2016年9月12日

随笔

往事如风，这些只言片语，或许对你的学习有一些参考价值

目录

[前言 4](#_Toc470786951)

[1、LAMP架构原理及搭建详解 5](#_1、LAMP架构原理及搭建详解)

[1.1、何为LAMP 5](#_Toc470786953)

[1.2、LAMP的架构以及通信过程 6](#_Toc470786954)

[1.2.1、LAMP的架构： 6](#_Toc470786955)

[1.2.2、lamp流程： 7](#_Toc470786956)

[1.2.3、A,M,P是怎么联动起来工作的呢： 7](#_Toc470786957)

[1.3、问题集锦 8](#_Toc470786958)

[2、GitHub 10](#_Toc470786959)

[2. git branch develop创建develop分支 14](#_Toc470786960)

[3. git checkout develop切换到develop分支 15](#_Toc470786961)

[4. 保证本地develop分支最新 15](#_Toc470786962)

[5. 基于本地develop创建你自己的开发分支 15](#_Toc470786963)

[6. 进行你该干的事情吧 16](#_Toc470786964)

[7. 将你修改的内容提交到本地版本库 16](#_Toc470786965)

[8. 再次保证你当前分支版本为最新的 18](#_Toc470786966)

[9. 将你的分支提交到远程仓库 18](#_Toc470786967)

[10. 将远程仓库中你的分支合并到远程仓库的develop中（origin/develop） 18](#_Toc470786968)

[11. 解决冲突 18](#_Toc470786969)

[12. 如何用本地版本库文件覆盖工作目录文件 19](#_Toc470786970)

[13. 将commit到本地版本库的内容撤销 19](#_Toc470786971)

[3、静态web网页设计 20](#_Toc470786972)

[4、div+css 20](#_Toc470786973)

[5、js 21](#_Toc470786974)

[6、php 22](#_Toc470786975)

[1、换行符 23](#_Toc470786976)

[2、foreach数组遍历 24](#_Toc470786977)

[3、php5.4以上版本htmlspecialchars输出为空问题 26](#_Toc470786978)

[1.苦逼的修改所有用到htmlspecialchars地方的程序 27](#_Toc470786979)

[2.直接修改源码，重编译！这也是目前我在线上做的方案。 28](#_Toc470786980)

[3.表单提交中Get和Post方式的区别有5点 28](#_Toc470786981)

[7、mysql 30](#_Toc470786982)

[1、mysql远程访问权限问题 30](#_Toc470786983)

[2、mysql版本问题 30](#_Toc470786984)

[3、sql语法 32](#_Toc470786985)

[8、github 33](#_Toc470786986)

# 前言

一年前的事，如今又重新来一遍，才知道当时做了多少工作，幸好，当时的系统镜像没有删除，而如今只是做一些修修补补的事情，但就是这样，还是感到有些力不从心，连最基本的命令都需要重新学习，还好基础不错。

每一次重来，势必都会有一些收获，linux这东西就是这么让人琢磨不定，魅力这东西没体会到，折磨人到是真的。

有个习惯，就是喜欢把遇到的问题记录下来，无论是给自己还是给别人，做个参考吧。

Lamp这套组合还真是经典，可是比较搞笑的是，我居然连怎么装centos都不知道了，基础模式最小安装，呵呵居然还要百度一下，分区更是搞笑，居然还要分boo/,有意义吗，交换分区还是要的。

Linux安装完了，apache、mysql、php通常是下载安装包，而我还是选择了最难的源码安装模式，因为这个模式需要做的工作非常的多，从编译方式，到启动方式，都需要你手动配置，这个真的很麻烦，也很复杂。

因为这个编译是以前编译的，也就是这个软件都安装完了，但我需要从新配置。

第一个问题就是apache的问题，因为是源码安装，rpm service这样的命令都查不到，复习一下：

rpm –qa |grep apache或者httpdld 无结果

service –list-all 无结果

好在当时的工作做的比较扎实，把这些东西都做到系统服务里了，这样就有办法查看了，要不就只能一个一个手动查询了。

chkconfig –list

看到了久违的apache mysql php-fpm，而且都已启动，说明他们都在正常工作，那就测试一下吧，

2了吧唧的我直接输入地址测试apache，就是报403 404这种错误，仔细研究才发如下问题：

1 apache主目录搞错 导致不能访问 需要编辑httpd.conf

2 mysql管理工具phpMyAdmin无法访问，主要是config.inc.php和config.defalut.php配置问题

3 高级功能不全，导入create\_phpmyadmin.sql

4 短密码问题，cookie问题

也不知道哪来的那么多问题，反正不顺利。

好在这些问题最后还是都解决掉了，网上的问题答案太不靠谱了，全是扯犊子，就说短密码问题，还是在老外的帖子里解决的，这个短密码不是随便敲几个字符就可以的，而是最少要46个字符，这个错误才能取消掉。奶奶的，网上的教程有多少是随便敲几个就可以，还有甚者居然说空格都行，这些教程都是抄袭的吧，根本就没有人测试，太坑了。

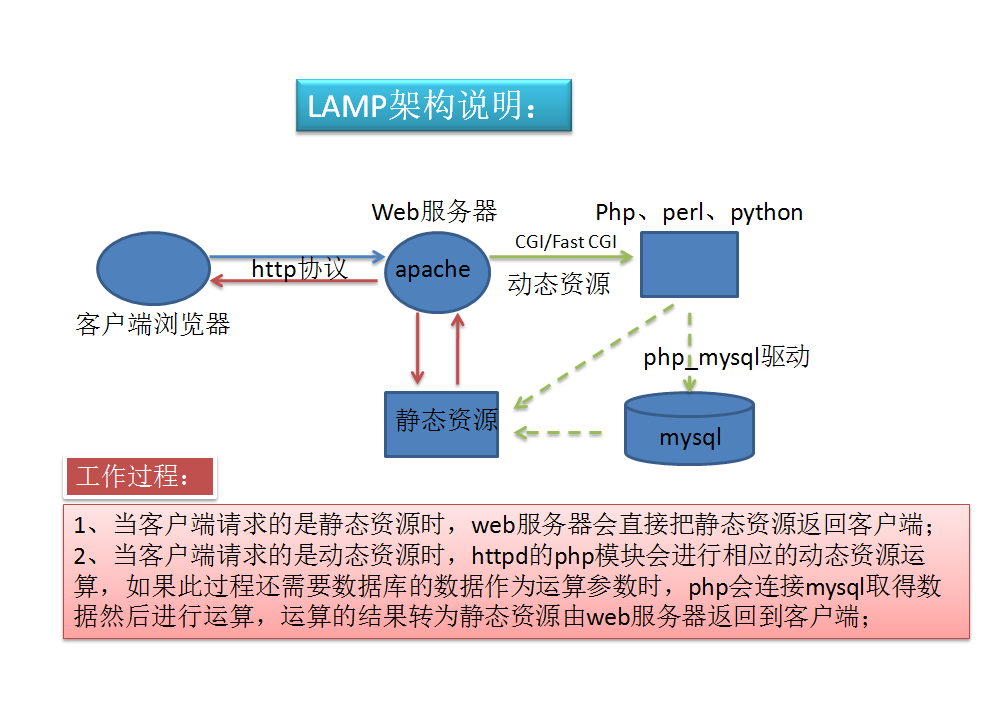
# 1、LAMP架构原理及搭建详解

## 1.1、何为LAMP

LAMP（Linux-Apache-MySQL-PHP）网站架构是目前国际流行的Web框架，该框架包括：Linux操作系统，Apache网络服务器，MySQL数据库，Perl、PHP或者Python编程语言，所有组成产品均是开源软件，是国际上成熟的架构框架，很多流行的商业应用都是采取这个架构，LAMP具有通用、跨平台、高性能、低价格的优势，因此LAMP无论是性能、质量还是价格都是企业搭建网站的首选平台。

## 1.2、LAMP的架构以及通信过程

### 1.2.1、LAMP的架构：



LAMP是一个多C/S架构的平台，最初级为web客户端基于TCP/IP通过http协议发起传送，这个请求可能是动态的，也可能是静态的。

所以web服务器通过发起请求的后缀来判断，如果是静态的资源就由web服务器自行处理，然后将资源发给客户端。如果是动态这时web服务器会通过CGI（Common Gateway interface）协议发起给php。

这里如果php是以模块形式与Web服务器联系。那么他们是通过内部共享内存的方式。如果是php单独的放置与一台服务器，那么他们是通过sockets套接字监听的方式通信（这又是一个C/S架构）。

这时php会相应的执行一段程序，如果在执行程序时，需要用到数据。那么php就会通过mysql协议发送给mysql服务器（也可以看作是一个C/S架构）。由mysql服务器处理，将数据供给php程序。

### 1.2.2、lamp流程：

1、用户发送http请求到达httpd服务器

2、httpd解析url获取需要的资源的路径，通过内核空间读取硬盘资源，如是静态资源，则构建响应报文，发回给用户

3、如果是动态资源，将资源地址发给php解析器，解析php程序文件，解析完毕将内容发回给httpd，httpd构建响应报文，发回给用户

4、如果涉及到数据库操作，则利用php-mysql驱动，获取数据库数据，返回给PHP解析器。

### 1.2.3、A,M,P是怎么联动起来工作的呢：

#### 1.2.3.1、apache + php结合的方式大概几种：

第一种：把php编译时直接编译成apache的模块、module模块化的方式进行工作(apahce默认的这种方式)。

第二种：CGI、通用网关接口、apache基于CGI跟php通信

第三种：fastcgi、也是一种协议、在这种模块下他们两个是这样结合的：

本来php是做为一个模块或者是php解析器运行的，不是监听在某个套接字上接收别人的请求的，而是让别人调用为一个进程使用的，可能是做为别人的子进程在运行，但是工作在fastcgi这种模块下的php自行启用为一个服务进程，它监听在某个套接字上，随时可以接受来自客户端的请求的，它也是有一个主进程的，为了可以响应多个用户的请求，它会启用多个子进程，这些子进程我们也可以称为工作进程，它也是有空闲进程的，一但有客户请求他马上使用空闲的进程响应客户端的请求，将结果返回给前端的调用者，在php5.3.3版本之前它是没有这个能力了，只能工作在模块和CGI的方式下，而在5.3.3之后这个模块直接被收进php模块中，这种模块就叫php-fpm。

所以在以后编译php时，要想跟apache结合，就要编译成php-fpm，这是基于fastcgi工作的模式，并启动这服务进程，也就意味着他是通过套接字跟前端的调用者通信，既然基于套接字通信了，那么前端的web服务器和后面的php服务器完全可以工作在不同的主机上，实现了所谓的分层机制。

apache不会跟数据库打交道，它是个静态web服务器，跟数据库打交道的是应用程序，作为应用程序的源驱动能够基于某个API跟服务器之间建立会话，而后它会通过我们的mysql语句发送给数据库，数据库再将结果返回给应用程序，不是php进程，而是php进程中所执行的代码。

#### 1.2.3.2、php + mysql的通信：

PHP跟mysql怎么整合起来呢，php又怎么被httpd所调用呢

首先httpd并不具备解析代码的能力，他要依赖于php的解析器，接着php本身不依赖于mysql，他只是一个解析器，能执行代码就OK了，那它什么时候用到mysql呢，如果要在mysql中存数据时才用到mysql，只是当php中有运行mysql语句时才用到mysql。

php语言要想联系mysql，通常用到php的驱动，rpm包的叫php\_mysql，php跟mysql没有一点关系，只有程序员在php中编写mysql语句时才连接mysql来执行sql语句的。

基于php-mysql去连接mysql只使用一个函数mysql\_connect();而mysql\_connect()正是php-mysql提供的一个API，只要指明要连接的服务器即可。

## 1.3、问题集锦

phpmyadmin提示：配置文件现在需要绝密的短语密码(blowfish\_secret)解决方法。

问题：在登录phpmyadmin时，有时会出现“配置文件现在需要绝密的短语密码(blowfish\_secret)”这样的错误提示。

解决方法：到phpmyadmin的安装目录里面找到config.inc.PHP文件，用记事本将其打开，通过记事本的查找功能查找“$cfg['blowfish\_secret']=’’”语句。在其单引号里面随便输入至少46个数字或字母（默认为空），最后保存。

总结：出现这样的错误，原因在于我们将config.inc.php和config.default.php里面的$cfg['Servers'][$i]['auth\_type']设置为cookie（默认为config，但这样不是很安全，所以有必要改为cookie）而$cfg['blowfish\_secret']却是空的。

既然将$cfg['Servers'][$i]['auth\_type']设置为cookie，

那么就应该通过某个cookie来访问它，

这正是通过$cfg['blowfish\_secret']来设置。

所以$cfg['blowfish\_secret']不能为空。

请引起注意：config.inc.php 此文件可能在phpmyadmin 搜索或找不到；或许我本人认为可能是：config.sample.inc 这个文件。

但通过本人在config.default.php 里查找：blowfish\_secret 后在找到的

$cfg['blowfish\_secret'] = ' '; 在单双引号里随意添加 符号 并保存文件即可。

最后在网页浏览器地址栏中输入：http://localhost/phpmyadmin 后按回车键；如果还是还没依然提示：

配置文件现在需要绝密的短语密码(blowfish\_secret)那就请再语言栏中选择其他语言即可显示出：

tools | blowfish generator | engage | wpa cracker

Enter the randomly generated set of characters below into your phpMyAdmin config.inc.php file for the $cfg[‘blowfish\_secret’] variable. You can technically use any phrase you like for the blowfish\_secret variable however the below makes it easy so you don’t have to think of one. Every time you refresh this page a new $cfg[‘blowfish\_secret’] will be randomly generated.

$cfg[‘blowfish\_secret’]: Article relating to a blowfish\_secret variable not entered error and some other phpMyAdmin configuration options.

The documentation states that any combination of characters up to 46 total can be used for the blowfish\_secret variable which is used for cookie authentication. The characters below are generated on the fly using lowercase letters, uppercase letters, numbers, and symbols. Each combination is never logged in any way so you can rest assured that your combination of characters is unique and will not be used by anyone else. Just copy the characters at the bottom of the page and enter them into the config.inc.php file located in the phpMyAdmin home directory on your server which by default on CentOS is /usr/share/phpmyadmin.

Example phpMyAdmin Blowfish Secret Variable Entry:

PHP

/\*

\* This is needed for cookie based authentication to encrypt password in

\* cookie

\*/

$cfg['blowfish\_secret'] =

'{^QP+-(3mlHy+Gd~FE3mN{gIATs^1lX+T=KVYv{ubK\*U0V'; /\* YOU MUST FILL IN THIS FOR COOKIE AUTH! \*/

Copy The Output Below To Use For Your phpMyAdmin $cfg[‘blowfish\_secret’]:

rFzsbs5PD#tjTp]@UDW][u-URHFn68f\*4kmE+CV[

If you would like a different character combination than the one above click the refresh button in your browser to generate a new set of characters.

Are there other online tools you would like to see the Question Defense team develop? Let us know about them here.

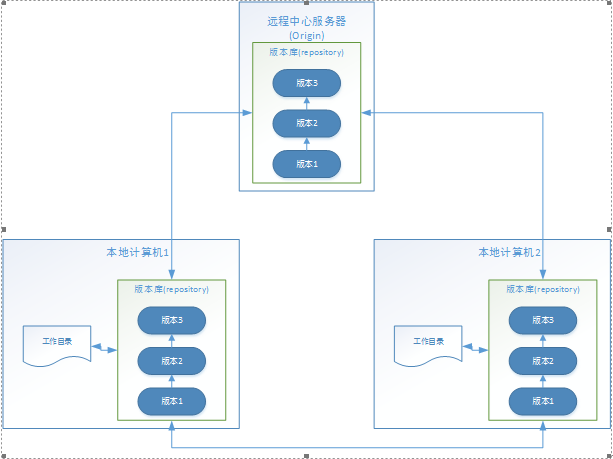
Have questions not answered by the article or this tool then click here to Engage with other technical people.

tools | blowfish generator | engage | wpa cracker

# 2、GitHub

一、Git基本概念

Git是分布式的版本管理系统，每台机器都可以基于远程中心服务器拥有自己的版本，如图：



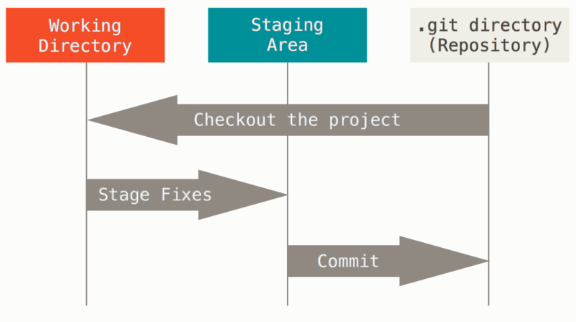
远程中心服务器一般叫做Origin repository。

下面来看一下本地计算机包含哪些内容，图片来自于Git官网，包含三个区域：

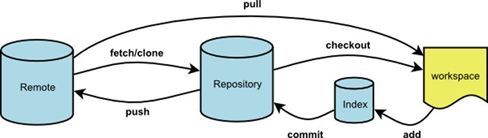
（1）工作目录（working Directory）：我们肉眼能够看到的目录及相应的文件；

（2）暂存区（Staging Area，也有叫Index）：中间过渡区域，可以通过git add命令将工作目录文件放入该区域；

（3）版本库（Repository）：存放版本、分支、Commits的相关信息，通过git commit命令将Staging Area区域的内容放在版本库中；版本库的信息以文件的形式存放在工作目录下的.git隐藏文件夹中；

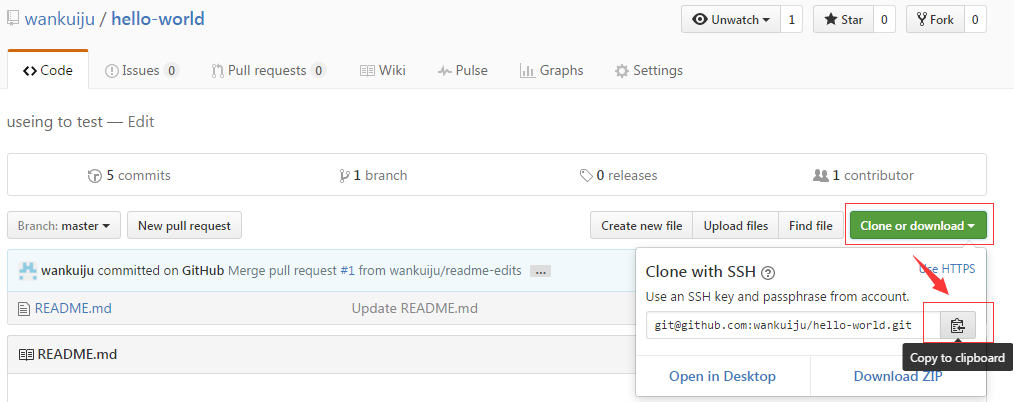


（4）命令集关系



二、Git前期准备

1、找到GitHub中项目地址，点击Clone or download按钮，获取Clone地址



2、由于Clone是通过SSH进行传送的，所以需要生成你机器的认证信息，生成SSH 认证信息，输入如下代码，一路回车：

ssh-keygen -t rsa -b 4096 -C youremail@example.com

具体到我自己就输入：

ssh-keygen -t rsa -b 4096 -C adonis1978@163.com

完成后进入到存储key的文件：

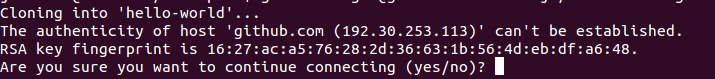
cat ~/.ssh/id\_rsa.pub

copy public key，放在GitHub – setting – SSH and GPG keys

开始Clone，clone前先确认项目存放位置，并进入到相应目录：

juwankui@ubuntu:~/workspace$ git clone git@github.com:wankuiju/hello-world.git

进行clone时会提示如下信息，输入yes，



项目clone完成后，自动创建项目文件夹，我们现在先进入到项目目录：

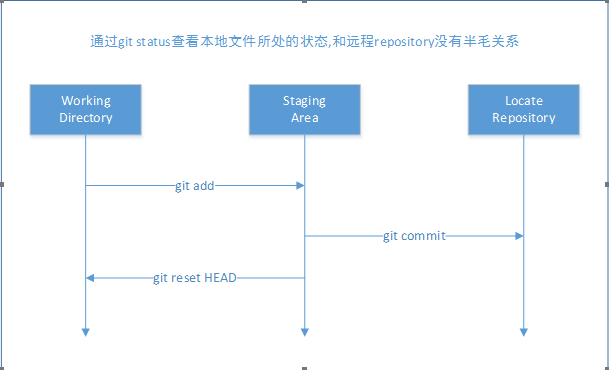
cd hello-world

通过ls –al 可以看到项目目录下存在一个.git的隐藏文件夹,就是我们刚才提到的用于存放当前工作空间的版本信息。

三、不涉及到协作、分支等因素的纯练习内容（不可用于实际项目）

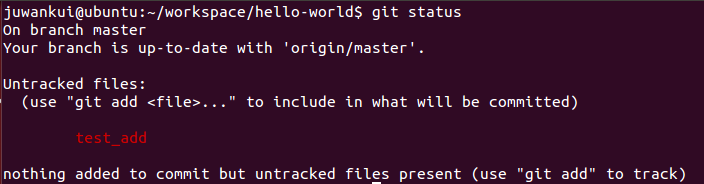
只有在项目第一次从远端下载到本地计算机时才采用clone方式，在后面我会介绍，项目已经存在的情况下，如何根据派发下来的任务进行分支相关的操作，这次暂且不说；

在当前项目目录下新增一个文件名为test\_add的文件，此时我们先不碰触远程版本库，只在本地计算机内玩耍，那么如下图：

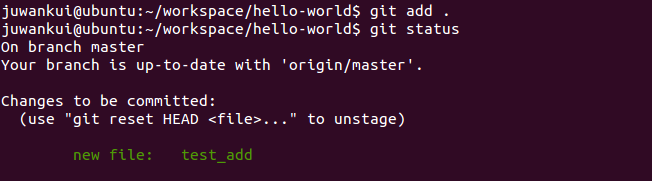


拿刚才我们新建的test\_add文件为例来看一下，文件创建完成后：

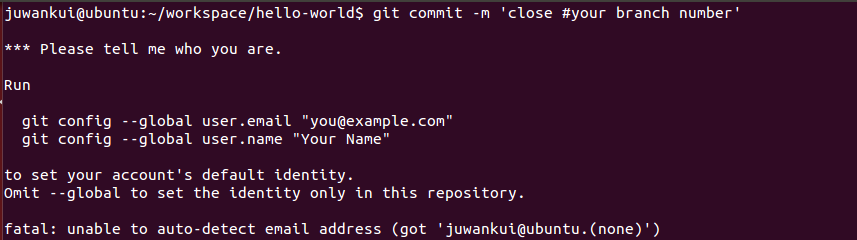
（1）git status看一下本地文件状态，其实git的说明已经非常清晰明了，平时如果出现问题认真看提示说明，就能解决一些问题，如下图告诉我们这个新文件还没有被track，就是没有和版本产生关系，可以通过git add filename，我一般使用 git add .来操作当前目录下所有文件：



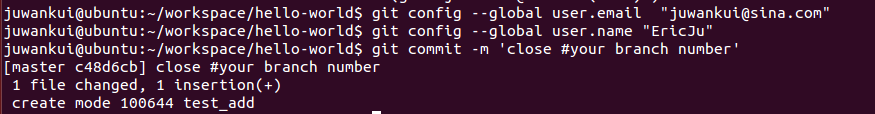
（2）git add 将test\_add纳入版本跟踪，随后再次git status，如下图，文件已经进入staging area：



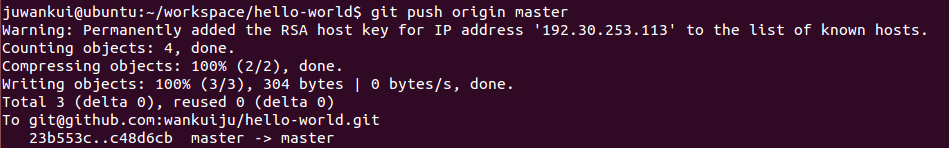
（3）git commit 将test\_add纳入本地版本库，随后执行git status 查看状态



可以看到由于我的本地计算机第一次使用git commit，所以提示需要设置邮箱和用户名（来至GitHub注册信息），按照提示操作，替换成你的信息即可：



（4） 到此，我们在本地玩的很痛快，下一步就需要和远程仓库进行交互了，通过git push将你的版本信息提交到远程仓库（origin repository），我们当前操作的是master分支，一般开发过程中不会直接操作master的。

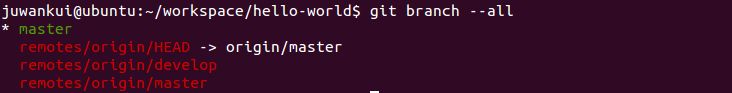


四、 模仿实际项目的操作

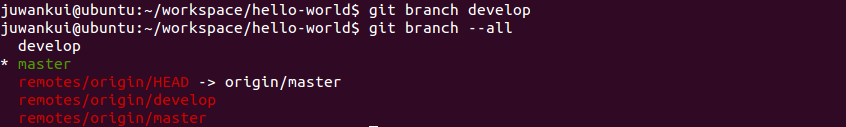
假设我们已经根据上一章的clone方法，将远程仓库（origin repository）中的项目下载本地（默认clone远程的master分支），还有一点需要注意，我们一般不会直接在master主分支上进行项目操作，那怎么办，可以在远程仓库中基于master分支建立一个develop分支，用于开发工作。

1. git branch –all 查看分支

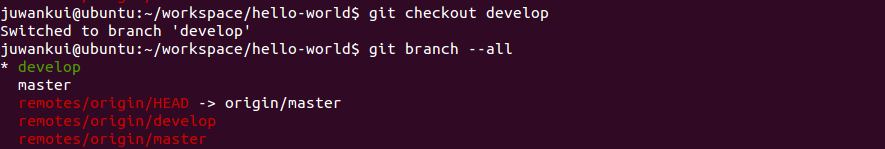
查看远程版本库和本地版本库中的分支信息，目前我本地只有一个分支，分支前面的“\*”代表本地当前分支，远程仓库有两个分支，并且远程仓库当前分支为origin/master（即 远程仓库中的/master分支）：



2. git branch develop创建develop分支



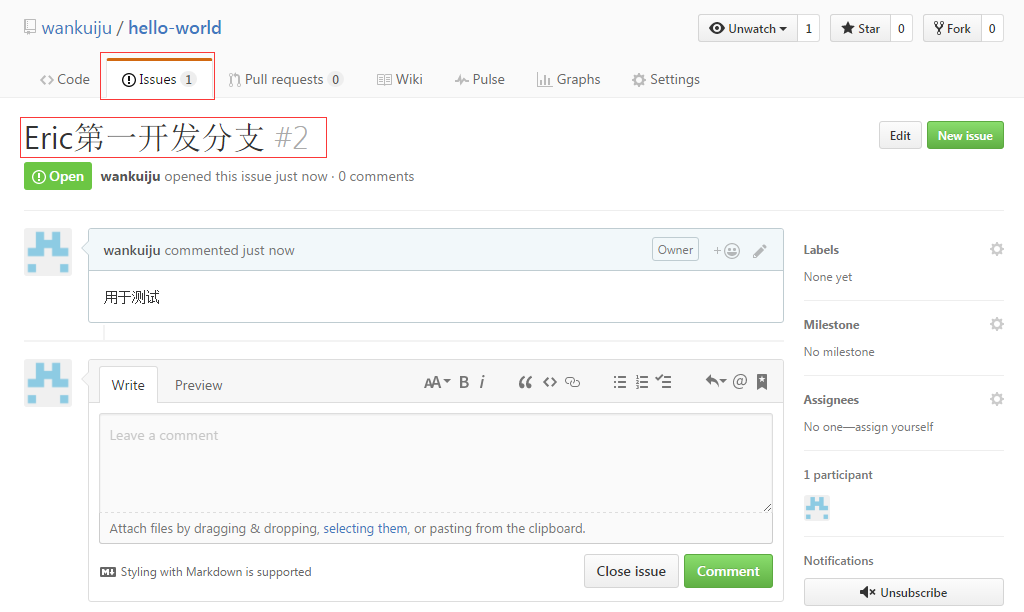
3. git checkout develop切换到develop分支

我们一般使用[**Git**](http://lib.csdn.net/base/git) checkout –b develop 一条语句来实现创建和切换到指定分支的操作，此处是为了演示。   


4. 保证本地develop分支最新

通过执行git fetch（从远程仓库中拉取所有分支信息到本地），git rebase origin/develop（将拉取的origin/develop的内容合并到本地当前分支），这样就保证当前分支develop同步于远程分支。

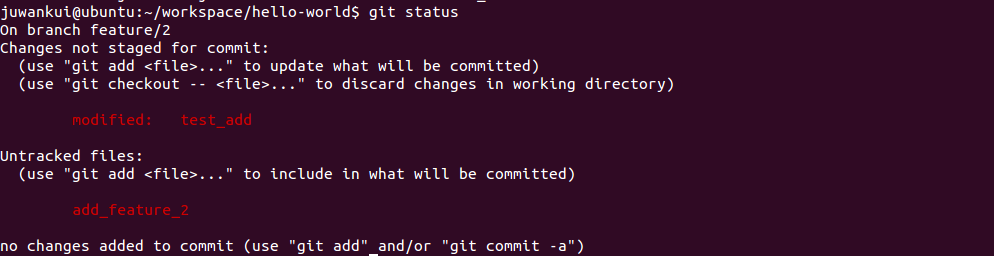
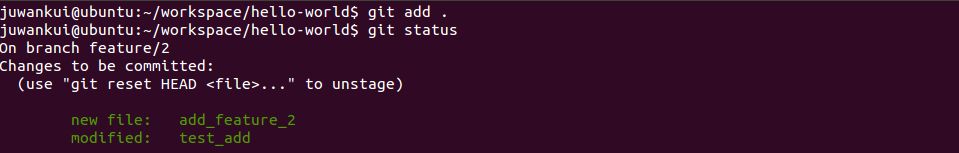
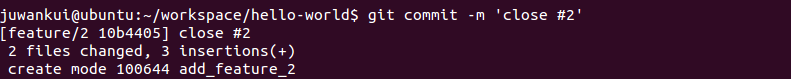
5. 基于本地develop创建你自己的开发分支

此处再次重申，创建你自己的开发分支一定要以develop为基础。   
我们通过git checkout -b feature/2来创建属于我们自己的第一个开发分支（develop开发分支属于大家的），其中“feature”一般为约定俗成的名字,“2”为项目组为你分配的Issues号码，如下图，暂时我们不管Issues操作，只需要知道你任务分支的编码就可。   
  
这里写图片描述

6. 进行你该干的事情吧

真正属于我们的开发分支已经创建完了，现在你可以为所欲为的修改你的代码了。

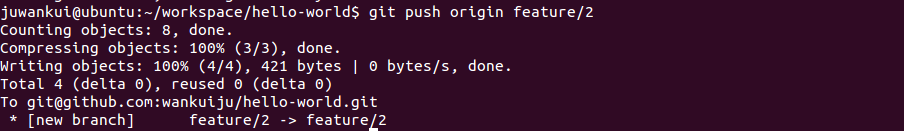
7. 将你修改的内容提交到本地版本库

(1) 新增或修改完内容后通过git status查看一下本地版本库状态   
  
(2) 按照提示,执行git add . (不要忘记后面的”.”哦) 添加当前工作目录文件到暂存区，如果想从暂存区回退到工作目录，可以使用git reset，参考下图   
  
(3) 执行git commit –m ‘close #2’命令将暂存区内容添加到本地仓库，close #2为约定俗成的，即关闭我们当前分支的任务，这只是一个说明代表我们完成了该分支，如果关闭需要进行其他动作，在这，我需要说明一下commit提交方式，git commit -m ‘close #your branch number’①每次使用该语句都会在本地生成一份commit信息，可以通过git log 查看， git commit –amend –no-edit②使用该语句不会创建新的commit信息，会在上一个commit基础上进行覆盖提交，当然，这些commit信息会随着后续push操作，将commit信息随着版本分支一同上传到远程repository，所以要根据自身项目情况决定（通常是第一次commit执行①，以后commit执行②）：   


8. 再次保证你当前分支版本为最新的

通过执行git fetch -p（从远程仓库中拉取所有分支信息到本地，-p就会在本地删除远程已经删除的分支），git rebase origin/develop（将拉取的origin/develop的内容合并到本地当前分支），这样就保证当前分支同步于远程develop分支。

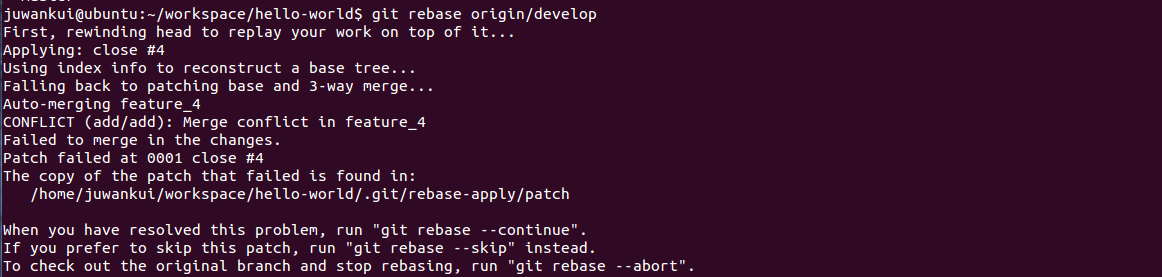
9. 将你的分支提交到远程仓库

通过执行git push origin feature/2将此分支提交到远程仓库，如果已经提交过，在保证你的版本库最新的情况下执行git push origin feature/2 –f   


10. 将远程仓库中你的分支合并到远程仓库的develop中（origin/develop）

一般由管理员或开发经理负责，统一在github上进行操作。

11. 解决冲突

如果远程仓库的develop分支和我们修改了同一个文件，在调用git rebase origin/develop   
命令时就会存在冲突的情况，如下图   
  
(1) 找到冲突文件，并删除冲突内容，主要是<<<<，==== ，>>>>三行的内容   
  
(2) 执行git add . 命令   
(3) 执行git rebase –continue

12. 如何用本地版本库文件覆盖工作目录文件

git checkout – bad\_filename 可以指定分支，默认为当前分支

13. 将commit到本地版本库的内容撤销

通过git log找到要退回的sha（commit后面的一长串字符），记住一定是要退回的sha，不是当前的sha，执行git reset sha（commit后面的一长串字符）

顶

# 3、静态web网页设计

# 4、div+css

这个东西也是必须学的东西，原来的系统用了fromset框架设计的，这种方式首先是速度上比较慢，其次兼容性差一些，所以现在开始改为流行的div布局方式来设计。

CSS属性介绍：

Float: Float属性是DIV+CSS布局中最基本也是最常用的属性，用于实现多列功能，我们知道<div>标签默认一行只能显示一个，而使用Float属性可以实现一行显示多个div的功能，最直接解释方法就是能实现表格布局的多列功能。

Margin: Margin属性用于设置两个元素之间的距离。

Padding: Padding属性用于设置一个元素的边框与其内容的距离。

Clear:使用Float属性设置一行有多个DIV后（多列），最好在下一行开始之前使用Clear属性清楚一下浮动，否则上面的布局会影响到下面。

实例讲解：下面使用实例如果做一个简单又基本的布局，效果如下图：

# 5、js

# 6、php

对于只有 PHP 代码的文件，结束标志(“?>”)最好不存在，PHP自身并不需要结束符号，不添加结束符号可以很大程度上防止末尾被添加额外的注入内容，让程序更加安全。

这个就比较有意思了，作为php的开篇语，居然是一句和大多数介绍php书籍相悖的概念，其实就是这样，很多东西，都是在以后的更新中不断的认知的，而且像php这种解释型语言，本身就不像编译和预编译那样严格。

在 PHP 中，所有用户定义的函数、类和关键词（例如 if、else、echo 等等）都对大小写不敏感。

不过在 PHP 中，所有变量都对大小写敏感。

这个php教程不是一个严格教程，但会就一些常用问题，进行特别说明。

1、函数调用方法

require include

require\_once include\_once

include 和 require 区别 ( 同时 include\_once 和 require\_once 区别)

相同的地方就是，都是可以引入别的页面.

不同的地方是: include 如果出现了错误,会继续执行，require 出现错误，则会终止程序.

证明:

小结: 我们应当使用哪个?

我们做项目的时候，基本上使用的 require\_once

☞ 注意我们的requir\_once / require ... 应当放在 php 最前面.

php\_uname()函数

1. 说明

php\_uname — 返回运行 PHP 的系统的有关信息。

原型：string php\_uname ([ string $mode = “a” ] )。

返回运行php的操作系统的相关描述，和 phpinfo() 最顶端上输出的是同一个字符串。 如果仅仅要获取操作系统的名称。可以考虑使用常量 PHP\_OS，不过要注意该常量会包含 PHP 构建（built）时的操作系统名。

2. 参数

mode 是单个字符，用于定义要返回什么信息：

‘a’：此为默认。包含序列 “s n r v m” 里的所有模式。

‘s’：操作系统名称。例如： FreeBSD。

‘n’：主机名。例如： localhost.example.com。

‘r’：版本名称，例如： 5.1.2-RELEASE。

‘v’：版本信息。操作系统之间有很大的不同。

‘m’：机器类型。例如：i386。

3. 示例

<?php

$sys\_info = php\_uname() . "\n";

if(strtolower(substr($sys\_info,0,5)) == "linux") {

echo "this is a linux system.\n";

}

echo "Host: " . php\_uname('n') . "\n";

?>

## 1、换行符

浏览器识别不了\n或\r\n，这两个换行符是文本换行符，文本文件有效；在网页中查看HTML源代码可以发现代码b成功实现了换行

总结：如果需要将结果输出到浏览器或打印到显示器，代码中使用</br>;如果只是在源代码中换行，则使用\n或\r\n

<?php

echo'hello</br>';

echo'world!';

?>

<?php

echo'hello\n';//unix系统使用\n；windows系统下\r\n

echo'world!';

?>

前一段代码浏览器中成功实现了换行，后面的代码浏览器中未能识别，所以如果要在浏览器中换行，只能使用</br>,如果只是在程序中使用换行，那么可以使用\n或者\r\n。

## 2、foreach数组遍历

在默认情况下，我们的数组元素的下标，是从0开始给你编号，但是实际上，也可以自己指定，比如

$myArray=array("1"=>"val1","2"=>"val2","3"=>"val3");

这样的定义方法实际上数组myArray没有0号元素，而是从1号开始，当然这个自定义的角标，完全可以是任何元素，而不是必须有序。

PHP 4 引入了foreach结构，和 Perl 以及其他语言很像。这只是一种遍历数组简便方法。foreach 仅能用于数组，当试图将其用于其它数据类型或者一个未初始化的变量时会产生错误。有两种语法，第二种比较次要但却是第一种的有用的扩展。

foreach (array\_expression as $value)

　　statement

foreach (array\_expression as $key => $value)

　　statement

第一种格式遍历给定的 array\_expression 数组。每次循环中，当前单元的值被赋给 $value 并且数组内部的指针向前移一步(因此下一次循环中将会得到下一个单元)。

第二种格式做同样的事，只是除了当前单元的键名也会在每次循环中被赋给变量 $key。

先来看第一个语句，这个语句比较简单，array\_expression指的是一个数组表达式，as $value语句将顺序取得该数组的值并保存到$value变量中，此种方法只能取得数组内的值，而不能取得数组的下标索引值。例如：

<?php

$myArray=array("1"=>"val1","2"=>"val2","3"=>"val3");

　　 foreach($myArray as $val) {

　　 print($val." ");

　　 }

其结果会输出：val1 val2 val3

再来看看第二种格式，第二种格式除了能像第一种格式一样得到数组内元素的值外，还能得到元素的索引值，并保存到$key变量中，如果数组的索引值未经过人工设定，则返回系统默认的设定值，看正面例子：

先看一个简单的一维数组：

<?php

$myArray=array("1"=>"val1","2"="val2","3"=>"val3");

　　 foreach($myArray as $key=>$val) {

　　 print($key."=>".$val.";");

　　 }

该程序将出输出：

1=>val1;2=>val2;3=>val3;

<?php

$myArray=array(

　　 "1"=>array("2"=>"val11","12"=>"val12","13"=>"val13"),

　　 "2"=>array("21"=>"val21","22"=>"val22","23"=>"val23"),

　　 "3"=>array("31"=>"val31","32"=>"val32","33"=>"val33")

　　 );

　　 print(" ");

foreach($myArray as $key=>$val) {

　　 print(" ".$key." ");

//判断$val的值是否是一个数组，如果是，则进入下层遍历

if (is\_array($val)) {

　　 print(" ");

foreach($val as $key=>$val) {

　　 print(" ".$key."=>".$val." ");

　　}

　　 print(" ");

　　}

　　 }

　　print(" ");

　　输出结果：

　　?

　　121

　　11=>val11

　　12=>val12

　　13=>val13

　　2

　　21=>val21

　　22=>val22

　　23=>val23

　　3

　　31=>val31

　　32=>val32

　　33=>val33

## 3、php5.4以上版本htmlspecialchars输出为空问题

从旧版升级到php5.4，恐怕最麻烦的就是htmlspecialchars这个问题了！当然，htmlentities也会受影响，不过，对于中文站来说一般用htmlspecialchars比较常见，htmlentities非常少用到。

可能老外认为网页普遍应该是utf-8编码的，于是苦了那些用GB2312，GBK编码的中文站......！

具体表现：

$str = "9enjoy.com的php版本是5.2.10";

echo htmlspecialchars($str);

gbk字符集下输出为空...utf-8下，输出正常。

为什么呢，原因在于5.4.0对这个函数的变化：

5.4.0 The default value for the encoding parameter was changed to UTF-8.

原来是什么呢？

string htmlspecialchars ( string $string [, int $flags = ENT\_COMPAT | ENT\_HTML401 [, string $encoding = 'UTF-8' [, bool $double\_encode = true ]]] )

Defines encoding used in conversion. If omitted, the default value for this argument is ISO-8859-1 in versions of PHP prior to 5.4.0, and UTF-8 from PHP 5.4.0 onwards.

原来是ISO-8859-1，5.4后默认变成utf-8！然后中文使用这个函数就输出为空白了。

国内一堆开源程序在5.4下都会有这样的问题，DISCUZ官方也建议用户不要升级到5.4

解决方案：

### 1.苦逼的修改所有用到htmlspecialchars地方的程序

1.1 其第二个$flags参数，默认是ENT\_COMPAT，因此改成

htmlspecialchars($str,ENT\_COMPAT,'GB2312');

为什么不是GBK？因为没有GBK这个参数，如果强行使用GBK，则报错给你看：

Warning: htmlspecialchars(): charset `gbk' not supported, assuming utf-8

为了能使用GBK，则改成：

htmlspecialchars($str,ENT\_COMPAT,'ISO-8859-1');

1.2.一样是改程序，但可以省略一个参数。

可以在网页头部加

ini\_set('default\_charset','gbk');

然后改成

htmlspecialchars($str,ENT\_COMPAT,'');

文档中有写：An empty string activates detection from script encoding (Zend multibyte), default\_charset and current locale (see nl\_langinfo() and setlocale()), in this order. Not recommended.

大概意思就是：传入空字符串则使用default\_charset的编码

1.3.封装一个函数吧...本来htmlspecialchars这个单词一直不好记。

function htmlout($str) {

return htmlspecialchars($str,ENT\_COMPAT,'ISO-8859-1');

}

然后去批量替换。

### 2.直接修改源码，重编译！这也是目前我在线上做的方案。

修改ext/standard/html.c

大概在372行

/\* Default is now UTF-8 \*/

if (charset\_hint == NULL)

return cs\_utf\_8;

把cs\_utf\_8改成 cs\_8859\_1

/\* Default is now UTF-8 \*/

if (charset\_hint == NULL)

return cs\_8859\_1;

编译后，原程序就不用做任何调整了。

windows下怎么办？这个，自己想办法编译吧，难度比较大...

提供一个网址供参考：http://www.jb51.net/article/63391.htm

引用其一句话：准备好咖啡、可乐，做好准备，可能要折腾数小时

### 3.表单提交中Get和Post方式的区别有5点

1.get是从服务器上获取数据，post是向服务器传送数据。

2.get是把参数数据队列加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。post是通过HTTPpost机制，将表单内各个字段与其内容放置在HTML HEADER内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。

3.对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。

4.get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制。但理论上，IIS4中最大量为80KB，IIS5中为100KB。

5.get安全性非常低，post安全性较高。

# 7、mysql

## 1、mysql远程访问权限问题

每次学习一个新的知识，都会遇到一个环境的搭建问题，这个问题可大可小，为啥这么说呢，比如你要学习linux，你首先需要安装某个版本的linux，并且选择某种安装模式，还需要安装相应的软件和库，你才能正常使用，当然你可以选择全部安装；再或者，你要搭建LAMP，如果你选择一键安装包，这个环境搭建就会变得非常轻松，当然你可能对LAMP环境并不了解，如果你选择安装程序，那么各种配置都需要你手动配置，如果你选择源码安装，那就是啥都要自己动手了，那么为啥搭建一个环境有这么多的方式呢，这个问题问得好，有区别吗，其实是有很大区别的，如果你是一个企业或者商用环境，他们会唯一的选择源码安装模式，而不会选择一键安装或者软件安装，为啥呢？

那是因为，源码安装，是通过本地编译器，根据本地机器编译的，是最适合本地机器的安装包，在效率上至少可以提高5%左右，还有一个重要的原因，就是安全问题，一键安装包，开放了很多服务器不可以开放的功能，这就是为啥源码安装如此麻烦，还要选择他的原因。

言归正传，无论你用那种方式，安装mysql，通常，我们都不是直接操作数据库，因为反复敲那些命令有些烦人，而选择客户端操作是一个好想法，通常客户端有web和客户端软件方式，web方式通常只要把phpmyadmin安装到home目录就可以了，而客户端软件，像sql front这类软件通常都会遇到一个远程访问权限的1130问题，这时你需要给用户开放这个远程访问权限。

## 2、mysql版本问题

还有一个在使用mysql之前必须说的，就是关心mysql版本问题，而带来的语法问题，

TYPE=MyISAM 和 ENGINE=MyISAM 都是设置数据库存储引擎的语句 ，(老版本的MySQL使用TYPE而不是ENGINE（例如，TYPE = MYISAM）。 MySQL 5.1为向下兼容而支持这个语法，但TYPE现在被轻视，而ENGINE是首先的用法。 一般地，ENGINE 选项是不必要的；除非默认已经被改变了，MyISAM是默认存储引擎。

所以直接将原来TYPE=MyISAM 改成ENGINE=MyISAM 就可以了

mysql数据乱码问题

ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO\_INCREMENT=8 ;

mysql有多重引擎，这些引擎都隐藏在mysql服务器内，全都能执行create ,select等命令。多数默认是MyISAM

三种不同的引擎简介：

InnoDB--可靠的事务处理引擎，不支持全文本搜索。

MEMORY--功能等同于MyISAM，数据存储在内存中，速度快，因此多适用于临时表。

MyISAM--性能极高，支持全文本搜索，但不支持事务处理。

latin1百度百科

　　latin1 　　Latin1是ISO-8859-1的别名，有些环境下写作Latin-1。 　　ISO-8859-1 　　ISO-8859-1编码是单字节编码，向下兼容ASCII，其编码范围是0x00-0xFF，0x00-0x7F之间完全和ASCII一致，0x80-0x9F之间是控制字符，0xA0-0xFF之间是文字符号。 　　ISO-8859-1收录的字符除ASCII收录的字符外，还包括西欧语言、希腊语、泰语、阿拉伯语、希伯来语对应的文字符号。欧元符号出现的比较晚，没有被收录在ISO-8859-1当中。 　　因为ISO-8859-1编码范围使用了单字节内的所有空间，在支持ISO-8859-1的系统中传输和存储其他任何编码的字节流都不会被抛弃。换言之，把其他任何编码的字节流当作ISO-8859-1编码看待都没有问题。这是个很重要的特性，MySQL数据库默认编码是Latin1就是利用了这个特性。ASCII编码是一个7位的容器，ISO-8859-1编码是一个8位的容器。

解决方案：

修改sql文件，修改创建表的部分，将CHARSET=latin1 改为 utf8

## 3、sql语法

### 3.1、sql简介

这个sql语句本身是一种标准化的语言，就是数据的操作语言，但由于不同数据库在处理字符的方式上的不同，导致sql语言并不通用，也就是说除了SQL标准中定义的操作外，其它都是不一样的。那么我们为啥不分开学习，这个标准的东西其实是这样的，就像操作系统，首先在数据库理论层上，概念是一致的，不同的只是各家数据库的表达方式，SQL标准规定了数据库的基本动作，这些都是一致的，所有SQL数据库都是支持的，所以先学习它们中公用的部分，也是数据库的基础部分，然后再学习私有部分。

SQL 是一门 ANSI 的标准计算机语言，用来访问和操作数据库系统。SQL 语句用于取回和更新数据库中的数据。SQL 可与数据库程序协同工作，比如 MS Access、DB2、Informix、MS SQL Server、Oracle、Sybase 以及其他数据库系统。

不幸地是，存在着很多不同版本的 SQL 语言，但是为了与 ANSI 标准相兼容，它们必须以相似的方式共同地来支持一些主要的关键词（比如 SELECT、UPDATE、DELETE、INSERT、WHERE 等等）。

注释：除了 SQL 标准之外，大部分 SQL 数据库程序都拥有它们自己的私有扩展！

由于我们使用的是mysql数据库，所以这里说的的都是mysql数据库支持的sql语句，下面看一下它们有什么不同，

/\*创建数据库\*/

CREATE DATABASE IF NOT EXISTS `test`;

/\*选择数据库\*/

USE `test`;

/\*创建表\*/

CREATE TABLE IF NOT EXISTS `user` (name varchar(50), age int);

/\*插入测试数据\*/

INSERT INTO `user` (name, age) VALUES('harry', 20), ('tony', 23), ('harry', 24);

这是一段标准的sql创建数据库和表的操作，但它在mysql数据库中却不能被执行，会报错，原因就是字符格式的问题。修改成如下就可以执行了：

/\*创建数据库\*/

CREATE DATABASE IF NOT EXISTS test;

/\*选择数据库\*/

USE test ;

/\*创建表\*/

CREATE TABLE IF NOT EXISTS user (name varchar(50), age int);

/\*插入测试数据\*/

INSERT INTO user (name, age) VALUES('harry', 20), ('tony', 23), ('harry', 24);

不仔细观察，你会发现没啥区别，实际上我们把数据库名称和表名称的单引号都去掉了，然后它就不会报错了。

类似这种问题，即使在同一种数据库中，不同版本之间也会有很大差异，好在，这些问题，通常都会在文档中发布，所以随时查看文档是一个好习惯。

### 3.2、sql语法

数据库表

一个数据库通常包含一个或多个表。每个表由一个名字标识（例如“客户”或者“订单”）。表包含带有数据的记录（行）。

下面的例子是一个名为 "Persons" 的表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

上面的表包含三条记录（每一条对应一个人）和五个列（Id、姓、名、地址和城市）。

您需要在数据库上执行的大部分工作都由 SQL 语句完成。

下面的语句从表中选取 LastName 列的数据：

SELECT LastName FROM Persons

重要事项

一定要记住，SQL 对大小写不敏感！

SQL 语句后面的分号；

某些数据库系统要求在每条 SQL 命令的末端使用分号。在我们的教程中不使用分号。

分号是在数据库系统中分隔每条 SQL 语句的标准方法，这样就可以在对服务器的一次请求中执行一条以上的语句。

如果您使用的是 MS Access 和 SQL Server 2000，则不必在每条 SQL 语句之后使用分号，不过某些数据库软件要求必须使用分号。

#### 3.2.1、SQL DML 和 DDL

可以把 SQL 分为两个部分：数据操作语言 (DML) 和 数据定义语言 (DDL)。

SQL (结构化查询语言)是用于执行查询的语法。但是 SQL 语言也包含用于更新、插入和删除记录的语法。

查询和更新指令构成了 SQL 的 DML 部分：

SELECT - 从数据库表中获取数据

UPDATE - 更新数据库表中的数据

DELETE - 从数据库表中删除数据

INSERT INTO - 向数据库表中插入数据

SQL 的数据定义语言 (DDL) 部分使我们有能力创建或删除表格。我们也可以定义索引（键），规定表之间的链接，以及施加表间的约束。

SQL 中最重要的 DDL 语句:

CREATE DATABASE - 创建新数据库

ALTER DATABASE - 修改数据库

CREATE TABLE - 创建新表

ALTER TABLE - 变更（改变）数据库表

DROP TABLE - 删除表

CREATE INDEX - 创建索引（搜索键）

DROP INDEX - 删除索引

#### 3.2.2、SQL SELECT 语句

SELECT 语句用于从表中选取数据。结果被存储在一个结果表中（称为结果集）。

SELECT 列名称 FROM 表名称

SELECT \* FROM 表名称

注释：SQL 语句对大小写不敏感。SELECT 等效于 select。

通常我们对sql语句中的固定格式采用大写，变量采用mac风格，就是首单词小写后面单词首字母大写，当然这只是个人风格，为了程序的可读性而已，不是硬性规定。

SQL SELECT 实例

如需获取名为 "LastName" 和 "FirstName" 的列的内容（从名为 "Persons" 的数据库表），请使用类似这样的 SELECT 语句：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

SELECT LastName,FirstName FROM Persons

结果：

|  |  |
| --- | --- |
| **LastName** | **FirstName** |
| Adams | John |
| Bush | George |
| Carter | Thomas |

现在我们希望从 "Persons" 表中选取所有的列。

请使用符号 \* 取代列的名称，就像这样：

SELECT \* FROM Persons

# 8、MariaDB

## 8.1、起源

MariaDB名字的含义　你是不是惊讶为什么使用MariaDB这个奇怪的名字来作为DBMS的名字？实际上，只要解释名字的由来你就觉得合乎情理了。MySQL是在Monty Widenius的女儿My出生后命名的，而不是外界常认为的单词“I”的所有格形式；Monty使用他的儿子的名字Max来命名数据库引擎MaxDB。现在，他最新的MariaDB项目使用他的小女儿的名字Maria来命名。

MariaDB数据库管理系统是MySQL的一个分支，主要由开源社区在维护，采用GPL授权许可。开发这个分支的原因之一是：甲骨文公司收购了MySQL后，有将MySQL闭源的潜在风险，因此社区采用分支的方式来避开这个风险。 MariaDB的目的是完全兼容MySQL，包括API和命令行，使之能轻松成为MySQL的代替品。在存储引擎方面，使用XtraDB（英语：XtraDB）来代替MySQL的InnoDB。 MariaDB由MySQL的创始人Michael Widenius（英语：Michael Widenius）主导开发，他早前曾以10亿美元的价格，将自己创建的公司MySQL AB卖给了SUN，此后，随着SUN被甲骨文收购，MySQL的所有权也落入Oracle的手中。MariaDB名称来自Michael Widenius的女儿Maria的名字，而MySQL是一个关系型数据库管理系统，由瑞典MySQL AB公司开发，目前属于Oracle公司。MySQL是最流行的关系型数据库管理系统，在WEB应用方面MySQL是最好的RDBMS(Relational Database Management System：关系数据库管理系统)应用软件之一。MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。MySQL所使用的SQL语言是用于访问数据库的最常用标准化语言。MySQL软件采用了双授权政策，它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择MySQL作为网站数据库。由于其社区版的性能卓越，搭配PHP和Apache可组成良好的开发环境。

## 8.2、结构

数据库通常分为层次式数据库、网络式数据库和关系式数据库三种。而不同的数据库是按不同的数据结构来联系和组织的。

1.数据结构模型

⑴数据结构

所谓数据结构是指数据的组织形式或数据之间的联系。

如果用D表示数据，用R表示数据对象之间存在的关系集合，则将DS=(D，R)称为数据结构。

例如，设有一个电话号码簿，它记录了n个人的名字和相应的电话号码。为了方便地查找某人的电话号码，将人名和号码按字典顺序排列，并在名字的后面跟随着对应的电话号码。这样，若要查找某人的电话号码(假定他的名字的第一个字母是Y)，那么只须查找以Y开头的那些名字就可以了。该例中，数据的集合D就是人名和电话号码，它们之间的联系R就是按字典顺序的排列，其相应的数据结构就是DS=(D，R)，即一个数组。

⑵数据结构类型

数据结构又分为数据的逻辑结构和数据的物理结构。

数据的逻辑结构是从逻辑的角度(即数据间的联系和组织方式)来观察数据，分析数据，与数据的存储位置无关；数据的物理结构是指数据在计算机中存放的结构，即数据的逻辑结构在计算机中的实现形式，所以物理结构也被称为存储结构。

这里只研究数据的逻辑结构，并将反映和实现数据联系的方法称为数据模型。

比较流行的数据模型有三种，即按图论理论建立的层次结构模型和网状结构模型以及按关系理论建立的关系结构模型。

2.层次、网状和关系数据库系统

⑴层次结构模型

层次结构模型实质上是一种有根结点的定向有序树(在数学中"树"被定义为一个无回的连通图)。下图是一个高等学校的组织结构图。这个组织结构图像一棵树，校部就是树根(称为根结点)，各系、专业、教师、学生等为枝点(称为结点)，树根与枝点之间的联系称为边，树根与边之比为1:N，即树根只有一个，树枝有N个。

按照层次模型建立的数据库系统称为层次模型数据库系统。IMS(Information Management System)是其典型代表。

⑵网状结构模型

按照网状数据结构建立的数据库系统称为网状数据库系统，其典型代表是DBTG(Database Task Group)。用数学方法可将网状数据结构转化为层次数据结构。

⑶ 关系结构模型

关系式数据结构把一些复杂的数据结构归结为简单的二元关系(即二维表格形式)。例如某单位的职工关系就是一个二元关系。

由关系数据结构组成的数据库系统被称为关系数据库系统。

在关系数据库中，对数据的操作几乎全部建立在一个或多个关系表格上，通过对这些关系表格的分类、合并、连接或选取等运算来实现数据的管理。

dBASEⅡ就是这类数据库管理系统的典型代表。对于一个实际的应用问题（如人事管理问题），有时需要多个关系才能实现。用dBASEⅡ建立起来的一个关系称为一个数据库（或称数据库文件），而把对应多个关系建立起来的多个数据库称为数据库系统。dBASEⅡ的另一个重要功能是通过建立命令文件来实现对数据库的使用和管理，对于一个数据库系统相应的命令序列文件，称为该数据库的应用系统。

因此，可以概括地说，一个关系称为一个数据库，若干个数据库可以构成一个数据库系统。数据库系统可以派生出各种不同类型的辅助文件和建立它的应用系统。

### 8.2.1、关系数据库由来：

网状数据库和层次数据库已经很好地解决了数据的集中和共享问题，但是在数据独立性和抽象级别上仍有很大欠缺。用户在对这两种数据库进行存取时，仍然需要明确数据的存储结构，指出存取路径。而后来出现的关系数据库较好地解决了这些问题。

1970年，IBM的研究员E.F.Codd博士在刊物《Communication of the ACM》上发表了一篇名为“A Relational Model of Data for Large Shared Data Banks”的论文，提出了关系模型的概念，奠定了关系模型的理论基础。尽管之前在1968年Childs已经提出了面向集合的模型，然而这篇论文被普遍认为是数据库系统历史上具有划时代意义的里程碑。Codd的心愿是为数据库建立一个优美的数据模型。后来Codd又陆续发表多篇文章，论述了范式理论和衡量关系系统的12条标准，用数学理论奠定了关系数据库的基础。关系模型有严格的数学基础，抽象级别比较高，而且简单清晰，便于理解和使用。但是当时也有人认为关系模型是理想化的数据模型，用来实现DBMS是不现实的，尤其担心关系数据库的性能难以接受，更有人视其为当时正在进行中的网状数据库规范化工作的严重威胁。为了促进对问题的理解，1974年ACM牵头组织了一次研讨会，会上开展了一场分别以Codd和Bachman为首的支持和反对关系数据库两派之间的辩论。这次著名的辩论推动了关系数据库的发展，使其最终成为现代数据库产品的主流。

### 8.2.2、关系数据库的简介

当下各种关系数据库管理系统

ORACLE

Microsoft SQL Server2005

SYBASE

INFORMIX

DB2

INGERS

Microsoft ACCESS

Visual FoxPro

开源关系数据库管理系统

MYSQL 最有名的开源数据库，先后被sun和Oracle收购，目前有闭源可能

MariaDB

简介：

Oracle是一个最早商品化的关系型数据库管理系统，也是应用广泛、功能强大的数据库管理系统。Oracle作为一个通用的数据库管理系统，不仅具有完整的数据管理功能，还是一个分布式数据库系统，支持各种分布式功能，特别是支持Internet应用。作为一个应用开发环境，Oracle提供了一套界面友好、功能齐全的数据库开发工具。Oracle使用PL/SQL语言执行各种操作，具有可开放性、可移植性、可伸缩性等功能。特别是在Oracle 8i中，支持面向对象的功能，如支持类、方法、属性等，使得Oracle 产品成为一种对象/关系型数据库管理系统。

具有良好的兼容性，可移植性，可联结性，高生产率，开放性。oracle的产品主要有数据库服务器，开发工具，联结产品。

Microsoft SQL Server是一种典型的关系型数据库管理系统，可以在许多操作系统上运行，它使用Transact-SQL语言完成数据操作。由于Microsoft SQL Server是开放式的系统，其它系统可以与它进行完好的交互操作。目前最新版本的产品为Microsoft SQL Server 2008，它具有可靠性、可伸缩性、可用性、可管理性等特点，为用户提供完整的数据库解决方案。

SYBASE支持企业内各种数据库应用需求（数据仓库，联机事务处理，决策支持系统，小平台应用），优良的开放结构可以工作在便携机到多处理系统等多种平台上。Sybase数据库的特点：

（1）它是基于客户/服务器体系结构的数据库

　一般的关系数据库都是基于主/从式的模型的。在主/从式的结构中，所有的应用都运行在一台机器上。用户只是通过终端发命令或简单地查看应用运行的结果。而在客户/服务器结构中，应用被分在了多台机器上运行。一台机器是另一个系统的客户，或是另外一些机器的服务器。这些机器通过局域网或广域网联接起来。

客户/服务器模型的好处是：它支持共享资源且在多台设备间平衡负载 ，允许容纳多个主机的环境，充分利用了企业已有的各种系统

（2） 它是真正开放的数据库

　由于采用了客户/服务器结构，应用被分在了多台机器上运行。更进一步，运行在客户端的应用不必是Sybase公司的产品。对于一般的关系数据库，为了让其它语言编写的应用能够访问数据库，提供了预编译。Sybase数据库，不只是简单地提供了预编 译，而且公开了应用程序接口DB-LIB，鼓励第三方编写DB-LIB接口。由于开放的客户DB-LIB允许在不同的平台使用完全相同的调用，因而使得访 问DB-LIB的应用程序很容易从一个平台向另一个平台移植。

（3） 它是一种高性能的数据库

Sybase真正吸引人的地方还是它的高性能。体现在以下几方面：

　　可编程数据库

　　通过提供存储过程，创建了一个可编程数据库。存储过程允许用户编写自己的数据库子例程。这些子例程是经过预编译的，因此不必为每次调用都进行编译、优化、生成查询规划，因而查询速度要快得多。

　　事件驱动的触发器

　　触发器是一种特殊的存储过程。通过触发器可以启动另一个存储过程，从而确保数据库的完整性。

　　多线索化

Sybase数据库的体系结构的另一个创新之处就是多线索化。一般的数据库都依靠操作系统来管 理与数据库的连接。当有多个用户连接时，系统的性能会大幅度下降。Sybase数据库不让操作系统来管理进程，把与数据库的连接当作自己的一部分来管理。 此外，Sybase的数据库引擎还代替操作系统来管理一部分硬件资源，如端口、内存、硬盘，绕过了操作系统这一环节，提高了性能。

Microsoft Access是在Windows环境下非常流行的桌面型数据库管理系统。使用Microsoft Access无需编写任何代码，只需通过直观的可视化操作就可以完成大部分数据管理任务。在Microsoft Access数据库中，包括许多组成数据库的基本要素。这些要素是存储信息的表、显示人机交互界面的窗体、有效检索数据的查询、信息输出载体的报表、提高 应用效率的宏、功能强大的模块工具等。它不仅可以通过ODBC与其它数据库相连，实现数据交换和共享，还可以与Word、Excel等办公软件进行数据交 换和共享，并且通过对象链接与嵌入技术在数据库中嵌入和链接声音、图像等多媒体数据。

DB2数据库是IBM公司的产品，DB2数据库核心又称作DB2公共服务器，DBZ数据库采用多进程多线索体系结构，可以运行于多种操作系统之上，并分别根据相应平台环境作了调整和优化，以便能够达到较好的性能。它支持从个人计算机到UNIX服务器、从中小型机到大型机、从IBM到非IBM（HP及SUN UNIX系统等）各种操作平台。DB2数 据库既可以在主机上以主/从方式独立运行，也可以在客户/服务器环境中运行，其中服务平台可以是OS/400、AIX、OS/2、HP UNIX、SUN Solaris等操作系统，客户机平台可以是OS/2或Windows、Dos、AIX、HP UX、SUN Solaris等操作系统。

DB2数据库的特色如下。

（1）支持面向对象的编程。DB2支持复杂的数据结构，如无结构文本对象，可以对无结构文本对象进行布尔匹配、最接近匹配和任意匹配等搜索。可以建立用户数据类型和用户自定义函数。

（2）支持多媒体应用程序。DB2支持大二分对象（BLOB），允许在数据库中存取二进制大对象和文本大对象。其中，二进制大对象可以用来存储多媒体对象。

（3）支持存储过程和触发器，用户可以在建表时显示地定义复杂的完整性规则。

（4）支持异构分布式数据库访问和支持数据复制。

此外，IBM提供了许多开发工具，主要有Visualizer Query、VisualAge和VisualGen等，通过这些功能很强的可视化应用开发工具，可以大幅度地提高软件的开发效率。

Oracle数据库产品是当前数据库技术的典型代表，Oracle的产品除了数据 库系统外，还有应用系统和开发工具等。目前的Oracle 9是一个面向对象的数据库系统，它既非纯的面向对象的数据库也非纯的关系数据库，它是两者的结合，因此叫做“对象关系数据库”。由于Oracle包括了几 乎所有的数据库技术，因此被认为是未来企业级主选数据库之一。Oracle主要有以下特点。

（1）对象/关系模型。Oracle使用了对象/关系模型，也就是在完全支持传统 关系模型的基础上，为对象机制提供了有限的支持。Oracle不仅能够处理传统的表结构信息，而且能够管理由C++、Smalltalk 以及其他开发工具生成的多媒体数据类型，如文本、视频、图形和空间对象等。这种做法允许现有软件开发产品与工具软件及Oracle应用软件共存，保护了客 户的投资。

（2）动态可伸缩性。Oracle引入了连接存储池和多路复用机制，提供了对大型 对象的支持，当需要支持一些特殊数据类型时，用户可以创建软件插件来实现。Oracle 8采用了高级网络技术，提高共享池和连接管理器来提高系统的可括性，容量可从几GB到几百TB，可允许10万用户同时并行访问，Oracle的数据库中每 个表可以容纳1000列，能满足目前数据库及数据仓库应用的需要。

（3）系统的可用性和易用性。Oracle提供了灵活多样的数据分区功能，一个分 区可以是一个大型表，也可以是索引易于管理的小块，可以根据数据的取值分区，有效地提高了系统操作能力及数据可用性，减少I/O瓶颈。Oracle还对并 行处理进行了改进，在位图索引、查询、排序、连接和一般索引扫描等操作引入并行处理，提高了单个查询的并行度。

（4）系统的可管理性和数据安全功能。Oracle提供了自动备份和恢复功能，改 进了对大规模和更加细化的分布式操作系统的支持，加强了SQL操作复制的并行性。为了帮助客户有效地管理整个数据库和应用系统，Oracle还提供了企业 管理系统，数据库管理员可以从一个集中控制台拖放式图形用户界面管理Oracle的系统环境。

（5）对多平台的支持与开放性。网络结构往往含有多个平台，Oracle可以运行 于目前所有主流平台上，如SUN Solarise、Sequent Dynix/PTX、Intel NT、HP UX、DEC UNIX、IBM AIX等。Oracle的异构服务为同其他数据源以及使用SQL和PL/SQL的服务进行通信提供了必要的基础设施。

MySQL是一个可运行在Windows平台和大多数的Linux平台上的半商业数据库。MySQL的普及很大程度上源于它的宽松，其中MySQL的Windows版本在任何情况下都不免费，而在包括Linux在内的任何UNIX平台下使用MySQL都是免费的。

MySQL客户库是客户/服务器结构的C语言库，它意味着一个客户能查询驻留在另一台机器的一个数据 库。然而MySQL真正的强项在于该库中的语言包装器，Perl、Pathon和PHP只是一部分。Apache的Web服务器也有许多模块，例如目录存 取文件等允许各种各样的Apache配置信息。使用MySQL，应用程序接口简单、一致并且相当完整，而且多平台ODBC驱动程序都能够自由获得。

MySQL包含一些与SQL标准不同的转变，大多数被设计成是对SQL脚本语言的不足的一种补充。但 是另一些扩展确实使 MySQL与众不同，例如LINK子句搜索是自动地忽略大小写的。而且MySQL也允许用户自定义SQL函数，也就是一个程序员可以编写一个函数然后集成 到MySQL中，其表现与任何基本函数没有什么不同。

MySQL可以说是数据库领域的中间派，它缺乏一个全功能数据库的大多数主要特征，但是又有比类似 Xbase记录存储引擎更多的特征。它需要一个服务守护程序，但是不像它们那样消费资源；查询语言允许复杂的连接查询，但是MySQL的主要的缺陷之一是 缺乏标准的参考完整性机制，所有的参考完整必须由程序员强制保证。可能有些数据库管理员会觉得系统缺乏主要数据库功能，但是MySQL对简单数据库是有其 价值的。

开放性:

SQL Server 只能在windows上运行，没有丝毫的开放性，操作系统的系统的稳定对数据库是十分重要的。Windows9X系列产品是偏重于桌面应用，NT server只适合中小型企业。而且windows平台的可靠性，安全性和伸缩性是非常有限的。它不象unix那样久经考验，尤其是在处理大数据量的关键 业务时.

Oracle 能在所有主流平台上运行（包括 windows）。完全支持所有的工业标准。采用完全开放策略。可以使客户选择最适合的解决方案。对开发商全力支持。

DB2 能在所有主流平台上运行（包括windows）。最适于海量数据。DB2在企业级的应用最为广泛,在全球的500家最大的企业中,几乎85%以上用DB2数据库服务器,而国内到97年约占5%.

可伸缩性,并行性

SQL server DB2 并行实施和共存模型并不成熟。很难处理日益增多的用户数和数据卷。伸缩性有限。

Oracle 平行服务器通过使一组结点共享同一簇中的工作来扩展windownt的能力,提供高可用性和高伸缩性的簇的解决方 案。 如果windowsNT不能满足需要, 用户可以把数据库移到UNIX中。

DB2 DB2具有很好的并行性。DB2把数据库管理扩充到了并行的、多节点的环境. 数据库分区是数据库的一部分，包含自己的数据、索引、配置文件、和事务日 志。数据库分区有时被称为节点或数据库节点

安全性

SQL server 没有获得任何安全证书。

Oracle Server 获得最高认证级别的ISO标准认证。

DB2 获得最高认证级别的ISO标准认证。

性能

SQL Server 多用户时性能不佳

Oracle 性能最高，保持windowsNT下的TPC-D和TPC-C的世界记录。

DB2 适用于数据仓库和在线事物处理性能较高。

客户端支持及应用模式

SQL Server C/S结构，只支持windows客户，可以用ADO,DAO,OLEDB ,ODBC连接.

Oracle 多层次网络计算，支持多种工业标准，可以用ODBC, JDBC,OCI等网络客户连接

DB2 跨平台，多层结构，支持ODBC,JDBC等客户

操作简便

SQL Server 操作简单,但只有图形界面.

Oracle 较复杂, 同时提供GUI和命令行，在windowsNT和unix下操作相同

DB2 操作简单,同时提供GUI和命令行，在windowsNT和unix下操作相同

使用风险

SQL server 完全重写的代码，经历了长期的测试，不断延迟，许多功能需要时间来证明。并不十分兼容早期产品。使用需要冒一定风险。

Oracle 长时间的开发经验，完全向下兼容。得到广泛的应用。完全没有风险。

DB2 在巨型企业得到广泛的应用，向下兼容性好。风险小。

## 8.3、存储引擎

### 8.3.1、什么是存储引擎

MySQL中的数据用各种不同的技术存储在文件（或者内存）中，这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并最终提供不同的功能和能力，这些不同的技术以及配套的相关功能在MySQL中称为存储引擎（也称为表类型）。

MySql支持多个存储引擎，包括处理事务安全表的引擎和处理非事务安全表的引擎：

ISAM是一个定义明确且历经时间考验的数据表格管理方法，它在设计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM的两个主要不足之处在于，它不支持事务处理，也不能够容错：如果你的硬盘崩溃了，那么数据文件就无法恢复了。

MyISAM管理非事务表，是ISAM的扩展格式。除了提供ISAM里所没有的索引和字段管理的大量功能，MyISAM还使用一种表格锁定的机制，来优化多个并发的读写操作。它提供高速存储和检索，以及全文搜索能力，受到web开发的青睐。MyISAM在所有MySQL配置里被支持，它是默认的存储引擎。

MEMORY存储引擎提供“内存中”表，被正式确定为HEAP引擎，也处理非事务表。HEAP允许只驻留在内存里的临时表格。驻留在内存里让HEAP要比ISAM和MyISAM都快，但是它所管理的数据是不稳定的，而且如果在关机之前没有进行保存，那么所有的数据都会丢失。在数据行被删除的时候，HEAP也不会浪费大量的空间。HEAP表格在你需要使用SELECT表达式来选择和操控数据的时候非常有用。要记住，在用完表格之后就删除表格。

InnoDB和BDB（BerkleyDB）存储引擎提供事务安全表。尽管要比ISAM和MyISAM引擎慢很多，但是InnoDB和BDB包括了对事务处理和外键的支持，这两点都是前两个引擎所没有的。

NDB Cluster是被MySQLCluster用来实现分割到多台计算机上的表的存储引擎，当前只被Linux， Solaris和Mac OS X支持。

### 8.3.2、如何更换引擎？

Globle：一种最简单的方法就是更改服务器配置，直接将其设置成你所需要的引擎。这个在win下通过更改服务器安装目录下的mysql.ini或my.ini中的default-storage-engine项即可，也可以通过运行MySQL ServerInstance Configuration Wizard做简单的设置。

PerTable:除了全局的方法外，还有一种更灵活的配置方法，那就是按表来设置引擎，这样我们就可以把那些需要用到事务处理的表设置成InnoDB，其他设置成MyISAM,将性能提升到极致，设置方法也比较简单：

创建一个新表时，可以通过在CREATE语句中ENGINE或TYPE选项来告诉MySQL要创建什么类型的表：

CREATE TABLE t (i INT) ENGINE = INNODB;

CREATE TABLE t (i INT) TYPE = MEMORY;

如果省略掉ENGINE或TYPE选项，默认的存储引擎被使用。当MySQL被用MySQL配置向导安装在Windows平台上，InnoDB存储引擎替代MyISAM存储引擎作为默认。当不可用的类型被指定时，自动用InnoDB表来替代。

还可以把表从一个类型转到另一个类型，使用ALTERTABLE语句：

ALTER TABLE t ENGINE = MYISAM;

ALTER TABLE t TYPE = BDB;

当不清楚当前数据库中各表的引擎时可以使用SHOW TABLE STATUS FROMDBname来查看。

### 8.3.3、如何选择存储引擎？

关于这个问题，我们需要考虑每个存储引擎提供了哪些不同的核心功能，一般把这些核心功能分为4类：支持的字段和数据类型、锁定类型、索引和事务处理。

支持的字段和数据类型

虽然所有这些引擎都支持通用的数据类型，例如整型、实型和字符型等，但是，并不是所有的引擎都支持其它的字段类型，特别是BLOG（二进制大对象）或者TEXT文本类型。

锁定类型

锁定机制主要是为了防止多个处理同时更新同一个数据。不同的存储引擎支持不同级别的锁定：表锁定、页锁定和行锁定。

表锁定：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。支持最多的就是表锁定，MyISAM和MEMORY支持这种锁定。MySQL的表级锁有两种模式：表共享读锁和表独占写锁。对MyISAM表的读操作，不会阻塞其他用户对同一表的读请求，但会阻塞对同一表的写请求；对MyISAM表的写操作，则会阻塞其他用户对同一表的读和写操作；MyISAM表的读操作与写操作之间，以及写操作之间是串行的！

行锁定：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。InnoDB表进行行级锁定。

页锁定：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。BerkeleyDB引擎支持页锁定。

索引

建立索引在搜索和恢复数据库中的数据的时候能够显著提高性能。不同的存储引擎提供不同的制作索引的技术。有些存储引擎根本不支持索引。

事务处理

事务处理功能通过提供在向表中更新和插入信息期间的可靠性。

## 8.4、语法

## 8.5、实例

# 9、github

写代码的就不用说了，不过，既然代码可以多人共同进行版本控制，其实写文档不是一样吗，只不过，就是文档的格式不是二进制而已，不过用来做更新中转还是不错的注意。

经常在家或者单位写同一个文档，这时就有一个版本问题，如何能自动同步呢，github是一个不错的选择，下面就github的一些用法做一个简单的介绍吧。

这个工具是linux的创始人开发的，主要是应用在linux内核的版本控制上，其功能和易用性上得到大家的认可，现在也是主要的版本控制软件之一。

言归正传，github的工作模式有3种，有网页版、客户端和命令行。

这几种方式大同小异，相比之下客户端可能是最方便的，但作为教程，这几种方式都会简单介绍一下：

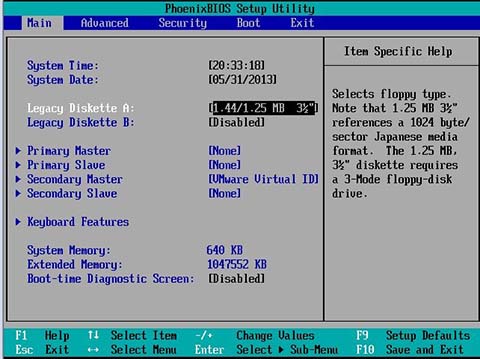
先来说一下网页版的吧，这个应该是功能最全的的，包括仓库的创建，克隆仓库，同步和下载仓库，以及删除仓库等。

# 10、操作系统原理

很多人学过这门课程，我想也同样有很多人学过了之后，还没有计算机到底是如何工作的，操作系统是如何来驱动这些硬件的，bios中断服务，系统内核，设备驱动程序是如何协调工作的，很多人根本就说不清，只知道安装驱动就可以工作了，当然你如果作为一个终端用户，你完全没有必要了解这些知识，而作为开发者，显然这些知识是必须的。

## 10.1、bios

BIOS是英文"Basic Input Output System"的缩略词，直译过来后中文名称就是"基本输入输出系统"。其实，它是一组固化到计算机内主板上一个ROM芯片上的程序，它保存着计算机最重要的基本输入输出的程序、开机后自检程序和系统自启动程序，它可从CMOS中读写系统设置的具体信息。 其主要功能是为计算机提供最底层的、最直接的硬件设置和控制。计算机在运行时，首先会进入BIOS，它在计算机系统中起着非常重要的作用。一块主板性能优越与否，很大程度上取决于主板上的BIOS管理功能是否先进。



BIOS技术源于IBMPC/AT机器的流行以及第一台由康柏公司研制生产的“克隆”PC。在PC启动的过程中，BIOS担负着初始化硬件，检测硬件功能，以及引导操作系统的责任。在早期，BIOS还提供一套运行时的服务程序给操作系统及应用程序使用。BIOS程序存放于一个断电后内容不会丢失的只读内存ROM中；系统断电或被重置 (reset) 时，处理器第一条指令的位址会被定位到 BIOS 的内存中，让初始化程序开始执行。英特尔公司从2000年开始，发明了可扩展固件接口（Extensible Firmware Interface），用以规范BIOS的开发。而支持EFI规范的BIOS也被称为EFI BIOS。之后为了推广EFI，业界多家著名公司共同成立了统一可扩展固件接口论坛（UEFI Forum），英特尔公司将EFI 1.1规范贡献给业界，用以制订新的国际标准UEFI规范。目前UEFI规范的最新版本是2.3.1，英特尔公司曾经预测，2010年，全世界或有有60%以上的个人电脑使用支持UEFI规范的BIOS产品。

CPU一上电代码执行的地址是0FFFFFFF0h，指向的是BIOS Flash。这可在《IA-32 Intel Architecture Software Developer's Manual》的Volume 3A Part1中“FFFFFFF0”中搜索到，前面DelphiGuy()已经贴出来了。然后先对SuperIO、Chipset做一些必要的初始化，之后对低端内存进行检测，如果有错就会Beep，不出错的话就会把BIOS Rom中的内容复制到内存中去，注意只是一部分内容而不是全部。之后就是跳转到内存中去执行了。

记不住是从什么时候开始了（其实是人懒，没去查），X86 CPU一开始的上电地址是通过下面的方式形成的：In realaddress mode, the base address is normally formed by shifting the 16-bit segment selector value 4 bits to the left to produce a 20-bit base address. However, during a hardware reset, the segment selector in the CS register is loaded with F000H and the base address is loaded with FFFF0000H. The starting address is thus formed by adding the base address to the value in the EIP register (that is, FFFF0000 + FFF0H = FFFFFFF0H).

DelphiGuy()好像还有一段没贴出来，就是CS在第一次被重新赋值后，实地址下指令执行地址的形成方式就变为CS左偏4bit加IP了。我理解为是在将BIOS Rom中的内容复制到内存中去时第一次被重新赋值的，因为这时需要改变CS的值进big flat mode。

前面还有人说：

还有一个疑惑，ram中留给system bios的地址空间是64K，可是现在主板bios动辄2M，甚至4M，那么整个real mode内存才1M，请问他怎么来寻址？如果bios让cpu跳转到bios芯片超过64K的地址处，但是此时那384K的UMA中已经没有地址对这个超过64K的地址做映射，那就更别提以后的rom shadow了。想不通！

首先，“主板bios动辄2M，甚至4M”这是指bit，也就是4M bit是512K byte，而且BIOS Flash中还有很多空间是留给Logo、CPU microcode以及ESCD之类的，所以1M足够了。

写得好乱，希望以后多来论坛逛逛能把表达能力提高点，呵呵

关于BIOS的入口地址0xFFFF0

一直都说开机加电之后，将CS设置为0xFFFF，IP设置为0x0000，这样组成的地址就是0xFFFF0，而这个就是BIOS的入口地址，之后CPU读取这个地址的代码，然后就巴拉巴拉的开始执行下去了。

现在想知道这个地址0xFFFF0是在主板上呢，还是在内存上。想着如果是统一编址的话，就可能是去读取主板上的ROM，但是看书上的图，有像是在RAM中，如果是在 内存 中，那是什么时候，由哪段程序将ROM中的程序读取到内存中的。自己猜的话，应该这个过程是固定的硬件实现的，将ROM中的所有内容拷贝到内存中最高位那一段中，之后，0xFFFF0就必然是指内存中的地址了。

百度了一下，有篇文章就是说这个的。

转了过来。原文地址：http://myuka.blog.163.com/blog/static/17498750920115593147503/

640KB~1MB 上位内存（这个区域的地址分配给ROM，相应的384KB的RAM被屏蔽掉。所谓的影子内存技术，就是把ROM内容读取到对应地址的RAM中，以后系统就从RAM中读取数据，而不是从原来的ROM读取数据，从而提高速度。）

1MB~ 扩展内存

注:

Shadow RAM也称为"影子内存",是为了提高计算机系统效率而采用的一种专门技术，所使用的物理芯片仍然是CMOS DRAM(动态随机存取存储器，参阅本书后面的内容)芯片。Shadow RAM占据了系统主存的一部分地址空间。其编址范围为C0000～FFFFF，即为1MB主存中的768KB～1024KB区域。这个区域通常也称为内存 保留区，用户程序不能直接访问。Shadow RAM的功能就是是用来存放各种ROM BIOS的内容。也就是复制的ROM BIOS内容，因而又它称为ROM Shadow，这与Shadow RAM的意思一样，指得是ROM BIOS的"影子"。现在的计算机系统，只要一加电开机，BIOS信息就会被装载到Shadow RAM中的指定区域里。由于Shadow RAM的物理编址与对应的ROM相同，所以当需要访问BIOS时，只需访问Shadow RAM而不必再访问ROM，这就能大大加快计算机系统的运算时间。通常访问ROM的时间在200ns左右，访问DRAM的时间小于100ns、60ns， 甚至更短。

在计算机系统运行期间，读取BIOS中的数据或调用BIOS中的程序模块的操作将是相当频繁的，采用了Shadow RAM技术后，无疑大大提高了工作效率。

386 之前与386之后，这个地址是不同的，但都在系统内存的最高 地址段。在386下为 FFFFFFF0H。因为CS段是16位的，EIP是32位的，为了得到一个32位地址，386给CS段增加了几个字段，这是隐藏的字段，系统可以通过 GDT,IDT将更改段选择子的字段，此时地址转换就不是 段地址左移4位 + 偏移地址，而是CS的Base字段＋偏移地址。

下面是一个例子

当系统加电后，系统会复位。此时在386以前的系统下CS＝F000H，IP＝FFF0H，Bios地址为段地址左移4位 + 偏移地址挤即 F0000H + FFF0H = FFFF0H

在386以前系统可寻址范围为1MB即 00000H~FFFFFH

在386下CS＝F000H，IP＝FFF0H，这是不变的，但是这时，CScs中的内容为:

Selector = F000H （这个就是你可以看到的那部分内容）

这时隐藏的部分是不能用的，因为在实地址模式下，所以Bios地址与386以前的地址一样，

但是386可寻址范围为4GB 即 00000000H~FFFFFFFFH，如果以这个地址（000FFFF0H）作为Bios地址的话，系统内存不连续，因此，386使用硬件置1的方式将A20～A31地址线置1，就变成FFFFFFF0H，并以此作为Bios地址。

这 个置1的结果是，隐藏的部分的一个字段Base＝FFFF0000H，这个操作不是由更改描述符表实现的，因为还没有进入保护模式，而且描述符表还没有 建立。这是硬件实现的，而且当进行一次段间跳转后，由于置1的结果就不能保存，因为硬件设计是从会将其置0，所以当执行完FFFFFFF0H处的指令 jmp ，Base＝00000000H，这时，Bios就使用1M以下内存。

关于BIOS影子内存：http://baike.baidu.com/view/568786.htm

关于入口地址的形成，有的文章上说是CS 0xFFFF和IP 0x0000的组成，有的文章上说是 CS 0xF000 和IP 0xFFF0的组合，我猜可能是不同硬件的初始化不同，只要最后形成的入口地址是 0xFFFF0就行了。如果不是我猜的那样，那回头在来补充吧。

首先，机器上电以后最先开始工作的芯片是南桥。因为电源上的DC ON信号是直接进南桥的，然后南桥收到DC ON信号，便一路发RESET信号给北桥到CPU,还有RESET信号到LPC、PCI总线，然后通过LPC总线或者是PCI总线去抓BIOS，地址总线的FFFFFFF0这个位置指向BIOS的第一条指令，同时CPU也RESET完毕，cs:eip硬件缺省复位成FFFFFFF0这个位置，那cpu获得的指令数据自然就会是bios中的第一条指令（jmp xxx）

而且，毫无疑问的，cpu上电第一次reset就是在实模式下面，然后通过执行bios中的代码将段基址0000的段限长改为4GB，这样cpu就进入了所谓的big real mode也就是flat mode，然后通过这个0000段才能访问4GB的内存空间。接着是对南桥的初始化，包括pcie控制器等，下面才是北桥，才是内存控制器。等内存扫完没有问题以后，就是把bios复制到内存中去，bios的代码才真正从内存中运行。在这之前为了加快速度还会有把cpu的catch设置成内存，把bios部分程序复制到catch中运行的。这里就各有各的做法了。

BIOS在POST时，最先检测的是CPU。

　　主板在接通电源后，系统首先由(Power On Self Test,上电自检)程序来对内部各个设备进行检查。在我们按下起动键（电源开关）时，系统的控制权就交由BIOS来完成，由于此时电压还不稳定，主板控制芯片组会向CPU发出并保持一个RESET(重置)信号，让CPU初始化，同时等待电源发出的POWER GOOD信号(电源准备好信号)。当电源开始稳定供电后(当然从不稳定到稳定的过程也只是短暂的瞬间)，芯片组便撤去RESET信号(如果是手动按下计算机面板上的Reset按钮来重启机器，那么松开该按钮时芯片组就会撤去RESET信号)，CPU马上就从地址FFFF0H处开始执行指令，这个地址在系统BIOS的地址范围内，无论是Award BIOS还是AMI BIOS，放在这里的只是一条跳转指令，跳到系统BIOS中真正的启动代码处。系统BIOS的启动代码首先要做的事情就是进行POST(Power On Self Test，加电自检)，由于电脑的硬件设备很多（包括存储器、中断、扩展卡），因此要检测这些设备的工作状态是否正常。

　　这一过程是逐一进行的，BIOS厂商对每一个设备都给出了一个检测代码（称为POST CODE即开机自我检测代码），在对某个设置进行检测时，首先将对应的POST CODE写入80H（地址）诊断端口，当该设备检测通过，则接着送另一个设置的POST CODE，对此设置进行测试。如果某个设备测试没有通过，则此POST CODE会在80H处保留下来，检测程序也会中止，并根据已定的报警声进行报警（BIOS厂商对报警声也分别作了定义，不同的设置出现故障，其报警声也是不同的，我们可以根据报警声的不同，分辨出故障所在。

　　POST自检测过程大致为：加电－CPU－ROM－BIOS－System Clock－DMA－64KB RAM－IRQ－显卡等。检测显卡以前的过程称过关键部件测试，如果关键部件有问题，计算机会处于挂起状态，习惯上称为核心故障。另一类故障称为非关键性故障，检测完显卡后，计算机将对64KB以上内存、I／O口、软硬盘驱动器、键盘、即插即用设备、CMOS设置等进行检测，并在屏幕上显示各种信息和出错报告。在正常情况下，POST过程进行得非常快，我们几乎无法感觉到这个过程。

在真正的操作系统引导过程中，由于BIOS的限制，只能从启动磁盘读取512byte的内容到内存中。

这就要求咱们不得不在这512byte中实现加载操作系统内核到内存的代码。

INT 13 是BIOS提供给咱们的读写磁盘的中断服务号。

首先看BIOS在内存中的位置排列：

最初的1M内存，是BIOS使用的。

0x00000 -- 0x003ff: 中断向量表

0x00400 -- 0x004ff: BIOS数据区

0x00500 -- 0x07bff: 自由内存区

0x07c00 -- 0x07dff: 引导程序加载区

0x07e00 -- 0x9ffff: 自由内存区

0xA0000 -- 0xBFFFF: 显示内存区

0xc0000 -- 0xfffff: BIOS中断处理代码

由此可见，有几个选择：在加载引导程序后，可通过引导程序将系统代码加载到：

0x00500 -- 0x07bff or

0x07e00 -- 0x9ffff。 后者的空间更大些。

下面来具体看一下BIOS的磁盘读写中断服务： INT 13，其参数寄存器内容为

AH: function ID, 2--READ

DL: driver ID

CH: 磁头号

CL: sector ID

AL: How many sector will be read

By the way, the aim position is ES:BX. So we need set the value before we start the interrupt.

第一步，CPU被复位后，首先被执行的代码的是BIOS（BIOS中最高地址的命令被最先执行，此命令直接指挥CPU跳转到Flash的相应区域去运行）。BIOS运行是以CPU的CACHE为内存的，一些BIOS代码拷贝到Cache中，因为此时内存并没有被初始化。BIOS的主要功能是作一些硬件初始化动作，读写一些硬件的寄存器：比如8259的寄存器等，让硬件处于“准备好”状态。最后BIOS去查看硬盘的第一个扇区，根据此扇区的某几个特殊标志位的值判断其是否为引导扇区。如果是则将此引导扇区（可能是512字节）拷贝到内存。拷贝到内存什么地方呢？一般由引导扇区代码中的第一条语句指定：org07c00h，这个地址是内存中留给boot sector的地址

第二步，此引导扇区由于空间很小，所以只能做一个简单的动作，找到Loader，然后将Loader拷贝到内存，最后将控制权交给Loader（也是一个跳转指令）。其找Loader的时候，需要根据文件系统特定的格式来查找，如下图所示：

具体过程大概是：在根目录区中根据Loader的文件名字找到文件对应的FAT的位置，然后根据FAT数据结构中的连锁关系找到此文件在数据区中所有的簇号。

第三步：Loader进入内存开始运行，它的两个主要工作任务是：1）加载内核入内存2）跳入保护模式。可能会做一些GDT,LDT,IDT以及内存的分页等等。

第四步：真正的系统内核开始运行。

# 11、网络文件共享和传输那点事

说到网络文件共享和传输，很多人会想到ftp和ssh，不过很多人分不清这些协议的原理，甚至混淆这些协议，下面就详细说说这些协议，以及目前我们如何做到不同文件系统之间的共享。

文件共享一直是不同操作系统间的一个大问题，为啥呢，这都是闭源系统惹的祸，学过操作系统的人都知道，文件系统属于操作系统内核的一部分，而且是和用户关系比较直接的一部分。

$ sshfs -o defer\_permissions root@yourdomain.com:/root local\_mount\_dir/

将服务器目录yourdomain.com:/root挂载到本地目录local\_mount\_dir/。 为了免输密码，建议提前将本地机器和服务器之间ssh打通信任关系。

$ umount local\_mount\_dir/

不想挂了，可别rm local\_mout\_dir哦，调用umount来取消挂载

哎，终于可以不天天登陆服务器了。用用Eclipse啥的高级工具搞搞开发去~

Mac上支持sshfs的话，请现在这里下载OSXFuse： <http://osxfuse.github.io/>

# 12、javascript

JavaScript 是世界上最流行的编程语言。js严格区分大小写

这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。

JavaScript 是脚本语言

JavaScript 是一种轻量级的编程语言。

JavaScript 是可插入 HTML 页面的编程代码。

JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

JavaScript 很容易学习。

JavaScript 与 Java 是两种完全不同的语言，无论在概念还是设计上。

Java（由 Sun 发明）是更复杂的编程语言。

ECMA-262 是 JavaScript 标准的官方名称。

JavaScript 由 Brendan Eich 发明。它于 1995 年出现在 Netscape 中（该浏览器已停止更新），并于 1997 年被 ECMA（一个标准协会）采纳。

## 12.1、javascript简介

HTML 中的脚本必须位于 <script> 与 </script> 标签之间。

脚本可被放置在 HTML 页面的 <body> 和 <head> 部分中。

<script> 标签

如需在 HTML 页面中插入 JavaScript，请使用 <script> 标签。

<script> 和 </script> 会告诉 JavaScript 在何处开始和结束。

<script> 和 </script> 之间的代码行包含了 JavaScript：

那些老旧的实例可能会在 <script> 标签中使用 type="text/javascript"。现在已经不必这样做了。JavaScript 是所有现代浏览器以及 HTML5 中的默认脚本语言。

<head> 或 <body> 中的 JavaScript

您可以在 HTML 文档中放入不限数量的脚本。

脚本可位于 HTML 的 <body> 或 <head> 部分中，或者同时存在于两个部分中。

通常的做法是把函数放入 <head> 部分中，或者放在页面底部。这样就可以把它们安置到同一处位置，不会干扰页面的内容。

<!DOCTYPE html>

<html>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>

function myFunction()

{

document.getElementById("demo").innerHTML="My First JavaScript Function";

}

</script>

</body>

</html>

我们把 JavaScript 放到了页面代码的底部，这样就可以确保在 <p> 元素创建之后再执行脚本。

外部的 JavaScript

也可以把脚本保存到外部文件中。外部文件通常包含被多个网页使用的代码。外部 JavaScript 文件的文件扩展名是 .js。如需使用外部文件，请在 <script> 标签的 "src" 属性中设置该 .js 文件：

<!DOCTYPE html>

<html>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">点击这里</button>

<p><b>注释：</b>myFunction 保存在名为 "myScript.js" 的外部文件中。</p>

<script type="text/javascript" src="/js/myScript.js"></script>

</body>

</html>

而js文件myScript.js内容如下：

function myFunction(){

document.getElementById("demo").innerHTML="My First JavaScript Function";

}

在 <head> 或 <body> 中引用脚本文件都是可以的。实际运行效果与您在 <script> 标签中编写脚本完全一致。

提示：外部脚本不能包含 <script> 标签。

## 12.2、JavaScript 通常用于操作 HTML 元素。

操作 HTML元素，如需从 JavaScript 访问某个 HTML 元素，您可以使用 document.getElementById(id) 方法。

请使用 "id" 属性来标识 HTML 元素：

例子

通过指定的 id 来访问 HTML 元素，并改变其内容：

<!DOCTYPE html>

<html>

<body>

<h1>My First Web Page</h1>

<p id="demo">My First Paragraph</p>

<script>

document.getElementById("demo").innerHTML="My First JavaScript";

</script>

</body>

</html>

JavaScript 由 web 浏览器来执行。在这种情况下，浏览器将访问 id="demo" 的 HTML 元素，并把它的内容（innerHTML）替换为 "My First JavaScript"。

## 12.3、javascript语法精华

## ①条件语句

通常在写代码时，您总是需要为不同的决定来执行不同的动作。您可以在代码中使用条件语句来完成该任务。

在 JavaScript 中，我们可使用以下条件语句：

if 语句 - 只有当指定条件为 true 时，使用该语句来执行代码

if...else 语句 - 当条件为 true 时执行代码，当条件为 false 时执行其他代码

if...else if....else 语句 - 使用该语句来选择多个代码块之一来执行

switch 语句 - 使用该语句来选择多个代码块之一来执行

### ⑴switch

工作原理：首先设置表达式 n（通常是一个变量）。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 关联的代码块会被执行。请使用 break 来阻止代码自动地向下一个 case 运行。使用 default 关键词来规定匹配不存在时做的事情。

switch(n)

{

case 1:

执行代码块 1

break;

case 2:

执行代码块 2

break;

default:

n 与 case 1 和 case 2 不同时执行的代码

}

### ⑵If

注意：请使用小写的 if。使用大写字母（IF）会生成 JavaScript 错误！

㈠

if (条件)

{

只有当条件为 true 时执行的代码

}

㈡

if (条件)

{

当条件为 true 时执行的代码

}

else

{

当条件不为 true 时执行的代码

}

㈢

if (条件 1)

{

当条件 1 为 true 时执行的代码

}

else if (条件 2)

{

当条件 2 为 true 时执行的代码

}

else

{

当条件 1 和 条件 2 都不为 true 时执行的代码

}

## ②不同类型的循环

JavaScript 支持不同类型的循环：

for - 循环代码块一定的次数

for/in - 循环遍历对象的属性

while - 当指定的条件为 true 时循环指定的代码块

do/while - 同样当指定的条件为 true 时循环指定的代码块

### ⑴for循环

for循环在语法上比较灵活，而且它和while循环比较类似

具体语法结构如下：

for (语句 1; 语句 2; 语句 3)

{

被执行的代码块

}

语句 1 在循环（代码块）开始前执行，通常我们会使用语句 1 初始化循环中所用的变量 (var i=0)，而且你可以初始化多个变量，中间用逗号隔开。而且语句 1 是可选的，也就是说不使用语句 1 也可以。

语句 2 定义运行循环（代码块）的条件。通常语句 2 用于评估初始变量的条件。语句 2 同样是可选的。如果语句 2 返回 true，则循环再次开始，如果返回 false，则循环将结束。提示：如果您省略了语句 2，那么必须在循环内提供 break。否则循环就无法停下来。这样有可能令浏览器崩溃。

语句 3 在循环（代码块）已被执行之后执行。通常语句 3 会增加初始变量的值。语句 3 也是可选的。语句 3 有多种用法。增量可以是负数 (i--)，或者更大 (i=i+15)。语句 3 也可以省略（比如当循环内部有相应的代码时）：

从上面的介绍中你会发现，for循环的三个语句可有可无，你就是啥也没有，像这样：

for( ; ; ){

代码块

}

也是完全没有问题的。下面看一段实例：

<!DOCTYPE html>

<html>

<body>

<p>for循环测试 。</p>

<button onclick="myFunction()">点击这里</button>

<p id="demo">xxxxxxxxxxx</p>

<script>

function myFunction()

{

var cars=["BMW","Volvo","Saab","Ford"];

var x="",i=j=0;

//for(var i=0,j=i;cars[i]; ){

for(i in cars){

j++;

x=x+"这是第"+j+"台车："+cars[i]+"<br>";

//i++;

}

document.getElementById("demo").innerHTML=x;

}

</script>

</body>

</html>

这段程序中实际上包含了两种循环，这里也顺带解释一下for in循环：

循环遍历对象的属性，实际上javascript的对象概念和通常oop语言的对象不太一样，因为它没有类的概念，而且所有数据都被视为对象，无论是数组，还是对象，甚至字符串都被看做对象，那么for in循环就是遍历该对象所有元素。

for(x in 对象){

}

上面我们看了一个数组的实例，下面看一个对象的实例：

<script>

function myFunction()

{

var x;

var txt="";

var person={fname:"Bill",lname:"Gates",age:56};

for (x in person)

{

txt=txt + person[x];

}

document.getElementById("demo").innerHTML=txt;

}

</script>

其中x代表对象元素个数，它会自动增加。所以循环体内没有变量增加的语句。

⑵ while do/while 循环

while (条件)

{

需要执行的代码

}

do

{

需要执行的代码

}

while (条件);

比较 for 和 while

cars=["BMW","Volvo","Saab","Ford"];

var i=0;

for (;cars[i];)

{

document.write(cars[i] + "<br>");

i++;

}

cars=["BMW","Volvo","Saab","Ford"];

var i=0;

while (cars[i])

{

document.write(cars[i] + "<br>");

i++;

}

## 12.4、Javascript 创建对象方法的总结

Javascript是一种基于对象（object-based）的语言，你遇到的所有东西几乎都是对象。但是，它又不是一种真正的面向对象编程（OOP）语言，因为它的语法中没有class（类）。使用Javascript创建对象的方法有很多，现在就来列举一下：

1. 使用Object构造函数来创建一个对象，下面代码创建了一个person对象，并用两种方式打印出了Name的属性值。

var person = new Object();

person.name="kevin";

person.age=31;

alert(person.name);

alert(person["name"]);

2. 使用对象字面量创建一个对象；不要奇怪person["5"],这里是合法的；另外使用这种加括号的方式字段之间是可以有空格的如person["my age"]。空格和折行无关紧要，声明可横跨多行：

var person =

{

name:"Kevin",

age:31,

5:"Test"

};

alert(person.name);

alert(person["5"]);

这里的空格和换行都是被忽略的，就是说你也可以这样写

var person={name:”kevin”,age:31,5:”Test”};

3. 使用工厂模式创建对象,返回带有属性和方法的person对象。

# 13、脚本语言和编程语言的区别

计算机语言的种类非常的多，总的来说可以分成机器语言，汇编语言，高级语言三大类。

计算机所能识别的语言只有机器语言，即由0和1构成的代码。但通常人们编程时，不采用机器语言，因为它非常难于记忆和识别。

汇编语言的实质和机器语言是相同的，都是直接对硬件操作，只不过指令采用了英文缩写的标识符，更容易识别和记忆。它同样需要编程者将每一步具体的操作用命令的形式写出来。汇编程序通常由三部分组成：指令、伪指令和宏指令。汇编程序的每一句指令只能对应实际操作过程中的一个很细微的动作，例如移动、自增，因此汇编源程序一般比较冗长、复杂、容易出错，而且使用汇编语言编程需要有更多的计算机专业知识，但汇编语言的优点也是显而易见的，用汇编语言所能完成的操作不是一般高级语言所能实现的，而且源程序经汇编生成的可执行文件不仅比较小，而且执行速度很快。

高级语言是目前绝大多数编程者的选择。和汇编语言相比，它不但将许多相关的机器指令合成为单条指令，并且去掉了与具体操作有关但与完成工作无关的细节，例如使用堆栈、寄存器等，这样就大大简化了程序中的指令。同时，由于省略了很多细节，编程者也就不需要有太多的专业知识。

高级语言主要是相对于汇编语言而言，它并不是特指某一种具体的语言，而是包括了很多编程语言，如目前流行的C、C++、Object C、PASCAL、BASIC等，这些语言的语法、命令格式都各不相同。 而我们通常所说的VB，VC，Delphi，Xcode等并不是编程语言，而是编程工具，这些不同公司针对某种语言而开发的集成开发的编译工具，当然这些工具的好处就是能够帮助程序员快速完成项目开发，尽管它们有些地方，并不完全遵守某种语言的标准。

高级语言所编制的程序不能直接被计算机识别，必须经过转换才能被执行，按转换方式可将它们分为两类:

解释类：执行方式类似于我们日常生活中的“同声翻译”，应用程序源代码一边由相应语言的解释器“翻译”成目标代码(机器语言)，一边执行，因此效率比较低，而且不能生成可独立执行的可执行文件，应用程序不能脱离其解释器，但这种方式比较灵活，可以动态地调整、修改应用程序。

编译类：编译是指在应用源程序执行之前，就将程序源代码“翻译”成目标代码(机器语言)，因此其目标程序可以脱离其语言环境独立执行，使用比较方便、效率较高。但应用程序一旦需要修改，必须先修改源代码，再重新编译生成新的目标文件(＊ .OBJ)才能执行，只有目标文件而没有源代码，修改很不方便。现在大多数的编程语言都是编译型的，例如Visual C＋＋、Visual Foxpro、Delphi等。

## Java 是编译型语言还是解释型语言？

有些答案对JAVA的理解还停留在上古时代或者教科书里。

其实，现在用编译型、解释型来分类编程语言已经有点力不从心了。

JAVA的第一道工序是javac编译，当然目标文件是BYTECODE。后续可能有三种处理方式：

1. 运行时，BYTECODE由JVM逐条解释执行，

2. 运行时，部分代码可能由JIT翻译为目标机器指令（以method为翻译单位，还会保存起来，第二次执行就不用翻译了）直接执行；

3. RTSJ。继JAVAC之后执行AOT二次编译，生成静态的目标平台代码（典型的就是IBM WEBSHPERE REAL TIME）。

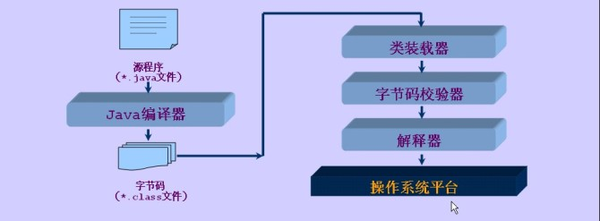
有的时候，可能是以上三种方式同时在使用。至少，1和2是同时使用的，3需要程序员手工指定。

所以很多地方都说“java是一种半编译、半解释执行”的语言；

近来(其实也不是很"近")Oracle的(以前是Sun的)HotSpot VM采用了jit compile(just in time compilation)技术，将运行频率很高的字节码直接编译为机器指令执行以提高性能, 所以当字节码被jit编译为机器码的时候，要说它是编译执行的也可以...

不过总体来讲，java的编译结果是被jvm“解释执行”的，所以这么说也能说通，而其实这个“是编译还是解释”这个概念在这里已经有点模糊了，理解它的过程就行了，不必下一个“精确”的定义;

而我自己仍然赞成“java是编译型语言”的说法，因为“编译”其本质就是“把一个相对高级的语言转换为另一个相对低级的语言”，而由java -> class文件的编译已经满足了这个特征; 而后面你要说jvm是“解释执行”的，那其实硬件对于机器码又何尝不是“解释执行”呢？



标记语言

标记语言，是一种将文本（Text）以及文本相关的其他信息结合起来，展现出关于文档结构和数据处理细节的电脑文字编码。与文本相关的其他信息（包括例如文本的结构和表示信息等）与原来的文本结合在一起，但是使用标记（markup）进行标识。如：HTML、XML

脚本语言

脚本语言是为了缩短传统的编写-编译-链接-运行（edit-compile-link-run）过程而创建的计算机编程语言。它的命名起源于一个脚本“screenplay”，每次运行都会使对话框逐字重复。早期的脚本语言经常被称为批量处理语言或工作控制语言。

一个脚本通常是解释运行而非编译。脚本语言通常都有简单、易学、易用的特性，目的就是希望能让程序员快速完成程序的编写工作。

如：JavaScript、VBScript、PHP

编译型语言

编译型语言：程序在执行之前需要一个专门的编译过程，把程序编译成 为机器语言的文件，运行时不需要重新翻译，直接使用编译的结果就行了。程序执行效率高，依赖编译器，跨平台性差些。如 C、C++

二、区别

1、标记语言不用于向计算机发出指令，常用于格式化和链接。

2、脚本语言介于标记语言和编程语言之间，脚本语言脚本语言不需要编译，可以直接用，由解释器来负责解释。

3、编译型语言写的程序执行之前，需要一个专门的编译过程，把程序编译成为机器语言的文件，比如exe文件，以后要运行的话就不用重新翻译了，直接使用编译的结果就行了（exe文件），因为翻译只做了一次，运行时不需要翻译，所以编译型语言的程序执行效率高。

我刚收到的时候很兴奋，就开始写回复。写啊写发觉已经比我平时发的帖还要长了，想着干脆把回复直接发出来好了。于是下面就是回复：

你好 ^ ^ 很抱歉拖了这么久才回复。码字和画图太耗时间了。

别说冒昧了，我只是个普通的刚毕业的学生而已，担当不起啊 =\_=||||

而且我也不敢说“很”了解，只是有所接触而已。很高兴有人来一起讨论JavaScript引擎的设计与实现，总觉得身边对这个有兴趣的人不多，或者是很少冒出来讨论。如果你发个帖或者blog来讨论这方面的内容我也会很感兴趣的～

想拿出几点来讨论一下。上面提出的问题我希望能够一一给予回答，不过首先得做些铺垫。

另外先提一点：JavaScriptCore从SquirrelFish版开始是“基于寄存器”的，V8则不适合用“基于栈”或者“基于寄存器”的说法来描述。

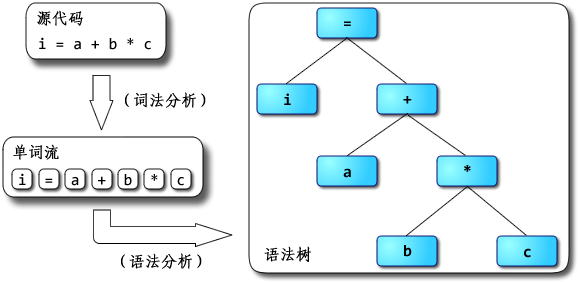
1、解析器与解释器

解析器是parser，而解释器是interpreter。两者不是同一样东西，不应该混用。

前者是编译器/解释器的重要组成部分，也可以用在IDE之类的地方；其主要作用是进行语法分析，提取出句子的结构。广义来说输入一般是程序的源码，输出一般是语法树（syntax tree，也叫parse tree等）或抽象语法树（abstract syntax tree，AST）。进一步剥开来，广义的解析器里一般会有扫描器（scanner，也叫tokenizer或者lexical analyzer，词法分析器），以及狭义的解析器（parser，也叫syntax analyzer，语法分析器）。扫描器的输入一般是文本，经过词法分析，输出是将文本切割为单词的流。狭义的解析器输入是单词的流，经过语法分析，输出是语法树或者精简过的AST。

（在一些编译器/解释器中，解析也可能与后续的语义分析、代码生成或解释执行等步骤融合在一起，不一定真的会构造出完整的语法树。但概念上说解析器就是用来抽取句子结构用的，而语法树就是表示句子结构的方式。关于边解析边解释执行的例子，可以看看这帖的计算器。）

举例：将i = a + b \* c作为源代码输入到解析器里，则广义上的解析器的工作流程如下图：



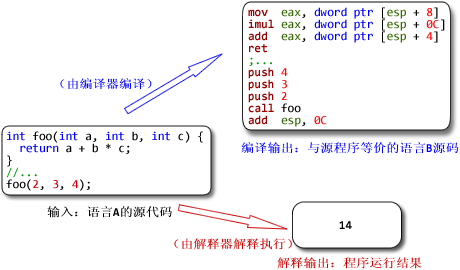
其中词法分析由扫描器完成，语法分析由狭义的解析器完成。

（嗯，说来其实“解析器”这词还是按狭义用法比较准确。把扫描器和解析器合起来叫解析器总觉得怪怪的，但不少人这么用，这里就将就下吧 =\_=

不过近来“scannerless parsing”也挺流行的：不区分词法分析与语法分析，没有单独的扫描器，直接用解析器从源码生成语法树。这倒整个就是解析器了，没狭不狭义的问题）

后者则是实现程序执行的一种实现方式，与编译器相对。它直接实现程序源码的语义，输入是程序源码，输出则是执行源码得到的计算结果；编译器的输入与解释器相同，而输出是用别的语言实现了输入源码的语义的程序。通常编译器的输入语言比输出语言高级，但不一定；也有输入输出是同种语言的情况，此时编译器很可能主要用于优化代码。

举例：把同样的源码分别输入到编译器与解释器中，得到的输出不同：



值得留意的是，编译器生成出来的代码执行后的结果应该跟解释器输出的结果一样——它们都应该实现源码所指定的语义。

在很多地方都看到解析器与解释器两个不同的东西被混为一谈，感到十分无奈。

最近某本引起很多关注的书便在开篇给读者们当头一棒，介绍了“JavaScript解析机制”。“编译”和“预处理”也顺带混为一谈了，还有“预编译” 0\_0

我一直以为“预编译”应该是ahead-of-time compilation的翻译，是与“即时编译”（just-in-time compilation，JIT）相对的概念。另外就是PCH（precompile header）这种用法，把以前的编译结果缓存下来称为“预编译”。把AOT、PCH跟“预处理”（preprocess）混为一谈真是诡异。算了，我还是不要淌这浑水的好……打住。

2、“解释器”到底是什么？“解释型语言”呢？

很多资料会说，Python、Ruby、JavaScript都是“解释型语言”，是通过解释器来实现的。这么说其实很容易引起误解：语言一般只会定义其抽象语义，而不会强制性要求采用某种实现方式。

例如说C一般被认为是“编译型语言”，但C的解释器也是存在的，例如Ch。同样，C++也有解释器版本的实现，例如Cint。

一般被称为“解释型语言”的是主流实现为解释器的语言，但并不是说它就无法编译。例如说经常被认为是“解释型语言”的Scheme就有好几种编译器实现，其中率先支持R6RS规范的大部分内容的是Ikarus，支持在x86上编译Scheme；它最终不是生成某种虚拟机的字节码，而是直接生成x86机器码。

解释器就是个黑箱，输入是源码，输出就是输入程序的执行结果，对用户来说中间没有独立的“编译”步骤。这非常抽象，内部是怎么实现的都没关系，只要能实现语义就行。你可以写一个C语言的解释器，里面只是先用普通的C编译器把源码编译为in-memory image，然后直接调用那个image去得到运行结果；用户拿过去，发现直接输入源码可以得到源程序对应的运行结果就满足需求了，无需在意解释器这个“黑箱子”里到底是什么。

实际上很多解释器内部是以“编译器+虚拟机”的方式来实现的，先通过编译器将源码转换为AST或者字节码，然后由虚拟机去完成实际的执行。所谓“解释型语言”并不是不用编译，而只是不需要用户显式去使用编译器得到可执行代码而已。

那么虚拟机（virtual machine，VM）又是什么？在许多不同的场合，VM有着不同的意义。如果上下文是Java、Python这类语言，那么一般指的是高级语言虚拟机（high-level language virtual machine，HLL VM），其意义是实现高级语言的语义。VM既然被称为“机器”，一般认为输入是满足某种指令集架构（instruction set architecture，ISA）的指令序列，中间转换为目标ISA的指令序列并加以执行，输出为程序的执行结果的，就是VM。源与目标ISA可以是同一种，这是所谓same-ISA VM。

前面提到解释器中的编译器的输出可能是AST，也可能是字节码之类的指令序列；一般会把执行后者的程序称为VM，而执行前者的还是笼统称为解释器或者树遍历式解释器（tree-walking interpreter）。这只是种习惯而已，并没有多少确凿的依据。只不过线性（相对于树形）的指令序列看起来更像一般真正机器会执行的指令序列而已。

其实我觉得把执行AST的也叫VM也没啥大问题。如果认同这个观点，那么把DLR看作一种VM也就可以接受了——它的“指令集”就是树形的Expression Tree。

VM并不是神奇的就能执行代码了，它也得采用某种方式去实现输入程序的语义，并且同样有几种选择：“编译”，例如微软的.NET中的CLR；“解释”，例如CPython、CRuby 1.9，许多老的JavaScript引擎等；也有介于两者之间的混合式，例如Sun的JVM，HotSpot。如果采用编译方式，VM会把输入的指令先转换为某种能被底下的系统直接执行的形式（一般就是native code），然后再执行之；如果采用解释方式，则VM会把输入的指令逐条直接执行。

换个角度说，我觉得采用编译和解释方式实现虚拟机最大的区别就在于是否存下目标代码：编译的话会把输入的源程序以某种单位（例如基本块/函数/方法/trace等）翻译生成为目标代码，并存下来（无论是存在内存中还是磁盘上，无所谓），后续执行可以复用之；解释的话则是把源程序中的指令逐条解释，不生成也不存下目标代码，后续执行没有多少可复用的信息。有些稍微先进一点的解释器可能会优化输入的源程序，把满足某些模式的指令序列合并为“超级指令”；这么做就是朝着编译的方向推进。后面讲到解释器的演化时再讨论超级指令吧。

如果一种语言的主流实现是解释器，其内部是编译器+虚拟机，而虚拟机又是采用解释方式实现的，或者内部实现是编译器+树遍历解释器，那它就是名副其实的“解释型语言”。如果内部用的虚拟机是用编译方式实现的，其实跟普遍印象中的“解释器”还是挺不同的……

可以举这样一个例子：ActionScript 3，一般都被认为是“解释型语言”对吧？但这种观点到底是把FlashPlayer整体看成一个解释器，因而AS3是“解释型语言”呢？还是认为FlashPlayer中的虚拟机采用解释执行方案，因而AS3是“解释型语言”呢？

其实Flash或Flex等从AS3生成出来的SWF文件里就包含有AS字节码（ActionScript Byte Code，ABC）。等到FlashPlayer去执行SWF文件，或者说等到AVM2（ActionScript Virtual Machine 2）去执行ABC时，又有解释器和JIT编译器两种实现。这种需要让用户显式进行编译步骤的语言，到底是不是“解释型语言”呢？呵呵。所以我一直觉得“编译型语言”跟“解释型语言”的说法太模糊，不太好。

有兴趣想体验一下从命令行编译“裸”的AS3文件得到ABC文件，再从命令行调用AVM2去执行ABC文件的同学，可以从这帖下载我之前从源码编译出来的AVM2，自己玩玩看。例如说要编译一个名为test.as的文件，用下列命令：

java -jar asc.jar -import builtin.abc -import toplevel.abc test.as

就是用ASC将test.as编译，得到test.abc。接着用：

avmplus test.abc

就是用AVM2去执行程序了。很生动的体现出“编译器+虚拟机”的实现方式。

这个“裸”的AVM2没有带Flash或Flex的类库，能用的函数和类都有限。不过AS3语言实现是完整的。可以用print()函数来向标准输出流写东西。

Well……其实写Java程序不也是这样么？现在也确实还有很多人把Java称为“解释型语言”，完全无视Java代码通常是经过显式编译步骤才得到.class文件，而有些JVM是采用纯JIT编译方式实现的，内部没解释器，例如JRockit、Maxine VM和Jikes RVM。我愈发感到“解释型语言”是个应该避开的用语 =\_=

关于虚拟机，有本很好的书绝对值得一读，《虚拟机——系统与进程的通用平台》（Virtual Machines: Versatile Platforms for Systems and Processes）。国内有影印版也有中文版，我是读了影印版，不太清楚中文版的翻译质量如何。据说翻译得还行，我无法印证。

3、基于栈与基于寄存器的指令集架构

用C的语法来写这么一个语句：

a = b + c;

如果把它变成这种形式：

add a, b, c

那看起来就更像机器指令了，对吧？这种就是所谓“三地址指令”（3-address instruction），一般形式为：

op dest, src1, src2

许多操作都是二元运算+赋值。三地址指令正好可以指定两个源和一个目标，能非常灵活的支持二元操作与赋值的组合。ARM处理器的主要指令集就是三地址形式的。

C里要是这样写的话：

a += b;

变成:

add a, b

这就是所谓“二地址指令”，一般形式为：

op dest, src

它要支持二元操作，就只能把其中一个源同时也作为目标。上面的add a, b在执行过后，就会破坏a原有的值，而b的值保持不变。x86系列的处理器就是二地址形式的。

上面提到的三地址与二地址形式的指令集，一般就是通过“基于寄存器的架构”来实现的。例如典型的RISC架构会要求除load和store以外，其它用于运算的指令的源与目标都要是寄存器。

显然，指令集可以是任意“n地址”的，n属于自然数。那么一地址形式的指令集是怎样的呢？

想像一下这样一组指令序列：

add 5

sub 3

这只指定了操作的源，那目标是什么？一般来说，这种运算的目标是被称为“累加器”（accumulator）的专用寄存器，所有运算都靠更新累加器的状态来完成。那么上面两条指令用C来写就类似：

acc += 5;

acc -= 3;

只不过acc是“隐藏”的目标。基于累加器的架构近来比较少见了，在很老的机器上繁荣过一段时间。

那“n地址”的n如果是0的话呢？

看这样一段Java字节码：

iconst\_1

iconst\_2

iadd

istore\_0

注意那个iadd（表示整型加法）指令并没有任何参数。连源都无法指定了，零地址指令有什么用？？

零地址意味着源与目标都是隐含参数，其实现依赖于一种常见的数据结构——没错，就是栈。上面的iconst\_1、iconst\_2两条指令，分别向一个叫做“求值栈”（evaluation stack，也叫做operand stack“操作数栈”或者expression stack“表达式栈”）的地方压入整型常量1、2。iadd指令则从求值栈顶弹出2个值，将值相加，然后把结果压回到栈顶。istore\_0指令从求值栈顶弹出一个值，并将值保存到局部变量区的第一个位置（slot 0）。

零地址形式的指令集一般就是通过“基于栈的架构”来实现的。请一定要注意，这个栈是指“求值栈”，而不是与系统调用栈（system call stack，或者就叫system stack）。千万别弄混了。有些虚拟机把求值栈实现在系统调用栈上，但两者概念上不是一个东西。

由于指令的源与目标都是隐含的，零地址指令的“密度”可以非常高——可以用更少空间放下更多条指令。因此在空间紧缺的环境中，零地址指令是种可取的设计。但零地址指令要完成一件事情，一般会比二地址或者三地址指令许多更多条指令。上面Java字节码做的加法，如果用x86指令两条就能完成了：

X86 asm代码

mov eax, 1

add eax, 2

（好吧我犯规了，istore\_0对应的保存我没写。但假如局部变量比较少的话也不必把EAX的值保存（“溢出”，register spilling）到调用栈上，就这样吧 =\_=

其实就算把结果保存到栈上也就是多一条指令而已……）

一些比较老的解释器，例如CRuby在1.9引入YARV作为新的VM之前的解释器，还有SquirrleFish之前的老JavaScriptCore以及它的前身KJS，它们内部是树遍历式解释器；解释器递归遍历树，树的每个节点的操作依赖于解释其各个子节点返回的值。这种解释器里没有所谓的求值栈，也没有所谓的虚拟寄存器，所以不适合以“基于栈”或“基于寄存器”去描述。

而像V8那样直接编译JavaScript生成机器码，而不通过中间的字节码的中间表示的JavaScript引擎，它内部有虚拟寄存器的概念，但那只是普通native编译器的正常组成部分。我觉得也不应该用“基于栈”或“基于寄存器”去描述它。

V8在内部也用了“求值栈”（在V8里具体叫“表达式栈”）的概念来简化生成代码的过程，在编译过程中进行“抽象解释”，使用所谓“虚拟栈帧”来记录局部变量与求值栈的状态；但在真正生成代码的时候会做窥孔优化，消除冗余的push/pop，将许多对求值栈的操作转变为对寄存器的操作，以此提高代码质量。于是最终生成出来的代码看起来就不像是基于栈的代码了。

关于JavaScript引擎的实现方式，下文会再提到。

4、基于栈与基于寄存器架构的VM，用哪个好？

如果是要模拟现有的处理器，那没什么可选的，原本处理器采用了什么架构就只能以它为源。但HLL VM的架构通常可以自由构造，有很大的选择余地。为什么许多主流HLL VM，诸如JVM、CLI、CPython、CRuby 1.9等，都采用了基于栈的架构呢？我觉得这有三个主要原因：

·实现简单

由于指令中不必显式指定源与目标，VM可以设计得很简单，不必考虑为临时变量分配空间的问题，求值过程中的临时数据存储都让求值栈包办就行。

更新：回帖中cscript指出了这句不太准确，应该是针对基于栈架构的指令集生成代码的编译器更容易实现，而不是VM更容易实现。

·该VM是为某类资源非常匮乏的硬件而设计的

这类硬件的存储器可能很小，每一字节的资源都要节省。零地址指令比其它形式的指令更紧凑，所以是个自然的选择。

·考虑到可移植性

处理器的特性各个不同：典型的CISC处理器的通用寄存器数量很少，例如32位的x86就只有8个32位通用寄存器（如果不算EBP和ESP那就是6个，现在一般都算上）；典型的RISC处理器的各种寄存器数量多一些，例如ARM有16个32位通用寄存器，Sun的SPARC在一个寄存器窗口里则有24个通用寄存器（8 in，8 local，8 out）。

假如一个VM采用基于寄存器的架构（它接受的指令集大概就是二地址或者三地址形式的），为了高效执行，一般会希望能把源架构中的寄存器映射到实际机器上寄存器上。但是VM里有些很重要的辅助数据会经常被访问，例如一些VM会保存源指令序列的程序计数器（program counter，PC），为了效率，这些数据也得放在实际机器的寄存器里。如果源架构中寄存器的数量跟实际机器的一样，或者前者比后者更多，那源架构的寄存器就没办法都映射到实际机器的寄存器上；这样VM实现起来比较麻烦，与能够全部映射相比效率也会大打折扣。像Dalvik VM的解释器实现，就是把虚拟寄存器全部映射到栈帧（内存）里的，这跟把局部变量区与操作数栈都映射到内存里的JVM解释器实现相比实际区别不太大。

如果一个VM采用基于栈的架构，则无论在怎样的实际机器上，都很好实现——它的源架构里没有任何通用寄存器，所以实现VM时可以比较自由的分配实际机器的寄存器。于是这样的VM可移植性就比较高。作为优化，基于栈的VM可以用编译方式实现，“求值栈”实际上也可以由编译器映射到寄存器上，减轻数据移动的开销。

回到主题，基于栈与基于寄存器的架构，谁更快？看看现在的实际处理器，大多都是基于寄存器的架构，从侧面反映出它比基于栈的架构更优秀。

而对于VM来说，源架构的求值栈或者寄存器都可能是用实际机器的内存来模拟的，所以性能特性与实际硬件又有点不同。一般认为基于寄存器的架构对VM来说也是更快的，原因是：虽然零地址指令更紧凑，但完成操作需要更多的load/store指令，也意味着更多的指令分派（instruction dispatch）次数与内存访问次数；访问内存是执行速度的一个重要瓶颈，二地址或三地址指令虽然每条指令占的空间较多，但总体来说可以用更少的指令完成操作，指令分派与内存访问次数都较少。

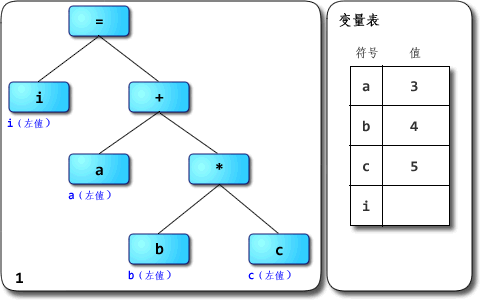
这方面有篇被引用得很多的论文讲得比较清楚，Virtual Machine Showdown: Stack Versus Registers，是在VEE 2005发表的。VEE是Virtual Execution Environment的缩写，是ACM下SIGPLAN组织的一个会议，专门研讨虚拟机的设计与实现的。可以去找找这个会议往年的论文，很多都值得读。

5、树遍历解释器图解

在演示基于栈与基于寄存器的VM的例子前，先回头看看更原始的解释器形式。

前面提到解析器的时候用了i = a + b \* c的例子，现在让我们来看看由解析器生成的AST要是交给一个树遍历解释器，会如何被解释执行呢？

用文字说不够形象，还是看图吧：



这是对AST的后序遍历：假设有一个eval(Node n)函数，用于解释AST上的每个节点；在解释一个节点时如果依赖于子树的操作，则对子节点递归调用eval(Node n)，从这些递归调用的返回值获取需要的值（或副作用）——也就是说子节点都eval好了之后，父节点才能进行自己的eval——典型的后序遍历。

（话说，上图中节点左下角有蓝色标记的说明那是节点的“内在属性”。从属性语法的角度看，如果一个节点的某个属性的值只依赖于自身或子节点，则该属性被称为“综合属性”（synthesized attribute）；如果一个节点的某个属性只依赖于自身、父节点和兄弟节点，则该属性被称为“继承属性”（inherited attribute）。上图中节点右下角的红色标记都只依赖子节点来计算，显然是综合属性。）

SquirrelFish之前的JavaScriptCore、CRuby 1.9之前的CRuby就都是采用这种方式来解释执行的。

可能需要说明的：

·左值与右值

在源代码i = a + b \* c中，赋值符号左侧的i是一个标识符，表示一个变量，取的是变量的“左值”（也就是与变量i绑定的存储单元）；右侧的a、b、c虽然也是变量，但取的是它们的右值（也就是与变量绑定的存储单元内的值）。在许多编程语言中，左值与右值在语法上没有区别，它们实质的差异容易被忽视。一般来说左值可以作为右值使用，反之则不一定。例如数字1，它自身有值就是1，可以作为右值使用；但它没有与可赋值的存储单元相绑定，所以无法作为左值使用。

左值不一定只是简单的变量，还可以是数组元素或者结构体的域之类，可能由复杂的表达式所描述。因此左值也是需要计算的。

·优先级、结合性与求值顺序

这三个是不同的概念，却经常被混淆。通过AST来看就很容易理解：（假设源码是从左到右输入的）

所谓优先级，就是不同操作相邻出现时，AST节点与根的距离的关系。优先级高的操作会更远离根，优先级低的操作会更接近根。为什么？因为整棵AST是以后序遍历求值的，显然节点离根越远就越早被求值。

所谓结合性，就是当同类操作相邻出现时，操作的先后顺序同AST节点与根的距离的关系。如果是左结合，则先出现的操作对应的AST节点比后出现的操作的节点离根更远；换句话说，先出现的节点会是后出现节点的子节点。

所谓求值顺序，就是在遍历子节点时的顺序。对二元运算对应的节点来说，先遍历左子节点再遍历右子节点就是从左到右的求值顺序，反之则是从右到左的求值顺序。

这三个概念与运算的联系都很紧密，但实际描述的是不同的关系。前两者是解析器根据语法生成AST时就已经决定好的，后者则是解释执行或者生成代码而去遍历AST时决定的。

在没有副作用的环境中，给定优先级与结合性，则无论求值顺序是怎样的都能得到同样的结果；而在有副作用的环境中，求值顺序会影响结果。

赋值运算虽然是右结合的，但仍然可以用从左到右的求值顺序；事实上Java、C#等许多语言都在规范里写明表达式的求值顺序是从左到右的。上面的例子中就先遍历的=的左侧，求得i的左值；再遍历=的右侧，得到表达式的值23；最后执行=自身，完成对i的赋值。

所以如果你要问：赋值在类似C的语言里明明是右结合的运算，为什么你先遍历左子树再遍历右子树？上面的说明应该能让你发现你把结合性与求值顺序混为一谈了。

看看Java从左到右求值顺序的例子：

一般来说 最简单的提高效率办法就是在开发中注意很多php的效率优化技巧。比如算法、语法、架构等。这些在经验和网上都能积累到很多方法。

如果你发现，效率的瓶颈不在这些无所谓的东西上。有些功能就慢在一些函数和功能上、那OK，哪里有问题就去解决哪里，用C扩展的方式解决。但是这样就有很多连带问题比如业务逻辑往往要因C扩展的方式而可能发生改变。

或者你发现，本身PHP这种解释性语言就是慢，哪里都慢（笑），怎么破。这就有了加速器，用缓存操作码的方式来避免再次解析 一些相关介绍和讲解题主可以简单看看这里

三款免费的PHP加速器：APC、eAccelerator、XCache比较

或者我们换一种思路，直接换成二进制语言执行会不会更快呢。题注可以看看 如下几种解决方案。c(Roadsend和phc)、c++(hiphop)、java(Quercus)、c#(Phalanger)。不过以hipop举例，这货

facebook/hhvm · GitHub 。是facebook的hhvm的前身，他其实就是将php转化为高度优化的C++代码然后用g++编译，还有一些其他的如code transformer。通过这些方式来从本质上将php变成另一门语言。不过这货部署之坑爹，使用之困难，调试之麻烦，编译时间之长久简直是反人类。

我觉得集上述优点之大成者，hhvm，就这么诞生了。

hhvm其实就是HipHop的改良版，目前稳定版是2.3，性能高于hiphop40%。http://hhvm.com/

语言表述不清楚，这里就不说嘛zend和hhvm的原理了，我觉得hhvm如果要和zend做对比的话就是在于hhvm如果开启了jit，hhvm就可以直接执行机器码。而zend虽然cache了opcode但是还是要通过调用C去执行机器码。

相关的文档网上有很多，题主可以关注一下

题主也可以看看这个解决方案 HippyVM，据说是专门用来打脸Facebook的

官方声称说是比原生PHP快7.x比脸书的hhvm快2x

贴一个官网看到的图