第7章 应用开发

本章内容

- 程序安装与配置
- C程序开发
- Java程序开发
- Web开发
- 版本控制

1. 程序安装与配置

- 几乎对于所有的应用，在开发之前，都需要安装和配置相关程序，并在Linux系统中设置相关的环境变量。
- 有多种不同的安装和配置方法，它们最终达到的效果一致。

安装程序与更新软件

- 软件包管理工具的作用是提供在操作系统中安装、升级、卸载所需软件的方法。
- 提供对系统中所有软件状态信息进行查询。
- 在GNU/Linux操作系统中，RPM和DPKG为最常见的两类软件包管理工具，它们分别应用于基于RPM软件包的Linux发行版本和基于DEB软件包的Linux发行版本。

(1) RPM包管理

- 一个RPM包里面包含已压缩的软件文件集以及该软件的内容信息（在头文件中保存），通常表现为以.rpm扩展名结尾的文件，例如package.rpm。
- RPM命令：

  rpm -? package.rpm，其中-?为操作参数。

  - -i：在系统中安装软件
  - -U：在系统中升级软件
  - -e：在系统中卸载软件
  - -p：对RPM包进行查询，这些参数通常组合起来使用

(1) RPM包管理（续)

- 常用的RPM包安装、升级、卸载和查询的命令:
  - 安装rpm包：rpm -ivh mypackage.rpm
  - 升级rpm包：rpm -Uvh mypackage.rpm
  - 卸载rpm包：rpm -ev mypackage
  - 查询已安装rpm包：rpm -qa | grep mypackage

(2) YUM包管理

- YUM是基于RPM包的管理工具。
- 能够从指定的源空间（服务器、本地目录等）中自动下载目标RPM包并且安装。
- 可以自动处理依赖性关系并进行下载和安装，无须繁琐地手动下载和安装每一个需要的依赖包。
- 可以对系统中所有软件进行升级。

(2) YUM包管理

- 常用的命令：
  - 安装rpm包：yum -y install mypackage.rpm
  - 升级rpm包：yum update mypackage.rpm
  - 卸载rpm包：yum remove mypackage.rpm
  - 列出已安装的rpm包：yum list
  - 列出系统中可升级的所有软件：yum check-update

(3) DEB包管理

- 一个DEB包里面包含了已压缩的软件文件集以及该软件的内容信息（在头文件中保存），通常表现为以.deb扩展名结尾的文件，例如mypackage.deb。
- 常用命令：
  - 查询系统中已安装的软件：dpkg -l mypackage
  - 安装DEB包：sudo dpkg -i mypackage
  - 卸载DEB包： #不卸载配置文件

    sudo dpkg –r mypackage

    #卸载配置文件

    sudo dpkg -P mypackage

(4) APT包管理

- APT与 YUM对应，它最早被设计成DPKG的前端软件，现在通过apt-rpm也支持RPM管理。
- APT的主要包管理工具为APT-GET。
- 常用的APT命令：
  - 更新源索引：sudo apt-get update
  - 安装包：sudo apt-get install mypackage
  - 下载指定源文件：sudo apt-get source mypackage
  - 升级所有软件：sudo apt-get upgrade
  - 卸载：sudo apt-get remove mypackage #不删配置文件

    sudo apt-get remove –purge mypackage #删配置文件

(5) ALIEN工具

- ALIEN工具可以将RPM软件包转换成DEB软件包，或把DEB软件包转换成RPM软件包。
- 在Ubuntu中使用ALIEN将DEB转换为RPM并安装：
  - 转换：sudo alien -d mypackage.rpm
  - 安装：sudo dpkg -i mypackage.deb
- 在RHEL中使用ALIEN将DEB转换为RPM并安装：
  - 转换：sudo alien -r mypackage.deb
  - 安装：sudo rpm -ivh mypackage.rpm

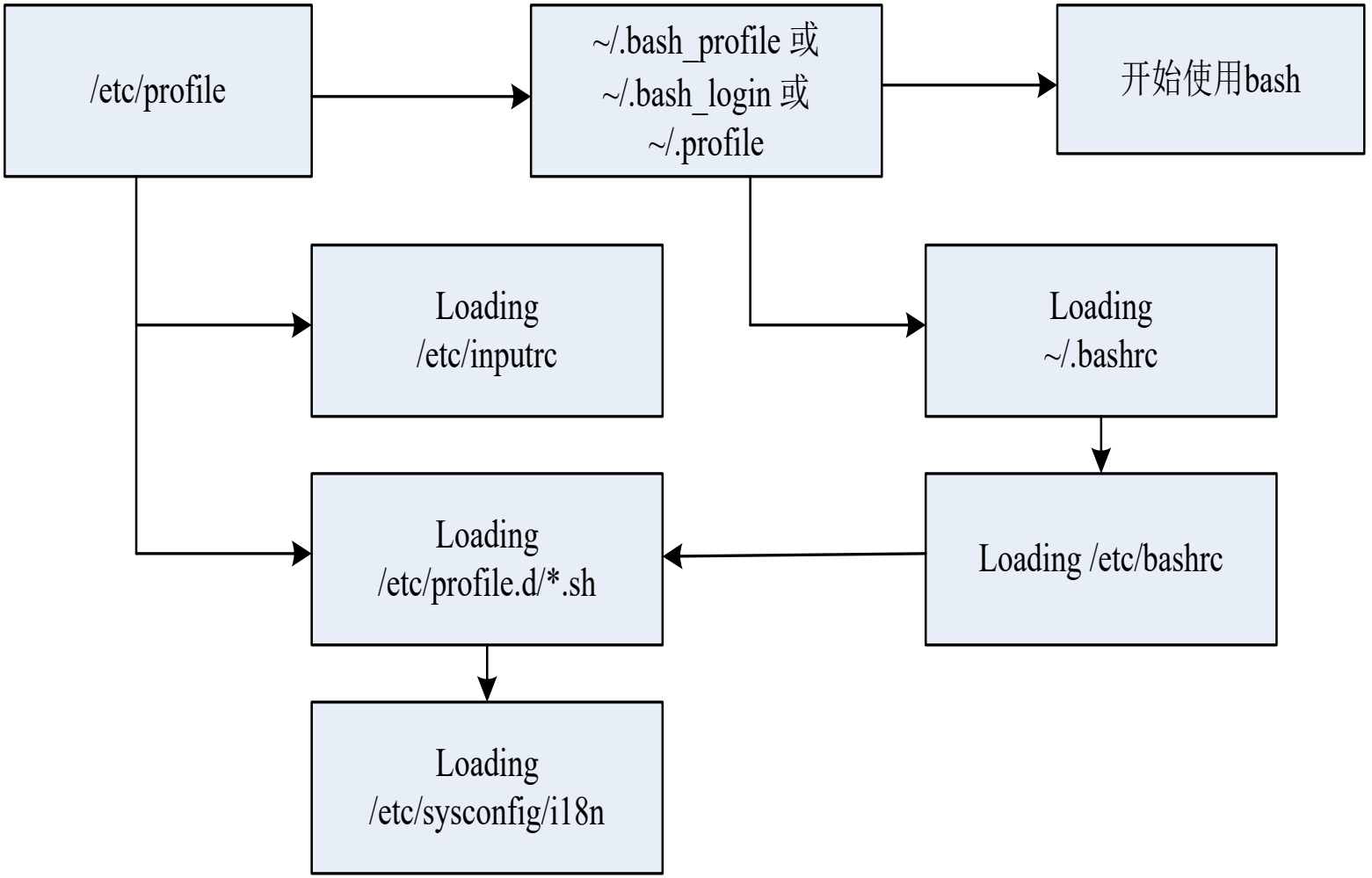
环境变量

- 环境变量和shell紧密相关，因为用户登录系统后就启动了一个shell。对于Linux来说通常这个shell是bash，但也可以重新设定或切换到其它shell。
- 根据发行版本的不同，bash 有两个基本的系统级配置文件：/etc/bashrc和/etc/profile。这些配置文件包含两组不同的变量：
  - shell变量，只是在特定的 shell 中固定（如bash），是局部的。
  - 环境变量，在不同 shell 中固定，是全局的。环境变量是通过shell命令来设置，设置好的环境变量又可以被所有当前用户所运行的程序所使用。

设置环境变量

- 对于bash这个shell程序：
  - 可以通过变量名来访问相应的环境变量
  - 通过export来设置环境变量
  - 与环境变量相关的文件见下页

与环境变量相关的文件


/etc/profile
~/.bash_profile 或
~/.bash_login 或
~/.profile
开始使用bash
Loading
/etc/inputrc
Loading
~/.bashrc
Loading
/etc/profile.d/*.sh
Loading /etc/bashrc
Loading
/etc/sysconfig/i18n

环境变量设置方法

(1) 使用命令echo显示环境变量

例：$ echo $HOME

(2) 设置一个新的环境变量

例：$ export HELLO="Hello!"

(3) 使用env命令显示所有的环境变量

例： $ env

(4) 使用set命令显示所有本地定义的 shell 变量

例：$ set

环境变量设置方法（续）

(5) 使用unset命令来清除环境变量

例：$ unset $TEST　#删除环境变量TEST

(6) 使用readonly命令设置只读变量，注意：如果使用了readonly命令的话，变量就不可以被修改或清除了。

(7) 使用命令echo显示环境变量

例：　# 增加一个环境变量 TEST

$ export TEST="Test..."

#将环境变量TEST设为只读

$ readonly TEST

环境变量设置方法（续）

（8）通过修改环境变量定义文件来修改环境变量

- **注意：一般情况下，这仅仅对于普通用户适用；要尽量避免修改根用户的环境变量定义文件，因为那样可能会造成潜在的危险。**

  例：$ cd ~　#切换到用户目录

  $ ls -a　#查看当前目录下所有文件和子目录，包含隐藏的文件

  $ vi .bash_profile　　# 修改环境变量定义文件

- **环境变量更改后，在用户下次登录时生效，如果想立刻生效，则可执行以下命令：**

  $ source .bash_profile

2. C 程序开发

- 近年来，Linux操作系统在嵌入式系统领域的延伸可谓如日中天，许多版本的嵌入式Linux系统被开发出来。
- ucLinux、RTLinux、ARM-Linux等。
- 基于Linux的C语言编程也成为社会所亟需的方法。

GCC

- GCC是一套由GNU开发的编程语言编译器。
- 一个C/C++程序从开始编码到生成可执行的二进制文件至少需要经过4个步骤：
  - 预处理（Preprocessing）
  - 编译（Compilation）
  - 汇编（Assembly）
  - 连接（Linking）
- GCC的基本使用格式：

  $ GCC [选项] <文件名>

GCC常用选项（1）

选项	含义	举例与说明
无	没有任何选项	GCC hello.c 产生一个可执行文件a.out
-o	指定输出文件名，默认为a.out	GCC -o hello hello.c 产生一个输出文件hello
-c	只编译，不汇编连接	GCC -c hello.c 产生一个目标文件hello.o
-g	产生供gdb调试用的可执行文件	GCC -g hello.c 产生一个可执行文件a.out，大小明显比只用-o选项编译汇编连接后的文件大
-O[0、1、2、3]	设置优化级别，level可以是0、1、2、3，默认为-O 0，即不进行优化	GCC -O -o test hello.c 产生一个经过优化的可执行文件test
-Dname[=definition]	在命令行上定义宏	GCC –Dlen=20 hello.c 等价于在hello.c的头文件中定义：#define len 20

GCC常用选项（2）

选项	含义	举例与说明
-Idir	把dir加到头文件的搜索路径中，而且GCC会在搜索标准头文件之前先搜索dir	GCC –I/home/linux/include -o test hello.c
-Ldir	把dir加到库文件的搜索路径中，而且GCC会在搜索标准库文件之前先搜索dir	GCC -L/home/linux/lib/ hello.c 在库文件搜索路径中添加/home/linux/lib目录
-llibrary	在连接的时候搜索library库	GCC -lapp hello.c 自动连接名为libapp.so的库文件
-w	禁止输出警告信息	GCC -w hello.c
-Wwarning	设置警告，可以设置的警告开关很多，通常用-Wall开启所有的警告	GCC –Wall hello.c 显示所有的警告信息

make工具与makefile文件

- make工具程序的主要用途是能自动地决定一个含有很多源程序文件的大型程序中哪个文件需要被重新编译。详细说明可以参考GNU make使用手册。
- makefile文件是make工具程序的配置文件，makefile文件会告诉make如何编译和连接一个文件。
- make的执行过程分为两个不同的阶段:
  - 第一阶段，它读取所有的makefile文件内容，记录所有的变量及其值、隐式或显式的规则，并构造所有目标对象及其先决条件的全景图。
  - 第二阶段，make使用这些内部结构来确定哪个目标对象需要被重建，并且使用相应的规则来操作。

make工具工作原理

- 当make重新编译程序时，每个修改过的c代码文件必须被重新编译。
- 如果头文件被修改，那么每一个包含该头文件的c代码程序都将被重新编译。
- 每次编译操作都产生一个与原程序对应的目标文件。
- 最后，当任何源代码文件都被编译后，所有的目标文件不管是刚编译的还是以前编译的，将连接在一起从而生成新的可执行文件。

Make使用方法

- make的使用方法：$ make
- 解释：它会到当前目录下寻找makefile文件，然后依照makefile中所记录的步骤一步一步地执行。
- 简单的makefile文件包含一些规则，形式为：

  目标(target): 先决条件(prerequisites)

  　　命令(command)

- “目标”可以是两种情况：
  - 其一，程序生成的一个文件的名称，如可执行文件或目标文件；
  - 其二，所要采取的动作，如“清除”(clean)。
- “先决条件”是一个或多个文件名，用作产生目标的输入条件。一个目标可能依赖多个文件。

makefile文件规则

- “命令”是make执行的具体操作。
- 一个规则可以包含多个命令，每个命令一行。注意：命令行之前需要输入一个制表符“Tab”。
- 自动变量是一种在命令行上根据具体情况能被自动替换的变量，其值基于目标对象及其先决条件。
  - 如“$^”表示规则的所有先决条件，包含他们所在目录的名称；
  - “$<”表示规则中的第一个先决条件；
  - “$@”表示目标对象。

C 程序开发举例

- 假定/home/linux/mypro目录下有
  - functions/greeting.h
  - functions/greeting.c
  - my_app.c
- 说明:
  - greeting.h文件定义了函数greeting
  - greeting.c文件实现了该函数
  - 文件my_app.c则调用了该函数

C程序示例

代码	文件名及其说明
/***greeting.h源代码***/ 1. #ifdef_GREETING_H 2. #define_GREETING_H 3. void greeting (char * name); 4. #endif	functions/greeting.h
/***greeting.c源代码***/ 1. #include <stdio.h> 2. #include “greeting.h” 3. void greeting (char * name){ 4. printf(“Hello $s!\r\n”,name); 5. }	functions/greeting.c

C程序示例（续）

代码	文件名及其说明
/***my_app.c源代码***/ 1. #include <stdio.h> 2. #include “greeting.h” 3. #define N 10 4. int main(void){ 5. char name[N]; 6. printf(“Your name please:”); 7. scanf(“%s”,name); 8. greeting(name); 9. return 0; 10. }	my_app.c

C程序示例（续）

代码	文件名及其说明
/***makefile源代码***/ 1. OBJS = my_app.o greeting.o 2. CC = GCC 3. CFLAGS = -Wall -O –g 4. my_app : $(OBJS) $(CC) $(OBJS) -o my_app 5. greeting.o: functions\greeting.c functions\greeting.h $(CC) $(CFLAGS) -c functions\greeting.c 6. my_app.o : my_app.c functions\greeting.h $(CC) $(CFLAGS) -c my_app.c -Ifunctons 7. clean: rm -rf *.o my_app	文件makefile 1. 定义变量OBJS 2. 定义变量CC 3. 配置编译器设置，并把它赋值给CFLAGS变量，-Wall指输出所有的警告信息，-O指在编译时进行优化，-g指编译debug版本 4. my_app依赖于目标文件 5. 编译产生目标文件greeting.o 6. 编译产生目标文件my_app.o 7. 删除编译过程中产生的所有.o文件

编译过程

- 编译过程：
- 首先切换到目录“/home/linux/mypro”，然后键入“make”即可编译。

```
$ cd /home/linux/mypro
$ make
```

- 运行过程：在目录“/home/linux/mypro”下，键入“./my_app”即可。

```
$ ./my_app
```

3. Java程序开发

- Java由Sun公司研发，其程序的运行依赖于Java虚拟机JVM。
- JDK是整个Java的核心，包括了Java运行环境、一堆Java工具和Java基础的类库。不论哪种Java应用服务器，其实质都是内置了某个版本的JDK。
- 最主流的JDK是Sun公司发布的JDK，还有很多其它公司和组织也都开发了自己的JDK，例如IBM公司开发的JDK，BEA公司的Jrocket，还有GNU组织开发的JDK等等。

Ant

- Ant是Apache基金会下的一个跨平台的构件工具，它可以实现项目的自动构建和部署等功能。
- Ant最早用来构建著名的Tomcat，其作者创作它的动机就是因为受不了makefile的语法格式。
- Ant可看成是一个Java版本的make，也正因为使用了Java，所以Ant是跨平台的。
- Ant使用XML定义构建脚本，相对于makefile来说，也更加友好。
- 与make类似，Ant也有一个构建脚本build.xml。

Ant构建脚本

build.xml源码	备注
<?xml version="1.0"?> <project name="Hello" default="compile"> <target name="compile" description="compile the Java source code to class files"> <mkdir dir="classes"/> <javac srcdir="." destdir="classes"/> </target> <target name="jar" depends="compile" description="create a Jar file "> <jar destfile="hello.jar"> <fileset dir="classes" includes="**/*.class"/> <manifest> <attribute name="Main.Class" value="HelloProgram"/> </manifest> </jar> </target> </project>	build.xml的基本结构也是目标（target）、依赖（depends），以及实现目标的任务。 比如jar目标用来创建应用程序jar文件，该目标依赖于compile目标，后者执行的任务是创建一个名为classes的文件夹，编译当前目录的java文件至classes目录。 compile目标完成后，jar目标再执行自己的任务。

Ant构建

- 和Make一样，Ant也都是过程式的，开发者必须显式地指定每一个目标，以及完成该目标所需要执行的任务。
- 例如你必须明确告诉 Ant 源码在哪里，输出的字节码要存储在哪里，如何将这些字节码打包成JAR文件等。
- 虽然Ant最近有些进展用来帮助减化程序，但对于开发者而言，Ant即使用XML来编写程序。

Maven

- make和makefile用于C/C++软件自动编译，Yum等用于Linux应用程序管理，Ant用于Java源码编译，java项目工程的包管理和源码管理则是Maven。
- 作为Apache组织中一个成功的开源项目，Maven主要服务于基于Java平台的项目构建、依赖管理和项目信息管理。无论是小型的开源类库项目，还是大型的企业级应用；无论是传统的瀑布式开发，还是流行的敏捷模式，Maven都能大显身手。
- 在Maven中，要从Java源码创建一个JAR文件，只需要创建一个简单的pom.xml。Maven与ant的区别见表7-4所示。

Maven与Ant的区别

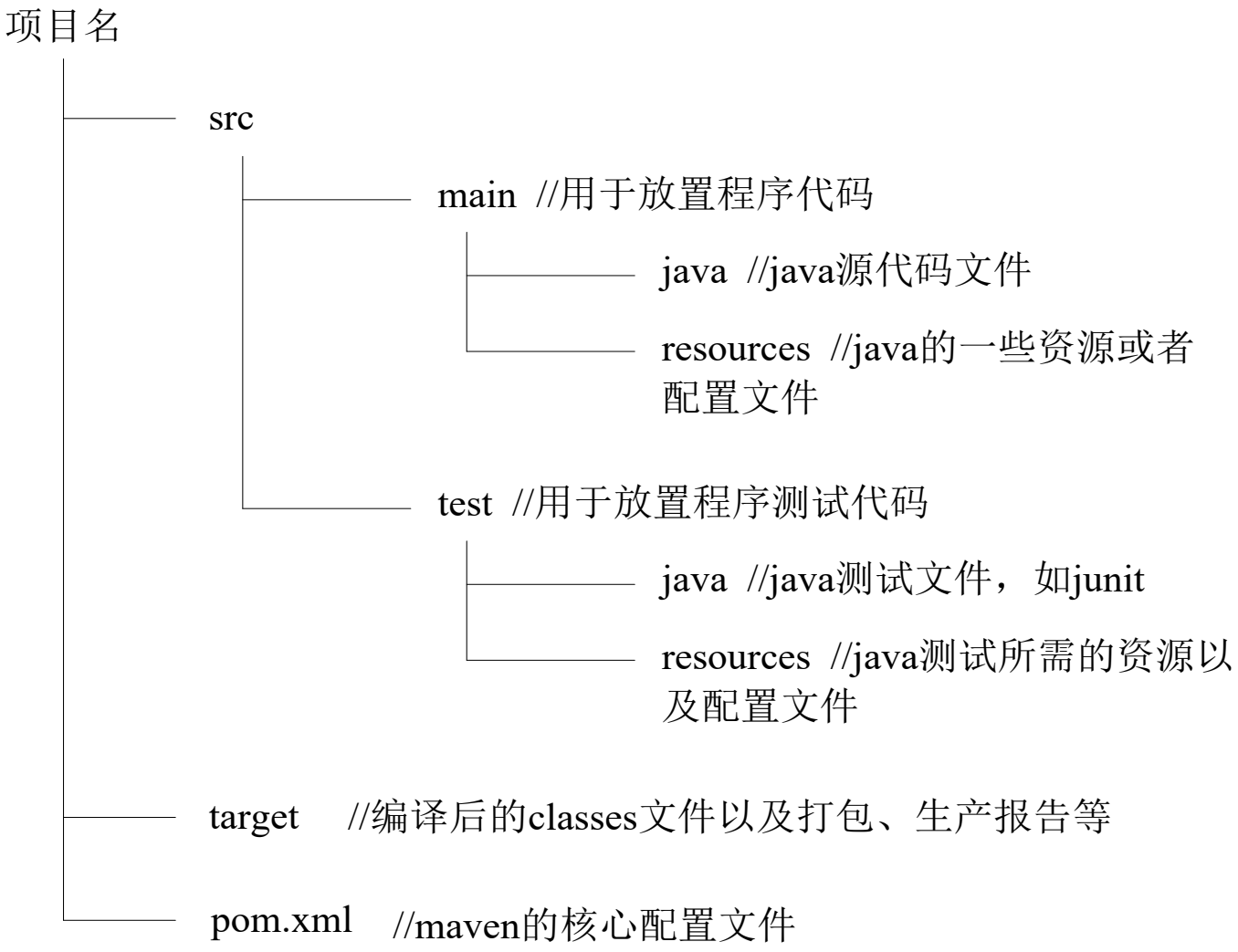
	Apache Ant	Apache Maven
目录结构	Ant 没有正式的约定，即没有一般项目的目录结构，但是必须明确告诉Ant其源代码和输出的位置。	Maven 拥有约定，它把字节码放到target/ classes ，然后把target 生成一个 JAR 文件。
处理方式	Ant 是程序化的，必须明确地告诉 Ant 做什么，什么时候做，以及编译、复制和压缩过程。	Maven 是声明式的，只需创建一个pom.xml 文件，然后将源代码放到默认的目录中，Maven 会处理。
生命周期	Ant 没有生命周期，必须定义目标和目标之间的依赖，必须手工为每个目标附上一个任务序列。	Maven 有一个生命周期，当运行mvn install 时，Maven 执行一系列有序的步骤，直到到达指定的生命周期。

Maven下载与安装

- Maven以及Ant的相关下载、安装与配置可以参考官网http://maven.apache.org/和http://ant.apache.org/。

- Maven的源代码目录组织结构是固定的，不能随便修改，一个简单的Java程序目录结构见下页。

基于Maven的Java源代码目录结构



Eclipse

- Eclipse是著名的跨平台开源IDE。最初主要用于Java程序开发，目前通过插件形式也成为C++、Python、PHP等其它语言的开发工具。
- Eclipse的本身只是一个框架平台，但是众多插件的支持，使得Eclipse拥有非常好的灵活性。
- 许多软件开发商都以Eclipse作为框架来开发自己的IDE。
- Eclipse的安装与配置可参考官网 http://www.eclipse.org/

4. Web开发

- 基于LAMP的架构具有成本低廉、部署灵活、快速开发、安全稳定等特点，是Web开发的优秀组合。
- LAMP（Linux+Apache+MySql+PHP的缩写），指一组通常组合在一起并用于运行动态网站或者服务器的自由软件，其中：
  - Linux是操作系统
  - Apache是网页服务器
  - MySQL是数据库管理系统或者数据库服务器
  - PHP是脚本语言。

LAMP

- 对于M除了表示MySQL之外，还可以将M的概念扩展到Memcached。
- Memcached是一个高性能的分布式内存对象缓存系统，通过在内存中维护一个统一的巨大的Hash表，用来存储各种数据，包括图像、视频、文件以及数据库检索的结果等。
- 而且，随着互联网Web2.0网站的兴起，非关系型数据库NoSQL则由于其本身的特点得到了非常迅速的发展。
- 具有代表性的NoSQL开源软件包括：Membase、MongoDB、Hypertable、Apache Cassandra、CouchDB等。

LAMP加速

- LAMP的加速可以从几个方面来考虑：

（1）Web服务器的加速

（2）缓存加速

（3）PHP加速

（4）数据库优化

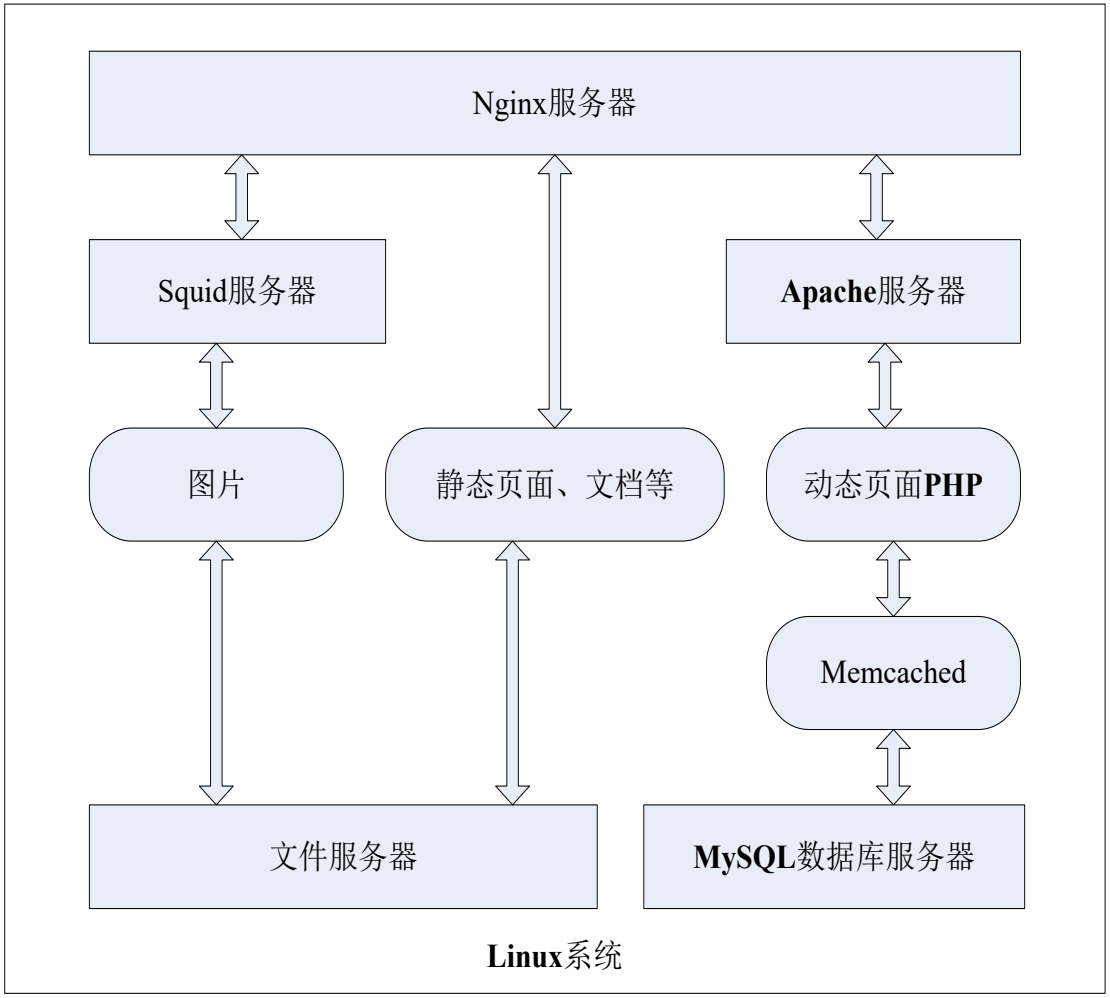
(1) Web服务器的加速

- Apache是LAMP架构最核心的Web Server，开源、稳定、模块丰富是Apache的优势。
- 但Apache的缺点是有些臃肿，内存和CPU开销大，性能上有损耗，不如一些轻量级的Web服务器（例如Nginx）高效。
- Nginx是一个高性能的HTTP和反向代理服务器，它不支持PHP和CGI等动态语言，但支持负载均衡和容错，可和Apache配合使用，是轻量级的HTTP服务器的首选。

(2) 缓存加速

- Apache提供了自己的缓存模块，也可以使用外加的Squid模块进行缓存，均可有效地提高Apache的访问响应能力。
- Squid cache是一个Web缓存服务器，可以作为网页服务器的前置cache服务器缓存相关请求来提高Web服务器的速度。
- 当访问量非常大时可考虑使用Memcached作为分布式缓存。

(3) PHP加速



LAMP网站架构图

PHP加速常用解决方案

- Squid + Apache + PHP + eAccelerator
  - 使用Apache负载PHP，使用Squid进行缓存，HTML或图片的请求直接由Squid返回给用户。很多大型网站都采用这种架构。
- Nginx/Apache + PHP(fastcgi) + eAccelerator
  - 使用Nginx或Apache负载PHP，PHP使用fastcgi方式运行，效率较高。
- Nginx + Apache + PHP + eAccelerator
  - 此方案综合了Nginx和Apache的优点，使用Apache负载PHP，Nginx负责解析其它Web请求，Apache端口不对外开放。

(4) 数据库优化

- 开源的数据库中，MySQL在性能、稳定性和功能上是首选，可以达到百万级别的数据存储，网站初期可以将MySQL和Web服务器放在一起。 ↓
- 但是当访问量达到一定规模后，应该将MySQL数据库从Web Server中独立出来，在单独的服务器上运行，同时保持Web Server和MySQL服务器的稳定连接。
- 当数据库访问量达到更大级别时，可以考虑使用MySQL Cluster数据库集群等方式来解决问题。

LAMP的几个经典应用

（1）Wiki网站搭建

- MediaWiki是全球最著名的开源Wiki引擎，运行于LAMP环境。从 2002年开始作为维基百科全书的系统软件，并有大量其它应用实例。
- 目前MediaWiki的开发得到维基媒体基金会的支持，MeidaWiki一直保持着持续更新。
- LAMP和MediaWiki可以组成一个优秀的维基网站，可以在互联网上运行，也可以在Linux局域网中运行。
- 用户可以低成本地发布、更新和维护自己的维基网站，在预算短缺的情况下，LAMP + MediaWiki组成维基网站具备明显的价格优势。

LAMP的几个经典应用（续）

（2）其它网站

- **Yahoo！（雅虎)是全球著名的互联网公司，其Web服务器的体系架构采用LAMP，而且Yahoo！还参与LAMP本身的开发工作，包括Apache分布式数据处理。其它几个网站见下表。**

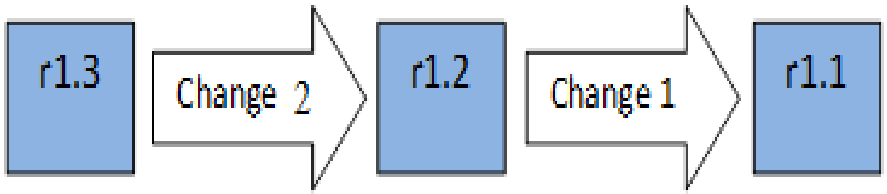
网站	系统	服务器	存储	脚本
Yahoo！	FreeBSD+Linux	Apache	MySQL	PHP
Facebook	FreeeBSD	Apache	MySQL+Memcached	PHP
Sina	FreeBSD+Solaris	Apache+Nginx	MySQL+Memcached	PHP
YouTube	Suse Linux	Apache	MySQL	Python
Flickr	Redhat Linux	Apache	MySQL+Memcached	PHP+Perl

5. 版本控制

- 版本控制软件提供完备的版本管理功能，用于存储、追踪目录（文件夹）和文件的修改历史，是软件开发者的必备工具，是软件公司必不可少的基础设施。
- 版本控制软件的最高目标，是支持软件公司的配置管理活动，追踪多个版本的开发和维护活动，及时发布软件。

VCS

- VCS（Version Control System，版本控制系统）主要用于管理开发过程中的源代码文件。
- 最早期较流行的VCS是70年代用于Unix平台的 RCS（Revision Control System，程序改版控制系统），采用C开发，能对文件进行集中式管理，已经被替代，现在很少有人在使用。



RCS历史版本追踪

CVS

- 90年代CVS（Concurrent Version System，并发版本系统）被应用在很多重要的开源项目中，它本身也是开源的，是一种GNU软件包。
- CVS继承了RCS的集中管理理念，采用复制-修改-合并（copy-modify-merge）的模式，实现协作式开发。
- 不足之处：
  - 合并不是原子操作。
  - 文件的附加信息没有被追踪。
  - 主要用于管理ASCII文件。
  - 分支和合并耗费时间按长。

Subversion

- Apache Subversion（简称SVN），是一个开放源代码的版本控制系统，2001年诞生，它的设计目标就是取代CVS。
- SVN在许多方面沿袭CVS，也是集中管理库。但CVS和SVN又有许多不同：
  - 文件存储模式不同。SVN采用关系型数据库来存储改变集，因此版本控制的相关数据变得不透明。
  - 版本管理对象不同。CVS中的版本是针对某个文件的，CVS中每次commit生成一个文件的新版本；SVN中的版本是针对整个文件系统的（包含多个文件以及文件组织方式），每次commit生成一个整个项目文件系统树的新版本。

Subversion（续）

- Subversion依赖类似于硬链接的方式来提高效率，避免过多的复制文件本身。
- Subversion不会从库下载整个主干到本地，而只是下载主干的最新版本。
- Subversion已经是Apache中自带的一个模块了，Subversion已经被应用于GCC、SourceForge等项目，并依然有活跃的开发，而CVS则逐渐沉寂。

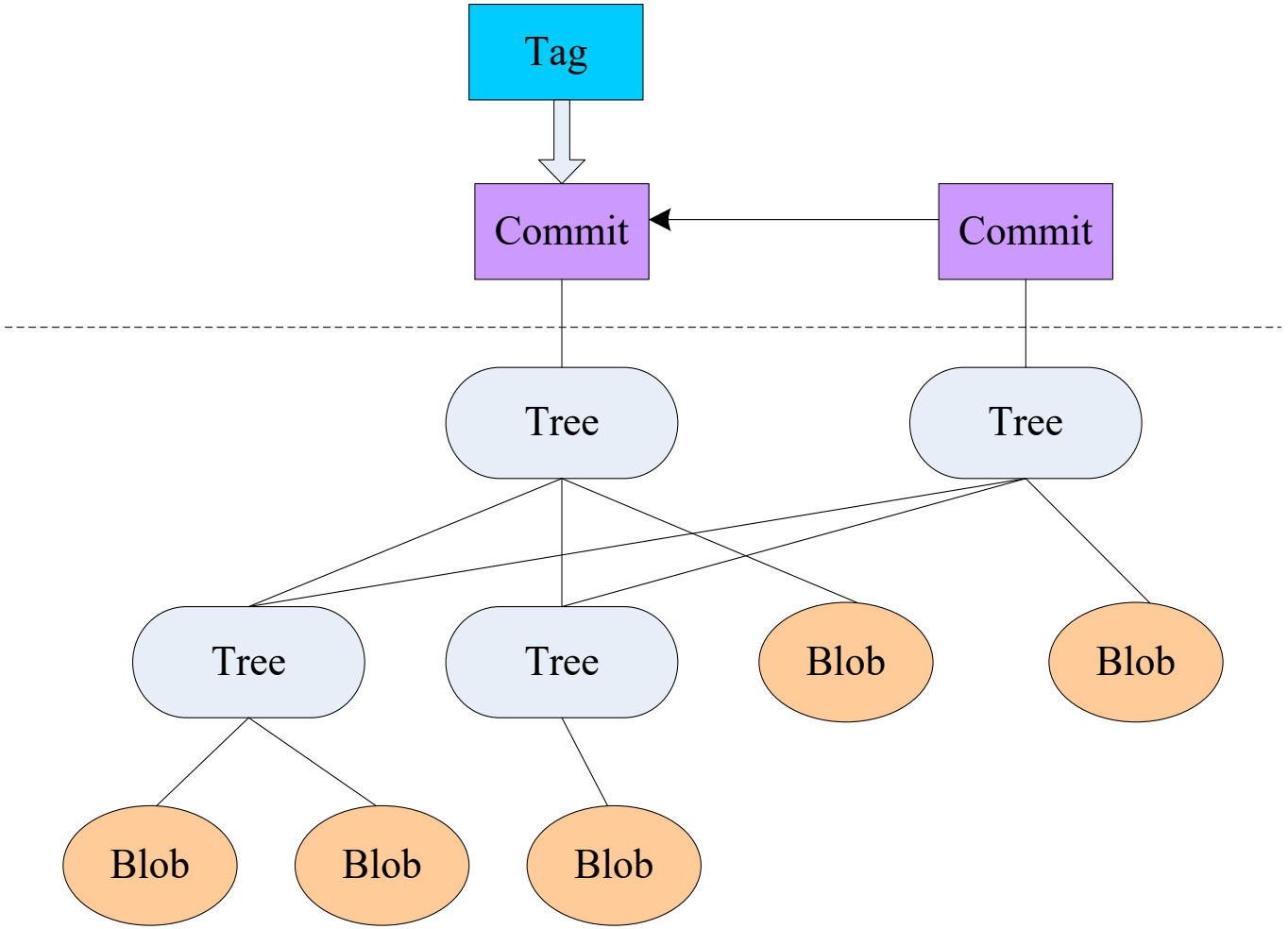
Git

- Git的作者也是Linus Torvald。
- Linux内核小组最初使用.tar文件来管理内核代码，但这远远无法跟上Linux内核代码的增长速度。
- Linus转而使用BitKeeper作为开发的VCS工具。
- BitKeeper是一款分布式的VCS工具，它可以快速地进行分支和合并，然而BitKeeper是闭源软件，由于使用证书方面的争议。
- Linus最终决定写一款开源的分布式VCS软件，即Git。

Git（续）

- 对于一个开发项目，Git会保存Blob、Tree、Commit和Tag四种对象。
  - 文件被保存为Blob对象。
  - 文件夹被保存为Tree对象。Tree对象保存有指向文件或者其它Tree对象指针。
  - 一个Commit对象代表了某次提交，它保存修改人、修改时间和附加信息，并指向一个文件树。这一点与Subversion类似，即每次提交为一个文件系统树。
  - 一个Tag对象包含有Tag的名字，并指向一个Commit对象。

Git更新过程


Tag
Commit
Commit
Tree
Tree
Tree
Tree
Blob
Blob
Blob
Blob
Blob

本章小结

- Linux系统几乎支持所有的应用开发。
- Linux系统下的C语言编程采用GCC这个GNU编译器套装、make工具和makefile文件，支持复杂的Linux内核代码的编写和自动编译。
- Java语言编程采用Ant、Maven和Eclipse开发平台，支持大型的企业级工程。
- 基于LAMP的架构具有成本低廉、部署灵活、快速开发、安全稳定等特点，是Web开发的优秀组合，关于LAMP的几个经典应用足以证明LAMP的强大和普适。
- 版本控制软件是软件开发者的必备工具，是软件公司必不可少的基础设施。