附录 A HTML 特殊字符

显示结果	描述	实体名称	实体编号
	空格		
<	小于号	<	<
>	大于号	>	>
&	和号	&	&
"	引号	"	"
'	撇号	'（IE 不支持）	'
￠	分	¢	¢
￡	镑	£	£
￥	日元	¥	¥
?	欧元	€	€
§	小节	§	§
©	版权	©	©
®	注册商标	®	®
™	商标	™	™
×	乘号	×	×
÷	除号	÷	÷

附录 B 云计算介绍

B.1 云计算历史

首先让我们来回顾一下云计算的历史：

- 最早提出云计算概念的人是斯坦福大学的科学家 John McCarthy 在 1960 年提出“计算机可能变成一种公共资源”；
- 1966 年 Douglas Parkhill 在他的著作 The challenge of computer Utility 中对这个理论提出了更深刻的解释，并且提出了私有资源，公有资源，社区资源等概念；
- 1966 年，著名的 ARPAN（Advanced Research Projects Agency Network）的负责人 J.C.R.Licklider 提出了“从任意点通过网络访问计算机程序”的设想；
- 1983 年，太阳电脑（Sun Microsystems）提出“网络是电脑”（“The Network is the Computer”）；
- 1997 年，Ramnath Chellappa 教授在他的演讲中提出“云计算/Cloud Computing”这个词；
- 1999 年 Salesforce.com 是现在公认的最早的云计算公司，提供基于云的 SaaS 服务；
- 2006 年 3 月，亚马逊（Amazon）推出弹性计算云（Elastic Compute Cloud；

EC2）服务；

- 2006 年 8 月 9 日，Google 首席执行官埃里克·施密特（Eric Schmidt）在搜索引擎大会（SES San Jose 2006）首次提出“云计算”（Cloud Computing）的概念。Google“云端计算”源于 Google 工程师克里斯托弗·比希利亚所做的“Google 101”项目；
- 2007 年 10 月，Google 与 IBM 开始在美国大学校园，包括卡内基梅隆大学、麻省理工学院、斯坦福大学、加州大学柏克莱分校及马里兰大学等，推广云计算的计划，这项计划希望能降低分布式计算技术在学术研究方面的成本，并为这些大学提供相关的软硬件设备及技术支持（包括数百台个人电脑及 BladeCenter 与 System x 服务器，这些计算平台将提供 1600 个处理器，支持包括 Linux、Xen、Hadoop 等开放源代码平台）。而学生则可以通过网络开发各项以大规模计算为基础的研究计划；
- 2008 年 1 月 30 日，Google 宣布在台湾启动“云计算学术计划”，将与台湾台大、交大等学校合作，将这种先进的大规模、快速将云计算技术推广到校园；
- 2008 年 2 月 1 日，（NYSE：IBM）宣布将在中国无锡太湖新城科教产业园为中国的软件公司建立全球第一个云计算中心（Cloud Computing Center）；
- 2008 年 7 月 29 日，雅虎、惠普和英特尔宣布一项涵盖美国、德国和新加坡的联合研究计划，推出云计算研究软件测试床，推进云计算。该计划要与合作伙伴创建 6 个数据中心作为研究试验平台，每个数据中心配置 1400 个至 4000 个处理器。这些合作伙伴包括新加坡资讯通信发展管理局、德国卡尔斯鲁厄大学 Steinbuch 计算中心、美国伊利诺伊大学香宾分校、英特尔研究院、惠普实验室和雅虎；
- 2008 年 8 月 3 日，美国专利商标局网站信息显示，戴尔正在申请“云计算”（Cloud Computing）商标，此举旨在加强对这一未来可能重塑技术架构的术语的控制权；
- 2010 年 3 月 5 日，Novell 与云安全联盟（CSA）共同宣布一项供应商中立计划，名为“可信任云计算计划（Trusted Cloud Initiative）”。
- 2010 年 7 月，美国国家航空航天局和包括 Rackspace、AMD、Intel、戴尔等支持厂商共同宣布“OpenStack”开放源代码计划，微软在 2010 年 10 月表示支持 OpenStack 与 Windows Server 2008 R2 的集成；而 Ubuntu 已把 OpenStack 加至 11.04 版本中；
- 2011 年 2 月，思科系统正式加入 OpenStack，重点研制 OpenStack 的网络服务。

B.2 云计算基础知识

这里来介绍一下建立云计算所用到的一些基础知识：

B.2.1 互联网技术

云计算是互联网技术的一个延伸，云计算背后的互联网技术是基于一系列的标准和协议的，它使得客户可以在任何地方访问任何设备中的数据。

B.2.2 虚拟化技术

虚拟化是一个广义的术语，在计算机方面通常是指计算元件在虚拟的基础上而不是真实的基础上运行。虚拟化技术可以扩大硬件的容量，简化软件的重新配置过程。CPU 的虚拟化技术可以单 CPU 模拟多 CPU 并行，允许一个平台同时运行多个操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率。

虚拟化技术可以在云服务器上创建更多的逻辑硬盘，内存等虚拟硬件设备以及操作系统，中间件等虚拟软件设备。

B.2.3 面向服务的体系架构（SOA）

面向服务的体系结构，是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。

SOA 基于标准格式与标准协议提供服务，为云计算中各个服务提供统一的接口做出了很大的贡献。

B.2.4 网格计算

网格计算即分布式计算，是一门计算机科学。它研究如何把一个需要非常巨大的计算能力才能解决的问题分成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终结果。

显而易见，通过网格计算可以通过一系列小功能的机器实现大规模的数据运算提供了可能，网格计算在云计算中起到了非凡的作用。

B.2.5 按量使用付费模型

云计算是通过使用互联网流量以及存储器使用空间进行收费的。典型的按量使用付费模型最常见的是我们日常生活中使用的水，电，气。

B.3 云计算定义

那么到底什么是云计算呢？由于云计算的概念太抽象了，就像当年如何定义“无线局域网”或“802.1X”一样，业界有各种各样的定义标准，并且每个标准都有一定的意义。到目前为止关于云计算的定义就超过了 100 中。在这里，我们在这里给出 CSA（Cloud Security Alliance）云计算安全联盟在 Security Guidance For Critical Area of Focus In Cloud Computing V3.0 的定义：

“云计算的本质是一种服务提供模型，通过这种模型可以随时，随地，按需地通过网络访问共享资源池的资源，这个资源池的内容包括计算资源，网络资源，存

储资源等，这些资源能够被动态地分配和调整，在不同用户之间灵活划分。凡是符合这些特征的 IT 服务都可以被称作云计算服务。”

B.4 云计算的标准

NIST（U.S. National Institute of Standards and Technology）美国国家标准与技术学院给出了一个云计算的标准——“NIST Working Definition of Cloud Computing/NIST 800-145”，它由五个基本特征、三个服务模型和四个发布模型组成。

B.4.1 五个基本特性

1，按需自助服务

视客户的需要，可以从每个服务提供商那里单方面地向客户提供计算能力，比如，服务器时间和网络存储，而这些是自动进行无需干涉的。自助服务是区分 B/S 架构与云计算的重要的标准。

2，广泛的网络访问

具有通过规范机制网络访问的能力，这种机制可以使用各种各样的客户端平台（例如，PC、笔记本电脑以及 PDA）。比如您是位作家，即使您手头的机器上没有安装 MS Word 等类似编辑文件功能的软件，或者使用的是 PAD 或者智能手机，您也可以随时随地的可以登录到 Google Doc 中去进行您的写作工作。

3，资源共享

提供商提供的计算资源被集中起来通过一个多客户共享模型来为多个客户提供服务，并根据客户的需求，动态地分配或再分配不同的物理和虚拟资源。资源包括以下类型：存储，计算能力，内存，网络带宽和虚拟环境等。

4，快速的可伸缩性

具有快速地可伸缩性地提供服务的能力。在一些场景中，所提供的服务可以自动地，快速地横向扩展，在某种条件下迅速释放、以及快速横向收缩。对于客户来讲，这种能力用于使所提供的服务看起来好像是无限的，并且可以在任何时间、购买任何数量。

5，可度量的服务

云系统通过一种可计量的能力杠杆在某些抽象层上自动地控制并优化资源以达到某种服务类型（例如，存储、处理、带宽以及活动用户账号）。资源的使用可以被监视和控制，通过向供应商和用户提供这些被使用服务报告以达到透明化。并且以此作为收费的基础依据。

B.4.2 服务模型

1，软件即服务 （SaaS）

客户所使用的服务商提供的这些应用程序运行在云基础设施上。这些应用程序可以通过各种各样的客户端设备所访问，通过瘦客户端界面像 WEB 浏览器（例如，基于 WEB 的电子邮件）。客户不管理或者控制底层的云基础架构，包括网络、服务器、操作系统、存储设备，甚至独立的应用程序机能，在可能异常的情况下，限制用户可配置的应用程序设置。

2，平台即服务 （PaaS）

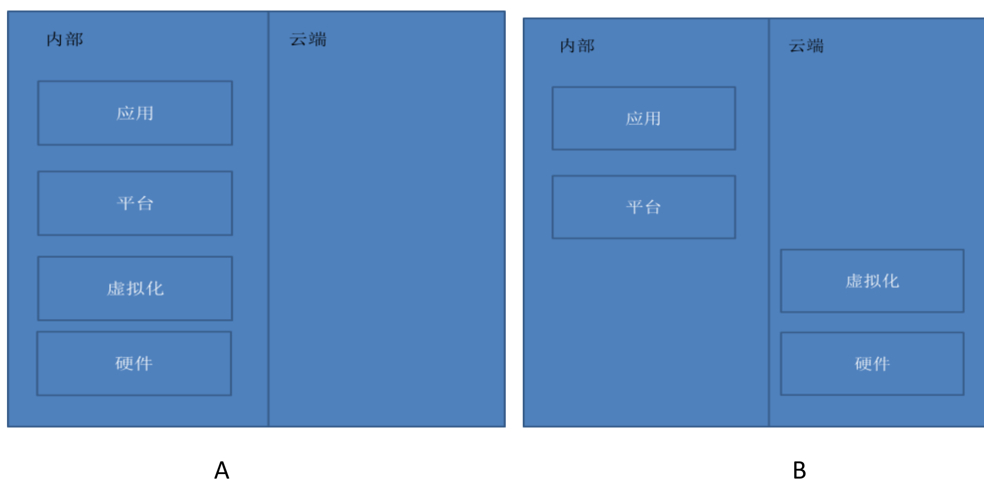
客户使用云供应商支持的开发语言和工具，开发出应用程序，发布到云基础架构上。客户不管理或者控制底层的云基础架构，包括网络、服务器、操作系统或者存储设备，但是能控制发布应用程序和可能的应用程序运行环境配置。

3，架构即服务 （IaaS）

向客户提供处理、存储、网络以及其他基础计算资源，客户可以在上运行任意软件，包括操作系统和应用程序。用户不管理或者控制底层的云基础架构，但是可以控制操作系统、存储、发布应用程序，以及可能限度的控制选择的网络组件（例如，防火墙）。

4，小结

为了更好解释这三种服务模型，我们看一下图 B-1。图 B-1（A）是没有购买云服务的情形，所有设备都部署在组织内部；图 B-1（B）是购买了 IaaS 后的情形，用户仅使用云端的硬件设备；图 B-1（C）是购买了 PaaS 后的情形，用户不仅使用云端的硬件设备，还使用了云端的平台（比如数据库，开发工具，WEB 服务器等等）；图 B-1（D）是购买了 SaaS 后的情形，除了硬件系统，平台搭建在云上，而且把应用也部署在云上了，也就是说购买了 SaaS 后，开发商把所有的软硬件设备都构建在云端。



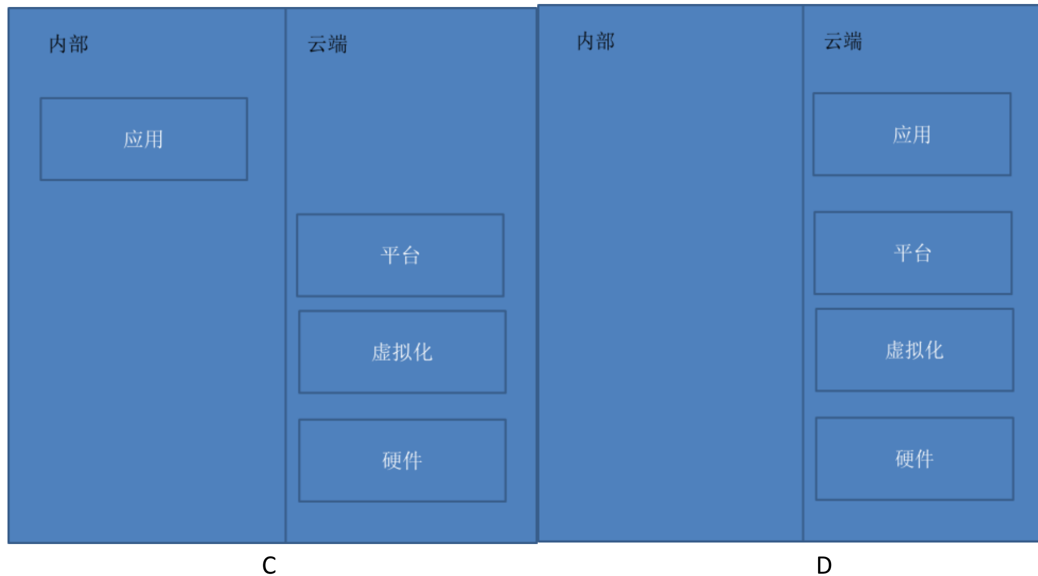


图 B-1 云的服务模型

B.4.3 发布模型

1，私有云

云基础架构被一个组织独立地操作，可能被这个组织或者第三方机构所管理，可能存在于某种条件下或者无条件存在。

2，社区云

云基础架构被几个组织所共享，并且支持一个互相分享概念（例如，任务、安全需求、策略和切合的决策）的特别的社区。可能被这些组织或者第三方机构所管理，可能存在于某种条件下或者无条件存在。与公有云相比，社区云的目的性更强，社区云的发起者往往具有共同的目的和利益的机构。比如软件测试机构为了对软件测试技术和经验有更好的研究和推广，作为研究，学习软件测试的目的，把相关的软件测试的资料，工具，论文等资源放在建立的一块云上，这样的云就叫做社区云。一般来说社区云比公有云要小。

3，公有云

云基础架构被做成一般公共或者一个大的商业群体所使用，被某个组织所拥有，并出售云服务。

4，混合云

云基础架构是由两个或者两个以上的云组成，这些云保持着唯一的实体但是通过标准或者特有的技术结合在一起。这些技术使得数据或者应用程序具有可移植性。例如，在云之间进行负载平衡的 Cloud Bursting 技术。

B.4 云服务器

云服务器与云存储是云计算中两个关键元素，这一节让我们先来谈谈云计算中的第一个关键的元素：云服务器。

B.5.1 云服务器概念

云服务器（Elastic Compute Service，简称 ECS）是一种处理能力可弹性伸缩的计算服务，其管理方式比物理服务器更简单高效。云服务器帮助我们快速构建更稳定、安全的应用，降低开发运维的难度和整体 IT 成本，使您能够更专注于核心业务的创新。

云服务器是云计算服务的重要组成部分，是面向各类互联网用户提供综合业务能力的服务平台。平台整合了传统意义上的互联网应用三大核心要素：计算、存储、网络，面向用户提供公用化的互联网基础设施服务。

云服务器平台的每个集群节点被部署在互联网的骨干数据中心，可独立提供计算、存储、在线备份、托管、带宽等互联网基础设施服务。

集群节点由以下硬件构成：

管理服务器：采取双机热备的方式，对整个节点的所有计算服务器、共享存储、网络进行管理，同时对外提供管理整个节点的 API。管理服务器上提供：管理服务（管理节点的计算服务器，对外提供管理接口)、DHCP 服务（为计算服务器的网络启动分配管理网段的 IP)、tftp 服务（为计算服务器的网络启动提供远程启动映象)、nbd 服务（为计算服务器提供网络块设备服务)。管理服务器上还会运行一个数据采集程序，他定时将各种性能数据采集下来并发送到中央的数据采集服务器上存储服务器群：存储服务器可以是 ISCSI 或内置存储容量比较大的 x86 服务器，通过 集群文件系统组成一个统一的存储池，为节点内的虚拟机提供逻辑磁盘存储、非结构数据存储以及整合备份服务。

计算服务器群：计算服务器是高配置的八核以上服务器，计算服务器无需安装操作系统，但必须具备网络引导功能，其上运行一个 Linux 微内核、云计算机软件、一个与管理服务器进行通讯的 Agent

交换机：按不同功能和节点性能要求配备多个三层交换机，分别负责管理网段、公网交换网段、内部交换网段、存储网段等

B.5.2 云服务器的特点

云计算服务器具有以下特点，即高密度（High-density)、低能耗（Energy-saving)，易管理（Reorganization)、系统优化（Optimization)。

1，高密度（High-density）

未来的云计算中心将越来越大，而土地则寸土寸金，机房空间捉襟见肘，如何在有限空间容纳更多的计算节点和资源是发展关键。

2，低能耗（Energy-saving）

云数据中心建设成本中电力设备和空调系统投资比重达到 65%，而数据中心运营成本中 75%将是能源成本。可见，能耗的降低对数据中心而言是极其重要的工作，而云计算服务器则是能耗的核心。

3，易管理（Reorganization ）

数量庞大的服务器管理起来是个很大问题，通过云平台管理系统、服务器管理接口实现轻松部署和管理则是云计算中心发展必须考虑的因素。

4，系统优化（Optimization）

在云计算中心中，不同的服务器承担着不同的应用。例如有些是虚拟化应用、有些是大数据应用，不同的应用有着不同的需求。因此针对不同应用进行优化，形成针对性的硬件支撑环境，将能充分发挥云计算中心的优势。

B.5.3 云服务器所要解决的问题

1，适应更高环境温度

在云计算数据中心中，空调系统的核心理念是注重 IT 设备的温度要求，高效解决区域化的制冷，在云时代，IT 设备在适应温度方面变得更强壮。据了解，目前欧美许多公司已经将云服务器放在北欧，加拿大等高纬度地区，以适应温度的影响。目前的通用服务器设计标准为 35℃进风温度，天地超云科技有限公司推出的高温节能服务器 ，最高可在 47℃的环境下正常运行。IT 设备的优化工作直接导致了数据中心空调温度标准的改变。ASHRAE （美国暖通空调协会） 在其数据中心标准中发布了 2008 版本和 2011 版本；在 2008 年版本中，数据中心的温度推荐标准：温度范围为 18℃~27℃；而在 2011 年的推荐标准中，对高温 IT 设备，则扩展到了 A1-A2 框体，温度范围为 0℃~35℃。所以，云服务器需要不断提高环境温度的适应能力，能耐高温、节能成了云服务器的一个发展方向。

2，更加节约空间

当前不计成本的高性能计算时代已经一去不复返了，解决尖端问题的高端系统同样也必须降低成本。为了避免由于服务器爆炸性增加而造成机房面积过快扩大以及随之而剧增的各种运行维护费用，数据中心要求大幅度缩小服务器的占地面积、提高计算密度、发展高密度计算。

B.6 云存储

这一节，让我们来看一下云计算上另外一个关键元素：云存储。

B.6.1 云存储概念

云存储是在云计算（cloud computing）概念上延伸和发展出来的一个新的概念，是指通过集群应用、网格技术或分布式文件系统等功能，将网络中大量各种不同类型的存储设备通过应用软件集合起来协同工作，共同对外提供数据存储和业务访问功能的一个系统。当云计算系统运算和处理的核心是大量数据的存储和管理时，云计算系统中就需要配置大量的存储设备，那么云计算系统就转变成为一个云存储系统，所以云存储是一个以数据存储和管理为核心的云计算系统。简单来说，云存储就是将储存资源放到云上供人存取的一种新兴方案。使用者可以在任何时间、任何地方，透过任何可连网的装置连接到云上方便地存取数据。

B.6.2 云存储模型

云存储系统的结构模型由 4 层组成。

1，存储层

存储层是云存储最基础的部分。存储设备可以是 FC 光纤通道存储设备，可以是 NAS 和 iSCSI 等 IP 存储设备，也可以是 SCSI 或 SAS 等 DAS 存储设备。云存储中的存储设备往往数量庞大且分布多不同地域，彼此之间通过广域网、互联网或者 FC 光纤通道网络连接在一起。

存储设备之上是一个统一存储设备管理系统，可以实现存储设备的逻辑虚拟化管理、多链路冗余管理，以及硬件设备的状态监控和故障维护。

2，基础管理

基础管理层是云存储最核心的部分，也是云存储中最难以实现的部分。基础管理层通过集群、分布式文件系统和网格计算等技术，实现云存储中多个存储设备之间的协同工作，使多个的存储设备可以对外提供同一种服务，并提供更大更强更好的数据访问性能。

3，应用接口

应用接口层是云存储最灵活多变的部分。不同的云存储运营单位可以根据实际业务类型，开发不同的应用服务接口，提供不同的应用服务。比如视频监控应用平台、IPTV 和视频点播应用平台、网络硬盘引用平台，远程数据备份应用平台等。

4，访问层

任何一个授权用户都可以通过标准的公用应用接口来登录云存储系统，享受云存储服务。云存储运营单位不同，云存储提供的访问类型和访问手段也不同。

B.6.3 云存储分类

云存储可分为以下三类：

1，公共云存储

像亚马逊公司的 Simple Storage Service（S3）和 Nutanix 公司提供的存储服务一样，它们可以低成本提供大量的文件存储。供应商可以保持每个客户的存储、应用都是独立的，私有的。其中以 Dropbox 为代表的个人云存储服务是公共云存储发展较为突出的代表，国内比较突出的代表的有搜狐企业网盘，百度云盘，乐视云盘，移动彩云，金山快盘，坚果云，酷盘，115 网盘，华为网盘，360 云盘，新浪微盘，腾讯微云，cStor 云存储等。

公共云存储可以划出一部分用作私有云存储。一个公司可以拥有或控制基础架构，以及应用的部署，私有云存储可以部署在企业数据中心或相同地点的设施上。私有云可以由公司自己的 IT 部门管理，也可以由服务供应商管理。

2，内部云存储

这种云存储和私有云存储比较类似，唯一的不同点是它仍然位于企业防火墙内部。至 2014 年可以提供私有云的平台有：Eucalyptus、3A Cloud、minicloud 安全办公私有云、联想网盘等。

3，混合云存储

这种云存储把公共云和私有云/内部云结合在一起。主要用于按客户要求的访问，特别是需要临时配置容量的时候。从公共云上划出一部分容量配置一种私有或内部云可以帮助公司面对迅速增长的负载波动或高峰时很有帮助。尽管如此，混合云存储带来了跨公共云和私有云分配应用的复杂性。

B.7 云计算平台

了解了以上知识以后让我们来看看目前比较流行的一些云计算平台。
云计算平台也称为云平台。云计算平台可以划分为 3 类：

- 以数据存储为主的存储型云平台；
- 以数据处理为主的计算型云平台；
- 以计算和数据存储处理兼顾的综合云计算平台。

B.7.1 开源的云平台

1，AbiCloud （Abiquo 公司）

AbiCloud 是一款用于公司的开源的云计算平台，使公司能够以快速、简单和可扩展的方式创建和管理大型、复杂的 IT 基础设施（包括虚拟服务器、网络、应用、

存储设备等）。Abiquo 公司位于美国加利福尼亚州红木市，它提供的云计算服务包括为企业创造和管理私人云服务、公共云服务和混合云服务，能让企业用户把他们的电脑和移动设备中的占据大量资源的数据转移到更大、更安全的服务器上。

2，Hadoop（Apache 基金会）

该计划是完全模仿 Google 体系架构做的一个开源项目，主要包括 Map/Reduce 和 HDFS 文件系统。

3，Eucalyptus 项目（加利福尼亚大学）

创建了一个使企业能够使用它们内部 IT 资源（包括服务器、存储系统、网络设备）的开源界面，来建立能够和 Amazon EC2 兼容的云

4，MongoDB（10gen）

MongoDB 是一个高性能、开源、无模式的文档型数据库，它在许多场景下可用于替代传统的关系型数据库或键/值存储方式。mongodb 由 C++写就，其名字来自 humongous 这个单词的中间部分，从名字可见其野心所在就是海量数据的处理。关于它的一个最简洁描述为：scalable、high-performance、open source、schema-free、document-oriented database。

5，Enomalism 弹性计算平台

它提供了一个功能类似于 EC2 的云计算框架。Enomalism 基于 Linux，同时支持 Xen 和 Kernel Virtual Machine（KVM）。与其他纯 IaaS 解决方案不同的是，Enomalism 提供了一个基于 TurboGears Web 应用程序框架和 Python 的软件栈

6，Nimbus（网格中间件 Globus）

Nimbus 面向科学计算需求，通过一组开源工具来实现基础设施即服务（IaaS）的云计算解决方案

B.7.2 商用云平台

1，微软

技术特性：整合其所用软件及数据服务；
核心技术：大型应用软件开发技术；
企业服务：Azure 平台；
开发语言：.NET。

2，Google

技术特性：储存及运算水平扩充能力；

核心技术：平行分散技术 MapReduce，BigTable，GFS；
企业服务：Google AppEngine，应用代管服务；
开发语言：Python，Java。

3，IBM

技术特性：整合其所有软件及硬件服务；
核心技术：网格技术，分布式存储，动态负载；
企业服务：虚拟资源池提供，企业云计算整合方案。

4，Oracle

技术特性：软硬件弹性虚拟平台；
核心技术：Oracle 的数据存储技术，Sun 开源技术；
企业服务：EC2 上的 Oracle 数据库，OracleVM，Sun xVM。

5，Amazon

技术特性：弹性虚拟平台；
核心技术：虚拟化技术 Xen；
企业服务：EC2、S3，SimpleDB、SQS。

6，Saleforce

技术特性：弹性可定制商务软件；
核心技术：应用平台整合技术；
企业服务：Force. com 服务；
开发语言：Java，APEX。

7，旺田云服务

技术特性：按需求可定制平台化软件；
核心技术：应用平台整合技术；
企业服务：netfarmer 服务提供不同行业信息化平台；
开发语言：Deluge（Data Enriched Language for the Universal Grid Environment）。

8，EMC

技术特性：信息存储系统及虚拟化技术；
核心技术：Vmware 的虚拟化技术，一流存储技术；
企业服务：Atoms 云存储系统，私有云解决方案。

9，阿里巴巴

技术特性：弹性可定制商务软件；
核心技术：应用平台整合技术；

企业服务：软件互联平台，云电子商务平台。

10，中国移动

技术特性：坚实的网络技术丰富的带宽资源；
核心技术：底层集群部署技术，资源池虚拟技术，网络相关技术；
企业服务：BigCloude-大云平台。

附录 C 开发编码规则

C1 MISRA C 2004 规则

MISRA（The Motor Industry Software Reliability Association 汽车工业软件可靠性协会）
MISRA 是汽车工业 C 语言编程指导，是目前公认的最优秀的嵌入式 C 语言的编码规范，在航空/航天、汽车、医疗、船舶、电信等对软件安全性要求比较高的行业得到了广泛的应用。
在 1998 年版的基础上，MISRA 组织最新发布了 MISRA-C:2004
MISRA-C：2004 包括 141 条规则，其中 121 条是强制（Required）遵守的，20 条是建议（Advisory）遵守的。MISRA 官方网站：www.misra.org.uk。

C.1.1 环境

Rule1.1（强制）：所有的代码应该遵守 ISO 9899：1990“Programming Language C”;
Rule1.2（强制）：只有当具备统一接口的目标代码的时候才可以采用多种编译器和语言；
Rule1.4（强制） 检查编译器/连接器以确保支持 31 一个有效字符，支持大小写敏感。

C.1.2 语言扩展

Rule 2.1（强制）：汇编语言应该封装起来并且隔离；
Rule 2.2（强制） ：源代码只能采用/*…*/风格的注释；
Rule2.3（强制）： 字符序列/*不能在注释中使用；
Rule 2.4（建议）：代码段不能注释掉；
注：应采用#IF 或者#ifdef 来构成一个注释，否则代码里如果有注释会改变代码的作用。

C.1.3 文档化

Rule 3.3（建议）：编译器对于整数除法运算的实施应该写入文档。

C.1.4 字符集

Rule C.1（强制）：只能使用 ISO 标准定义的字符集。

C.1.5 标识符

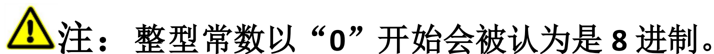
Rule6.5 （强制）：在内部范围的标识符不能和外部的标识符使用同样的名字，因为会隐藏那个标识符；
Rule 5.2（强制）： typedef 名称只能唯一，不能重复定义；
Rule 5.4（强制）： 标记名应该是唯一的标识符；
Rule 5.7（建议）：标识符不能重复使用。

C.1.6 类型

Rule 6.1（强制）：Char 类型只能用来存储使用字符；
Rule 6.2（强制）：signed 和 unsigned char 只能用来存储和使用数据值；
Rule6.3（建议）：对于基本的类型使用 Typedef 来表示大小和有无符号。

C.1.7 约束

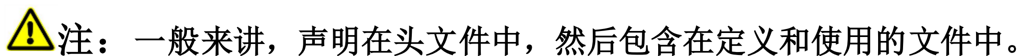
Rule 7.1（强制）：不要用八进制数。



注： 整型常数以“0”开始会被认为是 8 进制。

C.1.8 声明和定义

Rule 8.1（强制）：函数都应该有原型声明，且相对函数定义和调用可见；
Rule8.2 （强制）：无论何时一个对象和函数声明或者定义，它的类型应该明确声明；
Rule 8.5（强制）：头文件中不要定义对象或者函数；
Rule8.3（强制）：每个函数声明中的参数的类型应该和定义中的类型一致；
Rule 8.8（强制）：外部变量或者函数只能声明在一个文件中；



注： 一般来讲，声明在头文件中，然后包含在定义和使用的文件中。

Rule 8.12（强制）：数组声明为外部，应该明确声明大小或者直接初始化确定。

C.1.9 初始化

Rule 9.1（强制）：所有变量在使用之前都应该赋值；

C.1.10 数学类型转换（隐式）

Rule 10.1（强制）：整型表达式不要隐式转换为其他类型：
　a）转换到更大的整型；
　b）表达式太复杂；
　c）表达式不是常数是一个函数；
　d）表达式不是一个常数是一个返回表达式。
Rule 10.2（强制）：浮点数表达式不要隐式转换为其他类型：
　a）转换到更大的浮点数；
　b）表达式太复杂；
　c）表达式是一个函数；
　d）表达式是一个返回表达式。
<数学类型转换（明确）>
Rule 10.3（强制）：整型表达式的值只能转换到更窄小且是同样符号类型的表达式；
Rule 10.4（强制）：浮点表达式的值只能转换到更窄小的浮点表达式；
<数学类型转换>
Rule 10.6（强制）：所有的 unsigned 类型都应该有后缀“U”。

C.1.11 指针

Rule 11.1（强制）：指针不能转换为函数或者整型以外的其他类型。

C.1.12 表达式

Rule12.2（强制）：表达式的值应和标准允许的评估顺序一致；
Rule12.3（强制）：sizeof 操作符不能用在包含边界作用（side effect）的表达式；
Rule 12.4（强制）：逻辑操作符&&或者||右边不能包含边界作用（side effect）；
Rule 12.3（建议）：++和- -不能和其他表达式用在一个表达式。

C.1.13 控制语句表达式

Rule13.1（强制）：赋值语句不能用在一个产生布尔值的表达式中；
Rule13.3（强制）：浮点表达式不应该测试其是否相等或者不相等；
Rule13.4（强制）：for 控制表达式中不要包含任何浮点类型；
Rule13.6（强制）：数字变量作为 for 循环的循环计数不要在循环体内部被修改；

C.1.14 控制流

Rule 14.1（强制）：不要有执行不到的代码；
Rule 14.4（强制）：goto 语句不能使用；
Rule 14.5（强制）：continue 不能使用；
Rule 14.6（强制）：函数应在函数结束有一个出口；
Rule 14.7（强制）：witch，while，do ...while，for 语句体应是一个混合语句（括号）；
Rule 14.10（强制）：所有 if…else if 结构都应该由 else 结束。

C.1.15 Switch 语句

Rule 15.3（强制）：switch 的最后应是 default；
Rule 15.4（强制）：switch 表达式不能使用布尔表达式；
Rule 15.5（强制）：每一个 Switch 语句都应该有一个 case。

C.1.16 函数

Rule16.2（强制）：函数不能直接或者间接的调用自己；

注：safe-related 系统不能用递归，超出堆栈空间很危险。

Rule16.8（强制）：non-void 类型函数的所有出口路径都应该有一个明确的 return 语句表达式。

C.1.17 指针和数组

Rule17.1（强制）：指针的数学运算只能用在指向数组的地址上；
Rule17.3（强制）：>，>=，<，<=不能用在指针类型除非指向同一个数组；
Rule 17.5（建议）：不要用 2 级以上的指针。

C.1.18 结构和联合

Rule18.4（强制）不要用 Union。

C.1.19 预处理指令

Rule19.1（建议）：#include 语句的前面只能有其他预处理指令和注释；
Rule19.2（建议）：#include 指令中的头文件名称不能包含非标准的字符；
Rule19.5（强制）：宏不能在函数体内定义；
Rule19.8（强制）：类函数宏调用时不能没有它的参数。

C.1.20 标准库

Rule20.1（强制）：标准库中的保留标识符，宏和函数不能定义，重定义，和undefined；
Rule20.4（强制）：动态内存分配不能使用；

注：不能使用：malloc，calloc，free，realloc；

Rule20.9（强制）：输入输出库<stdio.h>不能用在产生嵌入式系统中；
Rule20.12（强制）：时间处理函数<time.h>不能使用。

C.1.21 运行时故障

Rule 21.1（强制）：通过使用一下手段确保把运行时故障最小化：
a）静态分析工具/技术；
b）动态分析工具/技术；
c）编写明确的代码避免运行时错误。

C2 C 语言编码规范

上一节介绍了 C 语言规范相对于嵌入式软件而言，要求比较严格。在这节我们介绍一个在普通软件企业中经常使用的 C 语言编码规范。

C.2.1 排版

1-1：程序块要采用缩进风格编写，缩进 TAB 键一个；。
1-2：相对独立的程序块之间、变量说明之后必须加空行；
1-3：较长的语句（>80 字符）要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读；
1-4：循环、判断等语句中若有较长的表达式或语句，则要进行适应的划分，长表达式要在低优先级操作符处划分新行，操作符放在新行之首；
1-5：若函数或过程中的参数较长，则要进行适当的划分；
1-6：不允许把多个短语句写在一行中，即一行只写一条语句；
1-7：if、while、for、default、do 等语句自占一行；
1-8：对齐只使用 TAB 键，不使用空格键；
1-9：函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case 语句下的情况处理语句也要遵从语句缩进要求；
1-10：程序块的分界符（如 C/C++语言的大括号“{”和“}”）应各独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及 if、for、do、while、switch、case 语句中的程序都要采用如上的缩进方式；

1-11：在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如->）后不应加空格；
1-12：程序结构清晰，简单易懂，单个函数的程序行数不得超过 100 行。

C.2.2 注释

2-1：一般情况下，源程序有效注释量必须在 20％以上；
2-2：说明性文件（如头文件.h 文件、.inc 文件、.def 文件、编译说明文件.cfg 等）的头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等，头文件的注释中还应有函数功能简要说明；
2-3：源文件头部应进行注释，列出：版权说明、版本号、生成日期、作者、模块目的/功能、主要函数及其功能、修改日志等；
2-4：函数头部应进行注释，列出：函数的目的/功能、输入参数、输出参数、返回值、调用关系（函数、表）等；
2-5：边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。没有用的注释要及时删除；
2-6：注释的内容要清楚、明了，含义准确，防止注释的二义性；
2-7：避免在注释中使用缩写，特别是非常用缩写；
2-8：注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开；
2-9：对于所有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义。变量、常量、宏的注释应放在其上方相邻位置或右方；
2-10：数据结构声明（包括数组、结构、类、枚举等），如果其命名不是充分自注释的，必须加以注释。对数据结构的注释应放在其上方相邻位置，不可放在下面；对结构中的每个域的注释放在此域的右方；
2-11：全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数或过程存取它以及存取时注意事项等的说明；
2-12：注释与所描述内容进行同样的缩排；
2-13：将注释与其上面的代码用空行隔开；
2-14：对变量的定义和分支语句（条件分支、循环语句等）必须编写注释；
2-15：对于 switch 语句下的 case 语句，如果因为特殊情况需要处理完一个 case 后进入下一个 case 处理，必须在该 case 语句处理完、下一个 case 语句前加上明确的注释。

C.2.3 标识符命名

3-1：标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或各位基本可以理解的缩写，避免使人产生误解；
3-2：命名中若使用特殊约定或缩写，则要有注释说明；
3-3：自己特有的命名风格，要自始至终保持一致，不可来回变化；

3-4：对于变量命名，禁止取单个字符（如 i、j、k），建议除了要有具体含义外，还能表明其变量类型、数据类型等，但 i、j、k 等作局部循环变量是允许的；
3-5：命名规范必须与所使用的系统风格保持一致，并在同一项目中统一，比如采用 UNIX 的全小写加下划线的风格或大小写混排的方式，不要使用大小写与下划线混排的方式。

C.2.4 可读性

4-1：注意运算符的优先级，并用括号明确表达式的操作顺序，避免使用默认的优先级；
4-2：避免使用不易理解的数字，用有意义的标识来替代。涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的枚举或宏来代替。

C.2.5 变量

5-1：去掉没必要的公共变量；
5-2：仔细定义并明确公共变量的含义、作用、取值范围及公共变量间的关系；
5-3：明确公共变量与操作此公共变量的函数或过程的关系，如访问、修改及创建等；
5-4：当向公共变量传递数据时，要十分小心，防止赋于不合理的值或越界等现象发生；
5-5：防止局部变量与公共变量同名；
5-6：严禁使用未经初始化的变量作为右值。

C.2.6 函数、过程

6-1：对所调用函数的错误返回码要仔细、全面地处理；
6-2：明确函数功能，精确（而不是近似）地实现函数设计；
6-3：编写可重入函数时，应注意局部变量的使用（如编写 C/C++语言的可重入函数时，应使用 auto 即缺省态局部变量或寄存器变量）；
6-4：编写可重入函数时，若使用全局变量，则应通过关中断、信号量（即 P、V 操作）等手段对其加以保护。

C.2.7 可测性

7-1：在同一项目组或产品组内，要有一套统一的为集成测试与系统联调准备的调测开关及相应打印函数，并且要有详细的说明；
7-2：在同一项目组或产品组内，调测打印出的信息串的格式要有统一的形式。信息串中至少要有所在模块名（或源文件名）及行号；
7-3：编程的同时要为单元测试选择恰当的软件测试点，并仔细构造软件测试代码、软件测试用例，同时给出明确的注释说明。软件测试代码部分应作为（模块中的）一个子模块，以方便测试代码在模块中的安装与拆卸（通过调测开关）；

7-4：在进行集成测试/系统联调之前，要构造好测试环境、测试项目及测试用例，同时仔细分析并优化测试用例，以提高测试的效率。
7-5：使用断言来发现软件问题，提高代码可测性；
7-6：用断言来检查程序正常运行时不应发生但在调测时有可能发生的非法情况；
7-7：不能用断言来检查最终产品肯定会出现且必须处理的错误情况；
7-8：对较复杂的断言加上明确的注释；
7-9：用断言确认函数的参数；
7-10：用断言保证没有定义的特性或功能不被使用；
7-11：用断言对程序开发环境（OS/Compiler/Hardware）的假设进行检查。
7-12：正式软件产品中应把断言及其它调测代码去掉（即把有关的调测开关关掉）；
7-13：在软件系统中设置与取消有关测试手段，不能对软件实现的功能等产生影响；
7-14：用调测开关来切换软件的 debug 版和正式版，而不要同时存在正式版本和 debug 版本的不同源文件，以减少维护的难度；
7-15：软件的 debug 版本和发行版本应该统一维护，不允许分家，并且要时刻注意保证两个版本在实现功能上的一致性。

C.2.8 程序效率

8-1：编程时要经常注意代码的效率；
8-2：在保证软件系统的正确性、稳定性、可读性及可测性的前提下，提高代码效率；
8-3：局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响；
8-4：通过对系统数据结构的划分与组织的改进，以及对程序算法的优化来提高空间效率；
8-5：循环体内工作量最小化。

C.2.9 质量保证

9-1：在软件设计过程中构筑软件质量；
9-2：代码质量保证优先原则；
9-3：只引用属于自己的存贮空间；
9-4：防止引用已经释放的内存空间；
9-5：过程/函数中分配的内存，在过程/函数退出之前要释放；
9-6：过程/函数中申请的（为打开文件而使用的）文件句柄，在过程/函数退出之前要关闭；
9-7：防止内存操作越界；
9-8：认真处理程序所能遇到的各种出错情况；
9-9：系统运行之初，要初始化有关变量及运行环境，防止未经初始化的变量被引用；
9-10：系统运行之初，要对加载到系统中的数据进行一致性检查；
9-11：严禁随意更改其它模块或系统的有关设置和配置；
9-12：不能随意改变与其它模块的接口；

9-13：充分了解系统的接口之后，再使用系统提供的功能；
9-14：编程时，要防止差 1 的错误；
9-15：要时刻注意易混淆的操作符。当编完程序后，应从头至尾检查一遍这些操作符，以防止拼写错误；
9-16：有可能的话，if 语句尽量加上 else 分支，对没有 else 分支的语句要小心对待；switch 语句必须有 default 分支；
9-17：禁止 GOTO 语句；
9-18：单元测试也是编程的一部份，提交联调测试的程序必须通过单元测试。

C.2.10 代码编辑、编译、审查

10-1：打开编译器的所有告警开关对程序进行编译；
10-2：在产品软件（项目组）中，要统一编译开关选项；
10-3：通过代码走读及审查方式对代码进行检查。

C.2.11 代码软件测试、维护

11-1：单元测试要求至少达到语句覆盖；
11-2：单元测试开始要跟踪每一条语句，并观察数据流及变量的变化；
11-3：清理、整理或优化后的代码要经过审查及软件测试；
11-4：代码版本升级要经过严格的软件测试；
11-5：使用工具软件对代码版本进行维护；
11-6：正式版本上软件的任何修改都应有详细的文档记录。

C.2.12 宏

12-1：用宏定义表达式时，要使用完备的括号；
12-2：将宏所定义的多条表达式放在大括号中；
12-3：使用宏时，不允许参数发生变化。

C3 Java 语言编码规范（Java Code Conventions）

C.3.1 介绍（Introduction）

1，为什么要有编码规范（Why Have Code Conventions）

编码规范对于程序员而言尤为重要，有以下几个原因：

- 一个软件的生命周期中，80%的花费在于维护
- 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发工程师来维护
- 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码

- 如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品
  为了执行规范，每个软件开发工程师必须一致遵守编码规范。每个人。

2，版权声明（Acknowledgments）

本文档反映的是 Sun MicroSystems 公司，Java 语言规范中的编码标准部分。主要贡献者包括：Peter King，Patrick Naughton，Mike DeMoney，Jonni Kanerva，Kathy Walrath 以及 Scott Hommel。
本文档现由 Scott Hommel 维护，有关评论意见请发至 shommel@eng.sun.com

C.3.2 文件名（File Names）

这部分列出了常用的文件名及其后缀。

1，文件后缀（File Suffixes）

Java 程序使用下列文件后缀：
文件类别文件后缀：
Java 源文件 .java；
Java 字节码文件 .class。

2，常用文件名（Common File Names）

常用的文件名包括：
文件名用途；
GNUmakefile makefiles 的首选文件名。我们采用 gnumake 来创建（build）软件。
README 概述特定目录下所含内容的文件的首选文件名。

C.3.3 文件组织（File Organization）

一个文件由被空行分割而成的段落以及标识每个段落的可选注释共同组成。超过 2000 行的程序难以阅读，应该尽量避免。“Java 源文件范例”提供了一个布局合理的 Java 程序范例。

1，Java 源文件（Java Source Files）

每个 Java 源文件都包含一个单一的公共类或接口。若私有类和接口与一个公共类相关联，可以将它们和公共类放入同一个源文件。公共类必须是这个文件中的第一个类或接口。
Java 源文件还遵循以下规则：
- 开头注释（参见“开头注”）；
- 包和引入语句（参见“包和引入语句”）；
- 类和接口声明（参见“类和接口声明”）。

开头注释（Beginning Comments）

所有的源文件都应该在开头有一个 C 语言风格的注释，其中列出类名、版本信息、日期和版权声明：

```
/*
     * Classname
     *
     * Version information
     *
     * Date
     *
     * Copyright notice
     */
```

包和引入语句（Package and Import Statements）

在多数 Java 源文件中，第一个非注释行是包语句。在它之后可以跟引入语句。例如：

```
package java.awt;
import java.awt.peer.CanvasPeer;
```

类和接口声明（Class and Interface Declarations）

下表描述了类和接口声明的各个部分以及它们出现的先后顺序。参见“Java 源文件范例”中一个包含注释的例子。

类/接口声明的各部分注解：

1）类/接口文档注释（/**……*/） 该注释中所需包含的信息，参见“文档注释”；

2）类或接口的声明；

3）类/接口实现的注释（/*……*/）如果有必要的话该注释应包含任何有关整个类或接口的信息，而这些信息又不适合作为类/接口文档注释；

4）类的（静态）变量首先是类的公共变量，随后是保护变量，再后是包一级别的变量（没有访问修饰符，access modifier），最后是私有变量；

5）实例变量首先是公共级别的，随后是保护级别的，再后是包一级别的（没有访问修饰符），最后是私有级别的；

6）构造器

7）方法应该按功能，而非作用域或访问权限分组。例如，一个私有的类方法可以置于两个公有的实例方法之间。其目的是为了便于阅读和理解代码。

C.3.4 缩进排版（Indentation）

4 个空格常被作为缩进排版的一个单位。缩进的确切解释并未详细指定（空格 vs. 制表符）。一个制表符等于 8 个空格（而非 4 个）。

1，行长度（Line Length）

尽量避免一行的长度超过 80 个字符，因为很多终端和工具不能很好处理。

注：用于文档中的例子应该使用更短的行长，长度一般不超过 70 个字符。

2， 换行（Wrapping Lines）

当一个表达式无法容纳在一行内时，可以依据如下一般规则断它：

- 在一个逗号后面断开；
- 在一个操作符前面断开；
- 宁可选择较高级别（higher-level）的断开，而非较低级别（lower-level）的断开；
- 新的一行应该与上一行同一级别表达式的开头处对齐；
- 如果以上规则导致你的代码混乱或者使你的代码都堆挤在右边，那就以制表符代之以缩进 8 个空格。

以下是断开方法调用的一些例子：

```
someMethod(longExpression1, longExpression2, longExpression3,
                        longExpression4, longExpression5);

var = someMethod1(longExpression1,
someMethod2(longExpression2,
                                    longExpression3));
```

以下是两个断开算术表达式的例子。前者更好，因为断开处位于括号表达式的外边，这是个较高级别的断开。

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                        + 4 * longname6; //PREFFER

  longName1 = longName2 * (longName3 + longName4
                                    - longName5) + 4 * longname6;
//AVOID
```

以下是两个缩进方法声明的例子。前者是常规情形。后者若使用常规的缩进方式将会使第二行和第三行移得很靠右，所以以制表符代之以缩进 8 个空格。

```
//CONVENTIONAL INDENTATION
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
                        Object andStillAnother) {
     ...
  }

  //INDENT 8 SPACES TO AVOID VERY DEEP INDENTS
private static synchronized horkingLongMethodName(int anArg,
            Object anotherArg, String yetAnotherArg,
            Object andStillAnother) {
     ...
  }
```

if 语句的换行通常使用 8 个空格的规则，因为常规缩进（4 个空格）会使语句体看起来比较费劲。比如：

```
//DON'T USE THIS INDENTATION
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) { //BAD WRAPS
doSomethingAboutIt();                    //MAKE THIS LINE EASY TO MISS
  }
  //USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
             || (condition3 && condition4)
             ||!(condition5 && condition6)) {
doSomethingAboutIt();
  }
  //OR USE THIS
if ((condition1 && condition2) || (condition3 && condition4)
             ||!(condition5 && condition6)) {
doSomethingAboutIt();
  }
```

这里有三种可行的方法用于处理三元运算表达式：

```
alpha = (aLongBooleanExpression) ?beta:gamma;
alpha = (aLongBooleanExpression) ?beta
:gamma;
alpha = (aLongBooleanExpression)
             ?beta
:gamma;
```

C.3.5 注释（Comments）

Java 程序有两类注释：实现注释（implementation comments）和文档注释（document comments）。实现注释是那些在 C++中见过的，使用/*…*/和//界定的注释。文档注释（被称为“doc comments”）是 Java 独有的，并由/**…*/界定。文档注释可以通过 javadoc 工具转换成 HTML 文件。

实现注释用以注释代码或者实现细节。文档注释从实现自由（implementation-free）的角度描述代码的规范。它可以被那些手头没有源码的开发工程师读懂。

注释应被用来给出代码的总括，并提供代码自身没有提供的附加信息。注释应该仅包含与阅读和理解程序有关的信息。例如，相应的包如何被建立或位于哪个目录下之类的信息不应包括在注释中。

在注释里，对设计决策中重要的或者不是显而易见的地方进行说明是可以的，但应避免提供代码中已清晰表达出来的重复信息。多余的的注释很容易过时。通常应避免那些代码更新就可能过时的注释。

注意：频繁的注释有时反映出代码的低质量。当你觉得被迫要加注释的时候，考虑一下重写代码使其更清晰。

注释不应写在用星号或其他字符画出来的大框里。注释不应包括诸如制表符和回退符之类的特殊字符。

1，实现注释的格式（Implementation Comment Formats）

程序可以有 4 种实现注释的风格：块（block）、单行（single-line）、尾端（trailing）和行末（end-of-line）。

块注释（Block Comments）

块注释通常用于提供对文件、方法、数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式。

块注释之首应该有一个空行，用于把块注释和代码分割开来，比如：

```
/*
 * Here is a block comment.
 */
```

块注释可以以/*-开头，这样 indent（1）就可以将之识别为一个代码块的开始，而不会重排它。

```
/*-
   * Here is a block comment with some very special
   * formatting that I want indent(1) to ignore.
   *
   *       one
   *             two
   *                   three
   */
```

注意：如果你不使用 indent（1），就不必在代码中使用/*-，或为他人可能对你的代码运行 indent（1）作让步。参见“文档注释”

单行注释（Single-Line Comments）

短注释可以显示在一行内，并与其后的代码具有一样的缩进层级。如果一个注释不能在一行内写完，就该采用块注释（参见“块注释”）。单行注释之前应该有一个空行。以下是一个 Java 代码中单行注释的例子：

```
if (condition) {
      /* Handle the condition. */
      ...
   }
```

尾端注释（Trailing Comments）

极短的注释可以与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。若有多个短注释出现于大段代码中，它们应该具有相同的缩进。

以下是一个 Java 代码中尾端注释的例子：

```
if (a == 2) {
return TRUE;                    /* special case */
```

```
    } else {
return isPrime(a);                    /* works only for odd a */
    }
```

5.1.4 行末注释（End-Of-Line Comments）

注释界定符“/”，可以注释掉整行或者一行中的一部分。它一般不用于连续多行的注释文本；然而，它可以用来注释掉连续多行的代码段。以下是所有三种风格的例子：

```
if (foo > 1) {
            // Do a double-flip.
            ...
    }
else {
return false;                    // Explain why here.
    }
    //if (bar > 1) {
    //
    //        // Do a triple-flip.
    //        ...
    //}
    //else {
    //        return false;
    //}
```

2，文档注释（Documentation Comments）

注意：此处描述的注释格式之范例，参见“Java 源文件范例”。

若想了解更多，参见“How to Write Doc Comments for Javadoc”，其中包含了有关文档注释标记的信息（@return, @param, @see）：

http://www.oracle.com/technetwork/java/index.html

若想了解更多有关文档注释和 javadoc 的详细资料，参见 javadoc 的主页：

http://www.oracle.com/technetwork/java/index.html

文档注释描述 Java 的类、接口、构造器，方法，以及字段（field）。每个文档注释都会被置于注释定界符/**…*/之中，一个注释对应一个类、接口或成员。该注释应位于声明之前：

```
   /**
      * The Example class provides ...
      */
public class Example { ...
```

注意顶层（top-level）的类和接口是不缩进的，而其成员是缩进的。描述类和接口的文档注释的第一行（/**）不需缩进；随后的文档注释每行都缩进 1 格（使星号纵向对齐）。成员，包括构造函数在内，其文档注释的第一行缩进 4 格，随后每行都缩进 5 格。

若你想给出有关类、接口、变量或方法的信息，而这些信息又不适合写在文档中，则可使用实现块注释（见 5.1.1）或紧跟在声明后面的单行注释（见 5.1.2）。例

如，有关一个类实现的细节，应放入紧跟在类声明后面的实现块注释中，而不是放在文档注释中。

文档注释不能放在一个方法或构造器的定义块中，因为 Java 会将位于文档注释之后的第一个声明与其相关联。

C.3.6 声明（Declarations）

1，每行声明变量的数量（Number Per Line）

推荐一行一个声明，因为这样以利于写注释。亦即，

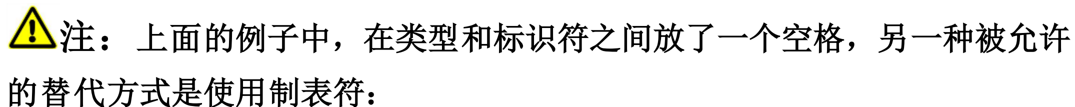
```
int level;    // indentation level
int size;     // size of table
```

要优于，

```
int level, size;
```

不要将不同类型变量的声明放在同一行，例如：

```
int foo,    fooarray[];     //WRONG!
```


注：上面的例子中，在类型和标识符之间放了一个空格，另一种被允许的替代方式是使用制表符：

```
int level; // indentation level
int size; // size of table
Object currentEntry;    // currently selected table entry
```

2，初始化（Initialization）

尽量在声明局部变量的同时初始化。唯一不这么做的理由是变量的初始值依赖于某些先前发生的计算。

3，布局（Placement）

只在代码块的开始处声明变量。（一个块是指任何被包含在大括号“{”和“}”中间的代码）。不要在首次用到该变量时才声明之。这会把注意力不集中的程序员搞糊涂，同时会妨碍代码在该作用域内的可移植性。

```
void myMethod() {
int int1 = 0;                 // beginning of method block

if (condition) {
int int2 = 0;          // beginning of “if” block
                ...
            }
    }
```

该规则的一个例外是 for 循环的索引变量。

```
for (int i = 0; i < maxLoops; i++) { ... }
```

避免声明的局部变量覆盖上一级声明的变量。例如，不要在内部代码块中声明相同的变量名：

```
int count;
   ...
myMethod() {
if (condition) {
int count = 0;          // AVOID!
                ...
         }
         ...
   }
```

4，类和接口的声明（Class and Interface Declarations）

当编写类和接口是，应该遵守以下格式规则：

- 在方法名与其参数列表之前的左括号“(”间不要有空格
- 左大括号“{”位于声明语句同行的末尾
- 右大括号“}”另起一行，与相应的声明语句对齐，除非是一个空语句，“}”应紧跟在“{”之后；

```
class Sample extends Object {
int ivar1;
int ivar2;
Sample(int i, int j) {
                ivar1 = i;
                ivar2 = j;
         }
int emptyMethod() {}
         ...
   }
```

- 方法与方法之间以空行分隔。

C.3.7 语句（Statements）

1，简单语句（Simple Statements）

每行至多包含一条语句，例如：

```
argv++;           // Correct
argc--;           // Correct
argv++; argc--;           // AVOID!
```

2，复合语句（Compound Statements）

- 复合语句是包含在大括号中的语句序列，形如“{ 语句 }”;
- 被括其中的语句应该较之复合语句缩进一个层次；

- 左大括号“{”应位于复合语句起始行的行尾；右大括号“}”应另起一行并与复合语句首行对齐；
- 大括号可以被用于所有语句，包括单个语句，只要这些语句是诸如 if-else 或 for 控制结构的一部分。这样便于添加语句而无需担心由于忘了加括号而引入 Bug。

3，返回语句（return Statements）

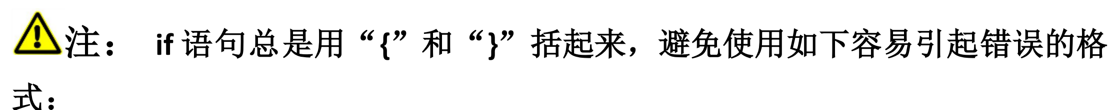
一个带返回值的 return 语句不使用小括号“()”，除非它们以某种方式使返回值更为显而易见。例如：

```
return;
return myDisk.size();
return (size?size:defaultSize);
```

4，if，if-else，if else-if else 语句（if, if-else, if else-if else Statements）

if-else 语句应该具有如下格式：

```
if (condition) {
statements;
   }
if (condition) {
statements;
   } else {
statements;
   }
if (condition) {
statements;
   } else if (condition) {
statements;
   } else{
statements;
   }
```


注： if 语句总是用“{”和“}”括起来，避免使用如下容易引起错误的格式：

```
if (condition) //AVOID! THIS OMITS THE BRACES {}!
statement;
```

5，for 语句（for Statements）

一个 for 语句应该具有如下格式：

```
for (initialization; condition; update) {
statements;
   }
```

一个空的 for 语句（所有工作都在初始化，条件判断，更新子句中完成）应该具有如下格式：

```
for (initialization; condition; update);
```

当在 for 语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在 for 循环之前（为初始化子句）或 for 循环末尾（为更新子句）使用单独的语句。

6，while 语句（while Statements）

一个 while 语句应该具有如下格式

```
while (condition) {
statements;
   }
```

一个空的 while 语句应该具有如下格式：

```
while (condition);
```

7，do-while 语句（do-while Statements）

一个 do-while 语句应该具有如下格式：

```
do {
statements;
   } while (condition);
```

7.8 switch 语句（switch Statements）

- 一个 switch 语句应该具有如下格式：

```
switch (condition) {
   case ABC：
statements;
         /* falls through */
   case DEF：
statements;
break;
   case XYZ：
statements;
break;
   default：
statements;
break;
   }
```

每当一个 case 顺着往下执行时（因为没有 break 语句），通常应在 break 语句的位置添加注释。上面的示例代码中就包含注释/* falls through */。

8， try-catch 语句（try-catch Statements）

一个 try-catch 语句应该具有如下格式：

```
try {
statements;
   } catch (ExceptionClass e) {
statements;
   }
```

一个 try-catch 语句后面也可能跟着一个 finally 语句，不论 try 代码块是否顺利执行完，它都会被执行。

```
try {
statements;
   } catch (ExceptionClass e) {
statements;
   } finally {
statements;
   }
```

C.3.8 空白（White Space）

1， 空行（Blank Lines）

空行将逻辑相关的代码段分隔开，以提高可读性。

下列情况应该总是使用两个空行：

- 一个源文件的两个片段（section）之间；
- 类声明和接口声明之间。

下列情况应该总是使用一个空行：

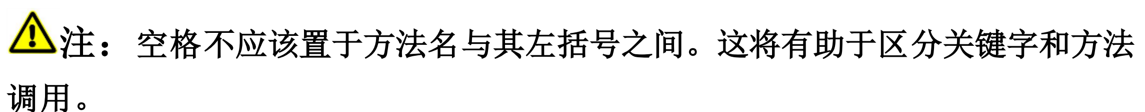
- 两个方法之间；
- 方法内的局部变量和方法的第一条语句之间；
- 块注释（参见“C.3.5.1-1”）或单行注释（参见“C.3.5.1-2”）之前；
- 一个方法内的两个逻辑段之间，用以提高可读性。

2，空格（Blank Spaces）

下列情况应该使用空格：

- 一个紧跟着括号的关键字应该被空格分开，例如：

```
while (true) {
        ...
  }
```



注： 空格不应该置于方法名与其左括号之间。这将有助于区分关键字和方法调用。

- 空白应该位于参数列表中逗号的后面；
- 所有的二元运算符，除了“.”，应该使用空格将之与操作数分开。一元操作符和操作数之间不因该加空格，比如：负号（“-”）、自增（“++”）和自减（“--”）。例如：

```
a += c + d;
a = (a + b) / (c * d);
while (d++ = s++) {
n++;
        }
printSize(“size is" + foo + “\n”);
```

- for 语句中的表达式应该被空格分开，例如：

```
for (expr1; expr2; expr3)
```

- 强制转型后应该跟一个空格，例如：

```
myMethod((byte) aNum, (Object) x);
myMethod((int) (cp + 5), ((int) (i + 3)) + 1);
```

C.3.9 命名规范（Naming Conventions）

命名规范使程序更易读，从而更易于理解。它们也可以提供一些有关标识符功能的信息，以助于理解代码，例如，不论它是一个常量，包，还是类。

标识符类型命名规则例子：

包（Packages） 一个唯一包名的前缀总是全部小写的 ASCII 字母并且是一个顶级域名，通常是 com，edu，gov，mil，net，org，或 1981 年 ISO 3166 标准所指定的标识国家的英文双字符代码。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门（department），项目（project），机器（machine），或注册名（login names）。

```
com.sun.eng
com.apple.quicktime.v2
edu.cmu.cs.bovik.cheese
```

类（Classes） 命名规则：类名是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免缩写词（除非该缩写词被更广泛使用，像 URL，HTML)。

```
class Raster;
class ImageSprite;
```

接口（Interfaces） 命名规则：大小写规则与类名相似 。

```
interface RasterDelegate;
interface Storing;
```

方法（Methods） 方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。

```
run();
runFast();
getBackground();
```

变量（Variables） 除了变量名外，所有实例，包括类，类常量，均采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。

变量名应简短且富于描述。变量名的选用应该易于记忆，即能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。临时变量通常被取名为 i，j，k，m 和 n，它们一般用于整型；c，d，e，它们一般用于字符型。

```
char c;
int i;
float myWidth;
```

实例变量（Instance Variables） 大小写规则和变量名相似，除了前面需要一个下划线。

```
int _employeeId;
String _name;
Customer _customer;
```

常量（Constants） 类常量和 ANSI 常量的声明，应该全部大写，单词间用下划线隔开。（尽量避免 ANSI 常量，容易引起错误）。

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final int GET_THE_CPU = 1;
```

C.3.10 编程惯例（Programming Practices）

1，提供对实例以及类变量的访问控制（Providing Access to Instance and Class Variables）

若没有足够理由，不要把实例或类变量声明为公有。通常，实例变量无需显式的设置（set）和获取（gotten），通常这作为方法调用的边缘效应 （side effect）而产生。

一个具有公有实例变量的恰当例子，是类仅作为数据结构，没有行为。亦即，若你要使用一个结构（struct）而非一个类（如果 java 支持结构的话），那么把类的实例变量声明为公有是合适的。

2，引用类变量和类方法（Referring to Class Variables and Methods）

避免用一个对象访问一个类的静态变量和方法。应该用类名替代。例如：

```
classMethod();                  //OK
AClass.classMethod();         //OK
anObject.classMethod();       //AVOID!
```

3，常量（Constants）

位于 for 循环中作为计数器值的数字常量，除了-1,0 和 1 之外，不应被直接写入代码。

4，变量赋值（Variable Assignments）

避免在一个语句中给多个变量赋相同的值。它很难读懂。例如：

```
fooBar.fChar = barFoo.lchar = 'c'; // AVOID!
```

不要将赋值运算符用在容易与相等关系运算符混淆的地方。例如：

```
if (c++ = d++) {              // AVOID! (Java disallows)
        ...
  }
应该写成
if ((c++ = d++) != 0) {
      ...
  }
```

不要使用内嵌（embedded）赋值运算符试图提高运行时的效率，这是编译器的工作。例如：

```
d = (a = b + c) + r;              // AVOID!
```

应该写成

```
a = b + c;
d = a + r;
```

5，其它惯例（Miscellaneous Practices）

圆括号（Parentheses）

一般而言，在含有多种运算符的表达式中使用圆括号来避免运算符优先级问题，是个好方法。即使运算符的优先级对你而言可能很清楚，但对其他人未必如此。你不能假设别的程序员和你一样清楚运算符的优先级。

```
if (a == b && c == d)         // AVOID!
if ((a == b) && (c == d))    // RIGHT
```

返回值（Returning Values）

设法让你的程序结构符合目的。例如：

```
if (booleanExpression) {
return true;
  } else {
return false;
  }
```

应该代之以如下方法：

```
return booleanExpression;
```

类似地：

```
if (condition) {
return x;
  }
return y;
```

应该写做：

```
return (condition ?x :y);
```

条件运算符“?”前的表达式（Expressions before “?” in the Conditional Operator）

如果一个包含二元运算符的表达式出现在三元运算符“?”之前，那么应该给表达式添上一对圆括号。例如：

```
(x >= 0) ? x: -x;
```

特殊注释（Special Comments）

在注释中使用 XXX 来标识某些未实现（bogus）的但可以工作（works）的内容。用 FIXME 来标识某些假的和错误的内容。

C.3.11 代码范例（Code Examples）

1，Java 源文件范例（Java Source File Example）

下面的例子，展示了如何合理布局一个包含单一公共类的 Java 源程序。接口的布局与其相似。更多信息参见“类和接口声明”以及“文挡注释”。

```
/*
  * @(#)Blah.java              1.82 99/03/18
  *
  * Copyright (c) 1994-1999 Sun Microsystems, Inc.
  * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
  * All rights reserved.
  *
  * This software is the confidential and proprietary information of Sun
  * Microsystems, Inc. (“Confidential Information”).    You shall not
  * disclose such Confidential Information and shall use it only in
  * accordance with the terms of the license agreement you entered into
  * with Sun.
  */
package java.blah;

import java.blah.blahdy.BlahBlah;

/**
  * Class description goes here.
  *
  * @version         1.82 18 Mar 1999
  * @author          Firstname Lastname
  */
public class Blah extends SomeClass {
        /* A class implementation comment can go here. */
        /** classVar1 documentation comment */
public static int classVar1;
```

```
    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
private static Object classVar2;
    /** instanceVar1 documentation comment */
public Object instanceVar1;
    /** instanceVar2 documentation comment */
protected int instanceVar2;
    /** instanceVar3 documentation comment */
private Object[] instanceVar3;
    /**
     * ...constructor Blah documentation comment...
     */
public Blah() {
        // ...implementation goes here...
    }
    /**
     * ...method doSomething documentation comment...
     */
public void doSomething() {
        // ...implementation goes here...
    }
    /**
     * ...method doSomethingElse documentation comment...
     * @param someParam description
     */
public void doSomethingElse(Object someParam) {
        // ...implementation goes here...
    }
}
```