docker 是容器技术的一种实现，是完整的一套容器管理系统
- Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然 后发布到任何流行的 Linux 机器上，也可以实现虚拟化。

使丿 IJG0 语言开发

Docker 使用 Google 公司推出的 Go 语言 进行开发实现，基于 Linux 内核的 cgroup» namespace,以及 AUFS 类的 Union FS 等技术，对进程进行封装隔离，属于操作系统层面 的虚拟化技术。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。最初 实现是基于 LXC,从 0.7 以后开始去除 LXC,转而使用自行开发的 libcontainer,从 1.11 开始，则进一步演进为使用 runC 和 containerdo

Docker 架构

- Docker 使用客户端-服务器(C/S)架构模式，使用远程 API 来管理和创建 Docker 容器。
- Docker 容器通过 Docker 镜像来创建。

Docker 的优点

更高效的利用系统资源

由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销,Docker 对系统资源的利 用率更高。无论是应用执行速度、内存损耗或者文件存储速度，都要比传统虚拟机技术更高 效。因此，相比虚拟机技术，一个相同配置的主机，往往可以运行更多数量的应用。

更快的启动时问

传统的虚拟机技术启动应用服务往往需要数分钟，而 Docker 容器应用，由于直接运行于宿 主内核，无需启动完整的操作系统，因此可以做到秒级、甚至毫秒级的启动时间。大大的节 约了开发、测试、部署的时间。

一致的运行环境

开发过程中一个常见的问题是环境一致性问题。由于开发环境、测试环境、生产环境不一致， 导致有些 bug 并未在开发过程中被发现。而 Docker 的镜像提供了除内核外完整的运行时 环境，确保了应用运行环境一致性，从而不会再出现「这项目在我机器上没问题啊」这类 问题。

持续交付和部署

对开发和运维(DevOps)人员来说，最希望的就是一次创建或配置，可以在任意地方正常运 行。使用 Dockerfile 定制应用镜像来实现持续集成、持续交付、部署。而口使用 Dockerfile 使镜像构建透明化，不仅仅开发团队可以理解应用运行环境，也方便运维团队理解应用运行 所需条件，帮助更好的生产环境中部署该镜像。

更轻松的维护和扩展

Docker 使用的分层存储以及镜像的技术，使得应用重复部分的复用更为容易，也使得应用 的维护更新更加简单，基于基础镜像进一步扩展镜像也变得非常简单。此外,Docker 团队 同各个开源项目团队一起维护了一大批高质量的官方镜像，既可以直接在生产环境使用， 又可以作为基础进一步定制，大大的降低了应用服务的镜像制作成本。

Docker 容器的安装和部署

安装 docker

```
#  apt update
#  apt install -y docker, io
```

启动并设置开机启动

```
# systemctl restart docker
# systemctl enable    docker
```

查看版本信息

```
# docker version Client:
Version:           18. 09. 1
API version:       1.39
Go version:        gol. 11. 6
Git commit:        4c52b90
Built:             Tue, 03 Sep 2019 19:59:35 +0200
OS/Arch:           linux/amd64
Experimental:      false

Server:
Engine:
  Version:         18. 09. 1
  API version:     1. 39 (minimum version 1. 12)
  Go version:      gol. 11. 6
```

```
Git commit:        4c52b90
Built:             Tue Sep 3 17:59:35 2019
OS/Arch:           linux/amd64
Experimental:      false
```

14. 2 Docker 配置与应用

Docker 镜像和容器的基本操作

镜像的基本操作命

令 docker search uos //docker 镜像查询

- # docker pull uosproject/uos 〃下载镜像
- # docker images 〃查看镜像
- # docker save uosproject/uos:latest -o load uos. tar 〃导出 uos 镜像为 uos. tar
- # docker -i nginx. tar rmi nginx //倒入本地镜像
- # docker 〃删除镜像

镜像的导出和导入

- # docker save uosproject/uos > /tmp/uos. tar #将镜像导出为 tar 文件
- # docker rmi uosproject/uos
- # docker load < /tmp/uos. tar #删除原有的 uos 镜像，通过文件导入镜像

  # docker images #列出所有镜像

容器基本命令

- # docker run -it uosproject/uos bash //以 bash 环境启动镜像
- # docker ps -a //#不加-a 只显示运行状态的 docker 进程，每次都随机创建容器名和容器 ID
- # docker stop 597b8cd3ca55 start 〃通过容器 ID 关闭容器 〃
- # docker 597b8cd3ca55 restart 启动容器
- # docker 597b8cd3ca55 logs
- # docker 597b8cd3ca55 //通过容器 ID 来查询日志
- # docker tag uosproject/uos: latest uosproject/txuos: vl 〃修改镜像的名称和标 签，默认标签为 latest
- # docker inspect uosproject/uos 〃查看镜像的底层信息
- # docker attach 597b8cd3ca55 〃连接到容器
- # docker ps 〃查看容器
- # docker exec -it 597b8cd3ca55 bash
- # docker top 597b8cd3ca55 〃查看容器进程列表

过滤 地址和 ip 地址

mac

- # docker inspect -f ’ ({. Networksettings. MacAddress})’ 597b8cd3ca55

开启新的终端执行以下命令：

```
docker inspect 9725ff87872e
apt install -y curl
# docker inspect -f ' ({. Networksettings. IPAddress}}' 597b8cd3ca55 示例：J1j
curl 172. 17. 0. 2 // 可以访问到容器提供的 web 页面
```

dockerfile 构建镜像的一般过姊

- mkdir uosdocker #创建 Dockerfile 文件目录，进入目录并将需要用到的代码 CP 到此
- vim Dockerfile #编辑 Dockerfile 内容
- docker build -t uos:20 uosdocker/ #构建镜像，指定镜像名称和版本
- docker images #列出刚刚创建的 uos 镜像

简记:

```
FROM uosproject/uos          #镜像名称 #标记
MAINTAINER uos               #运行命令

RUN apt update
RUN apt -y install openssh-server
RUN useradd -m uos
RUN /bin/echo 〃tios:uos〃 | chpasswd

EXPOSE 22                    #应暴露 22 端口
CMD ["/usr/sbin/sshd", "-D〃]      #执行命令
```

Dockerfile 参数详解

- **(1) FROM**

FROM 指定基础镜像，必须是第一条指令 # 定制 nginx 镜像的 Dockerfile

```
FROM nginx
RUN echo '<hl〉Hello, Docker!</hl>' > /usr/share/nginx/html/index. html
```

注：

> Docker Hub 上有很多高质量的服务类的官方镜像可以拿来直接使用，比如：nginx、redis、mysql、php、mongo \ tomcat 等，可以在其中找最符合的一个进行定制，

> 另外也有一些方便开发、构建、运行各种语言的镜像，比如：node、python > golang >如果没有找到对应服务的镜像，官方镜像中还提供了一些更为基础的操作系统镜像，比如：ubuntu、debian、fedora、centos、uos 等，也可以利用这些操作系统提供的软件库

- **(2) RUN**

RUN :用来执行命令行命令，格式有两种：

> 1. shell 格式：RUN〈命令〉，就像直接在命令行中输入的命令一样

> 2. exec 格式：RUN [〃可执行文件〃，“参数 1”，“参数 2”],更像是函数调用中的 格式

注意:每一个 RUN 命令都会在 docker 镜像中新建一层，所以应该尽量少用 RUN 命令，而口

要在 RUN 的最后要做必要的清除工作

一般的方式：构建层次太多，未做清理工作

```
FROM uos:stretch
RUN apt-get update
RUN apt-get install -y gcc libc6-dev make wget
RUN wget -O redis. tar. gz "http://download, redis. io/releases/redis-5. 0. 3. tar. gz"
RUN mkdir -p /usr/src/redis
RUN tar -xzf redis. tar. gz -C /usr/src/redis —strip-components=1
RUN make -C /usr/src/redis
RUN make -C /usr/src/redis install
```

更好的方式：一层构建，并在最后清理压缩包等缓存文件

```
FROM uos:stretch
RUN buildDeps=' gcc libc6-dev make wget' \
&& apt-get update \
&& apt-get install -y SbuildDeps \
&& wget -O redis. tar. gz "http://download, redis. io/releases/redis-5. 0. 3. tar. gz" \
&& mkdir -p /usr/src/redis \
&& tar -xzf redis. tar. gz -C /usr/src/redis —strip-components=1 \
&& make -C /usr/src/redis \&& make -C /usr/src/redis install \
&& rm -rf /var/lib/apt/lists/† \
&& rm redis. tar. gz \
&& rm -r /usr/src/redis \
&& apt-get purge -y —auto-remove SbuildDeps
```

- (3) Dockerfile 上下文

构建新的镜像

-t :指定镜像名称和 tag

.:上下文，表示将本路径下的所有文件打包上传到 docker daemon,进行定制镜像 $ docker build -t nginx:v3 .

- (4) COPY

用来从构建上下文目录中〈原路径〉的文件/目录复制到新一层镜像内的 < 目标路径〉位置， 格式有两种：

> 1, shell 格式：COPY [一 chown=<user>:<group>]〈原路径>...<§标路径〉

> 2, exec 合适：COPY [—chown=<user>:<group>]["原路径 1","〈目标路径>"] 原路径：可以是多个，甚至可以是通配符

目标路径：可以是容器内的绝对路径，也可以是相对于工作目录的相对路径(工作目录可以 用 WORKDIR 指令来指定，不需要事先创建，会自动创建)

注：COPY 会将原文件的各种数据都保留，比如 读、写、执行权限，可以通过 —choTO=<user>:<group>选

† (5) ADD

项来改变文件的所属用户及所属组。

ADD :和 COPY 指令的功能，性质基本一致，也可以通过一 chown 改变文件所属用户和所 属组，但是在 COPY 的基础上增加了一些功能：

> 1，原路径为 URL ： Docker 会试图下载这个文件放到目标路径去，默认下载后的文件 权限为 600,如果想要修改权限或者下载的是压缩包，需要解压，则还需要额外的一层 RUN 进行调整，还不如直接用 RUN 指令用 wget 进行下载，处理权限，解压缩，然后 清理无用文件更合理，所以该命令不常用，而 FL 不推荐使用。

> 2,原路径为 tar 压缩包：如果压缩文件格式为 gzip ， bzip2 以及 xz 的情况下， ADD 指令将自动解压这个压缩文件到〈目标路径〉去，只有此种情况适合使用 ADD 指 令。

注：ADD 指令可能会使镜像构建缓存失效，从而可能会令镜像的构建变的比较缓慢。

- (6) CMD

CMD :和 RUN 指令相似，也是两种格式：

> 1, shell 格式：CMD〈命令〉

> 2, exec 格式：CMD [“可执行文件“, “参数 1”, “参数 2” ...]

> 3,参数格式列表：在指定了 ENTRYPOINT 指令后，用 CMD 指定具体的参数

CMD 指令用于指定默认的容器主进程的启动命令的，例如 ubuntu 官方镜相默认的 CMD 是 bash ,我们也可以在容器运行时指定运行别的命令，如：

#直接进入 bash

$ docker run -it ubuntu

#修改默认的 CMD

docker run -it ubuntu cat /etc/os-release 注：在指令格式上，一般推荐使用 **exec** 格式，这类格式在解析时会被解析为 JSON 数组，因此一定要用双引号“而不要使用单引号。

#如果执行

CMD echo $HOME

#实际执行会变更为：

```
CMD [〃sh〃 〃-c〃 〃echo $HOME〃] 注：容器的前台执行和后台执行问题
```

Docker 不是虚拟机，容器中的应用都应该以前台执行，而不能像虚拟机用 systemd 去 启动后台服务，容器内没有后台服务的概念。例如：

#错误代码

#目的：启动 nginx 在后台以守护进程的形式在运行

CMD service nginx start

#实际 I：执行

sh 为主进程，执行完成进程退出，导致容器也会退出

```
CMD [〃sh〃 〃-c〃 〃service nginx start〃] #正确做法
```

nginx :可执行文件

```
CMD [〃nginx〃, 〃-g〃, 〃daemon off;〃]
```

- (7) ENTRYPOINT

> ENTRYPOINT：格式和 RUN 指令格式一样，分为 exec 格式和 shell 格式，目的和 CMD 一样，都是在指定容器启动程序及参数；

> 当指定了 ENTRYPOINT 后，CMD 的含义就发生了变化，不再是直接的运行其命令，而是 将 CMD 的内容作为参数传给 ENTRYPOINT 指令，换句话说实际执行时，将变为： 〈ENTRYPOINT〉〃〈CMD〉”

用处 1 ：让镜像变成向命令一样使用：

ENTRYPOINT 的使用

ENTRYPOINT "-s", "https://ip.cn"] 用处 2 :应用运行前的准备工作：比如数据库配置，初始化工作，此时可以传 ENTRYPOINT 一个脚木，然后通过 CMD 指定参数，在脚木最后执行。

示例：

```
FROM aduser
RUN groupadd -S redis && useradd -S -G redis redis


EXTRYPOINT ["docker-entrypoint. sh"] 7
EXPOSE 6379
CMD [〃:redis-server"] # docker-entrypoint. sh 脚本文件
```

```
#!/bin/bash
if [ 〃$1〃 = ' redis-server' -a 〃$(id -u)〃 = ' 0' ]; then chown -R redis
exec su-exec redis 〃$0〃 〃$矿
fi
exec
```

- (8) ENV

ENV :用来设置环境变量，格式有两种：

1, ENV <key> <value>

2, ENV <keyl>=<valuel> <key2>=<value2>...

在设置，环境变量之后，无论是后面的其它指令，如 RUN ,还是运行时的应用，都可以直 接使用这里定义的环境变量

#定义环境变量

```
ENV VERSION=1. 0 DEBUG=ON \
    NAME="Happy Feet"
```

- (9) ARG

ARG :构建参数，格式：

ARG〈参数名〉 [K 默认值〉]

> 构建参数和 ENV 的效果一样，都是设置环境变量

> 所不同的是，ARG 所设置的是构建环境的环境变量，在将来容器运行时是不会存在这些

环境变量的。

- (10) VOLUME

VOLUME：定义匿名卷，格式为：

1, VOLUME ["〈路径 1>"，"〈路径 2〉 〃...]

2, VOLUME〈路径〉

之前说过，容器运行时应该尽量保持容器存储层不发生写操作，对于数据库类需要保存 动态数据的应用，其数据库文件应该保存在卷中，为了防止运行时用户忘记将动态文件所保 存目录挂载为卷，在 Dockerfile 中，我们可以事先指定某些目录挂载为匿名卷，这样在运 行时如果用户不指定挂载，其应用也可以正常运行，不会向容器存储层写入大量数据。

```
#  /data 目录会在运行时自动挂载为匿名卷
VOLUME /data #运行时也可以覆盖这个挂载设置
#用 mydata 这个命名卷挂载到了 /data 这个位置，代替 Dockerfile 中的匿名卷的挂载 配置
docker run -d -v mydata:/data xxxx
```

- (11) EXPOSE

EXPOSE：声明端口，格式为：

EXPOSE〈端口 1> 端口 2>...]

该条指令是声明运行时容器提供的服务端口，这只是一个声明，在运行时并不会因为这 个声明应用就会开启这个端口的服务。这样声明带来两个好处：

1, 帮助镜像使用者理解这个镜像服务的守护端口，以方便配置映射

2, 在运行时使用随机端口映射，也就是 docker run -P 时，会自动随机映射 EXPOSE 的 端口

注：

要将 EXPOSE 和在运行时使用-p〈宿主端口〉Y 容器端口〉区分开来。-p 是映射宿主端口 和容器端口，就是将容器的对应端口服务公开给外界访问，而 EXPOSE 仅仅是声明容器打算 使用什么端口而已，并不会在宿主进行端口映射。

- (12) WORKDIR

WORKDIR :指定工作目录，格式为：

WORKDIR〈工作目录路径〉

该条指令可以来指定工作目录(或者称为当前目录)，以后各层的当前目录就被改为指 定的目录，如果该目录不存在，则会自动建立。

- (13) USER：指定当前用户，格式为：

USER〈用户名〉[Y 用户组〉]

该条指令和 WORKDIR 相似，都是改变环境状态并影响以后的层，WORKDIR 是改变工作 目录，USER 是改变之后层的执行 RUN , CMD 以及 ENTRYPOINT 这类命令的身份。注：如 果以 root 执行的脚本，在执行期间希望改变身份，比如希望以某个已经建立好的用户来运 行某个服务进程，不要使用 su 或者 sudo,这些都需要比较麻烦的配置，而且在 TTY ***缺失*** 的情况下经常出错，建议使用 gosu o

- (14) HEALTHCHECK

HEALTHCHECK：健康检查，格式为：

HEALTHCHECK [选项]

HEALTHCHECK NONE ：CMD〈命令〉：设置检查容器健康状况的命令

如果基础镜像有健康检查指令，使用这行可以屏蔽掉其健康检查指令

options:

> —interval*间隔〉：两次健康检查的间隔，默认为 30s;
> —timeout =<时长〉：健康检查命令运行超时时间，如果超过这个时间，本次健康检查 就被视为失败，默认 30s;
> —retries=<次数〉：当连续失败指定次数后，则将容器状态视为 unhealthy ，默认 3 次;

return value:

> 0 ：成功
> 1：失败
> 2：保留(不要使用这个值)

- (15) ONBUILD

ONBUILD：后构建指令，格式为：

ONBUILD〈其它指令〉

ONBUILD 是一个特殊的指令，它后面跟的是其它指令，比如 RUN, COPY 等，而这些指 令，在当前镜像构建时并不会被执行。只有当以以前镜像为基础镜像，去构建下一级镜像的 时候才会被执行。

如何发布镜像到公有仓库

上传镜像 uosproject/txuos: vl

上传镜像之前需要在 docker hub 注册账户，例如注册账户为 tiosl

```
#  docker login 〃登录验证
#  docker tag uosproject/txuos:vl uosl/uosproject: vl 〃更改镜像名称
#  docker push uosl/uosproject:vl
```

如何发布镜像到私有仓库

实例：搭建木地镜像仓库 在 UOS1 I：,运行仓库

```
docker pull docker, io/registry
docker run -it 一name=uosregistry docker, io/registry sh
find / -name registry           #查询 registry 仓库放在/var/lib/registry exit
docker stop uosregistry
docker rm uosregistry
#重新运行 registry 仓库到后台
```

开启新的终端执行以下命令：

```
docker run -d -p 5000:5000 -v /opt/registry:/var/lib/registry docker.io/registry
docker inspect 9725ff87872e
apt install -y curl
curl 172. 17. 0. 2 〃可以访问到容器提供的 web 页面
```

修改木机 docker 的启动参数并重启 docker

vim /etc/docker/daemon, json

{"registry-mirrors" :

["https://registry. docker-cn. com〃], "insecure-registries〃: ["192. 168. 200. 10:5000〃] }

#必须把 192. 168.200. 10:5000 添加为信任的安全 registry,否则报错,多个 registry 可 在 192. 168.200. 10:5000 后面继续写，用逗号分隔

systemctl restart docker

尝试 push 镜相到私有仓库

```
docker tag docker. io/httpd:2. 4 192. 168. 200. 10:5000/uosimages/uoshttpd:2. 4
docker push 192. 168. 200. 10:5000/uosimages/uoshttpd:2. 4
#将镜像推入 docker, io/registry 容器，作为内部仓库的镜像
curl 192. 168. 200. 201:5000/v2/catalog #查看 uosl 仓库内的镜像
```

在 uos2 h 安装 docker 环境

curl 192. 168. 200. 201:5000/v2/_catalog

firefox http://192. 168. 200. 201:5000/v2/_catalog docker pull 192. 168. 200. 201:5000/uosimages/uoshttpd :2. 4 #下载本地仓库镜像很

#如果报错 Get https://192. 168. 200. 201:5000/v2/_catalog: http: server gave HTTP response to HTTPS client,需要在 uos2 上的/etc/docker/daemon. json 文件中添加本地 仓库信任

登陆容器内执行操作

新建一个 nginx 容器

docker pull nginx:latest

docker run -it nginx:latest bash

docker ps -a #l 已录容器的 CID 号

docker start 9725ff87872e #9725ff87872e 为上一步记录的 CID 号 进入 nginx 容器

docker exec -it 9725ff87872e bash

修改 nginx 主页

```
#echo ' UOS web' > /usr/share/nginx/html/index. html
#nginx
```

Apache2 容器的搭建方式

实例：**httpd** 容器

```
docker pull docker. io/httpd:2.4 #下载 httpd 镜像
docker run -d -P 一name=uoshttpd docker. io/httpd:2. 4 #运行 httpd 容器
ps aux grep httpd docker     #查看 httpd 进程
top uoshttpd                 朴查看 uoshttpd 容器内进程
docker inspect uoshttpd | grep IP #查看容器内部 IP
```

浏览器打开：http://172. 17. 0.2/ 其它管理操作：

```
docker port uoshttpd              #获取 uoshttpd 的映射端口 #停止 httpd 容器
docker stop uoshttpd     #删除容器
docker rm uoshttpd
docker run -d -p 18888:80 --name=〃uoshttpd〃 一restart=always docker, io/httpd:2.4
```

参数说明：

> -P 自动分配端口

-P 手动分配端口，默认是协议为 tcp 可不写，udp 协议可用-p 18888:80/udp 格式

-d 后台运行

一 res tart 二 always 重启后自动启动

如何映射文件和文件夹

映射文件和文件夹

```
  docker run - d -p 18888:80 -v /uoshttpd/html:/usr/local/apache2/htdocs/ --name=uoshttpd
docker, io/httpd:2. 4 # 真机的 /uoshttpd 文件夹映射到容器的 ,/usr/local/apache2/htdocs/文件夹
  echo uos > /uoshttpd/html/index, html/
```

通过交互终端进入运行中的容器，后面可加脚木

```
docker exec -it uoshttpd bash
cat /usr/local/apache2/htdocs/index. html
exit
```

拷贝容器内文件到真机

```
docker cp uoshttpd:/usr/local/apache2/conf/httpd. conf /tmp/                    cat
/tmp/httpd. conf
```

开启新的终端执行以下命令：
docker inspect 9725ff87872e
关闭容器、删除容器、镜相 docker stop uoshttpd docker rm uoshttpd docker rmi httpd:2.4 rm -rf /uoshttpd/
curl 172.17.0.2 〃可以访问到容器提供的 web 页面

Mysql 数据库容器的搭建方式

实例：mysql 容器

```
docker pull mysql:5.6
docker run -d -p 33060:3306 -e MYSQL_ROOT_PASSWORD=uos -e MYSQL_USER=tios -e MY S QL_PAS
SWORD=uo s -e MYS QL_DAT ABAS E=uo s -v /uosmysql:/var/lib/mysql --name=uosmysql docker,
io/mysql:5. 6
```

#设置 mysql 数据库 root 密码 uos,新建用户 uos （密码 uos）,新建数据库 uos

配置 mysql 源 /etc/apt/source. list

deb [arch=amd64] https://download, docker, com/linux/debian stretch stable deb
http://mirrors. 163. com/deepin/ unstable main contrib non-free 安装 mysql 客户端
apt-get install -y mysql-server
docker inspect uosmysql grep IP
mysql -h172.17. 0. 2 -P33060 -uroot -puos
MySQL> show databases:

14.3 项目实战

1.部署 web 服务器并对外提供服务

要求：

- 在 serverl 上布署 apache2 容器，端口设置为 8080
- 将主页设置为 UOS will be the best OS.

2 .本地私有镜像仓库的部署

要求

- 在 serverl 上布置私有镜像仓库
- 配置 docker,上传 busy box 镜相

要求：

- 在 server2 上布署 mysql 容器
- 在 server2 上布置 wordpress 容器
- 使用浏览器配置 wordpress 使其对外提供 web 服务

第 15 章 Ansible 管理

15. 1 Ansible X 具概述

Ansible 的工作机制和组成部分

ansible 的介绍

- ansible 是一种自动化运维工具。
- Ansible 是一种集成 IT 系统的配置管理、应用部署、执行特定任务的开源平台，它是 基于 python 语言,由 Paramiko 和 PyYAML 两个关键模块构建。
- 集合了众多运维工具的优点，实现了批量系统配置、批量程序部署、批量运行命令等功 能。
- ansible 是基于模块工作的。
- ansible 不需要在远程主机上安装 client/agents,因为它们是基于 ssh 来和远程主机 通讯的。
- ansible 也被定义为配置管理工具。

配置管理工具通常具有以下功能：

- 确保所依赖的软件包已经被安装
- 配置文件包含正确的内容和正确的权限
- 相关服务被正确运行

常用的自动化运维工具技术特性比较：

项冃	Puppet	SaltStack	Ansible
开发语言	Ruby	Python	Python
是否有客户端	有	有	无
是否支持二次 开发	不支持	支持	支持
服务器与远程 机器是否相互 验证	是	是	是
服务器与远程 机器的通信是 否加密	是，标准的 SSL 协议	是，使用 AES 加密	是，使用 OpenSSH
平台支持	AIX ， BSD, HP-UX, Linux ， Mac OSX ， Solaris, Windows	BSD, Linux ， Mac OS X ， Solaris, Windows	AIX ， BSD ， HP-UX ， Linux ， Mac OS X ， Solaris

项目	Puppet	SaltStack	Ansible
是否提供 Web UI	提供	提供	提供，但是是商业版本
配置文件格式	Ruby 语法格式	YAML	YAML
命令行执行	不支持，大师可以通过配置 模块实现	支持	支持

Ansible 的组成

Ansible 主要由一下 6 部分组成

- (1) Ansible Playbooks：任务集，编排定义 Ansible 任务集的配置文件，由 Ansible 顺序依次执行，通常是 json 格式的 YML 文件。
- (2) Inventory： Ansible 管理主机的清单。
- (3) Modules： Ansible 执行命令的功能模块，多数为内置的核心模块，也支持自定义
- (4) Plugins：模块功能的补充，如连接类型插件、循环插件、变量插件、过滤插件等，改功能不常用。
- (5) API：供第三方程序调用的应用程序编程接匚 I；
- (6) Ansible：此处指的是组合 nventory> Modules、Plugins> API 的 Ansible 命令工 具，其为核心执行工具

ansible 工作流程

使用 Ansible 或 Ansible-playbooks 时，在服务器终端输入 Ansible 的 Ad-Hoc 命令集或 palybook 后，Ansible 会遵循预先编排的规则将 Playbooks 逐条拆解为 Play,再将 paly 组 织成 Ansible 可识别的任务(Task),随后调用任务涉及的所有模块(modules)和插件 (plugins),根据 Inventory 中定义的主机列表通过 SSH 将任务集以临时文件或命令的形 式传输到远程客户端执行并返回执行结果，如果是临时文件，则执行完毕后自动删除。

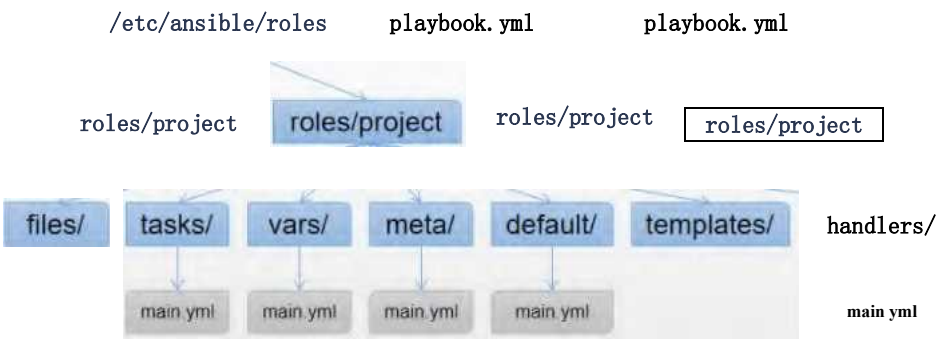
ansible 大体执行过程

Ansible 使用过程中会用到一些概念术语，我们先介绍一下。

Ansible 的与节点有关的重要术语包括控制节点，受管节点，清单和主机文件：

- 控制节点(Control node):指安装了 Ansible 的主机，也叫 Ansible 服务器端，管理 机。Ansible 控制节点主要用于发布运行任务，执行控制命令 o Ansible 的程序都安装 在控制节点上，控制节点需要安装 Python 和 Ansible 所需的各种依赖库。注意：目前 Ansible 还不能安装在 Windows 下。
- 受控节点(Managed nodes):也叫客户机，就是想用 Ansible 执行任务的客户服务器。
- 清单(Inventory):受控节点的列表，就是所有要管理的主机列表。
- host 文件：清单列表通常保存在一个名为 host 文件中。在 host 文件中，可以使用 IP 地址或者主机名来表示具体的管理主机和认证信息，并可以根据主机的用户进行分组。 缺省文件：/etc/ansible/hosts,可以通过-i 指定自定义的 host 文件。
- 模块(Modules):模块是 Ansible 执行特定任务的代码块。比如：添加用户，上传文 件和对客户机执行 ping 操作等 oAnsible 现在默认自带 450 多个模块，，Ansible Galaxy 公共存储库则包含大约 1600 个模块。
- 任务(Task):是 Ansible 客户机上执行的操作。可以使用 ad-hoc 单行命令执行一个 任务。
- 剧本(Playbook):是利用 YAML 标记语言编写的可重复执行的任务的列表，playbook 实 现任务的更便捷的读写和贡献。比如，在 Github 上有大量的 Ansible playbooks 共享， 你要你有一双善于发现的眼睛你就能找到大量的宝藏。
- 角色(roles):角色是 Ansible 1.2 版本引入的新特性，用于层次性、结构化地组织 playbookc roles 能够根据层次型结构自动装载变量文件、tasks 以及 handlers 等。

ansible roles 目录结构



Ansible 工具的特点概述

ansible 优点

- 只需要 SSH 和 Python 即可使用
- 无客户端
- ansible 功能强大，模块丰富
- 上手容易，门槛低
- 基于 Python 开发，做二次开发更容易

•使用公司比较多，社区活跃

ansible 特性

- 模块化设计，调用特定的模块完成特定任务
- 基于 Python 语言实现
- Paramiko、PyYAML （半结构化语言）
- Jinja2
- 其模块支持 JSON 等标准输出格式，可以采用任何编程语言重写

•部署简单

- 主从模式工作
- 支持自定义模块

•支持 playbook

- 易于使用

•支持多层部署

- 支持异构 IT 环境

Ansible 服务的安装和部署

ansible 安装

软件依赖关系

（1） 对管理主机

要求 Python 2. 6 版本以上

ansible 使用以下模块，都需要安装

paramiko、 PyYAML、 Jinja2、 httplib2、 six

（2） 对于被托管主机

<l>ansible 默认通过 SSH 协议管理机器

〈2〉被管理主机要开启 ssh 服务，允许 ansible 主机登录

〈3>在托管节点上也需要安装 Python2. 5 以上的版本

Ansible 的安装

```
pip install -y paramiko PyYAML Jinja2 httplib2 six          #安装 ansible 的依赖
apt install -y ansible                                      #安装 ansible
```

ansible 配置文件

- 配置文件目录：/etc/ansible/
- 执行文件目录：/usr/bin/
- ib 依赖库：/usr/lib/python2. 7/site-packages/ansible/
- help 文件：/usr/lib/python2. 7/site-packages/ansible

ansible 配置文件查找顺序

- 检查环境变量 ANSIBLE_CONFIG 指向的路径文件（export ANSIBLE_CONFIG=/etc/ansible. cfg）；
- . /ansible. cfg,检查当前目录下的 ansible. cfg 配置文件；
- ″/ansible. cfg 检查当前用户家目录下
- /etc/ansible. cfg 检查 etc 目录的配置文件。

ansible 配置文件参数详解 /etc/ansible/ansible. cfg ansible 有许多参数，下面我们列出一些常见的参数：

inventory = /etc/ansible/hosts #这个参数表示资源清单 inventory 文件的位置 #指向存放 Ansible 模块的目录，支持多个目录 library = /usr/share/ansible 方式，只要用冒号(:)隔开就可以 forks = 5 #并发连接数，默认为 5 sudo_user = root #设置默认执行命令的用户 remote_port = 22 #指定连接被管节点的管理端口，默认为 22 端口，建议修改，能够更加安全 host_key_checking = False True/False。 #设置是否检查 SSH 主机的密钥，值为 关闭后第一次连接不会提示配置实例 timeout =60 #设置 SSH 连接的超时时间，单位为秒 log_path = /var/log/ansible. log #指定一个存储 ansible 日志的文件（默认不记录日志）

定义 Inventory （主机列表）

- ansible 的主要功用在于批量主机操作，为了便捷地使用其中的部分主机，可以在 inventory file 中将其分组命名。
- 默认的 inventory file 为/etc/ansible/hosts。
- inventory file 可以有多个，FL 也可以通过 Dynamic Inventory 来动态生成。

Inventory 文件格式：

inventory 文件遵循 INI 文件风格，中括号中的字符为组名。可以将同一个主机同时归并到 多个不同的组中；此外，当如若目标主机使用了非默认的 SSH 端口，还可以在主机名称之后 使用冒号加端口号来标明。

```
[webservers]
www1. com:2222
ww2. com

[dbservers]
db1. com
db2. com
db3. com
```

如果主机名称遵循相似的命名模式，还可以使用列表的方式标识各主机，例如:

```
[webservers]
```

www[01:50]. example, com

[databases]

db-[a:f]. example, com **inventory** 其他的参数

ansible 基于 ssh 连接 inventory 中指定的远程主机时，还可以通过参数指定其交互方式； 这些参数如下所示：

- ansible_ssh_host # 远程主机
- ansible_ssh_port # 指定远程主机 ssh 端口
- ansible_ssh_user # ssh 连接远程主机的用户，默认 root
- ansible_ssh_pass #连接远程主机使用的密码,在文件中明文，建议使用-ask-pass 或 者使用 SSH keys
- ansible_sudo_pass # sudo 密码，建议使用--ask-sudo-pass
- ansible_connection # 指定连接类型：local, ssh, paramiko
- ansible_ssh_private_key_fi l e # ssh 连接使用的私钥
- ansible_shell_type #指定连接对端的 shell 类型，默认 sh,支持 csh, fish
- ansible_python_interpreter #指定对端使用的 python 编译器的路径

15.2 Ansible 基础应用

SSH 服务的免密钥登录

Ansible 是基于 ssh 连接的，为方便管理，一般我们会首先配置管理机登陆被管理机的免密 认证，以及其它一些必要的配置。

示例：

1、 准备虚拟机

4 台虚拟机：

(ansible 192. 168. 200. 10)
(nodel 192. 168. 200. 11)
(node2 192. 168. 200. 12)
(node3 192. 168. 200. 13)

2、 配置

免密登陆和 hosts 解析

- ssh-keygen 一直回车 #生成密钥
- ssh-copy-id nodel #拷贝私钥

所有机器都必须是免密登陆

- 4 台主机上面配置/etc/hosts,以 ansible 主机为例 vim /etc/hosts

192. 168.200. 10 ansible
192. 168.200. 11 nodel
192. 168.200.12 node2

|192. 168.200. 13 node3

Copy 到每台主机：

```
  for i in {10. .13}; do scp /etc/hosts 192. 168. 200. $i:/etc/hosts;  done 3、安装
ansible apt install -y ansible
```

4、配置主机清单

```
vim /etc/ansible/hosts [web]
nodel
node2 [db]
node3
```

5、结果测试

```
ansible all -m ping
node2 | SUCCESS => ( "changed" : false,
      " . " " "
       ping : pong
)
nodel I SUCCESS => ( "changed" : false,
      " . " " "
       ping : pong
)
node3 | SUCCESS => ( "changed" : false,
                                        " . " " "
                                       ping : pong
)

ansible web -m ping
node2 | SUCCESS => ( "changed" : false,
      " . " " "
       ping : pong
)
nodel I SUCCESS => ( "changed" : false,
      " . " " "
       ping : pong
```

Ansible 的常用命令和常用模块参数

ad-hoc 单行命令执行模式

批量执行单条任务,相当于在某个范围服务器组中的每一台上执行一个任务命令，如安装软 件、copy 文件等

获得模块的括助文档

- ansible-doc -l 列出所有可用的模块，版本 2. 7. 7, 2078 个模块
- ansible -s <module-name> 可以查看指定 module 的用法
- 参看官方帮助文档

示例：

```
ansible-doc -s apt
ansible-doc -l| wc -l
```

语法格式

ansible <host-pattern> [options]

常用格式：

ansible 主机模式　　-m 模块 [-a ‘参数’][-i 主机清单文件]

常用选项：

-b, —become：特权方式运行命令。
-s,以 sudo 来运行命令
-m：要使用的模块名称。
-a, -args：制定模块所需的参数。
-u：制定连接的用户名。
-h, —help 显示帮助内容。
-V, —verbose 以详细信息模式运行命令，可以用来调试错误。

示例：

ansible all -m apt -a “name=apache2 state=present" -b

主机模式

ansible 支持主机列表的正则匹配

- 全量：　all/*

•逻辑或
•逻辑非
•逻辑与　　&
- 切片：　　[]
- 正则匹配：以「开头

示例：

- ansible all -m ping 〃 . 〃 . #所有默认 inventory 文件中的机器 胴上
- ansible * -m ping ping #所有 122. 28. 13.X 机器
- ansible 121.28. 13.* -m ping #所有属于组 webl 或属于 web2 的机器
- ansible webl:web2 -m ping #属于组 webl,但不属于 web2 的机器
- ansible webl:!web2 -m ping #属于组 webl 又属于 web2 的机器
- ansible webl&web2 -m 1 ping #属于组 webserver 的第 1 台机器
- ansible webserver[0J -m ping #属于组 webserver 的第 1 到 4 台机器 :ample\.
- ansible webserver[0:5] (com|org) 〃 -m ping

示例：

•：• 多个组的并集

ansible web, db -m ping

或者

ansible , web:db* -m ping

•：• 多个组的交集

ansible 'web:&db' -m ping

•：• 多个组的差集

ansible 'web:!db' -m ping 注意：其中括起组名的必须是单引号

Ansible 常用的模块

文件模块

- copy:把文件从本机拷贝到远程
- file:设置文件的权限等属性
- lineinfile:确保文件中存在或不存在某些行
- syncchronize:能过 rsync 同步内容

软件包模块

- package:通过自动检测，使用操作系统原生的包管理器管理软件
- apt :通过 apt 进行软件包管理
- mount:文件系统挂载
- dnf :通过 dnf 进行软件包管理
- gem:管理 Ruby gems
- pip ：从 PyPI 进行 python 包管理

系统模块

- firewall:通过 firewal 1 管理服务和端口
- reboot:重启主机
- service:管理服务
- user:添加、删除、管理用户

网络工具模块

- get_url:通过 htp/htts/ftp 下载文件
- nmcli:管理网络
- tiri:与 Web 服务交互

Ansible 的模块实例和模块的综合应用

1、ping 模块

```
ansible all -m ping                                    #查看 ping 客户端组情况
```

2、command 命令模块

```
ansible all -m command -a "uptime"                     #查看客户端负载信息
ansible all -m command -a "useradd uostest" #在客户端添加用户 "touch
/uostest" #在客户端建立文件 "chdir=/tmp pwd" #
先改变目录，再执行 pwd
```

3、 shell 模块

让远程主机在 shell 进程下执行命令，从而支持 shell 的特性，如管道等 ansible all -m shell -a "echo "uostest:uostest" | chpasswd" chdir：在运行命令之前，切换到此目录。

ansible web "m shell -a "chdir=/tinp pwd"

executable：更改用于执行命令的 shell (bash, sh)。应该是可执行文件的绝对路径。

4、 script 模块

将 uos1 上的脚本在 uos2、uos3 和 uos4 上运行

vim /tmp/uosscript. sh

```
#!/bin/bash
touch /tmp/uosscript
useradd uosscript
echo "uosscript:uosscript" | chpasswd
```

在所有主机上批量执行 uosscript 脚本

```
ansible all -m script -a Vtmp/uosscript. sh"
```

5、 copy 模块

```
ansible all -m copy -a "src=/etc/fstab dest=/tmp/ owner=uostest group=root mode=0600"
ansible all -m copy -a "src=/var/Iog/apt/ dest=/tmp/ owner=uostest group=root mode=0750"
```

#src 主控端文件位置,dest 被控端目标位置,owner 文件复制过去后的所有者,group 文件复 制过去后的所属组,mode 文件的权限设定

6、 user 模块

```
ansible all -m user -a "name=uosuser state=present groups=uostest uid=2000 createhome=yes
home=/tmp/uosuser password=uosuser shell=/bin/bash comment='The user for uos' append=yes"
ansible all -m user -a "name=tiosuser state=absent remove=yes"
```

- name:用户名
- password:为用户设置登陆密码，此密码是明文密码加密后的密码
- upd at e_pas sword: always/on_create
  always: H 有当密码不相同时才会更新密码(默认)
  on_create:只为新用户设置密码

- shell:用户的 shell 设定
- groups:用户组设定
- home:指定用户的家目录

- state:present/absent
- append:yes/no
  yes:增量添加 group
  no:全量变更 group,只设置 groups 指定的 group 组（默认）
- remove：配合 state=absent 使用，删除用户的家目录->remove=yes
- expires：设置用户的过期时间，值是一个时间戳

7、file 模块

```
ansible all -m file -a 〃src=/etc/fstab dest =/tmp/uosfstab state=link'/
ansible all -m file -a 'zpath=/tmp/uosfile state=touch owner=uostest group=root mode=0600'z
```

- force：需要在两种情况下强制创建软链接，一种是源文件不存在，但之后会建立的情 况下；另一种是目标软链接已存在，需要先取消之前的软链，然后创建新的软链，有两 个选项：yes |no
- group：定义文件/目录的属组
- mode：定义文件/目录的权限
- owner：定义文件/目录的属主
- path：必选项，定义文件/目录的路径
- recurse：递归设置文件的属性，只对目录有效
- src：被链接的源文件路径，只应用于 stated ink 的情况
- dest：被链接到的路径，只应用于 stated ink 的情况
- state：
  - directory：如果目录不存在，就创建目录
  - file：即使文件不存在，也不会被创建
  - link：创建软链接
  - hard：创建硬链接
  - touch：如果文件不存在，则会创建一个新的文件，如果文件或目录已存在，则更 新其最后修改时间
  - absent：删除目录、文件或者取消链接文件

8、mount 模块

```
ansible all -m mount -a 〃src=/dev/sr0 path=/mnt/cdrom fstype=iso9660 opts=ro state=mounted'/
ansible all -m mount -a 〃src=/dev/sr0 path=/mnt/cdrom fstype=iso9660 opts=ro state=absentz/
```

- fstype：必选项，挂载文件的类型
- path：必选项，挂载点
- opts：传递给 mount 命令的参数
- src：必选项，要挂载的文件
- state：
  - present 添加到 fstab 中，不挂我

- absent 删除 fstab,并卸载资源
- mounted 添加到 fstab 中，并自动挂载
- amounted 不删除 fstab 信息，只卸载资源

9、service 模块

```
ansible all -m service -a "name=sshd state=started enabled=yes"
```

- arguments：给命令行提供一些选项
- enabled：是否开机启动 yes |no,要求状态(state)和启用(enabled)中至少有一个。
- name：必选项，服务名称
- runlevel：运行级别
- sleep：如果执行了 restarted,在则 stop 和 start 之间沉睡几秒钟
- state :对当前服务执行启动，停止、重启、重新加载等操作
  (started,stopped,restarted,reloaded)

10、setup 模块

```
ansible all -m setup                                          并查看所有主机信息
ansible all   -m setup -a   "filter=ansible_hostname"         并查看主机名 #
ansible all   -m setup -a   "filter=ansible_kernel"           查看内核信息
ansible all   -m setup -a   "filter=ansible_memory_mb"        #查看内存信息
ansible  all  -m  setup  -a  "filter=ansible_default_ipv4"      #查看网卡 ipv4 信息
ansible  all  -m  setup  -a   filter=*user*,                  #查看 user 相关信息
```

11、debug 模块

```
ansible all -m debug -a "msg 二 hahahahahaha"
```

- msg：调试输出的消息
- var：将某个任务执行的输出作为变量传递给 debug 模块，debug 会直接将其打印输出
- verbosity： debug 的级别(默认是 0 级，全部显示)

12、cron 计划任务模块

```
ansible all -m cron -a "name='uoscron' minute、'30' hour='12' job" touch /tmp/uoscron
user=root"
ansible all -m cron -a 'minute="*/1" job="Vusr/bin/date » /date. txt" name="date · 1
    ",
job
```

- minute= /hour= /day= /month= /weekday=某个值不写，默认就是*
- name:必选项，任务描述信息
- job:执行的任务，要加引号
- state :present （创建）/absent （删除） 13、get_url 模块

```
ansible all -m get_url -a 'url=http://192.168. 200. 10/favicon. ico dest=/tmp' # 配置好 url
```

- sha256sum：下载完成后进行 sha256 check；

- timeout：下载超时时间，默认 10s
- url：下载的 URL
- url_password> url_username：主要用于需要用户名密码进行验证的情况
- dest：将文件下载到哪里的绝对路径。如果 dest 是目录，则使用服务器提供的文件名，或者如果没有提供，将使用远程服务器上的 URL 的基本名称。
- headers：以格式“key： value，key： value”为请求添加自定义 HTTP 标头。

14、synchronize 模块

用于在主机间同步数据

- delete=yes 使两边的内容一样（即以推送方为主）
- compress=yes 开启压缩，默认为开启
- —excludes Git 忽略同步.git 结尾的文件
- mode^pull 更改推送模式为拉取模式

示例：

在 uosl I：

```
mkdir /uos
touch /uos/file{1.. 99）
ansible all -m synchronize -a ’ src=/uos dest=/tmp/ compress=yes'
```

在 uos2 I:

```
11 /tmp/uos
```

在 uosl I：

```
mkdir /uostest
ansible nodel -m synchronize -a ’ mode=pull src=/tmp/uos dest=/uostest'
11 /uostest/uos
```

15. 3 Ansible 高级应用

playbook 内容介绍和语法格式

playbook 模式（剧本模式）

- Playbook 是 Ansible 提供的最强大的任务执行方法。与 ad-hoc 命令不同，Playbooks 配置在文件中，可以重用和共享给其他人。
- playbooks 是以 YAML 标记语言来定义的。每个 playbook 由一个或多个 play 组成。play 的目标是将一组主机映射到任务中去。每个 play 包含一个或多个任务（task）,这些任 务每次执行一次。
- 从根本上来讲，所谓 task 无非是调用 ansible 的一个 moduleo 将多个 play 组织在一 个 playbook 中，即可以让它们联同起来按事先编排的机制同唱一台大戏。

playbooks 的组成部分

- Target section:定义要运行 playbook 的远程主机组

> hosts: hosts 用于指定要执行指定任务的主机，其可以是一个或多个由冒号分隔主机组

> user:指定远程主机上的执行任务的用户，还可以指定 sudo 用户等

- Variable section:定义 playbook 运行时使用的变量
- Task section:定义要在远程主机上运行的任务列表
  > name:每个任务都有 name,建议描述任务执行步骤,未通过 name 会用执行结果作为 name
  > 'module: opt ions':调用的 module 和传入的参数 args
- Handler section:定义 task 完成后需要调用的任务
  > notify:在 Task Section 在每个 play 的最后触发,调用在 hendler 中定义的操作
  > handler:也是 task 的列表

示例：

```
-hosts: master
  user: root
  vars:
    -motd_warning: * WARNING: Use by master ONLY'
  tasks:
    -name: setup a MOTD
      copy: dest=/etc/motd content—'{{ motd_warning }}〃
      notify: say something
  handlers:
    -name: say something
      command: echo "copy OK〃
```

yaml 语法格式

- 同级别数据元素必须有相同的缩进
- 子元素必须比父元素缩进更多
- 推荐缩进两个空格

•每个文件首行是“一”，最后一行是

- playbook 是 play 的列表，列表项表示为“-”
- paly 本身是 key:val 对的字典

括助查看

```
ansible-doc apt
```

执行 playbook

```
ansible-playbook 一 syntax-check **. yml        //检测语法格式
ansible-playbook **. yml              //执行 playbook
```

playbook 变量内容和变量定义

Ansible 变量简介

Ansible 支持利用变量来存储值，并在 Ansible 项目的所有文件中重复使用这些值。这可以 简化项目的创建和维护，并减少错误的数量。

通过变量，可以轻松地在 Ansible 项目中管理给定环境的动态值。例如：

- 要创建的用户
- 要安装的软件包
- 要重新启动的服务
- 要删除的文件
- 要从互联网检索的存档

命名变量

变量的名称必须以字母开头，并且只能包含字母、数字和下划线。

示例：web server：错误的变量名 web_server:正确的变量名

定义变量

可以在 Ansible 项目中的多个位置定义变量。这些变量大致可简化为三个范围级别：

- 全局范围：从命令行或 Ansible 配置设置的变量
- play 范围：在 play 和相关结构中设置的变量
- 主机范围：由清单、事实收集或注册的任务，在主机组和个别主机上设置的变量

优先级：

命令行变量»playbook 变量>>主机清单变量

ansible 变量优先级（由高到低）

1. ansible-playbook 命令中的变量，ansible-playbook -e var=value
2. task 变量
3. block 变量
4. role 中定义的变量和 include 变量
5. set_fact
6. registered 变量
7. vars_files
8. var_prompt
9. play 变量
10. host facts
11. playbook 中设置的 host_vars
12. playbook 中 设置的 group_vars
13. inventory 中设置的 host_vars
14. inventory 中设置的 group_vars
15. inventory 变量
16. role 中 defaults/main. yml 中定义的变量

playbook 中的变量

变量在 Ansible Playbook 中发挥着重要作用，因为它们可以简化 playbook 中变量数据的管 理。

在 playbook 中定义变量

Playbook 变量可以通过多种方式定义。一种常见的方式是将变量放在 playbook 开头的 vars 块中： - hosts: 192. 168. 86.132

```
  vars:
    package_name: httpd
  tasks:
    -name: install httpd
      yum:
        name: ({ package_name }}'                    #变量一定要写用引号
        state: present 也可以在外部文件中定义 playbook 变量。此时不使用 playbook 中的 vars 块，可以改为使 用 vars.files 指令，后面跟上相对于 playbook 位置的外部变量文件名称列表：
[root@ansible playbook]# mkdir vars && cd vars && echo 〃package_name: httpd〞 > httpd. yml
[root@ansible playbook]# vim test, yml -hosts: 192. 168. 86. 132
  vars_files:
    -vars/httpd. yml
  tasks:
    -name: install httpd
      yum:
        name: ' (( package_name }}'
        state: present
```

注：

声明了变量后，可以在任务中使用这些变量。若要引用变量，可以将变量名放在双大括号内。 在任务执行时，Ansible 会将变量替换为其值。需要注意的是：当变量用作开始一个值的第 一元素时，必须使用引号。这可以防止 Ansible 将变量引用视为 YAML 字典的开头。

在 inventory 主机列表中定义主机变量和组变量:

主机变量：可以在 inventory 中定义主机时为其添加主机变量以便于在 playbook 中使用。 例如：

```
[webservers]
wwwl. com http_port=80 maxRequestsPerChild=808
www2. com http_port=8080 maxRequestsPerChild=909 组变量：主机组变量针对组内所有的主机都生效。
```

示例：定义了 2 个主机组变量 ntp_server 和 nfs_server [webservers]

wwwl. com

ww2. com

[webservers:vars] ntp_server=ntp. com nfs_server=nfs. com

测试：

```
vim variable, yaml -hosts: all
  gather_facts: False
  tasks:
    -name: diplay Host Variable from hostfile
      debug: msg=〃The (( inventory_hostname }} Value is (( key }) 执行：
```

ansible-playbook variable, yaml 注意：如果主机同时定义了主机变量和主机组变量，名字相同时，主机变量生效，主机组变 量不生效；名字不同时，都可以调用。

从命令行覆盖变量

清单变量可被 playbook 中设置的变量覆盖，这两种变量又可通过在命令行中传递参数到 ansible 或 ansible-playbook 命令来覆盖。在命令行上设置的变量称为额外变量，使用 选择来指定额外变量

```
[root@ansible httpd]# ansible 192.168.86. 132 -i inventory -e 〃ansible passwrod=123456〃 -m ping //也可以把变量写在一个文件中用-e 选项来指定 [root@ansible httpd]# vim xx ans ible_password: 123456
[root@ansible httpd]# ansible 192. 168. 86. 132 -i inventory -e @xx -m ping
```

示例：使用已注册变量捕获命令输出

可以使用 register 语句捕获命令输出。输出保存在一个临时变量中，然后在 playbook 中可 用于调试用途或者达成其他目的

```
-name:
    hosts: 192. 168. 86. 132
    tasks:
      -name:
        user:
            name: cvg
            state: present
        register: useradd_result
      -debug: var=useradd_result                    //debug 模块用于将 install_result 注册变量的值转储到终端。
```

[root@ansible、]# ansible-playbook -C test, yml

管理机密和管理事实

管理机密

Ansible Vault

- Ansible 可能需要访问密码或 API 密钥等敏感数据，以便能配置受管主机，但是这些敏 感数据是任何有权限访问 Ansible 文件的用户都能查看，这样是不安全。
- Ansible 提供的 Ansible Vault 可以加密和解密任何由 Ansible 使用的结构化数据文件。

创建加密文件

- 使用 ansible-vault create filename 命令创建加密文件，然后输入的 vault 密码 [root@ansible group_vars]# ansible-vault create webserve

New Vault password:

Confirm New Vault password:

- 还可以用 **vault** 密码文件来存储 vault 密码，而不是通过标准输入途径输入 vault 密码。 这样做需要使用文件权限和其他方式来严密保护该文件。

[root@ansible group vars]# vim vault password

123456

```
[root@ansible group_vars]# ansible-vault create --vault-password-file=vault password webserve
```

查看加密的文件

使用 ansible-vault view filename 命令查看 Ansible Vault 加密的文件

```
[root@ansible group_vars]# ansible-vault view webserve
```

Vault password:

ans ible_pas sword: 123456

编辑现有的加密文件

使用 ansible-vault edit filename 命令。此命令将文件解密为一个临时文件，并允许编辑。 保存时，它将复制其内容并删除临时文件。

[root@ansible group_vars]# ansible-vault edit webserve

Vault password:

注意：不要用 vi/vim 直接编辑加密文件，这样会导致加密文件无法打开

加密现有的文件

使用 ansible-vault encrypt filename 命令。此命令可取多个想要加密文件的名称作为参 数。

//创建加密文件

```
[root@ansible group_vars]# vim webserve
```

ans ible_pas sword: 123456 //[root@ansible group_vars]# ansible-vault encrypt webserve

New Vault password:

```
Confirm New Vault password:
Encryption successful
```

解密现有的文件

使用 ansible-vault decrypt filename 命令永久解密。在解密单个文件时，可使用**- output**选项以其他名称保存解密的文件。

```
[root@ansible group_vars]# ansible-vault decrypt webserve
Vault password:
Decryption successful
[root@ansible group_vars]# cat webserve
ans ible_password: redhat
```

更改加密的密码

使用 ansible-vault rekey filename 命令更改加密文件的密码。此命令可一次性更新多个 数据文件的密钥。它将提示提供原始密码和新密码。

```
[root@ansible group_vars]# ansible-vault rekey webserve
Vault password:                    〃当前密码
New Vault password:                〃输入要更改的新密码
Confirm New Vault password:        〃再次输入要更改的新密码
Rekey successful
```

在使用 vault 密码文件时，请使用-new-vault-password-file 选项

```
ansible-vault rekey 一一 new-vault-password-file=NEW_VAULT_PASSWORD_FILE webserve playbook 和 ansible vault
```

要运行可访问通过 Ansible Vault 加密的文件的 playbook,需要向 ansible-playbook 命令 提供加密密码。如果不提供密码，playbook 将返回错误

```
[root@ansible httpd]# ansible all -i inventory -m ping
ERROR! Attempting to decrypt but no vault secrets found
```

- 要为 playbook 提供 vault 密码，可使用- vault-id ©prompt 选项。

```
[root@ansible httpd]# ansible 192.168.86. 132 -i inventory —vault-id ©prompt -m Ping _____
Vault password (default):
192. 168. 86. 132 | SUCCESS => (
    〃ansible_facts″: (
        /z di sc over ed_i nt erpr et er_pyt hon'z: 〃/usi7bin/python〃
    ),
    ″changed”: false,
    〃 · 〃 〃 〃
     ping : pong
)
```

- 也可使用- vault-pas sword-file 选项指定以纯文本存储加密密码的文件。密码应当在 该文件中存储为一行字符串。由于该文件包含敏感的纯文本密码，因此务必要通过文件 权限和其他安全措施对其

加以保护。

ansible all —vault-password-file=vault-pw-file webserve

- 也可以使用 ANSIBLE_VAULT_PASSWORD_FILE 环境变量，指定密码文件的默认位置。从 Ansible2. 4 开始，可以通过 ansible-playbook 使用多个 Ansible Vault 密码。要使用 多个密码，需要将多个- vault-id 或- vault-password-file 选项传递给 ansible - playbook** 命令。

ansible all --vault-id one@prompt —vault-id two@prompt webserve

〃这种用法一般不会使用

注意：©prompt 前面的 vaultIDone 和 two 可以是任何字符，甚至可以完全省略它们。不过， 如果在使用 ansible-vault 命令加密文件时使用-vault-id id 选项，则在运行 ansible-playbook 时，将最先尝试匹配 ID 的密码。如果不匹配，将会尝试用户提供的其他 密码。没有 ID 的 vaultID@prompt 实际上是 default@prompt 的简写，这意味着提示输入 vaultIDdefault 的密码。

变量文件管理的推荐做法

- 设置 Ansible 项目，简化管理，使敏感变量和其他变量保存在相互独立的文件中。然后， 包含敏感变量的文件可通过 ansible-vault 命令进行保护。
- 管理组变量和主机变量的首选方式是在 playbook 级别上创建目录。group_vars 目录通 常包含名称与它们所应用的主机组匹配的变量文件。host_vars 目录通常包含名称与它

们所应用的受管主机名称匹配的变量文件。

- 除了使用 group_vars 和 host_vars 中的文件外，也可对每一主机组或受管主机使用目 录。这些目录可包含多个变量文件，它们都由该主机组或受管主机使用

```
[root@ansible httpd]# tree .

I --- files
I --- group_vars
       [1]---- webserve
I I——vars                          〃不需要加密的变量
I          [1]-----  vault          〃需要加密的变量，用 ansible-vault 进行加密
| --- host_vars
| --- inventory
[1]---- playbook, yml
```

管理事实

描述 Ansible 事实

setup 模块用来收集事实，每个 play 在执行第一个任务之前会先自动运行 setup 模块来收 集事实。

Ansible 事实是 Ansible 在受管主机上自动检测到的变量。事实中包含有与主机相关的信息， 可以像 play 中的常规变量、条件、循环或依赖于从受管主机收集的值的任何其他语句那样 使用。

收集的事实可能包括：

- 主机名称
- 内核版本
- 网络接口
- IP 地址
- 操作系统版本
- 各种环境变量
- CPU 数量

•提供的或可用的内存

- 可用磁盘空间

借助事实，可以方便地检索受管主机的状态，并根据该状态确定要执行的操作。

- 可以根据含有受管主机当前内核版本的事实运行条件任务，以此来重启服务器
- 可以根据通过事实报告的可用内存来自定义 MySQL 配置文件

•可以根据事实的值设置配置文件中使用的 IPv4 地址

查看为受管主机收集的事实的一种方式是，运行一个收集事实并使用 debug 模块显示

ansible facts 变量值的简短 playbooko

```
-name: fact
  hosts: 192. 168. 86. 132
  tasks:
    -name: fact
      debug:
        var: ansible_facts
```

//Playbook 以 JSON（字典）格式显示 ansible_facts 变量的内容。

[root@ansible ']# ansible-playbook play, yml

Ansible 事实的变量名

事实	变量
• 短主机名	ansible_facts[a [4]hostname *]
• 完全限定域名	nsible_facts[an fqdn,]
• IPv4 地址	sible_facts['default_ipv4'] ['address,]
• /dev/vdal 磁盘分区的大小	ansible. '_facts['devices,] [Wda,] [*partitions*] ['vdal'] 'size']
• DNS 服务器列表	ansible_facts[, dns,] [*nameservers *]
• 当前运行的内核版本	ansible_facts['kernel']

示例： 取得
主机名

```
-name: fact
  hosts: 192. 168. 86. 132
  tasks:
    -name: fact
      debug:
        var: ansible facts [' hostname[5] ]
```

[root@ansible、]# ansible-playbook play, yml

//取出 sda 下的 sdal 的分区大小

```
-name: fact
  hosts: 192. 168. 86. 132
  tasks:
    -name: fact
      debug:
        var: ansible facts[' devices' ] [' sda' ] [' partitions' ] [' sdaT ] [' size']
```

[root@ansible ']# ansible-playbook play, yml

如果变量的值为散列/字典类型，则可使用下面这种语法来获取其值

ansible_facts [, dns,] ['nameservers']也可以写为 ansible_facts. dns. nameservers, 不推荐使用这种方法

//取得 DNS server 变量 -name: fact

```
  hosts: 192. 168. 86. 132
  tasks:
    -name: fact
      debug:
        var: ansible_facts. dns. name servers [root@ansible ']# ansible-playbook play, yml
```

在 playbook 中使用事实时，Ansible 将事实的变量名动态替换为对应的值 //取得机器名和对应的 ip

```
-hosts: all
tasks:
-name: Prints various Ansible facts debug:
msg: >
        The machine and ipv4 address of (( ansible (( ansible_facts ['machine'] )} is
 facts. default ipv4. address })
```

[root@ansible ']# ansible-playbook play, yml

Ansible 事实作为变量注入

在 Ansible2. 5 之前，事实是作为前缀为字符串**ansible_**的单个变量注入，而不是作为 ansible_facts 变量的一部分注入。例如，ansible_facts['distribution']事实会被称 为 ansible_distribution。

Ansible 事实名称对比

Ansible facts 形式	1 日事实变量形式
ansible facts[*hostname*]	ansible hostname
ansible facts[, fqdn,]	ansible fqdn
ansible_facts[' default_ipv4 '] ['address,]	ansible_default_ipv4['address']
ansible facts[, interfaces,]	ansible_interfaces
ansible facts[, dns,] ['nameservers']	ansible dns['nameservers *]

- 目前，Ansible 同时识别新的事实命名系统（使用 ansible_facts）和旧的 2. 5 前“作 为单独变量注入的事实”命名系统。
- 如果将 Ansible 配置文件的**[default]部分中 inject_facts_as_vars 参数设置为 False**,可关闭旧命名系统。默认设置目前为 True。
- 在未来的版本中旧的写法可能会默认更改为 False,所有建议用新的形式 [root@ansible ']# vim /etc/ansible/ansible. cfg # inject_fact s_as_vars = True

关闭事实收集

关闭收集事实的原因可能有：

- 不准备使用任何事实
- 希望加快 play 速度
- 希望减小 play 在受管主机上造成的负载
- 受管主机因为某种原因无法运行 setup 模块（如：受管主机可能是路由器、交换机等设 备）
- 需要安装一些必备软件后再收集事实

将 gat he r f acts 关键字设置为 no 可以禁用 setup -hosts: all

```
  gather_facts: no
  tasks:
  -name: Prints various Ansible facts
    debug:
      msg: >
        The machine and ipv4 address of (( ansible_facts['machine'] ))
        is (( ansible_facts. default_ipv4. address ))
```

[root@ansible、]# ansible-playbook play, yml

即使 play 设置了 gather_facts: no, 也可以随时通过运行使用 setup 模块的任务来手动收 集事实

```
-hosts: all
  gather_facts: no
  tasks:
  -name: start setup
    setup:
  -name: Prints various Ansible facts
    debug:
      msg: >
        The machine and ipv4 address of (( ansible facts['machine'] ))
```

```
        is (( ansible facts. default ipv4. address ))
[root@ansible、]# ansible-playbook play, yml
```

创建自定义事实

- 除了使用系统捕获的事实外，我们还可以自定义事实，并将其本地存储在每个受管主机 上。
- 这些事实整合到 setup 模块在受管主机上运行时收集的标准事实列表中。
- 它们让受管主机能够向 Ansible 提供任意变量，以用于调整 play 的行为

•自定义事实可以在静态文件中定义，格式可为 INI 文件或采用 JSONo 它们也可以是生 成 JSON 输出的可执行脚本，如同动态清单脚本一样

- 默认情况下，setup 模块从各受管主机的**/etc/ansible/facts. d 目录下的文件和脚本 中加载自定义事实。各个文件或脚本的名称必须以.fact**结尾才能被使用。
- 动态自定义事实脚木必须输出 JSON 格式的事实，而口必须是可执行文件。

使用 **INI** 格式编写自定义事实，INI 格式的自定义事实文件包含由一个部分定义的顶层值，后跟用于待定义的事实的键值对 [webserves]

webl = 192. 168. 11. 110

web2 = 192. 168. 66.66 使用 **JSON** 格式编写自定义事实，JSON 数据可以存储在静态文本文件中，或者通过可执行脚 本输出到标准输出

```
〃webserves〃: (
  〃webl”: "192.168.11.110",
  〃web2”: 〃 192. 168. 66. 66〃
```

•自定义事实文件不能采用 playbook 那样的 YAML 格式。JSON 格式是最为接近的等效格 式

- 自定义事实由 setup 模块存储在 ansible_facts. ansible_local 变量中。事实按照定义它们的文件的名称来整理 -hosts: all

```
  tasks:
  -name: Prints various Ansible facts
    debug:
      var: ansible facts[' ansible local' ] [' cwt' ] [' webserves' ] [' webl']
[root@ansible、]# ansible-playbook play, yml
```

使用魔法变量

一些变量并非事实或通过 setup 模块配置,但也由 Ansible 自动设置，我们称之为魔法变量，魔法变量可用于获取受管主机相关的信息。常用的有 4 个：

魔法变量 说明

- hostvars 包含受管主机的变量，可以用于获取另一台受管主机的变量的值。

如果还没有为受管主机收集事实，则它不会包含该主机的事实。

- group_names 列出当前受管主机所属的所有组
- groups 列出清单中的所有组和主机
- inventory_hostname 包含清单中配置的当前受管主机的主机名称。

因为各种原因有可能与事实报告的主机名称不同

可以使用 debug 模块报告特定主机的 groups 变量的内容

```
[root@ansible ansible all -m debug -a ’ var=groups’ 关于魔法变量的更多信息可以参考官方文档：
```

https://docs. ansible. com/ansible/latest/reference_appendices/special_variables. html#special-variables

playbook 剧本的编写实例精讲

实例：包含多个 play 的 playbook

vim tow. yml

```
—

-name: Execute NODE1          #一个 play
  hosts: node1                  #执行的主机组名
  tasks:                        #任务列表

    - name: install httj        #描述任务
      apt:                    #模块名
        name: apache2         #安装的包名
        state: present          #允许安装
    - name: Start service httpd, if not started #描述任务
      service:                #模块名
        name: apache2           #启动的服务名
        state: started           #启动的服务
-name: Execute NODE2
  hosts:
    - node2
  tasks:
    - name: install gcc
      apt:
        name: gcc
        state: present
```

执行:

```
ansible-playbook —syntax-check tow. yml          #检查 yml 语法
ansible all -m shell -a 〃apt update〃z           #执行更新 apt 源
ansible-playbook tow. yml                        #运维 playbook 的 yml 文件
curl 192. 168. 200. 11 〃验证可以访问到          #验证
```

实例：循环

同时启动 ssh 和 apache vim Circulates, yml -name: xhlx

```
hosts: node1
tasks:
  -name: sshd and apache2 are running
    service:
      name: 〃{{ item }}〃
      state: started
```

```
    loop:
      -sshd
      -apache2
```

ansible-playbook --syntax-check Circulates, yml

ansible-playbook Circulates, yml

实例：变量及条件判断

变量 my_service 已定义，如果定义了，安装相应的软件包，没定义则跳过、也不会报错 vim t jpd. yml -

name: panduan

```
hosts: all
vars:         #定义变量名
  my_service: apache2

tasks:
  -name: "{{ my_service }} package is installed"apt:
      name: "{{ my_service }}"
    when: my_service is defined
```

判断路径：

```
-hosts: all
  remote_user: root
  gather_facts: no
  vars:
    testpathl: "/etc"
    testpath2: "/haha"
  tasks:
  -debug:
      msg: "directory"
    when: testpathl is directory -debug:
      msg: "file"
    when: testpath2 is file
```

15.4 项目实战

1.通过模块批量完成主机的功能实现

环境说明：

一台物理机：pserver 192.168. 1. 254

三台虚拟机：

serverl 192. 168. 1. 10

server2 192. 168. 1. 20

server3 192. 168. 1. 30

要求：

- 配置 ssh：从物理机可以 root 免密登陆三台虚拟机；
- 配置 ansible 主机列表，serverl 在 db 组，server2 和 server3 在 web 组;
- 配置四台服务器的 hosts 文件，加入四台主机名和 IP;
- 使用 ping 模块测试四台主机的连通性；
- 使用 apt 模块在 db 组安装 mariadb-server；
- 使用 shell 模块获得 web 组服务器的 mac 地址；

2.编写剧本，根据企业环境部署 web 服务并进行优化

环境说明：

一台物理机：pserver 192.168. 1. 254

三台虚拟机：

serverl 192. 168. 1. 10

server2 192. 168. 1. 20

server3 192. 168. 1. 30

要求：

- 配置 ssh：从物理机可以 root 免密登陆三台虚拟机；
- 配置 ansible 主机列表，serverl 在 db 组，server2 和 server3 在 web 组;
- 配置四台服务器的 hosts 文件，加入四台主机名和 IP;

- 编写 web. yaml,在 web 组服务器中安装 apache2;在 db 组中安装 mariadb-server;
- 并启动安装的服务，检测服务的可用性；

3.编写剧本，完成嵌套循环，添加多用户

环境说明：

一台物理机：pserver 192.168. 1. 254

三台虚拟机：

serverl 192. 168. 1. 10

server2 192. 168. 1. 20

server3 192. 168. 1. 30

要求：

- 配置 ssh：从物理机可以 root 免密登陆三台虚拟机；
- 配置 ansible 主机列表，serverl 在 db 组，server2 和 server3 在 web 组;
- 配置四台服务器的 hosts 文件，加入四台主机名和 IP;
- 编写 useradd. yaml,使用循环，在所有虚拟机添加 tios[1-3]三个用户

4.编写剧本，完成 debug 检测

环境说明:

一台物理机：pserver 192. 168. 1.254 三台虚拟机：

serverl 192. 168. 1. 10
server2 192. 168. 1. 20
server3 192. 168. 1. 30

要求：

- 配置 ssh：从物理机可以 root 免密登陆三台虚拟机；
- 配置 ansible 主机列表，serverl 在 db 组，server2 和 server3 在 web 组;
- 配置四台服务器的 hosts 文件，加入四台主机名和 IP;
- 编写 mac.yaml,取得所有虚拟机 mac 地址（192. 168. 1. XX 所在网卡）
- 使用 debug var 获得命令输出

第 16 章 Shell 脚本

16. 1 Shell 简介

什么是 Shell

SHELL 即系统的外壳，相对于内核(kernel)来说，它负责人机之间的字符界面的接口，在 第二部分《2.1.2 SHELL 基本概念》节，我们已经介绍过 SHELL 的概念，本节我们主要说 SHELL 脚本的编写。

16.2 Bash 功能介绍

命令历史、命令别名、管道与重定向

命令历史
history
显示执行过的命令(保存的命令条数受系统环境变量 HISTSIZE 的制约)

定义或显示别名
alias: alias [-p][名称[=值]...]

- 不带参数时，'alias'以可重用的格式
  'alias 名称二值'在标准输出设备上打印别名列表。
- 否则，对于每个给定值的名称定义一个别名。
  值末尾的空格会使下一个词被检测作为别名替换展开。

示例： alias rm='rm -i 执行 rm 命令时自动带上-i 参数，即删除都需要确认

管道符号 I：将前一个命令的执行结果作为后一个命令的执行参数

```
cat /etc/passwd | grep root cat /etc/passwd | grep root
```

重定向
Linux 标准输入设备是键盘，标准输出设备是显示器，标准错误输出指的是显示器

设备	设备名	文件描述符	类型
键盘	/dev/stdin	0	标准输入
显示器	/dev/stdout	1	标准输出
显示器	/dev/stderr	2	标准错误输出

输出重定向：把要输出到显示器的内容，输出到文件中； > 表示覆盖写入〉〉追加写入 示例：

```
echo uos > test
cat test
echo bing > test
cat test
```

```
echo txuos » test
cat test
```

错误重定向：2>错误重定向 示例：

```
wadwadwad 2> test
cat test
```

双重输出重定向

```
find / -user uos > test
cat test
find / -name passwd >file 2> test          #将正确的输出结果与错误输出结果一次性单独地送到不同的地方
cat test
cat file
```

如果用户将不管是正确输出还是错误输结果都送到同一个指定的地方则可使用“&>或 &>〉”来完成。

```
find / -name passwd &> test
cat test
```

脚本里使用，将错误结果也输入进文件里 find / -name passwd > test 2>&1 cat test

输入重定向：不使用标准输入端口输入文件，而是使用指定的文件作为标准输入设备；使用“〈”来重定向输入源；使用。让系统将一次键盘的全部输入，先送入虚拟的'当前文档 然后一次性输入。

```
cat〉ok « EOF #交互式
123
456
EOF #结束符
cat ok
```

16.3 变量

自定义变量

变量用于存放程序运行中产生的变化的数值。 如：

```
a=10
a=$((a+1))
echo $a
```

变量的使用范围

每个变量都有其定义的使用范围，按使用范围，可以分为环境变量、全局变量、局部变量等

- 环境变量：指在系统运行的环境中有效的变量。
- 全局变量：一般指在 shell 脚本中全局有效的变量。
- 局部变量：一般指在 shell 脚本里的某个函数中有效的变量。

环境变量

- 环境变量：指系统启动后，设置的一些变量，分为全局环境变量，和用户环境变量。
- 全局环境变量：使用 export 变量名=值的方式定义，在系统运行过程中都有效，或在 /etc/profile 中定义。
- 用户环境变量：使用 变量名=值，只在用户登陆后的 shell 里有效，或在 、/. bash_profile 中定义。

定义文件：cat /etc/profile V- bash_profile

全局文件为/etc/profile,对所有用户有效；用户文件为 V bash_profile,仅对指定的用 户有效。

echo $HOSTNAME 环境变量 PSL 表示 Shell 环境的一级提示符，即命令行提示符(\u 用户名、\h 主机名、\W 工作目录、\$权限标识)

```
echo $PS1
$(debian chroot:+($debian chroot))\u@\h:\w\$
env
set
```

注：

- env：显示所有的环境变量
- set：显示所有本地定义的 shell 变量
- export：把一个变量变作环境变量。

位置变量

位置变量：指脚木在执行时，作为参数传递给脚木的变量是按照从 0 开始的顺疔标号的，可 以用$〈数字〉的方式引用。

如：

vim location.sh

```
#! /bin/bash
echo   $0      〃脚本的名称
echo   $1      〃第一个参数
echo   $2      〃第二个参数
echo   $*      〃所有参数
echo $#        //所有的综合
echo $$        //当前进程的进程号
echo $?//上一个程序的返回状态码
```

chmod +x location, sh　　〃添加可执行权限

测试：

./location 0 1 2 3 4 5

16.4 Shell 引号

反斜线

反斜线又称为转义符，如 touch /home/uos/Desktop/fi\\e \ 转义符 用于在命令中输入表示具有特殊含义的字符

单引号

单引号：成对使用，表示之内的字符串，没有其他含义，即不进行变量或转义字符替换 touch /home/uos/Desktop/’ fi\e’

双引号

双引号：成对使用，表示在其内的字符串，首先进行变量替换，然后作为一个整体参数来对 待。
touch "$HOME/filel”

反引号

反引号：用于在其它命令执行前，先执行的命令，与$()具有相同的效果 如：

```
for i in ’ls ‡. doc’        #注意这里有反引号
do
    echo $i
done
```

16.5 Shell 脚本

脚本格式范例

范例：

```
vim hello, sh
#!/bin/bash
echo hello shell
exit 0
```

说明：

> 通常，我们使用 vi 或 vim 来编辑 shell 脚本

>通常命名以.sh 为结尾，以方便区分

> 通常第一行写#!/bin/bash,表示使用 bash 来执行，大多数脚本都是以 bash 来执行的

> 中间是需要执行的命令和控制语句，示例中只有一行 echo hello shell

>通常，#后作为注释语句

>最后一行 exit 0 可省略

运行脚本的方式

两种方式

- 给脚本加上执行权限：chmod a+x hello, sh,然后直接运行：./hello, sh ./list, sh //指定相对路径

  /root/lish. sh //指定绝对路径

Shell 脚本简单案例

简单的脚本

```
bash list.sh sh    〃开启子进程 〃
list, sh source    开启子进程执行
list.sx            开启子进程
```

‡或者直接用 bash 来执行，作为“参数”：使用 bash> sh、source 来加载脚本文件

这里指把命令按执行的顺序写在脚本文件里，程序按顺厅执行即可，不需要流程的控制。 案例：编写脚本，分别提示输入用户名、密码，然后自动建立用户名并设置密码。

说明：

- stty 用于打开或关闭终端打入字符的回显
  将回显功能关闭(stty -echo),
  将回显功能恢复(stty echo) 。
- read 用于从终端读入一个变量值

```
vim user.sh
#!/bin/bash
read -p 〃请输入用户名：″username
stty -echo
read -p 〃请输入密码：“ passwd
stty echo
  1 〃〃
echo
useradd -m -s /bin/bash 〃$username〃
echo 〃$username:$passwd″      chpasswd
```

判断语句应用

If 判断语法

(1) if 单分支的语法组成:

```
if 条件测试
then 命令序列
Fi
```

(2) if 双分支的语法组成:

```
if 条件测试
then
命令序列 1
else
命令序列 2
fi
```

(3) if 多分支的语法组成:

```
if 条件测试 1 ;then
命令序列 1
```

elif 条件测试 2; then 命令序列 2

else

命令序列 n

Fi 实例 从键盘读取•个论坛积分，判断论坛用户等级 vim grade, sh

#! /bin/bash

read -p "请输入积分(0- 100) : " JF if [$JF -ge 90] ; then

echo 3JF 分，神功绝世〃

elif [$JF -ge 80] ; then

echo 3JF 分，登峰造极

elif [$JF -ge 70] ; then

echo "$JF 分，炉火纯青〃

elif [$JF -ge 60] ; then

echo 3JF 分，略有小成〃

else

echo 3JF 分，初学乍练〃

fi

条件判断的示例

文件表达式

if file] 如果文件存在

if [-d ...] 如果目录存在

if [-s file] 如果文件存在口非空

if [-r file] 如果文件存在口可读

if [-w file] 如果文件存在口可写

if [-x file 如果文件存在口可执行

整数变低表达式

if [inti -eq int2] 如果 inti 等于 int2
if [inti -ne int2] 如果不等于
if [inti -ge int2] 如果>=
if [inti ~gt int2] 如果〉
if [inti -le int2] 如果<=
if [inti -It int2] 如果<

学符中变址衣达式

If [$a = $b]如果 stringl 等于 string2

字符串允许使用赋值号做等号

if [$stringl != $string2]如果 stringl 不等于 string2

if [-n $string]如果 string 非空(非 0),返回 0(true)

if [-z $string]如果 string 为空

if [$sting]如果 string 非空,返回 0 (和-n 类似)

循环语句应用

for 循环

for in 循环

```
for 变量 in 值 1 值 2 值 3— do
    程序
Done
```

实例打印时间

vim for.sh

```
#!/bin/bash 打印时间
for time in morning noon afternoon evening do
    echo “This time is $time!〃
done
```

for ((初始值；循环控制条件；变量变化))

```
for ((初始值；循环控制条件；变量变化)) do
程序
done
```

实例从 1 加到 100

```
vim num. sh
#!/bin/bash
#从 1 加到 100
s=0
for(( i=l;i 〈=100;i=i+l))
#定义循环 100 次 do
    s=$(($s+$i))
    #每次循环给变量 s 赋值
```

done

echo "The sum of 1+2+.. +100 is : $s" #输出从 1 加到 100 的和 实例循环成套：编写脚本打印 **9*9** 乘法衣

vim 99. sh #!/bin/bash

for i in seq 9

do

for j in seq $i

do

echo -n "$i*$j=$[i*j] "

done

echo

done 输出乘法表:

1*1=1

2*1=2 2*2=4

3*1=3 3*2=6 3*3=9

9*1=9 9*2=18 9*3=27_____ 9*9=81

提示：echo -n 参数可以打印完成后不回车换行，默认 echo 输出完成后会换行，本例中 2*1 后不需要换行，

需要继续输出 2*2 **while iff** 环的语法 while 条件测试 do

执行命令

done

实验 **1** 到 **100** 的和

vim num. sh

```
#!/bin/bash
#
i=0
n=l                                                    〃定义循环变量
while [ $n -It 101 ];do                                    //定义循环条件 n < 101
i=$(( $i + $n ))                                           〃累加
n=$(( $n + 1 ))
done
echo $i
```

实验 **while** 死循环的•般格式:

vim while, sh

#!/bin/bash

while :

do

echo "hello world"

done

chmod +x while, sh

循环的中断和退出

通过 break> continue、exit 在 Shell 脚本中实现中断与退出的功能。

> break 可以结束整个循环；

> continue 结束本次循环，进入下一次循环；

> exit 结束整个脚本

实验 for 循环退出

vim test.sh

```
#! /bin/bash
for i in {1.. 5)
do
    [$i -eq 3 ] && break 〃这里将 break 替换为 continue, exit 分别测试脚本执行 效果
    echo $i
done
echo “Game Over” 实验 for 循环中断
```

编写脚木文件，找出 1-20 内 6 的倍数，并打印她的平方值 vim test, sh

```
#! /bin/bash
for i in (1.. 20)
do
    [$[i%6] -ne 0 ] && continue
    echo $[i*i]
done
```

chmod +x test.sh 实验 while 循环退出 计算数字的和

vim sum. sh

```
#! /bin/bash
SUM=0
while :
do
```

```
    read -p 〃请输入整数（0 表示结束）：” x
    [$x -eq 0 ] && break
    SUM=$[ SUM+x ]
done
echo ″总和是：$SUM”
```

```
# chmod +x sum. sh
# . /sum. sh
```

Case 分支控制语句应用

Case 语句

Case 的语法格式

case 语句匹配一个值或一个模式，如果匹配成功，执行相匹配的命令。

```
case 值 in
模式 1)
    command1
. .
9 9
模式 2)
    command1
. .
9 9
*)
    command1
. .
9 9
esac
```

实殴脚本提示输入 **1** 到 **4,** 与每一种模式进行匹配

```
vim case.sh
#! /bin/bash
read -p 请您输入您今天要吃什么：NUM
if [ ″$?” -ne 0 ]
then
    echo 〃请您输入{1|2|3}”
    exit 1
fi

case $NUM in
      1)
        echo 小二，来一碗米饭″
      . .
      9 9
      2)
        echo 小二，来一碗面条〃
      . .
      9 9
```

3)

echo 小二，来一锅包子″

esac

Shell 函数应用

使用 shell 函数

在 Shell 脚本中，将一些需重复使用的操作，定义为公共的语句块，即可称为函数。通过使 用函数，可以使脚本代码更加简洁，增强易读性，提高 Shell 脚本的执行效率

函数的定义方法

格式 1：

function 函数名 {

命令序列

格式 2：

函数名（） { 命令序列

实例：函数调用

1） 任务需求及思路分析

用户在执行时提供 2 个整数参数，这个可以通过位置变量$1、$2 读入。 调用函数时，将用户提供的两个参数传递给函数处理。

颜色输出的命令：echo -e 〃\033[32mOK\033[0nr。3X 为字体颜色，4X 为背景颜色。

2） 根据实现思路编写脚本文件

vim mycolor. sh #! /bin/bash cecho () (

echo -e ″\033[$1m$2\033[0nT cecho 32 OK

cecho 33 OK

cecho 34 OK

cecho 35 OK

chmod +x my color, sh 实验：Shell 版本的 fork 炸弹

```
    # vim test, sh #! /bin/bash
.0(
        .1.&
)
```

15.6 综合项目

环境说明：

一台物理机： pserver 192.168.1.254

三台虚拟机：

serverl	192. 168. 1.
server2	10
server3	192. 168. 1.

要求：

- 配置 ssh：从物理机可以 root 免密登陆三台虚拟机；
- 配置 ansible 主机列表，serverl 在 db 组，server2 和 server3 在 web 组；
- 配置四台服务器的 hosts 文件，加入四台主机名和 IP；

1. 快速自动化安装配置 DHCP 服务器脚本

要求：

- 编写 shell 脚本，自动安装 dhcp 服务器应用
- 自动配置监听网卡为系统第一块网卡，ip 为 192. 168. 1. 254/24
- 配置分配网段为 192. 168. 1. 0/24
- 配置分配 IP 范围为 192. 168. 1. 100~192.168. 1. 200
- 配置分配网关为 192. 168. 1. 254
- 配置分配掩码为 255. 255. 255. 0
- 配置分配 DNS 为 114. 114. 114. 114

要求 2：

- 编写 ansible playbook,命名为 dhcp. yaml,完成上述功能；

2. 检查密码，如果用户三次输入密码均错误，则退出脚本

要求：

- 编写 shell 脚本，分别从键盘读入用户输入的用户名和密码；
- 判断用户名为 uos,密码为 uos_123,不正确则重新输入；
- 直到 3 次机会都用完，仍不正确则执行退出 exit 11；

- 正确则显示 welcome

要求 2：

- 使用 shell 模块，使此脚木在 serverl 上运行；

3. 备份 MariaDB 数据库

要求：

- 在 serverl 安装 mariadb,建立 tiosdb,用户名：uos,密码：uos
- 将 mysql 库 user 表内容及表结构复制到 uosdb 的 uostable 表中；
- 编写脚本，每天晚上 12 点 0 分执行备份；
- 使用 mysqldump 备份 uosdb,设：mysql server : serverl,用户名：uos,密码：uos
- 备份文件压缩为 gz 格式，加 rsyncable 参数
- 备份文件存放于/backup 目录下，文件名为 uosdb+年月日.gz 格式；
- 在本机保留 30 天以内的备份文件；

要求 2：

- 编写 ansible playbook,命名为 mariadb. yaml,完成上述功能；

第 17 章 wine 应用迁移

17. 1 Wine 服务概述

Wine 服务概述和工作原理

Wine （" Wine Is Not an Emu la tor " 的首字母缩写）是一个能够在多种 POSIX-compliant 操作系统（诸如 Linux, mac OS 及 BSD 等）上运行 Windows 应用的兼容层。Wine 不是像 虚拟机或者模拟器一样模仿内部的 Windows 逻辑，而是將 Windows API 调用翻译成为动态 的 POSIX 调用，免除了性能和其他一些行为的内存占用，让你能够干净地集合 Windows 应 用到你的桌面。

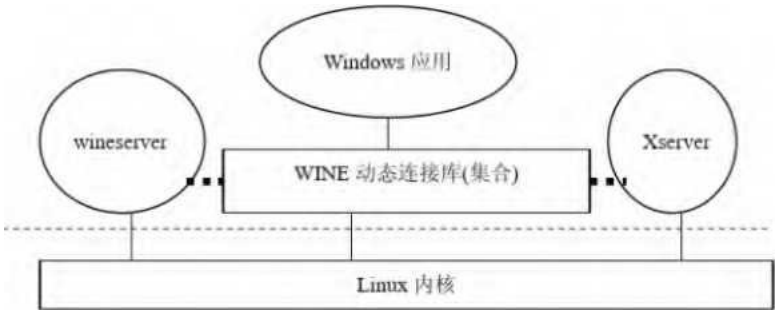
注意：

- Wine 不是模拟 Windows 的工具，而是运用 API 转换技术实做出 Linux 对应到 Windows 相对应的函数来调用 DLL （动态链接库）以运行 Windows 程序。
- Wine 可以工作在绝大多数的 UNIX 版本下，包括 Linux, FreeBSD,和 Solaris。另外， 也有适用于 Mac OS X 的 Wine 程序。
- Wine 不需要 Microsoft Windows 系统，因为这是一个完全由白分之白-的免费代码组成 的。如果有可利用的副本的话，它也可以随意地使用本地系统的 DLLso Wine 的发布是 完全公开源代码的，并口是免费发行的。（基于 LGPL 发布：GNU 宽通用公共许可证）

Wine 服务的系统结构和兼容性

wine 系统结构

Wine 是 Windows 应用软件与 Linux 内核之间的适配层，体现为一个 Wine 服务进程 （wineserver）和一组动态连接库（相当于 Windows 的众多 DLL）。此夕卜，Wine 的 GUI 用户界面 仍依赖于 XII,由动态连接库 xlldrv 和 XII 服务进程构成「其中 xlldrv 是作为 Wine 与 XII 之间的界面，而 XII 服务进程本来就存在，因为 GNOME 也要通过 XII 服务进程操作图形界面。



这样，在运行某个 Windows 应用时，系统中至少有 3 个进程与之有关：

- 应用进程本身 c 所有对 DLL 的调用均在该进程的上下文中运行。需要得到 Wine 的服务， 或者通过 Wine 间接提供的其他（特别是内核）服务时，应用进程经由 Wine 所提供的各种 动态连接库逐层往下调用。在 Wine 内部，这个进程往往需要通过 socket 和 pipe 与 Wine

服务进程通信，以接受服务进程的管理和协调；另一方面又可能经由另一个动态连接库 xlldrv 通过别的 socket 与 XII 服务进程通信，向其发送图形操作请求并接收键盘和鼠 标输入。

- Wine 服务进程。其主要的作用有：提供 Windows 进程间通信与同步的手段；Windows 进程和线程的管理，注册表服务，包括文件在内的各种 W32“对象”的管理；等等。
- XII 服务进程。图形的显示，以及键盘和鼠标输入。

实际上，具体的 Windows 应用进程是从另一个进程，即 Wine 的作业装入程序 wine 转化过来 的。用户一开始时启动的是 wine,而具体的 Windows 程序名是个参数，是由 wine 为具体目 标程序的运行建立起与 Wine 服务进程的连接、装入目标程序，并转入目标程序运行的。

Wine 特性与兼容性

Wine 通过翻译系统调用、重建 Windows 系统的目录结构、提供 Windows 系统库的替 代实现、提供系统服务以及提供其它常用重要应用程序（如 Internet Explorer^注册表编 辑器等）的替代实现等方式在类 Unix 平台中重现了 Windows 的运行环境。因没有利用虚 拟化或模拟等技术，其性能颇佳。

wine 起始于 1993 年，虽然仍有诸多不足，但 Wine 已相当成熟。前不久，知名游戏发布 平台 Steam 的厂商 Valve 还发布了基于 Wine 的开源项目 Proton,后者可协助将 Steam 客户端及 Windows 平台下的游戏运行于 Linux 系统。

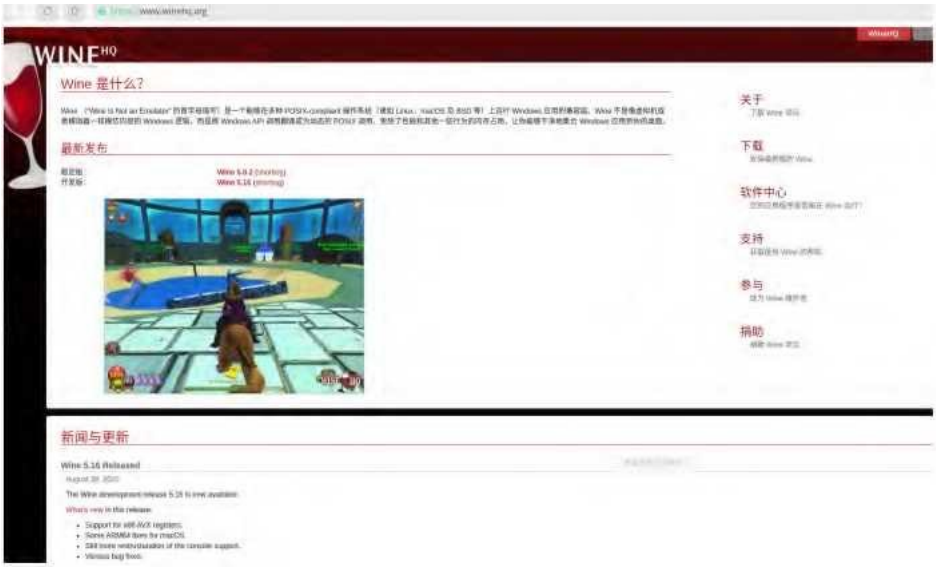
Wine 最显著的特性有：

- 对 Vulkan 的支持，Vulkan 是一套年轻的跨平台的 3D 图形计算 API。
- 对 Direct3D 12 的支持，Direct3D 是 Windows 平台的 3D 图形计算 API, D3D 12 是 随着 Windows 10 发布的最新大版本。
- 为 D3D 10 及 11 版本实现了更多特性。
- 支持游戏控制器（如游戏手柄等）
- 在 Android 中支持 High-DPI。
- .NET Mono 引擎更新至 4. 7. 5 版本。

Wine 项目官网： https:/’7’ww.winehq. org/

Wine 目前最新发布的稳定

版是 5.0.2,开发版 5. 16



Wine 服务的安装和部署

- UOS 系统中有两种版本的 wine 可以安装：deepin-wine 和 wine4. 0. 2
- UOS 默认安装的是 deepin-wine,目前也有两个版本：deepin-wine 和 deepin-wine5

建议直接使用默认安装的 deepin-wine5,如有其它需要安装 wine4. 0. 2,可以：

```
apt install -y wine
```

17.2 Wine 配置与应用

如何修改 Wine 字体

百度搜索 windows 字体进行下载，或者从安装好的 windows 系统中搜索 simsun.ttc
将 ttc 的字体文件拷贝至 V. wine/drive_c/windows/Fonts
新建文件 **vim zh. reg,** 内容如下

```
REGEDIT4
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\FontSubstitutes]
Arial = simsun
```

'ArialCE, 238"="simsun‘

'ArialCYR, 204"="simsun‘

'ArialGreek, 161"="simsun‘

'ArialTUR, 162"="simsun'

CourierNew = siinsun

CourierNewCE, 238"="simsun‘

CourierNewCYR, 204"="simsun‘

CourierNewGreek, 161"="simsun‘

CourierNewTUR, 162"="simsun'

"Fixe dSys"="s i msun"

Helv = simsun

Helvetica - simsun "MS Sans Seri f$^{f/=//}$simsun$^{/z}$

"MS Shell Dlg"="simsun"

"MS Shell Dig 2’ ’ =’ ’ simsun$^{/z}$

bystem - simsun

lanoma = simsun limes = simsun "Times New Roman CE, 238"="simsun”

"Times New Roman CYR, 204"="simsun” "Times New Roman Greek, 161"="simsun” "Times New Roman TUR, 162"="simsun” 1ms Kinn = simsun

在 zh. reg 目录下，终端命令：

regedit zh. reg

命令执行成功后无任何反应，表示成功

如何使 Wine 的软件可以连接互联网

- 在现有 windows 下找到 url. dll 以及 urlmon. dll 文件
- 拷贝至~/. wine/drive_c/windows/system32/ 目录下
- 重启计算机

Wine 服务的基本配置和实例应用

当安装 c原版的 wine 后： winecfg 〃可以对 wine 进行设置

WineifiB

wine staskmgr 〃任务管理器

任务管理器

又件住)选项(Q) 囱口 (W) 兩助(H)

成用程序 | 里程| 性拒|

透程数：6

内存占用：OkB/OkB

wine ***. exe //exe 软件安装
wine not epad, exe //启动记事本应用程序
wine regedit 〃注册表
wine notepad 〃记事本
wine wmplayer //wm 播放器
wineboot //重启 wine
wine uninstaller 〃卸载软件

Winetricks 一个图形化安装配置 wine 的工具(前提是安装原版的 wine),默认的 WINE 环 境缺少很多 WIN 平台的 DLL 库、字体等，导致安装和启动 windows 程序失败，而找到这些摩 和字体是很麻烦的事，winetricks 正是为这个而生的。

安装

apt install winetricks

启动

winetricks

如果知道需要安装的库名:

winetricks directmusic

17.3 项目实战

为更好满足工作对软件的需求，使用 Wine 技术安装软件程序

示例一：安装酷狗音乐

下载： http ″download, kugou. com/ 在下载目录，右键打开终端，执行：
deepwin-wine5 . /kugou9144. exe 〃文件版本号请根据实际情况修改

安装完毕双击桌面图标，启动酷狗

注：第一次启动会发现因缺少字体而显示有问题，按照《4.2.2 节中如何安装字体》配置 simsun 字体后，重启即可

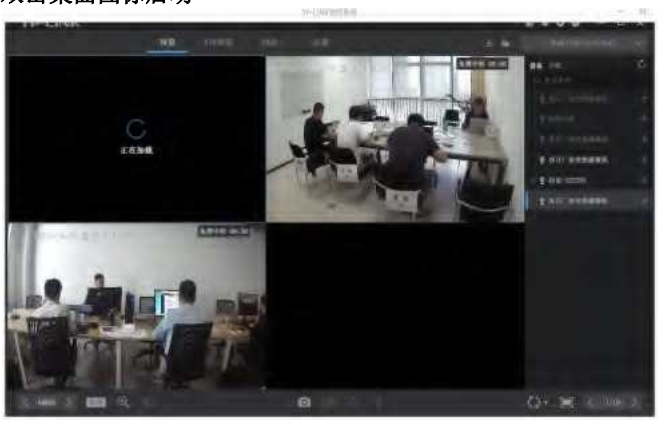
示例二：Wine 安装 tp-link 安防:

下载

https://service. tp-link. com. cn/detail_download_6764.html **安装**
deepin-wine . /TP-LIXK 安防系统_2. 10. 8. 145. exe

注意：安装过程中报建立桌面图标出错，不必理会，点击右键关闭安装程序 或 ctrl+C 停止 deepin-wine 的安装过程即可。

双击桌面图标启动



示例三：安装腾迅会议

下我
https://meeting. tencent. com/download-center. html?froni=1001

安装（请根据实际修改下载的文件名）

```
deepin-wine5 TencentMeeting_0300000000_2. 0. 0. 447. publish, exe
```

运行

登陆后，开个会议试试吧～

编辑桌面启动文件：

deepin-editor ~/. local/share/app 1 ications/wine/Programs/腾讯软件/腾讯会议/腾讯 会议.desktop

```
[Desktop Entry]
Name 二腾讯会议
Exec=env WINEPREFIX="/data/home/yanght/. wine" deepin-wine5 C: \\\\Program\\
FilesWWTencent\\\\WeMeet\\Wwemeetapp. exe
Type=Application
StartupNotify=true
Path=/data/home/yanght/. wine/dosdevices/c:/Program Files/Tencent/WeMeet
Icon=6389_wemeetapp. 0
S tart up WC1 as s=weme et app. exe
```

PS.可以从日志查询启动故障

```
  tail -f /var/log/syslog
```