

# CDN技术详解

Dissecting Content Delivery Network

雷葆华 孙颖 王峰 陈晓益 等著



CDN已经从为网站提供简单加速的增值服务逐步演变成互联网业务发展的必需品，从质量、效率、安全等各个方面为网站提供全面的保障。本书不仅从技术原理方面对CDN进行了详细的介绍，还从商业服务角度进行了剖析，是一本让网站运维和管理者都能从中获益的好书。

宗劫

蓝汛通讯技术有限责任公司 副总裁

CDN行业在中国曾是一个很专业的小圈子，圈里的人兴致盎然、交流充分，圈外的人却知之甚少。对整个行业来说，本书第一次如此细致地、完整地介绍了CDN的相关技术和市场状况，是一本很值得看的专业书籍。

刘洪涛

网宿科技股份有限公司 副总裁

互联网经过多年的发展，开始逐步向各个领域渗透和融合，渐渐成为人们生活不可缺少的重要组成部分。海量用户涌入互联网，导致海量的网络购物、游戏、视频等的需求，极大地推动了CDN技术的发展。CDN的研究，也成为科学技术领域的核心课题。本书不仅详尽介绍了CDN的历史、关键功能、相关技术，而且细致地分析了CDN的商业模式以及前沿技术趋势，对CDN的未来发展提出了一些前瞻性的观点和建议。本书非常适合CDN领域相关的同行阅读，也适合对CDN感兴趣的读者进行了解。

谢大维

中兴通讯股份有限公司 执行副总裁

据预测，2013年，互联网承载的90%的流量都是实时视频，而这些视频流量都会被各种形式的CDN在网络中缓存，以此达到优化网络流量、提高用户体验的目的。因此，全球运营商都在积极开展CDN的建设和运营，驱动CDN融入到固网和移动网络的基础设施层面，构筑统一管理、智能调度、多协议加速、体验运营的完整CDN战略。衷心希望本书能加速CDN知识的普及和应用。

李三琦

华为技术有限公司 IT产品线CTO



上架建议：计算机网络

ISBN 978-7-121-16528-3



9 787121 165283 >

定价：69.00元



策划编辑：刘皎

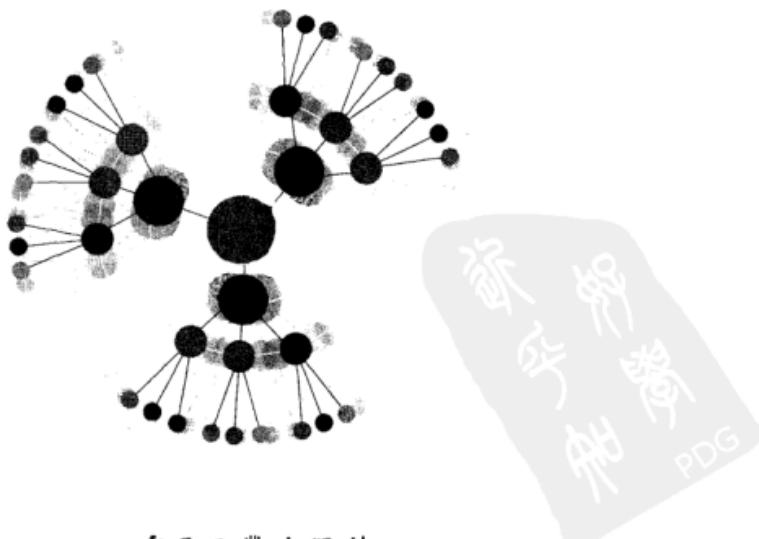
责任编辑：刘舫

封面设计：李玲

转型时代丛书  
中国电信北京研究院专家奉献

# CDN技术详解

雷葆华 孙颖 王峰 陈晓益 蔡永顺 王志军 著



电子工业出版社  
Publishing House of Electronics Industry  
北京•BEIJING

## 内 容 简 介

本书内容包括CDN技术的发展历程、关键技术、商业化服务现状，以及对未来的发展展望，对构成CDN系统的关键功能模块GSLB、SLB、Cache进行了重点讲解，除技术原理之外，还对实现这些功能模块所涉及的一些协议和开发工具进行了讲解，希望能帮助读者了解CDN这项技术，并对CDN系统的设计和开发有一些初步的体会。

本书适合从事互联网开发和运营工作的专业人士、电信运营服务从业人员，以及相关专业的高校学生。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

## 图书在版编目（CIP）数据

CDN技术详解 / 雷葆华等著. —北京：电子工业出版社，2012.6  
(转型时代丛书)

ISBN 978-7-121-16528-3

I. ①C… II. ①雷… III. ①计算机网络—网络结构 IV. ①TP393.02

中国版本图书馆CIP数据核字（2012）第051754号

策划编辑：刘 皎

责任编辑：刘 航

印 刷：三河市双峰印刷装订有限公司

装 订：

出版发行：电子工业出版社

北京市海淀区万寿路173信箱 邮编：100036

开 本：720×1000 1/16 印张：25.25 字数：404千字

印 次：2012年6月第1次印刷

印 数：4100册 定价：69.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010) 88258888。

# C转型时代丛书

## ● 指导委员会

主任委员：吴基传

副主任委员：杨杰

委员：陈俊亮 李未 韦乐平

邬贺铨 张继平（按拼音顺序排序）

## ● 编委会

主任：李志刚

副主任：侯春雨 赵慧玲

委员：毕奇 朱健 野永东 谢朝阳

陈自清 杨峰义 王晓平 张成良

# 目录

|                          |    |
|--------------------------|----|
| <b>第1章 引言</b>            | 1  |
| 1.1 CDN 的基本概念和产生背景       | 2  |
| 1.2 CDN 的基本工作过程          | 5  |
| 1.3 CDN 的发展历史            | 8  |
| 1.4 CDN 对互联网产业的价值和作用     | 13 |
| <br>                     |    |
| <b>第2章 CDN 技术概述</b>      | 16 |
| 2.1 CDN 的系统架构            | 17 |
| 2.1.1 功能架构               | 17 |
| 2.1.2 部署架构               | 21 |
| 2.2 CDN 系统分类             | 23 |
| 2.2.1 基于不同内容承载类型的分类      | 24 |
| 2.2.2 基于内容生成机制的分类和分层加速服务 | 27 |
| 2.3 小结                   | 29 |
| <br>                     |    |
| <b>第3章 内容缓存工作原理及实现</b>   | 32 |
| 3.1 内容缓存技术的发展背景          | 33 |
| 3.1.1 网站的问题和需求           | 33 |

|       |  |           |
|-------|--|-----------|
| 3.1.2 | CDN 出现前的网站服务技术 .....                           | 35        |
| 3.2   | Cache 设备的工作方式和设计要求.....                        | 38        |
| 3.2.1 | 正向代理 .....                                     | 39        |
| 3.2.2 | 反向代理 .....                                     | 41        |
| 3.2.3 | 透明代理 .....                                     | 42        |
| 3.2.4 | Web Cache 产品实现关键要素分析 .....                     | 44        |
| 3.3   | Web Cache 的实现基础——基于 HTTP 协议的<br>Web 缓存技术 ..... | 45        |
| 3.3.1 | Web 与 HTTP.....                                | 45        |
| 3.3.2 | HTTP 协议工作原理 .....                              | 48        |
| 3.3.3 | HTTP 中的 Cookie 和 Session.....                  | 68        |
| 3.3.4 | HTTPS 安全协议 .....                               | 74        |
| 3.3.5 | HTTP 协议中的缓存技术.....                             | 76        |
| 3.4   | Web Cache 技术实现关键点分析 .....                      | 82        |
| 3.4.1 | Web Cache 关键性能指标说明.....                        | 82        |
| 3.4.2 | 内容存储机制 .....                                   | 85        |
| 3.4.3 | 内容更新机制 .....                                   | 86        |
| 3.4.4 | Web Cache 协议优化 .....                           | 90        |
| 3.4.5 | Web Cache 安全实现机制 .....                         | 92        |
| 3.5   | 开源 Web 缓存代理软件——Squid .....                     | 94        |
|       | <b>第 4 章 集群服务与负载均衡技术 .....</b>                 | <b>97</b> |
| 4.1   | 服务器集群技术 .....                                  | 98        |
| 4.1.1 | 集群的基本概念 .....                                  | 98        |
| 4.1.2 | 集群的分类 .....                                    | 99        |
| 4.1.3 | 集群的系统结构 .....                                  | 101       |
| 4.1.4 | CDN 负载均衡集群 .....                               | 102       |
| 4.2   | Cache 集群协同交互方法 .....                           | 103       |
| 4.2.1 | ICP .....                                      | 104       |

|                                  |                             |            |
|----------------------------------|-----------------------------|------------|
| 4.2.2                            | HTCP .....                  | 105        |
| 4.2.3                            | Cache Digest .....          | 106        |
| 4.2.4                            | Cache Pre-filling.....      | 106        |
| 4.2.5                            | CARP .....                  | 107        |
| 4.3                              | 负载均衡技术的实现 .....             | 108        |
| 4.3.1                            | 负载均衡关键技术 .....              | 110        |
| 4.3.2                            | 负载均衡部署方式 .....              | 115        |
| 4.3.3                            | 服务器负载均衡 .....               | 118        |
| 4.3.4                            | 链路负载均衡 .....                | 125        |
| 4.4                              | 开源负载均衡软件 .....              | 130        |
| 4.4.1                            | LVS .....                   | 130        |
| 4.4.2                            | Nginx.....                  | 132        |
| <b>第 5 章 全局负载均衡工作原理及实现 .....</b> |                             | <b>134</b> |
| 5.1                              | 全局负载均衡在 CDN 系统中的作用 .....    | 135        |
| 5.2                              | 基于 DNS 解析的 GSLB 实现机制.....   | 136        |
| 5.2.1                            | DNS 的产生背景 .....             | 136        |
| 5.2.2                            | DNS 基本工作原理.....             | 137        |
| 5.2.3                            | 基于 DNS 解析的 GSLB 工作方式 .....  | 147        |
| 5.2.4                            | 负载均衡的策略判断条件信息 .....         | 150        |
| 5.2.5                            | 开源 DNS 服务软件——BIND .....     | 153        |
| 5.3                              | 基于 DNS 的 GSLB 应用部署方法 .....  | 155        |
| 5.3.1                            | GSLB 应用部署时的一些基本概念 .....     | 155        |
| 5.3.2                            | 负载均衡策略 .....                | 160        |
| 5.3.3                            | GSLB 部署中的关键问题.....          | 171        |
| 5.4                              | 基于应用层协议重定向的 GSLB.....       | 177        |
| 5.4.1                            | HTTP 重定向基本原理 .....          | 177        |
| 5.4.2                            | 基于 HTTP 重定向的 GSLB 工作流程..... | 180        |
| 5.5                              | 基于 IP 路由的 GSLB .....        | 181        |
| 5.6                              | 小结 .....                    | 184        |

|                                       |     |
|---------------------------------------|-----|
| <b>第 6 章 流媒体 CDN 系统的组成和关键技术 .....</b> | 189 |
| 6.1 流媒体系统工作原理概述 .....                 | 192 |
| 6.2 流媒体传送协议体系 .....                   | 195 |
| 6.2.1 RTP 和 RTCP .....                | 197 |
| 6.2.2 RTSP .....                      | 201 |
| 6.2.3 RTMP .....                      | 207 |
| 6.2.4 HTTP Streaming .....            | 216 |
| 6.2.5 MPEG-2 TS .....                 | 226 |
| 6.3 流媒体业务对 CDN 提出的要求和挑战 .....         | 228 |
| 6.3.1 流媒体加速与 Web 加速之间的业务差异 .....      | 228 |
| 6.3.2 流媒体 CDN 系统架构描述 .....            | 230 |
| 6.3.3 小结 .....                        | 232 |
| 6.4 流媒体 CDN 系统的关键技术实现 .....           | 233 |
| 6.4.1 Cache 的设计实现 .....               | 233 |
| 6.4.2 负载均衡系统设计实现 .....                | 238 |
| 6.4.3 内容分发机制设计实现 .....                | 240 |
| 6.4.4 组网模式 .....                      | 242 |
| 6.4.5 内容文件预处理技术 .....                 | 243 |
| 6.4.6 防盗链机制和实现 .....                  | 246 |
| <b>第 7 章 动态内容加速服务的实现 .....</b>        | 250 |
| 7.1 动态内容加速技术 .....                    | 251 |
| 7.1.1 业务逻辑层加速技术：边缘计算 .....            | 255 |
| 7.1.2 数据访问层加速技术：数据库复制 .....           | 257 |
| 7.1.3 用户数据层加速技术：用户数据复制 .....          | 261 |
| 7.2 应用加速技术 .....                      | 263 |
| 7.2.1 应用加速技术概述 .....                  | 263 |
| 7.2.2 广域网加速技术 .....                   | 264 |
| 7.2.3 SSL 加速技术介绍 .....                | 274 |

|                                   |            |
|-----------------------------------|------------|
| <b>第 8 章 CDN 商业化服务现状 .....</b>    | <b>279</b> |
| 8.1 CDN 产业分析 .....                | 280        |
| 8.1.1 CDN 产业链分析 .....             | 280        |
| 8.1.2 CDN 服务的价值分析 .....           | 282        |
| 8.1.3 CDN 服务运营方式分析 .....          | 285        |
| 8.2 CDN 的商业服务模式 .....             | 286        |
| 8.2.1 CDN 的计费方式 .....             | 286        |
| 8.2.2 CDN 的增值服务 .....             | 289        |
| 8.2.3 CDN 客户决策要点 .....            | 294        |
| 8.3 典型案例分析 .....                  | 296        |
| 8.3.1 视频网站 .....                  | 296        |
| 8.3.2 门户网站 .....                  | 297        |
| 8.3.3 政府网站 .....                  | 298        |
| 8.3.4 企业网站 .....                  | 299        |
| 8.3.5 云计算 .....                   | 300        |
| 8.3.6 小结 .....                    | 302        |
| 8.4 典型服务商介绍 .....                 | 303        |
| 8.4.1 国外 CDN 运营商的先驱——Akamai ..... | 303        |
| 8.4.2 国内运营商简介 .....               | 306        |
| <b>第 9 章 CDN 发展展望 .....</b>       | <b>309</b> |
| 9.1 新时代对 CDN 的要求 .....            | 310        |
| 9.2 CDN 技术发展趋势 .....              | 313        |
| 9.3 CDN 与云计算 .....                | 315        |
| 9.3.1 云计算——第三次 IT 革命 .....        | 315        |
| 9.3.2 CDN 是云计算吗 .....             | 317        |
| 9.3.3 CDN 与云计算技术的结合 .....         | 323        |
| 9.4 CDN 与 P2P .....               | 325        |
| 9.4.1 P2P 技术概述 .....              | 325        |

|                                      |            |
|--------------------------------------|------------|
| 9.4.2 P2P 流量的变化趋势及优劣势分析 .....        | 329        |
| 9.4.3 CDN 与 P2P 技术的结合 .....          | 334        |
| 9.5 CDN 的商业服务发展趋势 .....              | 337        |
| <br>                                 |            |
| <b>附录 A CDN 试验床实施指南 .....</b>        | <b>341</b> |
| A.1 试验床架构概述 .....                    | 342        |
| A.2 基础集群环境搭建 .....                   | 344        |
| A.2.1 服务器虚拟化环境部署 .....               | 344        |
| A.2.2 虚拟机管理基本操作 .....                | 347        |
| A.3 代理缓存环境搭建 .....                   | 349        |
| A.3.1 Apache HTTP 服务器的安装与配置 .....    | 350        |
| A.3.2 Squid 代理缓存服务器的安装与配置 .....      | 352        |
| A.3.3 CDN 试验床代理缓存功能的演示和验证 .....      | 355        |
| A.4 边缘节点四层负载均衡 .....                 | 357        |
| A.4.1 LVS 负载均衡服务器的安装与配置 .....        | 358        |
| A.4.2 CDN 试验床四层负载均衡功能的演示和验证 .....    | 362        |
| A.5 边缘节点七层负载均衡 .....                 | 364        |
| A.5.1 BIND 域名服务器的安装与配置 .....         | 365        |
| A.5.2 Nginx 负载均衡服务器的安装与配置 .....      | 368        |
| A.5.3 CDN 试验床七层负载均衡功能的演示和验证 .....    | 371        |
| A.6 多边缘节点负载均衡 .....                  | 373        |
| A.6.1 Apache 服务器和 BIND 服务器的配置 .....  | 374        |
| A.6.2 CDN 试验床多边缘节点负载均衡功能的演示和验证 ..... | 379        |
| A.7 小结 .....                         | 380        |
| <br>                                 |            |
| <b>参考文献 .....</b>                    | <b>381</b> |

# 第1章 引言



- || 1.1 CDN 的基本概念和产生背景
- || 1.2 CDN 的基本工作过程
- || 1.3 CDN 的发展历史
- || 1.4 CDN 对互联网产业的价值和作用



在这个信息互联的时代，互联网已经成了生活必需品，我们习惯了各种互联网应用俯拾即是，触手可及。也许很多用户都没有注意，在互联网这张大网上，已经悄然生长出一张时刻为人们服务的网络，它像一位隐形的快递员，将各种各样的内容交付给用户。这就是本书中我们要一起学习的内容分发网络（CDN，Content Distribute Network）。

## 1.1 CDN 的基本概念和产生背景

对于 CDN 这个名词，读者大可以望文生义：Content Distribute Network，直译成内容分发网络，或者也有人写成 Content Delivery Network，内容交付网络。很显然，CDN 完成的是将内容从源站传递到用户端的任务，我们当然不需要再解释什么叫做“内容分发”或者“内容交付”了，要解释的是 CDN 在这个分发或者交付的过程中体现了什么价值，为什么需要 CDN 来交付而不是直接通过互联网交付呢？这个话题有必要短话长说一下，这对理解本书中 CDN 的工作原理、各项关键技术都有帮助。

大家常说的互联网，是广义的互联网，由两层组成：一层是以 TCP/IP 为代表的网络层（也是狭义互联网概念）；另一层是以万维网 WWW 为代表的应用层。目前普遍存在一个认识误区，就是将互联网和万维网混作一谈。认清互联网的本质，辨识清楚互联网和万维网的区别，是理解整个互联网经济的关键和基础，也是认识 CDN 的基础。

以 TCP/IP 为核心的狭义的互联网（Internet），实际上是广义互联网的下层，是网络基础，更一般地说就是 TCP/IP 网络。这一层的主要作用是通过计算机之间的互联，将各种信息的数据报文以极低的成本进行传输，俗称“管道”，所有信息和内容在这个管道里进行传送。互联网的设计理念是：网络是中立和无控制的，任何人都没有决定权；网络是应用无关的，它的任务就是如何更好

地将数据包进行端到端传输。这个设计理念从互联网诞生之初到现在从未被撼动，任何针对某种（类型的）内容对互联网进行优化的尝试其最后效果都不甚理想。因此，我们可以认为互联网不会试图对任何内容进行传输优化。

以万维网 WWW 为代表的应用层，是广义互联网的上层。这一层包括很多种类型的流量和应用，邮件、软件、在线影视、游戏、电子商务、移动应用等，所有 SP（Service Provider，服务提供商）提供的都是这些用户看得见、摸得着的应用，它们丰富和方便了人们的生活，构成了我们常说的互联网业务和信息经济。

举个铁路的例子来解释两者的差别和关系：互联网是铁路轨道和信号系统，万维网则是在铁路上运行的列车之一。而在铁路上，除了万维网这个高速列车以外，还有慢车、通勤列车、货运列车和专业维修列车等。在互联网上，万维网是巨大的和非常重要的，但它并不是唯一。那些不使用 WWW 的应用同样运行在互联网上，互联网的巨大远远超过运行在其上的任何东西。

现在，我们看看网络层与应用层这上下两层的磨合中是否存在一个问题。从网络层面来看，在互联网这个铁路网中，有 4 个地方会造成列车拥堵，如图 1-1 所示，分别如下所述。



图 1-1 影响互联网传输的 4 个因素

（1）“第一公里”，这是指万维网流量向用户传送的第一个出口，是网站服务器接入互联网的链路所能提供的带宽。这个带宽决定了一个网站能为用户提供访问速度和并发访问量。一个网站，其服务的用户越多，对其出口带宽的要求就越大，当用户请求的数据量超过网站的出口带宽，就会在出口处形成

## CDN 技术详解

拥塞。越是业务繁忙时，用户的访问数量越多，这个拥塞就越严重，网站会在最需要向用户提供服务时失去用户。

(2) “最后一公里”，这是指万维网流量向用户传送的最后一段接入链路，即用户接入带宽。用户的平均接入带宽，是影响互联网上层应用发展的决定性因素之一。在互联网发展的初期，用户主要通过拨号上网或 ISDN 等方式上网，网络接入速度很低，所以互联网内容以带宽占用非常小的文字为主，Telnet、BBS 都是那时的主流应用。当万维网出现后，人机交互更加方便友好的多媒体内容开始在互联网上传播，接入带宽成为制约用户使用互联网的主要瓶颈。从 2001 年开始，电信运营商开始大力开展 ADSL 等宽带接入服务，随着带宽的不断提升和接入手段的丰富（光纤入户、Wifi、3G 等），近年来“最后一公里”的问题得到很大改善，特别是这两年中国电信等大力开展以光纤接入为手段的宽带提速服务，“最后一公里”的瓶颈问题已经基本得到解决。

(3) 对等互联关口。这里的“对等互联”是指不同基础运营商之间的互联互通，一般两个运营商之间只有两三个互联互通点，可以想象这两三个点上产生多么大的流量。当某个网站服务器部署在运营商 A 的 IDC 机房里，运营商 B 的用户要访问该网站，就必须经过 A、B 之间的互联互通点进行跨网访问。从互联网的架构来看，不同网络之间的互联互通带宽，对任何一个运营商网络的流量来说，占比都比较小，收敛比是非常高的，因此这里通常都是互联网传输中的拥堵点。

(4) 长途骨干传输。首先是长距离传输时延问题，从网站服务器到用户之间要经过网站所在 IDC、骨干网、用户所在城域网、用户所在接入网等，距离非常遥远，因此不可避免地带来较长的传输时延，影响用户体验，这一问题也是互联网本身无法解决的问题。其次是骨干网拥塞问题，由于互联网上的绝大部分流量都要通过骨干网络进行传输，这就要求骨干网络的承载能力必须与互联网的应用同步发展，但实际上两者并不是同步的，当骨干网络的升级和扩容滞后于互联网之上的应用的发展时，就会阶段性地使得大型骨干网的承载能力

成为影响互联网性能的瓶颈。

在应用层中，SP们都在不断优化业务体验，其中最值得关注的就是服务响应时间。服务响应时间基本是由服务器响应时间和网络时延组成的。影响服务器响应时间的因素包括协议处理时间、程序性能优化、内容读取速度等方面，网络时延则是由数据报文在网络传送中被各个路由器、交换机转发产生的时延总和。在互联网领域有一个“8秒定律”，用户访问一个网站时，如果等待网页打开的时间超过8秒，会有超过30%的用户放弃等待。根据KissmeTrics最近的一项调查统计：一个网站10秒后网页打不开，会有40%的用户跳出该页面；大部分手机用户愿意等待的加载时间为6~10秒；1秒延迟会导致转化率下降7%。算一下，假如一电子商务网站每天收入10万元，1秒钟的页面延迟将使它每年损失掉250万元。

CDN的产生与上面分析的一系列问题息息相关，如果这些问题没有手段缓解，那整个互联网将是与今天完全不同的另一番景象了。1995年，麻省理工学院教授，互联网发明者Tim Berners-Lee预见到当时互联网使用者已经习以为常的网络拥挤难题，未来会成为互联网应用的最大障碍。于是他向同事提出挑战，要发明一种全新的、从根本上解决问题的方法来推送互联网内容。他的这一提议造就了今天被大家普遍接受的互联网基础服务——CDN。

## 1.2 CDN的基本工作过程

使用CDN会极大地简化网站的系统维护工作量，网站维护人员只需将网站内容注入CDN的系统，通过CDN部署在各个物理位置的服务器进行全网分发，就可以实现跨运营商、跨地域的用户覆盖。由于CDN将内容推送到网络边缘，大量的用户访问被分散在网络边缘，不再构成网站出口、互联互通点的资源挤占，也不再需要跨越长距离IP路由了。

## CDN 技术详解

CDN 是如何工作的呢？让我们先看看没有 CDN 服务时，一个网站是如何向用户提供服务的。

今天我们看到的网站系统基本上都是基于 B/S 架构的。B/S 架构，即 Browser-Server（浏览器-服务器）架构，是对传统 C/S 架构的一种变化或者改进架构。在这种架构下，用户只需使用通用浏览器，主要业务逻辑在服务器端实现。B/S 架构，主要是利用了不断成熟的 WWW 浏览器技术，结合浏览器的多种 Script 语言（VBScript、JavaScript 等）和 ActiveX 等技术，在通用浏览器上实现了 C/S 架构下需要复杂的软件才能实现的强大功能。

用户通过浏览器等方式访问网站的过程如图 1-2 所示。

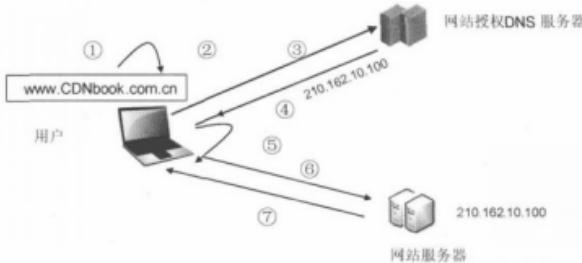


图 1-2 互联网用户服务访问流程

- ① 用户在自己的浏览器中输入要访问的网站域名。
- ② 浏览器向本地 DNS 服务器请求对该域名的解析。
- ③ 本地 DNS 服务器中如果缓存有这个域名的解析结果，则直接响应用户的解析请求。
- ④ 本地 DNS 服务器中如果没有关于这个域名的解析结果的缓存，则以递归方式向整个 DNS 系统请求解析，获得应答后将结果反馈给浏览器。

⑤浏览器得到域名解析结果，就是该域名相应的服务设备的 IP 地址。

⑥浏览器向服务器请求内容。

⑦服务器将用户请求内容传送给浏览器。

在网站和用户之间加入 CDN 以后，用户不会有任何与原来不同的感觉。最简单的 CDN 网络有一个 DNS 服务器和几台缓存服务器就可以运行了。一个典型的 CDN 用户访问调度流程如图 1-3 所示。

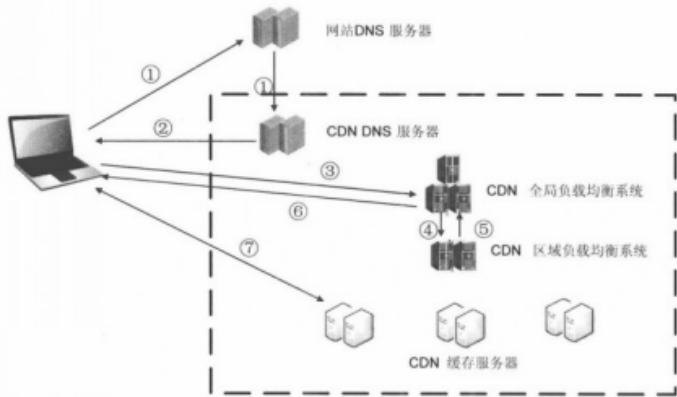


图 1-3 引入 CDN 后的典型用户访问流程

①当用户点击网站页面上的内容 URL，经过本地 DNS 系统解析，DNS 系统会最终将域名的解析权交给 CNAME 指向的 CDN 专用 DNS 服务器。

②CDN 的 DNS 服务器将 CDN 的全局负载均衡设备 IP 地址返回用户。

③用户向 CDN 的全局负载均衡设备发起内容 URL 访问请求。

④CDN 全局负载均衡设备根据用户 IP 地址，以及用户请求的内容 URL，选择一台用户所属区域的区域负载均衡设备，告诉用户向这台设备发起

请求。

⑤区域负载均衡设备会为用户选择一台合适的缓存服务器提供服务，选择的依据包括：根据用户 IP 地址，判断哪一台服务器距用户最近；根据用户所请求的 URL 中携带的内容名称，判断哪一台服务器上有用户所需内容；查询各个服务器当前的负载情况，判断哪一台服务器尚有服务能力。基于以上这些条件的综合分析之后，区域负载均衡设备会向全局负载均衡设备返回一台缓存服务器的 IP 地址。

⑥全局负载均衡设备把服务器的 IP 地址返回给用户。

⑦用户向缓存服务器发起请求，缓存服务器响应用户请求，将用户所需内容传送到用户终端。如果这台缓存服务器上并没有用户想要的内容，而区域均衡设备依然将它分配给了用户，那么这台服务器就要向它的上一级缓存服务器请求内容，直至追溯到网站的源服务器将内容拉到本地。

DNS 服务器根据用户 IP 地址，将域名解析成相应节点的缓存服务器 IP 地址，实现用户就近访问。使用 CDN 服务的网站，只需将其域名解析权交给 CDN 的 GSLB 设备，将需要分发的内容注入 CDN，就可以实现内容加速了。

### 1.3 CDN 的发展历史

CDN 是为互联网上的应用服务的，它伴随着互联网的发展而逐步成长，其发展过程中的高峰低谷、起起落落与整个互联网的发展轨迹基本保持一致，图 1-4 是 CDN 的历史发展曲线，我们借此来回顾一下 CDN 的发展历史。

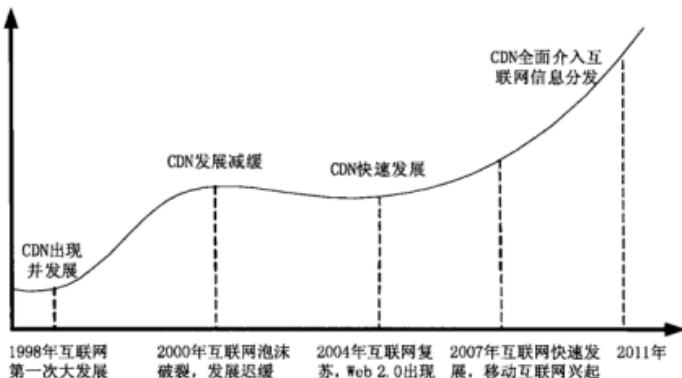


图 1-4 CDN 发展历史

### 1. 第一阶段：CDN 缘起

1991 年之后的近十年间，公众主要以拨号方式接入互联网，带宽低而且网民数量少，此时主要的瓶颈在最后一公里——用户接入带宽，而没有给提供内容的服务器和骨干传输网络带来太大的压力。随着互联技术的发展和网民数量的增加，给内容源服务器和传输骨干网络带来越来越大的压力，互联网瓶颈从接入段逐渐向骨干传输网络和服务器端转移。

1995 年，麻省理工学院教授，互联网发明者之一 Tim Berners-Lee 发起的一项技术挑战造就了后来鼎鼎大名的 CDN 服务公司 Akamai。Berners-Lee 博士预见到在不久的将来网络拥塞将成为互联网发展的最大障碍，于是他提出一个学术难题，要发明一种全新的、从根本上解决问题的方法来实现互联网内容的无拥塞分发，这项学术难题最终催生出一种革新性的互联网服务——CDN。当时 Berners-Lee 博士隔壁是 Tom Leighton 教授的办公室，一位麻省理工学院应用数学教授，他被 Berners-Lee 的挑战激起了兴趣。Leighton 博士意识到应用数学和运筹学可以解决网络拥塞的问题，于是他请研究生 Danny C. Lewin 和其他

## CDN 技术详解

几位顶级研究人员一起破解这个技术难题。随后另外几位计算机科学和数据网络方面的科学家也加入 Leighton 博士的队伍中来，他们开发了数学运算法则来处理内容的动态路由计算，并开始实施自己的商业计划，最终成立 Akamai 公司。这些世界级的科学家开发出一套突破性的运算法则，用于在网络服务器所组成的大型网络中智能组织路由和复制内容。

Akamai 公司通过智能化的互联网分发，结束了“World Wide Wait”（世界一起等待）的尴尬局面。公司 1999 年开始提供商业服务，并宣布世界上网络流量最大的互联网公司雅虎成为自己的合同客户。

Akamai 是全球第一家 CDN 网络运营商，从诞生之日起，就一直是全世界顶级的 CDN 服务商和 CDN 服务的领跑者。Akamai 的成功表明互联网内容分发业务有着巨大的市场前景。

### （2）第二阶段：CDN 第一次发展浪潮

1999 年到 2001 年是全球互联网发展的高潮期，HTTP 网页内容的加速需求非常大，CDN 成为产业关注的热点。2001 年，Limelight Networks 公司在美国亚利桑那州成立，是除 Akamai 之外最主要的 CDN 公司，自高盛将 1.2 亿美元投入 Limelight 后，全球 CDN 的发展呈现出风起云涌之势。Akamai 和 Limelight 分别代表了“节点租用”和“节点自建”两种发展模式，引领了全球 CDN 行业的技术潮流。在这一时期，除了如 Akamai、Limelight、Level3 等这样独立、专业的 CDN 服务提供商，大型的 IDC 企业看到 CDN 巨大的市场前景也纷纷转型，加入这一行业。IDC 企业的加入进一步推动了 CDN 行业的发展，与传统的独立 CDN 公司形成对峙之势。比如国外著名的 IDC——Digital Island 通过其遍布全美国的、数量众多的数据中心，建立了其自己的 CDN 网络，将 CDN 服务作为一种增值服务向它的数据中心的客户提供。

在中国，互联网的高速发展同样始于 20 世纪 90 年代末。以新浪、搜狐、网易三大门户为代表，众多资本、科技人才投入其中，网站和互联网服务如雨

后春笋般蓬勃生长，启动了国内第一次互联网发展高潮。互联网内容的丰富和功能的拓展吸引了越来越多的用户，网上冲浪成为当时最时髦的娱乐形式。网民数量的剧增给网络带来巨大的压力，导致网络服务质量和用户体验下降，同时限制了流媒体等新业务的发展。在这样的背景下，中国的 CDN 产业应运而生。1998 年，国内第一家专业 CDN 服务公司——蓝汛（ChinaCache）公司成立，2000 年蓝汛获得了信息产业部颁发的第一个 CDN 试运行许可证，正式成为我国第一家专业 CDN 服务提供商。2000 年 10 月初，网宿公司成立。2001 年 10 月，新浪成为我国第一个 CDN 服务商业用户，中国的 CDN 商用市场初具规模。

### 3. 第三阶段：互联网泡沫的冲击和 CDN 第二次大发展

2001 年，第一次互联网泡沫破碎，大量.com 公司倒闭，网站关闭。CDN 客户一夜之间骤减，一息尚存的网站也缩减成本开支，CDN 产业几乎立刻进入了停滞期。彼时 CDN 服务商鼻祖 Akamai 公司也难逃此劫，加上该公司的创始人之一、早期发明者之一的 Daniel C. Lewin 在 2001 年“9·11”恐怖袭击中不幸遇难，更为 CDN 行业的发展雪上加霜。

虽然互联网泡沫破裂对整个行业产生了巨大的冲击，但互联网复兴的种子已播下，第一次泡沫期间大量投资建设的基础设施，为日后产业的再次发展奠定了良好的物质基础。

从 2002 年开始，DSL 等宽带技术在全球逐渐普及，用户接入带宽提高到 Mb 级别，为网络流媒体服务提供了基础条件。从 2004 年起，伴随着互联网的回暖和发展，流媒体服务的发展和 Web 2.0 的兴起对 CDN 提出了新的技术要求，CDN 的需求开始回升并持续增加，CDN 又变得热门起来。

首先，传统 HTTP 和下载使网络数据量飞速上升，网络游戏产业逐渐成熟，特别是网络视频等需要高带宽的内容，对服务器和网络带宽的压力更大，对 CDN 服务需求迫切。其次，网站的内容类型不断增加和丰富，在新的需求下，

流媒体、Flash、视频和下载等网站内容及业务成了新的主要应用对象。为了给软件下载、视频流媒体、企业 Web 应用、B2B 交易和 Web 2.0 互动等各种服务加速，传统的 CDN 技术之上又增加了压缩、流量整形、智能路由和网络优化等技术。第三，随着 CDN 能够提供加速的内容类型不断丰富，其提供的服务也已从单纯的内容加速拓展到应用和服务的加速。Akamai 公司营销主管 Kieran Taylor 曾感慨地说：“‘内容分发网络’这个词确实有些过时了，我们的设想是为所有在线业务加速。”

除了带来新的技术要求和业务需求外，Web 2.0 和流媒体的发展将互联网的发展推向了新的高峰，互联网逐步形成了清晰稳定的赢利模式，生存力的增强使得互联网公司变得更加健康和稳定，为 CDN 服务生成了真正的市场和稳定的客户群。

总而言之，市场需求的急速膨胀与 CDN 自身的发展，包括技术的成熟、设备价格的下降等因素，共同引发了 CDN 的新一轮发展热潮。CDN 技术自诞生时的第一次爆发式发展之后，又迎来一段难得的发展盛世。在我国从 2006 年开始，随着网络视频应用的普及，CDN 进入快速发展时期，到 2009 年底，中国 CDN 市场营业收入已达到 5.01 亿元人民币。

与此同时，CDN 市场的风险投资交易金额逐年走高，2008 年 CDN 市场的风险资本交易金额更是达到了创纪录的 3.25 亿美元。蓝汛公司从 2004 年启动融资开始，英特尔投资、集富亚洲、InvEStor、启明创投、Ignition、Starr International、SIG 这 7 家基金都先后成为蓝汛的投资人，前后三轮融资共引进 5000 万美元。包括 2005 年获得的第一笔来自于集富亚洲、英特尔投资等的 850 万美元风险投资；2007 年获集富亚洲、英特尔投资和启明创投等参与的第二轮 3150 万美元的投资；2009 年 9 月，蓝汛再获英特尔投资及其他风险投资公司超过 1000 万美元的投资。2007 年 5 月，网宿科技获得首轮融资，总额为 4000 万元人民币。

2009 年 10 月 30 日，上海网宿科技股份有限公司作为首批创业板 28 家上

市公司之一，正式登陆深交所创业板上市。2010年10月1日（美国当地时间），蓝汛通信（NasdaqGM: CCIH）成功登陆纳斯达克交易市场。两家国内CDN服务领先者的成功上市，是国内CDN产业发展的重要里程碑，也标志着资本市场和公众对CDN产业的认可。

2010年开始的云计算风潮对CDN也产生了不小的影响。一方面，很多云计算平台在对外提供服务时不可避免地会用到CDN的分发能力。另一方面，CDN的技术特点使得它本身就很像一种云服务，很多CDN服务商也正在积极尝试各种云计算技术在CDN系统中的应用。云计算究竟会对CDN产生什么样的影响？我们拭目以待。但可以肯定的是，这将是一种良性的促进，也许将推动CDN技术和业务发展进入一个新的阶段。

## 1.4 CDN对互联网产业的价值和作用

在国内，很长一段时间CDN都是幕后英雄，默默地扮演着互联网“快递员”的角色。随着宽带网络的普及和发展，特别是2009年以来伴随着网宿和蓝汛的上市，CDN重新成为电信行业资本市场的热点，引起互联网公司、电信运营商和产业资本市场的关注。业界普遍认为，CDN最终将成为架设在传统IP网之上的一个必需的内容传递网络，将对整个互联网的架构和产业格局产生深远的影响。

首先，CDN的发展促进了整个互联网产业的进一步分工。纵观整个宽带服务的价值链，内容提供商和用户位于整个价值链的两端，中间依靠网络服务提供商将其串接起来。随着互联网工业的成熟和商业模式的变革，在这条价值链上的角色越来越多，也越来越细分。比如内容/应用的提供商、托管服务提供商、骨干网络服务提供商、接入服务提供商等。在这一条价值链上的每个角色分工合作、各司其职才能为用户提供良好的服务，从而创造多赢的局面。从内

## CDN 技术详解

容与网络的结合模式上看，IDC 热潮催生了内容托管服务提供商这一角色，但内容托管并不能有效解决内容的快速分发问题。骨干带宽被迅速消耗掉，IP 网络流量秩序濒于失衡，因此 CDN 成为一种显而易见的选择。CDN 将内容推到网络的边缘，为用户提供就近性的边缘服务，从而保证服务的质量和整个网络上的访问秩序，解决了困扰内容提供商的内容“集中与分散”的两难选择。

其次，对于电信运营商，CDN 是真正体现管道智能化的技术。CDN 与网络联系紧密，通过与各级网络之间的调度配合，在给用户提供优质服务的同时，也能降低骨干网的传输压力和峰谷变化。互联网的本质在于信息的有效传递，构建迅速良好的信息传递机制是永恒的话题。CDN 利用有效的缓存、均衡和智能路由选择等技术，对互联网信息进行协调组织，形成良好的信息传递保障机制，就像水系中的湖泊，在调节水量的同时，保证了主干和支流水系的平稳。CDN 的分发和缓存机制，保证了边缘节点临时存储一些热门访问内容，同时通过相应的调度策略，保证不同节点之间的压力平衡，避免某个路由或节点的压力过大。正是通过这样的缓存和调度策略，才确保整体分发网络的健壮性。

较大的 CDN 网络，尤其是商业化 CDN 网络，都部署了多层节点覆盖不同大小的区域，就像一个规划完善的物流系统，如图 1-5 所示。CDN 的内容注入点类似货物交付点，经过全国运营网络的分发，运送到目标城市，这一过程类似骨干传输；进入某个城市之后，首先到达所在城市的货场或临时仓储地，然后在城市内分发，城市内的货场相当于城域网的临时缓存，同时可以通过不同的货场选择来进行压力分配。所不同的是，快递系统可以在任何边缘点进行货物交付，而 CDN 通常全网只有一个或少数几个内容注入点。

CDN 可以根据不同的客户级别、内容类型和区域等信息，对内容传输实施不同级别的保障。利用流量标识与 CDN 智能网管的结合，运营商可以实现网络差异化服务能力。因此 CDN 对于电信运营商具有极其重要的战略意义。

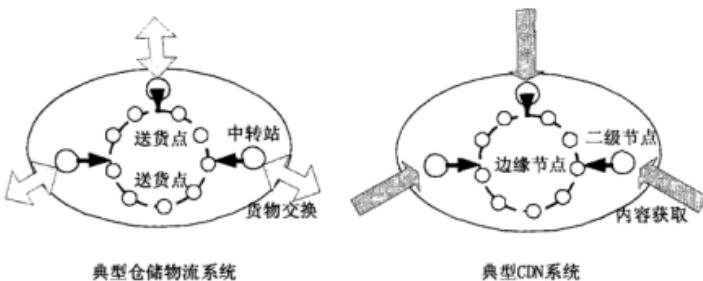
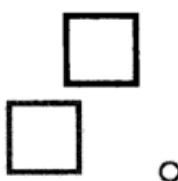


图 1-5 与仓储物流系统类似的 CDN 系统

第三，CDN 的进一步发展促进了互联网上除运营商和 SP 之外的第三服务业的蓬勃发展。CDN 技术和服务将内容提供者和内容使用者（消费者）连接起来，产业链进一步细化。在内容传输层面能够提供更多的增值服务，包括用户行为数据的统计分析，网络健康状态监控，网络调整等，这些服务将催生一批依赖于内容传输的第三方服务。基础电信运营商负责对互联网传输的基础网络进行保障，这一保障确保三层以下（甚至六层以下）的服务对内容提供者是透明的，CDN 进一步提升了网络的保障，内容制作者只需关注内容本身，和网络相关部分完全外包给 CDN 运营商，更加有利于降低中小型创业公司的起步成本。对于专业的 CDN 服务提供商而言，CDN 已不仅仅是传统意义上的网络加速和负载均衡，更重要的是广域网络服务的维护外包，利用基础网络运营商各地区的带宽价格差异，在满足用户需要的前提下，降低网络托管成本。同时通过不同客户之间业务交叉，利用不同客户之间的峰谷重叠，大幅度提高网络资源的利用率，避免租用网络资源的浪费。



## 第 2 章 CDN 技术概述

---

- || 2.1 CDN 的系统架构
- || 2.2 CDN 系统分类
- || 2.3 小结



CDN 基于这样的原理：1. 挑选最优设备为用户提供服务；2. 如果某个内容被很多用户所需要，它就被缓存到距离用户最近的节点中。

CDN 公司在整个互联网上部署数以百计的 CDN 服务器（Cache），这些服务器通常在运营商的 IDC 中，尽量靠近接入网络和用户。CDN 在 Cache 中复制内容，当内容的提供者更新内容时，CDN 向 Cache 重新分发这些被刷新的内容。CDN 提供一种机制，当用户请求内容时，该内容能够由以最快速度交付的 Cache 来向用户提供，这个挑选“最优”的过程就叫做负载均衡。被选中的最优 Cache 可能最靠近用户，或者有一条与用户之间条件最好的路径。

---

在本章中，将一一讲述 CDN 是如何完成 内容缓存、负载均衡、流媒体加速、动态内容加速 等任务的，以及完成这些任务涉及的各种关键技术。

---

## 2.1 CDN 的系统架构

### 2.1.1 功能架构

CDN 技术自 1998 年诞生以来，伴随着互联网的高速发展，其技术一直在持续演进和完善，但基本的 CDN 功能架构在 2003 年左右就已基本形成和稳定下来。从功能上划分，典型的 CDN 系统架构由分发服务系统、负载均衡系统和运营管理三大部分组成，如图 2-1 所示。

首先，来看看 分发服务系统。该系统的主要作用是实现将内容从内容源中心向边缘的推送和存储，承担实际的内容数据流的全网分发工作和面向最终用户的数据请求服务。分发服务系统最基本的工作单元就是许许多多的 Cache 设备（缓存服务器），Cache 负责直接响应最终用户的访问请求，把缓存在本地的内容快速地提供给用户。同时 Cache 还负责与源站点进行内容同步，把更

## CDN 技术详解

新的内容以及本地没有的内容从源站点获取并保存在本地。

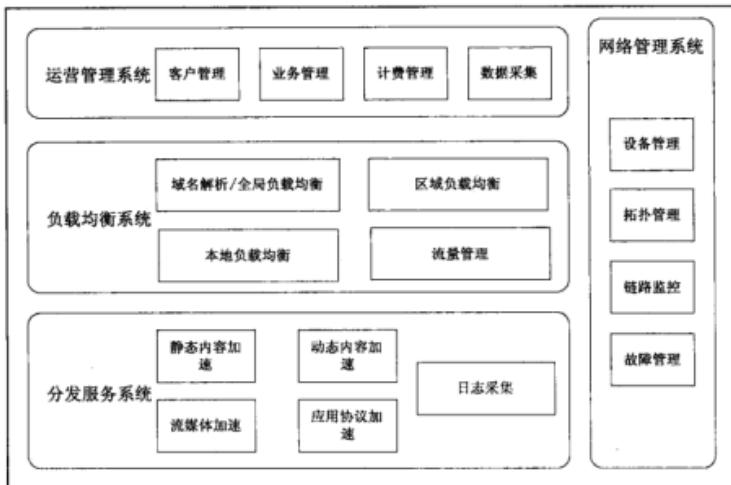


图 2-1 CDN 系统功能架构图

一般来说，根据承载内容类型和服务种类的不同，分发服务系统会分为多个子服务系统，如网页加速子系统、流媒体加速子系统、应用加速子系统等。每个子服务系统都是一个分布式服务集群，由一群功能近似的、在地理位置上分布部署的 Cache 或 Cache 集群组成，彼此间相互独立。每个子服务系统设备集群的数量根据业务发展和市场需要的不同，少则几十台，多则可达上万台，对外形成一个整体，共同承担分发服务工作。Cache 设备的数量、规模、总服务能力是衡量一个 CDN 系统服务能力的最基本的指标。

对于分发服务系统，在承担内容的更新、同步和响应用户需求的同时，还需要向上层的调度控制系统提供每个 Cache 设备的健康状况信息、响应情况，有时还需要提供内容分布信息，以便调度控制系统根据设定的策略决定由哪个 Cache（组）来响应用户的请求最优。

**负载均衡系统**是一个CDN系统的神经中枢，主要功能是负责对所有发起服务请求的用户进行访问调度，确定提供给用户的最终实际访问地址。大多数CDN系统的负载均衡系统是分级实现的，这里以最基本的两级调度体系进行简要说明。一般而言，两级调度体系分为全局负载均衡（**GSLB**）和本地负载均衡（**SLB**）。其中，全局负载均衡（GSLB）主要根据用户就近性原则，通过对每个服务节点进行“最优”判断，确定向用户提供服务的Cache的物理位置。最通用的**GSLB**实现方法是基于DNS解析的方式实现，也有一些系统采用了应用层重定向等方式来解决，关于**GSLB**的原理和实现方法将在本书第5章进行讲解。本地负载均衡（SLB）主要负责节点内部的设备负载均衡，当用户请求从**GSLB**调度到**SLB**时，**SLB**会根据节点内各Cache设备的实际能力或内容分布等因素对用户进行重定向，常用的本地负载均衡方法有基于4层调度、基于7层调度、链路负载调度等，具体的内容在本书第4章进行讲解。

CDN的运营管理与一般的电信运营管理类似，分为运营管理与网络管理两个子系统。运营管理子系统是CDN系统的业务管理功能实体，负责处理业务层面的与外界系统交互所必需的一些收集、整理、交付工作，包含客户管理、产品管理、计费管理、统计分析等功能。其中客户管理指对使用CDN业务的客户进行基本信息和业务规则信息的管理，作为CDN服务提供的依据。产品管理，指CDN对外提供的具体产品包属性描述、产品生命周期管理、产品审核、客户产品状态变更等。计费管理，指在对客户使用CDN资源情况的记录的基础上，按照预先设定的计费规则完成计费并输出账单。统计分析模块负责从服务模块收集日常运营分析和客户报表所需数据，包括资源使用情况、内容访问情况、各种排名、用户在线情况等数据统计和分析，形成报表提供给网管人员和CDN产品使用者。网络管理子系统实现对CDN系统的网络设备管理、拓扑管理、链路监控和故障管理，为管理员提供对全网资源进行集中化管理操作的界面，通常是基于Web方式实现的。

图2-2是国内第一家CDN服务商——蓝汛（ChinaCache）公司在2005年发布的《CDN技术白皮书》中描述的CDN系统架构，是一套实际的、运行良

## CDN 技术详解

好的商用 CDN 系统。我们以该架构为例，与图 2-1 展示的 CDN 系统功能架构进行对照分析，帮助读者理解 CDN 系统的组成。

在蓝汛 CDN 架构中，CCN ( ChinaCache Node ) 模块对应于图 2-1 中的分发服务系统，是 CDN 的基本服务模块，由分布于各个城市、各个运营商网络中的 Cache 设备和辅助设备组成。

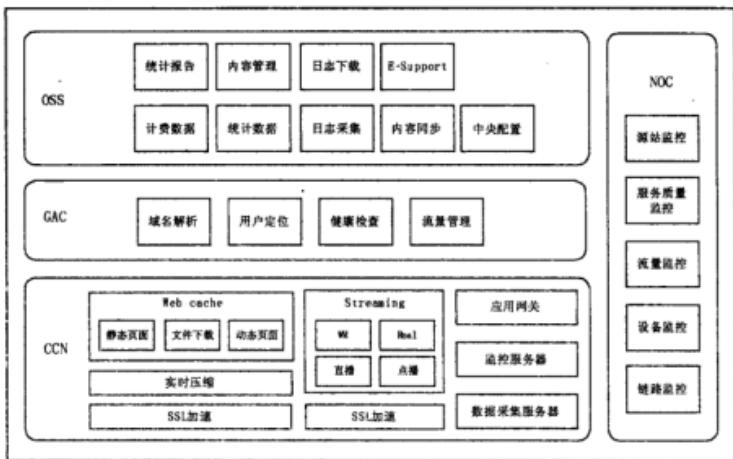


图 2-2 ChinaCache CDN 系统架构图

GAC ( Global Access Controller ) 模块对应于图 2-1 中的负载均衡系统，主要采用了智能 DNS 解析方案，负责通过域名解析应答实现将用户请求调度到最优服务节点的目的。

NOC ( Network Operating Center ) 模块对应图 2-1 中的网络管理系统，负责对全网进行  $7 \times 24$  小时的监控和管理。NOC 可以监控 ChinaCache CDN 中的链路状况、节点响应速度、设备运行状态，也可以监控到客户的源站点运行状况等信息，一旦发现异常马上采取相应措施予以解决，是保障 CDN 服务安

全可靠性的重要系统。值得一提的是，蓝汛的 NOC 中设置了源站点监控功能，对客户源站进行可达性监控，从而减轻或者避免由于源站故障造成的服务中断。

OSS (Operating Support System) 模块对应图 2-1 中的运营管理子系统，负责采集和汇总各个 CCN 的日志记录信息，然后由中央处理软件加以整理和分析，最后通过客户服务门户进行发布。蓝汛的客户可以通过 OSS 提供的查询界面来查询加速页面或频道的实时流量、流量分布、点击数量、访问日志等信息。

### 2.1.2 部署架构

CDN 系统设计的首要目标是尽量减少用户的访问响应时间，为达到这一目标，CDN 系统应该尽量将用户所需要的内容存放在距离用户最近的位置。也就是说，负责为用户提供内容服务的 Cache 设备应部署在物理上的网络边缘位置，我们称这一层为 CDN 边缘层。CDN 系统中负责全局性管理和控制的设备组成中心层，中心层同时保存着最多的内容副本，当边缘层设备未命中时，会向中心层请求，如果在中心层仍未命中，则需要中心层向源站回源。不同 CDN 系统设计之间存在差异，中心层可能具备用户服务能力，也可能不直接提供服务，只向下级节点提供内容。如果 CDN 网络规模较大，边缘层设备直接向中心层请求内容或服务会造成中心层设备压力过大，就要考虑在边缘层和中心层之间部署一个区域层，负责一个区域的管理和控制，也保存部分内容副本供边缘层访问。

图 2-3 是一个典型的 CDN 系统三级部署示意图。

节点是 CDN 系统中最基本的部署单元，一个 CDN 系统由大量的、地理位置上分散的 POP 节点组成，为用户提供就近的内容访问服务。CDN 节点网络主要包含 CDN 骨干点和 POP 点。CDN 骨干点和 CDN POP 点在功能上不同，中心和区域节点一般称为骨干点，主要作为内容分发和边缘未命中时的服务点；边缘节点又被称为 POP ( point-of-presence ) 节点。CDN POP 点主要作为直接

## CDN 技术详解

向用户提供服务的节点。但是，从节点构成上来说，无论是 CDN 骨干点还是 CDN POP 点，都由 Cache 设备和本地负载均衡设备构成。

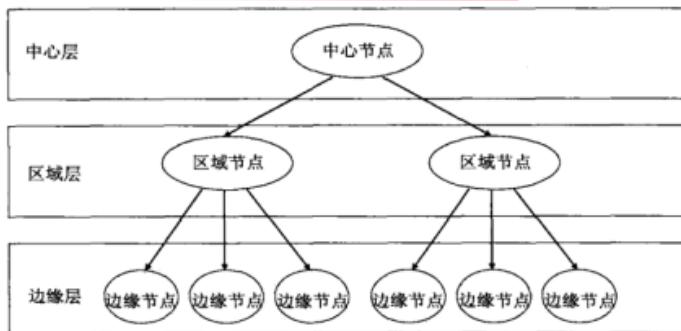


图 2-3 三级 CDN 系统部署图

在一个节点中，Cache 设备和本地负载均衡设备的连接方式有两种：一种是旁路方式，一种是穿越方式，如图 2-4 所示。

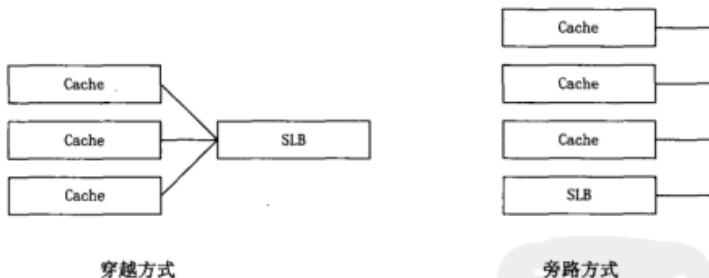


图 2-4 CDN 节点内 SLB 和 Cache 的连接方式

在穿越方式下，SLB 一般由 L4-L7 交换机实现，SLB 向外提供可访问的公网 IP 地址( VIP )，每台 Cache 仅分配私网 IP 地址，该台 SLB 下挂的所有 Cache 构成一个服务组。所有用户请求和媒体流都经过该 SLB 设备，再由 SLB 设备

进行向上向下转发。SLB 实际上承担了 NAT ( Network Address Translation, 网络地址转换) 功能, 向用户屏蔽了 Cache 设备的 IP 地址。这种方式是 CDN 系统中应用较多的方式, 优点是具有较高的安全性和可靠性, 缺点是 L4-7 交换机通常较为昂贵。另外, 当节点容量大时, L4-7 交换机容易形成性能瓶颈。不过近年来, 随着 LVS 等技术的兴起, SLB 设备价格有了大幅下降。

在旁路方式下, 有两种 SLB 实现方式。在早期, 这种 SLB 一般由软件实现。SLB 和 Cache 设备都具有公共的 IP 地址, SLB 和 Cache 构成并联关系。用户需要先访问 SLB 设备, 然后再以重定向的方式访问特定的 Cache。这种实现方式简单灵活, 扩展性好, 缺点是安全性较差, 而且需要依赖于应用层重定向。随着技术的发展, L4-7 交换机也可采用旁路部署方式, 旁挂在路由交换设备上, 数据流量通过三角传输方式进行。关于 SLB 的实现方式, 在本书第 4 章还将深入讲解。

在 CDN 系统中, 不仅分发服务系统和调度控制系统是分布式部署的, 运营管理系统也是分级分布式部署的, 每个节点都是运营管理数据的生成点和采集点, 通过日志和网管代理等方式上报数据。可以说, CDN 本身就是一个大型的具有中央控制能力的分布式服务系统。

## 2.2 CDN 系统分类

在第 1 章中已经讲到, CDN 的发展与互联网的发展相辅相成, 互为推手。从技术演进过程来看, 互联网应用的每一次突破都要求 CDN 技术产生与之相适应的发展变革, 因而 CDN 加速服务技术经历了从静态网页到动态网页, 再到流媒体和今天的云计算这样的演变和拓展过程; 而 CDN 技术的发展反过来也帮助互联网提高网站访问速度、带给用户更好的服务和上网体验, 促进互联网生成更多更新的应用形态。二者的相互促进使 CDN 逐步成为互联网的一项

重要的基础性服务，同时也不断产生出新的产品和服务类型。目前，从主流的 CDN 运营商来看，至少都可提供十几种到二十多种基础服务和产品，令人眼花缭乱。不过从技术角度分析，我们可以归纳出一些基本类型的 CDN 系统和服务，其他产品和服务都是从这些基本的服务类型衍生出来的。

可以从两个角度来对 CDN 基本服务进行分类，一是基于不同内容承载类型视角，二是基于不同内容生成机制视角。

### 2.2.1 基于不同内容承载类型的分类

从 CDN 承载的内容类型来看，主要有静态网页内容、动态网页内容、流媒体、下载型文件和应用协议，因而我们将 CDN 服务分为网页加速、流媒体加速、文件传输加速和应用协议加速。

#### 1. 网页加速

网页加速是最早出现的 CDN 服务类型，伴随着第一次互联网浪潮，大量网站涌现，而当时用户以窄带接入为主，网页的内容也主要以文字、图片、动画等形式为主，支持文本方式的电子邮件交换，因此 CDN 技术最初的应用重点就是用来对这些网页的静态内容进行加速。CDN 服务商通过将网页内容缓存到各个 CDN 节点上，并将用户请求调度到最优节点上来获得所需的内容，从而加速页面响应速度，减轻源站点的访问负担。这种网页加速服务主要面向各类门户网站、新闻发布类网站、访问量较大的行业网站、政府机构网站和企业门户网站等。随着 Web 2.0 的兴起和互联网应用的丰富，网页加速也逐渐从静态内容加速向动态内容加速扩展，支持股票行情、电子商务、在线游戏等网站的动态内容加速。

#### 2. 流媒体加速

从 2002 年开始，以 ADSL 为主的宽带接入技术被运营商普遍采用，用户

的主流接入带宽提升到 1MB 或 2MB，这一带宽水平已经能够支持网络视频业务，因此大量视频网站涌现，流媒体流量迅速跃升为互联网流量主力军。CDN 技术的应用重点逐步转向流媒体加速，关注视频文件的全网缓存、调度，用户播放器动作响应等。

流媒体加速的实现是通过将流媒体内容推送到离用户最近的 POP 点，使得用户能够从网络边缘获取内容，从而提高视频传输质量，缩短访问时间，节省骨干网络流量，避免单一中心的服务器瓶颈问题。流媒体加速服务又可以分为两类：

- 流媒体直播加速。直播（Live）与电视台或电台现在追求实效的现场直播方式一样，电视台或电台正在播放的节目或现场实时制作的直播节目可以在互联网上以流媒体的方式同步传输。网络电视曾经是一项非常受欢迎的服务，P2P 技术也是在流媒体直播业务的催生下红极一时。
- 流媒体点播加速。流媒体点播（On-demand）是流媒体在互联网上播放的另一种方式，它将流媒体以内容类别、版本等为索引按片段存放在服务器上，用户根据需要或感兴趣的内容选择播放。它与直播方式的最大区别在于，用户不受电视台播放节目时间和内容的限制，在自己合适的时间观看自己想看的内容。

### 3. 文件传输加速

文件传输加速服务一直是一项重要的 CDN 服务，通过使用 CDN 的分布式 POP 点提供下载服务，网站可以将大量文件下载的性能压力和带宽压力交给 CDN 来分担，提高用户的下载速度。目前 CDN 技术可以支持 HTTP 下载、FTP 下载和 P2P 下载等各种下载方式，主要用于软件厂商的补丁发布、杀毒软件厂商病毒库更新、网络游戏运营商的游戏客户端下载以及其他提供文件下载服务的网站，比如音乐网站等。

另外，目前一些领先的云计算服务提供商，如 Amazon、微软等在向用户提供云主机、云存储等服务时，也同时推出了 CDN 文件传输加速服务，主要是对在云中托管的站点或内容进行传输加速。

### 4. 应用协议加速

应用协议加速并不针对特定的内容类型进行加速，而是通过对 TCP/IP 等传输协议的优化，改善和加快用户在广域网上的内容传输速度，或者对一些特定协议，如 SSL 协议进行加速，解决安全传输时的性能和响应速度问题。主要的应用协议加速服务有如下几种。

- 广域网应用加速。其目的是“让广域网像局域网一样”，这实际上有两方面的意思：1) 使广域网的性能产生质的飞跃，尤其针对那些在局域网上可以正常运行，但一到广域网就受到极大影响的应用和协议，比如 CIFS 协议、NFS 协议。2) 能处理多种分布式企业网络环境下的各种应用和协议。广域网应用加速的目的是在不改变远程用户使用习惯的前提下，将分布式的 IT 基础设施如文件服务器、邮件服务器、网络附加存储 (NAS) 和远程办公室备份系统等集中起来，整合到统一的数据中心。这样，让企业位于世界各地的同事共享大型文件变得简单而高效，使他们感觉就好像在同一建筑里办公一样。广域网应用加速还支持通过长距离广域网链路进行文件备份与复制操作，在不升级带宽的前提下在现有的广域网上提供比以前丰富得多的应用服务。
- SSL 应用加速。许多基于网络的重要核心应用都采用了 SSL 技术来保证服务的安全性和私密性。由于需要进行大量的加密解密运算，SSL 应用对服务器端的资源消耗是非常巨大的。CDN 提供 SSL 应用加速后，由 CDN 的专用 SSL 加速硬件来完成加密解密运算工作，通过认证之后方可建立起数据传输通道。用户的源站点只需信任有限的 CDN cache，而无须面对海量用户，从而减轻了繁重的运算和认证压力。

- 网页压缩。现在的网页中含有大量的 Flash、图片等内容，文件体积比以前大得多，因此在 HTTP 1.1 协议中提出了对网页压缩功能的支持。在服务器端可以先对网页数据进行压缩，然后将压缩后的文件提供给访问用户，最后在用户的浏览器端解压显示。通过这种方式可以减少数据传输的时间，加快页面显示速度。CDN 利用这种网页压缩技术，为网站提供网页内容的压缩传输，从而加快内容传送速度。

应用协议加速服务的出现和逐步发展完善体现了 CDN 演进的第三阶段：多应用协议加速。这一阶段是从 2009 年开始的，一方面用户接入带宽向 4MB 或更高水平发展，另一方面，运营商对 3G 业务的大规模部署极大刺激了移动互联网的发展。移动互联网业务呈现新的特征，终端类型多样化，这使得 CDN 需要适应协议的不断增加和变化，以传统方式提供加速业务已经无法满足要求。因此，CDN 技术出现了应用和基础传输功能相分离的发展趋势，比如 HTTP Streaming 协议适配服务、云服务提供商（Amazon、微软等）提供的块传输服务等。

## 2.2.2 基于内容生成机制的分类和分层加速服务

从内容的生成机制来看，互联网上的内容主要有两类：一是静态内容，二是动态内容。其中，静态内容主要是指内容完全由网页 HTML 文件提供，任何人在任何时间浏览静态内容看到的都是一样的东西。而动态内容是指不同的访问者或在不同时间访问同一个 Web 页面时可能得到不同的页面内容，内容具有实时性，访问的过程具有交互性。因此，动态内容的提供除了和静态内容一样需要网页外，还需要采用数据管理系统和业务逻辑程序来使网站具有更多自动的和高级的功能。

主流的 Web 网站系统都能够在逻辑上划分为三个层次，即表现层、业务逻辑层和数据访问层，如图 2-5 所示，不同的层次在系统中有不同的功用。

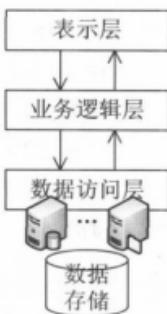


图 2-5 主流 Web 系统分层架构

表现层是以 Web 方式为用户提供访问界面，主要负责接收用户的请求以及业务逻辑层处理结果的返回及展示；业务逻辑层主要是针对具体业务逻辑的处理，能够根据表现层传来的用户需求向数据访问层发出数据查询要求并将查询结果进行相应的整合，返回给表现层向用户展现；数据访问层主要是以数据库、文件系统等方式对原始数据进行保存和管理，并为业务逻辑层或表示层提供数据查询服务。

在一个 Web 系统中，不同类型的内容由不同的系统层保存和提供。通常与用户直接交互的表现层提供大部分静态内容，例如图片、Flash 动画、多媒体文件以及部分静态网页片段等。而动态内容，就需要由业务逻辑层和数据访问层协作提供。其中，业务逻辑层是 Web 系统的核心层，负责处理所有的业务逻辑并生成动态内容。

了解了 Web 系统的分层架构以后，我们再回过头来看 CDN 的分类。CDN 实现网页内容加速主要依赖于内容边缘缓存和功能复制两类机制，本质就是将 Web 源站各个层次上的功能转移到 CDN 边缘 Cache 上完成。根据 CDN 完成的不同层面的 Web 功能转移，将 CDN 分为表示层复制和全站复制两大类。

对于 Web 网站提供的各种类型的静态内容（不论是网页、文件还是流媒体

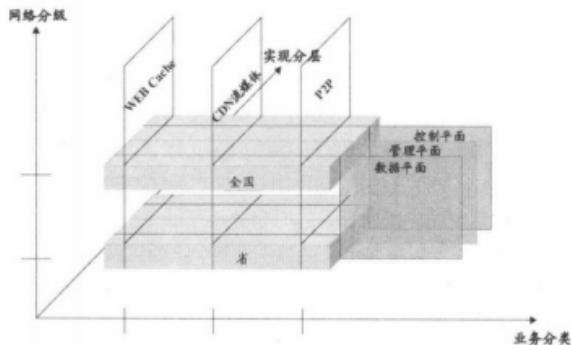
数据），其加速都可以通过在边缘 Cache 上复制 Web 系统的表示层来完成。在实现中，CDN 的 Cache 设备将以反向代理的角色接受用户发来的连接请求，然后在本地复制的数据表示层的静态数据中寻找满足用户需求的数据，直接反馈给用户。在 Cache 上命中的内容，则无须再向源站 Web 系统请求。这种情况下，Cache 上缓存的内容通常是完整的 Web 内容实体，例如网页嵌入内容、多媒体文件等。现在大多数商用 CDN 系统采用的都是这类只处理静态内容请求的网站加速方案。

对于当前日益丰富的动态内容加速，需要在 CDN 上复制和缓存业务逻辑层和后台数据访问层。其中，业务逻辑层在 CDN Cache 上的复制使之能够承担用户请求处理、应用数据计算、动态内容生成等工作，因此这类方法也被称为“边缘计算”。将 Web 应用程序或应用组件直接安装在 CDN Cache 中，目的是在最接近用户的位置完成应用处理，同时也分担了源站的计算压力。在某些应用场景中，动态内容的生成需要大量的数据支持，比如目录服务、交易数据等，仅仅将业务逻辑复制到边缘服务器中还不足以解决从源站获取其生成动态内容所需的数据而造成的传输性能瓶颈，因此还需要对数据访问层进行必要的复制，即除了在 Cache 上完成业务逻辑的运算工作，还复制了源站后台数据访问层的内容，用以加速动态内容的生成。这个过程的关键在于合理解决系统中多个数据副本间的一致性问题。另外，还有一些网站的动态内容是基于具体用户的个性化数据定制生成的，需要在数据访问层对用户数据进行特别的关注。用户数据本身也是依托于数据访问层的存储介质和管理系统存在的，但在访问模式等方面具有独特性，因此 CDN 需要制订相应的复制和缓存策略，并解决相关的隐私和安全问题。也许这样的讲解仍然有些令人困惑，没关系，在本书的第 7 章还将深入讲解动态内容的生成和加速技术。

### 2.3 小结

从本章的分析可以看出，CDN 系统从不同的角度去看，有不同的分解方式。

我们总结了一个 CDN 系统的三维模型，从逻辑上将 CDN 系统分成三个维度，希望能够帮助读者更全面地理解 CDN 系统。如图 2-6 所示，其中水平 X 轴方向按照不同类型业务能力进行划分，垂直 Y 轴方向按照中心、区域和边缘三级网络结构进行划分（根据网络规模情况，区域一级可选）；Z 轴方向按照数据平面、管理平面和控制平面进行划分。



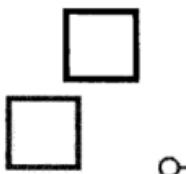
之所以在这个模型中按照立体空间的三个维度进行划分，是因为在整个系统中各个功能模块、控制机制之间相互交叉，从单一维度很难阐述清楚。

X 轴方向是贯穿系统头端到终端的垂直系统，每一种业务能力都可以有自己的中心、区域、边缘层设备，有自己独立的管理系统、负载均衡系统、分发服务系统。因此，在 X 轴方向上，每一个 CDN 服务能力都可以是一套相对独立的子系统。

Y 轴方向的分级划分是依据 CDN 逐级缓存、分级调度、分级服务的要求。CDN 的管理功能、调度功能、缓存功能、服务功能都是分级部署的，不同层级的相同功能实体之间相互配合。从控制功能上看，往往下一级控制实体是上一级控制实体的执行者；从缓存功能上看，上一级缓存是下一级缓存实体的内容

提供者；从服务功能上看，上一级服务实体是下一级服务实体的能力补充和备份。

Z轴方向上，管理层主要完成CDN网络管理和业务逻辑处理。网络管理提供IP网络层面的配置管理、故障管理、网络性能管理以及网络安全管理等，负责对整个CDN系统的网络和设备资源进行抽象，提取其业务能力，提供给上层业务平台。业务逻辑处理指CDN系统辅助其服务的业务平台完成部分应用层功能的实现，完成和其他系统之间的接口适配，负责系统的认证、计费、统计分析、系统管理、用户管理、SP管理等功能。控制平面负责对整个CDN网络的内容分布情况进行管理，对内容分发和访问路由策略进行控制，它是整个内容分发系统内容数据流向的控制点。数据平面承担实际的内容分发工作，根据Z轴方向与X轴、Y轴相交点的具体要求，数据平面设备具备不同的应用服务能力。



## 第3章 内容缓存工作 原理及实现

- 3.1 内容缓存技术的发展背景
- 3.2 Cache 设备的工作方式和设计要求
- 3.3 Web Cache 的实现基础——基于 HTTP 协议的 Web 缓存技术
- 3.4 Web Cache 技术实现关键点分析
- 3.5 开源 Web 缓存代理软件——Squid



缓存（Cache）是 CDN 的基础技术和组成单元，从外部看，整个 CDN 可以看成一个大的 Cache 向外提供服务。因此，了解 CDN 首先要从学习 Cache 开始，本章以 Web Cache 为例，对 Cache 的技术背景、工作原理、设计方法进行分析，并以开源的 Squid 进行实例分析。

## 3.1 内容缓存技术的发展背景

在互联网发展初期，网站的服务能力要求都不高，很多网站一台服务器就可满足需求。但随着网络内容和用户数量的增长，这些网站很快就面临服务瓶颈，最初的解决办法是提高服务器的硬件配置，后来又出现了多台服务器组成集群进行负载分担（详见第 4.1 节）的方法。但这些都无法解决网络延迟、“最后一公里”等网络问题，为此出现了内容缓存技术，并进一步发展成为内容分发服务技术。

### 3.1.1 网站的问题和需求

互联网最常见的访问方式是 B/S 模式，即浏览器（Browser）到网站服务器（Server）的访问模式，最简单的形式就是用用户的浏览器去访问远在异地的一个网站服务器，从服务器上获取你想要的信息。下面通过一个例子分析这种最简单的 B/S 模式将会出现哪些问题，从而催生了 CDN 服务的出现。

假设你觉得互联网是一个非常好的营销渠道，想要建立一个电子商务网站，为市场上知名的各个商家提供他们产品的销售服务，也为有各种各样商品需求的全世界用户提供浏览和购买商品的途径。那么，你为商家设置的电子店铺中会包含图片、文字、视频介绍以及相关的软件下载，用户可以在你的网站上通过浏览这些内容了解和体验他们所需要的商品。

起初，你认为你的网站就是为北京市内 10000 人左右的商家服务，所以购买了 10 台服务器，在电信运营商的北京 IDC 机房里租用了 10 台服务器的空间和 1GB 的带宽，然后网站运营开始。三个月后，由于你广告宣传的投入加大，货源渠道丰富，售后服务周到，网站购物体验好，用户已经超过 10000 人。

当越来越多的用户来到你的网站，他们会发现过去访问很顺利的网页很难打开或者出现访问错误，视频点播断断续续、商品大图无法打开，经过几次很差的体验后，用户开始慢慢不再喜欢访问这个网站，访问人数开始下降，所以你决定再增加 10 台服务器，再租用更多的带宽以满足北京用户的访问需求。

如果网站的用户黏性很高，你开始准备在全国进一步扩大商品销售，让离北京较远的用户（比如广州）也能享受商品订购和快递服务，你选择在北京机房继续增加服务器和带宽来满足要求。结果效果很差，网站服务器虽然并没有达到最大负荷，但用户在观看视频时经常断断续续或出现马赛克，网页显示很慢并且也经常出现错误。主要原因是用户访问的网站内容在从北京传到用户所在地的较远路途中，必然会在路由设备上排队等待，如果骨干网络极度拥塞，访问内容的时延以及丢失的数据包数量会非常大。

由于远处用户访问体验较差，你决定在用户访问较多的其他几个省份部署一套镜像系统，经过分析后，在广州和长沙各部署了 10 台服务器，租用了 1GB 的带宽。广州和长沙的用户通过选择相对应的服务器，能够得到满意的体验。但接下来你又遇到了以下新问题：

- 假设某款需要抢购的商品在你的网站上限量发售，北京服务器网页的更新是在凌晨 5 点，而你的网站通常是在每天晚上 12 点进行镜像间的内容同步，所以商品的网页需要 19 个小时后才能在广州和长沙的镜像上出现。在这逝去的 19 个小时中，北京的用户已将所有商品购买一空。
- 假设网站的广州镜像出现了故障，但用户可能在访问时并没有意识到

这一点，仍然会继续将请求发送到广州的镜像，或者在确定广州镜像出故障的情况下直接访问北京服务器，而不会智能地将请求转移到距离较近的长沙镜像。

- 北京服务器是系统的中心服务器，但用户和黑客在访问时都能够直接获取其 IP 地址，所以黑客通过直接向中心服务器发起 DDoS 攻击就可以轻而易举地攻破服务器。
- 随着电子商务的服务范围在全国不断扩大，你需要在全国的各个地方都购买服务器和相应带宽，并提供与这些资产相对应的维护人力，整个成本相对于收益并没有因为经济规模效应而大幅度减少，所以你的网站看上去赢利能力在不断削弱。

现在，我们回过头总结一下这个网站从一开始到现在所遇到的问题：

- 无法及时满足并发用户增长的需要。
- 在没有设置镜像前，无法满足较远用户的访问需要。
- 在设置镜像后，中心服务器与镜像之间信息同步不够及时。
- 很多情况下，一个镜像站点失效后，无法及时高效地调度到最近的镜像来服务。
- 通过部署镜像来解决问题会明显增加成本。
- 中心服务器 IP 地址的暴露导致易受黑客 DDoS 攻击。

那么，CDN 能否帮助网站解决这些问题呢？

### 3.1.2 CDN 出现前的网站服务技术

虽然上面提出的一系列问题正是 CDN 致力于解决的问题，但在 CDN 技

术形成商用能力之前，网站运营者们一直在不断尝试其他网站加速技术。

### 1. 扩展技术 Scale up/Scale out

**Scale up**，是指提高网站服务器的硬件水平。这是最简单、最直接的方法，比如增加高速处理器，配置更大的内存和硬盘，或是配置多处理器系统。用户依然是直接访问源站服务器，只是单台服务器能够提供服务的用户数增加了。这对于解决远距离传输带来的质量问题无效的，而且往往需要同时对整个系统进行硬件升级，灵活性和可扩展性都比较差。

**Scale out**，采用服务器集群。随着网站用户数的不断增长，网站的单台普通服务器再也满足不了大规模用户并发处理的要求，这时就产生了集群方案。一个集群由很多台服务器组成，以负载分担的方式处理同一个站点的用户请求。集群内需要引入负载均衡设备，根据用户请求的类型、请求内容、用户名称等，将请求分配到某台具体服务器上进行响应。

### 2. 镜像技术 Mirroring

镜像是冗余的一种类型，比如一个磁盘上的数据在另一个磁盘上存在一个完全相同的副本就称为磁盘镜像。镜像主要用于备份，在数字媒体服务和分发领域，是指将数字媒体的服务能力和完整内容都备份到网络上不同地址的另一个地方。

基于镜像技术的主要应用方式是镜像网站，即对整个网站中的内容进行镜像复制，并对镜像网站多点部署。这样，用户在访问网站时可以自主选择速度较快的镜像站点，从而得到更快更好的服务，降低主网站/原始网站的负载。例如，Web 或 HTTP 下载服务器在不同城市或针对不同运营商的 IDC 可以进行镜像处理，加速分发。

部署镜像站点是一段时间里被采用比较多的网站加速方法，至今一些网站仍在使用。镜像服务器上安装有一个可以进行自动远程备份的软件，每隔一定

的时间，各个镜像服务器就会到网站的源服务器上去获取最新的内容。这种方法常用来解决源站服务器和用户在不同运营商网络中的问题。镜像站点对主站点起到了用户分流作用和应急备份作用。不过，用户自行选择时往往带有一定的盲目性，有时并不能起到就近服务的作用。另外，对于镜像站点来说，每个镜像都是源站的百分之百复制，所以对体积庞大的网站来说，部署多个镜像站点成本是非常高的。

### 3. 缓存技术 Cache

缓存是将访问过的数字媒体存储起来，为后续的重复访问使用。如用户 A 通过缓存设备访问了一个由网站 [www.streamabc.com](http://www.streamabc.com) 提供的流媒体文件 sample.avi，当用户 B 要访问 sample.avi 时就不用再访问 [www.streamabc.com](http://www.streamabc.com) 了，缓存设备可以直接将已缓存的文件 sample.avi 发送给用户 B。基于缓存技术的主要应用方式有缓存代理和CDN。

缓存代理服务一定范围的访问域，访问域内的内容访问请求通过缓存代理执行。缓存代理缓存被访问过的内容，后续的相同内容访问直接通过缓存代理获得服务。缓存代理的一种复杂实现形式是部署分层缓存，就是在不同的物理位置部署多台缓存服务器，逻辑上采用分层模式，上层缓存作为下层缓存的内容存储器，保证及时向下一级缓存提供所需的内容。直接向用户提供服务的服务器只需向上一层缓存服务器请求内容，而非向源站请求。这样，获得内容的时间更短，同时也分担了源站服务器的压力。

CDN 通过将内容存储到离用户最近的地方实现面向大规模用户的就近服务。从某种意义上可以说，CDN 是在缓存技术的基础上发展起来的，或者说 CDN 是缓存的分布式集群实现。负载均衡系统和内容管理系统都是为了更好地实现各个缓存节点之间的协同工作。以上提到的分层缓存方案在原理上已经非常接近 CDN，所以 CDN 也可以理解为智能调度加上分层缓存技术的组合。

## 3.2 Cache 设备的工作方式和设计要求

前面提到了内容缓存技术，它是通过对内容副本进行缓存来满足后续相同的用户请求，是整个 CDN 系统实现的基础。在实际的网络部署应用中，内容缓存工作是通过 Cache 设备来实现的。下面，就让我们看看 Cache 设备是如何工作的，一个 Cache 设备的设计主要有哪些方面的要求？

通常来说，根据缓存内容的不同可以将 Cache 设备分为 Web Cache 和流媒体 Cache 两大类。Web Cache 设备主要用于缓存普通网页的内容和对象，同时大多数设备也具备文件下载、流媒体服务等能力；流媒体 Cache 设备主要是针对视频流媒体服务进行加速，功能相对单一。目前，大多数设备厂商在研发 Cache 设备时，不会对主要用于流媒体的 Cache 设备和主要用于网页缓存的 Cache 设备进行产品区分，但对于 CDN 服务运营提供商，由于在缓存的内容类型、用户行为、内容更新等方面有较大的差异，对性能的优化技术上需要采用不同的技术路线和实现方式，基本上都会将 Web Cache 和流媒体 Cache 作为两个相对独立的产品线进行开发和优化。

Web Cache 设备是最早出现的 Cache 设备类型，历经多年发展，目前已比较成熟，且有多种使用方式，因此本章将以 Web Cache 为例，分析和说明内容缓存技术的工作原理，为进一步了解整个 CDN 系统奠定基础。关于流媒体 Cache 的相关技术内容，读者可在第 6 章看到。

目前，市场上的 Web Cache 设备种类较多，一款 Web Cache 产品在投放市场之前需要考虑较多的市场因素，比如需要应用的场景、面临的客户群。因此，不同的市场需求对 Web Cache 的基本功能、性能、管控以及工作方式都有着不同的要求。

Web Cache 作为一种网页缓存技术，可以在用户访问网站服务器的任何一个中间网元上实现。根据 HTTP 协议的定义，在一次网页访问中，用户从客户

端发出请求到网站服务器响应请求内容的交互过程中，通常会涉及 4 个关键的网元：用户、代理、网关和 Web 服务器。其中，在用户端实现缓存技术在 HTTP 协议中有明确的规定，比如用户浏览器可以缓存已经访问过的网页，用户再次访问时可以直接访问这个网页副本。根据不同的应用场景和用户需求，Web Cache 通常作为代理或网关部署在用户的访问路径上，部署的位置不同、工作模式不同，对 Web Cache 有不同的要求。当 Web Cache 作为代理使用时，通常工作在正向代理或者透明代理的模式，Web Cache 可以在这两种模式下实现访问内容副本的缓存和服务；Web Cache 应用最多的地方还是在网关上，这也是 CDN 的典型应用场景，网关通常工作在反向代理模式。下面，我们对正向代理、反向代理和透明代理的工作方式进行介绍。

### 3.2.1 正向代理

正向代理（Forward Proxy）方式下，使用者需要配置其网络访问的代理服务器地址为 Cache 设备的地址，内网用户对互联网的所有访问都通过代理服务器代理完成。使用者也可以仅对特殊应用设置代理服务器，此时仅该类访问需要通过代理服务器代理完成。通常正向代理的缓存设备支持冗余配置，从而保证代理系统的稳定性和可用性。正向代理的工作原理示意图如图 3-1 所示。

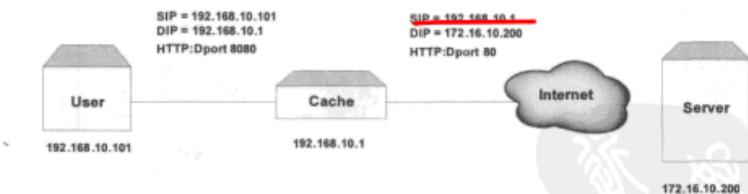


图 3-1 正向代理工作原理示意图

图 3-1 演示了一个正向代理的应用例子。用户主机和代理服务器部署在同一网络环境中，用户主机地址为 192.168.10.101，正向代理服务器的地址为

192.168.10.1，用户想要访问的外网服务器地址为 172.16.10.200。通常用户需要为所使用的主机配置正向代理服务器地址（192.168.10.1）和服务端口（8080），之后用户在上网时其主机对外网服务器的数据传输首先要传输给正向代理服务器，代理服务器检查代理缓存中是否保存了用户请求数据，如果有则直接返回给用户，如果没有缓存请求内容，则正向代理服务器负责发送用户主机请求数据到外网目标服务器，同时接收并缓存外网服务器响应数据，同时将响应数据反馈给用户主机。在执行正向代理功能时也可以完成安全认证和访问控制功能，比如可以设置某些特定用户在工作时间访问外网站点，或者禁止访问某些外部站点等。

正向代理多用于中小企业网络环境，Cache 设备作为企业网的出口网关提供代理服务、内容缓存、Internet 访问控制、安全认证等功能。在正向代理模式下，Cache 设备可以为企业网节省出口带宽，提高企业内部网络的安全性，防止员工滥用网络资源并在一定程度上防御病毒感染。

图 3-2 所示为正向代理部署示例，正向代理服务器部署在企业内部网络环境中，内部用户在访问外部网络时需要配置正向代理服务。

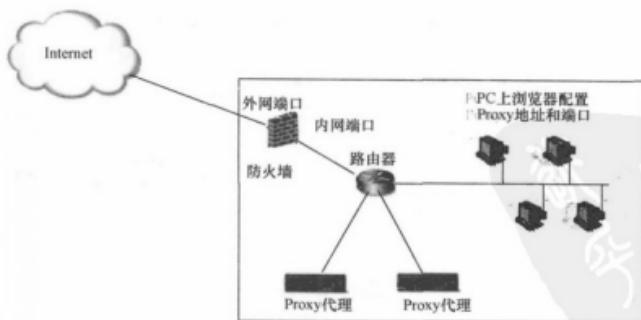


图 3-2 正向代理部署示例

### 3.2.2 反向代理

在反向代理（Backward Proxy）方式中，用户不需要配置代理服务器地址，Cache设备的地址作为被访问域的服务地址写入DNS记录，并利用Cache设备的内容路由/交换能力完成代理访问。反向代理和其他代理方式的区别是，反向代理专门对定制的内容进行加速，如域名 streamabc.com 之中的所有网页内容或域名 streamcde.com 之中的所有流媒体内容。

反向代理多用于大型ISP/ICP和运营商环境，对于运营商和ISP，反向代理可以实施透明的内容缓存，增加用户访问内容的速度和提高客户满意度。反向代理的优势在于对于用户来说，不会感觉到任何Cache设备的存在。同时，反向代理的部署方式有很多变化，ISP可以通过在服务器集群前部署简单的内容交换设备实现。在反向代理方式下，当Cache数量较多，网络规模较大时，需要部署GSLB（Global Server Load Balancing）来对全网中的Cache进行负载均衡，并对全网的内容分发策略进行设定，就形成了CDN网络的雏形。

图3-3演示了一个反向代理的应用实例。代理服务器（Cache）和应用服务器（Server）部署在同一网络环境中，用户主机地址为192.168.10.101，应用服务器地址为172.16.10.200，反向代理服务器地址为172.16.10.1，应用服务器对外访问地址为反向代理服务器地址172.16.10.1，用户直接访问代理服务器获取应用服务器提供的服务，而不需要配置任何代理服务。大致流程为，用户首先发送数据请求到外网的反向代理服务器，代理服务器检查代理缓存中是否保存了用户请求的数据，如果有则直接返回给用户，如果没有缓存请求的内容，则反向代理服务器将用户主机请求数据发送给应用服务器，同时接收应用服务器响应数据并反馈给用户主机，同时缓存用户请求相关内容。在执行反向代理功能时，代理服务器响应了大部分应用访问请求，大大减轻了应用服务器的负载压力。

## CDN 技术详解

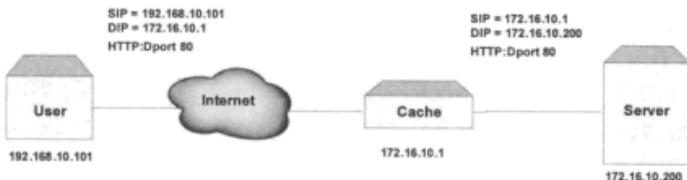


图 3-3 反向代理工作原理示意图

图 3-4 所示为反向代理部署示例，反向代理服务器部署与应用服务器部署在同一网络环境中，应用服务器借助反向代理对外提供服务，反向代理提供负载分担和安全隔离作用。

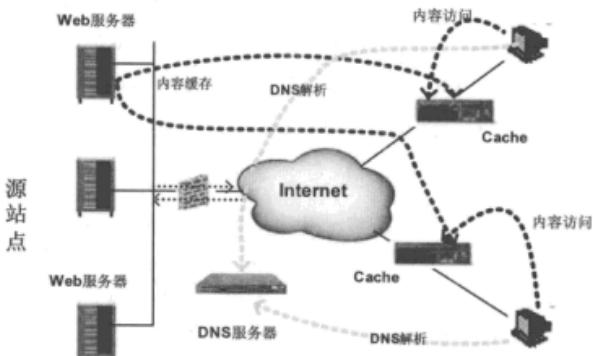


图 3-4 反向代理部署示例

### 3.2.3 透明代理

透明代理（Transparent Proxy）方式下，用户的浏览器不需要配置代理服务器地址，但是用户的路由设备需要支持 WCCP 协议（Web Cache Control Protocol）。路由器配置了 WCCP 功能后，会把指定的用户流量转发给 Cache，

由 Cache 对用户提供服务。另一种方案是利用 4 层交换机将用户的流量转发给 Cache，由 Cache 对用户提供服务。使用 WCCP 或 4 层交换都可以支持负载均衡，可以对多台 Cache 平均分配流量。这样可以扩展网络规模，支持大量的用户访问。用户不会感觉到任何 Cache 的存在。

透明代理可以看做是通过网络设备或协议实现的正向代理工作模式，因而具备很多与正向代理相同的特点，多用于企业网环境和运营商环境。Cache 设备作为企业网的 Internet 网关出口提供代理服务、内容缓存、Internet 访问控制、安全认证等功能。在透明代理模式下，Cache 设备可以为企业网节省出口带宽，提高企业内部网络安全，防止员工滥用网络资源并在一定程度上防御病毒感染。图 3-5 演示了一个透明代理的工作原理。

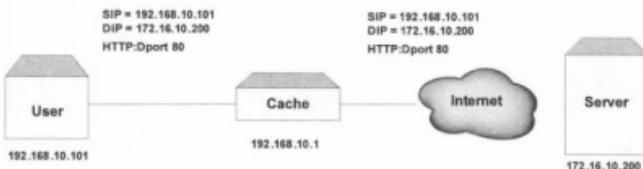


图 3-5 透明代理工作原理示意图

与正向代理部署方式类似，用户主机和代理服务器部署在同一网络环境中，用户主机地址为 192.168.10.101，正向代理服务器的地址为 192.168.10.1，目标应用服务器地址为 172.16.10.200。用户访问目标服务器时不需要配置任何代理服务，直接将服务请求的目标地址设置为应用服务器 IP 地址，用户主机请求数据在发往目标主机前被透明代理截获，透明代理检查代理缓存中是否保存了用户请求数据，如果有则直接返回给用户，如果没有缓存请求内容，透明代理服务器则将用户主机请求数据发送到目标服务器，同时监听外网服务器响应用户请求数据，用户主机保持相关数据在缓存中以便后期服务网内相同的访问请求。由于透明代理可以截获用户访问数据，在提供缓存功能的同时也可以完成与正向代理相同的安全认证和访问控制功能。

图 3-6 所示为透明代理部署示例，透明代理服务器部署在企业内部网络环境中，与正向代理不同，内部用户在访问外部网络时不需要配置任何代理服务。

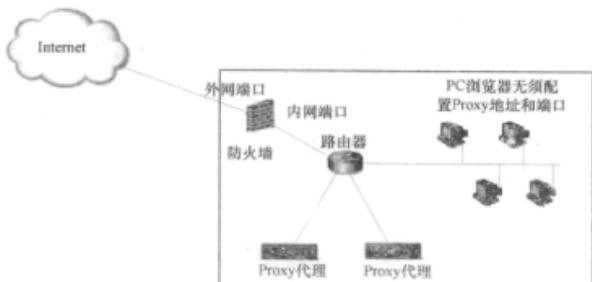


图 3-6 透明代理部署示例

### 3.2.4 Web Cache 产品实现关键要素分析

开发一款 Web Cache 设备需要考虑多方面的因素。

首先，Web Cache 的技术本质是缓存，在面向用户一侧时扮演服务器的角色，在面向网站的源服务器时扮演客户端的角色。因为 Web Cache 的介入，原始的客户端直接访问源服务器的访问过程就变成了从客户端到 Web Cache，再从 Web Cache 到源服务器的两个访问过程，所以原有的点到点的单次 HTTP 协议过程也变成了多次 HTTP 协议过程，因此 Web Cache 应具备 HTTP 协议所描述的基本功能。在 3.3 节将详细介绍关于 HTTP 协议的背景知识，描述协议中的关键消息和相关机制，特别是与缓存技术相关的机制。

其次，在实现最基本的 HTTP 协议能力和缓存工作方式的基础上，要根据应用场景和工作模式确定 Web Cache 应具备的功能，并明确应达到的性能指标。从前文可以看出，Web Cache 设备在不同的应用场景和工作模式下对性能的要求不同，比如面向企业用户时，Web Cache 主要工作在正向代理模式或者透明代理模式下，在根据企业内部用户数量来决定并发连接数量、并发流量大小、

存储空间大小以及缓存的大小的同时，也要考虑 Web Cache 对数据的分析、过滤以及安全保障能力，包括 CPU 处理能力在内的处理性能指标显得尤其重要；如果是面向 Web 站点的 Web Cache，为给大量互联网用户提供较好的加速服务，新增连接处理能力、并发连接处理能力、存储空间和缓存大小都是需要重点考虑的性能指标；与面向 Web 站点的 Web Cache 类似，面向运营商的 Web Cache 需要考虑包括新增处理能力、并发处理能力等在内的性能指标，但是对指标的要求将会更高，以达到运营商所要求的服务能力。在 3.4 节将重点介绍设计 Web Cache 设备时需要考虑的各项功能要求和性能指标，包括工作模式、内容更新机制、存储管理机制以及安全保障机制的描述。

### 3.3 Web Cache 的实现基础 ——基于 HTTP 协议的 Web 缓存技术

下面进入本章的主题——Web 缓存技术。在介绍 Web 缓存工作原理之前，我们首先对 Web 的工作机制和特点进行介绍，然后重点对 HTTP 协议的工作原理进行剖析，基于此再讲解具体的缓存技术。

#### 3.3.1 Web 与 HTTP

在本书第 1 章中我们曾提到，现在通常说的互联网是广义互联网，由网络层和万维网为代表的应用层共同组成。TCP/IP 奠定了互联网发展的基石，而互联网的蓬勃发展和被广大用户所接受是从万维网的诞生开始的。在 Web 出现以前，互联网上的信息只有文本形式。人们在进行信息检索的时候，不容易识别，而且索然无味。而 Web 有一个特点，即 Web 上的信息除了文本形式以外，还具有图形化和易于导航的特点。现在的 Web 可以实现将图形、音频、视频信息集合于一体的能力。同时，Web 的导航属性对于用户访问来说也是很便捷的，

用户只需从一个链接跳转到另一个链接，就可以在各站点之间进行浏览了。内容极大丰富，使用方便快捷、简单易懂是 Web 应用能够非常流行的很重要的原因。

Web 的另外一个优势是它的跨平台特性。在没有 Web 的时候，访问互联网也要考虑到不同系统平台的差别，有时会因为平台的不一致而无法浏览信息。而在 Web 上，无论你自己的系统平台是什么，都可以通过 Web 访问各种网页信息。对 Web 的访问是通过浏览器这样一个客户端软件实现的，有很多不同种类的浏览器可供使用，比如 NCSA 的 Mosaic、Microsoft 的 Internet Explorer。

此外，Web 具有良好的交互性。首先表现在它的超链接上，用户的浏览顺序和所到站点完全由他自己决定。其次，通过表单的形式可以从服务器获得动态的信息。用户通过填写表单（比如认证信息）向服务器提交请求，服务器可以根据包含有用户特定信息的请求返回响应消息。现在的 Web 已经从 1.0 升级到了 2.0，个人不再是作为被动的客体而是作为一种主体参与到了互联网中，个人在作为互联网的使用者之外，还同时成为了互联网主动的传播者、作者和生产者。所以个性化和社区的概念成为了互联网访问的核心，用户与网站的互动以及用户与用户之间的互动变得更加频繁。

从技术层面看，Web 架构的精华有三处：用超文本技术（HTML）实现信息与信息的连接；用统一资源标志符（URI）实现全球信息的精确定位；用应用层协议（HTTP）实现分布式的信息共享。

- HTML，全称是超文本标记语言（Hypertext Markup Language），是一种用于描述 Web 页面的标记语言，它通过标记符号来标记要显示的网页中的各个部分。网页文件本身是一种文本文件，通过在文本文件中添加标记符，就可以告诉浏览器如何显示其中的内容（比如文字如何处理、画面如何安排、图片如何显示等）。浏览器按顺序阅读网页文件，然后根据标记符解释和显示其标记的内容。但需要注意的是，对于不同的浏览器，对同一标记符可能会有不完全相同的解释，因而

可能对同一 HTML 文件显示出不同的效果。

- **URI**。有了 HTML 来编写网页，服务器就可以告诉浏览器要显示的文字、图片和视频内容等，这些内容资源（包括 HTML 格式的文件以及文件中所标记的图片等资源）是如何保存在服务器上等待浏览器来指名道姓地请求获取呢？这就需要 URI 来实现了。URI 为统一资源标志符（Uniform Resource Identifier），主要用于 Web 资源的定位和唯一标识。通常来讲，URI 可以表现为统一资源定位符（URL）或者统一资源名称（URN），或两者兼备。统一资源名称 URN 如同一个人的名称，而 URL 代表一个人的住址。换言之，URN 定义某事物的身份，而 URL 提供查找该事物的方法。比如某视频资源在网站上是以一串通过散列函数生成的随机字母来命名的，这串随机字母就标识了视频在互联网上的唯一名称，我们可以说这是一个 URN。用户如果需要进一步获取整个视频文件，网站服务器会根据内部的一个映射关系找到与这个 URN 对应的本地存储目录是什么，比如/abc/a/c.wmv，从而从这个目录下获取此资源，我们可以说这个目录是一个 URL。根据网站的不同设计，用户可以只采用 URN 或 URL，也可以两个标识都用，目前使用较多的还是通过 URL 来访问特定资源。
- **HTTP**。超文本传输协议（HyperText Transfer Protocol），描述了 Web 客户端如何从 Web 服务器请求 Web 页面，以及服务器如何把 Web 页面传送给客户的过程和相关消息。一个 HTTP 请求和响应的基本过程是：当用户通过点击某个 HTML 页面中的超链接或者直接在浏览器中输入网址来请求一个 Web 页面时，浏览器把该页面中各个对象的 HTTP 请求消息发送给服务器。服务器收到请求后，将这些对象包含在 HTTP 响应消息中作为响应。

HTTP 是 Web 技术中最为关键的应用层协议，也是目前互联网上应用最为广泛的一种网络应用层协议，下面我们将系统地认识这一协议。

### 3.3.2 HTTP 协议工作原理

HTTP 协议是由万维网协会 (World Wide Web Consortium) 和 Internet 工作小组 (Internet Engineering Task Force) 共同合作发布的。到 1997 年底，基本上所有的浏览器和 Web 服务器软件都实现了在 RFC 1945 中定义的 HTTP 1.0 版本。1998 年初，一些 Web 服务器软件和浏览器软件开始使用 RFC 2616 中定义的 HTTP 1.1 版本。HTTP 1.1 与 HTTP 1.0 后向兼容，运行 1.1 版本的 Web 服务器可以与运行 1.0 版本的浏览器“对话”，运行 1.1 版本的浏览器也可以与运行 1.0 版本的 Web 服务器“对话”。

#### 1. HTTP 与 TCP 的关系

虽然 HTTP 1.0 和 HTTP 1.1 协议本身没有规定支持其传输的底层协议是哪种协议，但通常情况下都是架构在 TCP 传输协议之上的，如图 3-7 所示。有时出于安全的考虑，HTTP 还需要经过 TLS 或者 SSL 层的封装，架构在 SSL 层之上的 HTTP 协议通常称为 HTTPS (Hypertext Transfer Protocol over Secure Socket Layer) 协议。

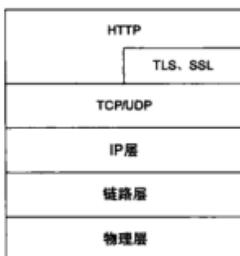


图 3-7 TCP/IP 架构中的 HTTP 应用层

HTTP 客户端（比如浏览器）首先发起建立与服务器的 TCP 连接。一旦建立连接，客户端和服务器的 HTTP 进程就可以通过各自的套接字（Socket）来

访问下层的 TCP( 比如 TCP 连接建立后, 在客户端和服务器端各有一个套接字, 此套接字包含了 IP 地址和端口号 )。不管是客户端还是服务器端, 套接字都是相应 HTTP 进程和 TCP 连接之间的接口。客户端可以通过套接字发送 HTTP 请求消息, 也从自己的套接字接收 HTTP 响应消息; 类似的, 服务器从自己的套接字接收 HTTP 请求消息, 也往自己的套接字发送 HTTP 响应消息。客户端或服务器端 HTTP 进程一旦把某个消息送入各自的套接字, 这个消息就完全落入 TCP 的控制之中。TCP 给 HTTP 提供一个可靠的数据传输服务, 这意味着由客户发出的每个 HTTP 请求消息最终将无损地到达服务器, 由服务器发出的每个 HTTP 响应消息最终也将无损地到达客户。我们可从中看到分层网络体系结构的一个明显优势——HTTP 不必担心数据会丢失, 也无须关心 TCP 如何从数据的丢失和错序中恢复出来的细节。

图 3-7 所示的是 TCP/IP 架构中的 HTTP 应用层。

HTTP 协议是一个无状态的协议, 即客户端向服务器端发送出请求时, 服务器并没有存储关于该客户端和请求的任何状态信息。即便同一个客户端在几秒钟内再次请求同一个对象, 服务器也不会响应说自己刚刚给它发送了这个对象。相反, 服务器会重新发送这个对象, 因为它已经彻底忘记自己早先做过什么, 同一个客户端的这次请求和上次请求没有任何关系。

上面提到了客户端在发出 HTTP 请求时需建立好相应的 TCP 连接, 又由于 HTTP 协议是无状态协议, 每次的请求和响应都是独立的, 所以在 HTTP 1.0 中, HTTP 使用的是非持久连接, 非持久连接过程如图 3-8 所示。

①客户端(比如浏览器)通过 HTTP 进程发起一个与服务器主机 www.netity.com 的 TCP 连接。服务器使用端口号 80 监听来自客户端的 HTTP 请求。

②客户端向与所建 TCP 连接相关联的本地套接字发出一个 HTTP 的请求消息。这个消息中包含一个路径名/netity/i\_movie/movie.shtml。

## CDN 技术详解



图 3-8 非持久连接请求和响应过程

③服务器经由与 TCP 连接相关联的本地套接字接收这个请求消息后，从服务器主机的内存或硬盘中取出对象 /netitv/i\_movie/movie.shtml，经由同一个套接字发出包含该对象的响应消息，服务器同时告知 TCP 关闭 TCP 连接。

④客户端通过套接字接收这个响应消息后，向服务器端终止 TCP 连接。

⑤客户端对收到的 /netitv/i\_movie/movie.shtml 对象进行分析，发现其中含有包括 JPEG 在内的多个对象的引用。因此，对 /netitv/i\_movie/ movie.shtml 中引用的其他对象重复步骤①~④。

以上步骤称为使用非持久连接，即 TCP 连接在每一次 HTTP 请求和响应完成后就关闭，如果客户端还要请求其他对象，需要重新为每个对象建立 TCP 连

接。当一个 Web 页面内包含多个对象并全部显示时，客户端需要与服务器端建立的 TCP 连接数较多，对整个时延和网络流量造成了较大的影响。

针对 HTTP 1.0 中 TCP 连接不能重复利用的情况，HTTP 1.1 采用了效率更高的持续连接机制，即客户端和服务器端建立 TCP 连接后，后续相关联的 HTTP 请求可以重复利用已经建立起来的 TCP 连接，不仅整个 Web 页面（包括基本的 HTML 文件和其他对象）可以使用这个持续的 TCP 连接来完成 HTTP 请求和响应，而且同一个服务器内的多个 Web 页面也可以通过同一个持续 TCP 连接来请求和响应。通常情况下，这个持续的 TCP 连接会在空闲一段特定的时间后关闭，而这个最大空闲时间是可以设置的。持续连接的 TCP 连接一般可以分为两类：带流水线和不带流水线。在不带流水线的持续连接下，用户的 HTTP 请求只能在上一个请求得到响应后才能发出；带流水线的持续连接则没有这个限制，客户端在 Web 页面发现引用时就可以发起 HTTP 请求，无须考虑上一个请求的响应消息是否已经收到。持续性连接的两种方式如图 3-9 和图 3-10 所示。



图 3-9 不带流水线的持续性连接

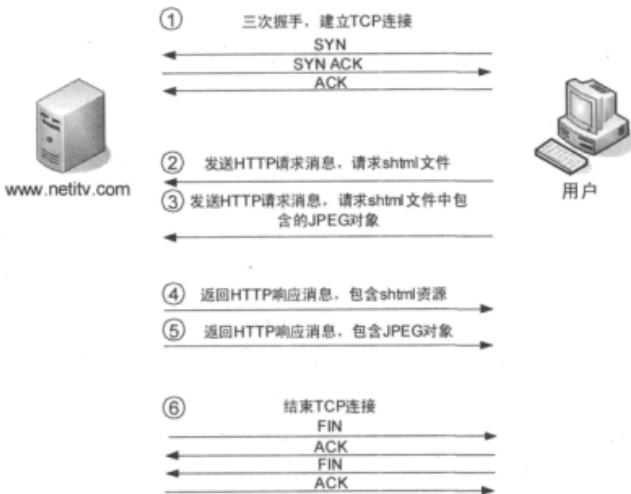


图 3-10 带流水线的持续性连接

在 HTTP 1.1 中, 客户端和服务器都默认支持持续的带流水线的 TCP 连接。如果客户端使用 HTTP 1.1 协议而又不希望使用持续的 TCP 连接, 则可以将请求消息中 `Connection` 头的值设置为 `close`; 同理, 使用 HTTP 1.1 的服务器如果不希望使用持续的 TCP 连接, 也可以将响应消息中的 `Connection` 头的值设置为 `close`。只要请求或者响应中包含了 `close` 值, TCP 连接将在本次 HTTP 请求和响应结束后关闭, 下一次 HTTP 请求必须重新发起一个 TCP 连接。关于 HTTP 请求头和实体的概念, 将在后面进行介绍。

## 2. HTTP 基本工作方式

说了这么多, 那么 HTTP 协议本身到底是如何工作的? 请求和响应消息中都包含什么样的信息? 这些信息是如何影响客户端、服务器、网关、代理的工作的? 我们又怎么利用这些信息实现 Web 的加速服务? 接下来将通过对 HTTP

协议进行详细的剖析来寻找这些答案。

如图 3-11 所示，HTTP 协议是一个标准的“请求+响应”协议，即客户端与服务器建立连接后，就向服务器发送一个 HTTP 请求，告诉服务器客户要操作什么，对什么进行操作以及怎么操作。服务器在完成请求分析后，通知客户端能不能做，为什么不能做等。为了实现这种信息交互，HTTP 协议规定请求消息中包含了请求方法（比如获取某个对象、删除某个对象）、统一资源标识符（操作对象的位置和名称）、HTTP 协议版本（比如版本 1.1）以及其他相关信息，而服务器的响应消息中包含了 HTTP 协议版本、成功或者错误的代码以及其他相关信息。

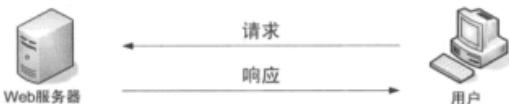


图 3-11 HTTP 的“请求+响应”过程

既然知道了 HTTP 是一种“请求+响应”的协议，接下来就看看协议中请求和响应是如何实现的。在详细介绍前，我们还是先了解一下 HTTP 协议中定义的几个关键概念。

**连接 ( Connection )**：为通信而在两个程序间建立的传输层虚拟链路，如 TCP 连接。

**消息 ( Message )**：HTTP 通信的基本单位，包括一个结构化的八元组序列并通过连接传输。图 3-12 是通过抓包软件 Wireshark 抓包看到的 HTTP 消息，右边的请求行 GET /netitv/i\_movie/movie.shtml HTTP/1.1 可以通过左边的八元组序列来表示。

**请求 ( Request )**：一个从客户端到服务器的请求信息包括应用于资源的方法、资源的标识符和协议的版本号，比如 GET /netitv/i\_movie/movie.shtml

HTTP/1.1。

|  |  |                   |
|--|--|-------------------|
| 0000   | 47 45 54 20 2F 6e 65 74 69 74 76 2F 69 5F 6d 6f          | GET /net/itv/i_ma |
| 0010   | 76 69 65 2F 6d 6f 76 69 65 2e 73 68 74 6d 6c 20          | vie/movi.e.shtml  |
| 0020   | 48 54 54 50 2F 31 2e 31 0d 0a 41 63 63 65 70 74          | HTTP/1.1 ..Accept |
| 0030   | 3a 20 69 6d 61 67 65 2F 67 69 66 2c 20 69 6d 61          | :image/ gif, ima  |
| 0040   | 67 65 2F 78 2d 78 62 69 74 6d 61 70 2c 20 69 6d          | ge/x-xbm tmap, im |
| 0050   | 61 67 65 2F 6a 70 65 67 2c 20 61 70 20 69 6d 62 67 65 2F | age/jpeg, image/  |
| 0060   | 70 6a 70 65 67 2c 20 61 70 70 6c 69 63 61 74 69          | pjpeg, applicati  |
| 0070   | 6f 6e 2F 78 2d 73 68 6F 63 6b 77 61 76 65 2d 66          | on/x-sho ckwave-f |
| 0080   | 6c 61 73 68 2c 20 61 70 70 6c 69 63 61 74 69 6f          | lash, applicatio  |
| 0090   | 6e 2F 6d 73 77 6F 72 64 2c 20 61 70 70 6c 69 63          | n/msword, applic  |
| 00a0   | 61 74 69 6F 6e 2F 76 6e 64 2e 6d 73 2d 65 78 63          | ation/vn d.ms-exc |
| Frame (149 bytes) Reassembled TCP (1481 bytes) |  |                   |

图 3-12 抓包看到的 HTTP 消息内容

**响应 ( Response )**：一个从服务器返回的信息包括 HTTP 协议的版本号、请求的状态（例如“成功”或“没找到”）以及其他一些信息，比如文档的 MIME 类型，如图 3-13 所示，响应消息返回状态行 HTTP/1.1 200 OK，也返回了 Content-Type 等信息。

|   |
|---|
| ≡ Hypertext Transfer Protocol                   |
| ≡ HTTP/1.1 200 OK\r\n                           |
| Cache-Control: private, max-age=0, no-cache\r\n |
| Pragma: no-cache\r\n                            |
| Content-Type: text/html\r\n                     |
| Connection: close\r\n                           |
| ≡ Content-Length: 0\r\n                         |
| Date: Wed, 29 Jun 2011 02:31:12 GMT\r\n         |
| Server: lighttpd\r\n                            |
| \r\n  |

图 3-13 通过抓包显示的响应消息

**资源 ( Resource )**：由 URI 标识的网络数据对象或服务，比如存储在服务器中的一个 HTML 文件。

**实体 ( Entity )**：数据资源或来自服务资源响应的一种特殊表示方法，它可能被包围在一个请求或响应信息中，是请求或响应消息的有效承载信息。一个实体包括实体头信息和实体内容。实体头信息包含与实体内容相关的原数据信息，包括：Allow、Content-Base、Content-Encoding、Content-Language、Content-Length、Content-Location、Content-MD5、Content-Range、Content-Type、

ETag、Expires、Last-Modified、extension-header 等。其中的 extension-header 允许客户端定义新的实体头信息。实体内容可以是一个经过编码的字节流，它的编码方式由实体头信息中的 Content-Encoding 或 Content-Type 定义，它的长度由实体头信息中的 Content-Length 或 Content-Range 定义。如图 3-13 所示，响应消息中的实体头信息 Content-type 指示实体内容的编码方式是超文本标记语言文本。

**客户端（Client）：**为发送 HTTP 请求而建立连接的应用程序，比如浏览器。

**服务器（Server）：**一个接受客户端连接请求并对请求返回信息的应用程序。

**源服务器（Origin server）：**是一个特定资源可以在其上驻留或被创建的服务器。在 Web 访问中，源服务器通常是编辑产生网页并保持其最新变动的源站。

**代理（Proxy）：**是一个处于客户端和服务器之间的中间程序。它对于客户端来说可以充当一个“服务器”，为客户端直接提供服务；对服务器来说可以充当一个“客户端”，在接收到其他客户端的请求后，自身再以一个客户端的方式向源服务器发起请求。从代理对请求和响应的处理来看，代理可以表现为透明方式和非透明方式。透明方式不会对请求和响应进行修改，但可能需要对代理服务器进行认证和授权；而非透明方式则需要修改请求和响应，以便为用户代理提供附加服务，包括注释、类型转换、协议简化、类型过滤等。

**网关（Gateway）：**也是一个作为中间媒介的服务器。代理主要代表客户端发起请求和提供服务，而网关则是代表源服务器来接收请求和提供服务，发出请求的客户端并没有意识到它在同网关打交道，而是感觉从源服务器获得了响应。代理和网关的位置见图 3-14。

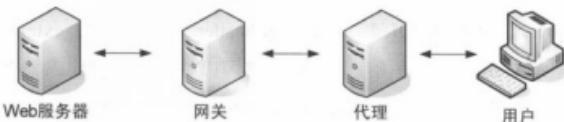


图 3-14 网关和代理的位置示意

**隧道 (Tunnel)**：是作为两个连接中继的中介程序员。尽管隧道可能由一个 HTTP 请求初始化，但通常认为它不属于 HTTP 通信。当被中继的连接两端关闭时，隧道也就随之消失了。比如主机 A 在防火墙内，主机 B 在防火墙外，防火墙只允许 80 端口通信，所以主机 B 如果要 Telnet 主机 A（端口 23），需要在 B 启动一个隧道程序客户端以及在 A 启动隧道程序服务器端，这时 B 与隧道客户端建立连接进行 Telnet 通信，隧道程序再把通信内容向 A 的 80 端口发送，从而通过防火墙，隧道服务器端从 80 端口收到内容后再转交给 23 端口。

**缓存 (Cache)**：是位于源服务器和客户端之间的服务单元。缓存会根据用户代理的请求从服务器获取相关的资源，在为用户代理提供服务的同时也将资源保存在本地，以便后续为相同的请求直接提供服务，而无须再从源服务器或其他服务器获取相关资源。这些保存属于原文件的副本，可以是 HTML 文件，也可以是图片、视频等多媒体对象。

了解了上述基本概念后，接下来详细分析 HTTP 的请求和响应。比如我们现在打开 IE 浏览器，在地址栏里输入 <http://www.netity.com/> netity/i\_movie/movie.shtml，然后按回车键，在网络正常的情况下会显示标题为“电影-天翼视讯”的页面。整个过程就是用户利用用户代理（IE）发出一个 HTTP 请求，请求远端 HTTP 服务器上的 www.netity.com 主机发送过来一个 Web 页面。我们通过抓包软件 Wireshark 捕捉到了这次请求和响应的全过程，如图 3-15 所示。

| No. | Time     | Source         | Destination    | Protocol | Request/Response   |
|-----|----------|----------------|----------------|----------|--|
| 1   | 0.000000 | 10.0.55.68     | 202.102.86.241 | TCP      | 4739 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WSZ=0 TSN=0 TSER=0 |
| 2   | 0.076652 | 202.102.86.241 | 10.0.55.68     | TCP      | http > 4739 [ACK] Seq=1 Win=8192 Len=0 MSS=1386 TACK_PENH=1        |
| 3   | 0.076652 | 10.0.55.68     | 202.102.86.241 | TCP      | 4739 > http [ACK] Seq=1 Win=8192 Len=0 MSS=1386 TACK_PENH=1        |
| 4   | 0.076652 | 10.0.55.68     | 202.102.86.241 | TCP      | 4739 > http [ACK] Seq=1 Win=8192 Len=0 MSS=1386 TACK_PENH=1        |
| 5   | 0.057258 | 202.102.86.142 | 10.0.55.68     | TCP      | 4719 > [ACK] Seq=1 Win=8192 Len=0                                  |
| 6   | 0.054496 | 202.102.86.142 | 10.0.55.68     | TCP      | [ACK] segment of a reassembled PDU                                 |
| 7   | 0.054496 | 10.0.55.68     | 202.102.86.142 | TCP      | 4719 > [ACK] Seq=1 Win=8192 Len=0                                  |
| 8   | 0.055204 | 10.0.55.68     | 202.102.86.142 | TCP      | 4719 > http [ACK] Seq=0 Win=2048 ACK=2773 Win=8192 Len=0           |
| 9   | 0.056761 | 10.0.55.68     | 202.102.86.142 | TCP      | 4720 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WSZ=0 TSN=0 TSER=0 |
| 10  | 0.056761 | 202.102.86.142 | 10.0.55.68     | TCP      | http > 4720 [ACK] Seq=1 Win=8192 Len=0                             |
| 11  | 0.056889 | 202.102.86.142 | 10.0.55.68     | TCP      | [ACK] segment of a reassembled PDU                                 |
| 12  | 0.056922 | 10.0.55.68     | 202.102.86.142 | TCP      | 4719 > http [ACK] Seq=0 Win=2048 ACK=2755 Win=8192 Len=0           |
| 13  | 0.058252 | 202.102.86.142 | 10.0.55.68     | TCP      | [ACK] segment of a reassembled PDU                                 |

图 3-15 通过抓包示意的 HTTP 请求和响应过程

从图 3-15 中可以看到，首先由用户代理（IP 地址 10.9.55.68，端口 4719）向服务器主机（IP 地址 202.102.86.141，端口 80，这里的服务器不是源服务器）发起 TCP 连接建立请求[SYN]，服务器同意用户请求并向用户代理响应[SYN, ACK]，用户代理再返回[ACK]，这样双方就通过三次握手过程建立起了一个 TCP 连接。此时，用户代理向服务器主机发送 GET /netivity\_i\_movie/movie.shtml HTTP/1.1 的 HTTP 请求，图 3-15 所示的下半部分是整个请求消息的内容，具体如下：

```
GET /netitty/i_movie/movie.shtml HTTP/1.1
Accept: /*
Accept-Language: zh-cn
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: www.netitty.com
Connection: Keep-Alive
Cookie: Hm_lvt_0f611770c88e01e560617addb8be1448=1309405044859; ...
```

以上是 HTTP 请求的实例，在 HTTP 协议中，请求的基本格式如下：

HTTP 请求行 \*格式为：请求方法（空格）URI（空格）版本号\*

.....  
通用头 n: (空格) 通用头 n 的值  
请求头 1: (空格) 请求头 1 的值  
.....  
请求头 n: (空格) 请求头 n 的值  
实体头 1: (空格) 实体头 1 的值  
.....  
实体头 n: (空格) 实体头 n 的值  
(空行)  
实体内容

在以上的例子中，请求行为 GET /nettv/i\_movie/movie.shtml HTTP/1.1，其格式为“请求方式 (空格) URI (空格) 版本号”。其中，“GET”是一种请求方式，表示是从服务器获取 URI 所标识的资源，这里的 URI 是一个扩展名为“shtml”的文件，用户代理支持的协议是 HTTP 1.1 版本的协议。“GET”是获取资源对象的请求方式，常用的请求方式还包括 HEAD、POST、PUT、DELETE、TRACE、CONNECT 等。表 3-1 描述了主要应用的几种请求方式。

表 3-1 HTTP 协议中的典型请求方式

| 请求方式 | 功能描述   |
|------|--|
| GET  | 请求获取由 URI 所指定的资源。这个资源通常包含在响应消息的实体中返回给请求者   |
| HEAD | 此方法和 GET 方法类似，只不过服务器不能在响应消息里包含资源内容，只返回头信息  |
| POST | 请求源服务器将请求消息中包含的实体作为请求资源的一个附属物。POST 可以完成以下功能：<br>(1) 对服务器上已存在资源进行注释<br>(2) 发布消息给一个公告板、新闻组、邮件列表，或者相似的文章组<br>(3) 提供一个数据块，如提交一个表单给一个数据处理过程<br>(4) 通过追加操作来扩展数据库 |

续表

| 请求方式    | 功能描述  |
|---------|---|
| PUT     | 请求服务器把请求里的实体存储为 URI 所标的资源。如果 URI 所指定的资源已经在源服务器上存在，那么此请求里的实体内容应该是此 URI 所指定资源的修改版本；如果请求 URI 指定的资源不存在，源服务器根据请求里的实体内容创建一个使用此 URI 所标识的资源 |
| DELETE  | 请求源服务器删除 URI 指定的资源。此方法可能会在源服务器上被人为干涉，所以服务器返回客户端的状态码不一定成功  |
| TRACE   | 用于触发远程的、应用层的请求响应回路。这个方法通常让客户端测试其连到服务器的网络通路，收到请求的每一个代理服务器和网关服务器不断转发请求，同时向客户端发出响应消息，直到最后的接收者。TRACE 请求不能包含实体                           |
| CONNECT | 用于一个代理服务器，使其提供隧道服务  |

以上例子中的请求 URI 为 `www.netitv.com/netitv/i_movie/movie.shtml`，表示访问主机 `www.netitv.com` 下 `netitv` 目录中 `i_movie` 子目录下的 `movie` 文件。

请求行下面是通用头、请求头和实体头。通用头不是应用于请求消息中的特定实体，而是应用于整个请求消息，通常既适用于请求消息，也适用于响应消息。通用头有 `Connection`，`Connection` 中的“`Keep-Alive`”表明了 TCP 连接是一个持续连接。典型使用的通用头介绍见表 3-2 所示。

表 3-2 通用头及其功能

| 通用头                     | 描述  |
|-------------------------|---|
| <code>Connection</code> | 指示客户端与服务器在进行 HTTP 通信时如何处理 TCP 连接，如果 <code>Connection</code> 的值为 <code>close</code> ，则表示本次 HTTP 请求响应后结束 TCP 连接；如果 <code>Connection</code> 的值为 <code>Keep-Alive</code> （HTTP 1.1 下为默认），则表示 TCP 连接一直有效 |
| <code>Date</code>       | <code>Date</code> 通用头域表明消息产生的日期和具体时间  |
| <code>Pragma</code>     | 被用于包含特定执行指令，这些指令可能被应用于请求和响应消息传递过程中的任何接收者。最常用的为 <code>Pragma: no-cache</code> ，表示对请求的实体内容不予缓存，后面继续介绍   |

续表

| 通用头               | 描述   |
|-------------------|--|
| Transfer-Encoding | 指示整个消息主体的传输编码方式,主要是为了实现在接收端和发送端之间进行安全的数据传输。比如 Transfer-Encoding: chunked 表示消息主体采用块编码的方式  |
| Upgrade           | 客户端可以通过它表示自己希望进行协议转换(比如从 HTTP 一个版本转换到另一个版本),如果服务器同意的话会切换到这个指定的协议,这个协议一般是指应用层协议   |
| Via               | Via 用来指明请求和响应消息在客户端和服务器之间传递时所经过的代理和网关以及相关的中间协议。比如 HTTP 1.0 的请求消息发送到代理 A, A 使用 HTTP 1.1 将消息转发给网关 B, B 再发送给源服务器,这时源服务器看到 Via 头域为 Via: 1.0 A, 1.1 B |
| Warning           | 携带相关警告信息,比如可以被代理和网关用来警告客户端所接收内容的过期状态以及警告客户端实体内容的编码格式发生了变化等   |

请求头主要包含本请求的附加信息以及客户端的附加信息,本例中的请求头包括 Accept、Accept-Language、Accept-Encoding、User-Agent 和 Host。这些请求头中的信息规定了与本请求和客户端相关的特有属性,比如 Accept-Encoding 中的 “gzip, Deflate” 表明用户代理可以接受 gzip 编码格式或者 Deflate 编码格式的内容。典型使用的请求头见表 3-3 所示。

表 3-3 常用的请求头及其描述

| 请求头            | 描述  |
|----------------|---|
| Accept         | 用于指定本请求所期望的服务器返回响应的媒体类型。比如 Accept: audio/*; q=0.2, audio/basic 表明用户客户端希望接收到 audio/basic 的媒体,但也可以接受其他任何 audio 类型(audio/*表示),但偏好程度只有 audio/basic 的 20% (这里的 q 是一个表示喜欢程度的质量参数) |
| Accept-Charset | 用来指定客户端能接受什么样的字符集响应。比如 Accept-Charset: iso-8859-5   |

续表

| 请求头             | 描述   |
|-----------------|--|
| Accept-Encoding | 和 Accept 头相似, 但 Accept-Encoding 限定的是客户端可以从服务器接收的内容编码。比如 Accept-Encoding: compress, gzip  |
| Accept-Language | 和 Accept 头相似, 但 Accept-Language 限定的是客户端可以从服务器接收的自然语言。比如 Accept-Language: en-us   |
| Accept-Range    | 指示客户端希望从服务器收到的内容的字节范围  |
| Authorization   | 客户端的某些访问要求服务器授权, 通常客户端会先在服务器的响应消息中收到包含有授权证书的 WWW-Authenticate 头, 然后在需要授权的请求消息里包含一个带有这个授权证书的 Authorization 请求头  |
| From            | 包含客户端当前操作用户的 E-mail 地址, 代表的是具体的用户, 而不是发出请求的主机  |
| Host            | 标识请求资源的网络主机和端口号, Host 头的值代表源服务器或网关的命名授权, 一般会在用户访问的 URL 中标明。比如对 <a href="http://www.netitv.com/netitv/i_movie/movie.shtml">http://www.netitv.com/netitv/i_movie/movie.shtml</a> 的请求在 HTTP 消息里通过以下形式表示:<br>GET /netitv/i_movie/movie.shtml HTTP/1.1<br>Host: www.netitv.com<br>其中 www.netitv.com 就是主机名 |
| If-Match        | 与 ETag 一起用来判断已缓存的内容是否被修改过。比如, 客户端在获取内容时会获取一个与内容相关的实体标签 ETag( 内容变化会使 ETag 变化 ), 下次再请求同样内容时, 会在请求头的 If-Match 中包含这个 ETag, 然后发给可能存有更新内容的服务器。服务器将收到的 ETag 与本地目前的 ETag 进行比较, 如果匹配返回所请求内容。这种方法在断点续传的时候使用较多  |
| If-None-Match   | 与 If-Match 用法类似, 结果相反。如果 If-None-Match 中包含了 ETag 的值, 服务器在进行比较后发现如果不匹配, 则返回所请求的内容, 否则不返回相关内容。这种方法在网页刷新的时候使用较多   |

续表

| 请求头                 | 描述   |
|---------------------|--|
| If-Modified-Since   | 当客户端请求服务器判断自己缓存的内容是否变化时，可以设置 If-Modified-Since 头，如果服务器上的内容修改时间比这个头的时间值还要新，那么将返回新的内容，否则不返回。这个与 If-Unmodified-Since 类似，是用来判断资源变化的另一种方式 |
| If-Unmodified-Since | 与 If-Modified-Since 正好相反，如果服务器上的内容修改时间比这个头时间的值还要新，则不返回新的内容，否则返回所请求内容   |
| If-Range            | 一般结合 Range 头一起使用。If-Range 的值可以是 ETag，也可以是具体的时间值。如果收到含有 If-Range 头的服务器发现所请求内容没有改变，则根据 Range 头指示的字节范围进行续传，否则返回整个文档内容                   |
| Max-Forwards        | 指示此 HTTP 请求可以途经的代理服务器或网关个数。每经过一个代理服务器或网关，这个值就会被减 1。如果减到 0，则代理服务器或网关将中止继续发送   |
| Proxy-Authorization | 与 Authorization 类似，只不过是需要代理服务器进行授权时才使用   |
| Range               | 客户端通知服务器只需返回部分内容，以及部分内容的范围。这对于较大文档的断点续传是有很大帮助的。如果客户机在一次请求中，只收到了服务器返回的部分内容，则客户端可以再发出一个带 Range 头的请求，这时服务器将会返回 Range 头值规定的那部分内容         |
| Referer             | 告诉服务器这次请求是通过点击哪个网页上的超链接转向过来的。由于可以使用 Telnet 来伪造 HTTP 请求，所以 Referer 是不可靠的。“Referer”是由于 HTTP 的作者拼写错误，在实际应用中沿袭了下来，不可以写成正确的拼写方式 Referrer  |
| User-Agent          | 用于指定浏览器的类型和名字，比如 Mozilla/4.0   |

另外，本例中除了 HTTP 1.1 协议规定的通用头和请求头以外，还包括了一些请求头的扩展头，比如 Cookie，后面会详细讲到 Cookie 的使用。

服务器在收到用户代理发送的 HTTP 请求后，返回相应的响应消息，抓包

结果如图 3-16 所示。

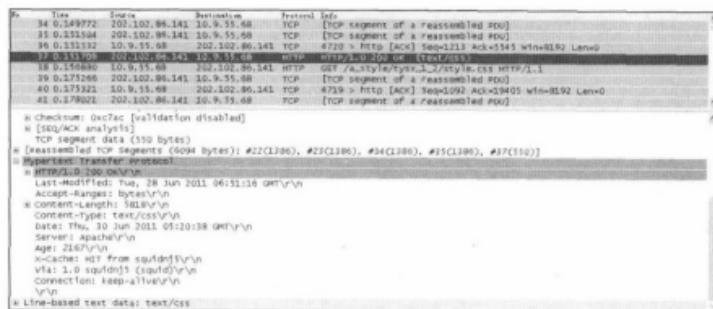


图 3-16 响应消息抓包示意图

从图中看到服务器返回的响应消息具体如下：

```

HTTP/1.0 200 OK
Last-Modified: Tue 28 Jun 2011 06:51:16 GMT
Accept-Ranges: bytes
Content-Length: 5818
Content-Type: text/css
Date: Thu, 30 Jun 2011 05:20:38 GMT
Server: Apache
Age: 2167
X-Cache: HIT from squidnj5
Via: 1.0 squidnj5 (squid)
Connection: keep-alive
(空行)
Line-based text data: text/css

```

在 HTTP 协议中响应消息的基本格式如下：

```

HTTP 状态行
通用头 1: (空格) 通用头 1 的值
.....
通用头 n: (空格) 通用头 n 的值
响应头 1: (空格) 响应头 1 的值

```

.....  
响应头 n: (空格) 响应头 n 的值  
实体头 1: (空格) 实体头 1 的值  
.....  
实体头 n: (空格) 实体头 n 的值  
(空行)  
实体内容

在上面的例子中，HTTP 的状态行主要是 HTTP/1.0 200 OK，其基本格式是“协议版本号(空格)状态码(空格)原因短语”。这里协议版本号是“HTTP/1.0”，“200”是状态号，表明请求成功。状态代码由三位数字组成，第一位定义了响应的类别。

1xx：指示信息——表示请求已接收，继续处理。

2xx：成功——表示请求已被成功接收、理解、接受。

3xx：重定向——要完成请求必须进行更进一步的操作。

4xx：客户端错误——请求有语法错误或请求无法实现。

5xx：服务器端错误——服务器未能实现合法的请求。

典型的状态号及其含义见表 3-4。

表 3-4 常用的响应消息状态码

| 状态码 | 原因短语                           | 描述                                      |
|-----|--------------------------------|---|
| 100 | Continue                       | 客户端必须继续发出请求                             |
| 200 | OK                             | 客户端请求成功                                 |
| 202 | Accepted                       | 接受和正在处理，但处理未完成                          |
| 203 | None-Authoritative Information | 返回的信息不确定或不完整                            |
| 206 | Partial Content                | 客户端发送了带 Range 的 GET 请求，服务器正确地返回了此范围内的数据 |

续表

| 状态码 | 原因短语                  | 描述   |
|-----|-----------------------|--|
| 301 | Moved Permanently     | 请求的数据已永久删除或永久地转移至其他地方                                    |
| 302 | Found                 | 指出被请求的文档已经临时移动到别处，此文档的新 URL 在 Location 响应头中给出            |
| 304 | Not Modified          | 被缓存的版本是最新的，可以继续使用  |
| 400 | Bad Request           | 客户端请求语法有错误，不能被服务器理解                                      |
| 401 | Unauthorized          | 表示客户端访问的资源受用户名和密码的保护，提示客户端应重新发出一个带有 Authorization 头的请求消息 |
| 403 | Forbidden             | 服务器拒绝提供服务  |
| 404 | Not Found             | 请求的资源不存在   |
| 500 | Internal Server Error | 服务器内部发生错误  |
| 502 | Bad Gateway           | 网关错误   |
| 503 | Server Unavailable    | 服务器当前不能处理请求，一段时间后可恢复正常                                   |

通用头前面已经介绍了，在这个响应消息里包括了 Date、Via、Connection。响应头包含响应的附加信息，给出了有关服务器的信息以及请求 URI 所指定资源的访问机制，以上示例中就包含了 Accept-Ranges、Server、Age，比如 Server 就表明源服务器用于处理请求的软件信息是 Apache。常用的响应头如表 3-5 所示。

表 3-5 常用响应头及其描述

| 响应头          | 描述  |
|--------------|---|
| Accept-Range | 指明服务器对用户请求范围的接受程度。比如 Accept-Range: 100-599。表示服务器只接受请求范围的第 100 字节到 599 个字节 |
| Age          | 当服务器使用缓存的内容去响应请求时，用该头部表明该内容从产生到现在经过多长时间                                   |

续表

| 响应头                | 描述  |
|--------------------|---|
| ETag               | 对应实体内容的一个实体标签，与实体内容紧密相关，实体内容发生任何改变都会使这个头的值发生变化  |
| Location           | 当客户端向服务器请求一个内容，但请求消息中 URI 所标识的内容发生了转移，那么接收者能将访问重定向于 Location 指示的 URI，如 Location: http://www.netitv.com/netitv/movie_1.shtml   |
| Proxy-Authenticate | 此头必须被包含在 407 响应（代理认证）里。此头域值由一个 challenge 和 parameters 组成，challenge 指明了认证机制，而 parameters 是一些与此代理相关的参数  |
| Retry-After        | 可以与 503（服务不可用）响应一起使用，服务器用它来告知请求端服务不可用的时长。也可与 3xx（重定向）响应一起使用，服务器用它来告知客户端发送重定向请求之前需要等待的最小时长，比如 Retry-After: 60  |
| Server             | 包含了源服务器处理请求的软件信息，比如 Server: Apache  |
| Vary               | 指定了一些请求头，这些请求头可被服务器用来决定缓存的没有过期的内容能否被允许去响应后续相同的请求，比如请求消息发送到代理服务器，其中 Accept-Encoding 头是 “gzip”，代理服务器向源服务器转发请求，源服务器给代理服务器返回响应消息，其中 Vary 头为 Vary: Accept-Encoding，此时代理服务器将响应内容与 gzip 一起缓存。如果后续用户再向代理服务器发送请求时，并且 Accept-Encoding 头的值是 gzip，代理服务器则可以直接使用未过期的缓存内容来响应，如果 Accept-Encoding 头的值是 deflate 或者其他，则需要重新从源服务器获取 |
| WWW-Authenticate   | 必须包含在 401（没有被认证）响应消息中，这个响应头应至少包含一个 challenge 值来标识认证方案以及一些与所请求的 URI 相关的参数  |

响应消息中还使用了实体头，包含了与实体内容相关的元数据信息，示例中出现的 Last-Modified、Content-Length、Content-Type 都属于实体头，比如 Last-Modified 所标识的时间值表示实体内容自这个时间后再没有被修改过。此外，示例中的 X-Cache 是一个非标准的扩展头，表明实体内容是否由某个代理

的 Cache 提供，也可以说成是否被某个代理的 Cache 命中，示例表明内容被 Squidnj5 这个代理命中。典型的实体头如表 3-6 所示。

表 3-6 典型实体头及其描述

| 实体头              | 描述   |
|------------------|--|
| Allow            | 指示可以使用的请求方式。比如 Allow: GET, HEAD, PUT   |
| Content-Encoding | 指示实体内容以哪种方式编码。比如 Content-Encoding: gzip  |
| Content-Language | 实体内容的国家语言，比如 Content-Language: zh-cn 表示简体中文  |
| Content-Length   | 实体内容的大小，比如 Content-Length: 3495  |
| Content-Location | 当实体内容的真正 URI 和请求 URI 不同时，在消息里提供这个真正的位置信息   |
| Content-MD5      | 实体内容的 MD5 摘要算法 Base-64 值，以提供实体内容的完整性校验。服务器可以通过对实体内容进行 MD5 摘要算法与此头的值是否相同来确定接收的请求是否没有错误与改变   |
| Content-Range    | 表示返回的内容是整个实体内容的哪个部分。比如 Content-Range: 2351-4767/8565，表明本次响应是实体内容的第 2351 字节到第 4767 字节，8565 是实体内容的总大小  |
| Content-Type     | 服务器在返回内容时需要告诉浏览器本响应的内容是什么类型的。HTTP 中把这种不同媒体类型的格式称为多媒体文件格式（MIME），而本实体头指出所传输实体内容的 MIME。由于 Web 服务器不知道所返回的内容文件是哪种 MIME，所以需要通过对 Web 服务器进行设置，使文件扩展名与 MIME 之间进行映射。比如 Content-Type: text/html; charset=ISO-8859-4 |
| Expires          | 指示当前内容在何时之后被认为过期，缓存在为客户端请求提供服务时，如果缓存内容已经过期，则不会用此缓存内容来直接提供服务。比如 Expires: Tue, 17 Jul 2011 18:06:45 GMT  |
| Last-Modified    | 指定所访问内容的最后更新时间   |

以上将 HTTP “请求+响应”的基本工作原理、请求消息和响应消息可以传达的信息以及所起的作用进行了描述，我们也大概知道 HTTP 是怎么回事了。作为 CDN 技术的基础技术之一，它对于我们理解现有 CDN 缓存设备的工作原理有很大帮助。除了以上描述的基本原理外，我们再来看看 HTTP 协议的几个关键应用：**Cookie 和 Session** 技术、安全协议以及最重要的缓存机制。

### 3.3.3 HTTP 中的 Cookie 和 Session

前面提到 HTTP 协议是一种无状态的协议，也就是说，当前的 HTTP 请求与以前的 HTTP 请求没有任何联系。显然，这种无状态的情形在某些时候将让用户觉得非常麻烦，比如在网上商城购物时，每购买一个商品都要重新输入一次用户名和密码，用户很快就会失去耐心，而且反复的输入也产生了更大的风险。所以，Web 访问通过使用 **Cookie** 和 **Session** 解决这个问题。

#### 1. Cookie 介绍

**Cookie** 在英文中是小甜饼的意思，这个词我们经常在浏览器中看到，但“小甜饼”怎么会跟浏览器扯上关系呢？在你浏览以前登录过的网站时可能会在网页中出现“你好，XX”，你会感觉很亲切，就像是吃了一个小甜饼一样。这其实是通过访问你客户端的一个文件来实现的，因此这个文件也就被称为“小甜饼”了。

当你浏览某网站时，网站服务器会通过在响应消息中设置一个指示信息要求客户端在用户机器上存储一个小文本文件，用来记录你的用户 ID、密码、浏览过的网页、停留的时间以及其他浏览或请求过的信息等，当你再次来到该网站时，网站通过读取 **Cookie**，得知你的相关信息，就可以做出相应的动作。比如在页面显示欢迎你的标语，或者让你不用再次输入 ID、密码就直接登录等。你可以在 IE 的“Internet 选项”对话框的“常规”选项卡中，选择“设置/查看文件”，查看所有保存到你电脑里的 **Cookie**。这些文件通常是以 `user@domain[数`

字]格式命名的，`user` 是你的本地用户名，`domain` 是所访问的网站的域名。`Cookie` 中的内容大多经过了加密处理，因此我们看到的是一连串看似随机的字母和数字组合，只有服务器处理程序才知道它们真正的含义。

`Cookie` 的工作机制是怎样的？当用户在浏览器中输入 URL，这时浏览器便会向网站主机发送一个读取网页的请求，在这个请求发送出去之前，浏览器先查看本地是否保存有此网站的 `Cookie` 文件。如果发现了 `Cookie` 文件，则将 `Cookie` 文件中的信息（一般都进行了加密）放在请求消息 `Cookie` 头中一起发送给服务器；如果没有发现 `Cookie` 文件，则不会发送任何 `Cookie` 信息。

假设客户端在发送请求时没有 `Cookie` 信息，网站服务器会认为用户是第一次访问这个网站，因此为用户在数据库中设置一个用户 ID，并记录与这次访问相关的一些信息（比如最近访问时间），然后服务器将用户 ID 和访问相关的信息包含在响应消息中返回给客户端（通过设置 `Set-Cookie` 扩展头来包含这个信息），客户端将获得的用户 ID 和这些信息保存在本地的一个 `Cookie` 文件里。

如果客户端在发送请求时本地有 `Cookie` 文件，则把 `Cookie` 信息也发送给服务器，服务器收到 `Cookie` 信息后会根据里面的用户信息（比如用户 ID）去检索数据库中保存的对应信息。服务器在数据库中找到用户相关的信息后，会根据这次请求消息更新数据库中的信息，基于更新后的数据库信息返回一个专门针对这个用户的页面，同时返回的还有服务器更新过后的 `Cookie` 信息，客户端保存的 `Cookie` 文件将得到更新（比如“最近访问时间”发生了更新）。`Cookie` 的传递过程如图 3-17 所示。

值得注意的是，`Cookie` 包含哪些信息不是规范的，各个网站都会有自己的保存内容。有的网站将用户 ID、用户所在地、用户最近访问时间等保存在 `Cookie` 中；有的网站则将用户 ID、用户购物信息、浏览喜好等保存在 `Cookie` 中；还有的会将登录用户名和密码保存在 `Cookie` 中。由于 `Cookie` 保存在客户端，存在一定的安全隐患，所以越来越多的网站趋向于尽量减少 `Cookie` 中包含信息的数量，很多网站只是将服务器生成的用户 ID 保存在 `Cookie` 文件中，而

与用户相关的其他信息都保存在服务器的数据库中。

服务器要求客户端保存的 **Cookie** 信息传送到浏览器后，浏览器会根据本地的设置来决定是否保存 **Cookie** 文件。如果浏览器本地不允许保存 **Cookie**，那么浏览器在关闭后 **Cookie** 文件自动删除；如果允许保存，则 **Cookie** 文件在浏览器关闭后还会保存一段时间，这段时间的大小是由服务器在发送 **Cookie** 信息时就决定好的，主要通过服务器处理软件在 **Cookie** 信息的 **Expiration**（有效期）属性中进行指示，如果这个属性没有被设置，则浏览器默认此 **Cookie** 文件是不被保存的，即浏览器关闭后就删除。

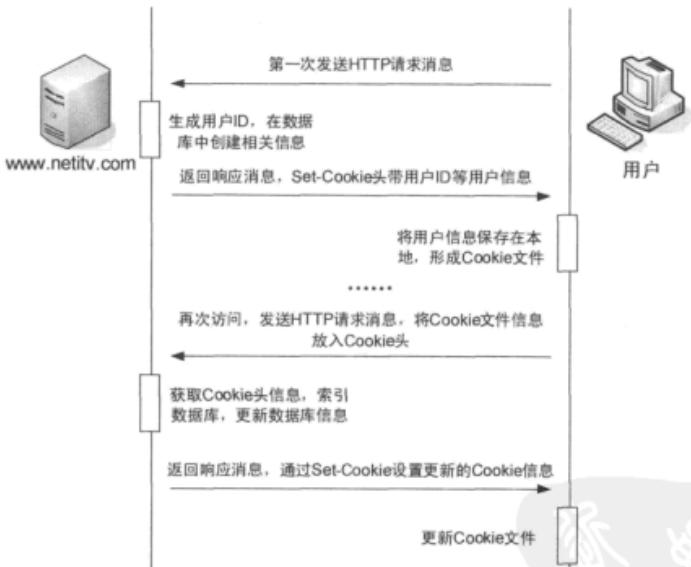


图 3-17 **Cookie** 使用原理

**Cookie** 在 RFC 2965 中进行描述，每个客户端最多保持 300 个 **Cookie**，每个域名下最多 20 个 **Cookie**（实际上一般浏览器现在都比这个多，如 Firefox 是

50个），而每个Cookie的大小最大为4KB。对于Cookie的使用，最重要的就是要控制Cookie的大小，不要放入无用的信息，也不要放入过多的信息，特别是一些很敏感的信息，因为Cookie并不那么安全。

Cookie文件保存在用户的机器中，而由于用户机器的安全级别往往较低，所以被恶意软件嗅探、篡改的几率就会大很多。如果Cookie文件含有用户ID和密码等敏感信息且又没有做加密处理，这部分信息将很容易暴露出去。即使Cookie文件在本地保存时做了加密处理，攻击者也可以通过用户机器的其他漏洞获取此文件，然后进行暴力破解来获取关键数据。另外，Cookie信息通过请求消息从客户端发送到服务器的过程中，攻击者会从网络中截获这部分信息。虽然这部分信息往往已经通过MD5加密，但攻击者可以不需要解密这部分信息，而是通过套用这个Cookie信息来冒充真正用户骗取服务器的信任，这种方法称为Cookie欺骗。所以，鉴于Cookie存在一定的安全风险，很多浏览器都有选项供用户禁用Cookie。

## 2. Session介绍

除了使用Cookie来保持HTTP连接状态以外，Session是另一种更为安全的方法。Session从原始意义上讲是会话，如果放在Web访问中，是指用户通过浏览器进入某个网站时，从进入网站到浏览器关闭所经过的这段时间，也就是用户与网站一次“会话”的时间。我们在这里介绍的Session是利用其定义来强调的一种保持用户访问状态的机制。

Session机制是怎么实现的？当用户A打开浏览器，输入URL向服务器发出请求时，服务器为这个用户分配了一个全局唯一的标识，称为Session ID，标识与这个用户的这次会话。这个Session ID可以保存在服务器的内存中，也可以写到文件里甚至数据库中，后者会增加每次读取Session ID的系统开销，但也能降低前者在停电或系统宕机情况下所带来的损失。如果在服务器为用户A提供网页的同时，另一个用户B也打开浏览器请求服务器提供服务，服务器会再为用户B分配一个Session ID，这两个Session ID是不能相同的，通常服

务器会使用散列函数为正在访问的不同用户产生各自的 Session ID，尽最大可能确保各个 ID 的全局唯一性。在为用户生成新的 Session ID 后，此 Session ID 会包含在响应消息中返回给浏览器。当用户下次访问服务器时就可以在请求中包含此 Session ID，服务器将收到的 Session ID 与数据库或文件中保存的 ID 进行比较，如果匹配则可以获知用户以前访问的信息以及相关状态，否则将需要重新生成 Session ID 并返回给浏览器。

服务器将 Session ID 返回给浏览器的方法有以下两种。

一种是 Cookie 方法，即采用 HTTP 协议中的 Cookie 机制来实现，服务器端产生 Session ID 后，通过 Set-Cookie 扩展头将 Session ID 传送到浏览器，而浏览器此后的每一次请求都会在 Cookie 头里带上这个 ID。

由于 Cookie 可能被人为禁止，所以必须有其他机制以便在 Cookie 被禁止时仍然能够把 Session ID 传递回服务器，这样就产生了另一种 Session ID 传送方法，叫 URL 重写。就是服务器在返回用户请求的页面之前，将页面内所有的 URL 后面附上 Session ID，这样用户在收到响应之后，无论点击哪个链接，都会再带上 Session ID，从而就实现了会话的保持。把 Session ID 附加在 URL 后面的方式也有两种，一种是作为 URL 路径的附加信息，表现形式为 `http://.../xxx;jsessionid=...`；另一种是作为查询字符串附加在 URL 后面，表现形式为 `http://.../xxx?jsessionid=...`。这两种方式对于用户来说是没有区别的，只是服务器在解析时的处理机制不同，采用第一种方式比较有利于把 Session ID 的信息和正常程序参数区分开来。

前面虽然提到 Session 的定义是用户访问某网站到关闭浏览器的会话时间，但在实际 Web 访问中，关闭浏览器并不意味着 Session 真正结束了，因为 Session ID 还会保存在服务器中。要在服务器中清除 Session ID 的信息，需要浏览器主动发出一个结束 Session 的请求，比如点击网页上的“注销”、“退出”等链接，而很多用户一般都会在不想继续访问时直接关闭浏览器，所以 Session ID 在过期（服务器设置的一个时间）之前会作为一个隐患继续存在。不管是采用

**Cookie** 的方式还是采用 URL 重写的方式，都有这种 ID 遗留问题，如果攻击者获取了 Session ID，即使用户已经关闭了相关联的浏览器程序，他们仍然可以在请求中包含这个 ID 来访问服务器，获取用户的关键信息。

**Session** 将与用户访问相关的敏感信息都保存在安全级别较高的服务器中，而不是用户机器上，所以像 **Cookie** 文件那样被窃取而导致信息丢失的风险大大降低。但是，与用 **Cookie** 来保持状态性一样，**Session** 仍然会在请求和响应中出现 Session ID 信息，这个信息仍然可以被网络攻击者捕获到，并冒充用户欺骗服务器，进行中间人攻击。

这样看来，**Session** 和 **Cookie** 方式存在以下的差异。

(1) **应用场景。** **Cookie** 的典型应用场景是 Remember Me 服务，即用户包括账户在内的关键信息通过 **Cookie** 文件的形式保存在客户端，当用户再次请求匹配的 URL 的时候，**Cookie** 信息会被传送到服务端，交由相应的程序完成自动登录等功能；**Session** 的典型应用场景是用户登录某网站之后，服务器将其登录信息以及其他关键信息放入由一个 **Session** ID 关联的数据库或文件中，在以后的每次请求中都检验 **Session** ID 来确保该用户合法。

(2) **安全性。** **Cookie** 将一些信息保存在客户端，如果不进行加密的话，无疑会暴露一些隐私信息，安全性很差，一般情况下敏感信息是经过加密后存储在 **Cookie** 中，但也很容易被窃取。而 **Session** 主要将信息存储在服务器端，如果存储在文件或数据库中，也有被窃取的可能，只是这种可能性非常小。**Session** 与 **Cookie** 在安全性方面都存在会话劫持的问题，即在网络通信中被攻击者捕获而用来冒充真正用户。总体来讲，**Session** 的安全性要高于 **Cookie**。

(3) **性能。** **Cookie** 存储在客户端，消耗的是客户端的 I/O 和内存，而 **Session** 存储在服务器端，消耗的是服务器端的资源。但是 **Session** 对服务器造成的影响比较集中，而 **Cookie** 很好地分散了资源消耗，就这点来说，**Cookie** 是要优于 **Session** 的。

(4) 时效性。Cookie 可以通过设置有效期使其在较长时间内存在于客户端，而 Session 一般只有比较短的有效期（用户主动清除 Session 或过期超时）。

(5) 其他。Cookie 的处理在开发中没有 Session 方便，而且 Cookie 在客户端是有数量和大小的限制的，而 Session 的大小却只以硬件为限制，能存储的数据无疑大了许多。

### 3.3.4 HTTPS 安全协议

前面在 Session 和 Cookie 中提到了 Session ID 和 Cookie 内容可能会被攻击者捕获的安全问题，实际上，类似的安全隐患在 HTTP 访问某些敏感页面时也会出现。比如用户通过网上银行访问自己银行账户信息页面或者涉及个人隐私的医疗信息，这些页面在响应给用户的过程中也会有信息泄露的可能性，所以关键信息需要在传输的过程中进行加密，这样，HTTPS 就出现了。HTTPS (Hypertext Transfer Protocol over Secure Socket Layer)，是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版，即 HTTP 下加入 SSL 层。HTTPS 的安全基础是 SSL。

用户在访问网上银行时经常可以看到浏览器地址栏的最右边有一把小锁的图标，而访问的网页的地址都是 https://，这说明用户正在使用 HTTPS 协议来进行 Web 访问，HTTPS 是如何工作的？图 3-18 展示了一个 HTTPS 进行安全通信的过程。

用户通过浏览器向服务端发出一个 HTTPS 的请求消息，服务器端在响应消息中包含一个服务器的公钥。公钥和私钥属于非对称密钥，是通过非对称加密算法得到的一对互不相同的密钥，用公钥加密的明文可以通过私钥来解密，用私钥加密的明文可以通过公钥来解密。公钥和私钥的名称差异只是相对于这两个密钥中哪个对外公开、哪个不对外公开而言的。顾名思义，公钥是对外公开的，任何人都可以通过服务器公布的公钥来加密发送给它的明文消息，而这

个消息也只有服务器才能解密，因为只有它知道私钥。浏览器在拿到公钥后就可以通过公钥来加密后面提到的对称密钥了。



图 3-18 HTTPS 的典型访问流程

浏览器收到服务器公钥后，会在本地产生一个（对）对称密钥，并用收到的服务器公钥对对称密钥进行加密，然后将加密后的对称密钥发送给服务器，服务器获取对称密钥后就可以在接下来的 HTTP 通信中对请求消息进行解密和对响应消息进行加密。对称密钥是什么？对称密钥之所以对称，是说发送方加密和接收方解密都使用相同的密钥，比如浏览器用对称密钥加密请求消息，服务器用对称密钥来解密。

为什么既要用到对称密钥又要用到非对称密钥？主要是因为对称密钥算法和非对称密钥在密钥分发和机器处理代价上的差异。对称密钥对文件内容进行加解密时处理效率高，处理时间短，消耗资源少，而非对称密钥则相反。在处理单个字符或者较少字节的文件时，对称密钥和非对称密钥的差别不大，但如果需要加密的文件是大块的文件，则对称密钥的优势就很明显了，所以在涉及 Web 页面或其他媒体内容传输等操作时，一般使用对称密钥加密会更加高效。但浏览器产生的对称密钥不能直接明文分发给服务器，否则攻击者获取后就可以窃取用户的敏感信息，这时非对称密钥就派上用场了。首先，因为对称密钥

是长度较短的一段随机字符，所以非对称密钥适合对对称密钥加密；其次服务器在开始只是把公钥分发给了浏览器，所以“公开的秘密”对攻击者而言自然也就不是什么秘密了。因此，有了非对称密钥和对称密钥的配合，HTTPS 就能保证整个通信的保密性。

### 3.3.5 HTTP 协议中的缓存技术

缓存是位于服务器和客户端的中间单元，主要根据用户代理发送过来的请求，向服务器请求相关内容后提供给用户，并保存内容副本，例如 HTML 页面、图片、文本文件或者流媒体文件。然后，当下一个针对相同 URL 的请求到来时，缓存直接使用副本响应 HTTP 请求，而不需要向源服务器再次发送请求。

如本章开头所描述的，使用缓存有三大好处。第一是减少响应延迟。缓存服务器距离用户更近，如果可以直接提供服务，响应时延将大大减少，使用户感觉 Web 服务器反应更快。第二是减少网络带宽消耗。当缓存直接使用副本为用户服务时，缓存与源服务器之间的通信链路带宽消耗将大大降低，提供服务的缓存越靠近用户，节约的网络资源越多，特别是对于拥塞程度较高的运营商骨干网来说更是如此。第三是降低源服务器负载。用户原本需要访问源服务器的大量请求都在缓存内直接得到服务，源服务器的响应次数大量降低。

HTTP 协议定义了各种各样的缓存控制方法，通过合理使用这些方法，适当配置网站缓存策略，既有助于网站服务更快、节省服务器负载和互联网的链接请求，又能最大程度地消除访问安全、时效性和用户统计方面所遇到的问题。缓存对于网站访问的改善是非常显著的，一个难以缓存的网站可能需要几秒去载入页面，而对比有缓存的网站页面几乎是即时显现，所以用户更喜欢速度快的网站并更经常访问。一般说来，缓存可以按照以下的基本原则工作：

- 如果响应消息的头信息告诉缓存不要保留副本，缓存就不会缓存相应内容。

- 如果请求信息需要源服务器认证或者涉及安全协议，相应的请求内容也不会被缓存。
- 如果缓存的内容含有以下信息，内容将会被认为是足够新的，因此不需要从源服务器重新获取内容。
  - 1) 含有过期时间和寿命信息，并且此时内容仍没有过期。
  - 2) 缓存内容近期被用来提供过服务，并且内容的最后更新时间相对于最近使用的时间较久。
- 如果缓存的内容已经过期，缓存服务器将向源服务器发出验证请求（通过 ETag 头信息或者 Last-Modified 头信息），用于确定是否可以继续使用当前内容直接提供服务。
- 在某些情况下（比如源服务器从网络中断开了），缓存的内容在过期的情况下也可以直接提供服务。
- 如果在响应消息中不存在用于判断内容是否变化的验证值（ETag 头信息或者 Last-Modified 头信息），并且也没有其他任何明显的新鲜度信息，内容通常不会被缓存。

以上原则告诉我们新鲜度和验证是确定内容是否可直接提供服务的最重要依据。如果缓存内容足够新鲜，缓存的内容就能直接满足 HTTP 访问的需求了；如果内容过期，而经源服务器验证后发现内容没有发生变化，缓存服务器也会避免将内容从源服务器重新传输一遍。

那么我们可以通过哪些方法来控制缓存或不缓存呢？

#### (1) HTML META 标签和 HTTP 头信息

HTML 文件的编写者会在文档的<HEAD>区域中加入描述文档的各种属性，这些 META 标签常常被用于标记文档不可以被缓存或者标记多长时间后过

期。META 标签使用很简单，但是效率并不高，因为能够读懂这个标记的浏览器只有少数几种，同时由于中间缓存几乎完全不解析文档中的 HTML 内容，所以也没有什么中间缓存（代理缓存和网关缓存）能读懂这个规则。如果要通过 META 标签来控制页面不缓存，一般情况下会在 Web 页面的<HEAD>区域中增加“Pragma: no-cache”的 META 标记。

### (2) 使用 Expires(过期时间)头信息来控制保鲜期

通常情况下，主要通过 HTTP 头信息来指示缓存和控制内容是否缓存。这些控制信息在 HTML 代码中是看不见的，一般由 Web 服务器自动生成，并在 HTTP 消息中进行标识。一个典型的 HTTP 1.1 协议响应消息的头信息看上去像这样：

```
HTTP/1.1 200 OK
Date: Mon, 18 Jul 2011 22:14:49 GMT
Server: Apache
Cache-Control: max-age=3600, must-revalidate
Expires: Mon, 18 Jul 2011 23:14:49 GMT
Last-Modified: Sun, 5 Jun 2011 16:38:21 GMT
ETag: "5e36-767-576d6f70c"
Content-Length: 1040
Content-Type: text/html
(空行)
HTML 代码
```

以上代码中包含了 Expires 头，指示 Mon, 18 Jul 2011 23:14:49 GMT 为过期时间。Expires 方式是 HTTP 控制缓存的基本手段，这个属性告诉缓存相关内容在多长时间内是新鲜的。过了这个时间，如果客户端向缓存请求这个内容，缓存就会向源服务器发送请求，检查文档是否发生了变化。几乎所有的缓存服务器都支持 Expires 方式。大部分 Web 服务器设置 Expires 方式有多种，最常用的是设置成一个绝对的时间值，比如将内容最后被修改的时间点加上一个特定的时间段（比如 1 个小时）所得到的时间值。

**Expires** 头信息对于控制静态图片文件的缓存特别有用，因为这些图片修改很少，可以给它们设置一个特别长的过期时间，这样会使得你的网站访问速度非常快。另外，它们对于控制有规律改变的网页也很有用，例如每天早上 6 点更新新闻网页，可以设置副本的过期时间也是这个时间，这样缓存就知道什么时候去取一个更新版本，而不必让用户不停地“刷新”了。

**Expires** 头信息属性值只能是 HTTP 格式的日期时间，其他的都会被解析成当前时间“之前”，即表示此内容已经过期，HTTP 的日期时间必须是格林尼治时间（GMT），而不是本地时间。

虽然过期属性非常有用，但它还是有些局限，首先是源服务器的时间和中间缓存的时间必须是同步的。如果彼此不同步，则会出现应该缓存的内容提前过期了，或者已经过期的结果没有及时更新。其次是过期时间的设置也存在一定的问题。如果设置的过期时间是一个固定的时间，而返回内容的时候没有更新下次过期的时间，那么之后所有访问请求都需要重新发给源 Web 服务器，这样反而增加了服务器的负载以及响应时间。

### (3) 验证

在 HTTP 1.1 中对缓存提出了验证的概念，验证的目的就是检验缓存内容是否可用。当中间缓存存在一个过期的缓存内容，并且对应的访问请求到达时，缓存应该首先向源服务器或者其他保存有未过期的缓存服务器请求验证来确定本地的缓存内容是否可用。这个过程就是一个缓存消息的验证过程。

HTTP 1.1 把这种验证后再决定是否返回消息内容的方式叫“有条件”的请求返回方法，这样可以避免从源服务器或其他缓存服务器获取整个内容的信息，从而减少网络流量。当源服务器生成了一个完整的响应消息时，它会附带一个验证信息，中间缓存在缓存内容时可以保存这个验证信息，当缓存内容过期以后，中间缓存可以使用它生成一个“有条件”的请求来向源服务器请求验证。而源服务器或者在与源服务器通信的路径上的其他缓存服务器（如果保存有未

过期的内容)在收到这样的请求以后就可以将请求中包含的验证信息与自己本地的验证信息进行比较。如果两个验证信息相等,那么返回一个带有特定状态码(比如 304 Not Modified, 表示内容未修改过)且消息主体内容为空的响应消息,在这种情况下就减少了网络流量;如果两个验证信息不相等就需要传输一个包含新内容的完整响应消息。

### (4) Cache-Control(缓存控制)HTTP 头信息

指定过期时间和验证是 HTTP 1.1 的基本缓存机制,也是缓存的隐含指令。但是在某些情况下,服务器或客户端可能需要给 HTTP 缓存提供显式的指令。因此,HTTP 1.1 介绍了 Cache-Control 响应头信息来让网站的发布者可以更全面地控制他们的内容,并对过期时间进行限制。有用的 Cache-Control 响应头信息包括如下几项。

**max-age**。缓存内容保持新鲜状态的最长时间。这个属性类似于过期时间,是基于请求时间的相对时间间隔,而不是绝对过期时间,单位是秒,即从请求时间开始到过期时间之间的秒数。

**s-maxage**。类似于 max-age 属性,它应用于共享缓存。

**public**。此属性标记认证内容也可以被缓存,一般来说,经过 HTTP 认证才能访问的内容是默认不能缓存的。

**no-cache**。强制将每次访问请求直接发送给源服务器,而不经过中间缓存进行前面提到的验证。这对那些需要在源服务器进行用户认证的应用非常有用,也适用于那些严格要求使用最新数据的应用。

**no-store**。强制缓存在任何情况下都不要缓存任何内容。

**must-revalidate**。告诉缓存必须遵循源服务器赋予的内容新鲜度。由于 HTTP 允许缓存在某些特定情况下返回过期数据,所以通过指定这个属性,源服务器可以告诉缓存,如果缓存内容处于过期状态,则在对访问请求进行响应

前必须到源服务器进行重新验证。

**proxy-revalidate**。**proxy-revalidate** 和 **must-revalidate** 基本相同，只是它不能应用于非共享的代理缓存。它允许在客户端缓存中保存那些经过权限认证的响应消息（包含有“**public**”来保证它们可以被缓存），在缓存内容没有过期之前，遇到相同的访问请求则可以返回缓存的数据而无须经过源服务器的验证，如果内容过期则仍需经过服务器重新验证。对于服务于多个用户的代理缓存来说，为了保证每个用户都是被授权的，仍需每次都去服务器进行验证（这样的授权响应同样需要使用 **public** 指令来允许它们能够被缓存）。

#### (5) Pragma HTTP 头信息

除了以上提到的几种典型缓存控制机制以外，还有一种使用 **Pragma** HTTP 头信息的方式。**Pragma** 属于通用头，用来包含特定的执行指令，这些指令可以适应于客户端、代理、网关、源服务器中的任何接收者，但是 HTTP 协议中认为 **Pragma** 指令规定的行为是可选的。

当“**Pragma: no-cache**”出现在请求消息中时，即使缓存设备中缓存了此请求响应所需的内容，也会直接将此请求转发到源服务器上。虽然在 HTTP 1.1 中提到了通过 **Pragma** 控制缓存的方法，但这主要是为了向 HTTP 1.0 兼容，因为支持 HTTP 1.0 的缓存主要还是通过这种方法来控制内容缓存的。HTTP 1.1 中主要还是通过前面讲到的 **Cache-Control** 头信息来控制缓存，所以协议要求当一个 HTTP 1.1 的请求从客户端发出时，既应该包含 **Pragma** 指令，也应该包含 **Cache-Control** 的控制指令，这样，请求从客户端发给源服务器的过程中，分别支持 HTTP 1.1 和 HTTP 1.0 的缓存设备都可以读懂指令的信息。如果发送请求的客户端本身只支持 HTTP 1.0，那么支持 HTTP 1.1 的中间缓存在收到请求消息后必须以 **Pragma** 中的指令来控制缓存。

很多人认为在 HTTP 头信息中设置了“**Pragma: no-cache**”后会让内容无法被缓存。但事实并非如此，HTTP 的规范并没有任何关于响应信息头 **Pragma**

属性的说明，而讨论的都是请求头信息的 Pragma 属性，即头信息由浏览器发送给服务器，实际上只有少数几种缓存服务器会遵循请求消息中的这个头信息。所以，在很多情况下，使用 Pragma 属性不一定管用。

## 3.4 Web Cache 技术实现关键点分析

引入 Web Cache 技术的主要目的是通过对内容副本进行缓存来满足后续相同的用户请求，使用 Cache 设备分担用户对源站点访问负载，从而提高 Web 站点的响应速度和用户访问并发量。在 CDN 系统中，Web Cache 多采用 3.2 节中介绍的反向代理工作方式，用于衡量该设备的关键性能指标，包括：用户访问并发数（即请求链接数量）、数据分发吞吐量（带宽）、丢包率、响应时间、服务命中率等，这些指标也是 Web Cache 设备实现时需要重点考虑的关键点。本节首先对 Web Cache 的关键性能指标进行说明，然后从如何提高 Cache 设备性能角度出发，对 Web Cache 设计和实现中需要重点考虑的几个方面进行简要分析。

### 3.4.1 Web Cache 关键性能指标说明

设计 Web Cache 时，需要结合具体应用场景和用户数量规模提出系统性能方面的需求，在实际运营中，一般最关心的性能因素包括以下几个方面。

#### 1. 并发量

采用 Web Cache，一方面能够大大提高 Web 站点并发用户数量，提高 Web 站点的响应速度，另一方面由于 Web Cache 本身硬件配置的高低限制了其处理性能，在设计 Web Cache 初期就需要规划好其能够处理的用户访问并发量，从而使 Web Cache 在完成部署后能够使用户并发量达到预期值。当用户访问峰值

超过 Web Cache 和 Web Server 的系统容量值时，可以采用虚拟排队等方式限制用户访问数量，以保证 Web 站点不会堵塞或者瘫痪。比如某站点规划的用户并发量为 5000，而每台 Web Cache 服务器所能承载的用户并发量为 1000，Web 站点所能承受的用户并发访问量也为 1000，则在建设 Web Cache 系统时至少需要部署 4 台 Web Cache 服务器，以达到站点规划的用户并发数量。

## 2. 吞吐率

Web Cache 的吞吐率是指单位时间内能够处理、转发的数据量大小，吞吐率是衡量缓存设备处理速度的重要性能指标。Cache 设备的吞吐率由 CPU 性能、网络接口卡性能、数据传输总线的大小、磁盘速度、内存缓冲器容量，以及软件对这些部件进行管理的有效程度共同决定。在实际应用中，Web Cache 所能达到的吞吐率除了受其本身处理性能的限制外，还要依赖于网络传输带宽速度和应用协议本身的传输效率。如果接入 Internet 的网络带宽较低，则 Web Cache 设备对外服务的吞吐率瓶颈就会出现在网络接入点。

在带宽保证方面，CDN 通常服务于多个内容运营商（ICP）、多种宽带应用，同时 CDN 所使用的基础网络又是无质量保证的 IP 网，为了保证 CDN 服务质量，需要在 CDN 网络规划中充分考虑为不同的 CDN 业务提供不同的网络带宽（带宽管理）和 QoS 保证的需求。

在 CDN 规划时，一方面需要合理部署并组织节点，避免广域垃圾流量的产生，另一方面建议在各 POP 内利用 L4 交换机所提供的带宽管理功能实现本地服务时的带宽管理策略。

## 3. 命中率

Cache 的服务命中率就是为用户提供内容服务时，如果该节点已缓存了要被访问的数据，可以直接为用户提供服务，这就叫做命中；如果没有的话，CDN 需要到内容源服务器取，就是未命中，需要回源。命中率 = 命中数 / 总请求数，这里所指的命中率是 Cache 服务 HTTP 请求命中率，典型的 Web Cache

命中率在 30% 到 60% 之间。另外一个类似的指标是字节命中率。Cache 命中率越高，说明 CDN 系统的缓存策略越合适，传递给源站的压力越小，因此缓存命中率是判断加速效果优劣的重要指标之一。

提高 CDN 服务命中率的方法有很多，比如增加 Web Cache 磁盘空间，从而可以缓存更多的内容对象，或者改用效率更高的内容更新算法使得 Web Cache 的单位服务时间内的命中次数更多。

### 4. 响应时间和丢包率

请求响应时间指用户发起内容访问请求到浏览器获取到内容之间的时间，是 Web 用户体验最重要的因素之一。响应时间主要由以下几个方面决定。

**DNS 解析时间：**DNS 解析是用户访问页面或者请求服务的第一步，通常此时间在 0.18~0.3 秒为正常，小于 0.18 秒为表现优良。

**建立连接时间：**指的是 IE 浏览器和 Web 服务器建立 TCP/IP 连接所消耗的时间，建立连接时间主要考量服务器硬件的处理性能，建立连接时间在 0.15~0.3 秒为正常，小于 0.15 秒为表现优良。

**重定向时间：**指从收到 Web 服务器重定向指令到 Web 服务器提供的第一个数据包之前的消耗时间，此时间通常小于 0.1 秒。

**收到第一个包时间：**指从 IE 浏览器发送 HTTP 请求结束开始，到收到 Web 服务器返回的第一个数据包消耗的时间。收到第一个包时间主要考量动态或回源的性能，此时间在 0.2~0.4 秒为正常。

**图片下载时间：**通常采用 150KB 大小的图片下载所使用的时间来评测 CDN 元素级加速性能，此时间在 1~2 秒为正常。

**页面总下载时间：**指页面所有内容全部到达浏览器的时间，总下载时间主要表示的是页面的总体耗时，不同类型的站点评定标准不同，通常此时间要求

在 10 秒以内。

**丢包率：**指的是 Web Cache 响应数据传输过程中所丢失的数据包数量占所发送数据包的比率，丢包率越高会导致重传的数据量越大，从而延长 Cache 响应时间。

### 3.4.2 内容存储机制

存储是 Web Cache 对缓存内容进行“持久化”的容器和载体，内容存储方案的设计将直接影响 Web Cache 的服务命中率、响应速度和投资成本。

从存储类型来看，主要关注存储的容量、成本和服务性能。存储容量越大，可缓存文件的数量也就越多，可以提高 Web Cache 服务命中率；成本越低，服务的附加值越高，越具备竞争能力。另外，还需要考虑所有缓存对象占用空间的平均值，也就是缓存对象大小的平均值，就 Web Cache 而言，缓存对象大小平均为 10~12KB。

目前常用的存储技术方案有三种：共享存储、本地附加存储（DAS）和分布式文件系统服务方式。其中，共享存储设备性能好、稳定和可靠性高，但投资成本较高；而采用分布式文件系统方式，可以基于廉价存储介质提供大容量、高性能、高可靠的存储服务，但是对部署实施技术要求较高，且该技术可能引入额外的网络时延，因此更适合流媒体等服务使用。通过配合文件分片预取等技术克服该缺点，本书的第 6 章会具体介绍该技术。考虑到 Web Cache 缓存对象大小平均为 10~12KB，最合适的存储技术是 DAS，通过在 Web Cache 服务器中内置大容量、高速的硬盘，在服务器内做 RAID 配置满足应用的要求。

从存储内容管理技术来看，主要关注存储内容的目录组织方式和管理方式。对于存储目录组织，目录层级越少，数据检索时间越短。比如在 Squid Web Cache 中，内容缓存目录设置为两级，以加快检索效率。此外，一些设备制造商采用了对象存储技术，在管理缓存内容时将所有缓存内容封装为内容对象，通过访

向主键来访问存储对象，从而屏蔽各种缓存内容类型、大小的差异，实现对 Web Cache 所有被缓存内容进行统一存储和统一操作，包括：内容的插入、查找、更新、删除等，并大大缩短了数据对象的检索时间，从而提高服务响应速度。

### 3.4.3 内容更新机制

在很长一段时间里，网站管理员们是很小心“缓存”这件事的，因为怕网上敌友难辨的代理会通过缓存“隐藏”掉他们的用户，让他们难以了解到底有多少人、什么样的人在使用他们的网站。事实上，无论管理员们是否喜欢，网上的代理服务器和用户的浏览器都可能自主启用缓存。幸运的是，他们不用对 CDN 的 Cache 有这种担心，CDN 只会为希望被缓存的网站服务，才不会傻到去缓存其他非客户网站的内容。

网站关于 Web Cache 缓存的另一点担心是怕给用户过期或失效的内容。这就是我们在这一小节要讨论的问题：如何选择合适的 Web Cache 内容更新机制。好的内容更新机制可以较为准确地判断或预测内容的冷热程度，从而实现高效新旧更替，在有限的存储空间中实现更高的命中率。

Web Cache 内容更新机制主要涉及两方面的内容：哪些内容需要缓存、缓存内容如何更新。这两方面的内容有些是在 HTTP 协议中进行定义（HTTP 1.0 和 HTTP 1.1），也有一些由 CDN 管理平台对 Cache 进行设置。

一般来说，Web Cache 会遵循以下基本规则。

1. 如果 HTTP 响应头信息告诉 Cache 不要缓存，那么 Cache 就不会缓存相应内容。
2. 如果对某内容的请求信息是需要认证或者安全加密的，Cache 也不会缓存相应内容。

3. 如果在 HTTP 响应中没有 ETag 或者 Last-Modified 头信息，Cache 会认为缺乏直接的更新度信息，默认该内容不可缓存。

4. 一个缓存的副本如果含有以下信息，Cache 会认为它是足够新的，会直接从缓存中送出，而不会向源服务器发送请求：

- 含有完整的过期时间和寿命控制的头信息，并且内容仍在生存期内。
- 浏览器已经使用过这个缓存副本，并且在同一个会话中已经检查过内容的新鲜度。

5. 如果缓存的内容副本已经旧了，Cache 将向源站服务器请求校验，用于确定是否可以继续使用当前副本继续服务。如果经校验后发现副本的原件没有变化，Cache 会避免从源站服务器重新获取副本。

根据经验，通常 HTML 文件、图片、css、xml、js、音频、流媒体等静态资源会被缓存，而动态地址、asp、aspx、py、jsp、php 等动态资源不被缓存。

关于在 HTTP 协议中进行定义的内容更新机制，在本章 3.3.5 节有详细讲述。这里我们重点讨论一下 Cache 与源站服务器之间的校验。

校验的概念是在 HTTP 1.1 中提出的，目的是检验缓存内容是否可用。HTTP 1.1 把这种验证后再决定是否返回消息内容的方式叫“有条件”的请求返回方法。校验的工作原理如下。

1. 源站服务器向 Cache 返回内容响应消息时，会附带一个验证信息，Cache 在缓存内容时保存这个验证信息。

2. 当有用户请求该内容时，如果 Cache 发现缓存内容过期，就使用验证信息生成一个“有条件”的请求来向源服务器请求验证。

3. 源服务器在收到这样的请求以后，将请求中包含的验证信息与自己本地的验证信息进行比较。如果两个验证信息相等，那么返回一个带有特定状态码

(比如 304 Not Modified, 表示内容未修改过)且消息主体内容为空的响应消息, 表示副本可以继续使用; 如果两个验证信息不相等, 源站服务器就会向 Cache 传输一个包含新内容的完整响应消息。

前面提到了“有条件的”请求, 其实所谓“有条件的”请求和普通的请求是一样的, 只是它包含了一个特殊的验证信息。并且“有条件的”验证包括正验证和负验证, 如果请求中要求服务器与消息附带的验证信息必须相等(使用请求消息的“*If-Match*”头)的则是正验证, 如果要求两者不相等(使用请求消息的“*If-None-Match*”头)的是负验证。

HTTP 1.1 协议描述的验证信息主要包括 *Last-Modified* 和 *Entity Tag* 两种, 分别对应了“*Last-Modified*”和“*ETag*”两个头信息, 其中“*Last-Modified*”属于实体头, *ETag* 属于响应头。通常, 不管是“*Last-Modified*”还是“*ETag*”, 这些验证信息都是由源服务器在产生实体内容或者内容更新时一起产生的, 然后在返回内容时一起返回给提出请求的 Cache。

在对内容是否发生变化进行验证时, 如果缓存内容在“*Last-Modified*”所标识的时间之后没有被修改过, 则认为它是有效的, 所以“*Last-Modified*”是一种有效的验证信息。一般“*Last-Modified*”头的值可以被包含在请求头“*If-Modified-Since*”和“*If-Unmodified-Since*”中。

“*Last-Modified*”在时间精度上有一定的缺陷, 因为 HTTP 协议的时间值是以秒为单位的, 如果一个响应消息在一秒内被修改了两次, 那么通过这个来验证某个内容是否发生变化就不准确了。在这种情况下可以用另外一个验证信息“*ETag*”。“*ETag*”可以由源服务器指定计算方法, 根据访问的内容来生成, 它是一个不透明的验证信息。请求消息在要求源服务器进行验证时, 可以将“*ETag*”的值包含在请求头“*If-Match*”、“*If-None-Match*”或“*If-Range*”中。

验证可以分为强验证或弱验证。强验证是要验证所访问内容的每一个字节

都没有变化，因为有任何的变化都会使相应的验证信息发生变化；而弱验证只验证所访问内容的语义有没有发生大的变化，验证信息只有在内容语义有明显改变时才会发生变化。弱验证信息可以应用于不要求精确一致的情况下，通常一个访问内容的修改时间可以被认为是弱验证信息。强验证信息应用比较广泛，比如使用 ETag，几乎所有的场合都可以用。如果用户提交的请求只需要验证实体内容的一部分是否发生了改变，那么应该使用强验证以获得正确的数据。

所有新一代的 Web 源服务器都对静态内容（比如文本文件）自动生成 ETag 和 Last-Modified 头信息，源站运营者不必做任何额外的设置。但是，源服务器对于动态内容（例如 CGI、ASP 或数据库生成的网站），并不知道如何生成这些信息，这就需要源站运营者自己考虑进行设置。像 ASP、JSP 或 PHP 源站吐出的动态内容，需要由 ASP、JSP、PHP 脚本程序控制 HTTP 协议返回头字段属性。对于那些频繁变化的内容，比如电子商务网站上关于某热销产品的评论，或者体育赛事的即时比分文件，源站会告诉浏览器不要缓存，但会通过设置 max-age 告诉 Cache 缓存 1 秒，这样算是在动态和缓存之间取一个折中吧。

网站的开发者们应该尽量让自己的网站缓存友好，这里不妨再赘述几条对于源站的建议。

1. 尽量保持内容 URL 稳定：如果在不同的页面上提供相同的内容，应该使用相同的 URL，这使网站内容便于被缓存。
2. 使用公共库存放每页都引用的元素，可以提高缓存效率。
3. 对于不经常改变的图片和其他页面元素，给 Cache-Control: max-age 属性设置一个较长的过期时间，这样可以尽可能多地利用 Cache 的缓存。
4. 对定期更新的内容设置一个 Cache 可识别的 Cache-Control: max-age 过期时间。

5. 只在必要的时候使用 **Cookie**, **Cookie** 是非常难被缓存的, 如果使用 **Cookie**, 控制在动态网页上。
6. 尽量避免使用 **POST**, 除非万不得已。**POST** 模式的返回内容通常不会被 **Cache** 缓存, 使用 **GET** 模式会好很多。

### **3.4.4 Web Cache 协议优化**

对 **Web Cache** 性能的优化除了可以从 **Cache** 存储机制、内容更新机制入手外, 还可以考虑对 **Web Cache** 的传输协议进行优化, 能够在增加数据传输吞吐率的同时提高 **Cache** 设备的服务响应速度。**Web Cache** 在提供内容分发服务时最常采用的是 **HTTP** 协议, 其传输优化可以分两层处理。一层是针对 **HTTP** 协议实施的优化, 利用 **HTTP** 协议本身的一些优化方法, 如 **HTTP** 连接聚合、**gzip** 压缩等; 另一层是基于 **TCP** 传输协议的优化, 采用协议压缩、减少广域网会话交互等方法, 相关内容将在本书 7.2 节进行详细介绍。

#### **1. HTTP 连接聚合**

**HTTP** 连接聚合的原理是, 把多个短连接转换成一个长连接, 从而减少连接。**HTTP** 连接聚合可以大大减少服务器频繁开启和关闭 **TCP** 连接处理所带来的资源消耗。尤其对于 **HTTP** 协议, 大部分的 **HTTP** 协议都采用短连接方式处理, 而服务器的 **TCP** 堆栈对 **TCP** 的每次连接建立都需要分配相应的资源, 在很短的时间内, 又要对这些资源进行回收, 造成性能的下降。

在本章 3.3.2 中有关 **HTTP** 连接聚合工作原理的详细讲述, 尤其是“带流水线的持续连接”, 能够高效地在一个 **TCP** 连接中聚合多个 **HTTP** 连接。

#### **2. HTTP gzip 压缩**

**gzip** 现今已经成为 **Internet** 上使用非常普遍的一种数据压缩格式, 或者说一种文件格式。**HTTP** 协议上的 **gzip** 编码是一种用来改进 **Web** 应用程序性能的

技术。平均压缩比可以达到 40%以上，甚至高达 80%，效果还是非常显著的。

同时，目前大多数的 Web 服务器软件均支持 HTTP gzip 压缩技术，如 Apache、IIS、Tomcat 等，均提供 gzip 配置选项。而大多数的浏览器，如 IE、Firefox、Chrome 也都支持 gzip 解压格式。实现 HTTP 传输内容压缩的配置方法主要是当 Web Server 响应客户端时回传采用 gzip 格式对文本文件进行压缩，并将 HTTP 头信息 Content-Encoding 字段设置为 gzip 属性。采用 gzip 压缩方法前后，HTTP 请求和响应流程如图 3-19 和图 3-20 所示。

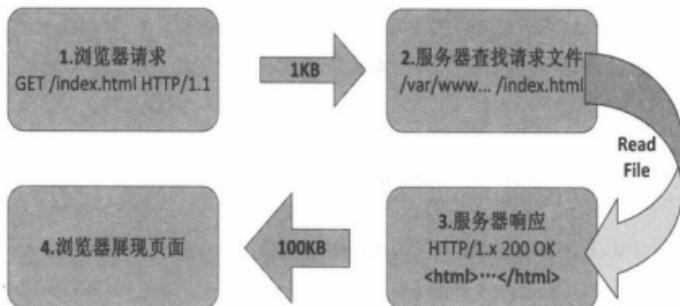


图 3-19 未采用 gzip 压缩的 HTTP 请求与响应流程

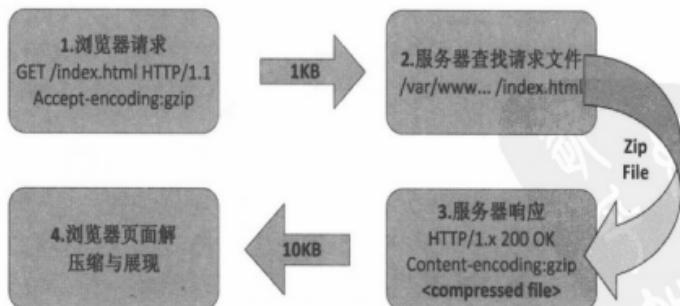


图 3-20 采用 gzip 压缩处理的 HTTP 请求与响应流程

服务器收到浏览器 HTTP 请求后，以文本文件形式返回给浏览器，没有经过任何压缩处理，门户网站 index.html 文件大小通常为 100KB 左右。

服务器收到浏览器 HTTP 请求后，查找到浏览器请求的 index.html 文件，经过 gzip 压缩方法处理，以压缩文件形式返回给浏览器，门户网站 index.html 文件文本大小约为 100KB，经过 gzip 压缩后大小通常为 10KB 左右。

由于网页中的图片、音乐、视频等媒体类型的文件已经被压缩处理过，如果再对这些类型的文件进行压缩不会取得很好的压缩效果，却要消耗一定的压缩时间，大多数情况需要压缩的文件是网页中出现最频繁的 HTML、CSS、JavaScript、XML 等文件，这类是本身是没有经过压缩的文本文件，可以取得较好的压缩效果。

### 3.4.5 Web Cache 安全实现机制

设计 Web Cache 时需要考虑的安全实现机制主要包含以下几个方面。

#### 1. 访问控制

访问控制是很多具备代理（Proxy）功能的 Web Cache 的一项重要功能，基于对用户或者被访问主机的 IP 地址、端口号、域名或者主机名、URL 样式执行允许通过或者屏蔽、过滤等操作，从而控制哪些用户可以访问哪些外部网络资源，以及哪些外部系统可以访问内部网络的哪些资源。常用的访问控制功能包含：

（1）允许访问控制列表（ACL）中的 IP 地址用户访问 Internet，拒绝其他 IP 地址的用户访问外网。

（2）在 ACL 中设置某个具体 IP 地址，创建拒绝规则来阻止这些 IP 地址的用户访问外网。

（3）设置屏蔽对指定站点、域名、URL 实施过滤，拒绝内网用户访问。

- (4) 限制内网用户可访问外网的时间。
- (5) 赋予某些用户特殊的访问权限，比如赋予管理员用户访问管理平台、监控系统、数据库等平台的权限。
- (6) 控制用户访问外网服务器的端口号，比如允许用户连接到任何非特权端口（1025~65535）。
- (7) 其他功能，比如减缓或加速内容发送、用户身份认证等。
- (8) 防止网络攻击。

## 2. 病毒防护

Web Cache 中的服务器也面临着病毒侵扰的问题，尤其是含 Nimda、红色代码等病毒的流量大量存在于 Internet 中，不但会干扰服务器的运行，还占据了宝贵的带宽。因此在部署 Web Cache 时也需要为缓存服务器操作系统做好病毒防护措施，如安装病毒扫描工具、杀毒软件等，同时某些扫描也可以针对 webcache response 做病毒安全扫描。比如 SquidClamav 可以针对 HTTP 数据流中的任何信息进行扫描，如 URL、HTTP 响应信息头、HTTP 响应内容主体，当 SquidClamav 扫描到病毒后执行 HTTP 请求重定向到告警页面，并对病毒页面进行查杀或者隔离，以避免将病毒数据发送到用户。

## 3. 网络安全防护

目前，在互联网上存在着各种各样的恶意攻击，如：DDOS、SYN 等。它们不但对在互联网上的数据包进行抓取、分析、破解，同时还采用恶意抢占带宽的方式耗尽带宽，使专业的互联网设备因为承受不了沉重的数据请求而被迫退出服务，而对 ISP 供应商们的服务承诺更无从谈起。因此，对于网络安全方面的隐患也必须考虑。针对 Cache 设备安全防护，需要在多个方面做好安全防护措施，如操作系统层、应用层安全加固措施，关闭一切不需要的服务和端口，定期做好安全漏洞扫描（如使用 Nessus 扫描工具，以及 Snort 入侵检测等），

以便及时发现系统安全漏洞并及时予以修复。

**Web Cache** 本身具有很强的抗攻击能力，它不但可以为网络中的服务器提供负载均衡功能，还提供了以下防攻击手段来保护网络中的服务器。

- (1) 通过对无效连接的管理来防止使用没有开放的服务进行攻击。
- (2) 实现源路由的跟踪，防止 IP 欺骗。
- (3) 不用 Ack 缓冲应答未确认的 SYN，防止 SYN 风暴。
- (4) 防止连续和接管的攻击。
- (5) 不运行 SMTPd、FTPd、Telnetd 等易受到攻击的进程。

以上这些固有的安全特性及抗攻击能力可大大提高网络的安全性。

#### 4. 内容加密

在设计 **Web Cache** 时需要考虑支持对经过 DRM（内容数字版权加密保护技术）加密的文件的分发和传输，可以透明传输这些经过 DRM 加密后的媒体文件。

例如，对于微软流媒体格式文件的加密，要求 Cache 支持微软的 DRM 能力，对需要这一功能的 SP 提供服务。

### 3.5 开源 Web 缓存代理软件——Squid

20 世纪 90 年代，美国科罗拉多大学博尔德分校开展了 Harvest 项目的研究，该项目的主要研究成果包括两款著名的软件，其中之一是商业软件 NetApp NetCache，而另一个则是大名鼎鼎的开源软件 Squid。Squid 遵循 GPL 许可证，

其 1.0.0 版本发布于 1996 年 7 月，随着多年的演进和发展，Squid 已经成为当前应用最为广泛的 Web 代理缓存软件。

Squid 的部署方式非常灵活，被广泛地用于 Web 内容缓存。网站通过在数量众多、更靠近用户的站点上部署 Squid，利用它复制和存储来自内容源站点的互联网对象（例如网页数据），当多个用户对同一内容发起访问请求时，可以直接在 Squid 代理缓存站点上获得其所需的内容而不必再访问源站点，提高用户访问数据的速度，缓解了源站点的访问压力。同时 Squid 还实现了代理服务功能，能够模拟用户的真实行为，例如当缓存站点中没有用户所需的内容时，Squid 可以以用户客户端的名义向源站点发出请求，并从源站点获得相关数据再转发给用户。

Squid 主要实现了以下功能。

#### (1) Web 代理

部署了 Squid 的代理缓存站点位于用户浏览器和源站点之间。在用户发出请求后，浏览器不再直接访问源站点取回网页数据，而是向 Squid 服务器发出请求，再由 Squid 服务器作为代理向源站点发出请求并将取回的数据回传给浏览器。当前，Squid 全面支持正向代理、反向代理、透明代理等多种工作方式。

#### (2) 内容缓存与加速

Squid 软件具有良好的缓存能力，它不断地将从源站点取得的数据存储到代理缓存站点本地并对其进行有效的管理。当浏览器请求的数据已经在 Squid 服务器上存在并且是保持最新更新状态的时候，Squid 将不再向源站点索取相关数据而直接从本地存储中向用户反馈其所需的内容，显著提高浏览速度和效率。

#### (3) ACL 访问控制

Squid 具有强大的访问控制列表功能，它能够实现对内部网络/外部网络的

访问控制，例如只允许特定主机访问其专属的网络而不能访问其权限范围之外的其他网络。另外，**Squid** 可以通过 ACL 访问控制功能限制网络流量，例如限制某些特定协议数据（例如 P2P）的传输等，从而提升网络整体安全性和性能。

### (4) 用户认证

**Squid** 在提供代理和缓存功能时，可以通过配置用户身份认证程序和用户账户信息，限定能够享用代理缓存服务的用户范围。同时，**Squid** 的配置文件具有非常详细的选项，例如可以指定用户身份认证的有效时间、是否区分大小写、高级用户身份认证等。

### (5) 日志

**Squid** 具有非常强大的日志功能，包含了很多详尽记录系统工作情况的日志文件。例如日志文件中除了记录服务器进程的运行情况外，还记录了用户的访问情况、缓存的存储状况、缓存的访问情况等内容。**Squid** 日志实时、准确地体现了**Squid** 服务器的运行状态，并为优化**Squid** 的性能提供了重要依据。

**Squid** 具有与商业软件完全相同的工作原理，它功能完备、配置便捷、文档丰富，已经被很多网站和 CDN 运营商所采用并取得了非常好的效果。但是**Squid** 在设计上也存在一定的局限，例如当它工作在反向代理方式时，用户发出的请求将立即被**Squid** 转给后台服务器，这就意味着在用户浏览器和后台服务器之间建立了一个长连接，该连接在请求完成之前是一直存在的。这种同步模式限制了**Squid** 能够支持的并发连接数目，在用户请求的数据规模较大或者用户端的网络速度较慢时，**Squid** 服务器将承受较大的压力。

**Squid** 官方网站的网址是 <http://www.squid-cache.org/>，网站上提供了最新版本的**Squid** 的下载链接和相关的软件手册。

## 第4章 集群服务与负载 均衡技术



- 4.1 服务器集群技术
- 4.2 Cache 集群协同交互方法
- 4.3 负载均衡技术的实现
- 4.4 开源负载均衡软件



CDN 的缓存技术能够将用户访问的内容缓存到网络边缘处，以实现用户的就近服务，改善服务质量。但是，当一台 Cache 设备无法满足服务的响应需求时，就要考虑通过一组服务器形成 Cache 集群为用户提供服务。本章首先对服务器集群（Cluster）的基本概念进行介绍，然后重点对 Cache 集群的协同工作方式和负载均衡（SLB）技术进行系统性的讲解，最后对有代表性的负载均衡开源技术——LVS 和 Nginx 进行介绍。

## 4.1 服务器集群技术

### 4.1.1 集群的基本概念

服务器集群是指将很多彼此相互独立的服务器通过高速网络连接在一起，形成一个并行或分布式系统。这些服务器运行一系列共同的程序，向外提供单一的系统映射，提供一个服务。从外部来看，整个集群就像是一台具有统一输入、输出的服务器一样。

相比单台服务器，服务器集群具有以下优势。

#### (1) 提高性能

对于一些计算密集型应用，需要非常强大的计算处理能力才能完成任务，而单台服务器是难以胜任这种计算任务的，只能利用集群技术将多台服务器的计算能力整合起来，通过并行计算获得超高的计算性能。设计良好的网络架构、高效的处理算法都是提高集群性能的重要手段。

#### (2) 降低成本

单台服务器为了获得较高的性能，必须要在软硬件上具有足够的支撑能力。

比如一些大型机设备，设计了专用的高性能处理器、硬件板卡、接口协议等，这也带来了极高的设计和制造成本。服务器集群通过多台服务器的并行处理可以提供高性能的计算能力，而组成这个集群的单台服务器并不一定具有非常高的软硬件配置和计算性能，在性价比方面具有优势。

### (3) 提高可扩展性

系统的扩展性是应对计算需求动态变化的必然需求，单台服务器需要通过改变软硬件的配置才能改变自身具有的能力，而集群只需调整集群中的网络节点数量即可做到集群规模及相关计算能力的缩放。另外，单台服务器软硬件设备的升级通常需要离线进行，而集群则可以在线完成服务器节点的增删操作，对于集群承载业务的连续性不会有影响。

### (4) 增强可用性

集群中的服务器可以在集群管理系统的统一管理下，实现不同服务器之间的负载均衡、容错备援等高可用机制。当部分服务器发生故障不能工作时，集群可以在很短的时间内完成故障切换，将故障服务器承担的任务转移到正常服务器上，将系统停运时间降低到最小，减少故障损失。

## 4.1.2 集群的分类

根据用途的不同，我们把服务器集群分为如下几类。

### (1) 计算集群

服务器计算集群通常被用于承载计算密集型任务，而并不用于 I/O 密集型的应用场景（例如 Web 服务或者数据库）。这类计算集群是以并行计算为基础的，它对外而言就好像是一台性能强大的超级计算机。计算集群又可根据其中节点之间耦合的紧密程度做更进一步的细分。例如，如果某个计算任务需要在节点之间进行频繁的通信，那么这意味着节点之间最好由专用的网络进行连接，

而且集群密度可能很高以尽量减少通信开销。实现这类场景通常使用 Beowulf 这样的高性能计算集群；而与之相对应的，如果某个计算任务只需使用较少的服务器节点，或者在服务器之间几乎没有通信的需求，则可以使用网格计算集群实现。

传统意义上，具有紧密耦合架构的计算集群是该领域研究的重点，它们通常被用于运行并行计算程序以应对具有极高性能要求的“超级计算”任务，例如复杂的科学问题。同时，对于计算集群而言，人们开发了大量诸如 MPI (Message Passing Interface)、PVM (Parallel Virtual Machine) 的中间件，以使得计算集群上运行的程序能够获得跨集群的可移植性。

### (2) 负载均衡集群

负载均衡集群的关键在于能够使多台彼此互联的服务器共同分担计算任务，即任务负载可以在集群中被尽可能地平均分配到多台服务器上处理，从而避免出现集群中某几台服务器超载而其他服务器闲置的情况，有效地改善集群性能。

集群中的负载通常包括计算任务处理负载和网络 I/O 流量负载两类。对于计算任务处理负载，负载均衡集群能够使用同一组计算应用程序为大量用户提供服务，其中每个节点都可以承担一定的处理负载，并且可以在节点之间动态分配负载，实现负载均衡；对于网络 I/O 流量负载，当网络服务程序接收了高入网流量而无法及时处理时，相关流量会被分发给在其他服务器节点上运行的网络服务程序，同时可以根据每个节点上不同的可用资源或网络环境进行优化。

### (3) 高可用集群

高可用集群中的部分服务器发生故障时，集群管理系统将及时发现故障并将由该部分承载的任务重新分派到其他正常工作的服务器上。其关键在于能够在系统高速运行的过程中尽可能快速地对系统故障做出响应。为了实现这一点，集群通常会在多台服务器运行冗余节点和服务，并用来互相跟踪，例如通过心

跳（heartbeat）机制探知服务器的工作状态。如果侦测到某个服务器节点发生失效，那么其替补者会在非常短的时间内取代它。因此，对于用户而言，高可用集群永远不会停机。

高可用机制能够有效地提高集群运行的可持续性，能够在集群发生故障时确保用户体验的一致性。同时，高可用集群中单台服务器的高可用性要求被降低，有助于降低系统成本。

在上述集群的分类中，高可用机制是确保集群运行质量的重要方法，因此通常是交融在其他两类集群中一起被实现的。而计算集群和负载均衡集群也有一定的类似性，例如都是在多个服务器节点间分发计算负载，但是它们的最大区别在于计算集群更多的是用于承载跨多个节点的并行计算程序，而负载均衡集群中每个服务器节点上通常是运行独立的软件系统，而与其他节点少有通信。因此，从这个意义来看，CDN 的节点系统可归类为一种典型的负载均衡集群系统。

### 4.1.3 集群的系统结构

典型的服务器集群的系统结构如图 4-1 所示。

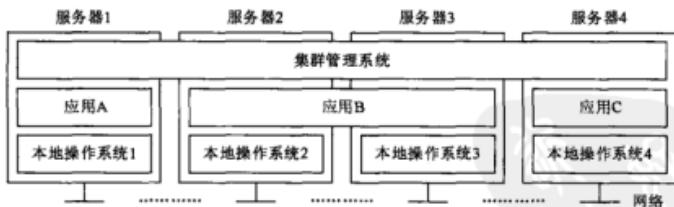


图 4-1 服务器集群系统结构示意图

如图 4-1 所示，服务器集群的系统结构主要分为 4 个层次。

(1) 网络层：网络是构成集群的基础，因为构成集群的多台服务器是通过网络互联的。网络层的关键技术包括网络互联结构、网络通信协议、信号传输技术等。

(2) 节点服务器操作系统层：集群中的各台服务器是集群计算能力的基本单元，它们具有一定的自治能力，能够独立完成集群分配到本地的任务，其关键技术主要包括高性能服务器架构、高性能操作系统内核技术等。

(3) 应用层：应用层由执行负载任务的软件构成，可在集群管理层的干预下实现相应应用功能。其关键技术包括并行程序开发环境、各类解决任务负载的串/并行应用等。

(4) 集群管理系统层：集群管理系统层是服务器集群的核心组件，是协调集群资源使之能够高效协同完成任务的关键。它的主要任务是对集群内的服务器资源及其上运行的任务进行管理和调度，以实现集群内负载的均衡，从而避免个别节点成为瓶颈，最大程度地发挥集群的整体性能。对于不同的集群类型，其集群管理系统的功用也有所不同，在计算集群中，其主要用途是为应用提供并行计算环境，而在负载均衡集群中，其主要用途是调度各个应用到合适的服务器上运行。

相比较其他层次，集群管理层是集群所特有的功能与技术的体现。正是由于集群管理系统对多台服务器的统筹、协调、管控和有机组织，才使之能够对外以一台服务器的形象展现，才产生了“集群”的概念。集群管理层的完善程度，直接决定着集群系统的性能、易用性、稳定性、可扩展性等诸多关键指标。

#### 4.1.4 CDN 负载均衡集群

CDN 的缓存服务器是典型的负载均衡集群系统，我们在本书的第 3 章了解到内容缓存技术，其核心思想是在内容服务器和用户之间架设缓存服务器，将用户最近访问过的媒体数据缓存在离客户较近的地方，以供后续重复访问使用。

当用户访问量较大时，单台缓存设备在处理繁重的内容分发任务时会在处理能力、吞吐能力等方面形成严重的性能瓶颈。在这种情况下，缓存服务器集群就是解决相关问题的有效手段。在用户和内容服务器之间部署缓存服务器集群，能够充分利用集群中各个节点形成的强大计算能力，同时各节点可以被并行访问，能够有效改善系统吞吐率。除了性能的改善，多台缓存服务器构成的集群在存储容量等方面的优势，使其在提高用户请求命中率方面有很大的提升。相关的系统示意图如图 4-2 所示。

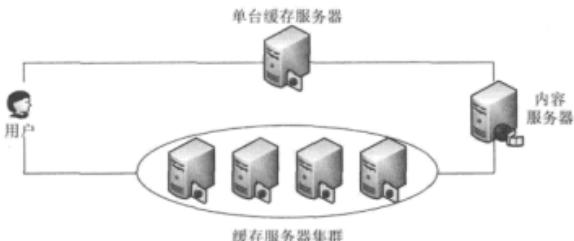


图 4-2 缓存服务器集群的部署位置

对于 CDN 系统中的缓存服务器集群而言，各台服务器之间需要知道彼此的内容缓存情况以提高访问效率，因此集群内部的协同交互是非常重要的。另外，在对外提供服务时，针对各台服务器上的处理能力进行必要的负载均衡从而合理分担用户请求也是改善用户访问体验的关键。

## 4.2 Cache 集群协同交互方法

Cache 服务器集群内部协同交互的主要目标是在各个服务器节点间建立良好的通信通道，以及时沟通服务器上的内容缓存情况，通过集群中服务器间的协作为用户提供良好的服务体验。

**Cache** 服务器集群之间的通信可以分为松散耦合和紧密耦合两大类。其中基于网络消息的松散耦合的 **Cache** 通信协议包括：ICP、HTCP、Cache Digest、Cache Pre-filling 等，采用特定数据结构管理的紧密耦合的 **Cache** 通信协议以 CARP 为代表。下面我们对这些通信协议做扼要的介绍，重点说明其工作原理和优缺点，有兴趣的读者可查找相关资料详细了解技术细节。

#### 4.2.1 ICP

RFC 2186 ICP（Internet Cache Protocol）定义了一种轻量级的消息格式，被用于在 **Cache** 服务器之间互相查询 Web 资源信息，以确定当前被请求的资源是否存在于其他服务器上。当一台 **Cache** 服务器向其邻居发出 Web 对象（主要是 URL 信息）查询请求时，接收到查询请求的服务器通过反馈包含了“命中（hit）”或者“失效（miss）”信息的 ICP 应答说明被查询的对象是否保存在自己这里。

ICP 普遍是基于 UDP 协议实现的，虽然这并不是协议本身的规定。这主要是考虑到 ICP over UDP 的方式更加适合 Web Cache 这类应用，因为一个 ICP 查询的请求和应答必须在非常短的时间内（例如一两秒之内）完成交互，这样才能保证 **Cache** 服务器能够快速从邻居服务器上获得 Web 对象。如果使用 UDP 协议的请求服务器在接收应答时发现错误，那么可能意味着服务器之间的网络发生了拥塞或者是被断开，而相应的应答服务器也就不再适合被选为请求服务器的邻居。另外，相比较支持可靠传输的 TCP 协议，UDP 协议包的规格更小，也更符合 ICP 轻量级消息格式的要求。当然，除上述这些适用性之外，ICP 也必须考虑 UDP 协议带来的安全问题。

需要注意的是，ICP 的请求和应答中并不包含与资源相关的 HTTP 头信息，例如访问控制、缓存指示等。因此尽管 **Cache** 服务器在通过 HTTP 协议获取相关资源前已经查询过了资源的可用性，但是仍旧可能发生 **Cache** 失效的情况（例如存在于 **Cache** 服务器上的某些 Web 对象并不允许邻居服务器对它的访问）。

ICP 已经有较长时间的历史了，当前已经发展至 ICP v2，大部分现有的 Cache 服务器也都能够以一定形式实现对 ICP 的支持。

### 4.2.2 HTCP

HTCP（Hypertext Caching Protocol）是用于发现 HTTP 高速缓存（Cache）服务器和缓存数据的协议，在 RFC 2756 中定义。它能够管理一组 HTTP Cache 服务器并监控相关的缓存活动。

HTCP 的运行机制与 ICP 类似，都是通过向邻居服务器发出查询请求并获得应答来反映 Web 对象在集群中的缓存情况。但是，与 ICP v2 只能包含 Web 对象的 URI 不同，HTCP 请求和应答中可以包含有完整的 HTTP 头文件的信息，这使得当后续的 HTTP 请求需要访问同样的资源时，HTCP 能够做出更精确的应答。另外，HTCP 采用了可变长度的消息格式，并扩展了 Cache 管理功能，例如能够监控远程 Cache 的增删、请求立即删除、发送 Web 对象提示（例如被缓存对象的第三方存放位置以及那些不可被缓存或者不可用的 Web 对象的第三方存放位置）。

一个 HTCP 消息包括了三个部分：头信息、数据、认证。其中，头信息部分主要用于告知消息的长度和协议的版本；数据部分主要用于描述具体的 HTCP 消息，其中包含各种 HTCP 操作码和操作数据，例如用于测试缓存应答是否存在的 TST、用于告知邻居更新缓存目标头部信息的 SET、用于告知邻居从其 Cache 中清除目标的 CLR、监控邻居 Cache 活动的 MON 等；认证部分是可选的，主要用于体现事务操作的认证信息。

在具体实现中，HTCP 消息可以通过 UDP 报文或者 TCP 连接传送。其中 UDP 报文是必须被支持的，而 TCP 协议则主要用于进行协议的调试。HTCP 一般选用 HMAC-MD5 共享密钥认证，因为如果不使用密码认证，该协议比较容易受到攻击。

### 4.2.3 Cache Digest

Cache Digest 的出现主要是为了解决 ICP 和 HTCP 协议在使用过程中的网络延迟和拥塞问题。Cache Digest 并不采用基于请求-回答模式的带内查询方法，而是在服务器之间建立对等关系，即每台 Cache 服务器上都保存了它的所有邻居的缓存信息摘要。当接收到用户的 Web 对象访问请求时，Cache Digest 直接在本地的 Cache 内容摘要中检索，并获知该被请求的 Web 对象 URI 是否在某个邻居 Cache 里。

相比较 ICP 和 HTCP，Cache Digest 实际上是一种空间换时间的思路，即利用 Cache 服务器本地的存储空间保存邻居服务器的 Cache 内容信息，从而节省每次查询过程中在网络上的传输延迟。对于 Cache Digest 而言，摘要算法的选择格外重要，考虑到摘要文件的传输时延和存储开销，所以摘要文件的规格要尽量小，但这样也可能导致查询的精度降低，算法的选择是一个平衡取舍问题。

使用 Cache Digest 需要特别考虑安全问题。比如服务器 A 生成了一个摘要文件，并将其发送给与之对等的服务器 B，B 会根据这份摘要去不同邻居获取内容。也就是说，A 的这份摘要文件能够直接影响流向 B 的网络流量，如果摘要文件被恶意篡改，B 就会面临严重的安全问题。为降低这种风险，可以考虑只允许服务器采用“拉”的方式主动从其他服务器上获得摘要文件，而不是被动地接收由别的服务器“推”来的摘要文件。

### 4.2.4 Cache Pre-filling

Cache Pre-filling 实现的是一种推送 Cache 内容的机制，它能够很好地应用在 IP 多播网络上。它使得预先被选定的资源能够被同时插入到目标多播组中的所有 Cache 服务器中，从而实现集群中各台服务器保存内容的同步。

当前，Cache Pre-filling 技术已经多有实现，特别是应用在卫星通信的场景中，它最大的优点在于能够同时向多个分布的地面上卫星接收器高速传输大容量

数据，从而在网络传输速度不高的情况下极大地改善数据访问体验。但总体而言，Cache Pre-filling 当前还缺乏统一的标准，各相关厂商普遍都是基于实际场景需要各自开发专用设备实现这类推送 Cache 技术，或者是在某些通用 Cache 设备上增加相关的专用模块。

#### 4.2.5 CARP

CARP ( Cache Array Routing Protocol ) 本质上是一个分布式的缓存协议，通过建立哈希函数用于划分 Cache 服务器集群的 URL 空间。CARP 的核心是为集群定义了一张 Cache 服务器阵列成员表，以及一个用于向 Cache 服务器上分发缓存 URL 信息的哈希函数。CARP 为用户提供 Web 对象 URL 的获取路径，该路径是根据服务器阵列成员的名称和相应的 URL 内容通过哈希操作而产生的，这就意味着对于任何特定的 URL 请求，都能够准确地知道其所需的信息存储在阵列中哪个 Cache 服务器上，而不用理会这是一个此前刚刚被请求并被缓存的信息，还是首次被点击需要传递和缓存的信息。

CARP 通过哈希算法将用户对 URL 的请求准确路由到服务器阵列中的任一成员上，消除了阵列中重复的缓存数据，实现了对 Cache 资源的高效定位。因为无须考虑更多的不可逆性和加密要求，CARP 采用的算法非常简单，具有极高的性能。

由于 CARP 被很多商用系统使用，我们来详细了解一下它的工作过程。首先，对 Cache 服务器阵列中的各个成员的名称字符串实施逐位左循环移动若干位的操作形成其对应的哈希 ID，这些信息将被补充到阵列成员表中，并在每台服务器中保存；其次，对众多的 URL 字符串采用类似的算法进行操作，获得相应的 URL 哈希值；然后，每个 URL 哈希值都要和各台服务器的哈希 ID 做异或操作并乘以一个常数，其得到的乘积再逐位左循环移动指定位数以获得相应的分数；最后，CARP 对分数值进行大小比较，并最终将 URL 对应的内容分配到与之操作具有最高分数的服务器上。在某台 Cache 服务器收到用户的 URL

请求时，它会将该 URL 对应的 URL 哈希值与本地保存的集群阵列成员表中各台服务器的哈希 ID 进行哈希运算，再根据分数的高低判定该 URL 内容应该在哪台服务器上。

考虑到不同 Cache 服务器的处理能力差异，CARP 在哈希算法中还引入了负载因子，以明确阵列中不同服务器能够承担的工作负载并为其分派合适的 URL 缓存内容，实现性能的优化。

CARP 是一种紧耦合的 Cache 通信方式，相比较基于网络消息的松散耦合的 Cache 通信，CARP 的主要优势体现在：

- (1) 无须资源查询的请求和应答过程，避免网络影响，降低传输开销。
- (2) 消除了重复缓存数据，系统中每个 URL 内容只保留一份，节省空间。
- (3) 具有更好的扩展性，因为无须和众多其他 Cache 服务器进行网络交互。
- (4) 可以灵活增删服务器节点，哈希算法的应用能够最小化节点数量变化导致的数据分布影响。
- (5) 能够确保所有的 URL 数据都能够有效地缓存在系统中。

### 4.3 负载均衡技术的实现

负载均衡（Server Load Balance），含义是将负载（工作任务）进行平衡、分摊到多个操作单元上进行执行，从而实现整个系统共同完成任务。

负载均衡提供了一种廉价又有效的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。根据负载均衡实施目的的不同，可以把负载均衡分为两种。一种是任务分担，大量的并发

访问或数据流量分担到多台设备上分别处理，每台设备都完成一个相对完整的请求响应过程；另一种是协同计算，把一个重负载的计算任务分担到多台设备上做并行处理，各台设备处理结束后，将结果进行汇总计算。

如前文所述，负载均衡的关键在于能够使任务负载在集群中的服务器上被尽可能均衡地承载，避免出现集群中某几台服务器超载而其他服务器闲置的情况。在实现了负载均衡的集群中，多台服务器通过网络连接在一起，通过在集群前端部署负载均衡设备，根据预先配置的均衡策略将用户请求在集群中分发，并维护服务器的可用性。负载均衡实现的示意图如图 4-3 所示。

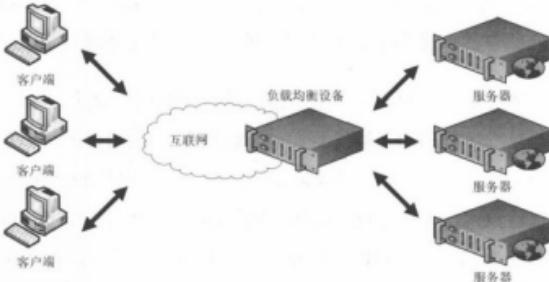


图 4-3 负载均衡实现示意图

从图 4-3 中可以看出，负载均衡设备能够在服务器之前截获到客户端发来的用户请求，然后按照预先配置的负载均衡策略将其分发到合适的后端服务器上。因此，客户端发出的业务请求报文首先都要发送到负载均衡设备的 IP 地址上，因为该地址并不负责处理实际的业务操作，因此通常将该地址称为虚拟 IP 地址，而将后端真正的业务服务器称为真实服务器。负载均衡设备通过虚拟 IP 地址和客户端通信，再将负载合理地分配给真实服务器。

在很长一段时间内，负载均衡主要是在 OSI 七层网络协议栈的第四层展开，而随着业务类型的日益丰富、用户精细化需求的不断提升和深度剖析技术的极

大完善，基于第七层的负载均衡技术已逐渐成为主流。

### **4.3.1 负载均衡关键技术**

负载均衡的目标是合理地将用户请求分发到合适的服务器上，从而达到系统处理的最优方案，其关键技术包括：负载均衡分发、会话持续性保证、服务器健康检测等。

#### **1. 负载均衡调度算法**

负载均衡调度算法的目标是将用户请求和相关流量高效、正确地分发到处于正常工作状态的服务器上，使得各台服务器尽可能地保持负载均衡。

目前商用系统中已经使用了非常丰富的负载均衡调度算法，总体而言调度算法可以分为静态算法和动态算法两大类。静态算法是指按照预先设定的策略进行分发，而不考虑当前服务器的实际负载情况，其实现比较简单快捷。典型的静态算法包括轮询、加权轮询、随机、加权随机、基于源 IP 的 Hash、基于源 IP 端口的 Hash、基于目的 IP 的 Hash、基于 UDP 报文净荷的 Hash 等；动态算法能够根据各服务器实际运行中的负载情况进行连接分发，具有更优的均衡效果，典型的动态算法包括基于最小连接、基于加权最小连接、最小响应时间等。

(1) 轮询 (Round Robin)：依次将请求分发到不同的服务器上，使得各台服务器平均分担用户的连接请求，该算法适用于集群中各服务器性能相当而无明显优劣差异的场景。

(2) 加权轮询 (Weighted Round Robin)：依次将请求分发到不同的服务器上，其中权值大的分配较多请求，权值小的分配较少请求，该算法利用权值标识服务器间的性能差异，适用于各服务器间性能不一的场景。

(3) 随机 (Random)：随机地将请求分发到不同的服务器上，从统计学角

度看，调度的结果为各台服务器平均分担用户的连接请求，该算法适用于集群中各服务器性能相当而无明显优劣差异的场景。

（4）加权随机（Weighted Random）：随机地将请求分发到不同的服务器上，从统计学角度看，调度的结果为各台服务器按照权值比重分担用户的连接请求，该算法适用于集群中各服务器性能存在差异的场景。

（5）基于源 IP 的 Hash（Source IP Hashing）：通过一个 Hash 函数将来自同一个源 IP 地址的请求映射到一台服务器上，该算法适用于需要保证来自同一用户的请求被分发到同一台服务器的场景。

（6）基于源 IP 端口的 Hash（Source IP and Source Port Hashing）：通过一个 Hash 函数将来自同一个源 IP 地址和源端口号的请求映射到一台服务器上，该算法适用于需要保证来自同一用户同一业务的请求被分发到同一台服务器的场景。

（7）基于目的 IP 的 Hash（Destination IP Hashing）：通过一个 Hash 函数将去往同一个目的 IP 地址的请求映射到一台服务器上，该算法适用于需要保证到达同一目的地的请求被分发到同一台服务器的场景。

（8）基于 UDP 报文净荷的 Hash（UDP Packet Load Hashing）：通过一个 Hash 函数将 UDP 报文载荷中特定字段的内容相同的请求映射到一台服务器上，该算法适用于需要保证 UDP 报文载荷中特定字段内容相同的请求被分发到同一台服务器的场景。

（9）最小连接（Least Connection）：根据当前各台服务器的连接数估算服务器的负载情况，把新的连接分配给连接数最小的服务器。该算法能够将连接保持时长差异较大的用户请求合理地分发到各台服务器上，适用于集群中各服务器性能相当、无明显优劣差异，而且不同用户发起的连接保持时长差异较大的场景。

(10) 加权最小连接 (Weighted Least Connection)：调度新连接时尽可能使得服务器上已经建立的活动连接数和服务器权值成一定比例，其中权值标识了服务器间的性能差异，该算法适用于集群中各服务器性能存在差异，而且不同用户发起的连接保持时长差异较大的场景。

(11) 最小响应时间 (Least Response Time)：调度新连接时尽可能选择对用户请求响应时间短的服务器，该算法适用于用户请求对服务器响应时间要求较高的场景。

不同的调度算法所实现的负载均衡效果不尽相同，可以根据实际应用场景的需求，选用不同的算法。例如在 Web 服务器集群可以采用轮询算法获得较高的访问响应，FTP 服务器集群可以采用最小连接算法减轻单台服务器的压力，报税等专用业务集群可以采用基于源 IP 端口的 Hash 算法确保用户访问的持续性。

除了上述传统的分发算法外，近年来针对内容分析的分发算法越来越受到负载均衡设备厂商的重视。举个例子来说，对于 Web 网站服务器集群而言，其使用的负载均衡设备不仅需要能够根据 IP 地址和默认 80 端口分发 HTTP 流量，还需要根据 HTTP 报文头或内容分配到指定服务器。这类设备可以对 HTTP 报文进行分析，获得 `Accept-Encoding`、`Accept-language`、`Host`、`Request-Method`、`URL-file`、`URL-function`、`User-agent` 等信息，并以此为依据向合适的服务器分发请求。

### 2. 会话持续性保证技术

会话持续性保证技术的目标是保证在一定时间段内某一个用户与系统的会话只交给同一台服务器处理，这一点在满足网银、网购等应用场景的需求时格外重要。

在实际的应用中，一次业务交互可能包含有多个 TCP 连接，不同业务的 TCP 连接有各自的特点。比如 FTP 业务的连接包含了一个控制通道和多个数据

通道，这些 TCP 连接之间存在显式的关联关系，即数据通道的 TCP 连接是通过控制通道协商得来的。因此，在负载均衡过程中只需要分析 FTP 业务的控制通道报文就可以获得各个数据通道的信息并进而将这些信息纳入到一个会话中，从而保证所有通道都访问同一台服务器。但是，对于 HTTP 业务而言，其各个 TCP 连接间就不存在这种显式的关联关系，在第 3 章中已经了解到，HTTP 是一个无状态协议。但是在某些场景中又要求一系列 HTTP 消息之间建立关联，比如 HTTP 网络购物，这只能依靠携带在数据报文中的相关信息表达连接之间的关联，例如源 IP 地址、Cookie 数据等。

这里讲的技术是负载均衡设备通过分析四层 TCP 数据包和七层 HTTP 消息中存在的隐式关联关系，确定处理连接请求的服务器，从而实现会话持续性保证。该领域的主要思路包括如下几点。

(1) 基于源 IP 地址的持续性保持：主要用于四层负载均衡，确保来自同一个源 IP 的业务能够分配到同一台服务器中。当负载均衡设备接收到某 IP 的首次请求时，建立持续性表项，记录下为该 IP 分配的服务器情况，在会话表项的生存周期内，后续具有相同源 IP 地址的业务报文都将被发往该服务器处理。

(2) 基于 Cookie 数据的持续性保持：主要用于七层负载均衡，用以确保同一会话的报文能够被分配到同一台服务器中。其中，根据服务器的应答报文中是否携带含有服务器信息的 Set-Cookie 字段，又可分为采用 Cookie 插入或者 Cookie 截取的方法。

- Cookie 插入保持：服务器的应答报文中不携带含有服务器信息的 Set-Cookie 字段，而由负载均衡设备在报文中添加相关信息。这样，客户端就会在请求报文中加入含有该服务器信息的 Cookie 字段，进而由负载均衡设备按照 Cookie 字段中的服务器信息将请求报文发给相应的服务器。
- Cookie 截取保持：服务器的应答报文中携带含有服务器信息的

**Set-Cookie** 字段，负载均衡设备根据用户配置的 **Cookie** 标识截取应答报文中的 **Cookie** 值。对于后续的客户端请求报文，如果能够匹配负载均衡设备中保存的 **Cookie** 值，则负载均衡设备直接将请求报文发给相应的服务器。

(3) 基于 SIP 报文 Call-ID 的持续性保持：主要用于七层负载均衡，用以确保 IP 会话中会话标识相同的 SIP 报文能够被分配到同一台服务器中。当负载均衡设备接收到某一个客户端的某一个业务的首次请求时，建立持续性表项，记录下为该客户端分配的服务器情况，在会话表项的生存周期内，后续具有相同会话标识的 SIP 业务报文都将被发往该服务器处理。

(4) 基于 HTTP 报文头的持续性保持：主要用于七层负载均衡，用以确保 HTTP 报文头中的某些关键信息相同的报文能够被分配到同一台服务器中。当负载均衡设备接收到某一个客户端的某一个业务的首次请求时，根据 HTTP 报文头关键字建立持续性表项，记录下为该客户端分配的服务器情况，在会话表项的生存周期内，后续具有相同 HTTP 报文头信息的报文都将被发往该服务器处理。至于什么信息被设置为关键字，是配置负载均衡策略的一个步骤。

### 3. 服务器健康检测技术

服务器健康检测技术的目标是及时发现和剔除工作状态不正常的服务器，保留“健康”的服务器池为用户提供服务。服务器健康检测的核心是定期对服务器的工作状态进行探测，收集相应信息，及时隔离工作状态异常的服务器。另外，除了标识服务器能否继续工作外，健康检测的结果还可以统计出服务器的响应时间，作为负载均衡设备选择服务器的依据。

当前，负载均衡领域已经使用了非常多的服务器健康检测技术，主要方法是通过发送不同类型的协议报文并通过检查能否接收到正确的应答来判定服务器的健康程度，主要包括以下几项。

#### (1) ICMP：向集群中的服务器发送 ICMP Echo 报文；若正确收到 ICMP

Reply，则证明服务器 ICMP 协议处理正常。

(2) TCP：向集群中的服务器的某个端口发起 TCP 连接建立请求，若成功三次握手建立 TCP 连接，则证明服务器 TCP 协议处理正常。

(3) HTTP：与集群中的服务器的 HTTP 端口（默认情况为 80 端口）建立 TCP 连接，然后发出 HTTP 请求，若收到的 HTTP 应答内容正确，则证明服务器 HTTP 协议处理正常。

(4) FTP：与集群中的服务器的 FTP 端口（默认情况为 21 端口）建立 TCP 连接，然后获取服务器上的文件，若收到的文件内容正确，则证明服务器 FTP 协议处理正常。

(5) DNS：向集群中的 DNS 服务器发出 DNS 请求，若收到正确的 DNS 解析应答，则证明服务器 DNS 协议处理正常。

(6) RADIUS：向集群中的服务器发送 RADIUS 认证请求，若收到认证成功的应答，则证明服务器 RADIUS 协议处理正常。

(7) SSL：向集群中的服务器发起 SSL 连接建立请求，若成功建立 SSL 连接，则证明服务器 SSL 协议处理正常。

针对不同的服务器功用，可以选择相应的健康检测方法，例如 ICMP 检测适用于服务器主机层面的健康检测，TCP 检测适用于业务层面的健康检测，HTTP 和 FTP 等方面的检测适用于应用层面的健康检测。一旦发现服务器处理出现异常，即可以将其上分担的负载通过负载均衡机制转移到合适的“健康”服务器上，保证整个集群系统的可用性。

### 4.3.2 负载均衡部署方式

负载均衡设备在具体实现中分直连和旁挂两种部署方式。此外，为了提高网络可用性，负载均衡设备的双机热备部署也是十分必要的。

### 1. 常用负载均衡部署方式

#### (1) 直连部署方式

直连部署方式比较简单，就是将负载均衡设备直接部署在报文必经之路上，作为服务器和客户端之间的路由设备，来往报文均直接由负载均衡设备进行路由，其部署示意如图 4-4 所示。

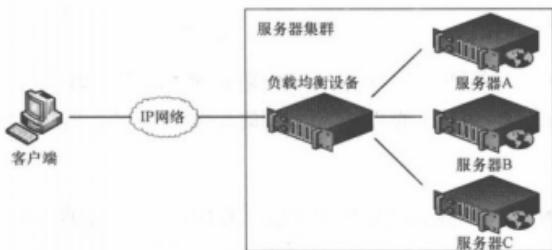


图 4-4 直连方式的负载均衡设备部署示意

#### (2) 旁挂部署方式

负载均衡设备的旁挂部署方式是指负载均衡设备并不作为服务器和客户端之间的路由设备，而只是旁挂在通用路由设备上，例如以板卡的形式插在路由器设备的扩展插槽中，其部署示意如图 4-5 所示。

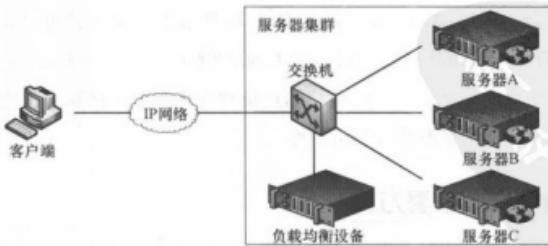


图 4-5 旁挂方式的负载均衡设备部署示意

在如图 4-5 所示的旁挂模式中，用于中转报文的路由交换设备的配置非常重要，因为客户端发送给服务器的请求流量如果要首先被负载均衡设备接收，就必须在路由交换设备上预先配置到负载均衡设备虚拟 IP 地址的路由信息。

一般情况下，旁挂部署方式中从服务器返回给客户端的流量可以不经过负载均衡设备，而是直接通过路由交换设备发给客户端。如果仍然希望流量先返回负载均衡设备，那么需要一些特殊的操作步骤，例如：令服务器和负载均衡设备处于同一个二层网络中，服务器将其网关设置为负载均衡设备；在路由交换设备上配置路由策略，将从服务器返回的流量定向到负载均衡设备上；负载均衡设备在转发客户端流量时进行源地址 NAT 转换。这些处理操作在负载均衡设备需要承担更多功用（例如 SSL 加速）的时候，是非常重要的。

## 2. 双机热备部署方式

无论是直连还是旁挂的部署方式，负载均衡设备始终处于网络的关键路径上，因此负载均衡设备的稳定性和安全性直接影响网络的可用性。为了避免出现负载均衡设备故障导致整个系统的单点故障，负载均衡设备必须支持双机热备，即在两台负载均衡设备之间通过备份链路进行对端设备上的业务备份，以保证两台设备的业务状态始终保持一致。当其中一台设备发生故障时，业务流量能够被及时切换到另一台设备上。因为另一台设备已经备份了故障设备上的全部业务信息，所以业务流量能够继续由该设备处理并分发，从而尽可能避免造成业务中断。

负载均衡设备的双机热备的实现主要有主备模式和负载分担模式两种类型。

(1) 主备模式：两台负载均衡设备中，一合作为主设备，另一合作为备份设备。主设备负责处理所有业务，并将产生的业务信息通过备份链路传送给备份设备。备份设备并不处理实际业务，而只用于备份。当主设备出现故障

时，备份设备及时接替主设备处理业务，在负责正常处理新发起的负载均衡业务的同时也要保证此前正在进行中的负载均衡业务不会中断。

(2) 负载分担模式：两台负载均衡设备均为主设备，都能够处理业务流量，同时又作为彼此的备份设备，负责备份对端设备产生的业务信息。当其中一台设备出现故障后，另一设备将负责处理全部业务，在负责正常处理新发起的负载均衡业务的同时，也要保证此前正在进行中的负载均衡业务不会中断。

### 4.3.3 服务器负载均衡

服务器负载均衡是将客户端请求在集群中的服务器上实现均衡分发的技术。按照位于七层网络协议栈的不同层的划分，服务器负载均衡可以分为四层（L4）负载均衡和七层（L7）负载均衡两种。其中，L4 负载均衡是基于流的服务器负载均衡，能够对报文进行逐流分发，即将同一条流的报文分发给同一台服务器；L7 负载均衡是基于内容的服务器负载均衡，能够对七层报文内容进行深度解析，并根据其中的关键字进行逐包转发，按照既定策略将连接导向指定的服务器。两者相比较，L4 负载均衡因无法对七层业务实现按内容分发，限制了它的适用范围，因此 L7 负载均衡受到了业界的极大重视并日渐成为服务器负载均衡的主流。

#### 1. 四层负载均衡

L4 负载均衡的实现主要有 NAT 方式和 DR 方式两种类型，它们适用于不同的应用场景。

##### (1) NAT 方式 L4 负载均衡

网络地址转换（NAT，Network Address Translation）属于广域网接入技术，它是一种将私有（保留）地址转化为合法 IP 地址的转换技术，被广泛应用于各种类型互联网接入方式和各种类型的网络中。

采用 NAT 方式实现的 L4 服务器负载均衡，后端服务器可以位于不同的物理位置和不同的局域网内。负载均衡设备在分发服务请求时，需要进行虚拟 IP 地址和目的 IP 地址转换，再通过路由将报文转发给具有相应目的地址的服务器。NAT 方式 L4 负载均衡的典型组网如图 4-6 所示。

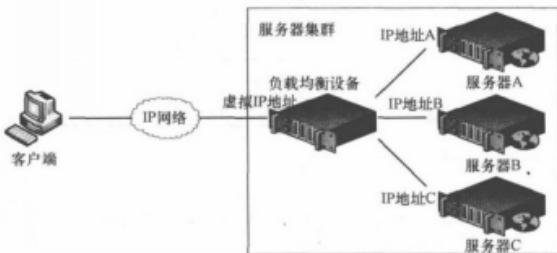


图 4-6 NAT 方式 L4 负载均衡的典型组网

图 4-6 中的主要组件包括如下几项。

- 负载均衡设备：负责将客户端服务请求分发到服务器集群中的具体服务器进行处理。
- 真实服务器：负责响应和处理各种客户端服务请求的服务器，多台服务器构成集群。
- 虚拟 IP 地址：集群对外提供的公网 IP 地址供客户端请求服务时使用。
- 服务器 IP 地址：服务器的 IP 地址供负载均衡设备分发服务请求时使用。这个 IP 地址可以是公网 IP，也可以是私网 IP。

NAT 方式 L4 服务器负载均衡的工作原理如图 4-7 所示。

工作流程是这样的：

- (1) 负载均衡设备负责接收客户端发送至虚拟 IP 地址的服务请求。

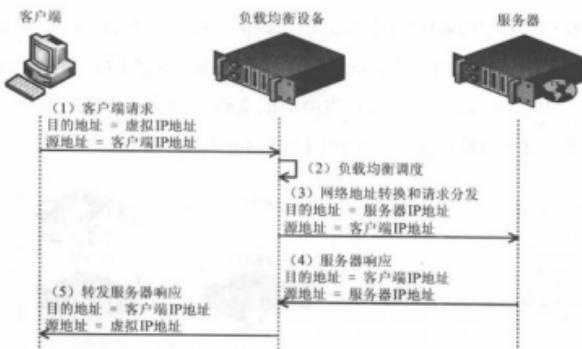


图 4-7 NAT 方式 L4 负载均衡的工作流程

(2) 负载均衡设备通过服务器可用性验证、会话持续性保证等对负载均衡算法进行调度，选择出负责响应和处理该请求的真实服务器。

(3) 负载均衡设备用真实服务器的 IP 地址改写请求报文的目标地址，再将请求发送给选定的真实服务器。

(4) 真实服务器的响应报文首先通过负载均衡设备。

(5) 负载均衡设备将响应报文的源地址修改为虚拟 IP 地址，再返回给客户端。

这种方法使用了网络地址转换（NAT）技术实现 L4 负载均衡，它具有组网灵活，适用于多种组网类型的特征。该方法对服务器没有额外的要求，不需要修改服务器的配置。

## (2) DR 方式 L4 负载均衡

DR (Direct Routing) 是另一种 L4 服务器负载均衡的实现方式。与 NAT 方式的 L4 负载均衡不同，DR 方式只有客户端发出的服务请求报文需要通过负载均衡设备，而服务器发出的响应报文则无须通过，这样能够有效减少负载均

衡设备的负载压力，避免负载均衡设备成为系统性能瓶颈。

DR 方式下，负载均衡设备在分发服务器请求时，不采用改变请求目的 IP 地址的方法，而是将报文的目的 MAC 地址替换为服务器的 MAC 地址，然后直接将报文转发给服务器。DR 方式 L4 负载均衡的典型组网如图 4-8 所示。

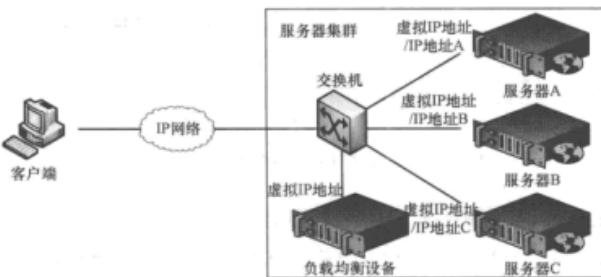


图 4-8 DR 方式 L4 负载均衡的典型组网

从图 4-8 中可以看出，DR 方式 L4 服务器负载均衡的实现方案与 NAT 方式类似，区别是负载均衡设备是以旁挂形式与交换机相连接的，而且集群中每台处理用户请求的服务器都拥有单独的虚拟 IP 地址。

DR 方式 L4 负载均衡的原理是为负载均衡设备和真实服务器同时配置虚拟 IP 地址，但是要求真实服务器的虚拟 IP 地址不能响应 ARP 请求，例如在环回接口上配置虚拟 IP 地址。用于响应服务请求的服务器上除了虚拟 IP 地址，还需要配置一个真实的 IP 地址用于和负载均衡设备通信。负载均衡设备要和真实服务器同处于一个二层网络内。因为服务器的虚拟 IP 地址不能响应 ARP 请求，所以从客户端发送给虚拟 IP 地址的报文将由负载均衡设备接收，负载均衡设备再将其分发给相应的真实服务器，而从真实服务器发送给客户端的响应报文则直接由交换机返回。相应的工作流程如图 4-9 所示。

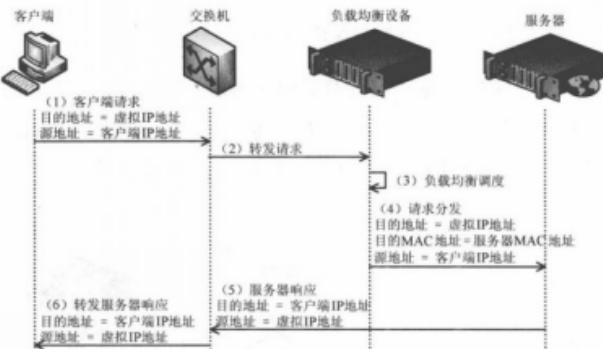


图 4-9 DR 方式 L4 负载均衡的工作流程

这种方法没有采用传统的通过查找路由表的转发方式来分发请求报文，而是通过修改目的 MAC 直接路由给服务器，DR 方式也就因此而得名。该方法只要求客户端到服务器的单方向请求报文经过负载均衡设备，负载均衡设备负担较小，不易成为瓶颈，具有更高的处理性能。

## 2. 七层负载均衡

L4 服务器负载均衡在截取数据流以后，对数据包的检查和分析仅局限于 IP 报文头部和 TCP/UDP 报文头部，而并不关心 TCP/UDP 数据包的有效载荷信息。而 L7 服务器负载均衡则要求负载均衡设备除了支持基于四层的负载均衡以外，还要解析数据包中四层以上的信息，即应用层的信息，例如解析 HTTP 内容，从而在数据包中提取出 HTTP URL 或者 Cookie 信息，用来作为负载均衡的依据。

L7 服务器负载均衡通过内容分析控制应用层服务分发，提供了一种高层的访问流量控制方式，与此前传统的 L4 负载均衡相比，L7 负载均衡具有如下优点。

- (1) 能够根据数据包内容（例如判断数据包是图像文件、压缩文件或者多

媒体文件等)把数据流量引向能够处理相应内容的服务器,提高系统可管理性和灵活性。

(2)能够根据连接请求的数据类型(例如根据URL判定用户发来的请求是和普通文本、图像等静态文档相关,还是和ASP、CGI等动态文档相关),将其引向相应的服务器处理,在提高系统性能的同时有助于改善安全性。

(3)能够根据应用层载荷保证会话持续性,相对于L4服务器负载均衡采用的基于地址的持续性保证方式更加精细。

L7负载均衡的典型组网如图4-10所示。

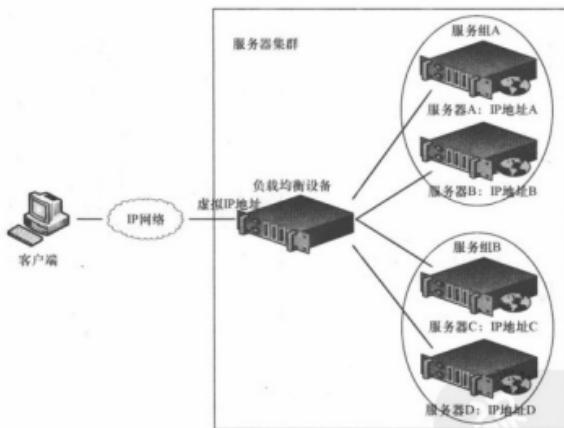


图4-10 L7负载均衡的典型组网

如图4-10所示,L7服务器负载均衡与NAT方式L4服务器负载均衡在实现上非常类似,其主要区别是增加了服务组的概念。服务组是一个逻辑概念,是指依据一些公共属性将多台服务器划分为不同的组。例如:可以将服务器划分为静态资料存储服务器组和动态交换服务器组,或者划分为音乐服务器组、视频服务器组和图片服务器组等,根据应用层属性划分的服务组内部各台服

务器更容易有相近的性能和特性。

我们结合图示 4-11 了解一下 L7 服务器负载均衡的工作原理。

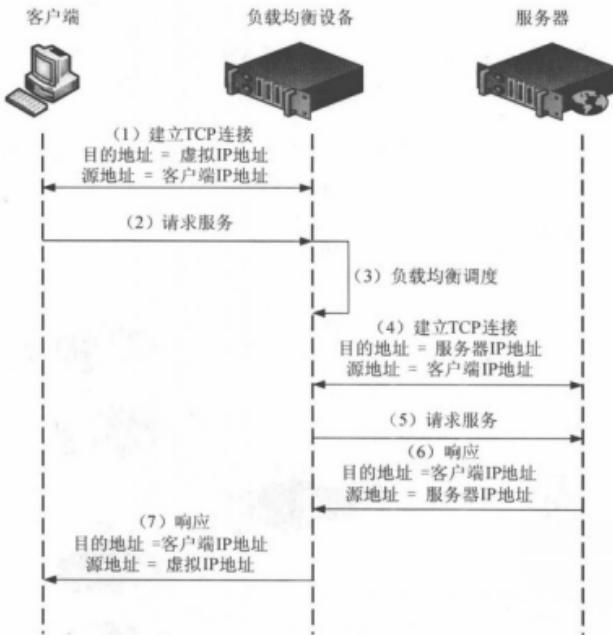


图 4-11 L7 负载均衡的工作流程

- (1) 客户端与位于服务器集群前端的负载均衡设备之间建立 TCP 连接。
- (2) 客户端将发送到虚拟 IP 地址的服务请求发送给负载均衡设备，负载均衡设备接收客户端请求。
- (3) 负载均衡设备通过服务器可用性验证、会话持续性保证、服务组匹配策略、负载均衡算法调度等步骤，选择出负责响应和处理该请求的真实服务器。

(4) 负载均衡设备利用客户端地址与真实服务器建立 TCP 连接。

(5) 负载均衡设备将客户端请求报文的目的地址重写为真实服务器的 IP 地址，并将该请求发送给相应的服务器。

(6) 真实服务器向负载均衡设备响应服务。

(7) 报文在通过负载均衡设备时，源地址被还原为虚拟 IP 地址，再返回给客户端。

L7 服务器负载均衡在负载均衡过程中能够对应用层协议进行深度识别，带来了很多更精细化均衡的可能，但它也对系统性能提出了非常高的要求，通常需要采用专用芯片以硬件电路的方式实现。同时，它需要针对每种应用层协议都配备相应的独立的识别机制，这极大地限制了 L7 服务器负载均衡的应用扩展性。由于 HTTP 协议应用广泛且协议相对简单，所以当前 L7 负载均衡技术对 HTTP 请求进行负载均衡的商用能力最强。

还要说明的是，在实际应用中，L4 负载均衡和 L7 负载均衡往往是搭配使用的，当然最好是在同一台负载均衡设备上兼具四层和七层功能。负载均衡设备首先从报文中提取 IP 地址和端口号，进行四层负载均衡，如果发现确有必要进行进一步的基于报文内容的转发，再实施七层负载均衡操作。

#### 4.3.4 链路负载均衡

前面我们了解了服务器负载均衡，下面还要介绍一种负载均衡，叫做链路负载均衡。这是指通过动态算法在多条网络链路中进行负载均衡。

为了规避运营商网络出口故障导致的网络可用性风险，解决网络带宽不足导致的网络访问失效等问题，企业用户通常会同时租用两家或者更多家运营商网络。在这种情况下，问题就是如何合理地运用这多个运营商的网络出口。传统的基于策略的路由可以在一定程度上解决问题，但是配置不方便，也难以动

态适应网络架构变化。最主要的是，这种方法无法根据带宽的实际使用情况动态调整报文分发，造成空闲链路吞吐能力的浪费。因此，链路负载均衡被提出来，希望能够解决传统路由技术存在的这一系列问题。

链路负载均衡根据业务流量的方向可以分为 Outbound 链路负载均衡和 Inbound 链路负载均衡两种情况。其中，Outbound 链路负载均衡主要解决的是企业内部业务系统访问外部互联网服务时如何在多条不同的链路中动态分配和负载均衡的问题；Inbound 链路负载均衡主要解决的是位于互联网外部的用户如何在访问企业内部网站和业务系统时动态地在多条链路上平衡分配，并在一条链路中断时能够智能地自动切换到另一条可用链路。

链路负载均衡与服务器负载均衡之间的主要区别是承担负载的对象不同，单就负载均衡设备而言，其关键技术、部署方式等内容都具有类似之处。其中，在负载均衡调度算法方面，链路负载均衡有一些特有算法，例如就近链路选择算法、基于链路带宽的调度算法等。特别是就近链路选择算法，能够在链路负载均衡过程中，实时探测链路的状态，并根据探测结果选择最优链路，保证流量通过最优链路转发。就近链路选择算法对链路状态的探测可以通过链路健康性检测实现，例如通过发送 DNS 报文和 ICMP 报文、建立 TCP 半开连接等方法在验证链路可达性的同时获得相关的就近链路计算参数。在实现中，就近链路计算参数主要包括：链路物理带宽、链路成本（例如链路月租）、链路延迟时间、路由跳数等。就近链路选择算法也可以根据这些参数进行加权计算，然后根据计算结果判断链路的优劣。

### 1. Outbound 链路负载均衡

当内网和外网之间存在多条链路时，Outbound 链路负载均衡可以实现在多条链路上分担内网用户访问外网服务器流量的功能，其典型组网如图 4-12 所示。

在图 4-12 中，主要元素包括如下几项。

（1）负载均衡设备：负责将内网到外网的流量在多条物理链路上分发。

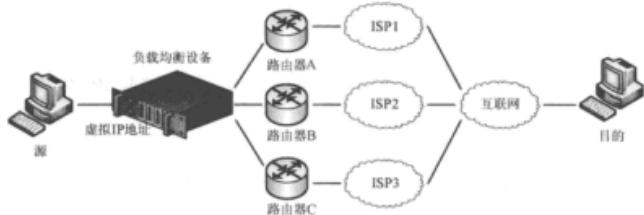


图 4-12 Outbound 链路负载均衡的典型组网

- (2) 物理链路：网络服务器提供商（ISP）提供的实际通信链路。
- (3) 虚拟 IP 地址：负载均衡设备对外提供的虚拟 IP 地址，用做用户发送报文的目的地址。

**Outbound** 链路负载均衡的原理是将负载均衡设备的虚拟 IP 地址作为内网用户发送报文的目的地址。用户在将访问虚拟 IP 地址的报文发送给负载均衡设备后，负载均衡设备通过链路选择算法选择出最佳的物理链路上，并将内网访问外网的业务流量分发到该链路上。相应的工作流程如图 4-13 所示。

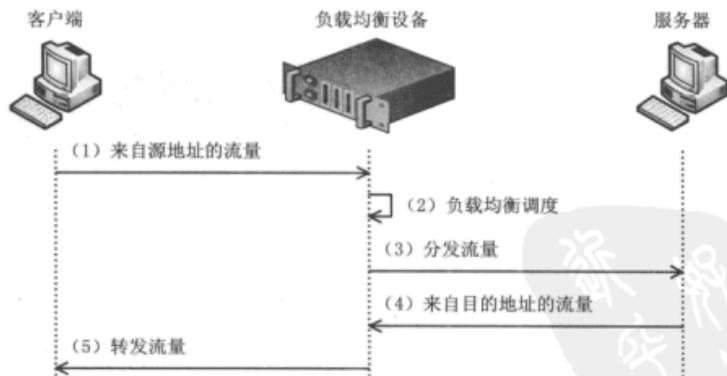


图 4-13 Outbound 链路负载均衡的工作流程

Outbound 链路负载均衡的技术特点包括如下几项。

- (1) 通过结合 NAT 技术进行组网，不同链路使用不同源 IP 地址，从而能够保证往返报文均经由同一条链路。
- (2) 通过健康性检测，可以检测链路上任意节点的连通性，从而有效地保证整条路径的可达性。
- (3) 通过负载均衡调度算法的调度，可以在多条链路间均衡流量，例如利用基于带宽的算法优化带宽利用、利用就近链路选择算法选择最优链路。

## 2. Inbound 链路负载均衡

前面提到，Inbound 链路负载均衡的主要作用是在多条链路上均衡调度外网用户访问企业内部网站和业务系统时的流量，并在一条链路中断时能够智能地自动切换到另一条可用链路。其典型组网如图 4-14 所示。

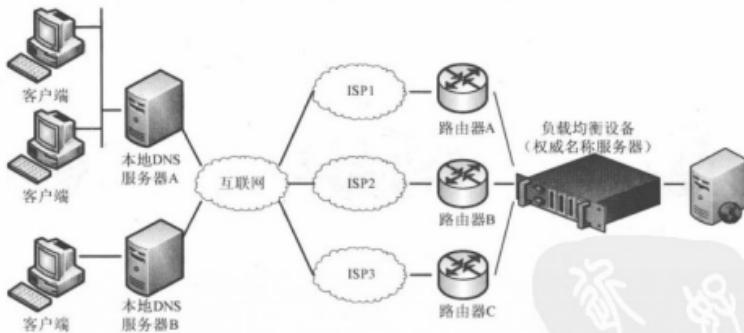


图 4-14 Inbound 链路负载均衡的典型组网

如图 4-14 所示，Inbound 链路负载均衡的实现方案虽然与 Outbound 链路负载均衡在很多方面具有相似性，但是因为针对的是不同传输方向上的业务流量，因此部分功能有如下几点区别。

(1) 负载均衡设备：负责引导外网流量通过不同的物理链路转发到内网，实现流量在多条物理链路上的分发。同时，负载均衡设备还需要作为待解析域名的权威名称服务器。

(2) 物理链路：网络服务器提供商（ISP）提供的实际通信链路。

(3) 本地 DNS 服务器：负责处理本地用户发送的 DNS 解析请求，将该请求转发给作为权威名称服务器的负载均衡设备。

Inbound 链路负载均衡的原理是将负载均衡设备作为权威名称服务器，用于记录域名与内网服务器 IP 地址之间的映射关系（即域名的 A 记录）。在本书第 5 章我们会了解到，一个域名会被映射为多个 IP 地址，每个 IP 地址则对应一条物理链路。当外网用户通过域名方式访问内网服务器时，本地 DNS 服务器依照递归原则会向负载均衡设备请求域名解析，负载均衡设备通过就近链路选择算法筛选、ISP 选择等步骤，选择出最佳的物理链路，并将通过该链路与外网连接的接口 IP 地址作为 DNS 域名解析结果反馈给外网用户。相应的工作流程如图 4-15 所示。

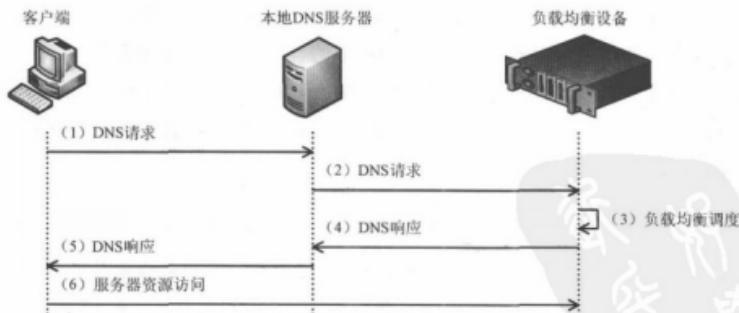


图 4-15 Inbound 链路负载均衡的工作流程

Inbound 链路负载均衡通过和服务器负载均衡相配合，可以同时实现多条链路间的均衡和多台服务器间的均衡。另一种应用方式是通过动态的链路选择

算法，最终实现以链路最优为基础的服务器负载均衡。

## 4.4 开源负载均衡软件

当前，业界已经有很多成熟的系统和软件用于实现集群的负载均衡，其中使用最为广泛的包括主要用于实现四层负载均衡的 LVS 和主要用于实现七层负载均衡的 Nginx，它们都是在开源社区的支持下研发的负载均衡软件。

### 4.4.1 LVS

LVS (Linux Virtual Server) 是由国防科学技术大学章文嵩在 1998 年 5 月发起的遵循 GPL 许可证的开源软件项目，其目标是为 Linux 集群系统研发高性能、高可用的负载均衡解决方案。当前，LVS 已经成为 Linux 标准内核的一部分，2.4 版本以后的 Linux 内核中已经完全内置了 LVS 的各个模块，用户可以直接使用 LVS 提供的各项功能。从项目创立至今，LVS 已经发展成为一个非常成熟的技术项目，可以被用于提供高可扩展的、高可靠的、高可服务的 Linux 集群系统服务，很多著名的网站和组织都使用了 LVS 技术，例如：Linux 的门户网站、向 RealPlayer 提供音频视频服务的 Real 公司、全球最大的开源网站 SourceForge 等。

负载均衡系统通常位于整个集群的最前端，由一台或者多台负载调度器组成，LVS 就安装和部署在负载均衡调度器上。安装了 LVS 负载调度器的主要功用类似于路由器，它含有完成 LVS 负载均衡功能所需的路由表，通过其中的路由信息将用户请求分发给集群中的应用服务器。同时 LVS 负载调度器还需要监测各台应用服务器的工作情况，并根据服务器的状态是否正常可用随时更新 LVS 路由表。

LVS 的核心是 IP 负载均衡，实现该功能的是 IPVS 模块，它被安装在负载调度器上并虚拟出一个 IP 地址提供给用户，用户必须通过该虚拟 IP 地址才能够访问到集群提供的应用服务，即所有的用户请求都首先经由虚拟 IP 地址达到负载调度器，然后负载调度器将从应用服务器列表中选择一个合适的节点响应用户请求。根据调度器将请求发送到应用服务器节点以及应用服务器返回数据给用户的方式上存在的差异，IPVS 具有 NAT（Network Address Translation）、TUN（IP Tunneling）、DR（Direct Routing）等不同的实现方式。其中，除了本书介绍过的 NAT、DR 方式外，TUN 是指将一个 IP 报文封装在另一个 IP 报文中的技术，IPVS 利用该技术先将用户请求进行封装再转发给应用服务器，响应内容则直接从应用服务器返回给用户。采用 TUN 方式，LVS 负载调度器只负责用户请求的调度而不处理响应数据，仅需耗费用于建立 IP 隧道的开销从而具有较高的性能，但是 TUN 技术的实现对服务器要求较高，即所有的服务器必须支持“IP Tunneling”或“IP Encapsulation”协议。另外，LVS 能够提供的负载均衡调度算法也比较丰富，例如轮询（Round Robin）、加权轮询（Weighted Round Robin）、最小连接（Least Connection）、加权最小连接（Weighted Least Connection）等。

LVS 的主要特点包括如下几点。

（1）性能好。LVS 集成在操作系统内核之中，在网络协议栈第四层对用户请求进行分发，自身没有流量的产生，因此具有极高的性能。

（2）配置简单。LVS 已经被广大的 Linux、FreeBSD 操作系统所支持，配置文件比较简单，操作便捷，降低了配置出错的几率。

（3）工作稳定。LVS 在实际应用中已被广泛证明具有极强的可靠性，同时它具有完整的双机热备方案，如 LVS+Keepalived 和 LVS+Heartbeat 等。

当前，除了 IP 负载均衡外，LVS 还实现了 KTCPVS（Kernel TCP Virtual Server）用于在网络协议栈的第七层进行负载均衡操作，以及其他一些扩展功

能模块。LVS 官方网站的网址是 <http://www.linuxvirtualserver.org>，网站上提供了最新版本的 LVS 的下载链接和相关的软件手册。

#### 4.4.2 Nginx

Nginx（读作 engine-x）是一款高性能的 HTTP 和反向代理服务器，同时它也是一个 IMAP/POP3/SIMTP 代理服务器。Nginx 于 2002 年由俄罗斯人 Igor Sysoev 开发，它遵循的是类 BSD 许可证（2-clause BSD-like license），其第一个公开版本 0.1.0 发布于 2004 年 10 月 4 日。2011 年 11 月 15 日，Nginx 的 1.0.10 稳定版正式发布。Nginx 具有非常优越的性能，特别是极高的并发处理能力，因此获得了众多互联网网站的青睐，根据互联网监测公司 Netcraft 的统计报告，截至 2011 年 11 月，全球最忙碌的 100 万个 Web 站点中就有 8.01% 的份额采用了 Nginx 软件，包括 WordPress、Hulu、Github、Ohloh、SourceForge、WhitePages、TorrentReactor 等。

Nginx 实现了完善的 HTTP 服务器功能，在实际的网站部署中，它经常被用于处理网络协议栈七层的负载均衡问题。Nginx 负载均衡的核心是 Upstream 模块，当 Nginx 接收到用户发来的 HTTP 请求后，它将创建一个到后端应用服务器的 Upstream 请求，待应用服务器发送回响应数据后，Nginx 再将后端 Upstream 中的数据转发给用户。Upstream 支持的负载调度算法主要包括基于源 IP 的 Hash、基于目的 URL 的 Hash、基于响应时间的公平调度等。

Nginx 的主要特点包括如下几点。

- (1) 调度灵活。Nginx 工作在网络协议栈的第七层，能够对 HTTP 应用请求进行解析和分流，支持比较复杂的正则规则，具有更优化的负载调度效果。
- (2) 网络依赖性低。Nginx 对网络的依赖程度非常低，理论上讲，只要能够 Ping 通就可以实施负载均衡，而且可以有效区分内网和外网流量。

(3) 支持服务器检测。Nginx 能够根据应用服务器处理页面返回的状态码、超时信息等检测服务器是否出现故障，并及时将返回错误的请求重新提交到其他节点上。

相对于 LVS，Nginx 主要用于网络七层的调度，在灵活性和有效性方面更具优势，同时它对服务器健康状态的检测也避免了用户访问过程中的连接断线。但是，网络七层信息处理的复杂度也使得 Nginx 在负载能力和稳定性方面与 LVS 相比有较大的差距。另外，它目前只支持 HTTP 应用和 EMAIL 应用，在应用场景上不如 LVS 丰富，而且也不具备现成的双机热备方案。总体而言，实际场景中可以考虑 LVS 和 Nginx 的结合使用，其中将 LVS 部署在前端用于处理四层的负载均衡，当需要更细节的负载调度时再启用 Nginx 以优化调度效果。

Nginx 官方网站的网址是 <http://nginx.org/>，网站上提供了最新版本的 Nginx 的下载链接和相关的软件手册。

# □ 第 5 章 全局负载均衡工作原理及实现

---

- 5.1 全局负载均衡在 CDN 系统中的作用
- 5.2 基于 DNS 解析的 GSLB 实现机制
- 5.3 基于 DNS 的 GSLB 应用部署方法
- 5.4 基于应用层协议重定向的 GSLB
- 5.5 基于 IP 路由的 GSLB
- 5.6 小结



## 5.1 全局负载均衡在 CDN 系统中的作用

上一章介绍的集群和本地负载均衡，实现了服务器群的高可用性。但如果服务器集群所在的数据中心断电呢？如果由于底层 IP 网络故障造成数据链路中断呢？又如果洪水或者地震造成数据中心瘫痪怎么办呢？在一个数据中心内，无论采用怎样的技术保障系统高可用性，总还是有一些不可抗力因素会导致整个集群无法工作。所以通常会把服务器分散部署，放在多个数据中心里。另外，CDN 系统总是希望使用离用户最近的设备为用户提供服务，这样就需要在全网不同位置部署多个节点。CDN 的全局负载均衡系统（GSLB）就是负责解决多个 CDN 节点之间相互协同问题，实现整个系统的大规模服务能力和高可用性。

全局负载均衡（GSLB）这一层次的负载均衡主要是在多个节点之间进行均衡，其结果可能直接终结负载均衡过程，也可能将用户访问交付下一层次的（区域或者本地）负载均衡系统进行处理。经过多年的发展，已有多种调度机制应用于 CDN 的全局负载均衡系统，其中最通用的是基于 DNS 解析方式的 GSLB，我们将在 5.2 节和 5.3 节分别从工作原理和实施应用方面做重点介绍，另外会对基于 HTTP 重定向、IP 路由等其他方法做简要的介绍。

GSLB 在全网进行负载均衡的目标，是保持各节点、各设备负载保持在一个有利于提供优质服务的水平（对于 CDN 服务商来说，可能还有从成本、收益角度考虑的负载分配，人为地造成负载不均衡）。为了实现这个目的，GSLB 会依靠一些灵活的均衡策略来进行负载判断，而这些策略的实现有赖于 GSLB 对全局信息的收集和分析能力。在区域或本地负载均衡系统中，每个服务节点都只掌握本节点内服务设备的信息，而在 GSLB 系统中，需要掌握所有节点中设备的信息。也就是说，所有节点设备的信息情况在 GSLB 产生了一次汇总，也可以通过 GSLB 了解其他节点设备的情况。GSLB 需要掌握的信息精细度，

取决于 GSLB 要执行的调度精度。关于这些策略和信息，在本章 5.2 节、5.3 节分别有详细讲解。

GSLB 的功能可以嵌入到负载均衡产品中，也可以打包成一个独立的产品。像 Foundry、Nortel、Cisco、Radware 等厂商都是把 GSLB 集成到它们的负载均衡产品中。也有些厂商把 GSLB 的功能从服务器负载均衡中分离出来做成独立的产品，例如 F5 Networks。

## 5.2 基于 DNS 解析的 GSLB 实现机制

基于 DNS 解析的 GSLB 系统，利用了 DNS 系统固有的域名解析、就近性判断、轮询算法等，可以很大程度借助独立于 CDN 系统之外的公共 DNS 系统来完成负载均衡，降低了对 CDN 系统本身的压力，所以是目前 CDN 服务商采用比较多的 GSLB 方案，也是技术比较成熟和稳定的解决方案。这种调度机制的基础就是 DNS 系统，因此下面我们先从基本的 DNS 工作原理说起。

### 5.2.1 DNS 的产生背景

Internet 将数量众多的主机连接在一起，要让这些主机进行通信，需要有一套名字标识体系，让主机之间能够彼此找到对方。实际上，在 Internet 上有多种方式进行主机标识，既可以使用主机名标识一台主机，也可以使用 IP 地址标识。主机在互联网上的位置主要靠 IP 地址进行标识，每个 IP 地址都由 4 个字节组成，有着严格的层次结构，以便于路由器进行识别和处理。但这种纯数字的标识方式不便于人的记忆，因此提出了主机名的标识方式，如 www.CDNbook.com 的形式，这种名字便于记忆，容易被人们所接受，在具体应用中多采用主机名方式访问主机。机器喜欢 IP 地址，人喜欢主机名，为了兼

顾人和机器的不同偏好，需要一个翻译机制实现两者之间的转换，这就是主机名与 IP 地址的映射关系。

在 Internet 的前身 ARPAnet 的年代，整个网络中只有几百台主机，所有主机信息以及主机名与地址的映射记录都存放在一个名为 HOSTS.TXT 的文件中。HOSTS.TXT 从一台名为 SRI-NIC 的主机上分发到整个网络，这台主机由斯坦福研究院的网络信息中心负责维护。ARPAnet 的管理员们通过电子邮件的方式将变更信息通知 NIC，同时定期 FTP 到 SRI-NIC 上，获得最新的 HOSTS.TXT 文件。

在 20 世纪 70 年代末，整个 ARPAnet 只有 200 台主机，但是到了 80 年代，ARPAnet 上主机数量飞速增长，到 80 年代末期主机总数增加到 10 万台，这种人工更新的方式显然不再适用了。频繁更新 HOSTS.TXT 造成了网络流量和处理器负载增加，并且名字冲突和数据一致性维护越来越困难。为了解决这一问题，网络管理者们希望找到一种合理的方法来解决主机信息的维护问题，既能分散管理，又能及时进行全网更新和同步，并且采用层次化的名字空间，保证主机名的全网唯一性。1984 年，南加州大学信息科学所的 Paul Mockapetris 发布了描述 DNS 的 RFC 882 和 RFC 883，就是最初的 DNS 规范，DNS 系统诞生了。

### 5.2.2 DNS 基本工作原理

#### 1. DNS 系统的组成和工作流程

DNS 实际上是一个应用层协议，但它通常被其他的应用层协议所使用，用于将用户提供的主机名解析为 IP 地址。例如：当需要访问 [CDNbook.com](http://www.CDNbook.com) 这个站点时，会在浏览器地址栏里输入 [HTTP://www.CDNbook.com](http://www.CDNbook.com) 这样一个 URL。实际上我们想要浏览的网页内容都存放在互联网中的某台服务器上，而浏览器的任务就是找到我们想要访问的这台服务器的 IP 地址，然后向它请求内容。从这一刻起，DNS 就开始工作了。其工作流程如图 5-1 所示。

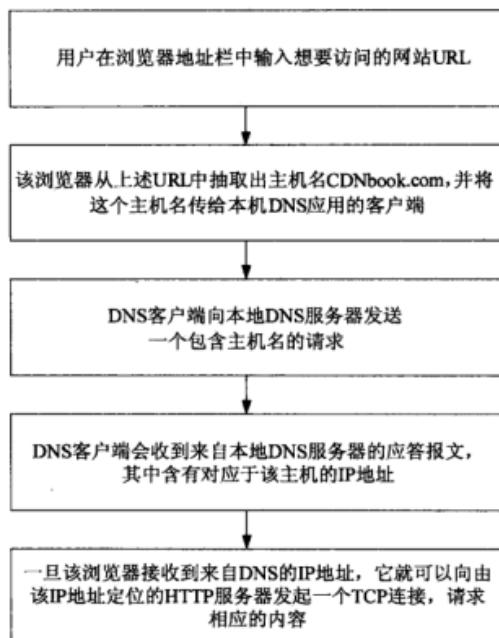


图 5-1 DNS 基本工作流程

从图 5-1 所示的流程可以看出，DNS 地址解析服务是在 HTTP 连接建立之前的一个过程。从用户主机上调用应用程序的角度看，DNS 是一个提供简单、直接的转换服务的黑盒子，但实际上 DNS 服务系统的运转相当复杂，由分布于全球的大量 DNS 服务器以及相关应用层协议共同组成（DNS 的详细标准请参考 RFC 1034 和 RFC 1035），下面我们就对 DNS 系统的工作过程进行介绍。

首先，来看看域名结构。整个互联网中的域名空间结构就像一棵倒置的树，如图 5-2 所示。这棵域名树的根称为 **根 DNS 服务器**，在 Internet 上总共有 13 个根 DNS 服务器（每个服务器其实都是一个集群组成的），大部分位于北美；其往下第一级节点称为 **顶级域 (Top Level Domain, TLD)**，目前共有 7 个

这样的顶级域名。

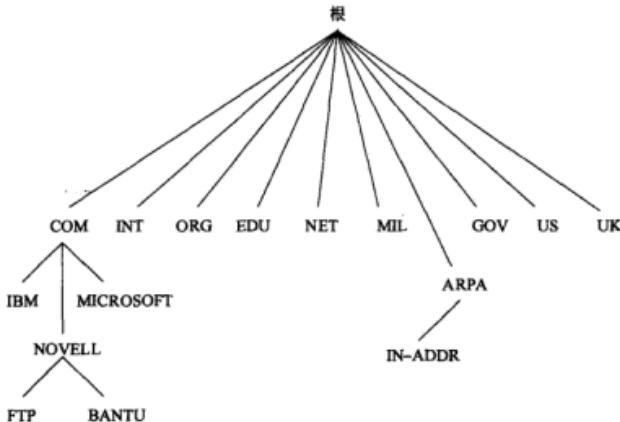


图 5-2 域名树形结构图

1. com: 商业组织
2. edu: 教育机构
3. gov: 政府部门
4. mil: 军事部门
5. net: 网络基础设施
6. org: 非营利性组织
7. int: 国际组织

每个顶级域由对应的顶级域（TLD）服务器负责管理，除了以上 7 个顶级域名，还有各个国家的顶级域名（如 cn、uk、ca 和 fr 等）也在这一级别进行管理。

每个顶级域再向下展开分支，每个分支域都是一个子域。比如 CDNbook.com 是域 com 的子域。CDNbook.com 也可再包含子域，比如 a. CDNbook.com、b. CDNbook.com。一个域就是域名空间中的一棵子树。域的名字也就是这棵子树的顶端节点的域名。例如在图 5-3 中，CDN.com 域的顶端节点就是 CDN.com。

在每个域中，会有一台或多台服务器用来保存这个域名空间的所有信息，并且响应关于该域名空间的所有请求。这种服务器就叫做这个域的**权威域名服务器**（也常称为**授权域名服务器**），它拥有这个域所有的域名信息。每个域都可以分为多个子域，而每个权威域名服务器可以给一个或多个区域进行解析。但即使各个区域被授权给同一个权威域名服务器，它们之间仍然是彼此独立的。例如，CDNbook.com 被划分为三个子域：a. CDNbook.com、b. CDNbook.com 和 c. CDNbook.com。每个子域都可以自行维护自己的权威域名服务器。一个域可以有多台权威域名服务器，但是只有一台是主域名服务器，这台主域名服务器负责向其他辅域名服务器分发每个域名空间的更新信息。

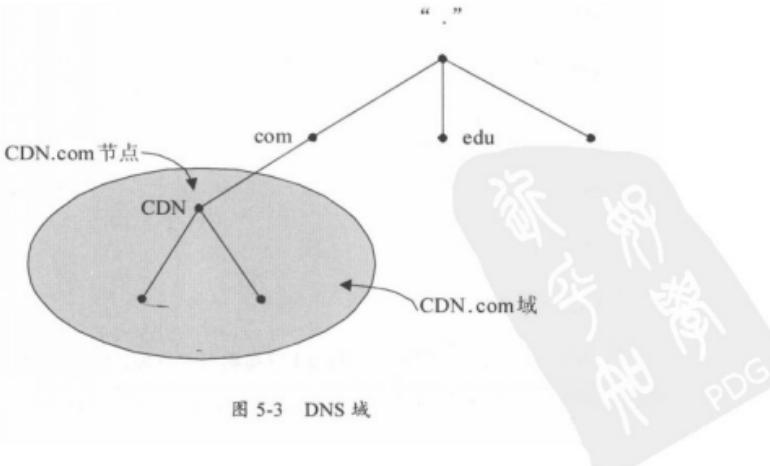


图 5-3 DNS 域

当一个子域被授权出去，它原本所属的域就不再包含它的数据，而只留下一些指针，这些指针指向相应子域的授权域名服务器。如果有一个询问该子域的信息的请求，所返回的应该是该子域权威服务器列表。

现在让我们总结一下：根 DNS 服务器、顶级域名（TLD）服务器和权威 DNS 服务器共同组成了 DNS 服务器层，共同维护分布式、层次化的 DNS 数据库。DNS 系统采用树形设计的一个主要目的就是为了分散管理。而这种分散管理是通过“授权”来实现的。对域进行授权，就是域管理组织把子域授权给其他组织进行管理，由子域管理者来维护子域中的数据，可以自由改动数据，包括对子域再进行划分，再授权。

在 DNS 系统中还有一类非常重要的域名服务器，叫做本地 DNS 服务器（LDNS, Local DNS Server），是用户所在局域网或 ISP 网络中的域名服务器。当客户端在浏览器里请求 URL: <HTTP://www.CDNbook.com> 时，浏览器会首先向本地 DNS 服务器请求将 <HTTP://www.CDNbook.com> 解析成 IP 地址，本地 DNS 服务器再向整个 DNS 系统查询，直到找到解析结果。那么客户端是如何知道本地 DNS 服务器在哪里呢？本地 DNS 服务器地址是客户端网络配置的一部分，或者通过 DHCP 方式分配给客户端。

本地 DNS 服务器得到浏览器的域名解析请求后，会采用递归查询方式（recursive query）或者迭代查询方式（iterative query）向 DNS 系统中的其他远程域名服务器提出查询请求。递归方式指每次查询请求都由本地 DNS 服务器发起，收到答复后再向下一个远程 DNS 服务器提出请求，直到获得结果。迭代查询指本地 DNS 服务器只将自己知道的最合适的结果返回给查询者，帮助它把查询过程继续下去，而它本身不再做其他任何查询。在实际应用中，递归方式是比较常见的。图 5-4 描述的就是一个典型的 DNS 递归查询过程。

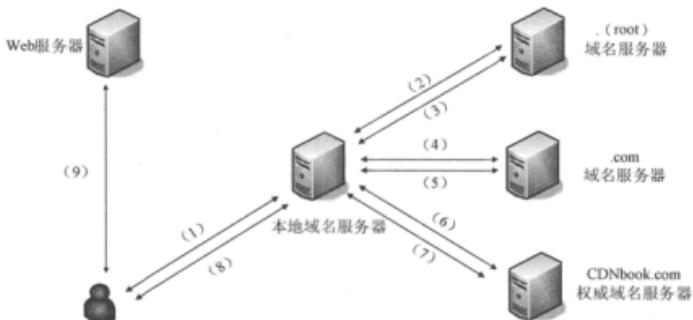


图 5-4 DNS 域名解析过程

本地 DNS 服务器首先会去根 DNS 服务器请求解析（此时条件是本地 DNS 服务器并没有关于这个域名的缓存。我们将在后面讲解 DNS 缓存），根 DNS 服务器会返回.com 域的域名服务器列表。本地 DNS 服务器在.com 域名服务器列表中选择一个 IP 地址，向这个 IP 地址的 DNS 服务器发送域名解析请求,.com 域的域名服务器将 CDNbook.com 域的权威域名服务器 IP 地址返回给本地 DNS 服务器。本地 DNS 服务器向 CDNbook.com 域的权威域名服务器发送解析请求，最后 CDNbook.com 域的权威域名服务器将站点 [HTTP://www.CDNbook.com](http://www.CDNbook.com) 的 Web 服务器的 IP 地址返回给本地 DNS 服务器。

前面我们提到，DNS 给使用它的互联网应用带来了额外的时延，有时时延还比较大，为了解决这一问题，需要引入“缓存”机制。缓存是指 DNS 查询结果在主机中的缓存，这是 DNS 系统中一个非常有特色且有用的功能。有了缓存，就不需要在每次查询的时候都经过复杂的递归过程。在区内主机对某个域名发起第一次查询请求时，负责处理递归查询的 DNS 服务器要发送好几次查询才能找到结果，不过在这过程中它也得到了许多信息，比如各区权威 DNS 服务器和它们的地址、域名解析最终结果。它会把这些信息保存起来，当其他主机向它发起查询请求时，它就直接向主机返回缓存中能够找到的结果。当然，DNS 服务器不能把数据永远放在缓存中，管理员会为这些数据设置一个生存期（time to

live)，这就是 TTL。超过 TTL 时间的数据会被清除，重新从权威 DNS 服务器上获取新的数据。

除了 DNS 服务器能缓存 DNS 响应信息之外，客户端浏览器也可以缓存 DNS 响应信息。比如 Windows 操作系统下，IE 会缓存 DNS 服务器返回的地址信息，当用户请求页面域名解析结果在 IE 自身的 DNS 缓存中能够查到时，就不会向 DNS 服务器发起请求了，这样可以加快浏览网页的速度。当消息记录时间超过 IE 浏览器设置的 DNS 缓存时间时，会重新向 DNS 服务器发起域名解析请求，用新的解析结果更新缓存。缓存生存期也是可以设定的，微软的技术支持网站提供了通过修改注册表来改变 IE 浏览器中默认 DNS 缓存时间的方法。

## 2. DNS 记录类型及报文格式

域名服务器是根据资源记录来对 DNS 请求进行应答的。在 DNS 系统中，最常见的资源记录是 Internet 类记录，资源记录是一个包含了下列字段的 4 元组：Name、Value、Type、TTL。其中，TTL 是该记录的生存时间，它决定了资源记录应当从缓存中删除的时间；Name 和 Value 的值取决于 Type，即记录类型。Internet 类资源记录主要分为以下几种类型：

### (1) A 记录，Address

A 记录用于描述域名到 IP 地址的映射关系。对同一个域名，可以有多条 A 记录。也就是说，一次 DNS 查找可以返回多个地址。

### (2) NS 记录，Name Server

NS 记录是域名服务器记录，用于指定该域名由哪个 DNS 服务器来进行解析。每个区域可以有多个域名服务器，因此可以有多条 NS 记录。

### (3) SOA 记录，Start Of Authority

SOA 记录用于指定该区域的权威域名服务器。每个区域允许且只允许有一

个 SOA 记录，它是资源记录的第一个条目。

#### ( 4 ) CNAME 记录

CNAME 记录用于描述别名与域名的对应关系，这种记录允许您将多个名字映射到同一台计算机。例如，有一台计算机名为 `host.mydomain.com` ( A 记录 )。它同时提供 WWW 和 MAIL 服务，为了便于用户访问服务，可以为该计算机设置两个别名 ( CNAME )：WWW 和 MAIL。这两个别名的全称就是 `www.mydomain.com` 和 `mail.mydomain.com`。实际上它们都指向 `host.mydomain.com`。当域名服务器查找一个域名时，找到一条 CNAME 记录，它会用记录中的规范名来替换这个域名别名，然后再查这个规范名的 A 记录，从而找到与规范名对应的 IP 地址，这样，就实现了对请求查找域名的 IP 地址响应。

#### ( 5 ) PTR 记录，Pointor Record

PTR 记录用于描述 IP 地址到域名的映射关系。这种描述方式只存在于 `in-addr.arpa` 这个特殊的域中。`in-addr.arpa` 域实际上就是一个以地址为标号的域名空间；如图 5-5 所示。由于在域名系统中，一个 IP 地址可以对应多个域名，因此从 IP 出发去找域名，理论上应该遍历整个域名树，但这在 Internet 上是不现实的。为了完成逆向域名解析，系统提供一个特别域，该特别域称为逆向解析域 `in-addr.arpa`。域中的节点都是用数字来做标号的，这些数字就是以点分字节来表示的 IP 地址（用点分隔 4 个 0 到 255 之间的数字称为“点分字节”表示法，是最常用的 IP 地址表示方法）。`in-addr.arpa` 域的最下面一层的最后一个字节相连的资源记录给出了这个 IP 地址对应的主机或网络的全称域名。PTR 记录描述了这种 IP 地址与域名逆向映射的关系。

DNS 消息是如何传送的呢？我们看一个实际的 DNS 应答数据包，查询者请求解析 `livecc.auswan.cccgslb.com.cn` 这个域名，得到的 DNS 应答如图 5-6 所示。

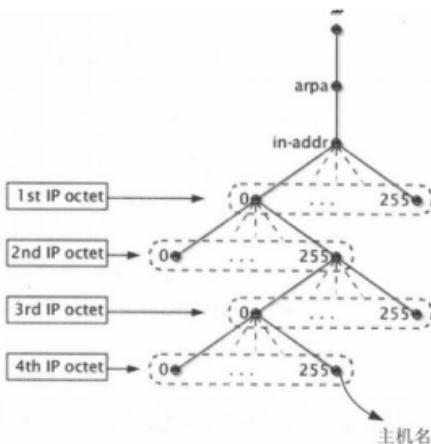


图 5-5 in-addr.arpa 域

```

    fvss001.rdb.cnc.ccgs1b.com.cn: type A, class IN, addr 60.217.249.178
    Name: fvss001.rdb.cnc.ccgs1b.com.cn
    Type: A (host address)
    Class: IN (0x0001)
    Time to live: 2 minutes
    Data length: 4
    Addr: 60.217.249.178 (60.217.249.178)

    fvss001.rdb.cnc.ccgs1b.com.cn: type A, class IN, addr 61.135.204.251
    Name: fvss001.rdb.cnc.ccgs1b.com.cn
    Type: A (host address)
    Class: IN (0x0001)
    Time to live: 2 minutes
    Data length: 4
    Addr: 61.135.204.251 (61.135.204.251)

```

|      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |           |            |          |         |         |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----------|------------|----------|---------|---------|
| 0000 | 00 | 1f | 16 | 1c | 7b | 00 | 18 | 82 | 3e | ac | a1 | 88 | 64 | 11 | 00 | ... | ...       | >...       | ..d..    |         |         |
| 0010 | 32 | 10 | 00 | ac | 00 | 21 | 45 | 00 | 00 | 46 | 4a | cc | 00 | 00 | 39 | 11  | 2...      | !E..       | ..J..    | .9.     |         |
| 0020 | e7 | c8 | ca | 64 | 26 | 97 | dd | d7 | 77 | d1 | 00 | 35 | ca | 0a | 00 | 92  | ...j...   | w..5..     | ..       | ..      |         |
| 0030 | 04 | 00 | 36 | 02 | 81 | 80 | 00 | 01 | 00 | 04 | 00 | 00 | 00 | 00 | 06 | 6c  | ..6...    | .....1     | ..       | ..      |         |
| 0040 | 69 | 76 | 69 | 76 | 03 | 06 | 61 | 75 | 73 | 44 | 61 | 6e | 03 | 63 | 6f | 6d  | livecc.au | swan.com   | ..       | ..      |         |
| 0050 | 00 | 00 | 01 | 00 | 01 | 00 | 0c | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00  | 00        | ..livecc.. | auswan.. | ..      | ..      |
| 0060 | 1d | 06 | 6c | 69 | 76 | 65 | 63 | 63 | 06 | 61 | 75 | 73 | 47 | 61 | 6e | 00  | ccgs1b.c  | on.cc..    | fvs00    | ..      | ..      |
| 0070 | 63 | 63 | 67 | 73 | 6c | 62 | 03 | 63 | 6f | 6d | 02 | 63 | 6e | 00 | c0 | 2f  | ccgs1b.c  | on.cc..    | fvs00    | ..      | ..      |
| 0080 | 00 | 00 | 05 | 00 | 01 | 00 | 00 | 07 | 09 | 00 | 12 | 07 | 66 | 76 | 73 | 30  | 01.rdb.c  | nc..x..    | ..x..    | ..x..   | ..x..   |
| 0090 | 30 | 31 | 03 | 72 | 64 | 62 | 03 | 63 | 6e | 63 | c0 | 3d | c0 | 58 | 00 | 01  | 01.rdb.c  | nc..x..    | ..x..    | ..x..   | ..x..   |
| 00a0 | 00 | 01 | 00 | 00 | 00 | 78 | 00 | 04 | 3c | d9 | f9 | 02 | c0 | 58 | 00 | 01  | ....x..   | ....x..    | ....x..  | ....x.. | ....x.. |
| 00b0 | 00 | 01 | 00 | 00 | 00 | 78 | 00 | 04 | 3d | 87 | cc | fb | 00 | 00 | 00 | 01  | ....x..   | ....x..    | ....x..  | ....x.. | ....x.. |

图 5-6 DNS 应答抓包截图

让我们来看看 DNS 的报文格式。DNS 的报文格式比较简单，主要是查询和回答报文，且查询和回答报文是相同的格式，如图 5-7 所示。

| 标识符               | 标志      |
|-------------------|---------|
| 问题数               | 回答 RR 数 |
| 权威 RR 数           | 附加 RR 数 |
| 问题 ( 问题的变量数 )     |         |
| 回答 ( 资源记录的变量数 )   |         |
| 权威 ( 资源记录的变量数 )   |         |
| 附加信息 ( 资源记录的变量数 ) |         |

图 5-7 DNS 报文格式

图 5-7 中各字段语义解释如下。

前 12 字节是首部区域，其中包括标识符、标志、问题数、回答 RR 数、权威 RR 数、附加 RR 数共 6 个字段。标识符字段 16bit，用于标识这个查询报文。标识符会被复制到对查询的回答报文中，以便让客户机用它来匹配发送的请求和接收到的回答。标志字段含有若干标志，1bit 的“查询/回答”标志位指出报文是查询报文 (0) 还是回答报文 (1)。当某 DNS 服务器恰好是被请求主机的权威 DNS 服务器时，1bit 的“权威的”标志位被放置在回答报文中。如果客户机希望 DNS 服务器执行递归查询，将设置 1bit 的“希望递归”标志位。如果该 DNS 服务器支持递归查询，在它的回答报文中会对 1bit 的“递归可用”标志位置位。在首部区域中，还有 4 个“数量”字段，分别给出了在首部后四类数据区域出现的数量。

问题区域包含正在进行查询的信息。该区域包括：名字字段，用于指出正在被查询的主机名字；类型字段，用于指出正在被询问的问题类型，例如是 A 类问题还是 MX 问题。

回答区域包含了对最初请求的名字的资源记录。每个记录中含有 Type ( 如 A、NS、CNAME、MX ) 字段、Value 字段和 TTL 字段。前面讲过，一个主机名可以对应多个 IP 地址，所以在一个回答报文的回答区域中可以包含多条 RR。

权威区域包含了其他权威 DNS 服务器的记录。

附加区域包含了其他一些有用的记录，例如，在一个 MX 请求的回答报文的回答区域包含了一条资源记录，给出了邮件服务器的规范主机名。该附加区域就可以包含一个类型 A 记录，提供对于该邮件服务器的规范主机名的 IP 地址。

读者可以尝试一下，在你的主机上使用 nslookup 程序就能够直接向一些 DNS 服务器发送 DNS 查询报文。nslookup 程序适用于多数 Windows 和 UNIX 平台。例如在一台 Windows 主机中打开命令提示符界面，直接键入“nslookup”命令即可调用 nslookup 程序，nslookup 接收到来自 DNS 服务器的回答后，会以可读方式显示包括在该回答中的记录。

### 5.2.3 基于 DNS 解析的 GSLB 工作方式

基于 DNS 解析的 GSLB 方案实际上就是把负载均衡设备部署在 DNS 系统中。在用户发出任何应用连接请求时，首先必须通过 DNS 系统来请求获得服务器的 IP 地址，基于 DNS 的 GSLB 正是在返回 DNS 解析结果的过程中进行智能决策，给用户返回一个最佳的服务器的 IP 地址。从用户的视角看，整个应用流程与没有 GSLB 参与时没有发生任何变化。

DNS 系统本身是具备简单负载分配能力的，这是基于 DNS 的轮询机制。如果有多台 Web 服务器同时为站点 [HTTP://www.CDNbook.com](http://www.CDNbook.com) 提供服务，CDNbook.com 的权威服务器可能会解析出一个或多个 IP 地址。权威域名服务器还可以调整响应中 IP 地址的排列方式，即在每次响应中将不同的 IP 地址置于首位，通过这种方式实现对这些 Web 服务器的负载均衡。我们回顾图 5-6 所示的例子，可以看到 DNS 系统反馈给客户端两个 IP 地址：60.217.249.18 和 61.135.204.251。这两个 IP 地址一个位于北京市，一个位于济南市。而客户端会首先尝试向 60.217.249.18 这个服务器发起请求，如果请求失败，才会向

61.135.204.251 发起请求。到下一个用户请求同样域名的解析时，权威域名服务器会在响应信息中将 61.135.204.251 排在首位，那么客户端就会首先尝试向它发起请求。这样，就通过 DNS 实现了两台服务器之间的负载均衡。

在具体应用中，基于 DNS 解析来实现 GSLB 有几种方法，下面我们分别讲述这几种方法。

### 1. 通过 CNAME 方式实现负载均衡

从前面的 DNS 记录类型介绍中我们了解到，CNAME 记录是描述一个域名或主机名的别名，域名服务器获得 CNAME 记录后，就会用记录中的别名来替换查找的域名或主机名。后面会查询这个别名的 A 记录来获得相应的 IP 地址。

通过 CNAME 方式来实现负载均衡，实际上是利用了 DNS 的两个机制：一是别名机制，二是轮询机制。具体操作简单地说就是：先将 GSLB 的主机名定义为所查询域名的权威 DNS 服务器的别名，然后将 GSLB 主机名添加多条 A 记录，分别对应多个服务器的 IP 地址。这样，本地 DNS 服务器会向客户端返回多个 IP 地址作为域名的查询结果，并且这些 IP 地址的排列顺序是轮换的。客户端一般会选择首个 IP 地址进行访问。

因为实现简单，并且不需要更改公共 DNS 系统的配置，所以通过 CNAME 方式实现负载均衡是目前业界使用最多的方式，在后面我们会用实际的例子来帮助读者再深入理解这种方式。

### 2. 负载均衡器作为权威 DNS 服务器

这种方式是把负载均衡器作为一个域名空间的权威 DNS 服务器，这样负载均衡器就会接收所有对这个域的 DNS 请求，从而能够根据预先设置的一些策略来提供对域名的智能 DNS 解析。

此时，负载均衡器已经接管了权威 DNS 服务器的所有解析工作，无论用户访问的网站是否是需要进行解析的域名，其 DNS 请求都会被发送到负载均衡器

上，所以整个域的 DNS 解析能力或多或少都会受到影响，影响的程度取决于负载均衡器能实现的 DNS 功能的程度，这因产品而异。事实上，有些产品，比如 F5 网络公司的 3DNS 具有完整的 DNS 功能以及增强的 GSLB 特性，Foundry、Nortel、Cisco 和 Radware 的产品能实现部分 DNS 功能。如果一个产品不能处理某些特定的 DNS 请求，它可能会选择丢弃请求，返回一个错误或者将请求转发到一个真实的 DNS 服务器上。

### 3. 负载均衡器作为代理 DNS 服务器

在这种方式下，负载均衡器被注册为一个域名空间的权威 DNS 服务器，而真正的权威域名服务器则部署在负载均衡器后面。所有的 DNS 请求都会先到达负载均衡器，由负载均衡器转发至真正的权威 DNS 服务器，然后修改权威 DNS 服务器返回的响应信息，从而实现负载均衡功能。

为实现这一过程，首先要将对外公布的权威 DNS 服务器的地址注册成负载均衡器上的 VIP 地址。真正的权威 DNS 服务器正常响应浏览器的 DNS 请求，返回域名解析结果列表，这个响应会先发送到负载均衡器，而负载均衡器会根据自己的策略选择一个性能最好的服务器 IP 并修改 DNS 服务器的应答信息，然后将应答信息转发给客户。负载均衡器只修改需要实现 GSLB 的域名的 DNS 查询响应，对其他请求透明转发，这样就不会影响整个域名空间的解析性能。

在代理方式下，如果权威 DNS 服务器与负载均衡器相隔很远，那么负载均衡器向权威 DNS 服务器转发 DNS 请求时就会产生额外的延时。解决这个问题的办法是，在负载均衡器上对 DNS 响应进行缓存，在缓存失效时间之前，对于之前已经从权威 DNS 服务器上获得过 DNS 响应的同一域名，就可以直接进行 GSLB 计算，响应用户 DNS 请求，不用去访问远程的权威 DNS 服务器了。

需要注意的是，在基于的 DNS 方式下无论采用何种工作方式，都会有一些请求不会到达 GSLB，这是 DNS 系统本身的缓存机制在起作用。当用户请求的域名在本地 DNS 或本机就得到了解析结果，这些请求就不会到达 GSLB。Cache

更新时间越短，用户请求到达 GSLB 的几率越大。由于 DNS 的缓存机制屏蔽掉相当一部分用户请求，从而大大减轻了 GSLB 处理压力，使得系统抗流量冲击能力显著提升，这也是很多商业 CDN 选择 DNS 机制做全局负载均衡的原因之一。但弊端在于，如果在 DNS 缓存刷新间隔之内系统发生影响用户服务的变化，比如某个节点故障，某个链路拥塞等，用户依然会被调度到故障部位去。

#### 5.2.4 负载均衡的策略判断条件信息

有些 DNS 服务器具有智能 DNS 功能，它在向本地 DNS 返回应答之前会先根据一些静态或动态策略进行智能计算。比如根据本地 DNS 服务器的 IP 地址进行就近性判断，进行响应时间或 IP 地址加权计算等，根据智能计算的结果选择一条或几条 A 记录返回给本地 DNS 服务器。返回的 A 记录可能直接是一台服务器的 IP 地址，或者是下一级均衡设备的 IP 地址。

那么，负载均衡计算所需的静态或动态策略所需要的信息都有什么呢？这些信息包括如下几项。

##### **(1) 服务器的“健康状况”**

GSLB 最重要的特性之一就是持续不断地监控各服务器的健康状况，健康检查的类型有多种，就像服务器健康检查一样，从二层、三层到四层，甚至是七层。未能通过健康检查的服务器不能作为备选的域名解析结果。

##### **(2) 地理区域距离**

这里的距离指用户本地 DNS 服务器的 IP 地址与服务器 IP 地址之间的路由距离。由于 DNS 系统本身的工作原理所限，GSLB 只能看到用户本地 DNS 服务器的 IP 地址，看不到用户终端的 IP 地址。

当用户使用错误的本地 DNS 服务器地址时，GSLB 返回的域名解析结果将不是最佳的答案。但由于绝大部分运营商都会限制其他运营商的用户使用自己

的 DNS，所以出现这种错误配置的比例比较小，除非用户在终端上手动设置了错误的本地 DNS 服务器。

### (3) 会话保持

会话保持是一些业务的特殊要求，比如大多数电子商务应用系统或者需要进行用户身份认证的在线系统中，一个客户经常需要与服务器交互几次才能完成一笔交易或者一个请求。由于这几次交互是密切相关的，服务器在一次交互时需要了解上一次或上面几次交互的处理结果，因此要求所有这些相关的交互过程都由一台服务器处理，而不能被分散到不同的服务器上。典型的例子就是基于 HTTP 的电子商务业务，一个客户完成一笔交易可能需要多次点击，而一个新的点击产生的请求，可能会重用上一次点击建立起来的连接，也可能是一个新建的连接。会话保持就是指在负载均衡器上设置这么一种机制，可以识别客户与服务器之间交互过程的关联性，在做负载均衡的时候考虑保证这些相关联的访问请求分配到同一台服务器上。如果 GSLB 的返回结果是本地负载均衡器（SLB），那么同样需要保证关联访问被调度到同一个 SLB 上，SLB 继续执行会话保持策略，保证用户访问被调度到同一台服务器上。

### (4) 响应时间

通过计算一个请求/响应的往返延迟时间，GSLB 能够计算出每个服务器的响应时间。最简单的方法就是通过测量服务器对健康检查的响应快慢来计算出站点的响应时间。

不过应该指出的是，如果服务器以集群或群组方式部署，那么 GSLB 测量出的响应时间实际上是这个集群或群组里面最好的服务器的响应时间。因为 GSLB 的健康性检查请求首先会发送到集群或群组的本地负载均衡器上，本地负载均衡器会像对待普通用户请求一样将请求转发给下面压力最小的服务器，下一个请求，可能被转发到另一台服务器。由于每次请求的响应时间可能有差异，所以最好的办法是计算一段时间的平均值。CDN 系统可以设置本地负载均

衡器负责计算本集群或本群组之内所有服务器的平均响应时间，并报告给 GSLB。

### (5) IP 地址权重

这是 GSLB 预先为每个 IP 地址分配的权重值，这个权重值取决于一些商业方面的考虑，比如某些地域的 IDC 带宽比较贵，CDN 系统就会尽量少用这些昂贵的资源。权重值决定了某个 IP 与其他候选 IP 相比分配到的流量比例。

### (6) 会话能力阈值

GSLB 控制器可以获得每个服务器当前可用的会话数和会话表大小的最大值，当前会话数/最大会话数比值超过定义的阈值时，该服务器将不再被选择。

### (7) 往返时间 (RTT, Round-Trip Time)

往返时间指用户向网站发出请求到从网站收到响应的时间间隔。这个时间是从用户的角度出发来衡量 CDN 加速效果的指标，它基本反映了用户与 CDN POP 节点之间的链路健康状况。测量往返时间可以由 CDN 发起来完成，也可以由用户发起来完成。如果由用户来发起测量，最后得到的数据会比较精确，但需要在浏览器中内嵌特定的客户端程序进行通信和统计，而这种情况往往较少，所以多数情况下还是需要 CDN 来完成。CDN 完成往返时间测量并用于 GSLB 的调度的方式有被动测量方式和主动测量方式，这两种方式各有优劣，可以在不同情况下使用，也可以混合使用。

### (8) 其他信息，包括服务器当前可用会话数、最少选择次数、轮询等

上面讲述了几种 GSLB 在选择服务器时会用到的信息，关于这些信息更具体的描述和相关算法会在本书的 5.3.2 节进行讲解。下面我们将看到 GSLB 是如何把这些信息组合起来实现全局负载均衡的。

第一种方式：依照条件重要性逐步去除不符合要求的服务器。比如，GSLB

设置的策略优先级依次是健康性、响应时间、负载情况。那么 GSLB 会首先排除不能提供服务的服务器，然后基于响应时间排除能力较差的服务器，最后基于负载情况选出负载较轻的服务器。如果此时还是有多个选择的话，GSLB 会采用默认的方法来进行选择，比如轮询。

第二种方式：采用加权法挑选最优服务器。通常在进行加权计算前，GSLB 会先使用健康检查和就近性排除大部分不能提供服务或距离过远的服务器。然后根据预先配置的加权策略，对其余每一项指标计算一个加权值，例如，给响应时间分配一个较高的权值，为路由开销分配一个较低的权值，同时考虑多个指标，从中选出条件最佳的服务器。

GSLB 回复给本地 DNS 服务器的应答信息有两种，一种是只回复它选出的最佳服务器的 IP 地址，另一种是回复多个服务器的 IP 地址列表，但是将它选择的最佳服务器 IP 地址放在第一位。第一种方式的缺点在于，如果 GSLB 推荐的服务器突然发生故障，而关于这个域名的 DNS 应答信息缓存在浏览器和本地 DNS 服务器中，那就只能等浏览器和本地 DNS 服务器记录过期，才能重新发起解析。在这期间的用户请求都会被导向故障服务器。第二种方式的缺点在于，很多本地 DNS 服务器会以轮询方式从地址列表里选择一个 IP 地址返回给客户端，而不一定是 GSLB 认为的最佳服务器。

### 5.2.5 开源 DNS 服务软件——BIND

BIND (Berkeley Internet Name Domain) 是一款遵循 BSD 许可证的开放源码的 DNS 服务软件，最早由美国加州大学伯克利分校 (UCB) 的 4 名研究生在 20 世纪 80 年代初期开发的，并由 UCB 的 Computer Systems Research Group (CSRG) 进行维护，直至版本 4.8.3。到了 20 世纪 80 年代中期，DEC 公司全面接手 BIND 的开发和维护工作，并先后发布了 4.9 和 4.9.1 两个版本，而后续的 4.9.2 版本则是在 DEC 公司前员工 Paul Vixie 创立的 Vixie Enterprise 的支持下开发的。从 4.9.3 版本起，BIND 的研发转入 Internet Software Consortium

(ISC) 名下负责，并先后在 1997 年 5 月和 2000 年 9 月分别发布了 BIND 8 和 BIND 9。BIND 9 对从前版本的 BIND 软件中存在的安全问题进行了全面修正，并成为当前互联网上使用最为广泛的 DNS 服务软件。

BIND 主要由一个负责域名解析及反解析的进程和一个域名解析库组成。当 BIND 服务器接收到一个域名查询请求时，它首先在本地的域名信息文件和缓存中查找是否有相应记录。如果没有命中，BIND 将递归查询能够解析这些域名的其他 DNS 服务器地址，直到获得正确的地址或者最终报错。

BIND 作为成熟的 DNS 服务软件，能够非常便捷地实现负载均衡。其基本原理是在 BIND 服务器中为同一个主机名配置多个 IP 地址，BIND 服务器在应答 DNS 查询时将按照 DNS 文件中主机记录的 IP 地址按顺序返回不同的解析结果，从而将用户发来的访问引导到不同的服务器上，从而达到负载均衡的目的。利用 BIND 进行负载均衡简单易行，而且服务器可以位于互联网上的任何位置。因此，BIND 非常适合于承担全局负载均衡的工作。例如，一个站点在多个区域内都拥有自己的服务器，为了使所有用户只使用一个域名就能访问到距离自己最近的服务器从而获得最快的访问速度，BIND 就可以身兼 DNS 服务器和负载均衡设备两个角色，为站点提供更优化的服务，将用户请求指派到最合适的服务器上。

但是，当前 BIND 负载均衡算法通常只是按照一定顺序的轮询操作，并不关心服务器的配置差异，也不检测服务器的当前运行状态，所以难以充分发挥高性能服务器的处理能力，不能为之指派更多的用户请求，甚至会出现用户请求集中在某一台服务器上的情况。同时，BIND 实现负载均衡还会受到自身提供的 DNS 服务的影响，例如本地 DNS 服务需要和其他 DNS 服务器经常交互，以及时更新 DNS 数据确保地址的正确随机分配，因此 BIND 服务器中的 DNS 刷新时间通常被设置得很短，然而过于频繁的刷新可能会增加 DNS 流量从而引入额外的网络压力；另外，如果某台服务器出现了故障，即使相关的 DNS 设置被及时修改调整，但还是要等待足够长的刷新时间才能发挥作用，而在此期间，

之前通过负载调度分配给用户终端访问的服务器地址仍将被故障服务器持有，从而不能被正常访问。为了改进 BIND 的应用效果，很多网站和 CDN 运营商都结合自身需要调整了 BIND 的配置参数（例如，轮询顺序、DNS 刷新间隔等），同时还对负载调度策略进行了更细的划分，例如 BIND 9 中提供的 VIEW 语句用于支持多链路负载均衡，有效地为来自不同 IP 地址段的 DNS 查询返回不同的域名解析结果。

当前，BIND 9 还在不断的改进中，BIND 10 的研发也已经被提上日程。BIND 官方网站的网址是 <http://www.isc.org/software/bind>，网站上提供了最新版本的 BIND 的下载链接和相关的软件手册。

## 5.3 基于 DNS 的 GSLB 应用部署方法

### 5.3.1 GSLB 应用部署时的一些基本概念

本节我们通过分析 GSLB 如何在 CDN 中的应用来进一步理解 GSLB 的工作原理。在此之前，需要介绍几个相关概念，以便帮助读者理解 CDN 中的逻辑资源是如何被 GSLB 划分，GSLB 又是如何根据这些资源划分完成复杂的调度工作的。

**域组（Domain Group）。**当网站使用 CDN 提供服务并用 GSLB 来解析其源站域名时，通常会由权威 DNS 设置一个 GSLB 认可的域名作为原域名的别名（CNAME）。有了别名，GSLB 就可以将它唯一映射成一个由多个虚拟服务器（Virtual Server）组成的服务池（Pool），这些虚拟服务器就是用户输入网站 URL 后经负载均衡调度直接提供服务的服务器。GSLB 在解析域名的时候会直接返回其中一个虚拟服务器的 IP 地址。比如 www.netitv.com.cn 域名 CNAME 设置为 www.netitv.cdn.com.cn。如果需要 GSLB 来进行解析 www.netitv.com.cn

## CDN 技术详解

的 IP 地址，那么就需要在 GSLB 上面设定域组 www.netitv.cdn.com.cn，这样 GSLB 就会按照内部设置的负载均衡策略，选择相关联的 Pool 以及虚拟服务器来提供服务了。

**服务池（Pool）**。如前面在域组的介绍里所述，Pool 是一个包含多个虚拟服务器的逻辑组概念。GSLB 在进行域名解析的时候，可供选择的虚拟服务器有多个，这些提供同种业务的多个虚拟服务器的集合就是服务池。GSLB 根据域组选择了服务池后，就会在负载均衡策略下，选择提供服务的虚拟服务器。

**虚拟服务器（Virtual Server）**。是用户访问网站时直接提供服务的虚拟设备，其 IP 地址就是 GSLB 最终解析返回的地址。在 CDN 的实现中，有一种典型应用，虚拟服务器在物理上可以看成是由四层交换设备进行负载均衡的一组服务器设备，如图 5-8 所示。四层交换设备与相连的服务器对外表现为一个或多个“IP 地址+端口”，每一个“IP 地址+端口”标识一个虚拟服务器的地址，也对应了一定的服务器资源。虚拟服务器之所以称为“虚拟”，是因为一个虚拟服务器可能对应的是多台物理服务器，或者一个物理服务器的资源被划分给多个虚拟服务器来使用。图 5-8 中示意了三个虚拟服务器，其中对应域组 www.b.cdn.com 的虚拟服务器地址为 192.168.1.4，端口为 80，它所对应的物理服务器资源为服务器 B 和服务器 C。

**区域（Region）**。就是用户的本地 DNS 所代表的地址范围，用户可以通过在浏览器中设置本地 DNS 地址来自定义所在区域。基于实际地域来区分区域是一种常见的应用，另一种常见的区域划分方式是基于不同电信运营商的，比如根据本地 DNS 所在运营商网络的不同，给中国电信的用户返回电信网内某虚拟服务器或者权威 DNS 的地址，给中国联通的用户返回联通网内某虚拟服务器或者权威 DNS 的地址。

**策略**。指通过脚本并按照一定规则来实现对 GSLB 的路由策略的控制方式。例如 GSLB 可以通过策略控制对同一域组的浙江用户使用 A 记录解析，而对四川用户使用 NS 方式解析。

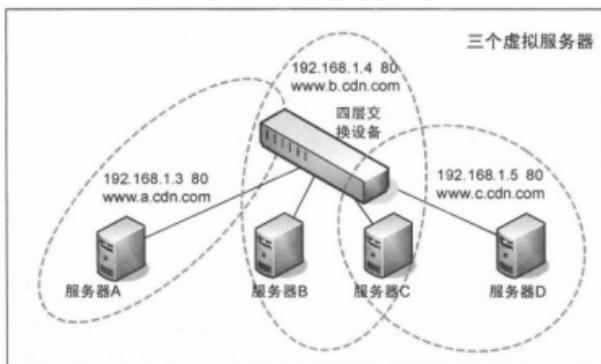


图 5-8 虚拟服务器的实际应用

图 5-9 大致描述了域组、服务池、虚拟服务器和区域的关系。

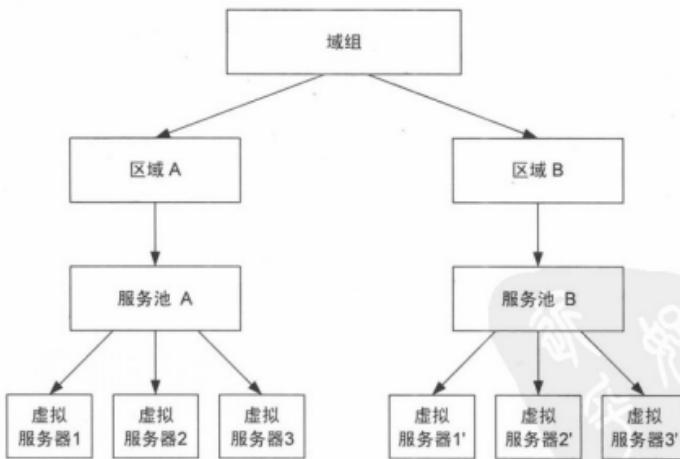


图 5-9 域组、服务池、虚拟服务器和区域的逻辑关系

对于以上关系，我们简单说明一下，以 [www.netity.com.cn](http://www.netity.com.cn) 为例，网站

CNAME 到 CDN 的 GSLB 上的域名为 [www.netitv.cdn.com.cn](http://www.netitv.cdn.com.cn)。在这个域组下，GSLB 能够根据用户的本地 DNS 地址将用户所属区域划分为南方或者北方，北京和西安各有一个虚拟服务器为北方用户提供服务，上海和广州各有一个虚拟服务器为南方用户提供服务。北京和西安的虚拟服务器一起组成一个服务池，上海和广州的虚拟服务器组成另一个服务池。

如果一个来自南京的用户访问 [www.netitv.com.cn](http://www.netitv.com.cn) 网站，在域名解析过程中，GSLB 判断存在 [www.netitv.cdn.com.cn](http://www.netitv.cdn.com.cn) 的域组，然后就根据区域的记录从南方区域的服务池中寻找合适的虚拟服务器来提供服务，最后 GSLB 根据就近性和其他负载均衡策略选择了上海的虚拟服务器并向用户的本地 DNS 返回了其 IP 地址和端口信息。如果上海的虚拟服务器发生故障宕机，GSLB 将会修改服务池中的虚拟服务器信息，南京的用户再次访问时，域名就被直接解析为广州的虚拟服务器地址了。

上面说到了 GSLB 在解析域名进行调度时所遵循的逻辑概念，一些主流的 GSLB 设备就是通过这逻辑定义来进行配置的，不过这些看起来有点难以理解，我们来看一个在实际组网时遇到的情况。

业内在谈论 CDN 组网时经常会说到 POP 节点的定义，POP 节点在物理设备组成上通常是一个服务器集群。对于前面所说的模型，POP 节点在很多时候就是一个虚拟服务器（相关内容可参见本书第 4 章中的集群内容），由四层交换机设备和后挂的多台 Cache 服务器组成。但是对应到上面说的逻辑模型，POP 节点往往就是一个虚拟服务器。假设 CDN 不是按上面说的只分成南方和北方两个独立的区域，而是将每个省看成一个独立的区域（这往往比较符合 CDN 网络的实际情况），那么每个省里的所有 POP 节点的集合就可以看成是一个服务池，这时 GSLB 就可以按照一定的负载均衡算法在这个服务池中选择最适合用户的 POP 节点来提供服务。比如 CDN 运营商在浙江省内部署了两个 POP 节点，浙江用户在访问通过 CDN 加速的网站时，GSLB 就会在“浙江”这个区域内选择资源使用较低（或者其他均衡策略）的 POP 节点来提供服务。

有些 CDN 运营商是独立对外提供服务的，他们可能在多个不同的电信运营商网络中都部署有自己的 POP 节点，比如在同一个区域的中国电信 IDC 机房和中国联通 IDC 机房都部署有自己的 POP 节点，这样做是为了更好地服务于不同电信运营商的宽带用户。注意，这里不要把电信运营商和 CDN 运营商混淆，电信运营商有可能自己也运营 CDN 业务，但很多时候，CDN 运营商是独立的 CDN 业务提供者，他们通常都会租用电信运营商的基础网络资源（比如 IDC）。对于一个覆盖多个运营商的独立 CDN 网络来说，通常 GSLB 设备会从逻辑上分为两层，第一层 GSLB 用来区分不同电信运营商网络，第二层用来区分不同的省份。比如，第一层 GSLB 会在区域中分别设置电信和联通 IP 地址段信息，第二层 GSLB 会在区域中分别设置每个运营商网络中各个省的 IP 地址段信息。

当用户访问 [www.netityv.com.cn](http://www.netityv.com.cn) 网站，网站权威 DNS 将域名 CNAME 到第一层的 GSLB 设备上（[www.netityv.cdn.com.cn](http://www.netityv.cdn.com.cn)），第一层 GSLB 看到本地有 [www.netityv.cdn.com.cn](http://www.netityv.cdn.com.cn) 的域组设置，则继续根据用户本地 DNS 的地址来判断是电信用户还是联通用户。假设用户是电信用户，那么第一层 GSLB 将域名又 CNAME 到第二层 GSLB 设备上（[www.netityv.ct.com.cn](http://www.netityv.ct.com.cn)），第二层 GSLB 看本地有这个域组，则继续根据用户本地 DNS 的地址来判断是哪个省的用户，找到这个省部署的所有 POP 节点，并使用负载均衡的策略选择了其中一个 POP 节点来提供服务。以上过程如图 5-10 所示。

从图 5-10 中可以看到，第一层 GSLB 和第二层 GSLB 都有各自的域组 [www.netityv.cdn.com.cn](http://www.netityv.cdn.com.cn) 和 [www.netityv.dx.com.cn](http://www.netityv.dx.com.cn)。第一层 GSLB 通过区域设置，将整个服务池分为电信的服务池和联通的服务池，第二层 GSLB 通过区域设置，将电信的服务池分为各省的服务池。这里的服务池就是提供相同业务的所有 POP 节点的组合，各省的服务池包含两个 POP 节点，POP 节点也是 GSLB 在调度配置中所认识到的虚拟服务器。GSLB 通过负载均衡策略最终返回一个 POP 节点地址，用户直接访问 POP 节点来获取网站缓存内容。

## CDN 技术详解

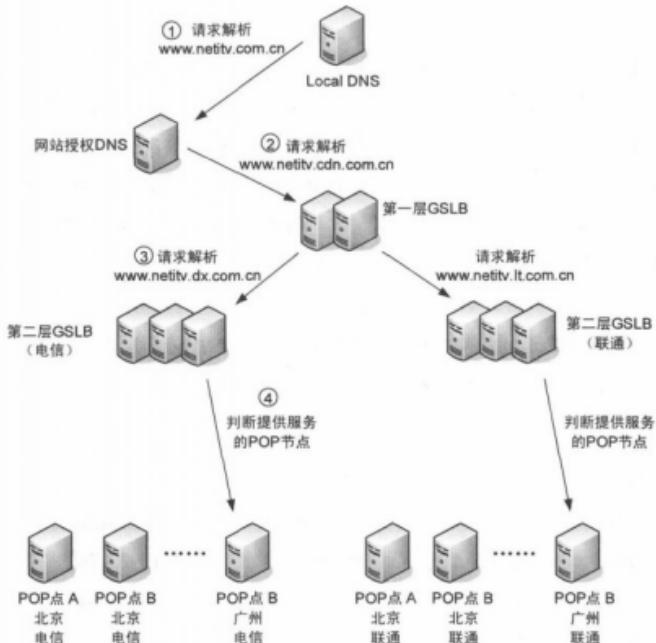


图 5-10 GSLB 在实际 CDN 中的应用

值得注意的是，GSLB 在进行调度时所使用的这些逻辑概念与在具体的应用中对应的实际网络设备会有差别，比如以上例子中提到带四层交换设备的 POP 节点是虚拟服务器的应用，但在某些应用中，POP 节点可能没有四层交换设备，而是直接由几个 Cache 服务器组成的，这样 GSLB 可能在配置中将每台 Cache 设备作为虚拟服务器来对待。

### 5.3.2 负载均衡策略

目前 GSLB 用来进行负载均衡的策略有多种，但总的来说分为两类：静态策略和动态策略。静态策略是可以在 GSLB 设备中直接进行配置的策略，比如

通过配置不同 POP 节点提供服务的 IP 地址段。而对于动态策略，GSLB 需要实时获取节点资源和网络状况的最新数据以调整调度的方式。

### 1. 静态策略

#### (1) 基于特定的用户源 IP 地址

基于用户源 IP 地址的策略与前面所说的区域概念类似，GSLB 可以设置成将特定的 IP 地址段定向到特定的 POP 节点或者虚拟服务器上。比如将 10.2.0.0/24 网段的请求都解析到地址为 10.2.4.200 的节点。

#### (2) 基于加权的 IP 地址

与前面基于用户区域的 IP 地址不同，这里参考依据的 IP 地址是服务池中的虚拟服务器的 IP 地址。GSLB 可以为服务池中各个虚拟服务器的 IP 地址分配一个不同权重，比如为 10.1.3.101 分配 80 的权重以及为 10.2.4.22 分配 20 的权重。这样 GSLB 在为用户请求轮询解析域名时，有 80% 的概率会选择 IP 地址为 10.1.3.101 的虚拟服务器。

#### (3) 基于加权的 POP 节点

基于加权的 POP 节点与基于加权的 IP 地址类似，不过后者是基于逻辑上的虚拟服务器，前者是基于物理上的节点。比如为北京 POP 节点分配 80 的权重以及为上海 POP 节点分配 20 的权重。

#### (4) 基于地理位置

GSLB 在配置区域后可以根据用户的本地 DNS 的 IP 地址来确定用户的地理位置。所以根据就近性原则，可以选择一个在地理位置上与用户距离最近的 POP 节点或者虚拟服务器来提供服务。

#### (5) 基于 POP 节点管理优先级

GSLB 为每个 POP 节点预设的介于 0~255 之间的优先级，当根据其他策

略得出的服务的 POP 节点有多个时，选择优先级较高的 POP 节点来提供服务。比如通过地理位置判断长沙的用户可以由武汉节点或者广州节点提供服务，但由于广州节点的优先级 220 大于武汉节点的 200，所以由广州节点来提供服务。

### (6) 基于简单的轮询

这是最简单的负载均衡策略，也是 DNS 系统最常用的均衡方式，即针对每个解析请求对所有可提供服务的 POP 节点进行依次轮询。比如提供服务的节点只有 10.1.3.101 和 10.2.4.22，那么第一个用户请求解析为 10.1.3.101，第二个用户请求解析为 10.2.4.22，第 3 个用户的请求又是 10.1.3.101，这样依次反复来提供解析。因此，每个服务节点为区域内的所有用户提供服务的概率是相等的。

### (7) 基于成本

不同 CDN 运营商在对外提供服务时，收费方式存在一定差别。有的 CDN 运营商因为各地 IDC 租用成本的差异而针对不同的 CDN 的 POP 节点卖出不同的价格（比如沿海 POP 节点较贵，西部 POP 节点较便宜）。这种情况下，GSLB 可以优先将所有的访问请求解析到成本最低的 POP 节点，在较低成本的 POP 节点流量达到阈值后再解析到成本较高的节点。

还有的 CDN 运营商可能会对网站租用的每个 POP 节点收取一个固定费用（类似于电信运营商宽带或移动的基本套餐费用），并保证一个固定数值的流量服务。如果 POP 节点的真实流量超过了这个固定数值，将针对超出的每单位流量进行额外收费。这种情况下，GSLB 在进行域名解析时可以先考虑让所有 POP 节点都达到固定流量，然后再考虑将额外的流量按照节点成本依次分配到各节点上。

基于成本的策略通常需要与其他策略一起使用，比如先使用基于地理位置的策略，然后再使用基于成本的策略，这样 GSLB 获得的服务池中应该只包含离用户较近的多个 POP 节点或者虚拟服务器，也就不会让离用户太远的便宜节点去服务本地的用户（比如乌鲁木齐的 POP 节点去服务上海的用户），否则就

失去 CDN 的意义了。

## 2. 动态策略

静态策略配置比较简单，调度效率较高，但由于没有考虑实时情况，均衡效果可能不够准确，均衡后各节点的负载情况仍然存在失衡的情况。静态策略在一段时间内能够反映网络流量和用户的分布情况，但过了一段时间后就不适用了，如果配置来不及更改，CDN 的效率将受到严重影响。动态策略是根据网络和服务器资源的实时情况来调整调度方式的策略，均衡效果好，能保证不会出现长时间的失衡，但其配置复杂，调度算法影响调度效率，服务器和网络为获取资源使用情况所付出的额外开销也较大，因此静态策略和动态策略在实际的 CDN 中都是结合使用的。比如在一些需要特殊保障的访问中，静态策略应用较多；在针对一次解析请求的两层调度中，第一层调度（区域间调度）采用静态调度较多，第二次调度（区域内调度）采用动态调度较多。

GSLB 所使用的动态策略包括如下几项。

### (1) 基于 POP 节点健康状况

GSLB 会定期对各 POP 节点或虚拟服务器进行四层和七层的健康检查，比如通过查询的方式来获取节点或者虚拟服务器的各种健康信息，如果 GSLB 收不到响应信息或者响应信息里包含了健康状况存在问题的信息，相关节点和虚拟服务器将不会被选为最佳的服务节点。

### (2) 基于相对会话能力

每个 POP 节点或者虚拟服务器都会保留一张会话表，每一个 TCP 或 UDP 会话都占用会话表中一个表项。GSLB 在统计相对会话能力时会用到当前会话数和最大会话数两个数据，当前会话数反映了 POP 节点或者虚拟服务器当前的负载情况，最大会话数反映了最大负载能力。GSLB 可以周期性地计算当前会话数与最大会话数的比值，然后与会话能力阈值进行比较。如果 POP 节点或者

虚拟服务器的会话数超过了会话能力阈值，GSLB 就不会再将此节点或虚拟服务器选为提供服务的最佳节点，这样就可以避免一个 POP 节点过分拥堵。

### ( 3 ) 基于绝对会话能力

基于绝对的会话能力不会去比较当前会话数与最大会话数的百分比，而是将当前会话数与设定好的阈值进行比较。如果 POP 节点的当前会话数超过这个阈值则不会将此节点或者虚拟服务器作为服务的最佳节点。

### ( 4 ) 基于物理服务器绑定

一个虚拟服务器可能与多个运行的物理服务器绑定，有的虚拟服务器绑定的物理服务器较多，有的绑定的服务器较少。GSLB 在为用户选择提供服务的虚拟服务器时，会考虑将绑定物理服务器较多的虚拟服务器提供给用户。

### ( 5 ) 基于主动测量的用户访问往返时间 ( Round-Trip Time )

前面说过，RTT 测量分被动测量和主动测量两种方式，各有优劣，可以在不同情况下使用，也可以混合使用。

主动测量方式是各 POP 节点或者虚拟服务器主动判断自己与用户本地 DNS 往返时间的方式。当 GSLB 收到来自用户本地 DNS 的解析请求时，GSLB 会通知所有 POP 节点或虚拟服务器对自己到此本地 DNS 的往返时间进行测量。根据采集到的往返值，GSLB 会选择其中值最小的 POP 节点或虚拟服务器的 IP 地址返回给本地 DNS。

主动测量方式能实时地、基本准确地反映用户与访问节点之间的链路时延，但也存在一定的弊端。由于主动测量方式采用 DNS 询问或者使用 ICMP 协议进行实时的频繁测量，在有些网络中可能会被安全策略所过滤而无法工作。另外，即使不被安全策略影响，主动测量产生大量额外的 DNS 询问或 ICMP 流量，大大降低网络利用率。

### (6) 基于被动测量的用户访问往返时间

被动测量方式不会主动去进行测量，也不会产生额外的数据流量，而是在用户向返回的 POP 节点或虚拟服务器建立连接时进行采集所得到的。测量的具体算法一般是从站点收到一个用户发出的连接请求（发送 TCP SYN）到接收到用户的确认（收到 TCP ACK）所经历的时间，而不是简单的 Ping 的响应时间，这样可以更精确地衡量访问最快的 POP 节点。GSLB 会定期从各节点获得往返时间的统计，用做选择站点的依据。

在被动测量中，由于访问网站的用户数量众多，GSLB 不可能将 POP 节点所统计的每个用户的往返时间保存起来当做节点选择的依据，所以 GSLB 将单独的用户 IP 地址汇聚成 IP 子网来进行记录，对应的往返时间也是这个 IP 子网中各 IP 地址往返时间的平均值，这也同 Internet 上 IP 地址总是成块分配的情况相符。

这里简单描述一下被动测量获取往返时间的过程（但应注意，不同设备在具体实现上是存在一定差异的）。

①初始情况下，CDN 所有 POP 节点都没有用户访问，GSLB 将所有 IP 子网到每个站点的往返时间都设置为一个极大的数（在实际中不可能出现的时间）。

②IP 子网 10.2.0.0/24 的用户通过本地 DNS 向 GSLB 请求域名解析，GSLB 随机选择了节点 BJ 返回给本地 DNS。

③IP 子网 10.2.0.0/24 的用户与节点 BJ 建立连接，节点 BJ 记录相关往返时间，并向 GSLB 报告，GSLB 数据库保存 10.2.0.0/24 子网与节点 BJ 的往返时间。

④接下来，又有来自 IP 子网 10.2.0.0/24 的用户在通过 GSLB 进行域名解析时，GSLB 会有 95% 的概率去参考先前记录的往返时间，而剩下 5% 的概率不参考往返时间，只是随机选择一个节点为用户提供服务。假设子网 10.2.0.0/24 的

另一个用户发出解析请求，GSLB 按照 5% 的概率随机选择了节点 SH 返回给用户。

⑤ 用户与节点 SH 建立连接，节点 SH 记录了 10.2.0.0/24 子网与它之间的往返时间，并向 GSLB 报告，GSLB 数据库保存 10.2.0.0/24 子网与节点 SH 的往返时间。

⑥ 如果 GSLB 在解析域名时，按照 95% 的概率选择参考往返时间，则会在记录的往返时间里选择一个最佳的节点提供服务（GSLB 可能会根据往返时间为每个节点分配一个权重）。

⑦ 接下来的用户请求都会按照④到⑥的步骤不断反复。最终 GSLB 会获取每个用户子网到不同 POP 节点的往返时间并不断更新这个时间。

以上是一个被动测量过程的例子，具体多大概率参考返回时间是可以人为设置的。被动测量方式能够更真实地反映用户的上网感受，在运营商网络中不会产生额外流量，也不会受到其他运营商或网络安全策略的影响。

### ( 7 ) 基于新建连接数

前面提到的相对会话数和绝对会话数都是通过当前会话数来衡量 POP 节点或虚拟服务器的负载情况，而当前会话数是 POP 节点或虚拟服务器上并发的 TCP/UDP 连接数。除了 TCP/UDP 并发连接数以外，每秒新建连接数也是衡量负载情况的一个重要指标，它指节点或者虚拟服务器每秒平均完成多少个 TCP/UDP 连接的建立。如果该数值高于预设的连接上限阈值，则该节点将不会被选择用来提供服务。这种基于新建连接数的策略在用户访问内容为 Web 页面时比较有效，因为 Web 页面的访问对象通常都是小文件，连接请求频繁且每次连接持续时间较短，反复新建连接和拆除连接对服务器 CPU 的消耗相对于其他资源更明显，更能反映 Web Cache 节点或虚拟服务器的负载能力。

### ( 8 ) 基于流量

主要指 POP 节点或虚拟服务器每秒完成多少比特数据的吞吐。基于新建

连接数的策略比较适合 Web 访问的负载均衡，而基于流量的策略则比较适合访问内容为流媒体的调度。用户对流媒体的访问通常都是大文件，每个连接的带宽要求高且每次连接持续时间较长，高带宽的吞吐对服务器磁盘 I/O 和网络接口 I/O 的消耗相对于其他资源更明显，更能反映流媒体 Cache 节点或虚拟服务器的负载能力。

#### (9) 基于 POP 节点访问次数

与静态策略中基于 POP 节点管理优先级的策略类似，这个策略通常与其他策略一起使用，在通过其他策略得到多个 POP 节点或者虚拟服务器时，可以使用访问次数最少的节点或者虚拟服务器来提供服务。

### 3. 算法实例

前面我们了解到 GSLB 实现负载均衡时的一些关键参考因素，不同厂家 GSLB 设备所使用的策略各有不同。有的厂家只适用静态策略来进行调度，有的厂家只使用动态策略，还有的厂家则两者结合一起使用。静态策略是由网络管理者在 CDN 管理平台中设置好下发给 GSLB 的，下发的信息是静态策略以及相对应的具体参数，比如静态策略是基于加权的 POP 节点，参数是每个站点的具体权重值；动态策略也是由网络管理者在 CDN 管理平台中进行设置并下发的，但下发的信息是动态策略以及对应的使用原则，这里的使用原则是指如何根据动态策略的变化来调整均衡结果的原则，也就是我们常说的负载均衡算法。

这里举例描述两个基于动态策略的负载均衡算法，一个基于流量负载（一个维度），另一个同时基于往返时间和流量负载（两个维度）。

#### (1) 算法 1：基于流量的负载均衡算法

这种只考虑流量因素，这在某些实际应用中是适用的，比如服务站点以流媒体 Cache 为主，每个用户到可供服务的各个 POP 节点之间的链路状况基本一

致，各 POP 节点的服务成本也基本相同，流量是决定节点负载均衡的最主要因素。

本算法主要采用负反馈的原理来控制各个 POP 节点上的流量，负反馈是一种基于偏差的调度算法。简而言之，当系统输出同期望值不相等时，控制系统根据系统输出和期望值之间的偏差来对系统施加控制，负反馈原理如图 5-11 所示。

从图 5-11 看到，当实际输出大于期望输出时，产生一个正的差异量，经过算法模块后产生一个负的控制量来减小系统的实际输出；相反，当实际输出小于期望输出时，差异量为负，经过算法模块后产生一个正的控制量来增大系统的实际输出。如果应用到负载均衡算法中，当某个 POP 节点的流量（实际输出）同它的负载能力（期望输出）相比偏小时（偏差为负），应该通过算法增大分配给该节点的流量，反之，则减小分配给该节点的流量。

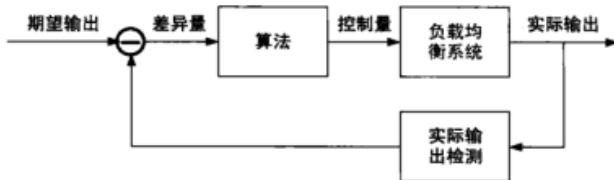


图 5-11 负反馈基本原理

具体的算法描述如下：

①在提供服务之前，GSLB 通过网管配置获知每个 POP 节点可供服务的最大带宽以及所有 POP 节点的最大服务带宽之和，比如节点 i 的最大服务带宽为  $B_i$ ，所有节点的最大服务带宽之和为  $B_{sum}$ 。然后 GSLB 计算每个  $B_i$  与  $B_{sum}$  的比值  $R_{i,B}$ ，并把它作为每个节点的期望负载能力保存起来。

②每个 POP 节点周期性地向 GSLB 上报本节点的流量信息，GSLB 使用  $T_i$

来记录每个节点当前的流量信息以及  $T_{sum}$  来记录所有节点当前流量之和，其中  $i$  表示节点  $i$ 。然后 GSLB 计算每个  $T_i$  与  $T_{sum}$  的比值  $R_{i,T}$ ，并将它与先前保存的  $R_{i,B}$  进行比较，即使用  $R_{i,T}$  减去  $R_{i,B}$ ，得到  $D_i$ 。

a) 如果  $D_i$  大于 0，则说明  $R_{i,T}$  大于  $R_{i,B}$ ，需要减小分配给此节点的流量。

b) 如果  $D_i$  小于 0，则说明  $R_{i,T}$  小于  $R_{i,B}$ ，需要增大分配给此节点的流量。

③GSLB 根据所得  $D_i$  的大小为每个  $D_i$  分配一个权重值。通常  $D_i$  越小，所分配的权重值  $W_i$  越大； $D_i$  越大，所分配的权重值  $W_i$  越小。比如  $D_1$  为 -0.3，分配的权重值  $W_1$  为 30； $D_2$  为 0.2，分配的权重值  $W_2$  为 5。

④GSLB 始终是根据  $D_i$  权重值  $W_i$  的大小成比例地将新流量分配到相应的节点上。这里的  $D_i$  相当于图 5-11 中的差异量，而权重值相当于控制量。

⑤GSLB 不断地计算各 POP 节点的  $D_i$  值，从而不断地修改分配到每个节点的权重值，使得各节点的流量相对于总流量处于一个长期的平稳状态。即使突发流量或某节点失效引起了节点间的流量不均衡，各节点也会在很短的时间内收敛到一个平稳状态。

差异量  $D_i$  与控制量  $W_i$  的映射关系是一个复杂的计算过程，各家的实现方式存在一定的差别，好的映射方式必然反映出极小的收敛时间，最终体现在各节点流量的长期稳定性上。比如我们可以设置映射关系之一是  $W_i = P * \sigma D * (1 - D_i)$ ，这里  $\sigma D$  表示所有节点  $D_i$  的标准差，反映出实际流量与期望流量的整体偏差程度， $P$  是一个调整系数，通过这个关系，可以看到如果实际流量相对于期望流量整体偏差较大，那么用户调整流量的权重值也会较大。另外，其他不变的情况下， $D_i$  越小， $W_i$  越大。

## (2) 算法 2：同时基于往返时间和流量的负载均衡算法

在这个算法中需要考虑的负载均衡策略有两个：一个是用户访问 POP 节点的往返时间，另一个是流量负载，这两个均衡策略同时使用来保证 GSLB 将用

户访问调度到合适节点。

用户访问 POP 节点的往返时间可以通过动态策略中提到的主动测量方式或被动测量的方式计算出来，节点的流量负载情况可以通过 GSLB 定期询问节点来获得。

具体的算法描述如下：

① 用户通过主动测量或者被动测量的方式定期获取每个 IP 子网到所有为其提供服务的节点的往返时延， $ds,i$  表示子网 s 到节点 i 的往返时延。同时，用户也会定期获取每个节点的  $D_i$ （如算法 1 中定义）。

② 当子网 s 的用户发起请求，GSLB 比较子网 s 到所有可供服务节点的  $ds,i$ 。

③ 首先剔除  $ds,i$  超过阈值的节点，剩下来的节点都可为子网 s 的用户提供服务。

④ 用户选择  $ds,i$  最小的节点作为提供服务的默认节点，剩下的节点作为备选节点。

⑤ 子网 s 的用户发出请求，GSLB 先检查默认节点。如果默认节点的当前流量已经达到最大服务带宽  $B_i$ （见算法 1），则直接将请求平均分配到未达到  $B_i$  的备用节点；如果默认节点的当前流量未达到  $B_i$ ，考虑下面的情况：

a) 如果默认节点的  $D_i$  小于某个阈值，比如  $D_i < -0.2$ ，则将请求全部调度到默认节点上。

b) 如果默认节点的  $D_i$  在某个阈值期间，比如  $-0.2 < D_i < 0.2$ ，则将请求按照一定的概率（比如 80%）调度到默认节点上，没有调度到默认节点上的请求平均分配到其他备选节点上。

c) 如果默认节点的  $D_i$  大于某个阈值，比如  $D_i > 0.2$ ，则将请求按照一个更小的概率（比如 50%）调度到默认节点上，没有调度到默认节点上的请求平

均分配到其他备选节点上。

⑥GSLB 不断地计算各个节点的  $D_i$  以及各个子网到它的  $ds_i$ , 从而动态地修改调度策略, 使得 CDN 在满足用户访问不受影响的基础上尽量实现各 POP 节点的流量均衡。

值得注意的是, 在上述步骤中, 阈值区间的数值以及相应的概率可以由 CDN 运营商根据实际网络的部署和用户需求灵活设置, 也可以通过一种算法来进行映射。

### 5.3.3 GSLB 部署中的关键问题

这部分主要讨论 GSLB 在 CDN 中如何与 CDN 的其他功能模块协同工作以及如何解决实际中遇到的问题。目前, CDN 在技术架构上还没有正式的国际标准和国内标准, 所以各家的实现机制是不一样的。比如专门对外提供服务的 CDN 运营商可能采用一种高度可管可控的架构来实现, 而网络 SP 自建的 CDN 只专注于自己的业务发展需要。以下介绍中涉及的 CDN 架构和相关组件模块是我们经过多年研究, 在参考业界主流运营商和厂家技术的基础上提出的, 并已在实际网络中实施部署, 这里仅供参考, 对于读者理解 GSLB 的工作机制能提供一定的帮助。

#### 1. 网站 SP 的接入

这里以基于 DNS 的 GSLB 部署方式为例, GSLB 为访问网站的用户进行域名解析, 将用户请求调度到一个合适的 CDN POP 节点或者下一级负载均衡器。网站 SP 如果要使用 CDN 为其提供加速服务, 首先需要在自己网站的权威 DNS 服务器上增加一个 CNAME 记录, 并指定该 CNAME 的权威 DNS 服务器为 GLSB。

GSLB 通过配置查看哪些 POP 节点可以为 SP 提供加速服务 (SP 租用了 CDN 的哪些 POP 节点), 然后在这些节点中通过负载均衡策略为访问的用户

分配对应的节点或虚拟服务器 IP 地址。这里面可能有三种情况：

(1) 如果 SP 要求 CDN 提供全网加速，GSLB 应识别所有用户所属的区域，将用户所属区域的节点都用来为其服务，所有用户在访问其网站时都能体验到 CDN 的加速服务。

(2) 如果 SP 只要求 CDN 提供部分区域加速（即只为一定区域范围内的用户提供加速服务），而用户来源由 SP 自己来控制，则 GSLB 识别此区域的用户，将此区域的用户调度到 SP 合同规定的 POP 节点进行服务。

(3) 如果 SP 除了要求 CDN 提供部分区域加速，也要求 GSLB 提供权威 DNS 的服务，则 GSLB 识别用户所属的区域，为一定区域内的用户提供加速服务，针对其他区域用户则提供该 SP 源域名的授权解析服务。

除了通过配置指定用于服务网站 SP 的 POP 节点以外，还需要在 CDN 内部 DNS 系统中添加 SP 源域名所对应的服务器地址，即源服务器地址，这样提供服务的 POP 节点的 Cache 设备才能够定期向 SP 源服务器更新需要加速的内容。

### 2. 用户访问调度

这里细致描述网站在使用 CDN 加速的情况下，一个 Internet 用户从点击 URL 到看到相应页面的全过程，这个过程也展示了 GSLB 的具体工作方式和与其他模块之间的交互。

流程中提到的 SLB（本地负载均衡）作用与四层交换设备类似，只不过工作机制不一样。四层交换设备相当于网关设备，所有针对 Cache 服务器的请求和响应都必须通过四层交换设备调度，进行负载均衡，所以四层交换设备和后挂的 Cache 服务器是组合成一个或多个虚拟服务器来对外服务的，对外不会出现 Cache 的 IP 地址。而这里提到的 SLB 设备对用户请求做应用层重定向，在进行本地负载均衡后，会给用户返回可供服务的 Cache 服务器的地址。

这里的 OCS 是 CDN 的内容中心，在有些 CDN 中（用于视频网站加速的情况较多），网站需要加速的内容全部先缓存在 OCS，然后再将一部分（通常是热门的内容）分发到各 POP 节点，所以 POP 节点在某些时候会出现本地不命中而需要回 OCS 取内容或者从其他 POP 节点取内容的情况。

下面是对这个流程的详细描述（假设用户访问的是视频类网站），如图 5-12 所示。

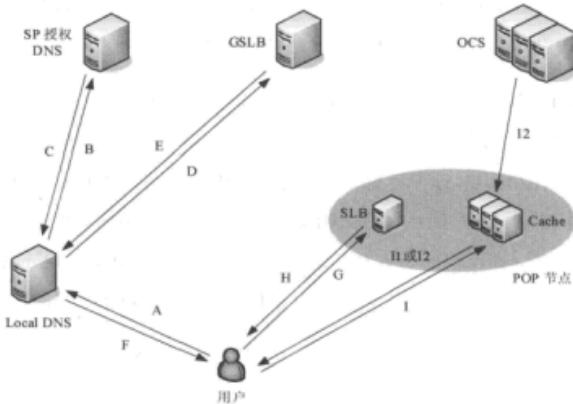


图 5-12 用户访问调度流程

- A. 用户访问网站的门户，点播某个影片，比如 <http://movie.netitv.com.cn/A/B/C.wmv>，用户请求本地 DNS 服务器解析域名 movie.netitv.com.cn。
- B. 本地 DNS 请求权威 DNS\_S (SP 的权威 DNS 服务器) 解析域名 movie.netitv.com.cn。
- C. 权威 DNS\_S 向本地 DNS 返回该域名的 CNAME (movie.netitv.cdn.cn)，以及负责解析该域名的权威 DNS\_C (即 GSLB IP 地址)。这里需要 SP 先配置其权威 DNS 服务器，将负责解析该 CNAME 的域名服务器指向 SLB。

- D. 本地 DNS 请求 GSLB 解析域名 movie.netitv.cdn.cn。
- E. GSLB 根据域名，判断是否存在资源限制，比如部分加速还是全网加速，然后根据本地 DNS 的 IP 地址判断用户就近性。综合判断后，选择最优的 SLB 的 IP 地址，向本地 DNS 返回这个域名解析结果。
- F. 本地 DNS 向用户返回域名解析结果。
- G. 用户根据域名解析结果，直接向 SLB 请求提供服务，URL 的形式保持不变，比如为 <http://movie.netitv.com.cn/A/B/C.wmv>。
- H. SLB 根据域名判断 Cache 服务器是否存在资源限制。在排除不能提供服务的 Cache 服务器后，SLB 综合考虑各 Cache 服务器的健康性、负载、连接数、Cache 服务器内容分布状况等，给出本 POP 节点最优的 Cache IP 地址，并且完成 PORTAL\_URL 到 CACHE\_URL 之间的映射，完成应用层重定向，比如将 <http://movie.netitv.com.cn/A/B/C.wmv> 映射为 [http://CACHE\\_IP/movie.netitv.cdn.cn/A/B/C.wmv](http://CACHE_IP/movie.netitv.cdn.cn/A/B/C.wmv)。SLB 向用户返回 HTTP 响应消息（状态码 302），并包含新的 CACHE\_URL 地址。
- I. 用户向 CACHE\_IP 地址所标识的 Cache 服务器请求提供服务。Cache 服务器完成 CACHE\_URL 到存储目录之间的映射，并判断内容在本地是否命中。
  - I1. 如果本地命中，Cache 直接为用户提供服务，流程结束。
  - I2. 如果未命中，Cache 则从 OCS 或其他 POP 节点来获取内容，参数为 PORTAL\_URL。

### 3. 异常流程

从技术实现上来看，纯粹基于 DNS 方式的 GSLB 只能完成就近性判断。为实现智能调度，大多数解决方案需要在 GSLB 设备附近以旁路的方式部署一台辅助设备（为方便描述，我们可称之为 GRM——全局资源管理设备），用以

实现和各 POP 节点的本地资源管理设备（我们称之为 LRM 设备）进行通信，完成 CDN 对各 POP 节点的状态检查，并根据 POP 节点的状态和流量情况，重新制订用户调度策略，将策略实时发送到 GSLB 中去执行。

GRM 每隔一段时间（时间长度可定制）向各 POP 节点的 LRM 设备发起数据通信接口，采集各节点的流量和在线并发连接数，根据 GRM 上已保存的各 POP 节点的业务能力情况进行判断，更新用户调度策略，下发到 GSLB 上。

由于 GSLB 的负载均衡策略需要根据 GRM、LRM 和 SLB 所获取的信息来确定，如果 SLB、GRM 和 LRM 在相互通信中出现错误，会有以下提到的异常情况发生。

（1）如果各 POP 节点的 LRM 设备发生故障，GRM 无法和 LRM 设备进行通信，那么 GRM 不再更新调度策略给 GSLB。在这种情况下，GSLB 必须支持默认模式，即在 GRM 反馈信息不够正确的情况下，能够启用一种比较通用的策略，来保证 GSLB 设备的负载均衡功能。

（2）如果各节点 SLB 设备发生故障，LRM 设备发出去警，并通知 GRM 设备，GRM 设备生成新的调度策略，将用户请求直接调度到该节点的 Cache 设备上（在调度到 Cache 设备上时，采用轮询或其他机制）。

#### 4. 网络攻击

因为 DNS 服务采用以 UDP 为基础的、默认无连接的访问方式，给分布式攻击带来了更大的便利。理论上，无论设备的实际服务能力有多大，都有可能被攻击而导致瘫痪，但是作为运营设备，GSLB 必须提供最大可能的业务保证，隐藏节点的存在在很大程度上可以避免这种情况的出现。

实际隐藏节点的实现方法就是在实际组网时除了部署正常工作的 GSLB 以外，再部署一台备份的 GSLB 设备，并将这一备份 GSLB 设备隐藏起来，不对外公布。这样一来，在工作的 GSLB 遭受 DNS 攻击而导致瘫痪的时候突然将

隐藏的 GSLB 作为服务设备公布，根据 DNS NS 轮询的工作原理，很快就会将原来隐藏的 GSLB 包含在服务列表中（具体时间根据 NS 纪录的 TTL 时间来决定）。这个被隐藏的 GSLB 的配置规则和工作 GSLB 配置相同，但是 TTL 时间不是工作 GSLB 的分钟级别，而是小时级别。即使攻击方发现新的服务节点而立刻进行攻击，因为各本地 DNS 已经得到正确解析，而且这个解析结果会缓存更长的时间，所以在很长一段时间内 CDN 都可以维持正常服务。

下面描述一下 GSLB 在受到网络攻击时的应对方式。由于负载均衡策略会消耗掉 GSLB 设备一定的资源，所以通常情况下 CDN 会使用两层 GSLB 来进行负载均衡，比如第一层部署一台 GSLB，第二层部署两台 GSLB 设备，然后再分别在第一层和第二层各部署一台隐藏的备份 GSLB 设备。

根据以上的假设条件，域名解析的正常情况是：本地 DNS 服务器在进行 DNS 请求的时候，首先将请求发送到 SP 的权威 DNS 服务器，权威 DNS 服务器通过 CNAME 的方式把本地 DNS 请求定向到第一层的 GSLB 进行解析；第一层的 GSLB 再通过轮询的方式把 DNS 请求定向到第二层的某台 GSLB 上，再由第二层的 GSLB 进行 A 记录的解析。

在第一层 GSLB 进行轮询并返回第二层 GSLB 地址时，本地 DNS 服务器保存这个地址的 TTL 可以设定为 30 分钟。第二层的 GSLB 在响应并返回 POP 节点地址时，本地 DNS 服务器保存这个地址的 TTL 可以设定为 5 分钟。这样，本地 DNS 服务器在用户第一次对域名进行解析请求时，这个请求需要先经过 SP 的权威 DNS 服务器，然后再经过第一层 GSLB 和第二层的 GSLB 来解析地址。如果在 5 分钟内，有其他用户使用相同的本地 DNS 解析相同的域名，就可以直接从本地 DNS 服务器的缓存中获取解析结果，而不需要再访问 GSLB。

如果处于正常工作状态的第一层 GSLB 设备受到网络攻击而不能工作，由于本地 DNS 服务器对第二层 GSLB 的缓存时间为 30 分钟，这样在 30 分钟内，用户通过这个本地 DNS 服务器解析相同的域名还可以访问到第二层的 GSLB。

如果之前没有用户通过这个本地 DNS 服务器访问该域名，本地 DNS 服务器没有缓存相应记录，也就无法同第一层的 GSLB 通信，也无法到达第二层 GSLB 并获得域名解析，此时需要把隐藏的备份 GSLB 设备公布出来，保证那些本地 DNS 服务器没有访问记录的区域的用户可以通过该设备访问整个系统。

如果是第二层的 GSLB 设备遭受到攻击，这种情况下，需要马上把第二层的隐藏 GSLB 设备公布出来，同时在第一层的 GSLB 设备上把该隐藏 GSLB 设备作为一条新的 NS 记录加入进来。

如果针对第一层或者第二层 GSLB 的网络攻击结束，应该再把备份 GSLB 设备重新隐藏起来。

## 5.4 基于应用层协议重定向的 GSLB

应用层重定向主要利用了 HTTP、MMS、RTSP 等协议本身的重定向机制来实现，由于各种应用层协议的重定向机制基本相同，因此我们就以 HTTP 协议为例，介绍相关的知识。

### 5.4.1 HTTP 重定向基本原理

在 HTTP 协议中，有三类重定向状态码：301 redirect、302 redirect 与 meta refresh。301 redirect 代表永久性转移（Permanently Moved），302 redirect 代表暂时性转移（Temporarily Moved），meta refresh 代表在特定时间后重定向到新的网页。

HTTP 状态代码是在服务器返回数据的第一行实现的，比如你访问 [www.g.cn](http://www.g.cn) 这个网址，Google 的服务器返回的数据第一行是：HTTP/1.1 301

Moved Permanently，页面自动跳转到 <http://www.google.cn>，表示 g.cn 这个 URL 被永久重定向到 <http://www.google.cn>。

三类重定向的作用都是将用户的资源请求转向到另外一个 URL，而这一节所说的 HTTP 重定向，是用于 CDN 均衡调度的，显然我们应该选择 302 redirect。因为负载均衡系统需要实现的是将用户立刻且暂时性地重定向到另一台服务设备上去。

比如浏览器请求 [www.CDNbook.com](http://www.CDNbook.com) 这个域名，服务器返回应答消息如下：

```
HTTP/1.1 302 Found
Date: Wed, 17 Mar 2010 08:11:11 GMT
Server: Apache/2.2.15 (Unix) mod_ssl/2.2.15 OpenSSL/0.9.7a DAV/2
PHP/5.2.9
X-Powered-By: PHP/5.2.9
Location: http://bj.CDNbook.com
Content-Length: 0
```

浏览器得到这个应答后，会再次发起请求，只不过请求的域名替换成 <http://bj.CDNbook.com>。有时候，域名替代也直接使用 IP 地址。

利用 HTTP 重定向的 GSLB 工作过程大致是：当用户在浏览器地址栏里输入一个域名，如 <HTTP://www.CDNbook.com> 时，本地 DNS 服务器返回的解析结果是 GSLB 的 IP 地址。用户端浏览器会向 GSLB 发起 HTTP 请求，GSLB 向浏览器返回响应请求的 HTTP 数据包，这个 HTTP 数据包的“头信息（Header）”中会包含一条“302 found”信息，告诉用户向某个 IP 地址的服务器请求内容。在这个过程中，GSLB 还会做一个动作，就是修改原来的 URL，把它改写成另一个 URL'，这个 URL' 中的域名就是用户要去进行内容请求的服务设备的主机名。用户向这个服务器请求内容得到应答后，才是真正开始浏览网页内容的过程。

图 5-13 所示的是截取的一段实际的网页浏览过程中的数据包。

## 第5章 全局负载均衡工作原理及实现



图 5-13 实际网页浏览过程抓包

可以看到，经过 DNS 解析后，用户得到 CDN GSLB 的 IP 地址：211.100.48.250，然后用户向这个 IP 地址请求内容，GSLB 返回给用户一个 HTTP 302 应答消息，消息中将用户请求的 URL 替换成含有 222.222.96.14 这个 IP 地址的新 URL，这个 IP 地址对应的设备就是用户需要再次发起请求的设备的 IP 地址。

与智能 DNS 方式相似，GSLB 在做 HTTP 重定向时，也就是在为用户选择最佳服务器时，也会综合考虑很多条件，比如服务器的负载情况、用户与各个服务器之间的链路质量、用户请求的内容所在位置等。区别在于，相比较于 DNS 方式，基于 HTTP 重定向方式的 GSLB 系统能够看到用户的 IP 地址，以及用户请求的具体内容，所以能够进行更精细的定位，这是 HTTP 重定向方式的最大优点。

当然 HTTP 重定向也有其不足。顾名思义，这种方法只适用于 HTTP 应用，不适用于任何其他应用。比如微软的 MMS 协议、RTSP 协议，就不能使用这种方式进行重定向。其次，由于 HTTP 重定向过程需要额外解析域名 URL，还需要与 URL 建立 TCP 连接并且发送 HTTP 请求，使得响应时间加长。第三，不同于 DNS 方式，没有任何用户请求能被外部系统终结，所有请求都必须进入 GSLB 系统，这将成为性能和可靠性的瓶颈。

### 5.4.2 基于 HTTP 重定向的 GSLB 工作流程

下面我们通过一个完整的流程来理解基于 HTTP 重定向的 GSLB 工作机制，示意图如图 5-14 所示。

(1) 用户首先向网站的本地 DNS 请求域名（[www.CDNbook.com](http://www.CDNbook.com)）解析。

(2) 由于网站已经预先进行了域名 CNAME 指向服务 CDN 的 GSLB 域名和 IP，所以本地 DNS 会向用户返回 GSLB 设备的 IP 地址。（注意，是一个具体的 IP 地址，这样 DNS 解析过程就终结了。当然，如果 GSLB 系统本身是有负载均衡的，那么返回的这个 IP 地址就是经过 GSLB 系统自身负载均衡后的某台具体设备的 IP 地址。）

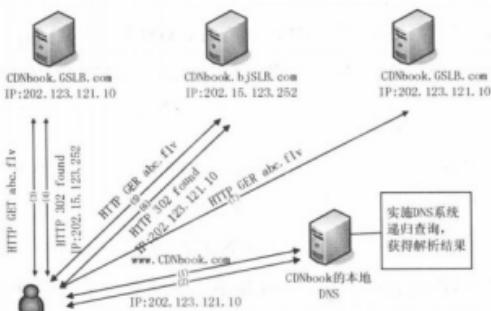


图 5-14 基于 HTTP 重定向的 GSLB 工作流程

(3) 用户向这台 GSLB 设备发起 HTTP GET 请求，请求该网站某个网页的内容。应该注意的是，在用户浏览一个完整的网页时，事实上是发起了多个 HTTP 请求，针对某个独立的对象，比如一张图片、一段视频、一段文字，都会存在这样一个独立的 HTTP 请求。

(4) GSLB 设备将综合分析用户 IP、内容分布、设备负载、链路状况等实时信息，为用户选择一个合适的服务单元。之所以称为“单元”而不是“设备”，

是因为很多实际的 CDN 系统的 GSLB 只负责负载均衡的第一步，将用户访问请求调度到一个合适的服务区，或者一个集群，再由区域均衡或本地均衡设备做下一步负载均衡工作。GSLB 设备向用户返回 HTTP 302 重定向应答，告知用户下一次请求应该发送的目的 IP 地址。如步骤（3）所述，这个 IP 地址可以是一台具体服务设备的 IP 地址，也可以是一个区域均衡设备的 IP 地址，或者一个集群的 L4 交换机的 IP 地址。

如果 GSLB 在自己的静态路径表中没有查到用户 IP 所在网段的信息，可以通过两种方式完成路由策略。方式一，GSLB 将用户请求通过轮询的方式定向到其他节点。方式二，GSLB 会以同样方式去查动态最近路径表，如仍没有记录，GSLB 会通知各 POP 点的 SLB 一同去测各 POP 点离用户的距离及时延，并报告核心节点确定最优站点，该 IP 地址所在的网段会被添加至动态最近路径表，供今后用户直接与最优的分配层节点的 SLB 设备建立连接。某些情况，也可采用轮询方式选择 SLB 设备。

（5）用户根据得到的 IP 地址向 CDN 节点发出媒体访问请求。

（6）如果这个 IP 地址的节点设备仍然是一个负载均衡设备，则通过负载均衡选择一台合适的服务设备，将其 IP 地址返回给用户。

（7）用户根据得到的 IP 地址向 CDN 服务设备发出媒体访问请求。

## 5.5 基于 IP 路由的 GSLB

基于 IP 路由的 GSLB 是基于路由器原有的路由算法和数据包转发能力工作的。直接讲解这种工作原理可能有些难以理解，下面我们通过一个具体应用场景来简单了解一下，如图 5-15 所示。

如图 5-15 所示，有两个本地均衡器 1 和 2，放在不同的 POP 点中，负责各自 POP 点内的服务器的负载均衡。先为这两个本地均衡器配置一个相同的 VIP 地址，对 IP 网上的路由器来说，这是到同一个 IP 地址的两条不同的路由。当终端 a 输入 URL 访问网站时，DNS 系统会把 VIP 作为域名解析结果反馈给终端。终端向这个 VIP 发送请求时，请求数据包经过路由器 A，路由器 A 查看路由表来决定通过哪条路径转发数据包至 VIP。它会发现有两条不同的路由，然后基于路由协议算法选择一条路由到达这两个本地均衡器中的一个。因为每次访问请求的终端 IP 地址不同，路由条件也不同，所以在多个路由器上优选的路由不同，从统计复用的角度来看基本是在负载均衡器 1 和 2 之间均匀分布的。这样，就通过路由器实现了两个 POP 点之间的负载均衡。

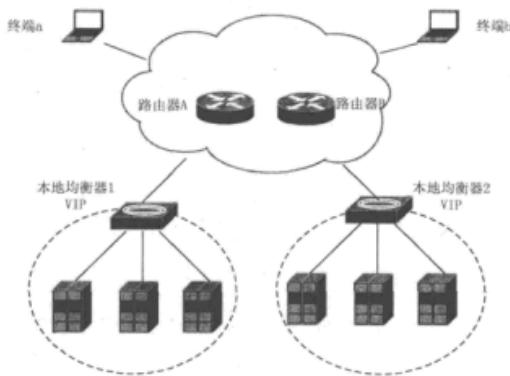


图 5-15 基于 IP 路由的 GSLB 实际场景

上面讲述的这种基于 IP 路由的 GSLB 方案主要被建议用于使用 IGP 协议的一个城域网或者自治域内部，很难实现基于 BGP 协议的跨自治域全国性负载均衡。因为在默认情况下，大多数 BGP 协议的路由器都不维护单台主机的路由，否则会导致路由表过于庞大。运营商为了控制路由条目的数量，也会丢弃主机路由。但是为了让基于 IP 路由的 GSLB 能正常工作，每台路由器都必须维护一条到 VIP 的主机路由。这样，就要求选用这种 GSLB 方案的 CDN 服务

商与运营商相协调路由器针对某个 VIP 的主机路由可用，这种协商很可能不被运营商接受，这也是本方案实施的局限性。

2006 年 Cisco 公司又提出一种 LISP 协议( Locator/ID Separation Protocol )，用于多数据中心选址。它的核心思想是对数据包进行 IP 嵌套，实现访问同一 IP 地址（内层 IP）的请求被路由到不同目的 IP（外层 IP）去。从图 5-16 所示的数据包结构可以看出这种 IP in IP 的工作方式。

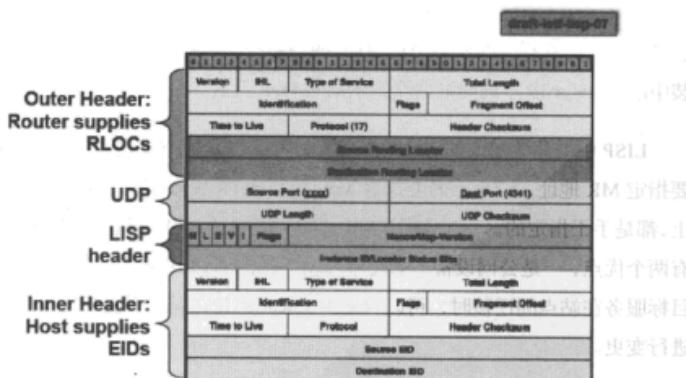


图 5-16 LISP 数据包

我们可以简单理解 LISP 的工作原理，如图 5-17 所示。

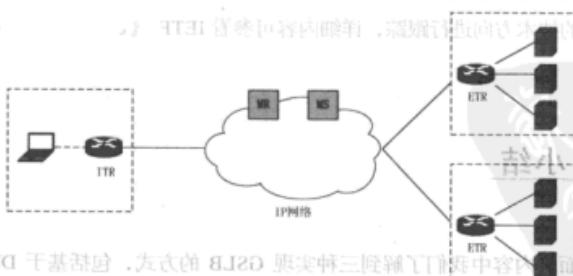


图 5-17 LISP 工作原理

为了便于理解，可以简单认为图 5-17 中的 ITR 和 ETR 是两台边缘设备，MR 与 MS 协同完成管理任务。

用户发往目的主机的数据报文第一次到达客户区域的 LISP 边缘设备 ITR 时，ITR 会根据报文的目的 IP 地址向 MR 请求查询对应的 ETR 的公网 IP ( Locator IP )，而这个 IP 地址的获得是通过 MR 与 MS、ETR 的通信获得的。获得 Locator IP 以后，ITR 会把它封装在数据报文的外层报头中，再将数据报文发到公网上。数据报文到达 ETR 时，ETR 会剥去外层报头，再根据内层报头的 IP 和端口信息进行转发。这样，即使多个用户访问的是同一个目的 IP，也可能因为外层封装中的 Locator IP 不同而被调度到不同的 POP 点去。

LISP 中各角色之间的关系大都是通过手工指定的方式建立的，如 ITR 上需要指定 MR 地址，ETR 上需要指定 MS 地址，ETR 发布哪些内部 IP 信息到 MS 上，都是手工指定的。只有 MR 和 MS 之间可以通过 BGP 来建立邻接关系。LISP 有两个优点，一是公网设备不需要学习站点内部明细 IP 路由项，二是当访问的目标服务在站点间迁移时，可以只变更 Locator IP，而不需要对真正的目的 IP 进行变更。

Cisco 的 LISP 目前也只处于试验阶段，距离能够推广商用应该还有几年时间，关于 LISP 是否适用于实现 GSLB，从原理上来看是讲得通的，但具体实现中会涉及很多需要解决的技术细节问题。本书不做过深分析了，可以把它当成一种可能的技术方向进行跟踪，详细内容可参看 IETF 《draft-ietf-lisp-07》。

## 5.6 小结

从前面的内容中我们了解到三种实现 GSLB 的方式，包括基于 DNS 解析的方式、基于应用层协议重定向的方式、基于 IP 路由的方式，其中 DNS 方式

的应用最为成熟和稳定，应用层重定向方式近年来在流媒体 CDN 中应用增多，IP 路由方式应用较少。任何一种方式都有它适合的场景和对象，在应用时应该充分考虑实际情况，扬长避短，才能获得最佳效果。下面我们分析一下三种方式的优缺点，帮助读者加深理解。

基于 DNS 解析方式下，GSLB 实现简单，内容请求路由简捷，负载均衡设备本身压力小，并且与分发内容类型及协议无关，是一种比较通用的实现方式。但应该看到，DNS 系统并不是天然做负载均衡的，所以它固有的工作机制会对 GSLB 系统带来一定局限性。

首先，DNS 误配置带来调度错误问题。用户与本地 DNS 服务器之间的距离直接影响这种方式负载均衡结果的准确性。某些情况下，用户与本地 DNS 服务器可能根本就不在同一个区域里，也就是说它们到同一个服务器的网络距离可能相差很大。因为 DNS 系统只能看到本地 DNS 服务器的 IP 地址，看不到真正用户的 IP，所以会把用户调度到不合适的设备上去。最常见的情形就是，一个用户手动配置了自己的本地 DNS 服务器 IP 地址，当这个用户移动到其他地方的时候，会继续使用这个本地 DNS 服务器。这样，GSLB 会依据本地 DNS 服务器的 IP 地址为用户提供一个离他很远的服务器，造成业务响应延迟大。

第二，DNS 缓存带来 GSLB 失效问题。我们已经知道，DNS 缓存机制会将一部分用户请求终结在本地，不会到达 GSLB。如果本地 DNS 服务器设置忽略 DNS 缓存的 TTL 值，就会无限期缓存 GSLB 的负载均衡结果，整个 GSLB 就等于失效了。同样的情况还会发生在浏览器缓存 DNS 记录时，除非关掉或重启浏览器，否则用户会一直按照缓存的记录访问同一台服务器。

第三，DNS 缓存无法应对服务器突发故障问题。在提供服务期间，服务器的健康性可能会发生变化，而此时 DNS 缓存还在起作用，用户会访问故障设备。这样我们只能祈祷 DNS 缓存尽快过期，才能启动新的 GSLB 负载均衡流程。

HTTP 重定向方式下，GSLB 调度精确性优于 DNS 方式，因为在这种方式下 GSLB 可以直接看到用户 IP、用户请求的具体内容以及 CDN 系统内部搜集的实时的全局信息。另外，由于 GSLB 的实现完全与公网无关，所以灵活性比较强，甚至可以根据客户的要求进行定制开发。HTTP 重定向方式的局限性体现在以下三个方面。

第一，GSLB 性能压力大。GSLB 作为所有用户请求进入 CDN 系统的第一跳，所有用户请求无一例外地都会到达这里，随着用户请求数量级的增加，GSLB 的压力也随之增大。同时 GSLB 作为全网的核心部件很容易受到攻击，有较大的安全隐患。

第二，协议扩展性较差。这一条从实现原理上也能理解，GSLB 是通过分析和改写服务器与客户端之间的应用协议交互信息实现重定向功能的，可能还要叠加一些权限判断、防盗链之类的定制功能。也就是说，应用层负载均衡需要对应用有一定的了解。所以对于新增加的应用协议，需要进行开发升级。而对那些不支持重定向的应用，还需要在 GSLB/SLB 上开发新的重定向接口。

第三，安全性问题。在这种方式下，每次重定向都会通过与客户端之间的应用协议交互完成，所以在客户端可以通过协议分析了解到整个服务器架构，这会导致一定的安全隐患。

基于 IP 路由的 GSLB 方式，好处在于使用者其实并没有购买或者开发一个 GSLB 设备，而是依靠公网上原有的路由设备来实现同样的功能，不需要对公网 IP 设备进行任何特殊配置，同时能够利用 IGP 的路由计算方式实现真正的就近服务和快速冗余备份。但如果仔细分析就会发现这种方式在多个 POP 点之间实现的负载均衡是一种概率上的均衡，而不是真正的均衡。如果路由器基于链路状态进行选路，那么上联端口大的那个负载均衡器被选到的概率就会大很多。如果某个热点事件是带有地域性的，用户都集中在同一个路由器上，那么这次热点事件的突发访问很可能就都被调度到同一个 POP 去。解决这种概率不均的方式是在部署 POP 点或 SLB 的时候就充分预测路由均衡性，尽量选择路由对

称的位置。

基于 IP 路由方式的 GSLB 另外一个突出的局限性在于无法实现会话保持。因为 IP 网是基于包交换的网络，也就是说，路由器有责任为每个数据包选择一条它认为最好的路径。但是在用户访问 Web 页面时，会产生多个 TCP 连接，这些连接被封装成多个数据包进行传送。路由器会把每个数据包都看成是独立的，并不关心它们属于哪个 TCP 连接。但是从应用层的要求考虑，应该把从一个终端发送的所有数据包传送到同一个本地均衡器上（同时本地均衡器应该保证同一个用户的同一个连接的数据包交付给同一台具体的服务器），这样才能保证用户业务使用的连续性。理想情况下，所有路径的路由开销保持不变，这样路由器就会为同一个连接的每个数据包选择相同路径。但现实中很少会出现这种情况。任何一个链路的故障，或者过度拥塞，都有可能导致路由开销的变化。最终结果就是，路由器在转发数据包时，可能突然从一条链路转移到另外一条链路上，这不仅会中断当前的连接，而且会中断会话保持，丢失所有的用户信息，比如购物车信息等，用户的体验会非常差。

通过前面的简单分析我们知道，每一种实现方式都有利弊，都有适合的场景和对象。在具体应用中应该不断在实现复杂度、均衡调度效果、成本之间取得平衡，系统优化是没有止境的。为了便于读者加深认识，我们用一张对比表格来结束本章的讨论，如表 5-1 所示。

表 5-1 GSLB 实现方式对比表

| 比较项 | 基于 DNS 解析方式                              | 基于 HTTP 重定向方式                   | 基于 IP 路由方式                 |
|-----|--|---------------------------------|----------------------------|
| 性能  | 本地 DNS 服务器和用户终端 DNS 缓存能力使 GSLB 的负载得到有效分担 | GSLB 处理压力大，容易成为系统性能瓶颈           | 借助 IP 网设备完成负载均衡，没有单点性能瓶颈问题 |
| 准确度 | 定位精确度取决于本地 DNS 覆盖范围，本地 DNS 设置错误会造成定位不准确  | 在对用户 IP 地址数据进行有效维护的前提下，定位准确且精度高 | 就近性调度准确，但对设备健康性等动态信息响应会有延迟 |

续表

| 比较项 | 基于 DNS 解析方式        | 基于 HTTP 重定向方式        | 基于 IP 路由方式      |
|-----|--------------------|----------------------|-----------------|
| 效率  | 效率约等于 DNS 系统本身处理效率 | 依靠服务器做处理，对硬件资源的要求高   | 效率约等于 IP 设备本身效率 |
| 扩展性 | 扩展性和通用性好           | 扩展性较差，需对各种应用协议进行定制开发 | 通用性好，但适用范围有限    |
| 商用性 | 在 Web 加速领域使用较多     | 国内流媒体 CDN 应用较多       | 尚无商用案例          |

# 第6章 流媒体 CDN 系统 的组成和关键技术

- 6.1 流媒体系统工作原理概述
- 6.2 流媒体传送协议体系
- 6.3 流媒体业务对 CDN 提出的要求和挑战
- 6.4 流媒体 CDN 系统的关键技术实现



## CDN 技术详解

在 Internet 诞生之后的很长一段时间里，网上的应用内容都是以下载方式提供的，内容以静态的文字、图片为主。相信很多读者都有关于 BBS、FTP 以及 Gopher 的深刻记忆，那是我们在校园里享受 Internet 娱乐的最初时期。但是网络内容从静态发展到动态，从文字发展到影音的趋势是必然的，就像从收音机发展到电影电视，技术从来不是业务发展的阻力。流媒体技术正是为了在网络中传播多媒体信息而产生的，用户不必再为下载观看一段视频文件等上几十分钟甚至几个小时，流媒体技术让人们在短短几秒钟内就能欣赏到精彩影像。

有人把流媒体等同于在线影音，这没什么问题。实际上，“流”是指内容在网上传送的状态，用户端得到的就是多媒体内容：电影、电视剧、赛事直播、音乐会、演出节目等。1995 年，Progressive Network 公司，也就是后来大名鼎鼎的 RealNetwork 公司，推出了第一个流式传送多媒体应用 RealAudio，实际上这只是一个在线音乐应用程序，还不是音视频混合的影音应用。同年，Xing 公司推出的 Streamworks 才是真正的流媒体影音播放应用程序。在流媒体技术应用的最初几年，受限于用户接入带宽（当然也有整个 IP 网带宽的因素），网上只能看到一两百 kb/s 速率的片子，清晰度非常低。近年来，随着运营商不断推进宽带提速，基本消除了历史上流媒体数据向用户端传送所遭遇的带宽瓶颈，因此多媒体业务作为一种高带宽应用得到前所未有的发展。从互联网业务流量占比上看，流媒体业务流量逐步取代 Web 内容成为网络主要流量，如图 6-1 所示。2009 年 Video to PC（面向 PC 的网络视频）流量占比已经位居第一，预计这一占比还将逐年攀升。

流媒体业务是一种对实时性、连续性、时序性要求非常高的业务，不论从带宽消耗上还是质量保障上来说，对 Best-effort 的 IP 网络都是一个不小的冲击。

高带宽要求。即使只为用户提供在普通笔记本上观看的视频节目，最低也需要几百 kb/s 的码率，才能够满足一定的清晰度要求。相应的，传送带宽要高于这个码率。这比起浏览普通网页只需几十 kb/s 带宽，是十几倍的差别。而用户对视频质量的要求不断提高，这种要求的提高是一个不可逆的过程，现在网

络视频已经转向电视这样的大屏幕，传送带宽要求高达几兆甚至几十兆，同时需要稳定的带宽保证。

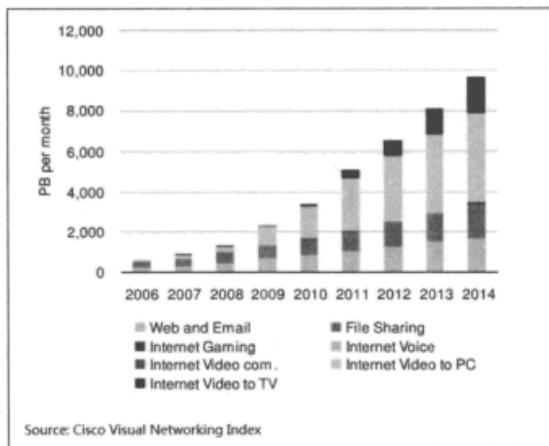


图 6-1 互联网各类应用流量发展

高 QOS 要求。比如码率为 750b/s 的 MMS 流媒体要求端到端丢包率小于 2%，双向时延小于 140ms。在 IP 网络上，并没有对视频流量实行端到端的 QOS 保障，所以要达到这样的 QOS 要求还是很难的。

流媒体业务还呈现较强的热点集中特性（虽然用户需求模型正在向着长尾方向发展），当热点事件到来时，同时在线用户数会瞬时飙升，对网络局部压力骤增，对源站服务器也会产生难以承受的压力。

组播、广播要求。直播类流媒体业务要求网络支持从单点（广播源）到多个接收点（用户终端）的流传输，以 IP 组播或广播技术来实现承载是最节省资源的方式，但目前的 IP 网络还难以提供端到端组播业务。

从上面的几点分析来看，有些需求是 IP 网络能够解决的，有些是短期内解决不了但可以规划解决的，而有些是怎么都解决不了的。Internet 原本是以简单、

高效的方式有序运行的，为了流媒体业务改变 Internet 的原始法则不实际也不合理。那么就会有新的技术出现以适应这种新的业务需求，这就是流媒体 CDN 技术。通过流媒体 CDN 来承载视频流量，能够实现以下优化目的。

CDN 服务节点部署在离用户比较近的网络边缘，对用户提供就近服务，能够缩短流媒体内容与用户之间的传送距离，解决长距离网络传输带来的质量下降问题，同时减少对骨干网段的带宽需求。

通过 CDN 边缘节点终结大部分用户访问，将用户大规模并发带来的流量激增压制在 IP 网络边缘，避免对骨干网产生冲击。

利用 CDN 节点服务器的流复制功能，实现应用层组播，实现流的树形分发，替代 IP 网组播。

流媒体技术并不是一项单一技术，而是融合很多技术之后产生的综合性技术，涉及流媒体数据采集、压缩、编码、存储、传送、播放、网络通信等方面。在这一章里，受篇幅所限，我们不能逐一深入讲解这些技术，而是挑选了一些流媒体 CDN 相关的关键技术，包括流媒体传送协议体系、流媒体 CDN Cache 设计要求、负载均衡设计要求、CDN 组网、流媒体内容预处理、防盗链等。当然，在此之前我们要简单理解一下流媒体服务的基本原理。对流媒体技术感兴趣的读者，建议翻阅一下流媒体领域的专著，对于理解流媒体 CDN 技术将有非常大的帮助。

## 6.1 流媒体系统工作原理概述

多媒体是把多种不同但相互关联的媒体，比如声音、视频、图形、动画等集成在一起而产生的一种存储、传播和表现信息的载体。有了多媒体技术，计算机就能够综合处理这些不同形式的信息，以丰富的展现形式极大改善人机交

互体验。而我们常说的流媒体，实际上是多媒体技术和网络技术相结合的产物，是经过压缩编码、流化处理，并通过网络传送给用户端播放的多媒体文件。当多媒体还不是流媒体的时候，体积一般比较大，需要用户下载到本地才能播放。相信读者都有在电脑上观看多媒体文件的体验，但是大家有没有仔细想过，多媒体文件在计算机上是怎样播放的呢？

下面介绍一下视频多媒体服务的基础知识。

几乎所有的播放器，如暴风影音、Media Player、RealPlayer 等，在播放视频的原理上是非常相似的，大致说来播放一个视频分为以下 4 个步骤。

1. Access，文件获取，指系统将多媒体文件从硬盘读入内存。
2. Demux，解复用，就是把通常合在一起的音频和视频分离（有的还有字幕文件）。
3. Decode，解码，将压缩编码后的音频和视频进行解码。
4. Output，输出，包括音频输出和视频输出。

拿播放一个 avi 视频文件来举例吧，系统首先将文件从硬盘读取到播放器的内存缓冲区中，然后播放器分析文件的封装格式，了解到这是一个 avi 封装的文件，按照文件尾部的索引部分正确地将音频和视频数据分离。之所以需要 Demux 这一步，是因为音视频在制作的时候实际上都是独立编码的，得到的是分开的数据。为了传输方便必须要用某种方式合起来，合起来的文件就是封装文件，相应的也就有了播放器解封装这一步。Demux 分解出来的音频和视频流分别送往音频解码器和视频解码器。因为原始的音视频都占用大量空间，而且是冗余度较高的数据，通常在制作的时候就会进行某种压缩。这就是我们熟知的音视频编码格式，包括 MPEG1 (VCD)、MPEG2 (DVD)、MPEG4、H.264、rmvb 等。音视频解码器的作用就是把这些压缩了的数据还原成原始的音视频数据。解码工作完成以后，就到输出步骤了。例如视频解码器输出的是一张一张

## CDN 技术详解

的类似位图格式的图像，但是要让人从屏幕上看到连续的视频画面，播放器会自动调用显卡加速功能，将画面呈现在显示屏上。

如果读者留心观察 Media Player 安装后在系统中创建的目录，就会发现不同的目录对应不同的任务，如/stream 目录对应的是 access 的功能，/mpdemux 对应的是 demux 功能，/libmpcodecs 对应的是解码功能，/libvo 和/libao2 分别是视频和音频的输出。

流媒体传送和播放的过程可以简单地理解成是把多媒体文件的存储和播放放在了网络的两侧，如图 6-2 所示。

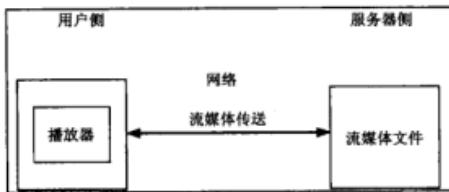


图 6-2 流媒体是多媒体文件存储和播放在网络两侧实现

正是因为文件存储和播放中间要经过网络传送过程，所以在流媒体技术领域，更多考虑的是高压缩比的编码、流的传输、播放控制、传输质量控制等。一套简单而完整的流媒体系统应该包括：①压缩编码工具，用于创建多媒体文件。②媒体库，用于存放多媒体文件。③流媒体服务器，对多媒体文件进行流化处理。④网络。⑤播放器。

流媒体的具体传送过程如图 6-3 所示。

1. 客户端 Web 浏览器首先向 Web 服务器请求一个展现描述文件，用来获取播放和控制所需的一些信息（关于展现描述文件，在本章的后续部分有更为详细的讲述）。

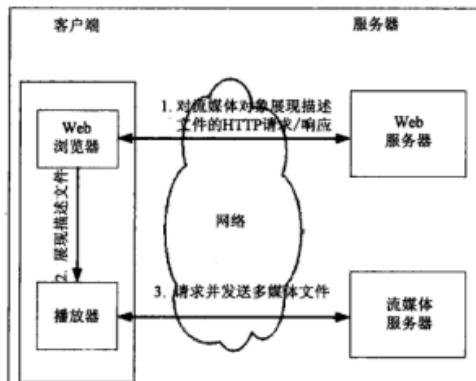


图 6-3 从流媒体传递到播放器的过程

2. 客户端 Web 浏览器将展现描述文件交付给本地播放器，播放器进行初始化。
3. 播放器向流媒体服务器请求相应的流媒体数据，并在传送过程中实时交互控制信息。一旦流媒体数据抵达客户端，满足播放器缓冲要求后，就可以开始播放了。

## 6.2 流媒体传送协议体系

如果读者对 Internet 传送技术有所了解就会知道，Internet 并不是为传送多媒体内容而设计的，最初它只用于传送纯文本性的资料，经过一段时间之后才加入了图像、动画等形式。而到现在，Internet 中大部分内容都是多媒体内容，Internet 本身对于传送这类内容是存在一定困难的，主要集中在以下三个方面。

第一，与纯文本性的数据相比，多媒体数据需要占用更多的网络带宽，这

种需求增加不是几倍、几十倍的关系，而是上千倍。

第二，多媒体需要实时的、稳定的网络传输，只有网络的带宽和时延抖动维持在一定水平，才能保证流媒体在用户端平稳播放。如果数据不能按时到达目的地，播放就会停止或中断。进一步，如果出现数据传输延时后，不能合理建立丢弃、恢复机制，那么网络拥塞就会更加严重。

第三，多媒体数据流突发性很强，单纯增加带宽往往不能解决这种突发问题。因此多数多媒体应用程序都有接收端的缓存机制，这需要合理地调节数据流的平稳度，避免造成程序的缓存上下溢出。

显然，寄希望于 Internet 自身来解决上述三方面困难是不现实的，Internet 倾向于保持其技术简单性和 Best-effort 理念。因此，设计流媒体传送协议在新的 Internet 时代势在必行。

在这一节中将向读者介绍几个流媒体传送协议，包括：实时流传输协议 RTSP、实时传输协议 RTP、实时传输控制协议 RTCP、实时消息传递协议 RTMP、HTTP 流化协议，这些是近年来比较常用的流媒体传送协议。从 TCP/IP 协议栈来看，这些协议都位于 TCP/UDP 的上一层，如图 6-4 所示。

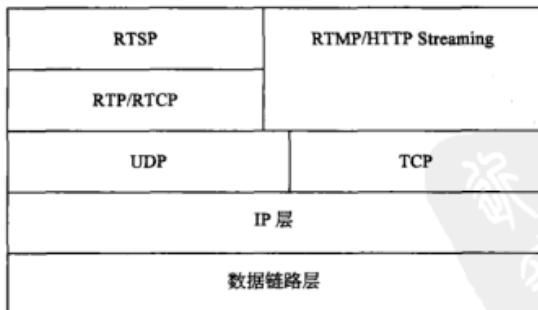


图 6-4 流媒体传送协议在 TCP/IP 协议栈中的位置

如果用一句话来概括这几个协议的作用，那么就是 RTSP 协议用来实现远

程播放控制，RTP 用来提供时间信息和实现流同步，RTCP 协助 RTP 完成传输质量控制，RTMP 和 HTTP Streaming 则是将流同步、播放控制、质量控制集成起来的企业自有流媒体传送协议。

### 6.2.1 RTP 和 RTCP

前面我们了解到，流媒体业务是一项对实时性、顺序性要求非常高的业务。但是数据包在传送过程中难免会出现丢包、时延抖动等问题，所以流媒体的发送方在将数据包送入传输层之前会加上首部字段，字段里包含了序号和时间戳。RTP（Realtime Transport Protocol）实时传输协议，就是这样一个定义包含音视频数据、序号、时间戳以及其他有用信息的标准分组结构的协议。简单地说，RTP 的任务就是提供时间信息和实现流同步。

RTCP 可以说是 RTP 的伙伴协议，因为 RTP 只负责流媒体数据包的交付，而不负责按顺序保证质量的交付，后一项任务往往要依靠 RTCP 来完成。RTCP（Realtime Transport Control Protocol）实时传输控制协议负责在会话参与者之间交换控制信息，RTCP 分组包中含有已发送的数据包的数量、丢失的数据包的数量等统计数据，服务器可以利用这些信息动态地改变传输速率，甚至改变有效载荷类型。RTP 与 RTCP 配合使用，能以有效的反馈和较小开销使传输效率最优化。在 UDP 中，RTP 如果使用一个偶数号端口，则相应的 RTCP 使用其后的奇数号端口。

1991 年 8 月，Lawrence Berkeley 国际实验室的网络研究工作组在 DARTnet 上发布了一个称之为“vat”的语音会议工具，所使用的协议就是 RTP 协议的前身。到了 1992 年，RTP 协议有了 1.0 版本，1995 年 11 月被 IESG 正式定为 Internet 标准。1996 年 1 月，Netscape 公司宣布其“Netscape Live Media”应用建立在 RTP 协议之上。微软也不甘落后，也宣布其 Netmeeting 软件支持 RTP 协议。

### 1. RTP 基础

RTP 通常运行在 UDP 之上。发送方在媒体数据块上加上 RTP 首部，形成 RTP 分组，RTP 分组又被封装在 UDP 报文中，UDP 报文被交付给 IP 层。接收方收到报文后，从中提取出 RTP 分组，然后从 RTP 分组中提取出媒体块，再将媒体块传递给媒体播放器进行解码和显示。RTP 数据包的位置和结构可以通过图 6-5 来理解。



图 6-5 RTP 数据包构成

RTP 数据包没有长度限制，它的最大包长只受下层协议的限制。从 RTP 数据包的组成来看，RTP 协议贡献的是 RTP 首部，这个首部包括媒体数据编码的类型、序号和时间戳。接收方依靠 RTP 首部字段信息来正确解码和播放后面的有效载荷。如果一个应用程序使用 RTP，而不是某种私有协议来提供负载类型、序号或者时间戳等信息，就很容易与其他使用 RTP 协议的应用程序进行互通。

RTP 允许为每个源（比如影片播放的音频流、视频流）分配一个独立的 RTP 分组流，但很多流行的编码技术在编码过程中，倾向于将音频和视频流捆绑在单个流中。这样，这一组音频流和视频流就只产生一个 RTP 流。

### 2. RTP 分组头部字段

RTP 首部字段包括有效载荷类型、序号、时间戳和源标识符，如图 6-6 所示。

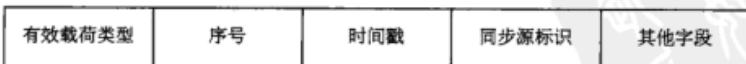


图 6-6 RTP 首部字段

其中有效载荷字段长度为 7bit，指示音频或视频流的编码类型，比如音频 PCM、自适应增量调制等，视频 MPEG1、MPEG2 等。如果发送方在会话过程

中决定改变编码方式，可以通过有效载荷类型字段把这种变化通知给接收方。表 6-1 给出了 RTP 支持的一些音视频有效载荷类型。

表 6-1 RTP 支持的一些有效载荷类型

| 有效载荷类型编号 | 音视频格式    |
|----------|----------|
| 0        | PCM μ律   |
| 1        | 1016     |
| 3        | GSM      |
| 14       | MPEG 音频  |
| 26       | 运动 JPEG  |
| 31       | H.261    |
| 32       | MPEG1 视频 |
| 33       | MPEG2 视频 |

序号字段长度为 16bit，是指 RTP 分组的编号。每发送一个 RTP 分组则该序号增加 1，而且接收方可以利用这个字段来检测丢包和恢复分组序列。比如接收方收到了序号为 100 和 103 的 RTP 分组，而没有收到序号为 101 和 102 的，就知道这两个分组在传输中丢失了，会设法隐藏或恢复这两个丢失的分组，保证播放损失最小。

时间戳字段长度为 32bit，它反映了 RTP 分组中的第一个字节的采样时刻。这个时刻是该分组的时间基准点，接收方能够通过它来去除传输过程中产生的时延抖动，从而保证播放同步。

同步源标识符（SSRC）字段长度为 32bit，用来标识 RTP 流的源。这个标识符不是 IP 地址或者 CNAME，而是当流开始的时候随机分配的一个数。通常，一个 RTP 会话中的每个流都有一个不同的 SSRC，而接收端可以通过这个标识符来鉴定数据是来自哪个源。

### 3. RTCP 基础

使用 RTCP 协议后，会话的参与方都会互相发送报告，从而得到数据传输

质量的反馈以及对方的状态信息。RTCP 数据包是一个控制包，由一个固定的报头和结构元素组成。RTCP 数据包的发送方一般会把多个包含成一个在底层协议中传输。

在 RFC 1889 文档中定义了 5 种传送控制信息的 RTCP 数据包，分别如下。

**SR： Sender Report**，发送端报告。SR 由发送方产生，包含发送端信息部分、媒体间的同步信息、数据包累计计算、发送的字节数等。

**RR： Receiver Report**，接收端报告。由接收方产生，主要是数据传输的质量反馈，包括接收的最大数据包数、丢失的数据包数、计算发送端和接收端的往返延迟等。

**SDES： Source Description**，源描述，包含对源的一些描述信息。

**BYE**：表示结束参与。

**APP**：应用特殊功能。

上面 5 种传送控制信息中，发送端报告和接收端报告对质量控制是最重要的信息，我们来进一步理解一下这两类消息。

发送端报告用于同步一次 RTP 会话中的不同流，信息包括：

- RTP 流的 SSRC。
- 时间戳和流中最近产生的 RTP 分组的墙上时钟时间( wall clock time )。
- 流中发送的分组数。
- 流中发送的字节数。

接收端报告包括几个字段，其中最重要的是：

- RTP 流的 SSRC，标识报告所对应的 RTP 流。

- 在 RTP 流中丢失的分组。接收方计算丢失的 RTP 分组数除以流中的 RTP 分组总数，如果发送方发现这个字段显示接收方收到的分组数过小，就会降低编码速率。
- RTP 流中收到的最后一个序号。
- 到达时延抖动，它是 RTP 流中连续分组之间的到达时间间隔变化的平滑估计。

对于 RTCP 协议的使用，应该注意的是对通信量的控制，否则会造成带宽浪费。通常建议将 RTCP 流量控制在会话带宽的 5% 以内。比如发送方对媒体流的发送速率为 4Mb/s，那么 RTCP 流量控制在 200kb/s 以内是比较合适的。

关于 RTP 和 RTCP 协议更为详细的描述，读者可以查阅 RFC 3550。

## 6.2.2 RTSP

### 1. RTSP 基础

RTSP(Real Time Streaming Protocol)，实时流传输协议，是由 RealNetworks 和 Netscape 提交的 IETF RFC 标准。RTSP 是一个应用层协议，属于 TCP/IP 协议体系，位于 RTP 和 RTCP 之上。

读者首先应该明确一个概念：RTSP 是一个用来实现播放控制的协议，而不是一个流媒体压缩协议或者传送协议。意思是，这个协议可以使用户像使用电视遥控器一样控制互联网中流媒体的播放、暂停/继续、快进、快退。但它不负责定义音视频压缩方案、分组封装方式，也不限制播放器如何缓冲。这些分别要由编码协议和实时交互协议来做。

下面会讲述 RTSP 的简单工作原理，在此之前，需要了解几个经常出现的术语，这有助于读者理解后面的内容。

- 表示（presentation）：作为一个完整的媒体信息，传送给客户端的一个或多个流的集合。
- 媒体流：单个媒体实例，比如，一个音频流或者一个视频流。当使用 RTP 时，流包括由 RTP 会话（session）中同一个源所创建的所有 RTP 和 RTCP 包。
- 会话（session）：指一次 RTSP “事务”（transaction）的全过程。比如，一个电影的观看过程。会话从客户端为连续媒体建立传输机制（SETUP）开始，到用停止（TEARDOWN）关闭流结束。这期间可能使用播放（PLAY）或录制（RECORD）进行流的传送。
- 连接：是以通信为目的，在传输层建立的两个程序间的虚拟信道。

RTSP 是一个 C/S 方式的带外协议（out-of-band protocol），RTSP 报文和媒体流使用不同的端口号，通常 RTSP 使用端口号 544。我们通过一个典型的例子来理解 RTSP 这种带外控制的工作方式，如图 6-7 所示。

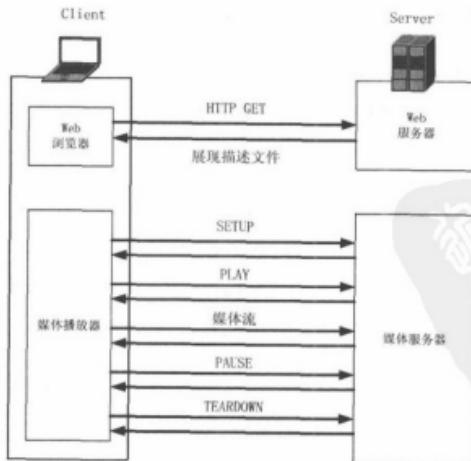


图 6-7 Client 与 Server 使用 RTSP 交互

客户端首先向服务器请求一个展现描述文件，这个请求是 Web 浏览器发起的，对端 Web 服务器返回的展现描述文件中包含一些指向几个媒体文件的引用，同时包含这几个媒体文件的同步指示。每个引用以 URL 的方法 rtsp://开始。描述文件就像下面这样：

```
<title>Shower</title>
<session>
<group language=en lipsync>
<switch>
<track type=audio
      e="PCMU/8000/1"
      src="rtsp://audio.example.com/shower/audio.en/lofi">
<track type=audio
      e="DVI4/16000/2" pt="90 DVI4/8000/1"
      src="rtsp://audio.example.com/shower/audio.en/hifi">
</switch>
<track type="video/jpeg"
      src="rtsp://video.example.com/twister/video">
</group>
</session>
```

Web 服务器在 HTTP 响应报文中封装这个展现描述文件，浏览器收到这个报文后，会根据展现描述文件中给出的信息来调用本地媒体播放器。比如上面例子中的展现，是由两个音频流，一个视频流组成的，并且它们之间有同步关系。在 RTSP 协议中，有“合控制”这样一个概念，对构成同一个流媒体展现中的不同流进行控制。服务器使用一条时间轴对这些不同的流进行控制，意味着客户端仅需发送一条播放或者暂停消息就可同时控制音频和视频流。大部分情况下，RTSP 协议中的控制都是指对一个表示中的流的合控制，但如果需要对单个流进行控制，可以在请求消息中说明。

播放器向媒体服务器发送一个 RTSP SETUP 请求，服务器用一个 RTSP OK 报文来响应，这就建立起了一个会话。随后，播放器发送一个 RTSP PLAY 请求，请求中携带需要播放的媒体流的 URL，媒体服务器用一个 RTSP OK 报文

响应，然后将播放器请求的媒体流送入它的带内信道。稍后，如果播放器发送一个 RTSP PAUSE 请求，表示需要暂停播放，媒体服务器用 RTSP OK 响应。当用户想结束播放时，播放器发送一个 RTSP TEARDOWN 请求，媒体服务器用 RTSP OK 响应。

RTSP 的这种交互方式与 HTTP 1.1 类似，而差别在于 HTTP 是不对称协议，客户端发出请求，服务器做出响应。RTSP 是对称协议，客户端和服务器都可发出请求。

RTSP 可以采用 TCP 传输，也可以采用 UDP 传输。这取决于服务器端对于客户端之间的网络情况的判断，如果服务器认为网络条件良好，响应时间正常，且丢包率在可承受范围之内，就可以采用 RTP over TCP 的方式发送数据。由于 TCP 不会丢包（其自身具有重传机制）；网络条件又好，因此客户端将获得较高的视听享受。如果服务器认为网络情况不理想，就会采用 UDP 协议，这可能会产生丢包，造成客户端播放过程中有马赛克等轻微体验劣化。但从整体而言，UDP 传输消耗的带宽要比 TCP 小许多，所以一般情况下使用 RTP over UDP 进行传送是完全可以满足要求的。

## 2. RTSP 方法

前面我们了解到，RTSP 控制消息与实际的数据流可能使用不同的独立协议进行传送，比如 RTSP 控制使用 TCP 连接，而数据流使用 UDP 连接。也就是说，服务器向客户端传送数据的时候，并不知道双方请求-应答交互的如何，即使媒体服务器没有收到请求，数据也会继续发送。在会话生命期内，同一个媒体流可以通过不同 TCP 连接上的 RTSP 请求来控制。所以，服务器需要维护“会话状态”以便使 RTSP 请求和流相互关联。

RTSP 对每一次资源访问建立一个会话（session），从客户端第一次请求访问对象开始，到媒体播放器送出一个“TEARDOWN”信息结束。表 6-2 中列举了一些与状态相关的重要方法（RTSP 中很多方法与状态无关）：SETUP、

PLAY, RECORD、PAUSE 和 TEARDOWN。

表 6-2 RTSP 中与状态相关的方法

| 方法       | 方向            | 要求 | 含义   |
|----------|---------------|----|--|
| DESCRIBE | Client-Server | 推荐 | 获得媒体对象的表示描述。DESCRIBE 请求中包含 RTSP URL、应答类型。对 DESCRIBE 请求的应答应描述当前情况   |
| OPTIONS  | Client-Server | 必选 | 获得可用的方法  |
|          | Server-Client | 可选 |  |
| SETUP    | Client-Server | 必选 | 服务器为流媒体分配资源，开始 RTSP 会话。SETUP 请求包含 RTSP URL、RTP 数据接收端口号、RTCP 数据接收端口号等。服务器对 SETUP 请求的应答通常是对请求中携带的参数的确认，以及服务器侧的相关参数 |
| PLAY     | Client-Server | 必选 | 服务器开始利用 SETUP 分配的资源传送数据。PLAY 请求可用于一个单一 URL，也可以用于多个 URL。PLAY 支持 range 参数，也就是说，可以从媒体流的中间某处开始传送数据（以便支持暂停、拖动操作）      |
| PAUSE    | Client-Server | 必选 | 暂停流传送，不释放相关资源，可以用 PLAY 请求继续传送  |
| TEARDOWN | Client-Server | 必选 | 停止传送指定的流媒体，释放相关资源  |

### 3. RTSP 消息

RTSP 消息由客户端到服务器的请求 (Request) 和由服务器到客户端的响应 (Response) 组成。请求和响应消息都使用 RFC 822 中实体传输部分规定 (作为消息中的有效载荷) 的消息格式。两者的消息都可能包括一行起始行、一个或多个头部域 (headers)、一行表示头部域结束的空行 (即 CRLF 前没有内容的行) 和一个消息主体 (message-body; 可选)。

## CDN 技术详解

下面我们看一个实际的 RTSP 会话交互：

```
Client:SETUP rtsp://audio.example.com/shower/audio.en/lofi
RTSP/1.0
    Cseq:1
        Transport:rtp/udp;compression;port=3056;mode=PLAY
    Server:RTSP/1.0 200 OK
    Cseq:1
    Session:4231
    Client: PLAY rtsp://audio.example.com/shower/audio.en/lofi
RTSP/1.0
    Range:npt=0
    Cseq:2
    Session:4231
    Server:RTSP/1.0 200 OK
    Cseq:2
    Session:4231
    Client: TEARDOWN rtsp:// audio.example.com/shower/audio.en/lofi
RTSP/1.0
    Cseq:3
    Session:4231
    Server:RTSP/1.0 200 OK
    Cseq:3
    Session:4231
```

在这个例子中，Client 端通过 SETUP 请求发起这个会话，提供需要的文件 URL 为 `rtsp://audio.example.com/shower/audio.en/lofi`，RTSP 版本 1.0。还指示应该在 UDP 上使用 RTP 协议进行传送，使用端口号为 3056。Server 端收到这个请求后，用一个 RTSP 应答回复。每个请求-应答组都有一个 Cseq 编号。随后 Client 端又发起 PLAY 请求，Server 应答后开始发送媒体流，最后以 Client 端的 TEARDOWN 请求结束这次会话。

RTSP 协议实际上能做的工作比这要多得多，关于 RTSP 协议更为详细的描述，读者可以查阅 RFC 2326。

### 6.2.3 RTMP

Flash 是当前 Web 应用的一种主流技术，现在较多的视频网站都采用了 Flash 播放器作为用户的视频播放客户端，而 Flash 客户端与 Flash 流媒体服务器之间的通信则只能采用 RTMP 协议来实现，视频网站如要使用这些技术和协议为用户提供服务，还需要付费给 Flash 技术的所有者 Adobe Systems 公司。

RTMP 协议全称为 Real Time Messaging Protocol，即实时消息传送协议，是 Adobe Systems 公司为 Flash 播放器和流媒体服务器之间传输音频、视频和数据所开发的私有协议。为了推广基于 Flash 平台的视频应用，给视频应用开发者和使用者带来更好的产品体验，Adobe 公司于 2009 年开放了 RTMP 协议，相关协议文档说明也可以在 Adobe 的官方网站下载。这里将对 RTMP 协议做一个详细的描述。

#### 1. RTMP 传输机制

在理解 RTMP 协议和具体流程之前，我们来看看 RTMP 协议中定义的基本通信单元：消息块（Chunk）和消息（Message）。RTMP 消息是协议中实现各种流媒体控制和应用的基本逻辑信息单元，消息从种类上可以分为协议控制消息、用于发送音频数据的音频消息、用于发送视频数据的视频消息、发送用户数据的数据消息、共享对象消息以及命令消息，属于相同逻辑通道的消息组成为一个消息流，这个逻辑通道通过消息格式中的“消息流 ID”字段来标识。

作为应用层协议，RTMP 协议架构在 TCP 层之上，但 RTMP 消息并不是直接封装在 TCP 中，而是通过一个被称为消息块的封装单元进行传输。消息在网络上发送之前往往需要分割成多个较小的部分，这些较小的部分就是消息块，属于不同消息流的消息块可以在网络上交叉发送。这样做可以保证各个消息流中的高优先级消息块能够严格按照时间顺序达到通信的对端。比如某个较长消息的实时性要求较低，如果不进行消息块处理，等长消息都发送完毕后再发送实时性要求高的短消息，则会对流媒体的播放质量造成影响。

属于相同逻辑通道的消息块组成一个消息块流，这个逻辑通道通过消息块格式中的“消息块流 ID”字段来标识。消息发送方可以将相同类型的多个消息流复用成一个消息块流，接收方根据消息块中的消息流 ID 解复用出各个消息流，并最终将消息块还原成消息。

RTMP 消息块流在 RTMP 协议中相对比较独立，它本身只负责数据的传输，而数据传输以外的控制功能则由更上层的 RTMP 协议来完成。

与很多通信协议类似，RTMP 消息在格式上也是包含包头和负荷两部分，包头包含了时间戳、消息长度、消息类型以及消息流 ID，而消息负荷则是消息中的实际数据，比如压缩的视频数据。图 6-8 是消息头的基本格式。

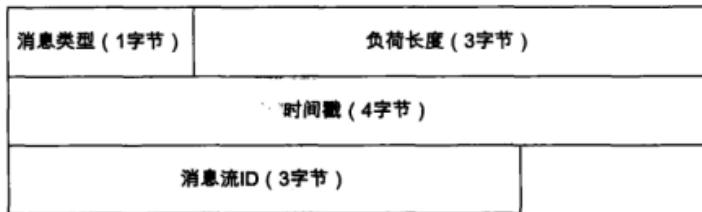


图 6-8 RTMP 消息头格式

消息包头的第一个字节是消息的类型，紧接着后面的 3 个字节是整个消息负荷的长度，后面接着 4 个字节表示具体的时间戳，最后的 3 个字节是消息流 ID。

消息块的大小是可以设定和更改的，RTMP 一般通过协议控制消息（后面会介绍到相关消息）来实现，可以设置的大小范围在 128 字节到 65536 字节之间，默认情况下是 128 字节。消息块的格式如图 6-9 所示。

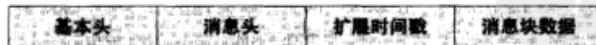


图 6-9 消息块的格式

第一部分是 1 到 3 个字节的消息块基本头，包括了消息块流 ID 和消息块类型，消息块类型决定了第二部分的编码格式。第二部分是消息块的消息头，这部分的编码方式在第一部分的消息块类型中规定。第三部分是 0 字节或者 4 字节的扩展时间戳，它对应于第二部分中所包含的普通时间戳。最后一部分是消息块的负荷，比如压缩的视频数据。

第一部分中的消息块类型指示了第二部分的编码格式，RTMP 中定义了 4 种编码格式，这 4 种编码格式的长度分别为 11 字节、7 字节、3 字节和 0 字节。采用 4 种编码格式是为了压缩传输的字节数，使得一个消息块流中的多个同类型消息块可以共用某些头部字节，因此部分消息块采用较多字节的格式，部分消息块采用较少字节的格式。11 字节的编码格式如图 6-10 所示。

| 时间戳             |        | 消息长度  |
|-----------------|--------|-------|
| 消息长度 ( cont )   | 消息类型ID | 消息流ID |
| 消息类型ID ( cont ) |        |       |

图 6-10 消息块的消息头（11 字节）

第一部分的时间戳就是前面提到的普通时间戳，如果时间值大于或等于 0xffffffff，这个字段应设置为 0xffffffff，扩展时间戳必须被发送。第二部分的消息长度不是指消息块的负荷的长度，而是未分割的消息的负荷长度，比如一个消息的负荷就是一帧视频数据。第三部分和第四部分分别是消息类型 ID 和消息流 ID。

长度为 7 字节的编码格式与 11 字节的相比，不包含消息流 ID，说明这个消息块的消息流 ID 与上一个消息块的消息流 ID 一致。长度为 3 字节的编码格式与 11 字节的相比，只保留了时间戳部分，使用这个格式的消息头说明消息块的消息流 ID 和消息长度都与上一个一致，消息长度为固定值的数据（比如音频流数据）可以使用这种编码格式。0 字节的编码格式表示消息块的消息头与

前一个完全一致，当只有单个消息发送时，被分割后的消息块（第一个消息块除外）可以使用这种类型的编码格式。

从消息块的格式看，消息块本身的长度并没有在消息块中被定义。RTMP 中消息块长度一般默认为 128 字节，如果要修改消息块的长度，需要客户端与服务器建立连接后通过发送信令来进行协商，理论上可以修改的最大长度为 65536 字节，这个长度值是指消息块负荷的长度，不包括前面的基本头、消息头和扩展时间戳三个部分。如果消息的长度超过了消息块的长度，消息需要被分割成几部分来装进消息块。比如消息的长度为 2000 字节，而消息块的长度为 1500 字节，所有消息需要被分成两部分，一部分 1500 字节，一部分 500 字节，最后变成两个消息块。

表 6-3 和表 6-4 是一个消息被分割成多个消息块的例子，这里消息块的长度为 128 字节。消息没有被分割前如表 6-3 所示，分割后的消息块如表 6-4 所示。

表 6-3 消息被分割前

|      | 消息流 ID | 消息类型 ID     | 时间   | 长度  |
|------|--------|-------------|------|-----|
| 消息 1 | 12346  | 9 ( video ) | 1000 | 307 |

表 6-4 消息被分割后

|       | 消息块流 ID | 消息块类型 | 头数据  | 头后字节数 | 总共字节数 |
|-------|---------|-------|--|-------|-------|
| 消息块 1 | 4       | 0     | Delta:1000<br>length:307<br>type:9,<br>stream ID: 12346<br>( 11 字节 ) | 128   | 140   |
| 消息块 2 | 4       | 3     | none ( 0 字节 )  | 128   | 129   |
| 消息块 3 | 4       | 3     | none ( 0 字节 )  | 51    | 52    |

## 2. RTMP 消息和关键流程

前一个小节讲述了 RTMP 消息和消息块之间的关系，了解了 RTMP 协议通

过消息块传输的基本传输机制，从中我们知道消息块本身只负责协议数据的传输，而 RTMP 协议中核心的上层控制功能需要各种类型的 RTMP 消息来完成，接下来我们分类介绍 RTMP 的各种消息以及与之相关的功能实现流程。

客户端与服务器通过交互消息来完成各种控制和管理功能，RTMP 的消息按照其实现的功能可以分为协议控制消息、命令消息、视频消息、音频消息、用户数据消息、共享对象消息。这些消息的具体类型通过消息格式中的“消息类型”字段来标识。比如类型 1~7 是 RTMP 规定预留给协议控制消息的类型指示 ID，RTMP 协议的使用者可以在开发中自定义其他类型的 ID 来实现其他控制功能。接下来我们主要介绍协议控制消息和命令消息的使用。

### 协议控制消息

协议控制消息主要包含了用于消息块传输和 RTMP 协议本身的信息，类型 1 和类型 2 用于消息块传输控制相关功能，类型 3 到 6 用于 RTMP 本身的控制功能，类型 7 用于边缘服务器与源服务器的控制通信。协议控制消息的消息流 ID 必须设置为 0，消息块流必须设置为 2，优先级为最高。表 6-5 描述了各种协议控制消息及其应用。

表 6-5 协议控制消息及其应用描述

| 协议控制消息     | 消息描述  |
|------------|---|
| 设置消息块大小的消息 | 用于通知对方新设置可供利用的最大消息块的大小  |
| 退出消息       | 用于通知对方如果正在等待后续的消息块来还原已经部分发出的消息，那么停止消息的接收，丢弃已经部分收到的消息并停止对消息的处理         |
| 确认消息       | 当客户端或者服务器收到的字节数等于窗口大小时，会向对方发送一个确认消息。窗口大小是发送方在未收到接收方任何确认消息之前可以发送的最大字节数 |
| 用户控制消息     | 客户端向服务器发送这个消息来通知对方相关的用户控制事件   |

续表

| 协议控制消息   | 消息描述  |
|----------|---|
| 窗口确认大小消息 | 客户端或者服务器通过此消息向对方确认窗口大小，消息负荷是一个窗口大小数值  |
| 设置带宽消息   | 客户端或者服务器可以向对方发送此消息来更新对方的输出带宽，这个输出带宽值通常通过窗口大小来表示。对方收到此消息后，如果发现当前的窗口大小值与此消息中的值不一致，则会发送窗口确认大小消息来确认窗口大小 |

### 命令消息

RTMP 服务器和客户端通过命令消息传递双方的命令信息，这些命令信息均采用 AMF 编码方式。AMF 编码方式是 Adobe 公司开发出的一种通信协议，它采用二进制形式，为基于 Flash 的播放器和远端服务器提供一种轻量级的、高效能的通信方式。目前 AMF 已经从 AMF0 版本发展到了 AMF3 版本。通过命令消息，用户可以执行连接、创造流、发布、播放、暂停的操作，接收方也可以通过命令消息向操作发送方返回请求命令的状态（比如 `onstatus` 和 `result` 消息）。发送方还可以通过命令消息来向接收方请求远程程序调用（RPC）。表 6-6 描述了各种典型的命令消息及其应用。

表 6-6 各种命令消息及其应用描述

| 命令消息                          | 消息描述  |
|-------------------------------|---|
| <code>connect</code> 命令消息     | 客户端向服务器发送连接命令来请求与服务器的一个应用程序建立连接                       |
| <code>call</code> 命令消息        | 发送方可以通过这个命令进行接收方的远端程序调用（RPC）。调用命令的参数中包含了被调用程序的名称      |
| <code>creatStream</code> 命令消息 | 客户端向服务器发送这个命令来创造一条用于消息通信的逻辑通道。音频、视频和元数据的发布都通过这条流通道来承载 |

续表

| 命令消息              | 消息描述  |
|-------------------|---|
| play 命令消息         | 客户端发送这个命令给服务器请求播放一个流。如果你创建了一个动态播放列表，各个节目需要不断切换和播放，这时需要调用 play 命令多次，并在命令的 reset 属性里传递 false 属性。相反，如果你只想播放一个列表中的特定节目，可以在使用这个命令时将 reset 属性设置为 True |
| play2 命令消息        | 与 play 命令不一样，使用 play2 命令可以在无须改变播放内容时间线的情况下将流切换成另外一个不同的码率。服务器会针对同一播放内容的几个不同码率维持几个文件，客户端可以通过 play2 来请求不同的码率                                       |
| DeleteStream 命令消息 | 当 NetStream 对象被毁坏以后，发送这个命令  |
| receiveAudio 命令消息 | 这个命令消息用来通知服务器是否给客户端发送音频流  |
| receiveVideo 命令消息 | 这个命令消息用来通知服务器是否给客户端发送视频流  |
| publish 命令消息      | 客户端向服务器发送这个命令消息来表示要发布一个命名的流   |
| seek 命令消息         | 客户端可以通过 seek 命令告诉服务器从某个特定时间点开始发送某个媒体文件或者播放列表流数据。如果服务器操作成功，则向客户端返回成功的通知，否则返回错误消息   |
| pause 命令消息        | 客户端可以通过这个命令告诉服务器暂停或者继续播放  |

下面简单描述 connect 命令和 play 命令的使用流程，如图 6-11 和图 6-12 所示。

connect 命令的流程描述如下：

(1) 客户端发送连接命令给服务器，请求与服务器上的一个应用程序建立连接。

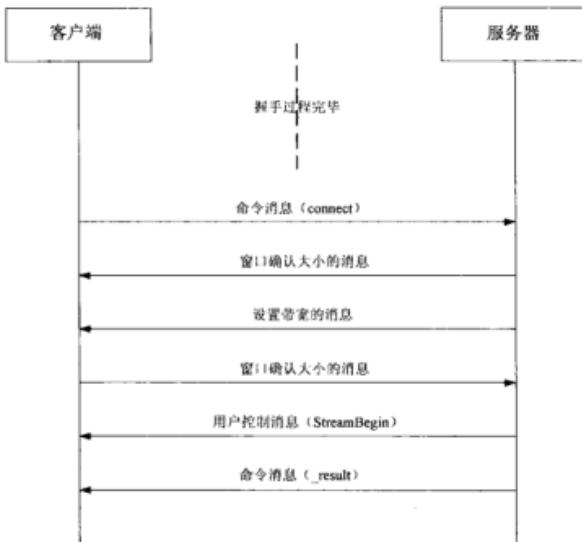


图 6-11 connect 命令的流程

- (2) 服务器收到连接命令后，服务器发送窗口确认大小的消息给客户端。服务器也与连接命令中提到的应用程序建立连接。
- (3) 服务器发送设置带宽的消息给客户端。
- (4) 客户端收到设置带宽的消息后，发送窗口确认大小的消息给服务器。
- (5) 服务器再发送另一个用户控制消息给客户端，比如 StreamBegin。
- (6) 服务器再发送一个控制命令通知客户端连接状态（成功或者失败）。这个消息也包括了一些特殊属性，比如 Flash 流媒体服务器（FMS）的版本、能力等。

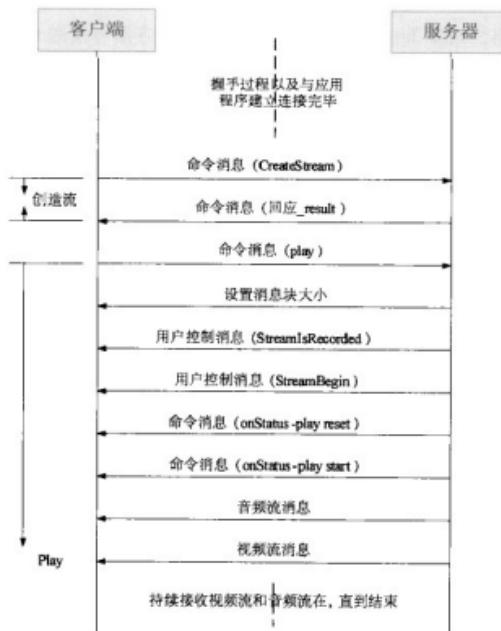


图 6-12 play 命令的流程

play 命令的流程描述如下：

- (1) 客户端在收到 CreateStream 命令后向服务器发送 play 命令。
- (2) 服务器收到 play 命令后发送协议控制消息来设置消息块的大小。
- (3) 服务器发送另一个用户控制消息，说明“StreamIsRecorded”以及相关消息流 ID。
- (4) 服务器发送另一个用户控制消息，通过“StreamBegin”向客户端表示流已经开始。

(5) 如果客户端前面的 play 命令发送成功，服务器再发送 Onstatus 的命令消息。如果 play 命令请求的相关流媒体没有找到，服务器会在 Onstatus 的命令消息中显示相关信息。

在以上信息交互完成以后，服务器就开始向客户端发送视频和音频数据。

#### **6.2.4 HTTP Streaming**

**RTSP/RTP 和 HTTP Streaming** 是目前应用最广泛的流化协议，目前电信运营商在 IPTV 的流化上主要以 RTSP/RTP 技术为主，而互联网视频网站则多倾向于使用 **HTTP Streaming** 的流化技术。

**HTTP Streaming** 的前身是 **Progressive Download** (渐进式下载)，它也是通过 **HTTP** 协议来传输文件。国内外较多的视频网站（如优酷、土豆等）一般都用 **Progressive Download** 的方式进行视频服务。**Progressive Download** 在用户点击播放视频节目时，会给用户发送视频文件，用户可以边下载、边播放，而无须等到文件下载完毕。如果用户暂停播放，服务器依然会给客户端发送视频文件，直至整个文件下载完毕或者用户关闭视频。这样，用户在决定退出视频时可能已经下载了较多的未播放部分，对于带宽资源是一种较大的浪费，尤其在并发访问较多的高峰时段。基于这点不足，人们提出了 **HTTP Streaming** 技术。**HTTP Streaming** 首先会将视频数据（包括直播的视频流和点播的视频文件）在服务器上进行编码，然后将编码后的数据进行更细粒度的分片，再把每个分片通过 **HTTP** 协议传输到客户端。与 **Progressive Download** 中客户端通过一个 **HTTP** 请求来下载整个视频文件的方式不同，**HTTP Streaming** 的客户端需要对视频文件的每个分片都发出一个 **HTTP** 请求，这样，在视频播放速度低于下载速度的情况下，客户端可以灵活控制 **HTTP** 请求的发出速度，从而保证用户在中途退出时不会出现下载浪费。另外，因为采用分片的特点，**HTTP Streaming** 还可以实现媒体播放过程中的码率切换，结合网络带宽资源，为用户提供更好的体验。在带宽资源充分的情况下，可为用户提供高码率的视频体验，在带宽

资源不足时可为用户提供低码率的视频体验。

与 **Progressive Download** 不同，**HTTP Streaming** 具有以下特点：**HTTP Streaming** 支持点播和直播，而 **Progressive Download** 只支持点播；**HTTP Streaming** 可以对分片文件进行加密，保证数字版权，而 **Progressive Download** 方式直接把媒体文件下载到客户端并进行缓存，无法保障版权所有；**HTTP Streaming** 是把媒体文件分割成多个小文件分片，支持在播放过程中根据网络带宽变化进行码率切换，而 **Progressive Download** 始终是以固定码率进行播放，不支持码率切换。

由于实现的业务能力相近，我们再把 **HTTP Streaming** 与 **RTSP/RTP** 做一比较：**HTTP Streaming** 基于 **TCP** 协议来传输，除了可靠性更高，也可以直接利用 **TCP** 的流控机制来适应带宽的变化；**HTTP Streaming** 可以将播放过的内容保存在客户端，除非版权控制或策略要求不能在客户端保存；**HTTP Streaming** 可以直接利用 **80** 端口来传输流媒体，在穿越只允许 **Web** 访问（**80** 端口）的防火墙时具有明显优势；**HTTP Streaming** 流化技术采用标准的 **HTTP** 协议来传输流媒体信息，只需要标准的 **HTTP** 服务器即可支撑，便于广泛推广。

目前，**HTTP Streaming** 几大主流阵营包括：**3GPP Adaptive HTTP Streaming**、**Microsoft IIS Smooth Streaming**、**Adobe HTTP Dynamic Streaming** 以及 **Apple HTTP Live Streaming**。下面将分别介绍这些流化技术的基本原理。

### 1. Apple HTTP Live Streaming

从概念上讲，**HTTP Live Streaming** 流化技术主要涉及三个部分：服务器组件、分发组件和客户端软件。

服务器组件主要负责从原始的音视频设备捕捉相应的音视频流，并对这些输入的媒体流进行编码，然后进行封装和分片，最后交付给分发组件来进行传送；分发组件主要负责接收客户端发送的请求，然后将封装好的流媒体分片文件连同相关的索引文件一起发送给客户端。对于没有采用 **CDN** 服务的源服务

器，标准的 Web 服务器就是一个分发组件，而对于大型的视频网站或者类似的大规模应用平台，分发组件还应包括支持 RTMP 协议的 CDN；客户端软件负责确定应该请求的具体媒体流，下载相关资源，并在下载后通过拼接分片将流媒体重新展现给用户。

在服务器组件中，硬件编码器捕获音视频数据后，将其编码成 H.264 的格式，然后通过流分割器软件进行封装和分片，以 MPEG-2 传送流（TS 流）的格式输出。分发组件的 Web 服务器或 CDN 中的 Cache 设备就是使用这些分片直接为客户端软件提供服务。除了为客户端提供分片文件，流分割器在分片后会产生一个索引文件来索引这些分片。索引文件的 URL 是 Web 服务器直接对外发布的，用户可以在登录网页时通过点击来直接获取。客户端软件在读取索引信息后，根据文件中的分片列表按顺序向服务器请求相关的分片文件，并在本地拼接后实现流畅播放。一个典型 HTTP Live Streaming 架构如图 6-13 所示。

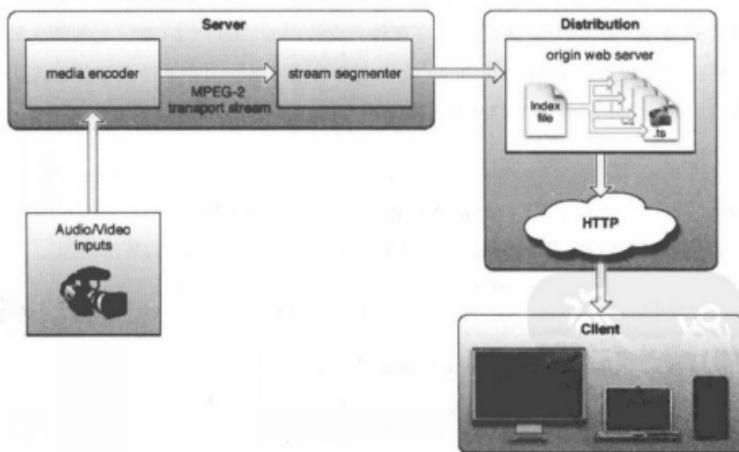


图 6-13 HTTP Live Streaming 的技术架构

图 6-13 中的输入数据可以是实时直播音视频数据，也可以是一个录制好的

点播音视频数据。音视频流或流媒体文件在经过编码、封装和分片后，变成多个以.ts 结尾的分片文件。流分割器产生的索引文件是以.M3U8 为后缀的，用户可以直接通过 Web 访问来获取。下面将详细介绍本流化技术架构中的各个组成部分。

### 服务器组件

服务器组件从功能组成上来讲，包含了一个媒体编码器和一个流分割器（或者文件分割器）。媒体编码器从音视频设备中获取实时的数据，对这些数据进行编码，然后进行封装并传输给流分割器。这里的编码格式应该是客户端所能支持的编码格式，比如 H.264 的视频编码格式和 AAC 的音频编码格式。编码后进行封装的格式也具有一定的要求，比如包含有音频和视频的数据通过 MPEG-2 的 TS 流格式封装，纯音频数据通过 MPEG 的基本流格式封装。这里所说的 MPEG-2 TS 流不要与 MPEG-2 的视频压缩方式混淆。TS 流只是一种打包封装的格式，它可以被不同的压缩格式拿来使用，比如 H.264。HTTP Live Streaming 只支持 H.264 的视频编码格式和 AAC 的音频编码格式。对于封装格式，如果是含有音频和视频的数据，HTTP Live Streaming 只支持 MPEG-2 TS 流的封装格式；如果是纯音频的数据，则只支持 MPEG-2 TS 流的封装格式或者 MPEG 基本流的格式。

流分割器是一个处理程序，它从服务器的媒体编码器读取封装的数据，然后再分割成一系列的小分片。每个分片都是一个独立的文件，但最后能重新拼接成一个连续播放的流。流分割器还会生成一个索引文件，包含了针对每个分片的索引信息和相关参考信息。流分割器每完成一个新媒体文件的分割，索引文件就会更新一次。索引文件中的索引信息可以被用来指示分片起始序列号、分片长度以及分片的 URL。流分割器也可以加密每一个分片，并同时创建一个相关的密钥文件。被分割出来的分片文件被保存为.ts 文件，索引文件本身被保存为以.M3U8 结尾的文件。M3U8 文件格式是 m3u 文件格式的扩展，而 m3u 在音频流化中用于保存 MP3 的播放列表，因此 HTTP Live Streaming 中的客户

## CDN 技术详解

端也能兼容 MP3 格式的音频流化。下面是一个简单的以.M3U8 结尾的文件示例，从中可以看出整个媒体流都包含在三个未经加密的 10 秒长分片文件中。

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-TARGETDURATION:10
#EXTINF:10,
http://media.example.com/segment1.ts
#EXTINF:10,
http://media.example.com/segment2.ts
#EXTINF:10,
http://media.example.com/segment3.ts
#EXT-X-ENDLIST
```

在上面的例子中，索引文件包含了各个分片的 URL，除此之外，索引文件也可能包含相关密钥文件以及针对不同带宽的替换流的索引文件。图 6-14 描述了一个索引文件中包含了其他替换流的索引文件。

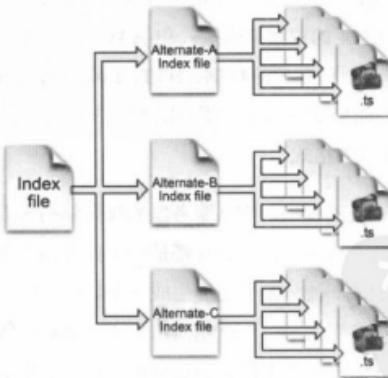


图 6-14 索引文件中的替换流索引文件

上面提到了流分割器，主要是针对实时获取的音视频来进行直播。如果你在服务器上已经保存了一个已经编码完毕的用于点播的流媒体文件，那么它的

流化将需要用到文件分割器。与流分割器类似，可以使用文件分割器来将流媒体文件封装到一个 **MPEG-2** 的格式中，并将它分成多个等长的分片来发给客户端。

### 分发组件

分发组件可以是 **Web** 服务器，也可以是 **CDN** 的 **Cache**，负责将分片文件和索引文件通过 **HTTP** 的方式发送给客户端，无须对现有的 **Web** 服务器和 **Cache** 设备进行额外的扩展、配置和升级，这也是 **HTTP** 流化的一大优势。在通过 **HTTP** 协议传输索引文件（**.M3U8**）和分片文件（**.ts**）时，消息中的 **MIME type** 分别为 **application/x-mpegURL**（或 **vnd.apple.mpegURL**）和 **video/MP2T**。源服务器在分发索引文件时，还应注意此文件的 **TTL** 属性。因为这些文件在实时直播时会不断重写，因此客户端在播放过程中对索引文件的每次请求都应该重新下载最新的版本，避免出现错误的情况。

### 客户端组件

用户通过网页点击播放一个视频时，客户端软件首先会根据 **URL** 来获取这个视频的索引文件。索引文件包含了可提供分片文件的具体位置、解密密钥以及可用的替换流。如果客户端选定了相关的流，将按顺序下载这些分片文件，一旦有足够的分片文件下载完毕，客户端将把这些分片拼接起来进行播放。客户端还需要向用户提供认证的接口以及向服务器进行认证，同时还需要获取相关的解密密钥并对分片文件进行解密。客户端会持续从索引文件中读取分片文件的 **URL** 并向分发部分获取相关的文件，直到读到索引文件中的 **#EXT-X-ENDLIST** 标签。如果没有遇到 **#EXT-X-ENDLIST** 标签，客户端将认为这个索引文件只代表了持续播放的媒体流的一部分，将会周期性地重新加载新版本的索引文件，客户端也将从新的索引文件中寻找新的分片文件和密钥信息。

## 2. 3GPP Adaptive HTTP Streaming

**Adaptive HTTP Streaming** 是 3GPP TS26234 提到的一种 **HTTP Streaming** 实

现方式。其架构如图 6-15 所示，主要分为内容准备组件、服务器组件以及客户端软件，并且支持 CDN Cache 设备。



图 6-15 Adaptive HTTP Streaming 的技术架构

内容准备组件负责进行媒体分片并将切片内容封装成符合要求的格式（如 3GP 格式的文件分片），然后生成分片的描述信息，同时向服务器组件推送这些媒体分片；服务器组件负责根据内容准备组件提供的媒体分片生成 MPD（Media Presentation Description，媒体呈现描述），响应客户端的请求，为其提供媒体文件；客户端软件负责请求和接收 MPD，请求相关内容分片，播放分片内容以及根据情况选择不同的码率集合。其中，MPD 描述了媒体文件的分片信息如分辨率、码率以及元数据等信息。客户端软件根据得到的 MPD 信息生成各媒体分片的 URL 信息并利用此 URL 向服务器组件请求响应的分片文件。其流程如图 6-16 所示，内容准备组件将媒体内容进行切片并封装成符合要求的格式，并生成媒体文件描述。然后把媒体内容以及 MPD 发送到服务器组件，服务器组件接受客户端请求 MPD，然后根据 MPD 请求发送媒体内容。图 6-16 中的 CDN Cache 是一个可选组件，如果使用 CDN 进行加速和内容分发，流媒体需先从服务器发送到支持 HTTP 协议的 CDN Cache，然后再由 Cache 交付给客户端软件。

### 3. Microsoft IIS Smooth Streaming

IIS Smooth Streaming 是在 Adaptive HTTP Streaming 基础上发展而来的一个实际产品。微软的全新一代 IIS 7 (Internet Information Services 7) 的 Media Server 3.0 模块提供了对 Smooth Streaming 和 Live Smooth Streaming 的支持。

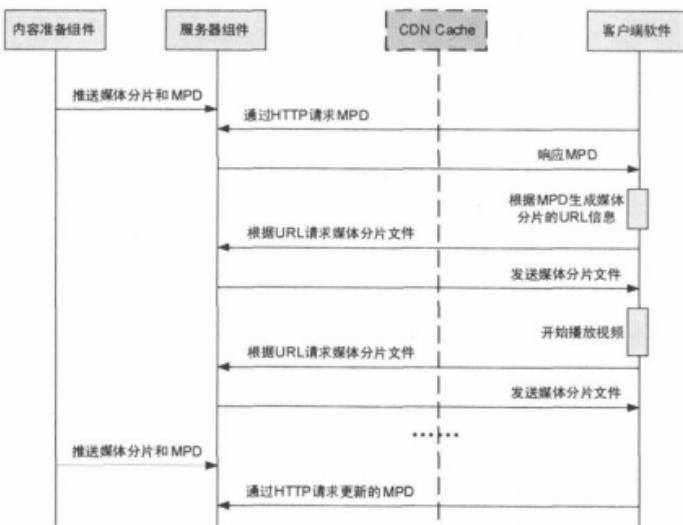


图 6-16 Adaptive HTTP Streaming 的基本流程

IIS Media Services 的扩展使得 Silverlight 客户端可以通过 HTTP 来适应流媒体的带宽，通过对视频源文件（如.wmv 等常用视频文件格式）的特殊处理，会在服务器上按照不同带宽要求生成多个不同版本的视频文件副本。IIS 在接收到 Silverlight 客户端请求后再决定将哪个视频文件副本通过流媒体发送到客户端，这个过程是完全动态的，也就是说当客户端的带宽发生变化时，传送流媒体的视频文件副本也会相应地改变。如当前客户端的带宽在 300kb/s 以下，则当前的流媒体文件副本可能为 280kb/s；当客户端的带宽上升到 2Mb/s 时，当前的流媒体文件副本就可能变成 1.5Mb/s 了。

IIS Smooth Streaming 在整个播放过程中完全是由客户端来维护其状态的，即码率的选择取决于客户端的决策判断，服务器在协议中是无状态的，只要根据客户端的不同请求返回相应的 HTTP 响应即可，这使得服务器的实现简单化。点播处理流程参见图 6-17。



图 6-17 Microsoft IIS Smooth Streaming 的基本流程

点播处理流程如下：

- (1) 用户根据一个 URL 链接向服务器请求 Manifest ( Media Presentation Description ) 文件。Manifest 文件是对媒体内容的描述，包括相关元数据等信息。
  - (2) 服务器返回 Manifest 信息。
  - (3) 客户端根据 Manifest 中的 URL 构造规则生成媒体分别的请求 URL 信息。
  - (4) 客户端服务器发送媒体分片请求消息。
  - (5) 服务器在收到请求消息后返回相应的媒体分片。

在播放的过程中，如果码率发生变化，URL 中码率相关的参数可用适当的值来代替客户端可以在最低码率 300Kb/s 与最高码率 2.436Mb/s 之间选择最适合当前网络带宽的码率进行播放。当然，码率的等级由编码器设置生成，可根据具体情况配置。

#### 4. Adobe HTTP Dynamic Streaming

Adobe Flash Player 10.1 和 Adobe AIR 2 在运行时引入了动态流的 HTTP 流媒体点播和在线视频直播服务。虽然实时消息协议（RTMP 协议）能够实现最低的选择延迟时间、最快的启动、动态缓冲和流加密，但动态流的 HTTP 缓存可以充分利用现有的基础设施（如内容交付网络、互联网服务供应商、办公缓存、家庭网络）。对于 HTTP Dynamic Streaming，点播和直播在内容准备工作流程上稍有不同。点播内容是通过一个简单的预编码生成 MP4 片段以及 Manifest 清单文件；直播的内容准备工作流程相对复杂一点，在播放的过程中生成 MP4。HTTP Dynamic Streaming 架构（参见图 6-18）与前面介绍的 HTTP Streaming 系统架构很类似，这里不进行深入阐述。

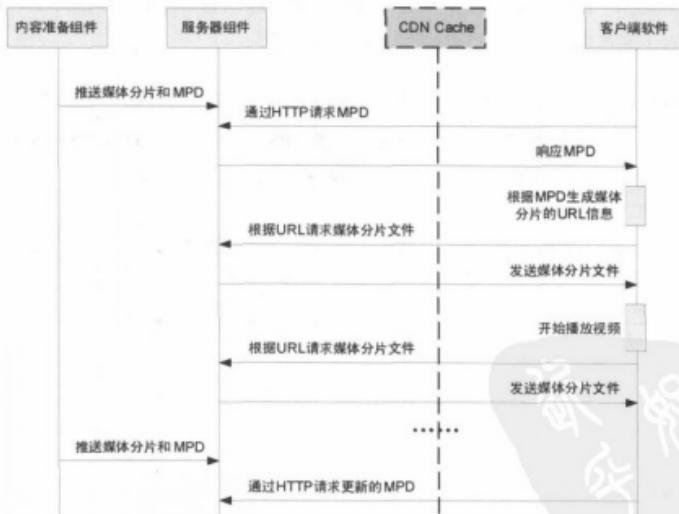


图 6-18 HTTP Dynamic Streaming 的基本流程

### 6.2.5 MPEG-2 TS

MPEG-2 TS 指 TS 格式封装的、MPEG-2 编码格式的媒体流。之所以把这个组合方式拿出来讲一下，是因为目前大多数 IPTV 系统使用的是这种内容源，其他业务系统如果使用来自电视台、电影公司和广电企业的内容源，基本也是这种格式的。

MPEG-2 是“活动图像及有关声音信息的通用编码”（Generic Coding of Moving Picture Associated Audio Information），由 MPEG（Moving Picture Experts Group）组织开发。很多年以来，MPEG-2 都是我国广电行业的通用标准。

TS 最初是 MPEG-2 系统层中定义的一种符合信息流格式，目标是基于宽带的数字视频广播，并且支持多种基本媒体流和多媒体编码标准。在国际上，TS 已经有十多年实际的大规模普遍性应用的历史了。

MPEG-2 系统标准定义了复用一个或多个音频、视频和数据元素流的方法，如图 6-19 所示。MPEG-2 系统标准有节目流（PS）和传输流（TS）两种数据流，PS 用可变长度包，TS 用固定尺寸包（188 字节）。

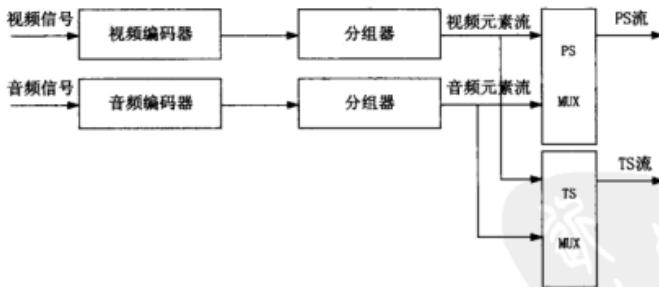


图 6-19 MPEG-2 定义复用方法

为了保证数字电视信号在传送中的抗干扰能力，从视频元素流到 TS 流这个过程中已经加入了 FEC 纠错码。在 DVB-S 标准中，规定了 5 种 FEC 速率——

1/2、2/3、3/4、5/6、7/8。以 7/8 为例，其实际意义是：在一个 TS 流中，只有 7/8 的内容是有节目内容的 PS，而另外的 1/8 则是 FEC 纠错码。

TS 复用那些没有共同时间基准的数据流，固定长度为 188 字节。这种固定长度的包结构对抗传输误码非常重要。当传输中破坏了一个 TS 包同步信息时，接收侧可以在固定位置检测它后面包中的同步信息，从而恢复同步，避免信息丢失。这也是广电系统最初选用 TS 码流的原因之一。在这 188 字节的包中，至少有 4 字节用做包头，其余字节用做传输存储数据、音频和视频信息。TS 包头含有同步、包私用、节目标识和错误状态信息。TS 数据包结构如图 6-20 所示。

TS 具有很好的开放性，能支持多种编码标准，包括 MPEG-1、MPEG-2、MPEG-4、H.264、VC-1，也包括我国的 AVS 标准。

随着 H.264 编码的广泛应用，广电系统也逐步转向支持 H.264 (MPEG-4 AVC) over TS 封装格式。一个典型的 H.264 TS 流的封装方式如图 6-21 所示。

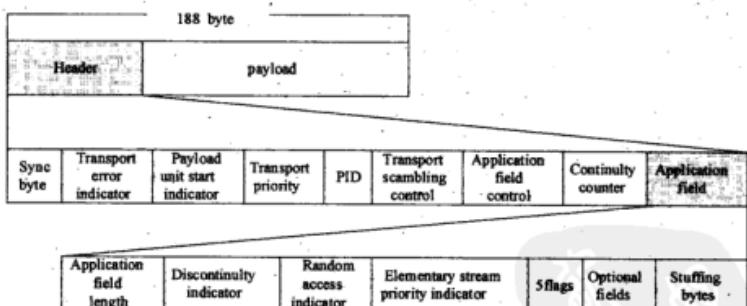


图 6-20 TS 数据包结构

| IP | UDP | RTP | TS | H.264 |
|----|-----|-----|----|-------|
|----|-----|-----|----|-------|

图 6-21 H.264 TS 封装结构

可以简单理解为：H.264 这一层完成原始文件的压缩编码，TS 这一层负责音视频的复用以及同步，RTP 这一层负责流的顺序传输，UDP 这一层负责数据包交付，IP 层负责传输路由选择。

## 6.3 流媒体业务对 CDN 提出的要求和挑战

### 6.3.1 流媒体加速与 Web 加速之间的业务差异

流媒体业务具有实时性、连续性、时序性的特点，流媒体提供的过程就是把音视频媒体信息由流媒体服务器通过网络连续、实时传输到客户电脑。在本章前面的内容中我们已经了解到，流式传输的过程中，客户不必等到整个文件全部下载完毕，而只需经过几秒或十几秒钟的启动时延即可播放。当音视频媒体在客户端播放时，其流媒体的余后部分将在后台继续下载。流式传输方式不仅使启动时延十倍、百倍地缩短，而且不需要太大的缓存容量。

显然，流媒体实现的关键技术就是流式传输，这与传统的以固定大小的文件方式提供的 Web 图片或文字内容差别很大。如果仅仅围绕 CDN 这件事来说，我们可以把它们之间的差异总结为内容类型、用户行为、内容管理和内容分发方式这几方面的差异。

**内容类型差异。**我们在网站上看到的除了流媒体的 Web 内容，包括文字、图片、动画、照片等，通过设计将它们组织在一起，形成一个完整的表达。这些内容都是以静态文件形式提供给用户的，也就是说，它们有固定大小，有始有终，终端则完全获取整个文件后才展现给用户。而流媒体内容则完全不同，它们可能没有固定大小，比如直播流，只有始没有终。或者可能体积特别大，无法很快完全获取，终端只能边下载边播放。更重要的是，用户几乎感觉不到网页内容传送过程中有什么丢包、抖动之类的网络问题，无非稍微等一会儿。

但如果流媒体播放过程中出现这样的问题，用户会看到花屏、马赛克，甚至画面停止，感觉是非常明显的。

**用户行为差异。**这个非常好理解，我们在电脑上看视频的时候，经常会拖动一下播放进度条，跳过不感兴趣的片段，或者对感兴趣的片段来回看好几遍，暂时离开的时候会单击暂停按钮。近年来随着IPTV、智能电视的发展，我们更喜欢在电视上观看在线视频，就像看电视和DVD光碟那样，用遥控器来控制前进后退，切换片段等。而对文字、图片等网页内容，我们是绝不会进行这些操作的。

**内容管理要求差异。**一张普通的网页展示出来，上面大大小小的内容元素成千上万，这些内容经常在一天内就被网站编辑们更新、替换。而如果读者留心影视网站就会发现，通常一个网站的片库也就几百部片子，几十部电视剧。一部电影节目，可能在网站上放上几个月。因为影视剧是需要采购版权的，更新需要比较高的成本，没有哪个网站愿意花钱买一堆用户不爱看的片子放在那儿。相比之下，流媒体文件体积大，数量小，生命周期长。因此，流媒体CDN在内容管理方面采取了一些不同的设计方式，比如需要建立内容名称与内部URL的映射关系，会对文件进行切片管理，会定期进行内容的冷热度判断，会在Cache设备中存储大量内容等。这些将在设计差异中进一步讨论。

**回源要求差异。**因为流媒体文件传送带宽需求高，而且往往需要维持TCP长连接，所以一旦CDN回源比例过高，源站服务器I/O将不堪重负。而Web内容本身生命周期就短，文件获取又很快，所以其回源比例大一些是没什么问题的。CDN对内容采取的分发方式分为PULL和PUSH两种。PULL是被动下拉的方式，PUSH是主动推送的方式。对于流媒体内容，系统一般会选择对热点内容采取PUSH方式的预分发，而普通的网页内容几乎100%是PULL方式的。通过PUSH的方式降低CDN回源对源站服务器的压力。关于PULL方式与PUSH方式的差异，将在设计差异中更详细地讲解。

### 6.3.2 流媒体 CDN 系统架构描述

流媒体 CDN 系统总体上符合 CDN 系统通用架构，由管理支撑子系统、负载均衡子系统、内容管理子系统、流媒体服务子系统组成，如图 6-22 所示。

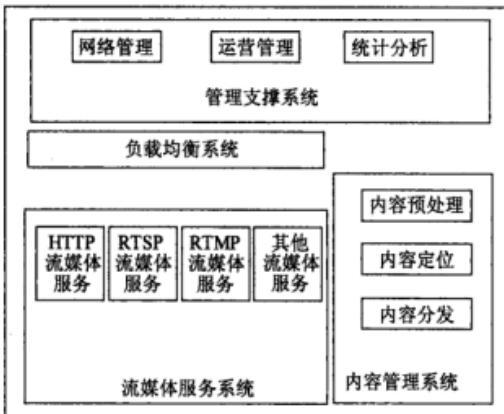


图 6-22 流媒体 CDN 系统架构框图

**管理支撑子系统**是 CDN 系统的网络管理和业务管理系统，主要功能包括如下几项。

**网络管理：**提供拓扑管理、节点管理、设备管理、配置管理、故障管理、性能管理以及网络安全管理等。网络管理模块不仅负责提供对整个系统进行日常运维的手段，还要负责收集执行业务策略所需的实时统计数据，比如为 GSLB 提供调度策略执行依据，在全网范围配置业务属性，分配网络和能力资源。不同的 CDN 服务提供者，会根据自身的不同考虑为网络中的各种资源建立资源模型，对底层的资源进行抽象，基于这些资源模型对 CDN 网络进行更加有效的管理，包括建立策略管理机制，使得 CDN 业务流程更加灵活。

**运营管理：**负责客户管理、客户自服务实现、产品/业务能力管理、工单管理、认证管理、计费和结算管理等。这些功能使 CDN 系统对外提供服务、销

售成为可能，也为CDN的客户提供了友好的业务操作界面。因此，虽然运营管理模块并没有参与具体业务能力提供，但却对整个CDN系统的商业竞争力起到至关重要的作用。

**统计分析：**包括日志管理功能和数据筛选、分析功能，以及报表生成功能。统计分析功能应该对按照不同的指标对CDN网络运行情况和服务情况进行统计分析，同时以灵活的、有针对性的方式向客户呈现。比如，网站客户可能比较关注他的网站用户分布在哪些区域，这些区域的人喜欢什么内容，不喜欢什么内容，以及用来进行内容采购的规划。这些数据只能从CDN中提取，网站自己很难获得。因此，CDN系统的统计分析结果对客户有着非常重要的参考价值。

**业务接口：**负责和其他系统之间的接口适配功能，包括与外部BOSS系统的接口、与门户系统的接口，并向SP提供自助服务接口。

**负载均衡子系统**负责用户访问调度，根据对用户位置、设备负载、内容位置等信息的判定，执行预先设置的负载均衡策略，将用户调度到最合适的节点设备上进行服务。在流媒体CDN系统中，用户访问的调度会更多考虑内容命中，主要是因为流媒体内容文件体积大，业务质量要求高，如果从其他节点拉内容再向用户提供服务会带来额外的延迟，影响用户体验。为进一步提高命中率，流媒体CDN系统普遍采用了对热点内容实施预先PUSH的内容分发策略，因此，负载均衡子系统与内容管理子系统之间会有比较频繁的交互查询行为。另外，因为流媒体CDN系统通常规模比较大，节点数目多，所以调度精确性要求要比传统的Web CDN更高，这对负载均衡子系统的性能、算法、灵活性都提出了更高要求。

**流媒体服务子系统**是指为用户提供流媒体服务的各种设备组成的服务系统，具体来说就是一些Cache设备或者Cache的集群，以及对集群和设备进行资源统计、配合负载均衡系统和运营管理进行工作所需要的周边网元设备。在流媒体服务系统中，主要关注的技术是对不同流媒体协议、不同编码格式、不同播放器、不同业务质量要求等的适应。一般来说，CDN服务商会将流媒体子系统中采用垂直部署业务能力的方式，也就是说，从中心Cache到区域

## CDN 技术详解

Cache、边缘 Cache 统一部署单独的业务能力。这里的业务能力，可以根据不同流媒体协议的适配和对用户提供服务的能力进行细分，也可以根据直播、点播这样的大业务类别来划分。因为不同的业务能力实现方式、设备要求、组网方式往往差别较大，所以彼此之间无法交叉互通，垂直部署的方式更有利于系统功能扩展和日常运维。有些服务商采用专用的流媒体协议服务器来组网，有些采用私有协议封装不同流媒体协议的方式。

内容管理子系统负责对整个 CDN 网络的内容分布情况进行管理，从内容进入 CDN 网络开始，内容管理系统就负责对内容进行预处理，以适应 CDN 内部分发要求和业务层面的要求；进行内容位置管理，用于访问调度时的内容定位；进行内容在 CDN 全网的分发，保证冷热内容的合理分布，从而使得 CDN 系统对用户提供服务的质量和成本得到优化。

### 6.3.3 小结

作为一个专业服务系统，CDN 的技术发展是稍微滞后于业务发展的，也就是说，业务变化推动了 CDN 技术的变化，这些差异性驱动 CDN 技术从传统的 Web CDN 向流媒体 CDN 演进。下面，我们用表 6-7 对流媒体 CDN 与 Web CDN 的差异进行归纳。

表 6-7 流媒体 CDN 与 Web CDN 的对比

| 主要差异点            |      | 流媒体 CDN           | Web CDN            |
|------------------|------|-------------------|--------------------|
| 业<br>务<br>差<br>异 | 内容类型 | 大文件、实时流、QOS 要求高   | 小文件、固定大小、QOS 要求低   |
|                  | 用户行为 | 拖曳、暂停等播放控制        | 下载后浏览              |
|                  | 内容管理 | 内容冷热度差异明显，内容生命周期长 | 内容冷热度差异不明显，内容生命周期短 |
|                  | 回源要求 | 回源比例小             | 回源比例大              |

通过上面的介绍可以看出，流媒体 CDN 与 Web CDN 的工作原理和实现机制基本相同，在本书前面的相关章节都已经进行了详细讲述，因此这一章我们

重点分析流媒体CDN系统中使用的差异性技术。在表6-8中，我们对流媒体CDN和Web CDN设计实现的差异点进行了小结。现在已经投入商用的CDN系统，基本都是同时提供Web CDN能力和流媒体CDN能力的，而且这两种能力的实现在系统内部几乎都是互相隔离的，从调度系统到节点设备都没有交叉互用，说明这两条技术路线已经差别比较大了。在随后的几节，我们将针对这些差异点，研究探讨流媒体CDN系统实现的关键环节和设计思路。

表6-8 流媒体CDN与Web CDN的设计差异

| 主要差异点 |        | 流媒体CDN                | Web CDN                         |
|-------|--------|-----------------------|---------------------------------|
| 设计差异  | Cache  | 支持多种流化协议，硬件配置大存储、高I/O | 支持多协议（HTTP、FTP等），硬件配置小存储、高性能CPU |
|       | 负载均衡   | DNS+HTTP重定向方式         | DNS方式                           |
|       | 内容分发方式 | 热片PUSH，冷片PULL         | 全PULL方式                         |
|       | 组网     | 多级组网，可能要求组播、单播混合组网    | 两级组网                            |

## 6.4 流媒体CDN系统的关键技术实现

### 6.4.1 Cache的设计实现

CDN的Cache设备是直接为用户提供内容的设备，主要有两个功能：一是对内容进行缓存，二是直接响应用户的content访问请求。

服务于流媒体业务的设备与普通服务器在功能要求上有很大差异，普通服务器面向计算，设计重点集中在计算性能、数据可靠性保障等；而流媒体设备面向资源，设计重点是对存储设备、内存、存储I/O、CPU运算等多种资源的

有效协调。流媒体 Cache 设备的直接设计目标和性能衡量指标是用户接收的媒体质量、用户端启动延迟，以及传送多媒体数据对网络资源的消耗等。

另外，上面我们讨论过，流媒体 CDN 和 Web CDN 在内容类型、用户行为上存在比较大的差异。也就是说，这两类业务对 Cache 的两个主要功能的要求都不一样。因此，流媒体 CDN 的 Cache 设备与 Web Cache 无论在软件实现还是硬件要求上差异性都很大，我们很少看到这两种业务共用一台设备。

### 1. 协议实现

从协议实现上来看，对于前进、后退、暂停等这些播放器行为的支持，依赖于 Cache 对终端请求的理解和响应，而这种响应，又依赖于传送协议对这些动作的支持。RTSP、RTMP 等流控协议就是因此而产生的，近两年 HTTP Streaming 发展得也非常火热，有逐渐成为网络视频主流流控协议的趋势。CDN 的 Cache 设备在系统中的角色主要是同时作为这些协议的服务器和客户端。

当用户请求的内容在 Cache 上命中时，Cache 直接向用户提供流服务，此时 Cache 设备充当流媒体服务器的角色，如图 6-23 所示。它从磁盘上读取文件，进行实时传输协议封装，再经过 IP 网络传送给客户端。

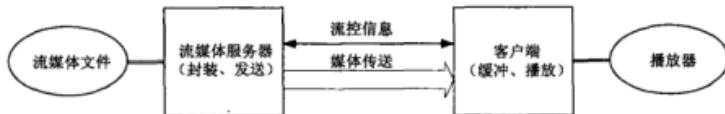


图 6-23 Cache 充当流媒体服务器

当用户请求内容未能在 Cache 上命中时，Cache 会从上一级 Cache 或者源站服务器获取内容，再提供给用户。但这个过程是一个边获取边提供的过程，而不是全部获取完成以后再提供给用户，这是流媒体服务的特殊要求。此时，Cache 在用户与另一个流媒体服务器之间扮演代理的角色，在两个连接之间建立协议转换，如图 6-24 所示。

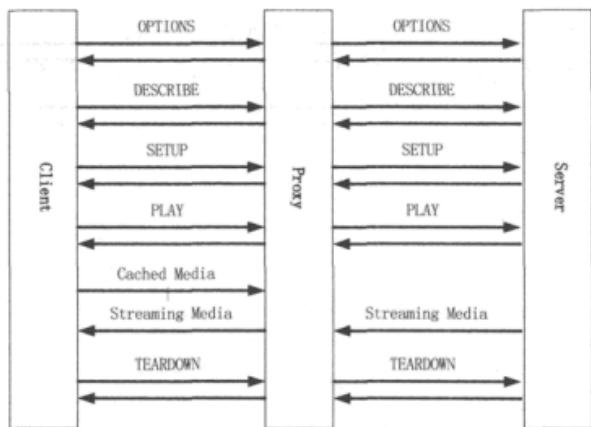


图 6-24 Cache 充当代理角色

由上面的分析可以看到，流媒体 CDN 的 Cache 设备必须支持多种编码方式和流控协议，身兼服务器、代理服务器、客户端的角色，才能完成任务。要做好流媒体 Cache，最重要的一项基本功就是对各种协议的理解、扩展和优化。

## 2. 缓存算法

从本书前面章节的内容中我们已经了解到，无论对于何种类型的 Cache 设备，缓存算法都是至关重要的。基于流媒体业务的特性，对流媒体 Cache 来说，除了与 Web Cache 同样要研究缓存替换算法外，还要研究媒体预取算法、前缀缓存算法等。

关于这些算法，业界有比较知名的基础算法实现，但多数 CDN 服务提供者还是会在这些基础算法之上叠加自己的独特算法，或者是将几种算法综合使用。理解下面几个比较知名的基础算法，会对读者理解缓存算法设计思路有很好的帮助。

**流媒体缓存替换算法。**缓存替换算法是管理缓存的主要手段，也是决定

Cache 服务器性能的核心因素。传统的替换算法，如 LFU (Least Frequency Used)、LRU (Least Recently Used)、LRU-Threshold 等；这类算法以访问频率或最近访问时间判断数据的冷热度，从而决定缓存内容的替换。这种算法是流媒体缓存替换算法的基础，但显然不够精细，后来研究人员在此基础上增加了一些更适用于流媒体内容和业务特征的算法。比如 Tewari 提出了一种基于资源的缓存算法 RBC (Resource Based Caching)，算法中强调缓存容量与磁盘 I/O 的平衡。这是因为流媒体文件通常是对磁盘持续读写的，相对于体积小的 Web 内容文件，一次访问产生的连续读盘时间长得多。磁盘 I/O 反映了 Cache 能够同时传输的流数，此时磁盘 I/O 成了一项重要的性能指标。为避免磁盘 I/O 和存储空间两种资源一种过载而另一种空闲的情况，在选择缓存对象时，基于资源的缓存算法设计了一种“跷跷板”策略，优先选择对稀缺资源占用少的媒体对象，以平衡系统对磁盘 I/O 和存储容量的利用率。

还有一种流媒体缓存替换算法，是针对分层编码的。不同媒体层的媒体片段被赋予不同的权值，权值首先与媒体层相关，基本层被用到的概率最高，因此权值最高。权值还与媒体片段被点播的频率成正比，与片段长度成反比。通过比较不同内容片段的权值，权值最小的首先被替换。

**媒体内容预取算法。**这种算法的依据是，用户接收流媒体内容有一定的连续性，一旦启动接收，则有很大概率会接收后续数据。因此，媒体内容预取算法就是在用户接收流媒体的同时，利用空闲网络资源，将后续数据预先下载到缓存 Cache 中。这类算法中有服务 Cache 向父 Cache 的预取算法，也有 Cache 和用户之间的预取算法。感兴趣的读者可以研究一下 Rejate 设计的基于滑动窗口的预取算法，和 Fan 设计的预推算法。这两个算法是比较有代表性的。

**前缀缓存算法。**所谓前缀，这里指的是媒体内容开始的一部分。这种算法的依据是，用户对流媒体服务的启动延迟相当敏感，因为除了最开始的这一部分，后续内容在网络带宽条件好的时候都可以预先推送到终端进行缓存，时间到了再播放。另外一方面，节目开始的部分是被访问最频繁的。前缀缓存算法

是将媒体内容的开始部分缓存在距离用户较近的 Cache 中，从而有效降低启动延迟。

### 3. 存储设计

流媒体 CDN 的 Cache 设备与 Web Cache 对 CPU 性能和 I/O 能力要求差别很大。简单地说，Web Cache 需要应对小文件、高并发要求，这样的业务特性对 CPU 消耗非常大，而流媒体 CDN 的 Cache 则需要应对大文件持续读写，TCP 长连接维持要求，这对磁盘 I/O 能力要求非常高。另外一个主要差别是，Web Cache 一般不需要配置很大的存储空间，而流媒体 Cache 则需要外挂 TB 级别的磁盘，因为流媒体文件实在是太占用存储空间了。

为了保证连续媒体数据的实时磁盘访问，需要将数据以合适的块大小分布在磁盘的各个区域，并以最佳的顺序从磁盘中读取。

从硬盘上读取数据，其时间消耗取决于磁头移动速度、磁盘旋转速度和数据块的大小。一般来说，磁盘的旋转速度恒定，读取数据的速率也是固定的，那么减少时间消耗的关键就在于如何减少磁头移动。目前比较成熟的算法是磁盘区域算法，它将磁盘分成若干区域，媒体数据按照一定的规则放置在这些区域内，在某个时间段内，磁头只在同一个区域内移动。另外，数据分块策略也是决定 I/O 效率的重要因素。如果不进行数据分块的话，当大多数用户集中请求一个节目时，存放该节目的磁盘必须同时响应很多点播请求，导致磁盘处于忙状态，而实际上其他磁盘则处于空闲状态，总 I/O 能力没有充分利用。因此，通常将媒体文件数据分成若干小数据块放置到不同的硬盘上，甚至分配到不同 Cache 的硬盘上，这样用户的点播请求会分配到多个磁盘上，从而平衡磁盘负载。这种算法称为分块和交错算法。现在有很多存储解决方案都提供了自带的磁盘负载平衡功能，如果 Cache 使用外挂磁阵的方式，可以依赖磁阵自带功能解决掉大部分磁盘负载均衡问题。否则就需要 Cache 自己考虑算法和计算处理，这会消耗掉一部分 Cache 设备的 CPU 处理能力，从而对 Cache 其他处理任务的性能造成一定影响。

分布式存储技术因其大容量、低成本的特点，目前也被业界关注和研究作为流媒体 CDN 系统的存储解决方案之一。常用的分布式存储技术包括分布式文件系统和分布式数据库，由于采用了数据副本冗余（每份数据复制 2~3 份）、磁盘冗余（Raid1、Raid10、Raid5）等技术，通常可以提供良好的数据容错机制，当单台存储设备断电或者单个存储磁盘失效时，整个存储系统仍能正常工作。分布式文件系统更常见于高性能计算领域，例如 Google 的 GFS、Sun Lustre 文件系统、IBM GPFS 文件系统等。数据以文件形式存放在系统当中，对外提供文件系统接口。典型分布式存储架构如图 6-25 所示。

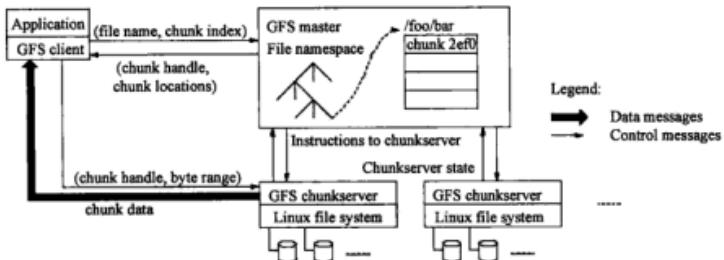


图 6-25 典型分布式存储架构（Google File System）

由于分布式文件系统提供 POSIX 标准文件系统接口，CDN 如果采用分布式文件系统进行存储，对于内容缓存应用系统而言使用分布式文件系统与使用本地文件系统没有任何差别。

#### 6.4.2 负载均衡系统设计实现

在负载均衡一章中我们讨论过，负载均衡设备在进行用户访问调度时，会综合考虑很多静态的、动态的参数，包括 IP 就近性、连接保持、内容命中、响应速度、连接数等。但没有哪个 CDN 会考虑所有参数，而是会根据业务特点进行一些取舍，否则均衡系统就太复杂了。而流媒体 CDN 在进行用户访问调度时，会更多考虑内容命中这一参数。内容是否命中可能在 GSLB 调度时就进

行考虑，也可能在 SLB 范围进行考虑。这是因为流媒体文件未命中带来的成本比 Web CDN 高得多。这里的成本是指 Cache 向其他设备查询、请求内容的过程，也指回源压力。

另外，流媒体 CDN 通常网络规模都比较大，这意味着网络内的节点设备数量多，部署位置也比较低。当然这两项“多”都是相对于 Web CDN 来说的。因此，流媒体 CDN 进行负载均衡的颗粒度要求也比 Web CDN 高。在本书第 5 章中我们讲过，有两种 GSLB 实现方式，一种是基于 DNS 的，一种是基于应用层重定向的。基于 HTTP 重定向方式的负载均衡由于能够看到用户 IP 和请求的具体内容，因而能够实现更精细、更准确的调度。因此，流媒体 CDN 大多会选用 HTTP 重定向方式，有的是与 DNS 方式结合使用，有的是直接使用。

对于基于 HTTP 重定向的负载均衡系统，在本书第 5 章已经有了比较详尽的讲述，本章不再重复。

对于流媒体服务 Cache 的选择，通常会基于一套考虑众多因素的算法，这里介绍一种基于增益模型的服务器选择算法的基本原理，读者可以通过这个算法的原理了解 Cache 选择过程中需要考虑的一些重要的因素，从而更容易理解其他算法。

基于增益模型的服务器选择算法的基本思路是以系统增益最大化为目标，若有一个候选 Cache，则增益最大者获选。这里系统增益包含两个部分：

- 用户接收媒体流获得直接增益，称为用户增益。
- 避免网络资源过多消耗获得间接增益，称为网络利用率增益。

其中，用户增益由媒体质量子增益、启动延迟子增益和网络消耗子增益构成，本书并不侧重讲解具体算法，这里只给出一个算法实例，对其他的算法只讲解原理。

这里， $\text{RevenueQ}(v)$ 、 $\text{RevenueL}(v)$  和  $\text{RevenueR}(v)$  分别是媒体质量子增益、

启动延迟子增益和网络消耗子增益，几个  $v$  参数分别对应子增益的权值。而媒体质量子增益、启动延迟子增益和网络消耗子增益又与视频信号的峰值信噪比、网络往返延迟、媒体片段大小等因素有关。

网络增益，是指网络传输的消耗代价。由于网络带宽是一项竞争性资源，所以在 Cache 选择算法上需要考虑这个因素。比如，如果对一个用户请求来说，几个 Cache 的可用输出带宽有显著差别，则应该优先选择链路带宽较高的 Cache：

$$\text{Revenue}_v(S_j, v) = \text{Rev}_v(v) \times p_N$$

给出用户增益和网络利用率增益后，我们定义 Cache 选择增益为：

$$\text{SelectionGain}(S_j, v) = r \times \text{Revenues}(S_j, v) + (1 - r) \times \text{Revenue}_v(S_j, v)$$

应该注意，在 Cache 选择算法中，增益的计算依赖于对某些网络参数的测量，当然测量越频繁计算实时准确性越高，但网络参数的测量是要消耗资源的，比如增加了计算负担，增加了网络带宽占用。所以在设计算法的时候应该考虑参数测量的效率问题。所幸对于现在的 IP 网络而言，很多网络参数还是比较稳定的，包括链路有效带宽、环路延迟等，所以测量所带来的资源消耗尚可接受。

上面所述的 Cache 选择算法设计思路，既适用于选择对用户提供服务的 Cache，也适用于选择父 Cache。

### 6.4.3 内容分发机制设计实现

内容分发是指内容文件在 CDN 内部从中心节点到各个区域、边缘节点的分发。这一过程的实现方式分为 PULL 和 PUSH 两种。

PULL，是一种被动的下拉方式，通常由用户请求驱动。当用户请求的内容在边缘 Cache 上未命中时，边缘 Cache 启动 PULL 方法从内容源或者其他 Cache 实时获取内容。边缘 Cache 需要支持边拉边放，即一边从其他位置获取内容，

一边将内容流化后提供给用户。在 PULL 方式下，内容的分发是按需的。

PUSH，是一种主动推送的方式，通常是由内容管理系统发起的，将内容从源或者中心媒体资源库分发到各边缘的 Cache 节点。分发的协议可以采用 HTTP/FTP 等。通过 PUSH 分发的内容一般是访问热度高的内容，这些内容通过 PUSH 方式预推到边缘 Cache，可以迅速响应用户访问请求。对于 PUSH 分发需要考虑的主要问题是分发策略，即在什么时候分发什么内容。一般来说，内容分发策略可以由 CDN 的使用者或者 CDN 管理员人工确定，也可以通过智能策略决定，即所谓的智能分发。它根据用户访问的统计信息，以及预定义的内容分发的规则，确定内容分发的过程。PULL 方式和 PUSH 方式内容分发示意图如图 6-26 所示。

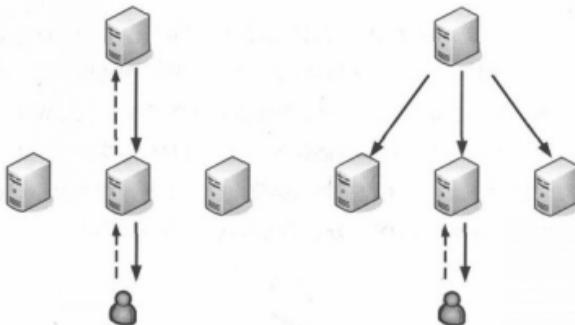


图 6-26 PULL 方式和 PUSH 方式内容分发示意图

在实际的 CDN 系统中，一般两种分发方式都支持，但是根据内容的类型和业务模式的不同，在选择主要的内容分发方式时会有所不同。通常，PUSH 方式适合内容访问比较集中的情况，如热点的影视流媒体内容，PULL 方式比较适合内容访问分散的情况。

有的 CDN 系统实现 PUSH 方式是主动模拟一个客户请求，使系统产生 PULL 响应。也就是说，PULL 方式和 PUSH 方式在技术实现上是没有什么差别

的，只是触发方不同而已。这种差异性读者能够理解即可，本书不再深入讨论。

#### 6.4.4 组网模式

我们已经了解到，对使用 CDN 服务的 SP 来说，CDN 的作用在于尽量就近为用户提供服务，帮助 SP 解决长距离 IP 传输和跨域传输带来的种种业务质量问题。因此，为用户提供服务的 Cache 设备一定部署在离用户比较近的地方。另一方面，CDN 的建设者从成本角度考虑，又不能把所有内容都存放在这些离用户最近的节点中，这会消耗大量存储成本，所以这些提供服务的 Cache 设备会根据需要从源站服务器或者其他 Cache 获取内容。这样就形成了 CDN 网络分层部署的概念。

从网络分层上看，Web CDN 通常是两级架构，即中心-边缘。而流媒体 CDN 通常有三级以上架构，即中心-区域-边缘。产生这种区别的原因在于流媒体回源成本比较高，源站服务器响应一次流媒体内容回源请求，要比 Web 内容回源消耗更多资源。尤其对于流媒体直播业务来说，只要直播节目没结束，服务器就需要长时间持续吐流，如果没有第二层节点作为中继，那么中心节点的压力将是不可想象的。流媒体 CDN 的分层部署方式如图 6-27 所示。

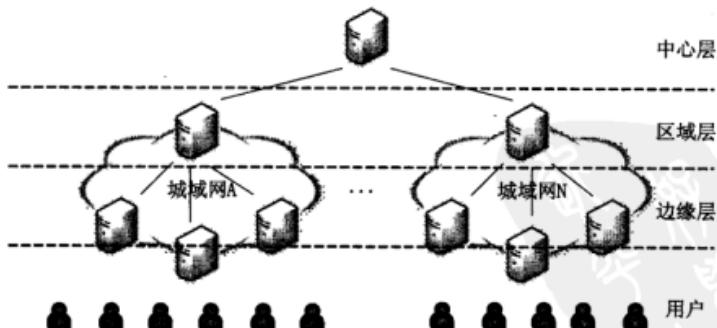


图 6-27 流媒体 CDN 的分层部署方式

简单地说，分层部署的方式，对点播业务而言的主要意义是节省存储成本，对直播业务而言在于减少带宽成本。在点播业务中，边缘 Cache 只需存储用户访问量大的内容或者内容片断，其余内容存储在区域 Cache 中。在直播业务中，边缘 Cache 从区域中心获取直播流，而不需要直接向中心节点获取，从而节省了区域中心到中心节点这一段的大部分带宽。因为直播流在各个 Cache 中都不需要占用很大的存储空间，只需少量缓存空间即可，所以直播业务方面并不用注重考虑存储成本。

考虑到电信运营商的 IP 拓扑和流量模型，区域中心 Cache 通常部署在重点城市的城域网出口的位置，以保障向各个边缘 Cache 的链路通畅。边缘 Cache 的位置选择则以整个节点能够提供的并发能力为主要依据，依据业务并发数收敛比，计算出单个 Cache 需要覆盖的用户规模，从而选择一个合适的部署位置。当然，边缘 Cache 离用户越近，服务质量越好，但覆盖的用户数越少，部署成本越高，这是一个平衡问题。因此，节点位置的确定要综合考虑服务质量、带宽成本、设备成本、IP 拓扑等因素。

#### 6.4.5 内容文件预处理技术

内容文件预处理是指视频内容进入 CDN 以后，进入内容分发流程之前，CDN 系统对内容进行的一系列处理过程。这个预处理过程的目的有几个：一是为全网内容管理提供依据，比如对内容进行全网唯一标识，对内容基础信息进行记录等；二是为提高 CDN 服务效率或降低系统成本提供手段，比如内容切片；三是为满足业务要求提供能力，比如对同一内容进行多种码率的转换以满足动态带宽自适应或三屏互动业务要求。

下面我们重点讨论视频转码和文件切片两项技术。

##### 1. 视频转码

视频转码（Video Transcoding）是指将已经压缩编码封装完成的视频流转

换成另一个视频码流，以适应不同的网络带宽、不同的终端和不同的用户需求。转码本质上是一个先解码，再编码的过程，因此转换前后的码流可能遵循相同的视频编码标准，也可能不遵循相同的视频编码标准。

视频转码的功能主要包括码率转换、空间分辨率转换、时间分辨率转换和编码格式转换。

**码率转换**不改变编码格式，只是将原始码率转换成新的码率以适合网络传送要求，一般是将高码率视频转换为低码率视频。码率转换主要包括四种方法：第一种是截断高频 DCT 分量，通过丢弃部分 DCT 系数，使得各块码流降低到预期码率。第二种是选择合适的量化步长，对解码后频域系数再进行量化来降低码率。第三种方法是利用提取的运动矢量和编码模式对图像重新编码，避免重做运动估计和编码模式选择。第四种方法是从视频流中提取相应的编码信息，只做一次 DCT、IDCT，采用运动矢量优化方法提取运动估计的精确性。前两种方法属于开环系统，不需要通过 DCT/IDCT 进行重建图像，虽然很大程度上降低了计算复杂度，提高了转码速度，但图像质量比较差；后两种方法属于闭环系统，对解码图像进行了重建，反馈了参考帧图像，重新计算了各宏块的残差，计算复杂度高，但获得的图像质量比较好。

**空间分辨率转换**指通过在“全解全编”架构中添加采样模块，利用下采样算法和运动矢量的映射算法以及伸缩算法来降低视频码流的空间分辨率。下采样算法对原有采样方法进行更改，比如像素平均、滤波采样等，缩小图像的空间尺寸。运动矢量的映射算法和伸缩算法指利用运动矢量的等比例缩放进行视频图像压缩，当空间分辨率降低后，一个宏块会对应原来的多个宏块，采用一定方法来计算合适的运动矢量作为新宏块的运动矢量，并将所得的运动矢量除以分辨率的压缩比，以获得低分辨率图像下最终的运动矢量。具体算法可以采用平均值法、中值法或任意选取一个矢量作为当前宏块的运动矢量。

**时间分辨率转换**指通过降低视频序列的帧率，降低对解码设备处理能力的要求。降低帧率并不是简单的丢弃帧，有时需要利用丢弃帧的运动信息重新合成运动矢量。时间分辨率转换的方法主要包括丢帧、帧类型转换和运动矢量合成算法。采用丢帧方式时应该选择合适的跳帧策略，避免对 I 帧产生影响。而

帧类型转换主要包括B帧到P帧的转码，即把原来前向、后向、双向运动矢量全都转换成前向运动矢量。运动矢量合成算法是指视频帧之间的运动矢量依赖关系因为丢帧以后产生了中断，通过利用丢弃帧中的运动信息，生成新的参考帧运动矢量。

**编码格式转换**是指将原始视频内容所采用的编码格式转换成终端能够解码播放的格式。这是典型的先解码再编码的过程。视频编码格式主要包括H.264、MPEG-4、MPEG-2、VC-1、REAL、H.263、WMV等，现在很多情况下是把他其他编码格式转换成H.264格式，这样就需要考虑H.264的多参考帧、多编码模式以及1/4像素运动估计等特点。

## 2. 文件切片

文件切片是指按照一定的规则把一个完整的文件切成大小一致的若干个小文件，这并不是流媒体CDN的新技术，在几年前的P2P系统中就采取了文件切片后分散在不同位置的方式，这样有利于某个peer同时从多个位置获取内容。近年来，由于流媒体CDN需要提供的内容体积越来越大，传统整片存储带来的成本消耗超出了CDN服务商的承受范围，所以大家开始重视文件切片。经过切片处理以后，在CDN的各个Cache只需存储某些切片，其余的切片可以从其他Cache获取，这样，整个系统内重复复制的内容数量就降低了。

文件切片的另一个目的是，使边缘Cache能够支持自适应码率业务。自适应码率的原理是，同一个视频节目预先压缩从低到高几种码率的文件，并采用统一策略对这几种码率的文件进行切片。比如按照统一的时间长度，或者统一的字节数等。在具体提供流服务时，Cache会与终端先协商一个初始码率，以这个初始码率进行传送。传送过程中，终端和Cache都会不断探测流传送的速度，当发现当前码率低于可用带宽时，就切换到高码率文件进行传送，反之则切换到低码率文件。这种码率切换的同时又要保证用户观看的视频流不中断，其基础就是不同码率的文件预先都进行了切片，码率切换过程是在两个独立切片之间完成的，对终端来说，每个切片都是一个完整的小文件，所以不会感觉到中断。SVC文件切片分层编码示意图如图6-28所示。

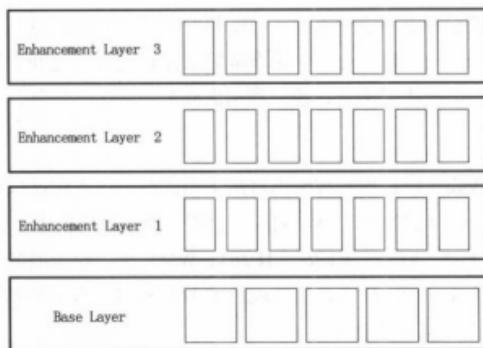


图 6-28 SVC 文件切片分层编码

对文件进行切片后，每个切片都被 CDN 系统看成一个独立的文件，需要针对每个切片进行内容管理。

#### 6.4.6 防盗链机制和实现

网站的防盗链指通过技术手段，直接在本网站页面上向最终用户提供其他网站的内容，从而免费获取其他网站的优质资源。防盗链绕过了该网站通过该内容获取正当利益的页面，比如广告页，因而侵犯了被盗链网站的正常商业利益。防盗链行为对被利用了资源的网站是不公平的，一方面损害被盗链网站的商业利益，另一方面也加重了被盗链网站服务器的负担。因此，防盗链是网站必须考虑的问题。

容易被盗链的内容很多，比如视频、下载资源、MP3、图片等。之所以在流媒体 CDN 系统这一章中将防盗链技术加以重点讲解，是因为视频内容被盗链的影响最大。合法提供视频内容，网站必须付出高昂的版权采购成本，而且提供流媒体服务对服务器资源消耗比普通网页内容大得多。

从技术上来说，防盗链有很多种实现方式，这里列举其中比较常用的 8 种。

### 防盗链方法 1：利用 HTTP Referer 字段

这是最早的，最常见的防盗链方法，常用于视频、MP3、图片这类容易被HTML嵌入到其他网页中的资源。网站服务器通过判断浏览器请求内容的HTTP数据包头的 **Referer** 字段值，能够获得浏览器所处的页面地址，这样就能知道用户此时是在本网站的页面上，还是在其他不合法的页面上。比如正常情况下，当用户浏览 <http://ent.sina.com.cn/bn/movie/more.html> 时单击一个链接去获取 <http://ent.sina.com.cn/movie1.flv> 文件时，浏览器在发出请求 movie1.flv 资源时会在 **referer** 字段中携带浏览器所处的页面地址（即 <http://ent.sina.com.cn/bn/movie/more.html>），所以当网站服务器收到 movie1.flv 资源请求的时候，先获取和判断 HTTP 的 **referer** 字段值，如果是从域名 [ent.sina.com.cn](http://ent.sina.com.cn) 过来的，则认为是合法的连接请求，否则就返回一个错误的提示信息。

### 防盗链方法 2：利用登录验证信息

这个方法适用于需要用户登录认证才能浏览的网站，比如论坛、社区、会员制网站等。当浏览器请求网站上的一个资源时，网站服务器会先判断此请求是否已经通过了身份验证（比如在 **asp.net** 里用 **session** 或 **form** 验证来记录登录状态），如果发现用户尚未登录则返回一个错误提示信息。

不过因为登录状态依赖于会话 ID，而会话 ID 往往储存于 HTTP 请求的 **cookie** 字段里，有些方案中会将这个 **session ID** 通过算法植入 URL 中，便于进行快速查询。

### 防盗链方法 3：使用 cookie 携带动态验证信息

方法是网站在页面里产生一个动态的 **cookie**，服务器在处理资源请求时先判断请求中是否携带了正确的动态 **cookie**，如果没有则返回错误提示信息。至于这个动态信息的产生方法有很多，只要服务器能逆向判断其合法性即可。比如对 **asp.net** 的网页程序，直接往 **session** 里存一个字符串或数字，然后在处理资源请求时先检查 **session** 里是否存在这个字符串或数字即可。

#### 防盗链方法 4：使用 POST 下载

客户端浏览器请求资源都使用的是 HTTP 的 GET 方法，服务器使用 POST 方法向客户端返回数据。可以将下载链接换成一个表单（Form）和一个按钮（Submit），将待下载的文件的名称或 ID 放到表单的一个隐藏文本框（Input）里，当用户单击提交按钮时，服务器先判断请求是否为 POST 方式，如果是则读取目标资源的二进制数据并写入响应对象（在 asp.net 里是 response.BinaryWrite 方法）。

#### 防盗链方法 5：使用图形验证码

相信读者都遇到过这种方法，就是需要用户输入图形验证码中的数字或者字母之类的，这个方法可以保证资源的使用者是“人”，而不是其他网站的服务器或下载工具。图形验证码的实现方法在很多资料中都有介绍，这里就不再重复了。这个方法的缺点是会让用户感到很麻烦。

#### 防盗链方法 6：使用动态密钥

当用户单击一个资源链接时，服务器先计算一个临时的 Key，然后记录这个 Key 以及它所对应的资源 ID 或文件名，再通过一定的算法或规则为资源生成一个新的 URL，这个新 URL 里需要包含这个 Key。当浏览器向服务器请求这个资源时，程序先检测 URL 里是否有合法的 Key 存在，如果存在则返回对应的资源数据。

#### 防盗链方法 7：在内容中插入数据

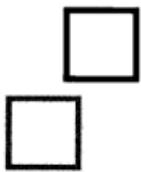
这种方法是指在内容的某个地方插入一些随机的字节，比如 mp3 的 tag 区中，rar/zip 的备注区中，这样整个文件的哈希值（即散列值、指纹值）就会发生改变，服务器在进行哈希校验时，等于同时做了合法性检查。同时配合方法 6，可以达到较好的防盗链的效果。

#### 防盗链方法 8：打包下载

这个方法跟方法 7 类似，只是方法 7 是在原始文件里修改，而本方法是在原始文件基础上进一步封装，同样使得资源的哈希值改变。但是这种方法不适

用于在线流媒体业务，只能用于下载业务。

使用CDN进行内容分发的网站，就会要求CDN的流媒体服务器实现上述某些防盗链技术，帮助维护网站的正当利益。有些方法是适合CDN实现的，比如利用HTTP Referrer字段的方法，而有些方法不太适合CDN实现，比如利用登录验证信息的方法、动态图形验证码的方法等。具体应用的时候采用哪种方式，通常由网站决定，CDN方面配合实现。因此，对客户防盗链机制的适应性开发和调试是CDN承接每个使用者之前必须进行的工作。



## 第7章 动态内容加速 服务的实现

7.1 动态内容加速技术

7.2 应用加速技术



在本书第3章到第6章，我们学习了Web加速和流媒体加速相关的技术，总的说来，这些都属于静态内容加速的范畴。随着Web 2.0的兴起，产生了动态网页、个性化内容、电子交易数据等内容的加速，这些就涉及了动态内容加速技术。在第2章CDN技术概述中，曾介绍过典型的Web系统由表现层、业务逻辑层、数据访问层这三层组成。静态内容的加速，都是对于表现层的加速，而对于动态页面等内容的加速，则要涉及逻辑层和数据访问层的加速技术，本章7.1节首先来了解这两层的加速技术。

近二十年来，互联网和信息技术得到了飞速发展，电子商务、电子政务已渗透企事业单位的每个角落，网络办公自动化已成为现代办公的主流方式，一个公司内部处理的信息也在快速增长。同时，随着经济增长和全球化的发展，越来越多的企业在不同的地点开设分支机构，对企业的办公协同提出更高的要求，原先在局域网范围使用的电子化流程需要在广域网范围进行传送，对整个办公系统，特别是广域网的性能、安全性等提出更高的要求。另外，不断增加的IT系统在各分支机构的部署也带来了基础设施和维护成本的增加。在这一背景下，2006年前后，业界提出了应用交付/加速技术来解决这些问题，本章7.2节将对应用加速技术进行介绍。

## 7.1 动态内容加速技术

当前Web系统提供的内容主要分为静态和动态两类。其中，静态内容由纯粹的HTML文件所编写的网页所展现，改变静态内容需要重新编辑HTML文件，因此内容相对固定而且更新比较麻烦，在互联网发展早期的网站中多被采用。动态内容的提供则不仅仅是HTML页面的设计和编辑，它还需要有后台数据库、应用逻辑程序的支持，以实现与用户的动态交互。提供动态内容的Web网站能够实现更丰富、更灵活的功能，满足用户定制化的个性需求，已经成为

当前 Web 系统的主流。

为了实现动态内容的交付，典型的 Web 系统的逻辑架构产生了如图 7-1 所示的演变。



图 7-1 典型 Web 系统逻辑架构的演变

如图 7-1 所示，Web 系统由表现层、业务逻辑层、数据访问层三个层次组成，而在当前随着用户对个性化需求的不断增强，此前依赖于数据访问层存在的用户数据越来越受到业界的关注，因此衍生为单独的一层。

### 1. 表现层

表现层是 Web 系统与外部系统的交互界面，这一层通常由 HTTP 服务器组成，负责接收用户的 HTTP 内容访问请求，从文件系统中读取静态文件提供给用户，需要时向中间的应用逻辑层发起请求。在表现层处理的静态文件主要包括如下三项。

(1) 网页中嵌入的 Web 对象：如图片、样式表、Flash 动画、Java 程序、ActiveX 控件等。

(2) 多媒体内容：如视频和音频文件。对于在线播放的视频多媒体内容，为了保证用户端流畅播放，通常采用 HTTP Streaming 协议，将源文件流化预处理以后再向用户传送。

(3) 网页片段：网页片段是指那些功能相对独立的组成网页的内容单元，每个片段都是一个独立的信息实体。例如，一张普通网页可能是由新闻、专题、导航条、广告组成的，这些网页片段可能在网站的每个页面都会出现，所以使用划分“片段”的方法可以提高内容的重用率。这时，表现层还需要完成网页组装，以便将整页面交付给用户。

## 2. 业务逻辑层

业务逻辑层是 Web 系统的核心层，负责处理所有的业务逻辑和动态内容的生成。内容的动态生成通常涉及个性化内容处理、数据处理等工作，因此需要与前端的表现层、后端的数据访问层通信。在业务逻辑层处理的动态内容包括如下三项。

(1) 检索结果：如电子商务网站的个人购物车、商品搜索。

(2) 与呈现无关的网页内容的动态生成：如内容管理系统或基于 XML 的 Web 文件。在这些系统中，内容（即使它的生命周期相对长）由一个动态模板实时生成，有些专业软件是专为这种动态处理而设计的，如 DBMS。

(3) 用户社交内容：如社交网站、博客或论坛。

## 3. 数据访问层

数据访问层位于系统的后端，负责管理 Web 系统的主要信息和数据存储，通常由数据库服务器和存储设备组成。不同的 Web 网站，数据库具有不同的功用。

(1) 电子商务网站：数据访问层存储商品信息、检索商品目录、生成购物清单、管理消费卡等。

(2) 内容管理系统：数据访问层实时提供网页模板和网页资源。

(3) 论坛和社区网站：数据访问层存放文章、评论、邮件等。

### 4. 用户数据层

用户数据层负责存储用户信息数据和关联关系，这些信息用于生成用户个性化内容。用户信息可能有如下几个来源。

(1) 用户主动提供：通常是用户自行在网站页面上注册填写。

(2) 用户行为分析结果：通常从网站日志中经过数据挖掘分析提取。

### 5. CDN 的复制机制

Web 网站借助 CDN 技术能够获得更好的扩展性和高性能，核心在于 CDN 采用的缓存（Caching）和复制（Replication）机制，其中缓存是将最近经常被访问的源服务器拥有的内容复制到边缘服务器上，可被视为具有特定策略的复制。

CDN 的复制机制是指将源 Web 系统逻辑架构的各个层次的相应功用复制到边缘服务器上实现，以缓解源系统的处理压力，主要内容包括如下几项。

(1) Web 系统表现层的复制。通过复制，边缘服务器能够负责静态内容的管理和提供，该方法在传统 CDN 中被广为使用，边缘服务器又被称为代理服务器，通过反向代理加速静态文件的交付。

(2) Web 系统业务逻辑层的复制。通过复制，CDN 被用于改进动态生成内容的交付性能，该方法又被称为边缘计算，即将应用程序和业务组件直接在 CDN 的边缘服务器中计算，从而直接在靠近用户的地方生成动态 Web 内容。

(3) Web 系统数据访问层的复制。通过复制，CDN 边缘服务器能够具备生成动态内容和掌管内容生成数据的能力，而源服务器只负责管理基础架构和存放数据的主版本。

(4) Web 系统用户文件的复制。通过复制, CDN 边缘服务器能够掌控用于生成用户定制化内容的数据。

### 7.1.1 业务逻辑层加速技术: 边缘计算

顾名思义, 边缘计算用于将应用程序、数据和计算能力(服务)从网络中的少量集中点推送到网络的逻辑边缘位置。边缘计算复制并分发在多个分布式网络中的 Web 服务器中保存的信息片段。为了帮助读者理解这种技术, 下面以 Akamai 的 Edge Computing 服务为例, 介绍边缘计算和传统 Web 应用部署模型在提供 J2EE Web 应用时的差异。

传统 J2EE 互联网部署架构如图 7-2 所示, 交付 J2EE Web 应用程序的基础设施架构通常包含一系列组件: 负载均衡器、HTTP Web 服务器、缓存服务器、应用服务器、数据库、监控服务器等。用户通过 URL 直接访问部署在企业或者数据中心的 Web 应用系统, 由应用服务器根据用户请求参数进行动态内容响应。

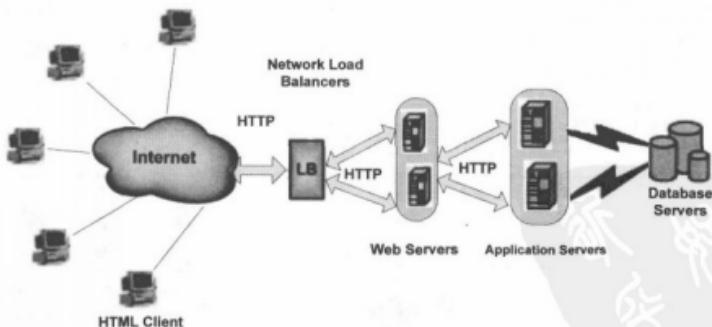


图 7-2 传统 J2EE 互联网部署架构

Akamai 公司 2003 年推出的 Edge Computing 服务是按需分布式计算平台的典型实例。Akamai Edge Computing 平台由分布在全世界多个运营商网络中的成

千上万台服务器组成，这些服务器部署在网络边缘，靠近用户访问点。J2EE 应用程序处理分布在 Edge Computing 整个平台，从而使计算任务可以按需地靠近服务请求用户。这些全球分布的边缘服务器实现工业级的协议来支持简单到页面组装，复杂到 J2EE 事物处理的任务。为了进一步扩展分布式计算能力（在表示层基础上），Akamai 支持两个版本的 J2EE 应用服务器：Tomcat 和 IBM WebSphere。之后 IBM 和 Akamai 将 WebSphere V5.0 服务器集成并部署到 Edge Computing 分布式计算平台上面，而 Akamai Edge Computing 计算服务的企业客户基于按需使用互联网计算资源的模式运行 J2EE Web 应用服务。

图 7-3 给出了典型的 Akamai 边缘计算部署模型，包括：用户（使用浏览器）、企业 J2EE 应用系统（运行业务逻辑、原有系统、数据库等）、分布式网络服务器（Edge Computing 平台）运行支持 J2EE 应用编程模型的 WebSphere 或者 Tomcat 应用服务器。

从图 7-3 可以看出，用户在使用 Akamai 边缘计算进行动态内容分发服务时，不仅将应用程序（Business Logic）复制到了边缘计算平台中，同时将企业数据以及数据访问程序也复制到了边缘计算平台中，进而在利用边缘计算平台提供分发服务时减少从企业读取数据的次数和读取数据的大小，降低系统响应延时。

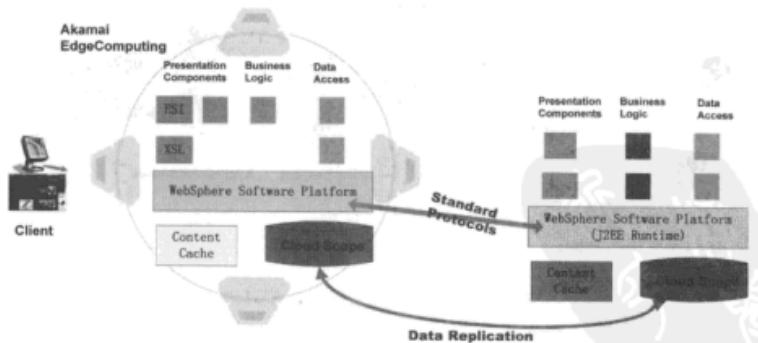


图 7-3 Akamai 边缘计算部署模型

Akamai 边缘计算仍然遵循 J2EE 标准部署模型，不需要使用任何自有 API 接口，因此使用边缘计算只是改变了原有 J2EE 程序的部署模型，而对 J2EE 编程模型没有影响，仍可以依赖标准 J2EE 开发工具和最佳实践来开发应用程序。但是部署在边缘计算平台上的应用程序必须设计成两级软件架构，即一端程序运行在边缘计算平台当中，另一端程序运行在企业端部署平台当中。

经过设计、开发、测试，计算边缘子程序最终被上传和部署到 Akamai 边缘计算平台中运行。Akamai 提供了两种边缘计算程序部署接口：①通过浏览器访问 Akamai 企业门户，进行应用程序部署；②通过基于 SOAP Web Service API 接口编写调用程序，实现边缘计算程序的上传和激活。

### **7.1.2 数据访问层加速技术：数据库复制**

在此前介绍的 CDN 对 Web 系统应用逻辑层的复制中，每台 CDN 边缘服务器都具有完整的应用逻辑处理代码，而在系统后台使用的仍旧是源站点的数据存储，因此边缘服务器上的应用逻辑只能共享集中化的系统数据库。在这种情况下，对于那些瓶颈在后端数据访问层而不是业务逻辑层的 Web 应用，还是难以解决系统的扩展性和性能问题。因此，就需要考虑用 CDN 对应用数据进行复制和管理，利用边缘服务器代替源站 Web 系统的后台数据访问层中的数据库系统，及时响应业务逻辑层提出的数据查询需求。

当前，在边缘服务器上复制 Web 系统数据访问层的方法主要分为整体复制和部分复制两类方法，其中部分复制又可以分为不感知内容的缓存（Content-Blind Caching）和感知内容的缓存（Content-Aware Caching）两种技术。

#### **1. 不感知内容的缓存**

不感知内容的缓存是将源站 Web 系统中最经常被应用逻辑查询的数据复制到边缘服务器的存储空间中，因此边缘服务器中保存的总是后台数据访问层

中此前被查询过的内容。在这种情形下，边缘服务器可以在本地对那些相同或者类似的查询操作进行处理，从而缓解后台服务器的负载压力，提高系统的整体性能。

这种不缓存数据内容，而只是直接缓存查询结果的方法被广泛用于复制 Web 系统的数据访问层，比较知名的如 *GlobeCGC*。业界新近提出了一类通过使用查询结果缓存实现对后台数据访问层进行动态复制的技术，能够有效改善系统的扩展性。例如，*DotSlash* 框架中的 *QCache* 模块，通过在多个协作 Web 站点间签订协议，使得系统能够根据实际需要启动临时的分布式的查询结果缓存，用以缓解部分站点流量猛增导致的超载压力。

对这种基于查询结果的缓存技术，影响效能的关键在于缓存的命中率。为了使得应用逻辑查询的结果能够经常被后续查询利用，就要采用优化的查询匹配引擎。例如，一个新来的查询所需的数据可能由多个此前已有并被缓存的查询结果组合而成。最早在 *DBproxy* 系统中提出了这种加强的查询支持。另外，*DBCache* 系统支持对数据库表的缓存，可以在边缘节点上缓存后端集中数据库服务器上的数据库表的子集，以利于查询的快速实现。

另一个重要要求是，必须保证被缓存数据的一致性。因为传统的基于 TTL 的方法并不适用于所有的 Web 应用，因此人们提出了一些专用的一致性强化机制。例如，通过多播消息系统将数据无效信息发送给每个缓存设备，或者使用散列函数保证过期的数据不会再被缓存和提供给应用逻辑。

当前，已有众多的厂商产品能够支持在边缘服务器上提供查询结果缓存，如 BEA 的 *WebLogic* 和 IBM 的 *WebSphere*。其中，IBM 的 *WebSphere* 采用实体查询表（Materialized Query Table，MQT）技术实现查询结果缓存，每张实体查询表保存的是预先计算出来的具体查询结果，这个结果可能涵盖了对后端数据库上一张表或者多张表的实施查询后获得的结果。在实体查询表被创建并分发到各个节点后，只要后续的查询能够被实体查询表全部或者部分匹配，那么该查询都可以由实体查询表提供结果。*BEA WebLogic* 中的 EJB 采用的也是

类似方法。

应该指出的是，即使使用了数据一致性强化机制，CDN 边缘服务器上仍然难免出现数据更新不及时的情况，这是由 CDN 节点地理上广泛分布的特性决定的。在大部分数据访问请求为“读”操作的场景中，这种情况不会构成大的障碍，但对于以支付操作为代表的一类交互应用场景中，数据库的更新会非常频繁。在这类场景下，就不能采用查询结果缓存技术了。说得简单点，查询结果缓存技术只适用于那些重复发送相同查询要求的应用，而其他一些场景中更适合于采用在边缘服务器上直接复制部分或者全部后台数据访问层中的数据。

## 2. 感知内容的缓存

感知内容的缓存，是指基于一定的使用模式、网络状态等信息，对源站数据存储内容进行选择，并将其中的一部分复制到边缘服务器的存储空间中。在边缘服务器中可以运行自己的数据库服务器，并保存源站数据库中的部分视图。比较典型的部分数据复制方法是根据数据访问模式（例如数据之间的关联关系），将后端源站数据库中的相关数据推向边缘服务器。复制操作的目的是改进最终用户可感知的响应时间，因此需要在数据副本的替换算法（例如 HotZone 算法）中充分考虑网络延迟问题。

广为人知的 GlobeDB 系统提供了一个非常好的复制机制，它采用基于数据分区的数据库部分复制技术，有效减少了数据更新带来的网络流量。但是，这个解决方案必须依赖于一个专用服务器，它能够在全局上掌控数据库的管理信息以支持复杂操作，例如跨越数据分区的查询，但也因此导致它将成为系统中新的流量瓶颈。而后来的 GlobeTB 系统则全面改进了 GlobeDB 方法，它不仅以降低延迟为目标，同时还关注增加被复制数据库的吞吐率，其关键点在于以分布式数据库代替单台集中部署的数据库，降低了产生性能瓶颈的风险。

同样，部分数据库复制技术也必须考虑数据一致性问题。解决这一问题的关键在于及时处理源站侧发生的数据变化，即有更新、删除或者插入等查询操

作时发生的情况。边缘服务器应该能够将数据更新操作与只读操作相隔离，将数据更新操作发送至源站数据库上，而把其他一些查询操作转发到本地数据库拷贝上。

目前也有众多厂商产品能够支持数据库的部分复制，例如 MySQL 的 DBMS 就支持在多个数据库副本间进行数据划分，而类似的特征在 IBM DB2 和 Oracle 产品中也有体现。总体而言，部分复制方法多数用于对资源进行局部复制的场景，需要一个全局的中央管理单元负责处理和分发各个数据分区的请求和操作。然而中央管理单元的存在将限制系统的可扩展能力，所以这种方法并不太适用于 CDN 这样的分布式场景，当前大部分商业化产品中使用的还是基于查询结果的不感知内容的缓存技术。

### 3. 数据库整体复制

数据库的整体复制是指在多台边缘服务器上同时保存和后端数据库完全一致的拷贝，同时确保多拷贝之间的一致性。在整体复制技术中，边缘节点可以自行生成完整的动态内容。但也正因为过于依赖边缘节点的本地数据，各节点之间的数据一致性问题就显得格外重要，因此也得到业界长期关注。

确保数据一致性，关键在于及时将写操作带来的更新传播给各个副本。传播方法分为积极（eager）的和懒惰（lazy）的两种：积极的方法是在数据更新提交之前进行多个副本间的协作，而懒惰的方法则是在更新被提交之后再传播给各个副本。在积极的方法中，所有副本就好像一个数据库一样，但这在性能、扩展性等方面存在严重缺陷，从而很难被实现。相反，懒惰的方法在性能和扩展性方面具有优势，因此广泛应用于商业产品。懒惰的方法需要解决的主要问题是更新事件之间的冲突，故务必引入冲突解决机制。

Web 系统通常是在集中的源服务器上存放主拷贝，而在各个边缘服务器上存放相应的拷贝副本。只读操作可以直接在边缘服务器上执行，通过访问本地数据库拷贝获取数据即可。而对于更新操作，所有的数据库访问请求将被重定

向到主拷贝上，由主拷贝将更新传播给各个拷贝副本。这么做的最大挑战在于边缘服务器必须了解应用逻辑，因为它需要知道请求中是一个包含了写操作的更新操作还是一个单纯的只读操作。

在 CDN 中，数据库副本通常分布在地域广阔的广域网环境中，如果某个 Web 应用产生了大量数据更新操作，为了保障数据一致性，就会有数量庞大的数据副本在广域网内传递，造成流量剧增，业界也提出了采用分组通信的方法来缓解这种问题。但总体而言，如何提升广域网环境下数据库复制的性能和扩展性是业界公认的难点，仍然存在着非常广阔的研究空间。

### 7.1.3 用户数据层加速技术：用户数据复制

用户数据是 Web 系统实现个性化、动态应用以及跟踪用户行为特征的基础。在用户访问 Web 站点时，用户发出的请求和相关参数（例如访问内容标识、cookie、会话对象信息等）将被传递给业务逻辑层，业务逻辑层判别哪些数据需要从数据库、文件系统或者其他外部数据源中获取，从而向后端数据访问层发出相应的查询请求。接下来，业务逻辑层利用这些结果生成 Web 页面并返回给表现层，供用户浏览。从上面这个流程可以看出，Web 系统的很大一部分工作就是根据用户请求实现应用和内容的动态生成，因此，如果能够在边缘服务器上复制和缓存用户数据，就能够极大提高内容生成速度。

在系统实现中，用户数据层是依托于数据访问层存在的，它通常需要利用数据访问层提供的存储资源。与数据库复制类似，用户数据层的复制也是在源站数据库与边缘服务器之间建立数据关联关系，能够及时检测用户数据的变化情况，并更新边缘服务器上相应的数据。但是，对于 CDN 系统而言，用户数据与其他数据在访问模式上存在差异，由此导致复制技术实现时的特殊性。

（1）一般情况下，用户只和一台边缘服务器进行交互，因此在该用户会话存在期间通常只有一台边缘服务器需要访问相应用户的描述文件，这种访问模

式对于用户数据相关的一致性保持和复制策略设定具有非常关键的影响。在实际场景中，根据用户访问需求的不同，一个完整的用户数据集可以被划分并分布到多台边缘服务器上，而边缘服务器之间并不需要进行用户数据的复制，所以用户数据相关的一致性问题就仅限于确保边缘服务器上保存的用户数据与后端源数据库服务器上的用户数据保持一致。可以采用前文介绍的不感知内容的缓存或者感知内容的缓存等数据库层复制技术进行用户数据管理。

(2) 如果在实际应用中，用户有可能在连续的会话过程中在多台边缘服务器之间迁移，那么系统就有必要确保用户的数据文件能够随着用户的迁移而迁移。但是，之前的大多数用于系统后台数据访问层的复制策略中并没有对这类行为进行特别的优化，只有最近提出了少部分相关研究。例如，CONCA 系统为了满足用户移动性需求提供了对数据随用户而移动的支持，相关的技术在 Tuxedo 系统的用户个性化数据处理中也有应用和扩展。

除了用户数据的访问模式，用户数据的记录形式也会对其加速方法产生非常大的影响。很多 Web 站点为了提供比较灵活的页面格式和布局，通常使用框架集 (frameset) 结构记录用户的定制信息。例如，在一个用户定制索引页的框架集标记下标明多个框架 (frame) 并引入该用户所需的页面链接字符串。当浏览器接收到用户定制的索引页后，将对其进行解析并自动对框架集中各个框架的关联页面发出访问请求，在用户实际点击链接之前就能够完成页面内容的准备。针对这一形式记录的用户数据，边缘服务器可以将用户定制索引页及其框架集中定义的所有关联页面进行缓存。除此以外，还有一类更加灵活的基于动态组装的用户数据记录方法，是在接收到用户请求后再组装用户请求的定制页面内容。在这种情况下，如前所言的几个关联页面可以被复制和保存在边缘服务器上，但是因为不存在特定的索引页，所以边缘服务器在收到专门的页面定制消息后，需要负责将用户所需的相关内容动态地组装到一个页面上并提供给用户，而不仅只是具备页面内容的缓存能力。

用户数据层复制考虑的另外两个关键问题是：安全保密和隐私保护。Web

系统用户数据的获得主要有两个渠道：第一个是通过 Web 页面表单由用户显式填写并提交，第二个是从 Web 系统的日志文件、cookie、用户点击历史等内容中挖掘出用户个人信息。第二个渠道对用户是透明的，用户不会感觉到它的存在。为了避免这些后台手段被用于进行一些未经许可的用户信息采集和分析行为，导致用户隐私泄露，业界提出了很多方法，例如 P3P（Platform for Privacy Preferences）。每个遵循 P3P 标准的 Web 站点都需要提供一个基于 XML 编码的文件，用于描述该站点上哪些数据可以被采集、如何进行管理、保存在何处等信息，遵循 P3P 标准意味着站点必须为每份用户数据副本都提供足够的隐私级别保障。

## 7.2 应用加速技术

### 7.2.1 应用加速技术概述

在过去的三十年里，互联网经历了一个由 Host 到 Terminal 这种高度集中的模型向大量分布式的 Client/Server 模型演变的过程。与此同时，全球化也已成为一个不可阻挡的力量，各种各样的组织和企业变得越来越分散，管理者、决策者、工程师和其他专业人士等都可能散布在全国甚至世界各地的分支办公室里，而他们的彼此沟通和工作要求就像在一个办公室一样。这些企业都需要部署大量的 IT 基础设施以实现远程协同办公。随着这种分布式网络节点的增加，整个系统复杂度不断提高，给企业的信息化部门带来沉重的维护服务工作量。这种维护工作面临着两个挑战：一是分支点的信息控制所带来的信息安全和数据一致性问题；二是应用系统的性能问题。

为了应对挑战，很多企业都计划将其信息化基础设施进行再中心化：首先将信息存储和应用控制点集中，以此降低安全风险；其次通过 IT 基础设施集中，

进一步提高维护效率和降低成本。这时所有的 CIO 都要面临一个挑战：就是一个集中化的基础设施，需要高速广域网连接和高性能保障。简而言之，就是希望“广域网像局域网一样”工作。这就是应用加速（交付）技术提出的背景。

应用加速网络（Application Delivery Networking，简称 ADN），利用网络优化和加速设备，确保客户业务应用能够快速、安全、可靠地交付给内部员工和外部用户群。可见，应用交付的宗旨是保证企业关键业务的可靠性、可用性与安全性，其应用目标可归纳为如下几点：

- 提高用户和 IT 部门的生产力。
- 对各种应用进行加速。
- 节省广域网带宽使用。
- 减少远程机构对 IT 设施的需求。

应用加速网络实际上是传统的网络负载均衡的升级和扩展，综合使用了负载平衡、TCP 优化管理、链接管理、SSL VPN、压缩优化、智能网络地址转换、高级路由、智能端口镜像等各种技术。由于篇幅所限，下面重点介绍广域网加速技术和 SSL 加速技术。

## 7.2.2 广域网加速技术

### 1. 广域网性能问题分析

衡量广域网传输性能和质量的主要指标是带宽和时延。与局域网相比，通常广域网的带宽更低，时延更大。由于目前很多办公应用，如 Windows CIFS 文件共享、企业资源计划（ERP）系统、企业数据备份系统、数据库等，最初都是针对局域网设计的，所以这些应用在广域网上使用时会影响执行效果。如何在现有的广域网条件下进行应用的加速呢？首先要明白广域网性能问题的产生原因和对应用的影响。

### 原因 1：带宽问题

这个原因比较好理解，广域网的带宽成本要远大于局域网，而资源也不会像局域网那么充足。当带宽不足时，大文件或数据的传送会受到影响，如发送带有大附件的邮件、FTP 文件的上传和下载、一些文件共享操作等。

### 原因 2：传输时延问题

广域网的传输时延在电子邮件和下载的时代并不受重视，但在实际的网络工作环境中，时延对整个链路的数据吞吐量影响比带宽影响更大。传输时延是由多方面原因造成的，下面来逐一进行分析。

首先，物理距离带来一定的时延。数据在光纤中以光速传送，途经多个路由设备和交换设备，源和目的设备之间的物理距离越长，其传输所需的时间也越长。

其次，TCP 协议的工作机制也带来一些延迟。这其中的第一类 TCP 延迟是由 TCP 协议的端到端应答机制造成的。在 TCP 协议中，从一端到对端（比如在服务器和客户端之间）所能够传输的数据量受 TCP 窗口大小限制。当该窗口已满，发送方就无法发送更多的数据，直到接收方确认已经接收了窗口中的部分数据。如果数据报窗口太小的话，势必会限制数据从一方传送到另一方并进行应答的速率，进而影响到整条链路的数据吞吐能力。从理论上说，这个瓶颈出现的几率很小，因为已经有很好的机制能允许 TCP 协议使用足够大的数据报窗口，而且现在流行的操作系统也都实现了这些机制。然而，客户端和服务器上的 TCP 协议的默认设置通常更适用于局域网而不是广域网。

图 7-4 表明了一条 TCP 链路 T1 (1.544Mb/s) 在最大 64KB 窗口下进行 TCP 连接时有效数据吞吐能力随延迟变化的情况。在低延迟的情况下，T1 线路能达到带宽允许的最高数据吞吐量，但是当数据延迟超过 40ms 时，网络往返延迟 (RTT) 对吞吐量的影响就要超过带宽所带来的影响了。

图 7-5 所示的情况和图 7-4 类似，不过增加了一条 45Mb/s 链路 T3 以供比

较。在该图的尺度下, T1 链路 (1.544Mb/s 宽) 的数据吞吐能力情况基本稳定, 而 T3 链路的吞吐能力则很快就下降到与 T1 线路相差无几的水平上。在延迟大于等于 40ms 及 TCP 窗口较小的情况下, T3 链路的数据传输能力并不比 T1 更具优势。

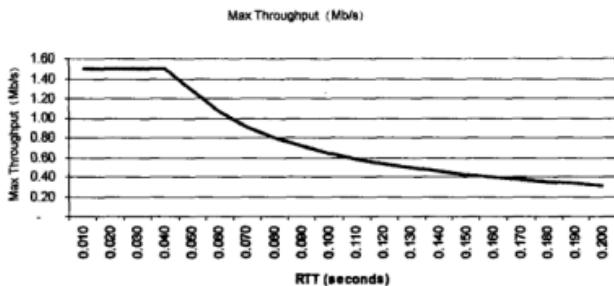


图 7-4 1.544Mb/s 带宽下 TCP 协议随网络延迟最大数据吞吐曲线图

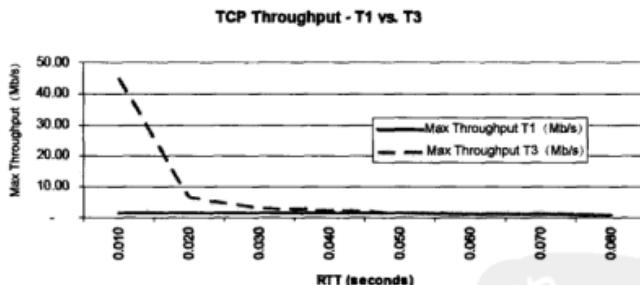


图 7-5 45Mb/s 和 1.544Mb/s 带宽下 TCP 协议随网络延迟变化数据吞吐曲线图

此外, TCP 协议的慢启动和拥塞控制行为也可能引起传输时延。按照 TCP 协议的工作机制, 如果数据传送在一段时间内保持正常, TCP 窗口大小会逐渐变大, 而一旦发生传输失败其窗口大小会立即缩小。如果网络同时具有高带宽和高延迟特性, 这种行为就会导致带宽浪费, 延长数据的传输时间。简而言之, 该问

题的原因在于 TCP 并不是总能利用最大窗口进行传输。

### 原因 3：应用协议工作效率低问题

上面提到，当受到 TCP 数据窗口大小和数据消息响应的限制时，实际带宽再充裕也起不到什么帮助作用。类似的，如果一个应用在应用层就受到应用消息大小和数据消息响应确认机制的限制时，不管带宽有多充裕，该应用协议数据传输效率都不会高。

TCP 协议有一个问题，就是其协议本身比较“啰唆”。在每一个 TCP 传输过程启动时都需要三次握手确认的过程，如果应用协议最初即被设计用于广域网环境，例如 HTTP 和 FTP，那么一般不会碰到这类问题。正如第 3 章缓存部分介绍过的，HTTP 1.1 采用了效率更高的持续连接机制，即客户端和服务器端建立 TCP 连接后，后续相关联的 HTTP 请求可以重复利用已经建立起来的 TCP 连接，不仅整个 Web 页面（包括基本的 HTML 文件和其他对象）可以使用这个持续的 TCP 连接来完成 HTTP 请求和响应，甚至同一个服务器内的多个 Web 页面也可以通过同一个持续 TCP 连接来请求和响应。这样，可以有效提高链路利用率。

但是如果应用协议最初是被设计用于局域网的，通常在广域网上就会面临严重问题。比如，在微软的 Windows 上通过 CIFS 协议进行文件共享时，这类协议中有比较多的握手、确认等机制，在正式数据传输以前，可能有很多次应用的同步和协商，这就导致了数据传输的延迟。

图 7-6 显示了 T1 链路上采用 CIFS 协议和 TCP 协议时的数据吞吐能力受数据延迟影响时的变化情况。当然，随着延迟的增加，两者的吞吐能力都会快速下降。但应该注意到，这时 CIFS 吞吐能力下降是引起网络传输速率下降的主要瓶颈。对于在广域网上有更好表现的应用协议（如 HTTP、FTP）来说，TCP 延迟可能是个主要问题；而对于 CIFS 协议之上的文件共享应用来说，不管 TCP 优化得有多好或带宽有多充裕，都不足以克服高延迟时间所带来的直接影响。

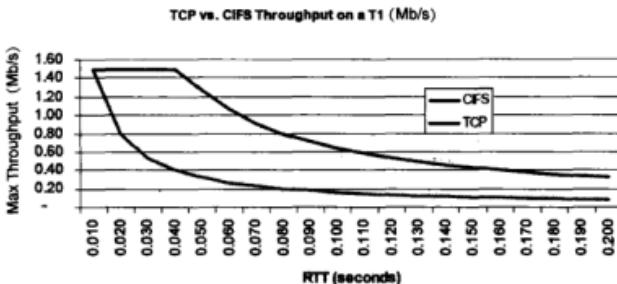


图 7-6 1.544Mb/s 带宽下 CIFS 和 TCP 协议在网络时延下数据吞吐曲线图

## 2. 广域网加速的关键技术

上面分析了造成广域网数据传输性能问题的原因，下面针对这些原因，看看广域网加速技术是如何解决的。

首先，针对网络带宽不足的问题，通常有两种思路：一是加大 IT 投入，对带宽进行扩容和升级；二是看如何减少跨越广域网的数据传输量，这里主要涉及的技术就是数据压缩和缓存技术。其次，针对广域网时延问题，目前最常用的办法是在广域网的两端成对部署设备，通过对 TCP 协议进行优化，实现整个传输过程的优化。最后，针对应用协议低效的问题，一般是从应用层优化本身来做，即针对应用协议进行传输的改进，比如采用通过预测客户行为做一些数据包的提前发送等技术，同时缓存的应用也是一个手段。表 7-1 概括了在不同的网络层次中可以采用的广域网加速优化技术及其优化原理。

表 7-1 广域网加速优化技术及其优化原理

| 针对层次      | 优化技术       | 优化原理  |
|-----------|------------|---|
| 传输<br>发起端 | 原始数据<br>优化 | 通过压缩、重复数据删除和字典等技术，可节省绝大多数传输数据量，节约带宽，提高服务器性能 |
|           | 数据缓存<br>技术 | 将类 HTTP 的业务、图片、文字等缓存在本地，只传输动态内容，减少带宽占用      |

续表

| 针对层次         | 优化技术         | 优化原理   |
|--------------|--------------|--|
| 物理层<br>(硬件)  | 提升设备性能       | 基于现有的 TCP/IP，通过硬件方式提高性能，提高大量 TCP 并发连接和会话重组等处理能力                    |
| 网络层 (IP)     | QoS 和流量控制    | 通过协议识别，实现在同一端口中不同应用的真正区分，进而通过分流实现时延敏感应用的带宽保障                       |
| 传输层<br>(TCP) | 代理设备         | 在传输两端各架设代理设备，所有的响应报文都在本地完成，只有真正发起请求时才通过链路，相当于同时在服务器和客户端进行协议欺骗      |
|              | TCP 协议优化     | 通过在广域网两端部署专用设备，在不影响基本传输情况下，通过各种手段对 TCP 窗口、响应、启动等机制进行改进，从而提高协议机制的效率 |
| 应用层          | 应用代理<br>(缓存) | 将常用的应用程序缓存在本地并配置好，用户可不用在本地等待类似于认证等会话过程，而是直接开始下一个应用，实现流水作业          |

下面对广域网加速的几个关键技术进行简要介绍。

### 数据层面的传输优化技术

通常，广域网加速设备都是在广域网的两端成对部署的，且对于用户是透明的。广域网加速的工作原理主要包含数据切片、数据索引压缩、数据缓存、数据重构与恢复等几个重要步骤。

图 7-7 所示是一个典型的数据层传输优化的工作流程。

- (1) 数据请求从客户机直接发往服务器。
- (2) 服务器响应客户机请求，开始数据发送。
- (3) 在传输过程中，服务器端的广域网加速设备会自动截获数据响应，对数据进行分段。
- (4) 数据分段后，将这些数据与设备中缓存的数据进行比对，只有新的字

节会被发送，且在正式发送前要经过压缩处理。

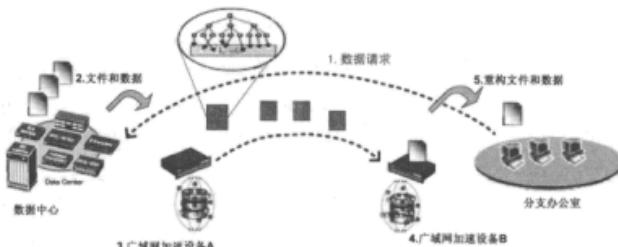


图 7-7 广域网数据层传输优化工作流程

(5) 在被压缩之后经 WAN 进行传送，由广域网加速设备 B 接收。

(6) 设备 B 对接收到的数据进行重构，再送交客户机，整个过程对客户机和服务器是透明的。

目前，这种基于数据分段压缩缓存的传输优化方法的核心思想是借助重复数据删除、传输数据压缩技术来减少广域网传输的数据量。此种方法已经被主流的广域网加速设备厂家广泛采用，其中的典型代表是 Riverbed 公司的 SDR (Scalable Data Referencing) 技术，其中涉及的几个关键技术环节为数据碎片化、层次化数据指针、数据索引压缩和数据缓存。

数据碎片化，顾名思义，就是在应用层将数据分成一个个小的数据块，便于后续的数据比对使用。广域网加速设备在传输数据前会将缓存中的数据与数据切块进行对比，从而找出哪些数据是重复数据，不再发送，哪些数据是新鲜的、需要传输的数据。

数据压缩和指针技术一般是放在一起使用的，在对数据分段后，会对每一段数据生成一个数据指针，对于重复内容，只传输指针。这样的机制使得链路上只传输发生了变化的内容，有效地减少了数据传输量。对于新的内容，压缩后进行传输，又进一步减少了数据传输量。这里，指针的设计和压缩算法是影

响性能的关键。Riverbed 的 SDR 技术采用了层次化的数据指针技术，据称可以做到用 16 字节的指针代表上兆字节的数据；在压缩算法设计上，要求同时兼顾数据压缩比和压缩/解压缩时间，既要有效压缩，又要避免压缩和解压缩带来额外延时。

### TCP 协议优化技术

基于 TCP 协议优化技术的广域网加速设备，其原理是尝试对 TCP 协议进行调优设置，从而使广域网通信数据得到优化，达到更高效的通信。

举个简单例子，服务器端网络加速设备终结了反馈给客户的 10000 个 TCP 数据包，它把数据拆出来，发现原来有很多数据，通过调节 TCP 窗口大小以及重复数据删除技术重新装进新的包，只是把更新过的数据传递给客户端，原来 10000 个 TCP 数据包，现在只要 100 个数据包就可以传输，需要传输的数据只需要在 100 个数据包里面。使得传输数据包的数量大大减少，传输时间也随之降低。

高速 TCP 传输技术是一项比较常见的 TCP 协议优化技术，它允许用户在单个 TCP 连接中最高获得 800Mb/s 的带宽，通过协议的优化避免发生 TCP 传输出现快速衰减、带宽使用率差等情况（如前文所解释的窗口特性）。图 7-8 所示为 TCP 协议自适应拥塞窗口原理图。

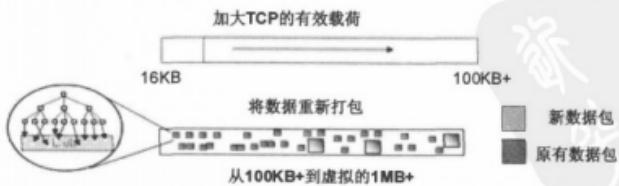


图 7-8 TCP 协议自适应拥塞窗口原理图

高速 TCP 传输包含以下关键技术。

(1) 自适应拥塞窗口：基于网络时延等特征自动调整窗口大小，可以在给定用户网络条件下实现最高带宽传输。

(2) 有限制地快速重传：设置被重发的数据包比其他数据包传输优先级高，从而使等待重传数据包的应用得到快速响应，减少应用程序等待时间。

(3) 连接池：网络加速设备维护一个预先建立好的 TCP 连接“池”，当有数据传输需求时，从连接池中挑选一条可用连接进行传输。这样节省了 TCP 连接建立时间，尤其是对数据量小、时间短暂的连接效果更加明显。

### 应用协议优化技术

图 7-9 所示为低效率应用协议示例图。

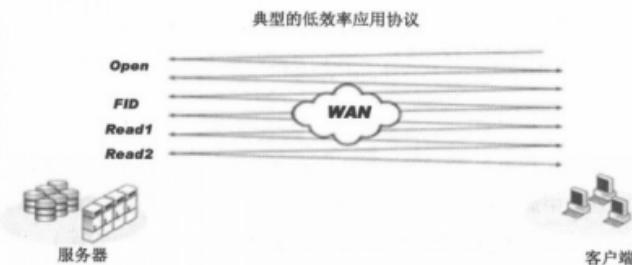


图 7-9 低效率应用协议示例图

应用程序通常具有比 TCP 协议本身更多的往复数据传输需求，比如下面这个 FTP 协议批量文件传输的交互过程：

```
Step1: Client: TELNET 127.0.0.1 21
      Server: 220 Serv-U FTP Server v4.0 for WinSock ready...
Step2: Client: USER ADMIN
      Server: 331 User name okay, need password.
Step3: Client: PASS ****
      Server: 230 User logged in, proceed.
Step4: Client: PASV
```

```

    Server: 227 Entering Passive Mode <127,0,0,1,4,191>
Step5: Client: REST 0
    Server: 350 Restarting at 0. Send STORE or RETRIEVE.
Step6: RETR test.rar
    Server: 150 Opening ASCII mode data connection for test.rar
<94370 bytes>.
    Server: 226 Transfer complete.

```

下面介绍如图 7-10 所示的应用协议加速原理。

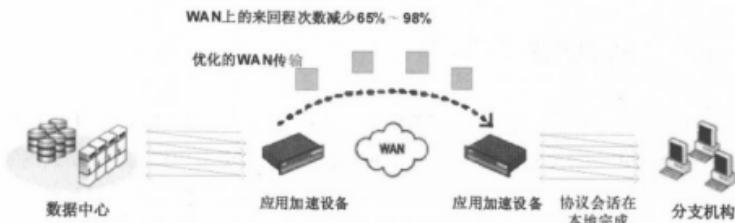


图 7-10 实现跨广域环境应用加速流程

原来的连接方式是在服务器与客户端之间建立一个 TCP 连接，在加入了一对应用加速设备后变成三段 TCP 连接，即客户端—本地应用加速设备、本地应用加速设备—远程应用加速设备、远程应用加速设备—服务器。两端的加速设备会针对不同应用协议分别与客户端、服务器进行会话，减少客户端与服务器之间的直接会话，而实现的关键是加速设备必须理解具体的应用协议，如文件共享、备份、上网、数据库连接等，这样才能使原来客户端与服务器的远程广域网会话能在本地完成。比如文件共享应用（CIFS、NFS），在真正数据传输以前，可能总计有一千次广域网范围的握手同步信息。部署了应用加速设备后，客户端本地的应用加速设备能够在本地响应客户端请求，与对端代表服务器的应用加速设备批量进行一千次握手。本来在广域网应用层的一千次会话交互，现在变成了在局域网的一千次会话交互，真正在广域网的会话交互可能只要几十次。

还是看上面 FTP 协议的具体例子，应用加速设备可以捕获 Step1~Step4 中

的 Client 命令，并对 Client 的登录和认证操作按照正常通过状态发出响应。在收到 PASV 命令时将前面 4 个步骤接收到的 Client 命令发送到对端应用加速设备，由其一次性快速完成与 Server 端的登录和认证会话。

上面介绍了数据层传输优化技术、TCP 协议优化技术、应用层协议优化技术，这是广域网传输加速的三项基本技术，在具体实现中，各设备厂商在细节上会有所差别。

### 7.2.3 SSL 加速技术介绍

在本书第 3 章我们了解到 HTTPS 安全协议是以安全为目标的 HTTP 通道，通过在 HTTP 下加入 SSL 层，能够实现传输加密，避免用户数据、交易数据等重要数据被窃取。SSL 加密是一种处理器密集型加密算法，如果用服务器软件处理会消耗大量 CPU 资源，所以一般会在提供业务能力的服务器外围部署专门的 SSL 加速设备，采用硬解密方式实现。

在 SSL 会话中，计算量最大的部分当属 SSL 握手阶段，这个阶段的主要工作是协商会话密钥，该密钥通常是对称密钥，将被贯穿应用于相应的会话过程中；与此同时，SSL 握手消息本身的加密和签名则是包含在证书中的非对称密钥，使用这种非对称密钥比对称密钥对计算资源的消耗更大。

针对这种情况，SSL 加速技术会将对 SSL 握手过程的处理任务交给专用硬件设备，有的实现方式是将对称加解密操作保留在服务器上由软件实现，而有的则是让 SSL 专用硬件以代理的身份全权处理 SSL 操作，将去除了 SSL 层的 HTTP 内容交给服务器。

#### **1. SSL 的基本原理和实现**

Internet 通信协议，无论是 IP 协议还是 TCP 协议，其初始设计中都没有考虑安全问题，因此 SSL 提出要满足以下 4 方面要求。

(1) 可认证性：每个通信参与方都应该能验证其他参与方是否与其声明身份相符，而不是一个冒名顶替者。

(2) 隐私性：确保第三方不能窃听通信双方的通信内容。

(3) 完整性：能自动地或者非常容易地检测到通信信息在传播中的任何篡改行为。

(4) 不可抵赖性：发送者不能自称没有发出过接收者从他那里收到的内容。

SSL 最早由 Netscape 开发，最初也存在着诸多问题，但随着其不断发展，多数问题被相继修正。目前使用的 SSL V3 已经成为最流行、使用最广泛的互联网安全协议。

在这里，我们再简单重述一下第 3 章讲过的 SSL 基本原理。SSL 的基础是基于成对密钥的公钥加密系统，其中一个密钥是为外界所知的公钥，另外一个密钥是只有密钥拥有者知道的私钥。无论是哪个密钥都能够对信息加密，而同时只有另一个密钥能够进行解密。当一个发送者使用接收者的公钥对传输加密时，只有接收者能够将其解密，这样就可以保证隐私性；而当一个发送者用自己的私钥对传输加密时，接收者只有利用发送者的公钥才能解密，这样保证了可认证性。

SSL 会话的启动首先要完成 SSL 握手过程，用于双方交换公钥。在交换过程中，客户端和服务器需要使用相应的公钥和私钥对彼此进行认证，主要方法是双方出示包含了由彼此都信任的证书管理机构签名的数字证书。数字证书能够防止那些使用自己的公钥却声明为别人的冒名顶替者的出现，同时证书管理机构在对公钥进行签名时也绑定了密钥和身份的对应信息，避免了信息发送者抵赖。

图 7-11 所示是基于软件的 SSL 实现，服务器的处理器负责各个会话初始的密钥交换以及后续的数据加解密。这种密集的计算过程会使服务器性能承受

极大压力，使得其他事务处理能力大大降低。因此基于软件的 SSL 实现只适用于管理少量 SSL 流量的场景，例如部门、工作组或者非商业化用途的 Web 服务器。



图 7-11 基于软件的 SSL 实现

## 2. SSL 加速的基本原理和实现

SSL 加速通常指的是基于硬件的 SSL 实现，利用专用硬件设备处理 SSL 任务，具体可分为基于服务器加速板卡和基于专用加速设备两类方法。

### (1) SSL 加速板卡

在服务器上安装一块 SSL 加速板卡，可以有效分担服务器 CPU 处理 SSL 事务的压力。SSL 加速板卡通常有一个或多个协处理器用于实现 SSL 计算。这些协处理器可能采用通用 CPU，也可能采用定制的 ASIC 芯片和 RISC 指令集芯片。每块加速板卡每秒至少能够处理几百个 SSL 事务，当前大多数商用 Web 服务器都已经支持将 SSL 加解密工作卸载到这种加速板卡上，如图 7-12 所示。



图 7-12 基于服务器加速板卡的 SSL 实现

一般情况下，Web 服务器每秒需要管理数千个未加密的事务，如果没有加速板卡的话，就意味着每秒至少有数百个 SSL 事务需要处理，因此很大一部分

服务器处理能力将被用于 SSL 事务，降低了服务器应有的 Web 事务流量。如果能够在服务器上引入多块 SSL 加速板卡，那么服务器的 SSL 事务处理能力就能够很容易地扩展到每秒处理上千个事务，而与此同时，服务器的处理能力不受任何影响。用于板卡的加速技术的发展将会进一步地改进 SSL 处理能力。

如果 Web 事务的流量要求超过了一定的阈值，那么就需要在站点内运行多台 Web 服务器共同提供服务，这时就必须使用负载均衡设备在服务器间调整负载压力以获得扩展性和容错性。

在 Web 服务器集群场景中，使用 SSL 加速板卡会存在一定问题，主要是无法保持会话持续性。通常一个 Web 服务器集群是通过一台四层交换机连接的，负载均衡设备通过检查 HTTP 请求头部信息和 cookie 相关信息来保持会话持续性。但如果由 Web 服务器的加速板卡来处理 SSL 请求，就意味着数据报文穿过负载均衡设备时是带有加密的，这样负载均衡设备就无法看到 cookie 信息，从而无法按照原来的策略实现会话持续性的保持了。

为了解决服务器集群访问中的会话持续性问题，人们提出了一种基于 SSL 会话标识的方法。但用户也可能在浏览器的安全特征设定中要求与后端服务器重新协商 SSL 会话标识，这样的话，SSL 会话标识方法仍旧不能确保会话的持续性。

除了上述问题，在这类基于服务器加速板卡的 SSL 实现中，管理员面临的最大挑战是证书管理。因为集群中的每台服务器上都必须具备唯一性数字证书，而证书的购买、部署、维护等将极大地增加 SSL 实现的总体成本。

## (2) SSL 加速设备

SSL 加速设备是嵌入 SSL 加速板卡的独立设备，用于对加密流量进行解密，并将解过密的数据信息发送给后台服务器。在相反方向上，它负责加密由后台服务器发来的明文数据再将其转发给客户端。SSL 加速设备的工作原理如图 7-13 所示。

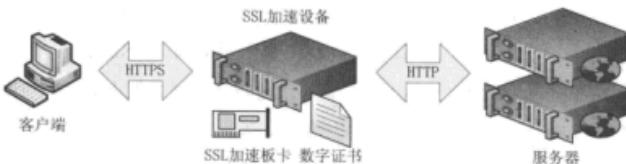


图 7-13 基于专用加速设备的 SSL 实现

由于 SSL 加速设备终结了 SSL 会话，后台服务器现在完全被释放出来用于数据服务或者运行应用程序，它不再需要跟踪任何 SSL 连接，这可以极大改善系统性能。另外，使用 SSL 加速设备的另一个优势是可扩展性，管理员可以将多个 SSL 加速设备组成一组，以应对更大规模的 SSL 事务处理需求。

在很多应用场景下，SSL 加速设备都是和负载均衡设备一体实现的，这主要出于成本考虑，而且能够应对支持会话持续性需求。

## 第8章 CDN 商业化服务现状

- 8.1 CDN 产业分析
- 8.2 CDN 的商业服务模式
- 8.3 典型案例分析
- 8.4 典型服务商介绍



## 8.1 CDN 产业链分析

### 8.1.1 CDN 产业链分析

伴随着互联网的发展与成熟，互联网产业链日趋稳定。在整个产业链中，从平台设备、内容到最终用户，各角色各负其责。内容提供商和用户位于价值链的两侧，中间依靠网络服务提供商将其串接起来。互联网内容的提供经过了 SP 自建集群服务器、IDC 内容托管、镜像网站、CDN 分发 4 个阶段。随着互联网工业的成熟和商业模式的变革，在这条价值链上的角色越来越多也越越来越细分。示意图如图 8-1 所示。



图 8-1 互联网产业链示意图

但是，IDC 并不能解决内容的有效发布问题，托管在 IDC 中的内容还是位于网络的中心，所以不能解决网络骨干带宽的占用问题和建立 IP 网络上的流量秩序。因此将内容推到网络的边缘，为用户提供就近性的边缘服务，从而保证服务的质量和整个网上的访问秩序就成了一种显而易见的选择。而这就是 CDN 的服务模式。CDN 解决了困扰内容提供商（CP）的内容“集中与分散”的两难选择，无疑对于构建良好的互联网价值链是有价值的，也是不可或缺的最优网站加速服务。

近年来，CDN 服务内部也在逐渐细化，正在形成 CDN 自己的产业链，包括接入和设备托管、硬件设备、软件系统和运营保障等环节，如图 8-2 所示。

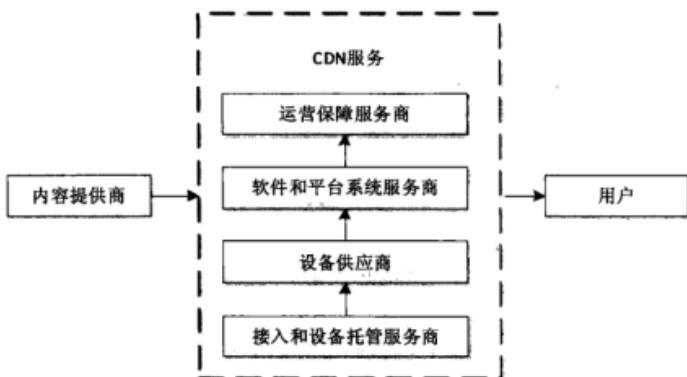


图 8-2 CDN 细分价值链

一般情况下，接入和设备托管服务由电信运营商承担，狭义的 CDN 服务供应商主要涵盖软件系统服务和运营保障服务，部分 CDN 服务供应商对硬件设备进行定制，承担了部分设备供应商的角色。总而言之，CDN 服务呈现软硬件分离的趋势，很多商用 CDN 系统都是在通用服务器上开发自己的 CDN 系统软件实现的。这也是 CDN 技术一直难以在国际组织中取得一致性标准意见的最主要原因。CDN 产业上游为内容供应商，下游为最终用户，内容供应商是采购 CDN 服务的主体，是商用 CDN 系统的主要客户来源。在产业链的内部，承担网络接入和设备托管的电信运营商具有基础和主导作用。

内容服务商是网络内容/应用的提供者。在 CDN 加入以后，提供内容的角色不再处于网络中心，而是分散在网络边缘，这是与传统互联网结构的最主要区别。CDN 为用户提供就近性的边缘服务，从而保证服务的质量和整个网络上的访问秩序。内容提供商需要为 CDN 付费，同时也是 CDN 价值的最大体现者。

CDN 网络运营商是 CDN 产业链的核心，通过提供平台和建立收费机制以及与内容供应商分账的商业模式来创建“正反馈”价值链，使得整个价值链良

性运行。从这个角度讲，CDN 的建设和运营对于电信运营商是有帮助的。

### **8.1.2 CDN 服务的价值分析**

CDN 作为互联网时代的基础性专业服务，其价值主要体现在以下几个方面。

#### **1. 专业细分**

分工细化是产业生态逐渐成熟的必然结果。随着分工的演进，产业链条延长，链条内企业专业化生产水平提高，从而更有效地提高了专业化企业对其专门领域内知识与技能的积累效率与速度，促进新技术和新功能的发明与使用。迂回分工链条的延长进一步提高了生产效率。合理的分工可以形成良好的产业环境，使个体能在短时间内积累到非专业化状态下要花数倍时间才能积累的经验，创造出显著的熟能生巧的动态效果，从而也可以通过促进个体的专业化来加快分工的持续发展。

CDN 作为互联网产业链的关键一环，由于目前并没有国际统一的标准规范，个性化特征明显，每个企业的专业经验积累就更加重要，通过大量研发投入和实际运营经验积累，形成 CDN 运营的专业化优势。

#### **2. 外包服务**

CDN 服务所提供的，不仅仅是分发技术，还包括遍及全国的外包服务网络。在网站运营的过程中，其内容提供方往往集中在一个地点，因此希望内容的接入和维护尽可能集中。但网站覆盖的用户是遍及全国的，在不同地区用户群和消费能力均有不同，现在的富媒体应用，尤其是高清视频，要求内容提供尽可能靠近用户。对于单一网站来讲，如果建立全国的网站运营维护体系，不仅成本高昂，而且面临巨大的运营风险，不符合当前互联网站的轻资产需求。

CDN 承担的全网服务正好可以解决这一困境。CDN 运营商建立的全国维护体系，是面向多个网站的，这样就有效地规避了单一网站运营波动的风险，同时可以充分利用不同地区的资源差价和员工培训、聘用机制，替网站承担靠

近用户服务的责任。

### 3. 集中采购

CDN运营商需要大批量采购架设CDN网络所需要的硬件和机房带宽资源，从而建立了完整的资源采购体系，包括自行采购和通过代理采购。现有的CDN运营商部分也运营IDC托管业务，在向电信运营商采购带宽和机架资源时，由于采购量较大，具备更强的议价能力，同时多年的CDN资源采购，已经形成独有的资源采购体系，可以通过当地代理商取得更加低廉的价格。

同时，CDN服务总的趋势是软硬件分开，不再通过专用硬件实现相关功能，而是通过持续技术研发，在通用服务器上采用软件进行实现，这样就大幅度降低了专用硬件的采购成本。此外，大规模集中采购通用服务器也能够降低CDN运营商的运营成本。

### 4. 按需提供

网站的实际带宽需求，根据业务内容、用户行为、消费模式和定价方式等，都在不断变动，往往会出现突增和突降的情况。若所有带宽均自己采购，在无法对实际使用情况进行准确预估的前提下，如果过多储备资源，会造成成本的上升和浪费；如果储备资源不足，在网站的爆发点会形成带宽瓶颈，在用户最感兴趣的时间点造成网站无法访问或无法流畅访问，其损失更大。CDN面向多家网站服务，对于储备的这部分带宽平时不使用，往往具备一定容量的资源池，同时和电信运营商达成带宽资源储备协议，当某个网站产生突发的高带宽需求时进行调度，真正做到按需分配。

同时，不同网站的消费模式往往不一样，如证券网站的流量高峰往往在早上九点半到下午三点的股市开市时间，而视频网站的高峰往往在下班以后到睡觉前的傍晚时间段，如果各自采购，只能按各自的峰值需求进行储备，如图8-3所示。

从图8-3中可以看出，甲、乙用户如果自己单独购买带宽组建CDN服务，均需购买B大小的资源，甲乙共需购买 $B \times 2$ 带宽。而如果他们共同从CDN服

务商购买服务，由于彼此的峰谷叠加，CDN 服务商只需要从电信运营商购买 B 大小的资源即可提供相应的服务。

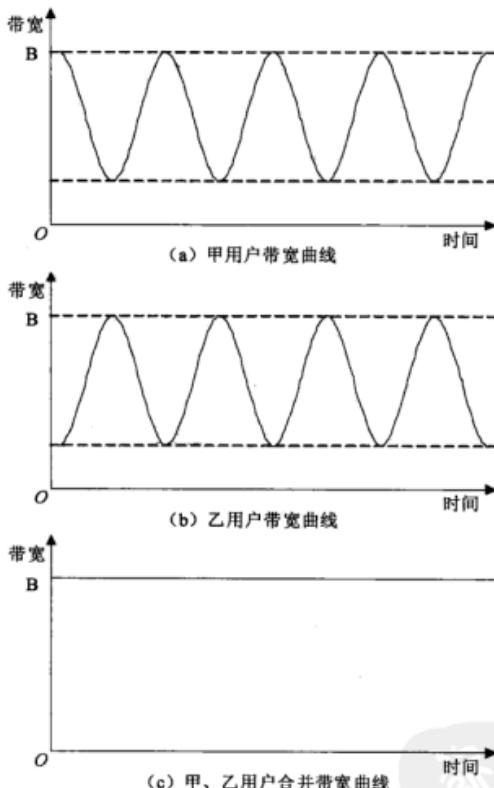


图 8-3 CDN 合并带宽峰谷

在互联网的实际运行过程中，由于某一区域网民数相对稳定，网民的上网习惯和上网时间也相对稳定，考察网民的整体行为，其带宽需求一般为一个稳定的曲线。只是这些上网时间根据网民的不同选择，是在不同网站之间分配的，而总的需求处于稳定状态，因此面向网民提供多个网站内容的 CDN 服务节点

的运营压力一般处于稳定状态，比单个网站更加可预测和可控制。

### 8.1.3 CDN服务运营方式分析

CDN主要有以下三种运营方式：自建、租用和混合。

#### 1. 自建

自建CDN是指内容供应商自己租用带宽和托管服务，并自行开发和管理相应的CDN系统和平台。自建CDN需要内容供应商的规模足够大，足以支撑一个全网的CDN服务，同时具备相当的技术力量和维护力量，需求相对单一明确，复杂度不高，个性化较强。自建的CDN系统多以某种开源系统作为基础，针对自身的个性化需求进行开发设计，从而更好地适应内容运营的需要。自建CDN在国内应用较多，各大门户网站和视频网站均有自建的CDN网络。

#### 2. 租用

CDN运营商向电信运营商采购接入和托管服务，加入CDN技术增值之后，将CDN网络带宽出租给内容供应商，即利用CDN为内容供应商提供有质量保证和局部个性化的内容分发服务，而以占用的带宽和存储容量为标准向内容供应商收费。国外的一些CDN运营商（如Akamai）目前采用这种方式。国内很多中小型网站由于不具备自己组建CDN的能力，基本采用租用的方式；同时，政府、企业和行业客户更希望采购专业化的服务，也大多采用租用方式。国内大型网站基本都有自建CDN部分，同时也采购一些CDN服务支撑企业的发展。随着互联网产业链的成熟，专业化分工更细，网站租用的比例将越来越高。

#### 3. 混合

混合方式是指内容供应商采用自建和租用混合的方式对自己的CDN服务进行整体布局规划。混合的选择标准包括地域、业务和峰值分担等。地域混合式是指内容供应商自主采购部分地域的网络服务，其余补充的地域采用租用的方式；业务混合式是指内容供应商自行开发核心业务的分发（如视频或Web Cache等），而将其余的服务对外租用；峰值分担是指内容供应商日常稳定流量需求自己满足，遇到特殊峰值或流量波动时，在自己带宽无法满足的时候临时租用。

企业采用何种方式采购 CDN 服务，需要结合企业自身情况、业务需求、客户分布、流量变化等多个因素加以考虑。总的来看，产业链细化和服务外包是一个整体趋势。

艾瑞咨询分析数据显示，2002~2009 年间，全球 CDN 市场保持持续、稳定、快速的增长，增长率维持在 30% 左右。随着全球 CDN 市场走向成熟，总体增长趋势开始缓慢下滑，但不排除互联网新经济因素的影响，在短期对 CDN 服务市场出现拉动。2007 年全球 CDN 市场规模达到 31.80 亿美元，比 2006 年增长 25.69%。根据艾瑞的预测，到 2014 年，整体市场规模将达到 219.41 亿美元，如图 8-4 所示。



图 8-4 全球 CDN 市场规模预测

## 8.2 CDN 的商业服务模式

### 8.2.1 CDN 的计费方式

CDN 服务一般有两种计费方式，即带宽计费和流量计费。在确定计费方式

之前，要先搞清楚 CDN “带宽” 和 CDN “流量”的区别，因为这两个概念所使用的单位都是 M 或者 G，致使很多客户自己也混淆。带宽的单位其实是 bit/s (b/s)，而流量的单位就只是 bit。这就可以看出，带宽指的是一种传输速度，而流量是一种传输数量。如图 8-5 所示，带宽是一段时间的连续函数，流量则是这个函数针对时间的积分。

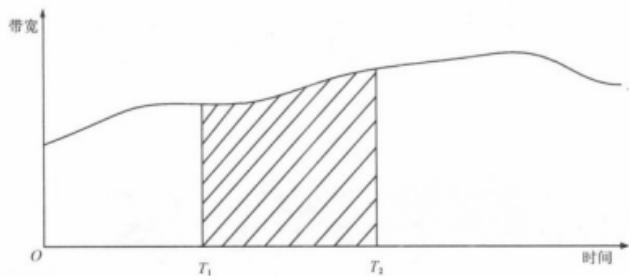


图 8-5 带宽和流量之间的关系

如图 8-5 所示，在某个时间点，如图中  $T_1$  和  $T_2$  均有一个瞬时带宽值，流量则是这两个时间之间的面积。目前 CDN 计费以带宽为主，流量计费使用的不多。按照流量来计费是个比较科学的做法，每个用户根据自己每月实际使用的量来支付带宽费用，就和家里交水电费一样。但是这种付费方式需要有强大的监测系统做支撑，而且这种监测措施还要得到客户广泛认可。

一个实际运营的 CDN 网络，有多层和多个节点，计费采集的是面向用户服务的边缘节点的总和，如图 8-6 所示。

在图 8-6 的 CDN 架构中，边缘结点有  $T_1$ 、 $T_2$ 、 $T_3$ 、 $T_4$  共 4 个，这 4 个节点就是计费采集点。因为节点同时分发不同源站的内容，主要是以域名作为统计指标，将某一域名下的流量或带宽汇总，形成计费依据。

由于带宽往往是连续变动的（参看图 8-7 中的实际带宽流量），采用带宽计费时，根据取值方法的不同，可以分为两种方式：峰值计费和 95 计费。

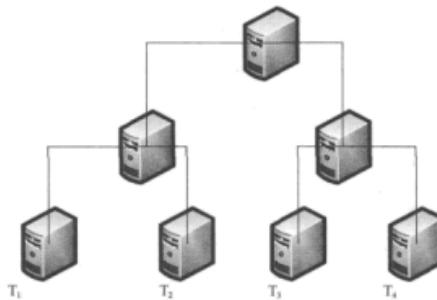


图 8-6 CDN 计费采集点

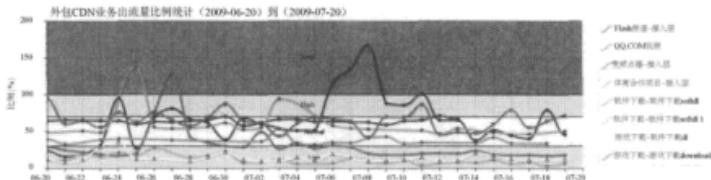


图 8-7 CDN 使用带宽连续变化

峰值计费采用一个时间段内的最高点作为计费依据，这种计费方式在遇到特殊流量变化时，往往会造成费用激增；另外一种方式是 95 计费，即在一个计费周期内（如一个月），每 5 分钟采集一个平均带宽（5 分钟内流量除以以秒为单位的时长），一个月内采集 8000 多个计费值，抛弃掉前 5% 的计费值，剩下的最高值作为计费依据。根据一般经验，95 计费大约为峰值计费的 80% 左右。

在实际的商务运作中，还有其他更为灵活的商务模式，如保底带宽实际流量计费、封顶带宽计费等。采用保底带宽实际流量计费时，网站向 CDN 运营商购买一个保底带宽，即使无法达到这一带宽依然按照这一带宽计费，同时 CDN 运营商需要保障这一带宽的提供。在超过这一带宽时，采用“尽量保障”的方式，即 CDN 运营商有空余资源才予以保障，超出部分另行计费。采用封顶带宽计费时，则是网站向 CDN 运营商购买一封顶带宽，超出带宽的部分直

接回到源站，不再使用 CDN 网络进行分发。

### **8.2.2 CDN 的增值服务**

在提供基本内容分发服务的同时，各 CDN 服务商均针对用户的不同需求，开发出围绕 CDN 出租服务的一系列周边增值服务，归纳一下目前提供的增值服务主要有以下几类：安全服务、增强 CDN 服务、用户数据分析服务、网站数据分析服务以及日志分析处理服务，后两类增值服务主要建立在强大的日志分析和处理技术上。

#### **1. 安全服务**

自从信息时代开始的那一天起，危害网络和信息安全的攻击威胁就一直伴随着它。近年来，这类攻击变得更加复杂，且规模更大，其危害程度和攻击力屡次突破人们的想象。短短几年中，分布式拒绝服务（DDoS）攻击规模急剧扩大，从每秒几十 Gb 到数百 Gb，2008 年间总计发生了 19 万起 DDoS 攻击。根据 Arbor Networks 的调查结果，最大的 DDoS 攻击规模在 7 年中增加了 100 倍，从 2001 年的 400 Mb/s 到 2007 年的 40 Gb/s。2008 年的恶意代码攻击数量已超过了之前 20 年的总和，2009 年上半年就已超过 2008 年全年的总量，每 8 秒就出现一个新的威胁标志。安全公司 Sophos 预测，每 3.6 秒就有一个新的网页受到病毒感染。网络应用安全联盟指出，目前超过 87% 的网络应用程序存在“高风险”或更高等级的漏洞。

CDN 利用广泛的服务器部署，利用多重技术防止网站受到攻击和盗链，在很大程度上保证了网站的可用性和对网站内容（尤其是音视频和软件下载）的知识产权保护。一般 CDN 使用的安全技术包括如下几项。

- 源服务器流量卸载，即通过对源服务器访问流量的分担，降低源服务器的服务压力。
- 源服务器隐藏，即只是保证 CDN 某些高层分发节点服务器能够访问

到源，源服务器对于普通互联网是隐藏的。

- DNS 保护，即通过分布式安全加强型 DNS 基础架构来处理最终用户的 DNS 请求，从而为客户的系统提供高扩展性和容错性的 DNS 服务。
- 网站热备服务，即利用分布式服务器，提供网站的完整热备功能，一旦源服务器出现严重问题，可以通过 DNS 调度直接切换到热备系统。

总之，CDN 将单一网站扩展到全网，大大降低了网站的风险，用分布式的方案提高了网站的健壮性。

### 2. 增强 CDN 服务

这是指 CDN 服务提供商在通用加速功能之外，针对客户的特殊需求专门开发的特殊内容加速服务，主要有如下几项。

- 针对移动互联网的加速服务。该服务针对移动互联网的特殊内容和终端要求，实时识别移动设备的请求，然后将其重定向到指定的移动网站服务器，确保提供移动优化的体验。
- 网络存储服务。为客户提供源站文件大容量存储和上传，将不同文件分发到不同服务器上进行存储，以实现数据的分布式存储；让每台机器只为相对固定的用户服务，以实现平行的构架和良好的扩展性。
- 区域化加速服务。利用 CDN 服务的精确定位技术与分布式特点，为网站提供地理区域定制化加速服务，为不同地区的用户展示不同的内容。

各家 CDN 服务提供商的增强服务不一而足，各显神通，这里不再一一列举。

### 3. 用户数据分析服务

由于 CDN 服务提供商同时接入了大量用户和网站，从而掌握海量的用户

行为数据，这些数据可以为用户消费预测、广告投放设计、广告投放评估等提供依据。

以 Akamai 公司为例，其用户数据库由 550 多个渠道零售商、产品制造商、旅行和远程通信网站组成，提供每季度高达 135 亿美元的匿名消费者购物交易数据。这个独一无二的数据库使 Akamai 可以理解顾客的各种购买行为方式，从而准确地定位市场内的用户。

在海量用户行为数据之外，还可以针对某一网站的用户行为数据进行深入分析，包括如下几项。

- 网站总体访问情况分析，如浏览量、用户数、停留时间、最大访问时段、最大访问地区、首页访问量、人均浏览数、最大错误页面、最大离开页面等，这些是网站运营中普遍关注的信息。
- 行为特征分析，可以根据客户对“用户特定行为”的定义，分析用户在该特定行为中每一步都是如何进入又是如何离开的，以及用户在网站停留期间的具体行为特征。
- 渠道来源分析，通过对直接登录、连接访问、搜索引擎、广告访问、电子邮件等渠道访问用户的深度分析，从而判断网站的知名度和市场推广效果。
- 市场推广分析，分析企业在所有媒体上进行的广告与电子邮件推广的曝光率、点击率、点击转化率和客户转化率。
- 页面结构分析，页面的热点是将物理页面划分成若干区域，这些区域需要个性化定义，在该物理页面上直观给出各区域的浏览数及对本页的贡献度。

CDN 所采集的用户行为数据，只是用户的访问数据，不包括用户数据的具体内容和实际身份，不会泄露用户个人隐私，法律风险较低。

#### 4. 网站数据分析服务

网站数据分析是指对网站的实际运行情况进行深入分析，一方面可以及时预测网站的发展趋势，另一方面可以作为网站系统的优化依据，提高系统性能。

CDN 服务提供商通过分布在全球不同网络层面（第一公里、中间一公里、最后一公里）的监测点，对网站应用进行“由外至内”跨用户、跨浏览器、跨终端及跨地域的监测，审视从最终用户的浏览器至防火墙内整个网站应用传输链的每个环节，最大限度地发现并解决影响业务的各类问题，以此优化网站及应用程序的性能、可用性和质量，确保持续、高质的网站用户体验。

同时，CDN 服务商可以对 Web 运行状态进行详细、全面的分析与还原，帮助网站运营管理者精确掌握：网站/频道/专栏等详细的访问指标、用户的准确来源、网站内容受欢迎程度、网站正在呈现的错误，以及各项指标的变化趋势与同期比较等，使得企业各级管理人员透视日常工作的有效性，及时发现问题，提高经营及管理水平。

#### 5. 日志分析处理服务

前面提到用户数据分析服务和网站数据分析服务是 CDN 运营商向网站等客户提供的增值服务，有的网站客户可能会更加倾向于自己进行挖掘和分析，这时 CDN 运营商也可以基于客户的这个需求而直接提供日志服务。

没有使用 CDN 服务的时候，网站 SP 一般通过用户访问源站或者镜像网站的记录来分析用户的访问情况，从而知道用户对网站的偏好、活跃度、分布范围等。使用 CDN 后，用户不能直接访问源站，所以统计用户访问记录的工作自然需要 CDN 来做，CDN 的分布式架构和完善的运营支撑体系也表现出比源站更高效的统计优势。在 CDN 中，这种海量数据的记录和分析都是通过日志统计技术来实现的。除了通过统计用户的访问日志来了解用户的访问情况，还可以通过日志记录的设备运行状况、管理员操作记录等来分析网络中需要优化和改进的部分。

日志分析处理服务在技术上需要经过日志的记录、日志的压缩和传输、日志的分析和报表三个阶段，最后形成的报表将定期上传到客户子服务平台，为客户提供信息参考。

(1) 日志的记录。日志记录是 CDN 中每个被管理设备所必需具备的基本管理功能。虽然不同的 CDN 在日志分类上各有差异，但大体上可以分为用户访问日志和系统日志。用户访问日志记录 CDN 用户访问 Cache 设备或者 Cache 设备之间的互访记录，通常包括用户 IP 地址、访问的内容对象、访问是否命中等；系统日志通常包括 BOSS 对设备进行的全局配置、网络管理员对设备进行的特定操作、系统运行中出现的性能指标异常等。一般情况下，可为网站客户提供增值业务参考的信息主要是用户访问日志，系统日志作为 CDN 提高自身运营能力的基础参考信息。

日志记录的颗粒度表示日志对一条访问记录或者操作记录得有多详细，通常日志记录颗粒度的大小直接影响日志数据量的增长速度和分析效率。CDN 在进行日志记录配置时通常需要考虑日志记录的配置是否会对记录日志的设备的正常业务造成影响，是否会对报表分析的性能造成影响。因此，在正常运行情况下，日志的记录会兼顾网站客户的需求和系统性能，只记录相关联的关键数据，除非出现网络或业务异常情况而需要诊断时才会开放更多的记录资源。对于 CDN 系统而言，网站客户比较关心的统计值包括峰值、累计和并发数量，统计的时间可以是时间点或时间段，时间颗粒度可以细化到分钟、小时、日、周、月或年。统计的区域颗粒度可以细化到节点、省、市、县或区。

(2) 日志的压缩和传输。如图 8-8 所示，CDN 各节点和设备产生的日志文件需要定期传输到统计分析模块进行分类、聚合等一系列的分析工作。网站客户和 CDN 网络管理员对日志分析结果的时间要求不如性能监控数据那样实时，因为统计结果往往只是一段时间的一个统计值，用于为后续的业务调整和网络优化提供参考，所以日志无须实时上传给统计分析系统。但考虑到日志数据量增长速度较快，节点存储空间有限，以及客户对统计数据的新鲜度有 SLA 要求，

所以日志上传的时间间隔也不宜太长。目前比较典型的情况是每天上传一次，通常会选择在业务量最小的凌晨时段。即使每天上传日志文件，仍然需要对一天的日志进行压缩，然后以 FTP 协议的方式发送给统计分析系统。

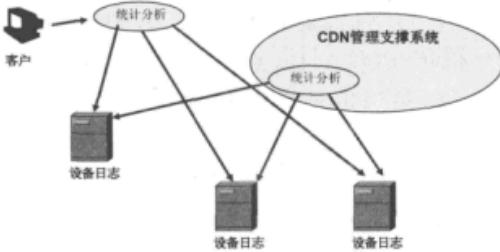


图 8-8 日志传送

(3) 日志的分析和报表。统计分析系统是 CDN 网络管理支撑系统的重要组成部分，同时又是一个可配置的高度独立的个体模块。也就是说，统计分析系统能够经过简单的定义后直接提供给客户成为客户的统计分析工具，而不需要与管理支撑系统捆绑。由于统计分析系统需要处理的数据量很大，原始的日志文件会先在各节点设备中经过一次分析处理，这种初步分析只能根据统计报表的要求有针对性地挑选出适合分析的日志记录。经过初步分析的日志文件会在压缩后再 FTP 到统计分析系统中进行再分析，最后形成符合客户和网络管理员所定制的报表文件。

### 8.2.3 CDN 客户决策要点

客户在购买 CDN 服务时，需要综合考虑的因素较多，主要有以下几个方面。

1. 服务：CDN 作为一种专业服务，其服务水平是首要要求。服务水平主要涉及针对客户的服务态度和服务意识，出现问题之后的反应速度，对问题的

处理能力等方面。CDN服务需要提供 $7\text{天} \times 24\text{小时}$ 运维支持，随时解决问题，以保证网站的畅通性，减少对网站运维团队的压力和人员要求。

2. 平台：需要考察CDN平台的可靠性和稳定性，也要考虑其安全性和灾备能力以及突发情况下的应变能力。对于平台的能力，客户往往无法直接看到和测试，重点应该考察客户系统的运营时间和相关客户反馈信息等。

3. 网络覆盖：CDN是面向全网进行分发的，其网络覆盖能力非常重要，尤其是和网站的自身目标客户的重合度，看其是否能覆盖全国甚至全球主要网络。

4. 技术：需要提供CDN服务的可扩容性和功能可扩展性，一方面要能满足现在的运营需求，另外要能符合网站未来的规划需求。

5. 定制：是否能根据网站的个性化需求进行定制开发，如提供特殊协议加速、消费行为分析、客户行为跟踪等相关信息。

在初步选择CDN服务商之后，需要跟踪测试相关服务指标和信息，进一步分析测试其服务能力，测试内容包括如下几项。

- CDN布署后的分发性能具体指标，是否有提升，提升幅度多大。
- 提供了多少主机节点。
- 主机节点分布在哪些区域和运营商。
- 每台主机节点的性能如何，可用性是否稳定。
- 目标客户是否正确命中对应主机节点，或匹配度是否合理。
- 单台主机节点的覆盖范围或承载比例如何。
- CDN节点对源站的同步效率做得如何。
- CDN对内容的发布技术是否提供到位并长期有效。

- CDN 节点故障源站有无检测能力，如何对其进行及时报警。

## 8.3 典型案例分析

CDN 技术的出现，给互联网带来了革命性的变化，尤其是大带宽、高并发的内容推送类型网站，在发展到一定规模之后，CDN 成为必备的发展基础，以下就对几种典型类型的网站如何使用 CDN 服务进行分析。

### 8.3.1 视频网站

美国职业篮球协会（NBA）成立于 1946 年，是一个包括美国和加拿大 30 支球队在内的全球性体育和娱乐品牌。在 2007/2008 赛季，全球有 215 个国家以 41 种不同的语言转播了 900 多场比赛和 45000 多小时的节目。NBA.com 是世界上最大的也是访问量最多的网站群，由超过 60 个独立网站组成，包括 NBA.com、WNBA.com、NBDL.com、球队网站和 NBA.com 的国际站。这些网站向全世界的球迷大范围地提供关于联赛的新闻、成绩、统计数字、宽带视频热点、独家幕后花絮和梦幻 NBA 游戏。

NBA 作为体育市场的领军品牌，积极致力于依靠科技拉近与全世界球迷的距离，这需要借助合适的方案才能帮助他们有效地管理和分发丰富的媒体资产，这一方案必须满足以下三个关键要求，以支持公司的品牌和目标。

- 高效地管理数字资产：随着需要管理的数字资产不断增加，NBA 需要确保自己拥有各种必备的工具。
- 稳定的性能：该机构希望全世界的球迷在登录其互动网站时能获得相同的体验。

- 严格的许可证授予权限制：由于体育广播授权一般只限特定时段和特定地区，所以 NBA 需要确保在网上提供在线服务内容时没有违反任何一款分配协议。

最终 NBA 采用了 Akamai 的流媒体播放和管理系统，Akamai 为每场 NBA 现场球赛进行音频和视频编码，因而该机构仅需要一到两名员工就可以实现每年网播 1500 多场赛事，使网上广告收入增加 500%。NBA.com 网站的流量已实现每月成倍增长，每月有分布在 222 个国家的 3500 多万独立用户访问 NBA 网站内容。NBA 创下了月度访问 NBA.com 突破 1460 万人次、访问全球音频流媒体突破 3040 万次的记录。同时，CDN 服务协助 NBA 激活了重要的业务模式，可以帮助其向合作伙伴网站、获得许可的个人和网站以及桌面工具发布内容，确保其内容的发布不会违反合同义务。

### 8.3.2 门户网站

新浪，是一家服务于中国及全球华人社区的领先在线媒体及增值资讯服务提供商，成立于 1998 年 12 月，和搜狐、网易、腾讯并称为“中国四大门户”网站，新浪网主要提供网络媒体及娱乐服务。新浪在全球范围内的注册用户超过 2.3 亿，日浏览量突破 6 亿次，是中国大陆及全球华人社区中最受推崇的互联网品牌之一。通过与国内外千余家内容供应商达成合作关系，新浪设在中国大陆的各家网站提供了三十多个在线内容频道。新浪及时全面地报导涵盖了国内外突发新闻、体坛赛事、娱乐时尚、财经及 IT 产业资讯等内容，汇聚各行业精英的新浪博客，以及优质独家的宽带互动视频产品，更成为上亿中国互联网用户生活中不可或缺的部分。

新浪采用了 ChinaCache（蓝汛）做的 CDN 系统，同时采用基于动态 DNS 分配的全球服务器负载均衡技术。在北京地区，ChinaCache 将 www.sina.com.cn 的网址解析到 libra.sina.com.cn，然后 libra.sina.com.cn 做了 DNS 负载均衡，再将 libra.sina.com.cn 解析到 16 个 IP 上；这 16 个 IP 分布在北京的多台前台缓存

服务器上，使用 Squid 作为前台缓存，各个频道的前台缓存集群与 www.sina.com.cn 的前台缓存集群是相同的。新浪 CDN 系统每天承受大量的访问，服务一直十分稳定、快速。

### **8.3.3 政府网站**

政府门户网站是政府形象和宣传的窗口，随着电子政务平台的日趋完善，各类政府网站逐步由简单的政务信息静态发布向政务办理、信息搜集调查等综合互动过渡。同时民众对政府网站的关注度大大提高，特别是突发事件要求网站有足够的带宽储备；另外，政务信息的公开也使网站容易成为黑客的攻击对象，对安全性提出了较高的要求，政府门户网站在流媒体应用中所面临的挑战有如下几项。

- 如何保证政府网站内容的及时推送和安全稳定，是政府网站的两大首要难题。
- 如何应对各种可能的恶意攻击。
- 如何应对突发事件不可预知的访问流量。

新华网由中国国家通讯社和新华社主办，为国家重点网站，具有全球影响力。新华网依托新华社遍布国内外的多个分支机构，组成了覆盖全球的新闻信息采集网络，提供权威、丰富、快捷的新闻信息，大量的现场报道、独家报道和精彩的多媒体报道。作为国家的主要媒体类网站，要求网站具有很高的服务可用性，同时新华网的性质要求网站具有很高的安全性，而且新华网每天都要响应大量的用户访问，对源站服务器的性能有很高要求。

新华网采用了 ChinaCache（蓝汛）的 WebCache 和国外 CDN 服务后，网站安全性得到很大的提升，抗攻击能力大大增强，服务可用性得到有效保证，各种服务得到实时监控，网站服务器访问压力得到缓解，用户对新华网的访问

速度明显提升。

### 8.3.4 企业网站

自 1966 年创业开始, BestBuy 现已是北美消费电子产品、个人电脑、娱乐软件、设备和相关服务的顶尖特色零售商。作为一家持续发展的创新型财富 100 强公司, BestBuy 雇用了 14 万名员工, 在美国、加拿大和中国运营着 1150 家零售商店。2000 年, BestBuy.com 开始提供品种繁多的产品, 该网站提供 60 万个库存单品编号, 是商店承载能力的 20 倍, 每年接待 4 亿访问者, BestBuy 网上商店连续四年人选互联网网络零售商 50 佳。BestBuy 需要满足以下三个关键要求, 才能起到支持其品牌和目标的作用。

- 提供极佳的客户体验: 公司希望能够保证客户即使是在购物旺季也可以享受到最好的在线购物体验。
- 保护品牌形象: BestBuy 深知网站就是品牌的延伸, 它期望解决方案能够确保网站的可靠性, 无论何时何地都可以为客户提供服务。
- 降低运营成本: 由于零售商密切关注利润率, BestBuy 一直在努力避免成本超出预算。

采用 Akamai 的 CDN 服务之后, 购物车交易的速度提高了 20%。BestBuy.com 在 2007 财政年度第三季度的营业收入较 2006 财政年度第三季度提高了 30%, 2008 财政年度第三季度与 2007 财政年度同期相比提高了 65%。根据 BestBuy 调查, 有 60% 的网站客户在到商店购物前会先在网站上仔细查看所要购买的产品。CDN 技术确保网站的动态内容和应用程序运行良好, 始终处于可用状态, 提高了用户满意度和忠诚度。有了 Akamai 的帮助, BestBuy.com 能够支持高出正常流量 470% 的负载而无须扩大硬件规模, 网站每月为 1.19 亿访客提供服务, 带宽需求却下降了 52%。

### 8.3.5 云计算

作为传送网络的 CDN 和作为核心节点的云计算之间具有天然的互补性，彼此之间密不可分，其间的关系也呈现出不同的形态。

#### 1. 为云计算服务的 CDN

随着云计算的蓬勃兴起，各大 CDN 服务商也为此做好了准备，其中就有 Akamai。在 Akamai 的服务中，专门提供了针对亚马逊 EC2 的加速服务方案，服务方案的名称为“针对亚马逊 EC2 的优化服务”（英文名：Akamai Cloud Optimization for Amazon EC2）。

云计算平台通常由少数几个地方的“大数据中心”提供服务，而用户分散在全球各处，即使是在同一个国家，也需要穿越多个运营商和路由，大大降低了用户的使用体验，因此需要一个高度分布式的应用交付架构，帮助云计算提供者克服较慢的响应时间和不稳定的可用性。数据表明，即使大型数据中心位于世界上最大的网络内，用户和数据中心之间的平均距离也将超过 1500 千米。

Akamai 针对 Amazon EC2 提供的云优化服务，可以帮助使用 EC2 提供服务的企业提高性能和可用性，增强应用程序和 Web 资产的安全性，并可使用户进一步远离云基础设施。Akamai 的云优化服务可以解决云计算用户的远距离应用交付问题以及网络拥堵，从而帮助企业在其云战略中充分获益，示意图如图 8-9 所示。

#### 2. 云计算和 CDN 整合之后共同为客户服务

IBM 和 Akamai 联合推出的针对云计算加速的解决方案，旨在面向相关员工、合作伙伴及客户加速交付网络和云应用程序。该解决方案是将 IBM WebSphere 技术与 Akamai 的应用交付网络进行集成后提供给用户。

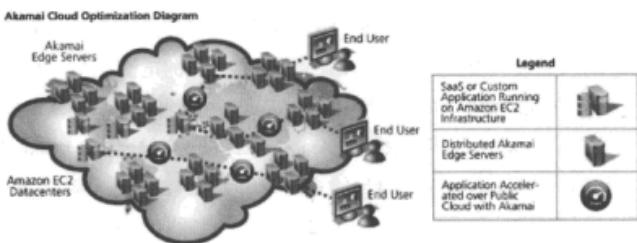


图 8-9 Akamai 针对 Amazon EC2 提供的云优化服务

该解决方案可以减少信息从数据中心通过互联网传送到公共互联网时用户所需的时间，帮助企业更快地交付应用程序，同时还可通过一个由数以万计的服务器组成的可扩展网络，为应用程序提供保护和防护，以应对流量的突然暴增或防止蓄意的恶意攻击。此服务已经在美国、澳大利亚、巴西、加拿大、新西兰以及新加坡上市，并将在 2011 年陆续在其他国家推出。

欧洲营销资源管理系统提供商 BrandMaker GmbH 认为，此服务将以更具成本效益的方式加快其软件通过云计算交付至全球客户的速度。该公司软件的当前用户超过 1500 家，业务遍及 84 个国家。该公司的客户包括 Commerzbank、Ernst & Young（安永会计事务所）、Lufthansa（汉莎航空公司）、Sara Lee、Siemens（西门子）、UBS（瑞士联合银行）以及其他许多知名公司。

Akamai 应用与网站加速部副总裁 Willie Tejada 表示：“使用云应用程序或者‘软件即服务’应用程序的企业用户常常需要忍受较低性能和可用性。通过双方的合作，IBM 和 Akamai 具备了所需的技术，足以应对这些越来越常见的应用程序交付模式所带来的挑战。这些公共和混合网络产品的面世还将帮助客户更快、更有效地推出自己的 WebSphere 应用程序，而且成本更低。”

### 3. 架构于云计算之上的 CDN 服务

亚马逊公司在 2008 年发布了自建的 CDN 服务——Amazon CloudFront。

CloudFront 是利用亚马逊自己的基础设施开发的，属于自助服务，没有定期预付款的要求，不需要签署长期合同，而是采取即用即付的定价方式。CloudFront 能与 Amazon S3 无缝协作，用户把要通过 CloudFront 服务传输的对象的原始版本存储在 S3 上。客户只需将对象放入 Amazon S3 的一个储存器（bucket）里，然后使用简单的 API 呼叫，向新推出的 CloudFront 服务注册这个储存器，接下来 CloudFront 就会返回一个域名，用于通过边缘节点读取内容。

Woot 是一家网络商店，每天为客户提供一个新产品。Woot 正在使用 CloudFront，将产品的图片传送给商店的网上顾客。Woot 公司的零售 IT 技术总监 Luke Duff 说，“我在想如何解决图片托管问题时浪费的分分秒秒。我无疑是想要图片传输的延迟短，可靠性高，不用我去处理一大堆麻烦事。幸亏有了 Amazon CloudFront，我再也不用浪费时间去管这些叫人难以忍受的烦心事了。”

Playfish 网站使用 Amazon CloudFront 来分发其社会化游戏。Playfish 首席技术官 Sami Lababidi 说，“我们已经迅速发展到了 2500 多万注册玩家，现在每个月提供 20 亿分钟以上的游戏服务。不管用户身处何地，都能通过 CloudFront 快速下载我们的游戏，可以说，CloudFront 能使用户更快速地得到我们的游戏。有了 AWS 的服务，我们在发展过程中可以保持灵活，真正使用了才支付费用，而无须签订任何长期合同或者做出任何使用承诺。”

### 8.3.6 小结

从上述案例可以看出，CDN 已经渗透到互联网的各个领域，过去使用 CDN 服务的基本上是大型网站，一方面因为 CDN 的成本较高，另一方面只有大型网站会面临较大的访问和带宽瓶颈，从而将分发服务单独进行规划和建设。随着云计算的风行，越来越多的网站和服务将基于云计算架构提供，而 CDN 也开始推出众多的自助标准化的服务，很多中小型甚至大型网站将综合利用云计算和 CDN 服务，以适应迅速增长和变化的用户市场。美国最著名的 DVD 在线

租用服务商——Netflix，2011年4月已经达到2360万用户，到2011年总数字收入达到15亿美元。然而，这么大规模的公司竟然没有自己的IT架构，其CDN由Akamai提供，而视频存储则是租用Amazon的云计算服务，公司只管理用户和财务信息等关键数据，CDN和云计算共同实现了Netflix公司的轻资产方式运营和发展。

## 8.4 典型服务商介绍

### 8.4.1 国外CDN运营商的先驱——Akamai

美国Akamai公司成立于1998年，是世界上第一家CDN网络运营商，也是全世界顶级的CDN服务商。作为内容分发网络(CDN)领域的领先者，Akamai拥有50%以上的市场份额，其客户包括Yahoo、MSNBC、Intuit、Hulu、Apple、微软等著名互联网企业，还有像Audi、NBC、Fujitsu、美国国防部和纳斯达克这样的公司和机构，服务对象覆盖网络媒体、网络音乐、网络社区、网络金融等多个领域，凡是具有网络加速需求的一般企业都是其潜在客户。

作为国际上最大的CDN服务商，Akamai公司目前在70个国家部署了超过1100个网络以及61000多个服务器，并且可以将基于定制的Web、流媒体或者Flash等内容直接传送给用户，能达到2Tb/s的速度。

Akamai采用的运营模式主要是租用电信宽带形成CDN网络，再将CDN网络带宽以电信带宽3~4倍的价格卖给或租给互联网网站运营者，而以占用的带宽和存储容量为标准向内容网站收费。

Akamai内容分发业务(CDN)能够提高企业所有基于Web内容的性能、可靠性和可扩展性。用户通过Akamai的分布式EdgePlatform从Internet的边缘传输内容可以减少应用程序的响应时间，以提高他们的体验质量。Akamai

的内容分发业务包括 Akamai 流媒体、企业边缘套件（Edgesuite Enterprise）和分发边缘套件（Edgesuite Delivery）。

Akamai 是最早的 CDN 提供商，也是目前全球最大的 CDN 运营商，成立之后一直保持高速增长。2003 年至 2007 年，Akamai 业务收入复合增长率超过了 30%，2007 年业务收入达到了 6.36 亿美元，比 2006 年增长了 2.07 亿美元，增幅达到了 48.25%。2008 年增长幅度下降，增长率只有 24%，收入约为 7.91 亿美元。Akamai 公司的签约用户同样保持了高速增长，2003 年公司的签约用户为 1126 户，2007 年已经增长到了 2645 户，复合增长率达到了 7.8%，平均每年签约用户增长 380 户。只有 2008 年受到互联网泡沫破灭的大环境影响，签约用户仅增加了不到 10%，达到 2885 户。

作为全球领先的 CDN 服务提供商，Akamai 的运营思路充分展示了一个高水平的互联网技术服务公司的整体发展思路和特点，主要包括以下几个方面。

### （1）作为 CDN 技术的引领者，具备源源不断的技术创新动力和广阔视野

1998 年，参与解决网络拥塞问题的工作小组参与了麻省理工学院年度“创业竞赛”，其 CDN 商业提案在 100 个项目中脱颖而出，成为六个优胜者之一。Akamai 获得了麻省理工学院特定知识产权的专属授权，在 1999 年 4 月推出商业服务，并宣布全球最活跃的网络公司 Yahoo! 成为其签约客户。

可以说，Akamai 自创始之时就成为 CDN 技术的奠基者，并在商业实践中不断推动技术的发展。多年来，Akamai 发布了一系列针对 CDN 的前沿研究，并将这些研究成果应用于自己的服务之中，从而成为目前尚未形成国际统一标准的 CDN 产业的标志性企业。

### （2）将自己的创新技术不断整合，最终形成真正适应客户需求的产品架构

Akamai 已经建立起完整的从技术到能力，最终形成面向客户的产品架构，具体如图 8-10 所示。

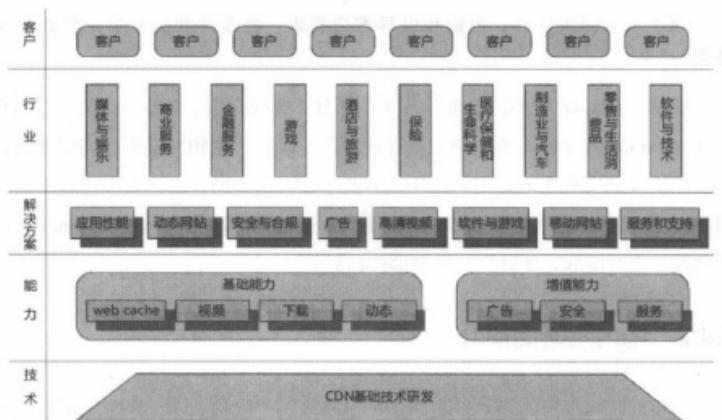


图 8-10 Akamai 产品体系架构示意

从图 8-10 可以看出, Akamai 通过持续滚动的技术研发, 将各种服务能力根据对客户的理解加以包装, 形成不同类型的解决方案; 然后根据客户的分类, 将适应客户需要的解决方案进行组合, 形成一系列客户案例。这一产品架构模型充分保证了底层架构的稳定, 尽最大可能将客户的需求进行分类和模型化, 避免某些客户的个性化需求对整体技术和运营产生影响和冲击。

### (3) 始终将客户放在第一位, 给客户提供咨询顾问式的完整解决方案

Akamai 在销售过程中, 展示了自己对于客户需求的强大理解能力, 这种理解能力源自多年的运营经验。客户在使用 CDN 服务之前, 对于 CDN 不一定具备充分的知识, 对于自己的实际需求也没有确切的描述。Akamai 所提供的咨询式服务, 可以根据客户的行业、规模以及其他信息和现有的服务客户进行对比, 从而帮助客户理顺和挖掘需求, 为客户提供真正需要的服务。这种服务模式, 一方面保证了客户需求得以全面满足, 另一方面也能挖掘出客户的一些潜在需求(如安全、广告等), 同时避免了针对客户的大规模定制开发工作量, 使得原有的工作能够得到最大限度的复用。

#### (4) 以案例为核心，组织和引导客户需求，将自身能力和客户需求完美结合

通常，Akamai 将案例分析分为 4 个部分：情况介绍、面临的挑战、目标和选择 Akamai 的理由。情况介绍是针对客户背景、行业和发展进行简要介绍；面临的挑战是客户发展过程中为什么需要 CDN 的协助；目标是描述客户需要 CDN 带来哪些改进、提升和价值；选择 Akamai 的理由则是阐述 Akamai 如何帮助客户应对挑战，帮助客户达到那些目标。

### 8.4.2 国内运营商简介

伴随互联网应用的蓬勃兴旺，CDN 已经被门户网站、网络游戏、电商平台、视频网站等视为保障运营的技术。据亚马逊统计，网站访问速度每增加 100ms 的延迟就会导致收入下降 1%。而 Shopzilla 将页面载入时间从 7s 缩减到 2s 后，网站的客户转化率就提升了 7%~12%，页面请求增加了 25%。雅虎的统计数据则显示，网站访问速度每 400ms 的延迟就会导致流量下降 5%~9%。而在提高电商网站访问速度和可用性方面，专业的 CDN 服务给网站带来的变化更为明显。近年来，如淘宝、苏宁、易购、米兰网、麦考林、麦包包等知名电子商务企业，都在和专业的服务提供商合作，以提高网站的访问速度、可用性和安全性。

国内 CDN 产业的总体规模已超过了 20 亿元，业内人士预测，未来 5 年国内 CDN 市场的规模还会扩大 5 倍。为了更好地了解国内 CDN 商业服务现状，以下对国内几家主要的 CDN 服务供应商进行简要介绍。

#### 1. 中国电信

中国电信拥有世界上规模最大的中文互联网，其 CDN 依托于网络优势，同时辅助以多年的专业化运维经验和全国统一的运维管理体系，为用户提供高质量的网络保障。

中国电信自 2004 年开始建设 CDN，在 CDN 的研发和建设过程中，采用自主研发建设与合作相结合的方式，在保持自身网络技术优势的同时，迅速扩大规模和引进新技术，成为国内 CDN 市场的新兴强大力量。到目前为止，中国电信 CDN 已经建成 79 个节点，网络容量达到 1TB 规模。建设初期主要用于自营的互联星空网站和 IPTV 等业务的分发服务，随着网络不断扩容，能力也获得提升，逐渐对外提供服务。目前，中国电信 CDN 能够提供流媒体加速（包括点播和直播）、网页加速、下载加速和应用加速等各种服务，主要用于加速互联网网站在中国电信网络范围内的分发服务。

中国电信 CDN 网络已为奇艺、乐视、凤凰等多家网站提供 CDN 服务，作为 2010 年上海世博会全球合作伙伴，为上海世博会官方网站提供了 CDN 全程保障服务。

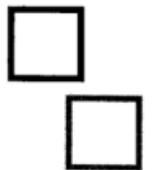
## 2. ChinaCache（蓝汛）

ChinaCache（蓝汛）公司成立于 1998 年，是中国第一家专业 CDN 服务提供商。2000 年 ChinaCache 获得信息产业部 CDN 服务许可。截至 2011 年上半年，ChinaCache 在全国 113 个城市的机房中部署了节点，同时管理 400GB 带宽资源。其 CDN 网络覆盖国内各大运营商，其客户包括搜狐、网易、eBay、易趣、淘宝、慧聪、携程、新华网等几十家，另外还有“国网”等政府网站，以及诺基亚、宝马汽车、可口可乐等传统企业网站。2006~2009 年，蓝汛净收入从 7000 万元增长到 2.72 亿元；2010 年全年收入 4.034 亿元（约合 6110 万美元），比 2009 年增长 48.1%，蓝汛的最大客户变成了中国移动，其子公司购买了大量蓝汛的移动互联网服务，占蓝汛 2010 年上半年总营收的 12.8%。

2010 年 10 月 1 日，蓝汛通信（ChinaCache International，即 CCIH）在纳斯达克上市，融资 8400 万美元。2011 年初，美国中国概念股网站 China Analyst 评出了 10 大最具成长性的中国概念股，其排名依据是华尔街分析师对公司未来 3~5 年每股赢利增长率的平均预测值。其中，蓝汛通信以长期每股赢利增长率 45% 的成绩排名第五。

### 3. 网宿科技

网宿科技成立于 2000 年 1 月 26 日，其前身为上海网宿科技发展有限公司，具备全国 IDC、ISP 经营许可证，可向用户提供 Web 加速、高级应用加速、流媒体加速和大文件下载加速等服务。在运营 CDN 业务的同时，网宿科技的主营业务还包括 IDC 相关业务，提供分布式托管、全程运维管理服务、网络安全等专业服务，是国内最早开展 IDC 和 CDN 业务的厂商之一。2010 年网宿科技总收入为 3.623 亿元，比 2009 年增长 26.22%，其中 CDN 业务收入 1.88 亿元，比 2009 年增长 38.11%。截至 2011 年上半年，网宿科技包含 IDC 和 CDN 在内的带宽储备已达 900GB。网宿科技 CDN 包含 200 多个节点，可承载带宽（峰值）高达 600GB，其网络覆盖了国内的各大互联网运营商，为全国近 2000 家客户提供着高效率的 7 天 24 小时、全年 365 日的专业运维服务。



## 第9章 CDN发展展望

- 9.1 新时代对CDN的要求
- 9.2 CDN技术发展趋势
- 9.3 CDN与云计算
- 9.4 CDN与P2P
- 9.5 CDN的商业服务发展趋势



CDN 是介于业务平台与基础网络之间的“网上网”，对业务变化的响应远比基础网络迅速。有新的加速需求，就会催生新的 CDN 技术，甚至周边技术变革也会对 CDN 产生一定推动作用。随着移动互联网、三网融合、云计算三大产业热点接踵而至，CDN 也迎来了新一轮技术演进风潮。

## 9.1 新时代对 CDN 的要求

CDN 是伴随着互联网的成长而逐步发展起来的，从前如此，未来也将如此。CDN 总是以满足应用需求为目标，再以后反馈方式促进应用优化，而不大可能超越应用需求的发展。我们可以把 CDN 的发展简单划分为三个阶段，如图 9-1 所示。

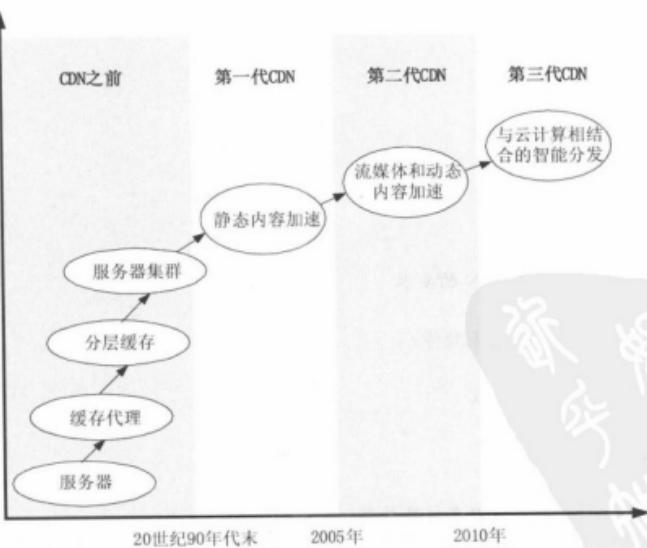


图 9-1 CDN 演进路线图

第一代 CDN 是以静态内容加速为主的，当时 Internet 流量主要是由一些静态文本和图片文件构成，这些文件的数据量都比较小。为实现高效的内容分发，以 Akamai 为代表的 CDN 服务提供商通过高度分布式的服务器部署方式来构建 CDN，在运营商的数据中心中部署服务节点，这样可以将用户请求的内容分配到离用户最近的服务节点上。然后采用基于 DNS 的请求路由机制，使用户从最近的服务节点上获取信息。

第一代 CDN 能够很好地实现 Internet 中静态内容的分发，解决“最后一公里”的问题，但时间推进到 21 世纪初，各种多媒体内容（如音频、视频等）开始成为 Internet 上的主要流量，第一代 CDN 技术被业务驱动迅速发展到第二代 CDN。第二代 CDN 融合了流媒体数据采集、压缩、编码、存储、传送、播放等技术，形成了以流式分发为核心的分发技术体系。当然此时 CDN 需要适应的还有以 Web 2.0 应用为代表的具有强烈交互性、实时处理需求的动态内容。但总的来说，第二代 CDN 还是以传统的资源出租方式提供服务为主。到 2010 年，云计算几乎是以迅雷不及掩耳之势席卷了整个 IT 界。云计算带来了网络架构、设计和服务提供的新思路，势必对 CDN 产生重大影响。一方面，新的 CDN 服务能够以云计算典型的 pay-as-you-go 方式提供，如亚马逊的 CloudFront 和 MetaCDN，将分发服务从内容进一步拓展到用户在线业务的各个方面。这样不仅能为客户带来利益，CDN 提供商也能通过按使用收费的模式配置资源，避免大量基础设施的投资。另一方面，CDN 服务提供商也可以利用云计算技术重新构建自己的 CDN 平台，获得成本、性能优势。

在国内，CDN 技术正处于第二代向第三代的过渡期。云计算当然是一个重要的驱动力，但 CDN 演进的最主要驱动力还是源自业务需求。从 2008 年 3G 牌照发放后，国内开始进入移动互联网时代，形成新一波发展浪潮，移动终端上的业务提供前所未有的繁荣；而随着“三网融合”进入实质性推进期，迅速成为一个新的产业热点。移动互联网和三网融合业务，势必对 CDN 提出更高要求和更多样的需求，CDN 的重要作用也日益凸显。

### 1. 需要适应的流量更大、性能要求更强

据预测，全球互联网总流量将在 2012 年达到 500 EB ( $1E=10^9G$ )，其中视频流量将占 90%以上。因此，未来高清视频内容传送能力将成为对 CDN 网络的基本要求。另一方面，由于 CDN 能够将网络流量最大程度压制在接入层，避免大量重复内容流量进入骨干网，能够大大缓解运营商基础 IP 网络的扩容压力，因此 CDN 网络建设逐渐成为传统运营商的自身需求。运营商 CDN 将比传统用 CDN 规模更大，与基础网络协同性更强。

### 2. 需要管理的内容更多，存储成本亟待降低

导致网络中内容呈海量增长的原因主要有三个：第一，网络视频内容本身正在朝高清、3D 方向发展，原来一部网络电影只有几百 MB，现在要几个 GB 到几十个 GB；第二，用户需要的内容不再高度集中，而是呈现典型的长尾趋势，冷片不冷，热片不热。第三，随着终端形态的多样化，尤其是移动终端的加入，要求同一个视频内容以多种不同格式的文件副本存在。这三个原因使得 CDN 网络需要管理的内容量以惊人的速度增长，对 CDN 来说存储容量、文件管理难度都增大很多。如果沿用传统的整片存储方式，全网存储成本将是难以承受的。目前业界普遍引入了文件分片技术，推动了 CDN 文件管理从本地管理、本地缓存为主向全网跨域调度的发展。

### 3. 需要更智能的调度，调度策略持续优化

2010 年以前，大多数用户用电脑通过固定宽带上网，而今天，用户在各种各样的终端上使用互联网，包括智能手机、iPad、电子书、在线播放器等。台式机、笔记本和移动终端的界限越来越模糊，用户需要随时随地体验业务，同一个业务在不同终端上呈现。以上这些业务的新变化要求 CDN 能够根据用户身份、接入方式、业务跨平台等要求进行调度，调度策略更加综合化和灵活。

## 9.2 CDN 技术发展趋势

为适应整个产业的发展潮流，CDN 需要在技术上做进一步提升，“智能化”是未来 CDN 网络的核心特征。

### 1. 存储智能化

这里所说的存储是指内容在 CDN 服务节点中的存储方式。传统 CDN（相对于智能化 CDN 而言）采用分级整片存储方式，如果有 10000 部影片，那么全部边缘节点都要按整片存储其中 20%，即 2000 部，各个区域中心节点（或者叫二级节点）存储其中 80%，即 8000 部，而中心节点通常会存储全部影片，以便下级节点回源调用（此比例可以通过管理系统进行配置）。

当用户对视频业务需求向着高清、海量、长尾方向发展，平均单片大小和片库片量都会暴增几倍甚至几十倍，这将带来巨大的存储空间需求，按照传统方式组织 CDN 存储将面临高昂的成本压力。因此，CDN 的存储已经向以文件切片为基础的全网共享存储发展。这种方法是把一个整片切成 N 个小块，按照用户访问热度分散存储在不同节点上，用户访问时，由提供服务的边缘节点设备将用户所需文件从其他节点调用过来。除了热度极高的文件分片，其他分片都不必存放在边缘节点上，也不必全网重复存储。

CDN 存储智能化的关键技术点在于文件分片热度的判断策略和跨节点快速文件调度。

### 2. 调度智能化

传统 CDN 的调度依据是用户 IP 就近性、设备负载情况、内容命中情况。而为了满足新的业务需求，智能化 CDN 要求在调度依据中增加以用户使用情况为中心的依据，包括：Who（谁在用）、Where（在哪里用）、When（什么时间用）、What（要用什么）、How（怎样用），这里包含了对用户身份的认

证，对用户使用业务记录的分析，对用户所在网络情况的判定等，只有具备了这些分析能力，才能帮助业务平台实现智能的业务提供。比如跨屏的内容连续播放，用户偏好推荐，不同接入带宽适配等。

CDN 调度智能化的关键技术点在于以用户为中心的策略信息库的建立，CDN 应能借助外部认证系统获得用户身份信息，从而将业务使用环境、使用记录围绕用户身份组织成调度策略。

### 3. 组网智能化

前面提到过，不同的业务能力对 CDN 的技术要求各不相同，这主要是由各业务类别对 CDN 服务设备的要求差别比较大引起的。比如流媒体加速业务，其特征是大文件高缓存，同时要在应用层进行传送质量优化；而网页加速业务，其特征是小文件高吞吐。业务特性不同，使得设备硬件和软件配置都不一样。也就是说，不会有一张 CDN 网络适用于所有业务的承载加速要求。今天满足要求的 CDN，也许明天就不满足要求了。

因此，智能化 CDN 会采用统一架构和统一管理下的分业务能力垂直组网方案，即管理系统和全局调度系统统一设置，对下接口标准化，而不同业务能力可选用不同技术体制、不同厂家的方案，各业务能力之间不做水平互通。这样，当需要扩展新的业务能力时，只需要增加新的垂直业务子网，完全不会影响到已有业务能力的提供。

另外，当前虚拟机组网技术已经成熟并且大规模商用，CDN 网络也可以尝试采用虚拟机技术进行组网。在统一的虚拟机资源池中，根据业务能力需求进行虚拟机配置和调用。正常情况下，调用和配置一个虚拟机子网只需一天时间，业务关闭时可同步释放虚拟机资源。因此，这一方式可以使业务上线速度大大提高，也极大降低业务试错成本，更适应当前日新月异的互联网生态环境。

### 4. 数据分析智能化

CDN 中记录着大量业务使用数据，比如对视频业务来说，哪些片子的被访

问量最大，片子的哪一段访问量最大，访问时间和区域集中在哪里等，这些信息对业务智能化提供非常重要。CDN 智能化对网内的数据分析提出了新的要求，一是要能够将业务数据对应到用户身份，二是能够在 CDN 内部完成一定的用户偏好、使用行为分析整理工作，这些分析结果一方面用于 CDN 的智能化调度，另一方面要能够以标准化接口提供给外部业务平台，包括使用 CDN 的业务平台和其他需要使用这些数据的业务平台。

## 9.3 CDN 与云计算

### 9.3.1 云计算——第三次 IT 革命

云计算由全球最大的搜索引擎服务提供商 Google 在 2007 年首先提出，当时的主要目的是充分利用已有投资，以虚拟化技术作为 IT 资源的整合手段。也许谁也没有料到，这样一个“新瓶装旧酒”的技术会在一夜之间掀起第三次 IT 革命浪潮。但事实是，它确实迅速蔓延成燎原之势，互联网公司、IT 厂商和电信运营商纷纷投入。

20 世纪 90 年代，HTML 出现，Web 发展为主流的信息交互方式，互联网开始进入商业化时代，人们开始更多地通过浏览器来获取和分享信息，即 B/S 模式。2001 年后，虽然第一次互联网泡沫破裂，但在这一时期大量投资建设的宽带基础设施和风险投资、股市等进入的资金为整个互联网的再次发展埋下了种子。到 2004 年，Web 2.0 技术得到发展，进一步改变了信息的产生和交互模式——普通人也可以参与信息的制造和传播，同时信息的交互形式更为丰富，出现了博客、SNS、微博等新型应用。互联网用户规模不断翻番，数据量呈现爆炸性增长，为满足用户需求和业务运营需要，大型互联网业务提供商不断建设新的数据中心，用来进行海量数据的存储和处理。像 Google 这样的搜索公司，

搜索引擎致力于收集全球数据，因此它们必须拥有全球规模最大的数据中心，同时也要求其能够统一管理和调度这些庞大的、分布的数据资源。在这个过程中分布式技术逐渐成熟，更多的应用可以基于这些技术来提供，人们通过浏览器得到的东西不再仅仅来自服务器，也可以来自几千甚至几万台服务器组成的庞大集群，B/S 模式开始走向 B/C（Cloud，即庞大的服务器集群），即所谓的云计算。

虽然在云计算概念提出之前，Google、Amazon、Salesforce、Zoho 等就已经在技术和服务上进行了很多实践，但这个概念的提出还是在整个信息行业掀起了轩然大波。人们开始从 Google、Amazon、Salesforce 的实践中看到了 IT 未来发展的“雏形”，并且认识到云计算并非简单的模式变化，它带来的将是对 IT 技术、商业模式和运营模式的全面变革，也将对整个信息产业产生难以预估的影响。

云计算是一种典型的破坏性创新，它利用技术进步效应，从 IT 产业不断高涨的成本投入和运营薄弱环节切入，重构传统的 IT 市场结构，创新或优化 IT 产品和服务，具体表现在以下三个方面。

1. 由于云计算技术和服务模式的应用，使得 IT 应用、产品或服务的获取和使用变得更为简便。在云计算的应用环境下，用户只需一台有浏览器的终端，连上网络，即可使用云计算服务。计算资源、开发平台以及软件应用的获取和使用变得更为简便。
2. 由于云计算技术的引入，使得 IT 应用、产品、服务的成本显著降低。虽然云计算暂时难以达到与本地计算模式同样的稳定性、时效性以及私密性，但是依靠成本上的优势已经吸引了大量的中小企业以及部分大型企业的关注，逐渐扩大其影响力，为市场所接受。
3. 云计算对传统 IT 产业链产生破坏性的冲击。云计算使得 IT 产业格局发生了变化，一方面使旧的 IT 产业链加速整合，另一方面对传统 IT 产业链

中的企业带来了颠覆性的挑战，新的重量级参与者加入产业链的竞争。

哈佛大学的 Clayton Christensen 理论研究表明，在如今迅猛演变的商业世界里，通过破坏性创新创建的新业务成功概率比通过传统的维持性创新要高 10 倍以上。换句话说，云计算已不仅局限于技术革命，它在未来十年左右将颠覆现有的产业格局，创造新的产业链和更大的产业体系，不断开创新的商业模式，并革新传统的 IT 运营体系。与亨利·福特开创自动化汽车生产线对工业的影响一样，云计算的出现也将改变整个 IT 产业的面貌，带来 IT 业的产业革命。

### 9.3.2 CDN 是云计算吗

在笔者从事云计算相关研究的过程中，经常被问到 CDN 与云计算的关系：CDN 是不是一种云计算？是属于 IaaS，还是 PaaS，抑或 SaaS？要回答这个问题，我们需要从商业模式和技术特征两个角度切入，看看 CDN 是否具备云计算的主要特征。

首先，从商业服务模式上看，CDN 是否是一种云计算？

图 9-2 描述的是传统 IT 建设方式与资源需求之间的矛盾。由于资源的需求

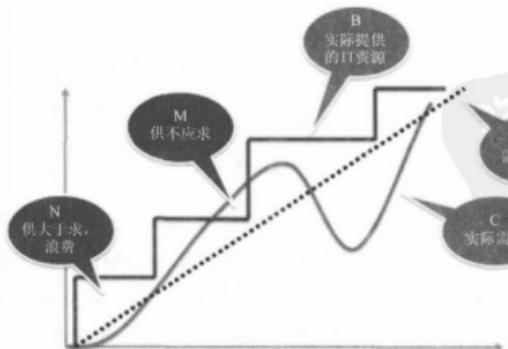


图 9-2 传统 IT 建设方式与资源需求之间的矛盾

是线性增长的，而资源的建设却是阶梯式发展的，因此大多数时间使用者都会处在两种不理想状态中，要么资源供大于求，形成浪费，要么资源供不应求，影响业务运行。

对于互联网公司来说，这种不理想状态带来的问题更为严重。互联网业务的一个显著特点就是传播速度快，容易形成热点效应（如图 9-3 所示），热点时间内的页面访问尖峰是每个网站追求的目标，如果在这段时间资源能力跟不上，会导致大量的用户访问失败，花很多时间、精力、财力吸引来的用户就会流失掉，美梦就会变成噩梦。但是为了不可预计的访问流量，去购置大量的服务器和带宽资源，平时都是闲置状态，这对很多互联网公司，特别是那些互联网创业型公司，在资金等方面是不可接受的。

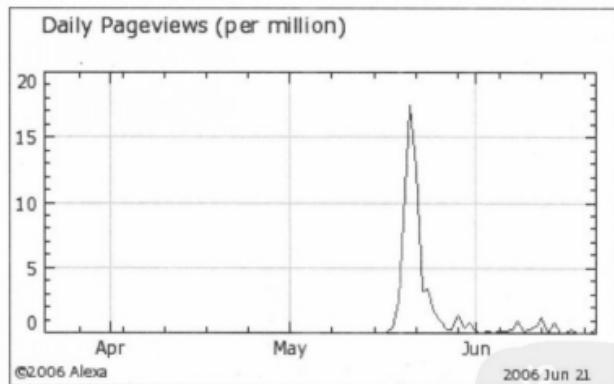


图 9-3 互联网热点效应

对于使用者来说，云计算最典型的特征和最直观的好处就是弹性伸缩和按需提供。客户可以根据自身的需要租用相应的资源，按使用量付费。对云计算资源的维护工作由云服务公司提供，大大简化了客户在基础设施维护上投入的工作量，相应的业务响应时间也大幅缩减。对使用者来说，IT 基础设施的投资，特别是初期投资大幅减少，降低了运营门槛。

从商业模式角度来说，CDN 完全符合云计算的按需提供资源的特征，它将网络分发服务这一专业工作独立出来，用户只需简单地对接调试就可使用该服务，并根据客户网站的并发访问量计费，按需扩充资源服务能力。换句话说，CDN 和云计算都促进了 IT 产业的分工细化和协同，降低了互联网公司的运营门槛，使得互联网公司可以专注于其核心业务的发展。

CDN 与云计算还有一个共性就是规模经济性。一方面是因为资源的集中购置、维护和管理，使采购成本和单位设备的维护成本降低。同时部署大规模基础资源也带动了相关绿色节能技术的发展；另一方面，资源的统计复用促进了整个系统设备使用率的提高，从而节省成本。图 9-4 是某搜索引擎在日本和英国的访问量在一天时间内在日本和英国的访问量，可以看到由于时区的原因，其访问流量形成峰谷互补，从而平滑了整体的利用率。图 9-5 是 Alexa 对美国前 4 名的购物网站和网上报税网站的统计图，由于假期效应，网络购物网站的页面访问高峰在圣诞节前后，而报税网站在每年的 1~3 月，因此多个不同行业的网站服务叠加之后，也会形成峰谷互补效应。

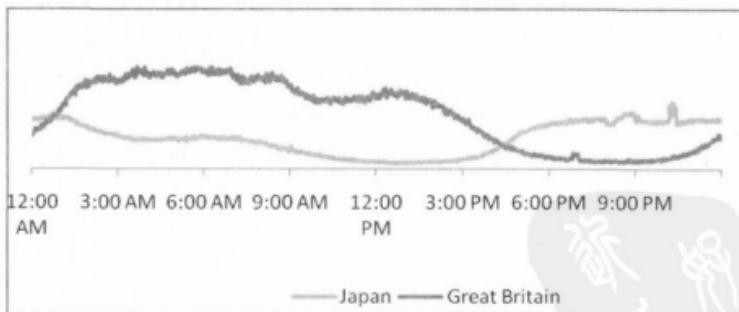


图 9-4 某搜索引擎在日本和英国的访问量形成峰谷叠加

完成商业模式角度的分析，我们再从技术角度来分析，看 CDN 属于哪种类型的云计算？

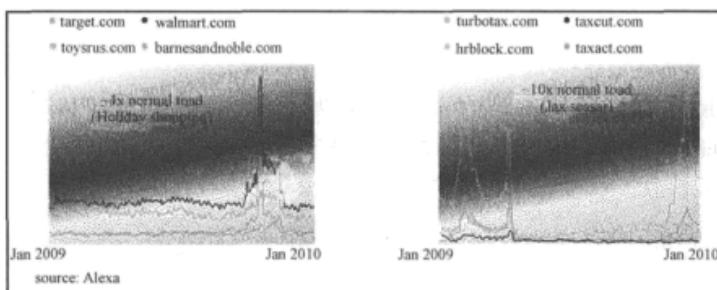


图 9-5 Alexa 对美国前 4 名的购物网站和网上报税网站的统计图

按照业界普遍认识和理解，云计算主要分为两大技术流派：分布式集群架构和虚拟化架构。它们各有优势和特点，也存在较大差异，共同支撑云计算的技术本质。

**分布式集群架构**的主要使用者和倡导者是互联网公司，这些公司的业务数据量随着业务发展飞速膨胀，海量数据的存储、查询、处理对基础设施和软件提出了极高的要求。采用传统的 IT 架构和技术不仅成本很高，而且不能满足快速扩展需要，分布式集群架构正是适用于解决这些问题的。因此从本质来看，分布式集群架构满足的是一种新的 IT 需求。以 Google 为例，它的分布式集群架构是在之前并行处理、分布式计算和网格计算技术的基础上提出的，是对原有架构的一种完善，与传统的分布式计算架构主要区别有以下几点。

1. 节点是同构的（网格的节点则是异构的），例如 Google 的所有服务器节点都是基于兼容的 Linux 系统，硬件结构上也是一致的，这样大大降低了分布式计算技术的复杂度。
2. 节点之间通过高速网络互联，相比于通过普通互联网互联的网格技术，分布式集群节点之间的数据复制效率更高，而且更有利于高效的任务调度和数据传递（例如，可以将任务调度到跨区域的多个数据中心进行计算，输出的结果再通过高速网络返回，而不需要进行远程大数据量传送）。

3. 假设节点是不可靠的，通过软件方式来保证系统的可靠性，例如，GFS 对于每份数据都会至少存放在三个以上的节点。

4. 将信令流和数据流分离，解决控制节点的瓶颈问题。例如 GFS 一般只需要采用单个主控节点，就可以控制几百个，甚至几千个数据节点。

5. 假设数据文件大部分是比较大的文件，因此可以将数据文件切分为大的数据块（例如 Google 的 GFS 切分为 64MB）。这样的好处是元数据量比较少，可以保证主控节点的内存空间足以保存所有数据节点信息，同时减少对数据节点的访问压力。当然，由于互联网中也有大量的小数据文件，目前不少公司也在探索新的存储系统。

从以上比较可以看出，新的分布式架构对存储模型和计算模型进行了一系列改进，更适用于互联网应用，同时也能够保证很高的安全性（可靠性）、可用性和吞吐能力。同时，应该认识到，每个互联网公司的分布式集群架构都是面向某种应用需求而设计的，它不是一个通用的、完美的计算架构，不能支持所有的计算和存储需求，也不可能适用于所有应用场景。

虚拟化架构主要由传统 IT 公司（包括软件和硬件公司）提出，目的是提升资源利用率、降低数据中心运营成本、加快新业务部署速度、提升整个 IT 系统的资源分配和维护的灵活度，这些都是以现有 IT 能力为基础的优化，而不是为了满足新的 IT 业务需求。虚拟化技术最早是在 20 世纪 50 年代末出现的，当时只是大型机上的一种资源共享方式。真正具有历史意义的是 20 世纪 90 年代出现在 X86 架构上的虚拟化技术。

虚拟化技术简单的理解就是在同一硬件系统上可以同时运行多个软件系统（包括操作系统和应用软件），不同软件系统由一个调度器（称为虚拟机监视器，VMM）来调度使用底层的硬件资源，从而可以实现多个系统同时分享同一个硬件资源。VMM 实际上是一种新的软件架构，它还只是单个服务器上的一种新型软件架构，专业人士喜欢把它称为“承载操作系统”的操作系统。随着云计

算技术的出现，IT 公司（包括 IBM、Oracle、VMWare、微软等）在 VMM 的基础上增加了高一级的管理平台，可以统一管理多个服务器上的 VMM，以便决定在哪台服务器上部署虚拟机，实现虚拟机伸缩以及在 VMM 之间完成虚拟机迁移，这样就形成了基于虚拟化的云计算架构，并在最近几年获得了快速的发展。

不论分布式集群架构，还是虚拟化架构，云计算的两大技术流派都代表了 IT 技术发展的趋势。

1. 硬件与软件分离，硬件资源可高度整合共享，任务/应用可跨硬件调度，总体上降低了 TOC ( Total Ownership Cost，总体拥有成本)。虚拟化架构让异构软件系统可以运行在虚拟化的统一资源池上，从而支持更多的应用系统运行，充分挖掘已有资源潜力；而分布式架构则让计算任务和存储对象尽可能切分为最小的粒度，充分“填充”到分布式的资源空间中，尽可能使用最便宜的节点。
2. 资源聚合放大，解决 IT/互联网的数据增长带来的存储处理压力。虚拟化架构通过将各个节点的空闲资源聚合重新利用，从而“放大”了现有资源能力，支撑数据增长带来的存储和处理需求；分布式架构则通过实现大量分布式节点的动态组合，支撑了海量数据的存储与计算。

如果将 CDN 与这两种云计算架构相对比，可以认为 CDN 具备分布式集群架构的基本特征，是面向一种单一类别的任务而设计的系统。首先，CDN 的每个节点基本上都是同构的，承担的任务、组成方式基本相同，差别只在于处理能力不同；其次，CDN 在设计时，假定每个节点都是不可靠的，用充足的冗余保护机制，任何单点失效都不会影响整个系统的运行；最后，CDN 在系统设计上就采用了业务流和数据流分离的结构。

从对外提供的服务能力上来看，我们认为 CDN 更接近于一种 IaaS 服务：一是因为 CDN 解决的是网络基础设施服务的问题，帮助用户建设基础的服务器和存储设备，实现内容的分发，保证用户服务质量；二是因为 CDN 的服务

主要是按资源使用量收费，如带宽、流量、并发连接等。事实上，目前大多数 IaaS 服务提供商正是同时提供 CDN 服务的，像 Amazon 公司的 CloudFront 就是一种 CDN 服务。

### 9.3.3 CDN 与云计算技术的结合

应该说，云计算的异军突起给 CDN 带来了新的发展机遇。人们在关注云计算的同时，也把眼光投向了它的相关领域，CDN 算是其中之一。CDN 可以借鉴很多云计算领域的新技术，而当云计算技术在 CDN 领域获得新的应用时，也能极大地推动其自身发展。

随着 CDN 的应用场景不断丰富，网络规模不断扩大，如何对系统开展有效管理已经成为 CDN 领域的重要议题。在实际建设和部署中，CDN 需要强大的容错、自恢复能力，在确保系统稳定性的基础上需要具有按需缩放、自动维护能力。这让人很自然地想到云计算核心技术之一：服务器虚拟化。如果能够将服务器虚拟化技术应用于 CDN 建设，将使 CDN 在系统管理和服务能力上产生质的飞跃。服务器虚拟化技术通过对物理服务器的处理器、内存、输入/输出（包括磁盘、网络等）等设备资源进行虚拟化，进而将这些虚拟资源组合为多台彼此隔离的虚拟机，能够减少服务器占用空间、降低购买软硬件设备的成本、提升系统安全性和容灾能力、节省能源和维护成本，特别是在优化系统管理复杂度方面更具优势。在 CDN 系统部署中引入虚拟化技术，可以带来更灵活的资源配置和更优化的部署方法，能够根据用户的需求快速地调整服务器的处理能力和设备数量。此外，虚拟化技术支持物理服务器之间的虚拟机迁移，能够更好地满足业务运行的持续性，避免硬件故障造成的服务失效。更关键的是，虚拟机的运维（例如部署、迁移等）可以通过自动化的管理工具自行完成，从而大大降低管理难度。例如，在 CDN 边缘节点中部署服务器虚拟化平台，在访问流量突然增加的情况下，虚拟化管理工具就能够根据此前设定的管理策略自动地调整资源配置，创建新的虚拟机并部署操作系统和应用软件，进而

## CDN 技术详解

将其接入原有系统中运行，达到扩大节点处理能力，及时缓解系统压力的目的。

另一项可应用于 CDN 的云计算技术是云存储。云存储以海量低成本存储见长，同时提供优于普通存储方案的数据安全性，这为 CDN 解决高清视频等带来的存储成本问题提供了一个新的选择。当前主流的云存储技术普遍采用基于 X86 架构的存储服务器搭建分布式集群，而非使用传统的专用存储设备，具有明显成本优势和较好的设备兼容性。云存储方案中的存储管理可以独立于 CDN 系统而存在，可以对数据访问过程进行带外控制，管理通道和数据通道彼此隔离的设计在支持大文件存取时具有更优的访问效率。在存储过程中，一个文件可以被分为多个片段，并在系统中不同存储节点上保存多个副本，具有更好的可靠性。用户对文件的访问，可以同时和存储有文件片段的多个存储节点建立连接，并行地对文件的各个部分进行读写操作，系统性能极高。云存储系统的规模扩展非常便捷，在系统容量不能满足存储需求时，只需在新增加的服务器节点上部署分布式存储中间件并进行必要的网络配置，即可快速地将该节点纳入原有云存储集群的管理体系中。云存储技术非常适合于大文件的读取密集型访问，这也正是 CDN 的主要应用场景。但应该注意的是，一些云存储系统为了提高大数据访问效率，并没有做到严格的 POSIX 兼容，而且很多都只提供了面向对象的访问接口，与此前 CDN 系统广泛使用的 FTP 等标准文件访问接口有所差异，因此应用于 CDN 系统就需要进行必要的接口定义和数据转换，以确保访问过程的一致性。

商用 CDN 系统的日志处理要求非常高，除了正常日志记录，还要求能综合运用统计分析、数据挖掘等方法及时捕捉用户行为特征和发现用户目标需求，从而进行有针对性的内容管理和资源调度。在这个过程中，有大量的非结构化数据需要采集和处理，新型的基于计算集群的分布式计算模型能够针对海量大规模非结构化数据提供高性能的并行处理能力，从而很好地改善 CDN 的智能性。比如 Google 公司提出的 MapReduce 方案，受到函数式编程语言的启发，将大规模数据处理作业拆分成若干个可独立运行的 Map 任务，分配到集群

不同的服务器上去执行，生成特定格式的中间文件，再由若干个 Reduce 任务合并这些中间文件获得最后的输出文件。在 MapReduce 的实现架构中，计算服务器与存储服务器的部署位置相重合，使得计算和数据能够尽可能地贴近，避免了将大量数据从存储设备传往计算服务器的传输延迟。这样，我们想到可以在 CDN 系统的边缘节点上部署分布式日志分析系统，加速用户数据处理并提供快速反馈。同时，计算功能的实现可以复用分布式云存储集群，使得系统具有更强的海量数据处理支持和更优的系统扩展性。

云计算本身是一种互联网服务，强调的是由云数据中心集中提供资源，而弱化终端的软硬件能力。在使用云计算服务后，所有的内容都将被保存在远程的数据中心，用户通过网络远程访问。因此，类似于其他互联网服务，云计算服务也可以采用 CDN 系统作为加速手段，以获得更高的访问效率和更优的用户体验。

## 9.4 CDN 与 P2P

CDN 与 P2P 是经常被拿来进行对比的两种技术，原因是两者都是数字媒体文件的重要传送机制，大部分用户会同时使用到这两种机制，从而产生体验对比。应该说，两种技术各有所长，在不同场景和需求下具有另一种技术所不具备的优势。然而，技术的发展总是盛衰相继，分分合合，并没有非此即彼的绝对性。我们在这里以开放的、探索性的思路讨论 P2P 技术，也讨论 CDN 与 P2P 相融合的技术可行性。

### 9.4.1 P2P 技术概述

P2P 是 Peer to Peer（对等网络）的简称，P2P 网络中的各个节点则被称为

Peer(对等体)。当前业界并没有对 P2P 这个概念的定义达成一致性意见, Intel 公司和 IBM 公司给出的定义是比较有代表性的。其中, Intel 将 P2P 定义为“通过系统间的直接交换达成计算机资源与信息的共享”的系统,这些资源与服务包括信息交换、处理器时钟、缓存和磁盘空间等。IBM 则提出更具广泛性含义的定义:P2P 是由若干互联协作的计算机构成的系统,该系统需要具有以下一项或多项特征。

- 系统依存于边缘化(非中央式服务器)设备的主动协作,每个成员直接从其他成员而不是从服务器的参与中受益。
- 系统中成员同时扮演服务器与客户端的角色。
- 系统应用的用户能够意识到彼此的存在而构成一个虚拟或实际的群体。

无论是基于哪种定义,从本质上讲,P2P 技术是一种用于不同计算机用户之间、不经过服务器直接交换数据或服务的技术,如图 9-6 所示的虚线体现了用户之间的直接通信情况。P2P 不同于传统的 Client/Server 模式,网络中的每个 Peer 都具有相同的地位,并且同时具备客户端和服务端的双重特性,能够同时作为服务使用者和服务提供者。

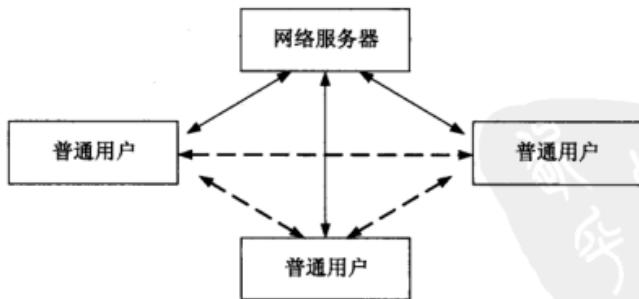


图 9-6 P2P 网络通信方式

早在 20 世纪 70 年代, P2P 技术就已经出现, 典型代表是 USENET 和 FidoNet 这两个分散、分布的信息交换系统, 而 P2P 技术的真正大规模应用要归功于文件交换软件 Napster 的推广。P2P 技术在国内的兴起从 2003 年下半年开始, 以 BT ( BitTorrent ) 为代表的 P2P 技术急速升温, 用户数量飞速发展, 在文件交换、分布式计算、协同工作、分布式搜索和电子商务等领域有着广泛的应用。

BT 系统中的文件发布者会根据要发布的文件生成一个.torrent 文件, 即种子文件, 其中包含 Tracker 信息和文件信息两部分。其中, Tracker 信息主要是 BT 下载中需要用到的 Tracker 服务器的地址和针对 Tracker 服务器的设置; 文件信息是针对目标文件计算而成, 并依据 BitTorrent 协议的相关规则进行编码的信息。BT 的主要原理是把将被下载的文件虚拟分成大小相等的块, 其中块大小必须为 2K 的整数次方, 并把每个块的索引信息和 Hash 验证码写入.torrent 文件中。下载时, BT 客户端首先解析.torrent 文件得到 Tracker 地址, 然后连接 Tracker 服务器。Tracker 服务器回应下载者的请求, 为下载者提供其他下载者(包括发布者)的 IP。下载者再联系其他下载者, 根据.torrent 文件, 两者分别对对方告知自己已经有的块, 然后交换对方没有的数据。此时不需要其他服务器参与, 分散了单个线路上的数据流量, 因此减轻了服务器负担。典型的 BT 服务流程如图 9-7 所示。

- ①用户通过网上搜索或种子发布网站, 找到 A.torrent 文件。
- ②单击下载 A.torrent, 并自动启动 BT 客户端软件, 此时用户终端成为一个 Peer。
- ③确认下载后, BT 客户端软件连接 Tracker, 上报本机信息(同共享用户信息)和要共享的文件的信息。

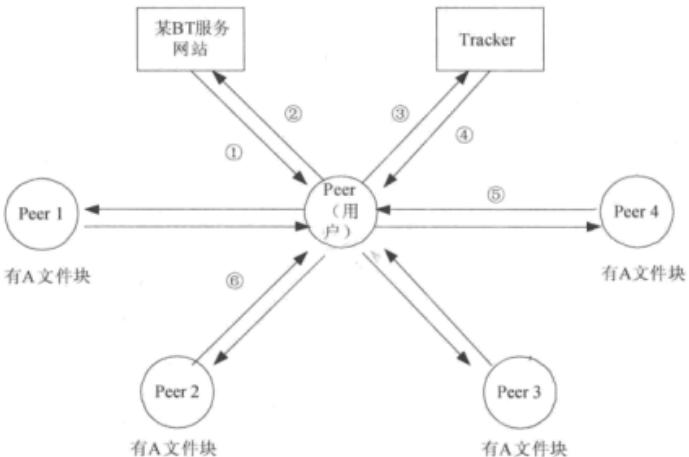


图 9-7 BT 典型服务流程图

④Tracker 向本机发送信息包括：与 Tracker 通信的时间间隔，拥有/正在下载 A.torrent 的其他 Peer 的 ID、IP 地址和端口号。

⑤与上一步骤中获知的其他 Peer 建立连接。

⑥从连接成功的 Peer 处下载数据（本机完成文件下载后，可以提供给其他 Peer 下载）。

上述流程中，第 1~2 步，用户通过上网搜索等方式，获取.torrent 文件，进入内容入口点；第 3~4 步，Peer 与 Tracker 服务器连接，获取连接列表；第 5~6 步是数据传输过程。可以看出，在数据连接和传输阶段，内容都分布在 Peer 客户端的存储空间中，用户之间直接进行点对点连接，交换内容分片，实现了 P2P 传输。

从某种意义上说，P2P 计算可以说是一种向传统互联网技术的回归，体现

了因特网的本质，因为因特网最初的设计目标就是让网络上的计算机互相之间可以直接通信而不需要中介，如图 9-8 所示。互联网的雏形——APARNET，就是美国军方为防止“中枢神经系统”遭到毁灭性打击而设计出的具有分布式特性的网络。



图 9-8 P2P 是对原始互联网的回归

#### 9.4.2 P2P 流量的变化趋势及优劣势分析

P2P 的对等特性使其天然具备良好的可扩展性和系统级的可靠性。P2P 服务来自于每个 Peer，当 Peer 增加的时候，可以提供服务的资源也随之而增加，同时也消除了来自于服务器的性能瓶颈和单点故障问题。另外，P2P 具有巨大的成本优势，利用其对等传输的特性，运营商可以依靠用户侧资源建立起庞大的服务网络，达到节省带宽和服务器购置成本的目的，具有良好的规模经济性，因此被众多的服务提供商所采用。P2P 技术的广泛应用，能够将互联网的存储模式由“内容位于中心”模式转变为“内容位于边缘”模式，改变了此前互联网以大网站为中心的状态，重返“非中心化”，给用户更大的权利。

2001 年前后，P2P 流量以惊人速度增长，到 2004 年年底，P2P 应用已超过 HTTP 协议应用，成为互联网上最普遍的应用和带宽的主要消耗者。如图 9-9 所示，据统计，当时 P2P 流量约占服务提供商（ISP）网络总流量的 60%~80%。

## CDN 技术详解

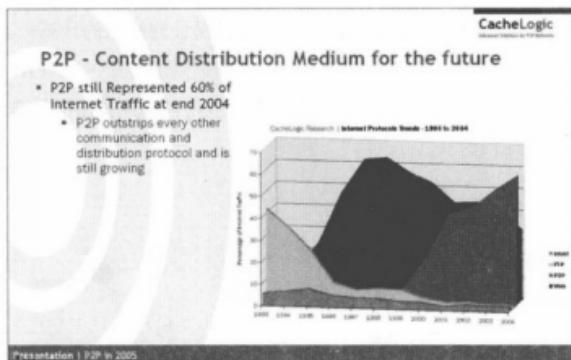


图 9-9 网络流量占比图

在 ipoque 于 2007 年发布的统计报告中，当年 P2P 流量在互联网上的占比达到历史最高，世界各地的 P2P 流量占比差别较大，从 49% 到 83% 不等。部分地区的夜间 P2P 流量峰值能达到整个网络流量的 95%，如图 9-10 和表 9-1 所示。

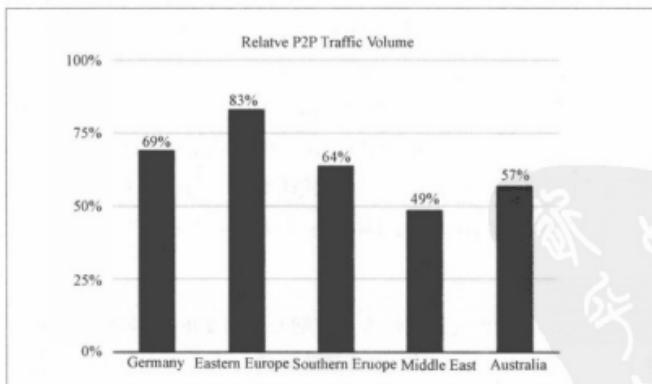


图 9-10 P2P 流量占比 (Source:ipoque)

表 9-1 P2P 流量在各地的流量占比 (Source:ipoque)

| Protocol          | Germany | Eastern Europe | Southern Europe | Middle East | Australia             |
|-------------------|---------|----------------|-----------------|-------------|-----------------------|
| P2P               | 69.25%  | 83.46%         | 63.94%          | 48.97%      | 57.19%                |
| HTTP              | 10.05%  | -              | -               | 26.05%      | -                     |
| Media Streaming   | 7.75%   | -              | -               | 0.07%       | 0.02%<br>(boost only) |
| DDL               | 4.29%   | -              | -               | 8.66%       | -                     |
| VoIP/Skype        | 0.92%   | -              | -               | 0.57%       | 0.51%                 |
| IM                | 0.32%   | -              | -               | 0.24%       | 0.36%                 |
| E-Mail            | 0.37%   | -              | -               | 0.79%       | -                     |
| FTP               | 0.50%   | -              | -               | 0.62%       | -                     |
| NNTP              | 0.08%   | -              | -               | 0.23%       | -                     |
| Tunnel/Encryption | 0.32%   | -              | -               | 1.65%       | -                     |

正当业界普遍认为 P2P 将是未来的互联网发展趋势并预测 P2P 流量还会不断增长时，在 2007 年后，P2P 流量的增长势头出人意料地开始减缓，特别是在近几年其流量占比在持续下降。如图 9-11 和表 9-2 所示，从 2008 年开始，所有地区的 P2P 流量占比都在下降，其流量的增长明显慢于其他类型的流量。

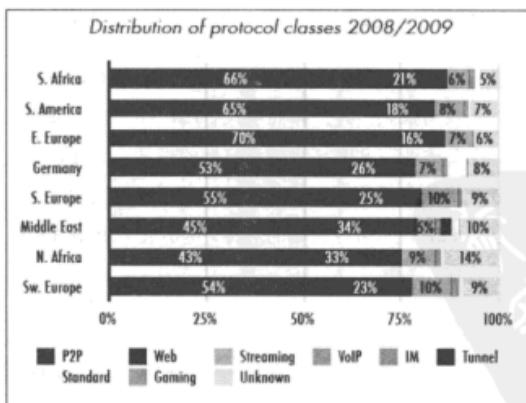


图 9-11 P2P 流量增长减缓 (Source:ipoque)

表 9-2 P2P 流量增长减缓 (Source:ipoque)  
*Protocol class proportions 2008/2009 versus 2007*

| Protocol Class | Eastern Europe |        | Germany       |        | Middle East   |        | Southwestern Europe |        |
|----------------|----------------|--------|---------------|--------|---------------|--------|---------------------|--------|
|                | 2008/<br>2009  | 2007   | 2008/<br>2009 | 2007   | 2008/<br>2009 | 2007   | 2008/<br>2009       | 2007   |
| P2P            | 69,95%         | 83,46% | 52,79%        | 69,25% | 44,77%        | 48,97% | 54,46%              | 63,94% |
| Web            | 16,23%         | -      | 25,78%        | 14,34% | 34,49%        | 34,71% | 23,29%              | -      |
| Streaming      | 7,34%          | -      | 7,17%         | 7,75%  | 4,64%         | 0,07%  | 10,14%              | -      |
| VoIP           | 0,029%         | -      | 0,86%         | 0,92%  | 0,79%         | 0,57%  | 1,67%               | -      |
| IM             | 0,002%         | -      | 0,16%         | 0,32%  | 0,50%         | 0,24%  | 0,08%               | -      |

那么，P2P 流量占比下降背后的原因是什么呢？

一方面，源自 P2P 系统本身存在的缺点。

(1) P2P 系统的可用性问题很难解决。尽管从整个系统而言，P2P 是可靠的，但是对于单个内容或者单个任务而言，P2P 是不稳定的。每个 Peer 可以随时终止服务，甚至退出系统，因此系统中的交换内容随时可能被删除或者被终止共享。作为商业化的服务系统，这些缺陷是致命的。因此，很多服务提供商采用了 P2P 与 CDN 等技术相结合的方式，辅助以中心服务设备，但这样做导致成本增加。

(2) P2P 一直没有找到良好的商业模式。P2P 在安全、管理和版权等方面存在很多问题，主要体现在 P2P 技术缺乏有效的管理，并且具有匿名发布的特点，所以大多数 P2P 服务都将不可避免地和知识产权发生冲突，同时有许多病毒制造者编写利用 P2P 技术进行传播的病毒。这造成 P2P 很难形成良性循环的价值链，整个产业难以维继。

(3) 用户对 P2P 客户端的安装存在抵制情绪。用过 P2P 的读者都知道，使用任何 P2P 应用之前都会被提示必须安装一个客户端软件。这种植于本机的软件会令用户时刻担心自己的数据和隐私被泄露，也担心 P2P 软件对本机资源的

过度占用（很不幸，这两点担心都被现实验证了）。因此，在B/S方式可用的场景下，用户将不倾向于选择安装P2P客户端进行加速传输。以国内某大型视频网站为例，其拥有2亿用户，但只有600万左右的用户安装了该网站开发的P2P客户端插件，安装率仅为3%，主要的服务还是依靠在全国各地数据中心建设的CDN来提供。

除了受限于上述三方面原因，更根本的原因在于整个产业环境发生了变化。从2007年开始，云计算、光纤宽带、3G网络、智能手机等的出现改变了整个产业形态。个人用户终端从以PC为中心转移到以智能手机等便携终端为中心的时代，终端的处理能力变弱，P2P所依赖的终端资源不复存在；同时，3G网络的建设和光纤宽带的升级，使得网络的覆盖范围和接入速度得到很大提升，带宽不再是应用的瓶颈；此外，云计算的出现，提供了中心化的大规模基础设施，让用户将数据都放在云端，更好地实现数据的协同管理。这三重技术浪潮的汇合改变了用户的使用行为和习惯，在用户看来，互联网不再只是一个信息传递的管道，更是一个存储容量巨大、计算能力超强的计算机。人们不仅可以通过互联网访问所要的信息，传递所需的数据，更可以直接在互联网上使用各种应用。于是，大家不再关心从何处获得计算能力，而只关心是否能够及时使用所需的应用；同时，人们也不再关心数据存储的具体位置，而只关心数据是否触手可得。这对于队伍日益庞大的使用移动设备访问互联网的用户而言尤为重要。因此，P2P技术的用户基础被大大削减。

进一步分析，如果我们将时间拉长，视角拉宽，就可以看到服务器、终端和网络这三大因素是如何主导着整个产业技术趋势演变的，如表9-3所示。在早期的大型机时代，网络还没有出现，计算能力和数据都是集中式的；到了20世纪80年代，个人电脑开始逐渐进入千家万户，计算能力和数据从集中式变成了分散式；进入20世纪90年代，互联网的商业化开始兴起，数据又一次集中起来，当然不再是大型机了，而是数据中心，但是计算能力依然是分散式的；2000年后，P2P技术得到了广泛的应用，海量的数据不再存放于数据中心，而是分散在个人电脑里；2010年后，云计算兴起，分散的计算

## CDN 技术详解

能力造成了浪费，人们开始尝试将计算能力集中起来，数据也在一定程度被集中。

产业技术的发展总是呈现出螺旋式上升的路线，不同技术之间不断出现迭代和反复，很少有某一种技术方向会永远取代另一种技术方向。但这种迭代和反复不是简单的“复古”，而是被赋予新的内涵。例如，比起个人电脑时代，在 P2P 时代中的数据分散就是一种可管理的分散。对于互联网而言，不论是集中式还是分散式，都在不断的发展之中并存，只不过在不同的时期，某一种趋势会变得更加明显而已，而不是完全替代以前的技术。如果把互联网看做计算机，P2P 也可看做一种另类的云计算，一种广义的云。

表 9-3 服务器、终端和网络对产业技术演变的影响

| 时间          | 服务器             | 终端            | 网络            | 标志        | 计算能力 | 数据存储 |
|-------------|-----------------|---------------|---------------|-----------|------|------|
| 20世纪60~70年代 | 强大的计算能力,价格昂贵    | 哑终端           | 无             | 大型机和 UNIX | 集中   | 集中   |
| 20世纪80年代    | 强大的计算能力,价格昂贵    | 强计算能力、低成本     | 仅限于学术研究       | 个人电脑      | 分散   | 分散   |
| 20世纪90年代    | PC 服务器出现,成本大幅下降 | 性能摩尔定律,价格逐步下降 | 互联网商业化兴起,网络普及 | 互联网       | 分散   | 集中   |
| 21世纪00年代    | 性能摩尔定律,价格平稳     | 性能摩尔定律,价格逐步下降 | 带宽大幅提升,价格大幅下降 | P2P       | 分散   | 分散   |
| 21世纪10年代    |                 | 便携终端出现        | 带宽提速加快        | 云计算       | 集中   | 集中   |

### 9.4.3 CDN 与 P2P 技术的结合

在 9.3 节我们已论述过，CDN 在云计算时代将会发挥更为重要的作用，而

CDN的下一步技术发展，也可以考虑与P2P进行结合，事实上近两年业界也经常有人做这种尝试。表9-4是对CDN和P2P做流媒体分发方案的综合对比，通过这些比较可以看出，CDN和P2P流媒体解决方案具有很强的互补性。因此，如果能够利用P2P的可扩展能力来降低流媒体对分发资源的要求，而加之以CDN的可靠性、可管理性，保证流媒体分发质量并确保对整个系统的管理能力，就可以构建一个可管理的、能够承载电信级内容应用的内容承载平台。

表9-4 CDN与P2P流媒体解决方案的对比

| 组网方式   | P2P                      | CDN                |
|--------|--------------------------|--------------------|
| 建设成本   | 低                        | 相对较高               |
| 可扩展性   | 用户越多服务能力越高               | 有容量限制              |
| 可靠性    | P2P节点故障不影响服务             | 不存在单点故障但故障可能影响部分服务 |
| 突发访问支持 | 很好                       | 很好                 |
| 服务质量   | 总体服务质量很好，但存在慢启动现象        | 服务质量好              |
| 网络资源占用 | 同一内容可能被多次传送              | 能节省骨干网络带宽          |
| 可管理性   | 管理困难                     | 管理便捷               |
| 安全性    | 解决了服务器安全性，但带来了很多客户端安全性问题 | 安全性高               |
| 版权问题   | 存在严重版权问题                 | 版权问题容易控制           |
| 客户端    | 需要安装                     | 操作系统支持             |
| 网络维护   | 简单                       | 复杂                 |
| 商业模式   | 无清晰模式                    | 有成熟模式              |

这两种方案虽然在计算模型上存在很大差异，但在用户服务流程和工作机制方面却存在不少共性，服务过程都可以归纳为三个阶段：（1）获取内容入口点；（2）选择内容服务点；（3）内容分发传输，两者能够结合互补正是源于这种共性。

## CDN 技术详解

依据 P2P 和 CDN 技术融合层面的不同，目前有三种主流的融合方案：控制层面融合方案、网络叠加融合方案和设备层面融合方案。

### (1) 控制层面融合方案

CDN 和 P2P 作为两个相对独立的子网络平行建设，通过构建一个统一的管理和调度系统，在业务管理和资源调度方面对 CDN 和 P2P 网络进行统一管理。也就是说，系统同时支持用 CDN 和 P2P 技术为用户提供服务。统一的控制系统根据业务的需要，选择该内容使用 P2P 分发、CDN 分发还是同时支持使用两种分发。

在这个方案中，CDN 和 P2P 网络的设备是分离的，不需要对现有的 CDN 网络做任何改动，只需新增加 P2P 服务设备即可实现。主要的融合工作体现在业务管理和资源调度方面。

### (2) 网络叠加融合方案

该方案由 CDN 核心和 P2P 边缘构建两级架构的内容承载平台，该架构的核心层是由受控的 CDN POP 节点构成的 CDN 网络，提供可靠的内容分发源(类似于种子)和可靠的基础服务；边缘层由用户终端充当 Peer 构成。每个 CDN POP 节点都作为一个独立的核心，所有指向该 CDN POP 的 Peer 组成一个 P2P 子网。其中，受控的 CDN 网络核心主要用于解决 P2P 存在的内容可用性和服务可靠性问题，在具体实现时 CDN 核心网络也可以按照 P2P 的方式进行内容的分发；P2P 边缘网络主要用于提供面向用户的流服务传输能力。

该方案 CDN POP 节点是 P2P 网络和 CDN 网络的融合点，也是整个网络最关键的部分。每个 CDN POP 节点都要为它管辖的子网提供目录服务和跨子网服务。

在提供服务时，子网内部使用 P2P 流的对等传送方式，但不允许有跨域的 P2P 流，这是为了保证服务质量。如果子网内没有找到拥有该内容的 Peer，那

么 CDN Cache 会负责从内容源获取内容，将该内容中转提供给 P2P Cache，再以 P2P 方式进行分发。

### (3) 设备层面融合方案

如果需要真正实现 CDN 和 P2P 的融合，向用户屏蔽是 CDN 还是 P2P 服务的差异，需要在设备层面进行融合，对现有的 CDN 设备进行改造。

此方案建立在传统的 CDN 系统基础上，在骨干网层面保留了原有的 CDN 架构和功能，包括内容缓存机制、全局负载均衡机制、骨干网内容分发流程、认证计费相关机制等。而在边缘节点引入了 P2P 技术来进行文件及流媒体的共享，将 P2P 的流量严格限制在同一边缘节点的区域内。

这种做法会带来三个好处：一是可以提高用户请求的响应速度，开始服务时由 Cache 直接为用户提供服务，然后在一定的时间内，根据客户端从其他 Peer 处可获取数据的能力逐步降低直至取消 Cache 的直接服务，改为全部由其他 Peer 提供数据；二是用户可由 C/S 方式平滑过渡到 P2P 方式，有利于提供统一的业务体验和 P2P 客户端的推广；三是若该内容在该边缘节点域内未命中，则由 Cache 启用边缓存边播放功能，向用户提供服务。

## 9.5 CDN 的商业服务发展趋势

在互联网产业生态系统中，CDN 是作为专业化的“电子数据物流服务体系”出现的，专业化的服务是其最根本的商业特征。整个互联网产业演进到今天，已成长为一个与每个人日常生活息息相关的重要产业，CDN 的价值将越来越向服务的本质回归。为了更好地提供服务，CDN 会产生进一步的细分，进一步提升其专业化能力，更好地满足云计算时代、移动互联网时代的业务要求。

### 1. 服务价值提升

在国外，CDN 自诞生之日起，就是作为一种专业化的网络加速服务出现的，将内容分发的相关系统部署和维护工作从网站建设维护中独立、外包给专业的服务提供商。Akamai 在采购大量资源的基础上，通过独立研发的技术和超前的服务意识，将带宽以原价格 4~5 倍出售给大的网站客户。服务领先的指导思想，促使 CDN 运营服务商不断提升技术研发水平，和互联网共同发展，以适应日益增加的互联网分发需求。

但在国内，CDN 产业却走上了一条完全不同的道路。国内正式推出大规模商业 CDN 服务之时，恰好遇到第一次互联网泡沫破灭，因此 CDN 服务商不得不大量利用开源软件作为基础架构，采用低成本、低价格的方式推进市场，CDN 沦为简单的 IDC 资源转售服务。尽管后来自主研发投入加大，技术质量逐步提升，但此时 CDN 已经不再神秘，对 CDN 有较大规模需求的互联网站开始自建 CDN。在自建 CDN 的压力之下，CDN 服务的增值一直没有真正体现出来，更多体现为全网服务和 IDC 资源集中采购等原始 ISP 模式，CDN 错过了高水平集中发展的最佳时机。在定价方面，国内 CDN 产品的价格主要由 IDC 资源价格决定，形成增值的部分不到 50%。

随着云计算推动的新一轮互联网发展浪潮的到来，CDN 的服务价值有望被重新挖掘出来。在 CDN 的市场中，呈现出较强的客户集中性，国内排名前 10% 的互联网站，使用了 95% 以上的 CDN 容量，尤其是对带宽有较强需求的视频网站。这样，我们可以认为 CDN 的市场由窄而高的头部以及宽而低的长尾构成。

在 CDN 的头部市场，激烈的竞争和自建 CDN 的压力，迫使 CDN 服务商提供更加高水平（对比其他服务商和自建）的服务。同时，CDN 的客户网站们为了争夺用户，也会不断推出有特色的个性化业务，要求 CDN 能够快速响应这些新业务需求，提供快而高度专业的定制化服务，这些服务将形成较高的附加值。尽管竞争激烈，但因为市场较窄，无须大量的营销成本和客户关系维系。

成本，CDN服务商可以专注于客户的需求，协助客户共同开拓市场从而共享市场成长。个性化、高附加值的服务，是这一市场领域的明显特征。

在CDN市场的长尾中，集中了大量中小网站，这些网站无力自建CDN网络，由于国内各运营商之间的互联互通资源吃紧，中小网站对能够提供跨域分发的CDN服务需求旺盛。而且伴随国内外互联网的迅猛发展，不断有新的互联网创业公司和投资进来，这将是CDN的一片蓝海。针对这一市场，主要提供标准的加速服务，通过低成本大规模的资源储备和自助的服务界面，降低单一用户的获得成本和服务成本，向大规模增长迅速的中小网站提供内容加速服务。

总之，在未来国内的CDN市场中，技术和服务的增值空间将体现得越来越明显，从而为CDN赢得良好的市场发展空间。

## 2. 市场进一步专业化细分

当前互联网行业呈现越来越细分和专业化的倾向，如移动互联网市场的迅猛增长，物联网的蓬勃发展，大量基于网络的应用（如诸多SaaS）和游戏不断涌现，以Facebook和Twitter为代表的新一代社交型网站脱颖而出。这些专业化应用模糊了原有互联网内容提供和内容接受相分离的传统模式，呈现出更为明显的互动趋势。在这种情况下，要求CDN服务商也要不断提供新的服务模式，不断进行技术研发，就在笔者撰写本书的同时，Akamai推出了专门针对移动互联网的加速解决方案，国内蓝汛、网宿等公司也在跟进。市场的细分，导致CDN服务产品更加丰富，不再是简单的内容加速服务。

在市场细分的同时，产业链的分工也在进一步细化。云计算将众多CP、SP从繁杂的硬件采购和维护中解脱出来，专注于业务设计和客户体验优化，它们不再愿意自己投入成本维护一张对网络技术运维经验要求很高的分发网络。另外，CDN本身的产业链条也在进一步细化，如原来以Cisco为代表推出的网络加速硬件被标准服务器和软件相分离的模式所替代，CDN实现了软硬件分

离，未来基于云计算的 CDN 服务也会将 CDN 服务商部分从硬件维护中解脱，专注于 CDN 系统软件的研发和优化。

CDN 市场发展的专业化趋势将强化产业链各角色之间的清晰界定，为 CDN 高附加值服务奠定了基础。

### 3. CDN 成为云服务的标准配置

随着云计算技术不断向各应用领域蔓延，越来越多的服务和内容将被转移到云端，包括简单的计算能力（IaaS）、云端应用开发平台（PaaS）和大量基于云端的应用（SaaS）。这一趋势将要求用户的终端和服务器保持更紧密、更顺畅和更安全的联系，只有接入网络的终端设备（无论是 PC 还是移动终端）才能享受云服务。在云端获取方式下，CDN 成为未来这一方案蓝图中必不可少的一环。

CDN 自身的发展将逐步和云计算紧密联系在一起。CDN 将成为云服务的一部分，大量云计算服务都是采用分布式架构，连接不同服务器之间的数据和应用交付以及将云服务请求发送到正确节点，正是 CDN 所擅长的工作。云计算的客户通过云发布自己的应用时，也需要同时采购良好的 CDN 服务，才能更为顺畅地将云服务推送到用户那里。

移动互联网的迅猛发展，导致终端形态更为多样化，原有基于 Wintel 架构的需要更多考虑终端适配；同时，移动互联网相对于有线网络来说，带宽仍显不足，而且网络不够稳定，需要良好的带宽管理、缓存机制和终端策略，这些都给 CDN 带来足够的发展空间。

总之，CDN 一方面为云计算服务提供者提供内容分发服务（如微软、Google 等均采购了 Akamai 的 CDN 服务），另外一方面也会和云计算服务形成打包方案（如 Riverbed 和 Amazon 的备份服务进行合作）推出更具竞争力的整套解决方案。

# 附录 A CDN 试验床实施 指南



- A.1 试验床架构概述
- A.2 基础集群环境搭建
- A.3 代理缓存环境搭建
- A.4 边缘节点四层负载均衡
- A.5 边缘节点七层负载均衡
- A.6 多边缘节点负载均衡
- A.7 小结



本实施指南介绍如何利用开源软件搭建 CDN 试验床，循序渐进地完善试验床的功能，以使读者能够通过动手实践对本书介绍的 CDN 核心技术有更为深入和全面的理解。实施指南首先描述 CDN 试验床的整体架构，然后详细介绍试验床搭建的 4 个步骤：基础集群环境搭建、代理缓存环境搭建、边缘节点内负载均衡环境搭建、全局负载均衡环境搭建。

## A.1 试验床架构概述

试验床涵盖了 CDN 系统实施和运行中的关键组件，包括源站点、站点代理缓存、集群负载均衡、全局负载均衡等，其整体架构如图 A-1 所示。

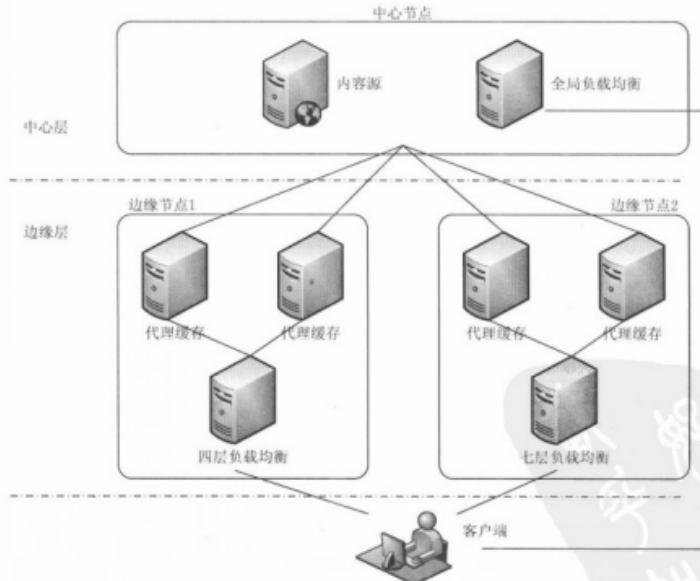


图 A-1 CDN 试验床整体架构

如图 A-1 所示，CDN 试验床具有中心层和边缘层两层架构，包括用户终端

在内的所有组件节点都位于同一个局域网内。CDN 试验床中心层部署的中心节点是用于产生访问内容的源站点服务器，为了简化系统架构，试验床中只设计了一台 Web 服务器构成源站点，它将作为中心节点为用户提供 Web 页面的访问（在实际部署架构中，可能会设立专门的源站点，而中心节点则只负责从源站点复制向用户分发的内容）。边缘层部署了两个边缘节点，每个节点具有由两台服务器构成的集群对中心源站点服务器的 Web 页面进行访问代理和内容缓存，以缓解中心节点的访问压力，改善用户的访问体验。同时，边缘层中还部署了负载均衡服务，用于在节点内的代理缓存服务器集群中按照一定的规则合理地调度访问负载。其中，边缘节点 1 中部署的是基于用户请求中 IP 地址、访问端口等信息解析的负载均衡组件（即四层负载均衡），而节点 2 中部署的是基于网络协议栈 4 至 7 层数据内容解析的负载均衡组件（即七层负载均衡）。另外，试验床的中心层中还部署有全局负载均衡组件，它能够根据用户访问内容的特征和边缘层各个节点的部署和运行情况，将用户请求负载转发到合适的边缘节点上。试验床架构是对真实的 CDN 系统的抽象，它全面地展现了 CDN 系统的基本特征和技术实现原理。

CDN 试验床能够支持典型的 CDN 业务流程。当用户在通过域名访问中心节点源服务器提供的 Web 站点时，其请求首先由全局负载均衡组件根据预设的调度策略将其分发到边缘节点 1 或者边缘节点 2 上，进而边缘节点内部的负载均衡组件将针对请求中的 IP 地址、端口信息、应用数据等内容采用有针对性的负载调度策略，将其转发给代理缓存组件。如果用户请求在代理缓存服务器保存的内容中命中，那么代理缓存服务器可以直接提供用户所需的 Web 内容，如果没有命中则代理缓存服务器负责从源站点上获取相关的内容并反馈给用户，然后将该部分内容缓存在边缘节点上。

CDN 试验床的各个关键组件都由开源软件搭建，主要包括用于实现代理缓存功能的 Squid、用于实现四层负载均衡的 LVS、用于实现七层负载均衡的 Nginx、用于实现基于域名的全局负载均衡的 BIND 等，它们全面地体现了 CDN 的核心技术的特征。在本书正文相应章节对相关软件的基本情况进行了

简要介绍的基础上，本实施指南将重点结合 CDN 系统的部署和应用的实际需求对各个软件的安装配置和应用效果进行说明和验证。

## A.2 基础集群环境搭建

CDN 试验床整体上是一个服务器集群，每台服务器上均安装和部署了能够实现 CDN 关键技术的开源软件用于实现 CDN 中的相应功能。在服务器虚拟化技术飞速发展的今天，利用虚拟机搭建试验床是非常好的选择。

### A.2.1 服务器虚拟化环境部署

本实施指南以 KVM 虚拟机为基础搭建 CDN 试验床的服务器集群，首先在物理服务器上安装包含了 KVM 虚拟化技术的 CentOS 5.5 操作系统，然后通过系统中的 virt-manager 等工具管理 KVM 虚拟机的生命周期，每台虚拟机中安装的同样是 CentOS 5.5 操作系统。

CentOS (Community ENTerprise Operating System) 是著名的 Linux 发行版本之一，它的源代码主要来自商业化的 RHEL (Red Hat Enterprise Linux) 中的开源代码部分，因此具有较高的稳定性，在很多企业级用户中都有广泛应用。当前，CentOS 操作系统已经与 KVM 虚拟化技术进行了很好的整合，这使得服务器虚拟化环境的部署非常便捷，但是在部署过程中，仍需要注意以下要点。

#### 1. 检查物理服务器的硬件虚拟化支持情况

KVM 的实现要基于硬件支持的虚拟化技术，对 CPU 具有较高的要求，例如 CPU 需要支持 Intel VT 或 AMD-V 等硬件虚拟化技术，如果服务器 CPU 不具备硬件虚拟化的支持能力，将不能部署和应用 KVM 虚拟化技术。

随着虚拟化技术的日渐成熟，当前主流的处理器都已经提供了对硬件虚拟化技术的支持，而在 Linux 操作系统中检查物理服务器是否支持硬件虚拟化技术的方法也非常简单。以本实施指南编制中采用的 CPU 型号为 Intel Xeon E7520 的物理服务器为例，在操作系统的命令行界面中输入指令：

```
# vi /proc/cpuinfo
```

可以看到本实施指南编制中采用的服务器 CPU 信息，如图 A-2 所示。

|                 |            |   |
|-----------------|------------|---|
| processor       | :          | 0   |
| vendor_id       | :          | GenuineIntel  |
| cpu family      | :          | 6   |
| model           | :          | 46  |
| model name      | :          | Intel(R) Xeon(R) CPU  |
| stepping        | :          | 6   |
| cpu MHz         | :          | 1862.050  |
| cache size      | :          | 18482 KB  |
| physical id     | :          | 0   |
| siblings        | :          | 8   |
| core id         | :          | 0   |
| cpu cores       | :          | 4   |
| apicid          | :          | 0   |
| fpu             | :          | yes   |
| fpu_exception   | :          | yes   |
| cpuid level     | :          | 11  |
| wp              | :          | yes   |
| flags           | :          | fpu vme de pse tsc msr pae mce cx8 apic sep srtt pge mca cmov pat_pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm syscall nx rdtscp lm const ant_tsc nonstop_tsc pnpi monitor ds_cpl [VMX] est tm2 ssse3 cxl6 xtpr sse4_1 sse4_2 popcntlahf_lm bogomips : 3724.10 |
| clflush size    | :          | 64  |
| "/proc/cpuinfo" | [readonly] | 768L, 22016C  |

图 A-2 服务器 CPU 对硬件虚拟化技术的支持信息

图 A-2 中的 flags 中，方框中的 vmx 标志说明该 CPU 支持 Intel VT 技术。如果是支持 AMD-V 技术的 AMD CPU，则会显现 svm 标志。

需要注意的是，某些服务器在 BIOS 中默认是不打开硬件虚拟化支持的。这种情况下，即使 CPU 提供了对硬件虚拟化技术的支持，但是其相关 flags 信息中仍不会出现相关标志，因此在安装和部署服务器虚拟化技术前务必要确认服务器的 BIOS 设置。

### 2. 初始安装包含服务器虚拟化支持的 CentOS

本实施指南编制中，在物理服务器上和 KVM 虚拟机中安装的均是 CentOS 5.5 操作系统，其内核版本为 2.6.18-194.el5。CentOS 提供了便捷的图形化安装界面，如果需要在初始安装中就包含对 KVM 虚拟化技术的支持，那么只要在选择安装软件包的时候，选择“虚拟化”复选框即可，如图 A-3 所示。



图 A-3 CentOS 安装虚拟化技术支持软件包

在选择了安装虚拟化相关软件包后，CentOS 中将能够提供 KVM 等虚拟化技术运行的支持环境和相关的虚拟机管理工具。

### 3. 在现有 CentOS 中增加服务器虚拟化支持

如果在已经安装好的 CentOS 操作系统中增加对 KVM 虚拟化技术的支持，就需要补充安装相关软件包。CentOS 提供了 yum 指令用于方便地添加/删除/更新 RPM 包，它能自动解决包的依赖性问题，便于对大量系统的更新管理。输入指令：

```
# yum groupinstall KVM
```

即能一次性地安装好所有与 KVM 虚拟化相关的软件，例如 KVM 模块以及 virt-manager 等虚拟化管理工具。

如果 KVM 软件包安装成功，那么输入指令：

```
# lsmod | grep kvm
```

即可观察到当前内核中 KVM 模块的状态。

## A.2.2 虚拟机管理基本操作

当前已经有很多开源的虚拟化管理工具被开发，以方便对虚拟机的部署和应用进行管理，virt-manager 就是其中一款。virt-manager 具有方便易用的图形化界面，提供了完整的虚拟机生命周期的管理功能，例如启动（bootup）/关机（shutdown）、暂停（pause）/继续（resume）、挂起（suspend）/恢复（restore）等，能够便捷地创建虚拟机并对虚拟机的工作状态进行监控。virt-manager 的主用户界面如图 A-4 所示。



图 A-4 virt-manager 的主用户界面

如图 A-4 所示，当前的物理服务器上已经部署了名为 client 的 KVM 虚拟机，用鼠标选中 localhost 一行后再单击“新建”按钮即可按照需求创建新的虚拟机。

在 virt-manager 的管理下，虚拟机的创建过程简洁明了，只需按照界面的提示依次提供虚拟机名称、选择虚拟机虚拟化技术（KVM 为完全虚拟）、指定虚拟机操作系统安装文件的放置位置、分配虚拟机使用的块设备分区或文件磁盘映像的存放位置和空间大小、设置虚拟机内存容量和 CPU 信息等，就可以获得定制的 KVM 虚拟机。图 A-5 是按照上述步骤创建的名为 apache 的虚拟机的配置信息，该虚拟机将被作为 HTTP 服务器用于在 CDN 试验床中部署源 Web 站点。

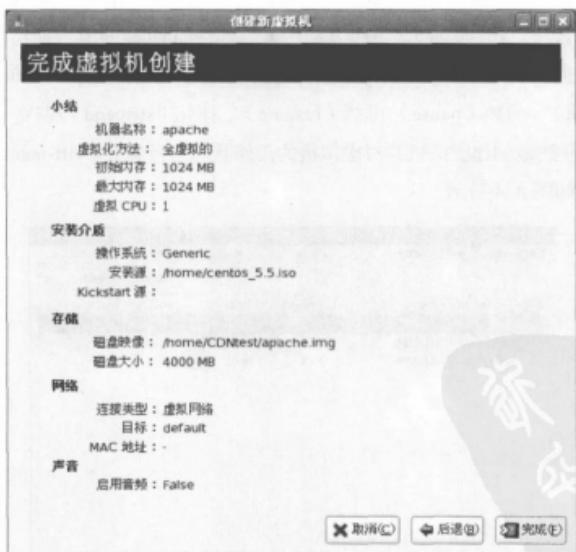


图 A-5 virt-manager 的虚拟机配置信息界面

在图 A-5 所示的界面上单击“完成”按钮，虚拟机操作系统的安装将被启

动，后续的安装过程与在物理服务器上安装操作系统没有任何区别（建议可以先采用最简安装，然后根据 CDN 试验床搭建过程中的实际需求安装必要的软件）。

除了 virt-manger、CentOS，虚拟化软件包中还包括了其他一些工具，相关信息如表 A-1 所示。通过使用这些工具，能够有效地提高虚拟机的使用和管理效率。

表 A-1 常用的开源虚拟化管理工具

| 工具名称         | 功能简介              |
|--------------|-------------------|
| virt-clone   | 克隆已有的虚拟机映像文件      |
| virt-convert | 转换虚拟机定义文件的格式      |
| virt-image   | 通过命令行从映像描述文件创建虚拟机 |
| virt-install | 通过命令行/图形界面创建虚拟机   |
| virt-viewer  | 显示图形化的虚拟机控制台界面    |

### A.3 代理缓存环境搭建

CDN 试验床中的代理缓存环境包括三部分内容：Web 源站点服务器、代理缓存服务器、客户端。所有的服务器和客户端都由虚拟机担当，Web 站点服务器上安装了 Apache HTTP 服务器软件，代理缓存服务器上安装了 Squid 服务器软件，用户客户端则安装了图形界面 Linux 桌面操作系统并使用 Firefox 浏览器，具体的网络部署情况如图 A-6 所示。



图 A-6 代理缓存环境的部署

### A.3.1 Apache HTTP 服务器的安装与配置

Apache HTTP 服务器软件具有极高的稳定性和处理性能，它的安装与配置都比较简单，因此成为当今互联网上使用最为广泛的 Web 服务器软件。

#### Apache HTTP 服务器的安装

Apache HTTP 服务器软件的官方网站是 <http://httpd.apache.org/>，同时还提供了专门的软件下载站点 <http://archive.apache.org/>，通过 wget 指令从该网站上下载所需软件版本的压缩文件并将其解压到 Apache 服务器的指定目录中，输入指令序列：

```
# cd /usr/local/src/  
# wget http://archive.apache.org/dist/httpd/httpd-2.2.21.tar.gz  
# tar -zxvf httpd-2.2.21.tar.gz
```

即在 /usr/local/src/ 目录下创建了 httpd-2.2.21 目录，后续将在该目录下进行软件的安装配置，输入指令序列：

```
# cd httpd-2.2.21  
# ./configure --prefix=/usr/local/apache
```

上述安装配置选项中指定了 Apache HTTP 服务器软件的安装位置为 /usr/local/apache，通过使用 ./configure --help 还可以查看到更多的安装配置选项可供选择。安装配置完成后，输入如下指令序列进行软件的编译和安装：

```
# make  
# make install
```

如果安装过程中没有报错的话，那么 Apache HTTP 服务器就已经安装成功，相关的可执行文件将按照此前的配置要求放置在 /usr/local/apache 目录中。需要注意的是，所有的软件安装都需要编程语言编译环境的支持，因此如果安装配置过程出现系统未安装 gcc 软件的错误信息，就需要首先利用如下指令安装必要的 C/C++ 语言编译环境：

```
# yum install gcc
```

上述 Apache HTTP 服务器软件的下载、配置、编译等步骤是典型的 Linux 操作系统上的应用软件安装方法，本实施指南中后续的开源软件的安装也将是类似的过程。

在 Apache HTTP 服务器软件安装好以后，进入 /usr/local/apache/bin 目录，可以看到 apachectl 文件，它是 Apache 服务器控制和管理 HTTP 服务器功能的应用程序，通过执行如下指令启动 HTTP 服务：

```
# cd /usr/local/apache/bin/  
# ./apachectl start
```

因为 CDN 试验床采用的是局域网环境，因此需要按照图 A-6 所示的要求调整服务器集群的网络环境，其中子网掩码为 255.255.255.0，默认网关为 172.16.0.11。试验床集群中的其他服务器（包括客户端）在完成软件安装和部署后也需要对网络配置做相应的修改。网络配置完成后，在客户端浏览器的地址栏中输入 Apache HTTP 服务器的 IP 地址，将能看到如图 A-7 所示的内容。



图 A-7 Apache HTTP 服务器安装成功界面

需要注意的是，Apache HTTP 服务器的防火墙配置可能会导致客户端对服务器访问的不可达，因此需要通过 `iptables` 指令进行相关配置，或者直接使用 `/etc/rc.d/init.d/iptables stop` 指令将防火墙关闭。

### Apache HTTP 服务的配置

Apache HTTP 服务器的配置文件 `httpd.conf` 保存在 `/usr/local/apache/etc` 目录下，其中在 CDN 试验床搭建中需要重点关注的项目如表 A-2 所示。

表 A-2 Apache HTTP 服务器配置文件项目说明

| 配置项目         | 简要说明  |
|--------------|---|
| ServerRoot   | 描述 HTTP 服务器软件的安装路径，与安装配置时填写的 <code>--prefix</code> 参数一致 |
| Listen       | 描述 HTTP 服务侦听的端口号，默认是 80                                 |
| DocumentRoot | 描述用于 HTTP 服务的 Web 页面文件的存放位置                             |

## A.3.2 Squid 代理缓存服务器的安装与配置

利用 `virt-manager` 工具创建名为 `squid-1` 的 KVM 虚拟机，并在其上部署 Squid 代理缓存服务器软件。

### Squid 代理缓存服务器的安装

Squid 的代码下载和编译安装的指令序列如下：

```
# cd /usr/local/src/
# wget http://www.squid-cache.org/Versions/v3/3.1/squid-3.1.16.tar.gz
# tar -zxfv squid-3.1.16.tar.gz
# cd squid-3.1.16
# ./configure --prefix=/usr/local/squid
# make
# make install
```

完成上述操作后，Squid 代理缓存服务器软件将被安装在 `/usr/local/squid` 目

录下。

需要注意的是，Squid 的安装过程除了需要 gcc 编译器支持之外，还需要安装一些库函数等开发编译环境，相应的指令如下：

```
# yum groupinstall "development tools"
```

在 Squid 代理缓存服务器软件安装好以后，进入 /usr/local/squid/sbin 目录，可以通过执行 squid 程序运行代理缓存服务。其中，如果是第一次运行该软件，需要执行如下指令首先创建缓存目录：

```
# cd /usr/local/squid/sbin/  
# ./squid -z
```

因为上述指令的执行将自动在 /usr/local/squid/var/logs 目录下创建 cache.log 文件并写入关于创建缓存目录的 log 日志信息，所以需要打开该目录的可写权限。为此，可以在执行 ./squid -z 之前执行如下指令：

```
# chmod 777 /usr/local/squid/var/logs/
```

创建缓存目录后，通过执行指令：

```
# ./squid
```

即可开启 Squid 服务，同时自动在 /usr/local/squid/var/logs 目录下创建 access.log 文件用以记录后续的缓存访问日志信息，而通过执行指令：

```
# ./squid -k shutdown
```

即可安全地关闭 Squid 服务。

### Squid 代理缓存服务的配置

Squid 代理缓存服务器的配置文件 squid.conf 保存在 /usr/local/squid/etc 目录下，其中定义了 Squid 服务器运行时所需的各项配置，执行如下指令序列，可以打开并查看该配置文件：

```
# cd /usr/local/squid/etc/  
# vi squid.conf
```

配置文件 `squid.conf` 中定义了和缓存功能相关的配置参数，其中比较重要的是缓存目录 `cache_dir` 的定义。在运行过程中，`Squid` 会在对用户请求的网页地址进行哈希的基础上生成缓存对象，并将其存放在缓存目录中，而 `Squid` 在启动后将首先在内存中建立哈希表，并记录下硬盘中缓存对象的配置情况。利用 `cache_dir` 可以定义多个缓存目录，典型的 `cache_dir` 的配置如下所示：

```
cache_dir ufs /usr/local/var/cache 100 16 256
```

其中，`ufs` 是默认的存储机制，与之相应的还可能有 `aufs`、`coss`、`diskd`、`null` 等选择，后续的系统目录描述了缓存对象放置位置所在的顶层目录，数字 `100` 设定了该缓存目录下的文件容量上限为 `100MB`，`16` 和 `256` 则分别说明了缓存目录下二级目录树中第一级和第二级目录的数量。通过设置 `cache_dir`，可以调整 `Squid` 服务器的缓存支持能力。

配置文件 `squid.conf` 中还提供了灵活的配置参数用于实现代理服务功能。在 CDN 应用场景中，代理缓存服务器通常是工作在反向代理方式下以实现访问的加速，即用户以向代理缓存节点（IP 地址为 `172.16.0.12`）发出访问请求的方式获得 Web 源节点（IP 地址为 `172.16.0.11`）提供的内容，如果代理缓存节点上已经缓存有相关内容则直接将其返回给用户，否则代理缓存节点将把请求转发给 Web 源节点，然后将从源节点获得的内容转发给用户并缓存在本地。针对这一需求，CDN 试验床中的 `Squid` 服务器配置文件被编辑为能够支持反向代理功能。其中，最关键的是在 `squid.conf` 文件的最后增加如下语句：

```
http_port 80 accel vhost vport  
cache_peer 172.16.0.11 parent 80 0  
http_access allow all
```

上述配置语句的相关说明如下：

- `http_port` 配置选项主要用于启用 Squid 服务器对 80 端口的侦听，使之能够处理来自用户的 HTTP 访问请求并提供 Web 服务（`accel`、`vhost`、`vport` 等参数指明了 Web 服务器请求处理端口的绑定信息），如果不做该设置，Squid 服务器将默认侦听 3128 端口，并只能提供缓存服务。
- `cache_peer` 配置选项说明了如果访问请求在 Squid 服务器本地缓存中没有命中的话，需要将请求转发给本级（`sibling`）或者上一级（`parent`）节点以寻找用户所需的内容。在 CDN 试验床中，Web 源节点将作为 Squid 服务器的上一级节点，从其 80 端口接收 Squid 服务器发来的此前缓存未命中的 HTTP 请求。
- `http_access` 配置选项对服务访问控制进行了约束，通过设置相关策略，当前的 Squid 服务器将允许所有的 IP 地址对其进行 HTTP 访问。

`squid.conf` 中还提供了更多的配置选项能够适用于更复杂的应用场景，例如系统中存在有多个 Web 源节点，可以根据配置文件中的相关说明调整相应配置。在修改完配置文件后，需要在`/usr/local/squid/sbin` 目录下运行下列指令检查配置文件的语法正确性并重新配置和启动 Squid 代理缓存服务器：

```
# ./squid -k parse  
# ./squid -k reconfigure
```

### A.3.3 CDN 试验床代理缓存功能的演示和验证

根据反向代理服务的实现原理，可以非常便捷地演示和验证 Squid 代理缓存服务器的相关功能。具体的，只需在客户端浏览器的地址栏中输入 Squid 服务器的地址，即可观察到浏览器上出现 Web 源站点提供的页面内容，如图 A-8 所示。

## CDN 技术详解

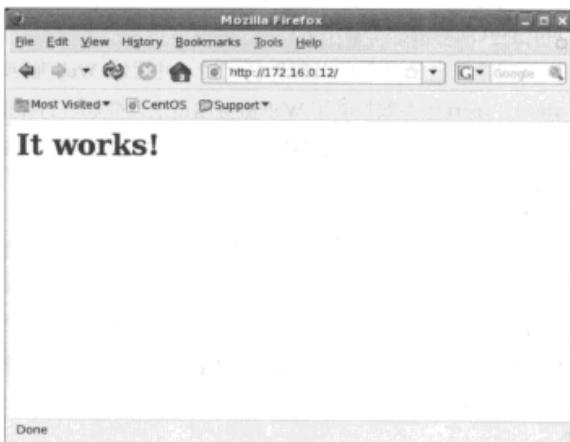


图 A-8 Squid 代理缓存服务反向代理功能演示

保存在 /usr/local/squid/var/logs 目录下的 access.log 日志文件中记录了代理缓存服务的一些工作细节，它包含了用户访问服务器的时间（从 1970 年 1 月 1 日到访问时所经历的秒数）、具体的请求内容等，日志的部分截图如图 A-9 所示。

```
1322032002.274      1 172.16.0.10 TCP MISS 404 401 GET http://172.16.0.12/favicon.ico - FIRST_UP_PARENT/172.16.0.11 text/html  
1322032003.574      0 172.16.0.10 TCP MEM_HIT/200 398 GET http://172.16.0.12/- NONE text/html
```

图 A-9 Squid 代理缓存服务器实现反向代理功能的日志信息

图 A-9 所示记录了客户端浏览器连续两次访问 Squid 代理缓存服务器的相关信息，第一条体现了 Squid 服务器在被用户访问时发生本地缓存不命中，进而向其上一级节点（即 Web 源站点）发出内容获取请求的过程，第二条体现了 Squid 服务器在本地内存中发现了客户端所需内容而直接将其返回给用户的过

程。如果此时对 Web 源站点上保存在 /usr/local/apache/htdocs/ 目录中的 Web 内

容进行修改，再由客户端访问 Squid 代理缓存服务器，会发现页面内容没有任何变化，这是因为 Squid 服务器上已经保存了 Web 源站点的静态网页内容，所以在处理相关访问请求时将直接从本地缓存中为客户端提供旧的内容。在实际的部署和应用中，可以通过设定 Squid 配置文件中的 refresh\_pattern 字段，对被缓存页面的内容规则、缓存时间等指标进行定义。为了方便演示，也可以利用 Squid 软件安装目录中自带的 squidclient 工具直接删除缓存的内容，相关的指令序列如下所示：

```
* cd /usr/local/squid/bin/  
* ./squidclient -p 80 "http://172.16.0.11"
```

通过运行 squidclient，Squid 服务器上保存的 IP 地址为 172.16.0.11 的通过 80 端口访问的 Web 页面数据都将被删除。此时如果再通过客户端浏览器访问 Squid 代理缓存服务器，就将显示出 Web 源站点的更新内容。但是，如果当前客户端访问的内容存在于 Squid 服务器内存中（如图 A-9 所示），那么 Squid 服务器有可能继续为客户端返回过期的内容。

在上述代理缓存功能演示和验证过程中，需要注意随时清除客户端浏览器上保存的 Web 页面内容，以避免演示效果出现偏差。另外，还要确认 Squid 代理缓存服务器的防火墙设置是否允许接受来自客户端的 HTTP 访问请求。

## A.4 边缘节点四层负载均衡

CDN 试验床边缘节点中的四层负载均衡场景的搭建除了按照前文介绍的方法部署一台 Apache HTTP 服务器、两台 Squid 代理缓存服务器（虚拟机名称分别为 squid-1 和 squid-2）、Linux 客户端，还需要安装 KVM 虚拟机用于承载 LVS 负载均衡服务器，具体的网络部署情况如图 A-10 所示。

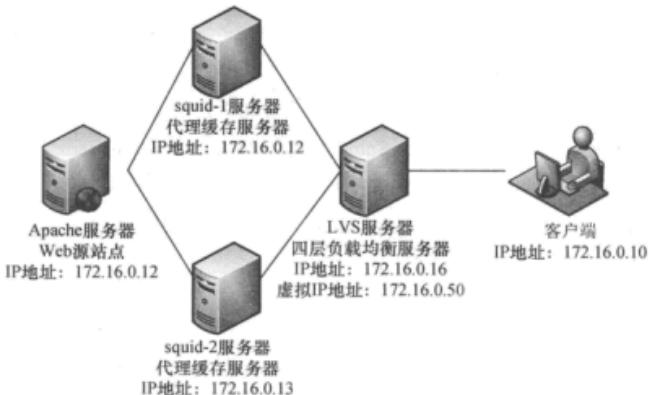


图 A-10 四层负载均衡环境的部署

#### A.4.1 LVS 负载均衡服务器的安装与配置

利用 `virt-manager` 工具创建名为 `lvs` 的 KVM 虚拟机，并在其上部署 LVS 负载均衡服务器软件。

##### LVS 负载均衡服务器的安装

与其他软件不同的是，LVS 以 Linux 内核模块的方式运行，因此在安装 LVS 负载均衡服务器之前，必须要确认当前是否已经将 LVS 相关模块编译进操作系统内核中，相应的查看指令如下：

```
# modprobe -l | grep ipvs
```

LVS 实现四层负载均衡的关键模块是 IPVS，如果该模块没有被加载的话，就需要重新编译操作系统内核。在确认操作系统内核中已经包含了 IPVS 模块后，需要安装名为 `ipvsadm` 的软件在用户空间中对 LVS 四层负载均衡进行配置和管理，其代码下载和解压的指令序列如下：

```
# cd /usr/local/src/
# wget http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-
1.24-6.src.rpm
# rpm -ivh ipvsadm-1.24-6.src.rpm
# mv /usr/src/redhat/SOURCES/ipvsadm-1.24.tar.gz /usr/local/src/
# tar -zxvf ipvsadm-1.24.tar.gz
```

ipvsadm 的安装版本需要根据 Linux 操作系统的内核版本进行选择，因为本实施指南编制时各台虚拟机中安装的操作系统内核均为 2.6.18-194.el5 版本。针对该版本内核，LVS 官方网站上只提供了源代码的 rpm 包，要经过一些处理后才能获得相应的压缩包。在确认相关的语言编译环境已经安装好后，ipvsadm 的编译安装指令序列如下：

```
# cd ipvsadm-1.24
# ln -s /usr/src/kernels/2.6.18-274.12.1.el5-x86_64/ /usr/src/linux
# make
# make install
```

ipvsadm 的安装需要编译部分内核代码，因此需要通过建立符号链接将虚拟机操作系统的源代码保存目录链接到该版本 ipvsadm 的 Makefile 中定义的头文件目录 /usr/src/linux 中。另外，ipvsadm 的代码包中没有提供 configure 脚本，因此不需要执行安装配置步骤。

### LVS 负载均衡服务的配置

LVS 四层负载均衡服务器提供了非常丰富的配置选项，例如负载均衡实现方式、负载调度算法选择等，这些配置均通过运行 ipvsadm 完成，在 /usr/local/src/ipvsadm-1.24 目录下，执行指令：

```
# ./ipvsadm -h
```

即可查看帮助信息中提供的 LVS 四层负载均衡的各项功能和相关参数，例如 LVS 四层负载均衡能够支持 NAT（Network Address Translation）、TUN（IP Tunneling）、DR（Direct Routing）等多种方式。因为当后台服务器节点向用

户返回其所需内容时，采用 DR 方式将无须再经负载均衡节点对返回内容进行处理，具有较高的性能，所以在 CDN 领域广为应用，同时它也是 LVS 四层负载均衡实现方式的默认选项。本实施指南将针对 DR 工作方式进行相应的 LVS 负载均衡服务器的配置。

遵照负载均衡领域的惯用术语，图 A-10 中给出的 squid-1 和 squid-2 两台边缘节点服务器将被称为真实服务器，lvs 服务器用于为客户端提供访问服务的 IP 地址将被称为虚拟 IP 地址。DR 工作方式要求真实服务器的 IP 地址和虚拟 IP 地址必须属于同一网段，图 A-10 中的网络部署则体现了这一要求。

DR 工作方式除了要对负载均衡服务器进行必要的配置外，还需要修改各台真实服务器的配置（主要是禁止它们响应 ARP 报文），因此后续的内容将首先对负载均衡服务器进行配置，然后再对真实服务器进行配置。

负载均衡服务器的配置需要首先修改操作系统配置文件，在操作系统的 /etc/sysctl.conf 文件的最后增加如下语句以修改内核参数：

```
net.ipv4.ip_forward = 0  
net.ipv4.conf.all.send_redirects = 1  
net.ipv4.conf.default.send_redirects = 1  
net.ipv4.conf.eth0.send_redirects = 1
```

修改完配置文件后，执行：

```
# sysctl -p
```

即可将上述改变在系统运行时更新到内核参数中。接着，进行负载均衡服务器的网络设置，执行指令序列如下：

```
# ifconfig eth0:0 172.16.0.50 netmask 255.255.255.255 broadcast  
172.16.0.50 up  
# route add -host 172.16.0.50 dev eth0:0
```

上述指令将负载均衡服务器为客户端提供访问服务的虚拟 IP 地址配置为

172.16.0.50，并设置了相应的路由。后续的负载均衡服务器的配置将主要通过运行 ipvsadm 管理工具完成，指令序列和相关说明如下：

```
# ./ipvsadm -A -t 172.16.0.50:80 -s rr
```

创建负载均衡服务器的虚拟服务及其相应端口（80 为 HTTP 服务），指令中 rr 参数表示选择的负载调度算法为轮询算法（Round Robin）。

```
# ./ipvsadm -a -t 172.16.0.50:80 -r 172.16.0.12 -g
```

建立虚拟服务与后台真实服务器 squid-1 提供的 HTTP 服务的链接，并使用 DR 工作方式。

```
# ./ipvsadm -a -t 172.16.0.50:80 -r 172.16.0.13 -g
```

建立虚拟服务与后台真实服务器 squid-2 提供的 HTTP 服务的链接，并使用 DR 工作方式。

如果需要修订虚拟服务的配置，可以通过 ipvsadm 的-D 或者-C 选项消除单条虚拟服务记录或者整个虚拟服务表。

真实服务器 squid-1 和 squid-2 的相关配置主要通过修改系统配置文件完成，在操作系统的/etc/sysctl.conf 文件的最后增加如下语句以修改内核参数：

```
net.ipv4.ip_forward = 0  
net.ipv4.conf.lo.arp_ignore = 1  
net.ipv4.conf.lo.arp_announce = 2  
net.ipv4.conf.all.arp_ignore = 1  
net.ipv4.conf.all.arp_announce = 2
```

上述语句的作用是使得真实服务器不再转发 IP 数据包和处理 ARP 协议包。修改完配置文件后，执行：

```
# sysctl -p
```

更新内核参数。接着，还需要对真实服务器的网络进行相关配置，以 squid-1

服务器为例，相关的指令序列如下：

```
# ifconfig lo:0 172.16.0.50 netmask 255.255.255.255 broadcast 172.16.0.50 up  
# route add -host 172.16.0.50 dev lo:0
```

至此，通过上述配置，负载均衡服务器将虚拟 IP 地址（172.16.0.50）提供给用户访问，并将用户请求以轮询方式分派给后台的真实服务器，而由真实服务器返回给用户的 Web 页面内容则不再通过负载均衡服务器。

### A.4.2 CDN 试验床四层负载均衡功能的演示和验证

在客户端浏览器的地址栏中输入 LVS 负载均衡服务器对外提供的虚拟 IP 地址，即可观察到浏览器上出现用户需要访问的 Web 源站点提供的页面内容，说明用户访问请求已经被负载均衡服务器分派给了后端的代理缓存服务器进行处理，如图 A-11 所示。

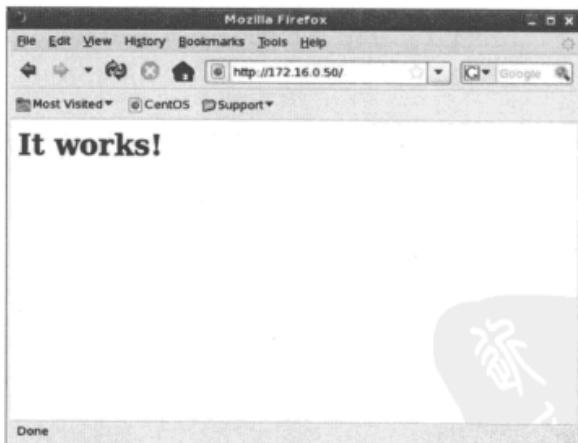


图 A-11 LVS 负载均衡服务功能演示

在客户端浏览器多次对虚拟 IP 地址进行访问后，在 LVS 负载均衡服务器

上运行如下指令序列：

```
# cd /usr/local/src/ ipvsadm-1.24/
# ./ipvsadm -l
```

将可以观察到此前的负载分派情况，如图 A-12 所示。

| IP Virtual Server version 1.2.1 (size=4096) |  |                    |           |       |   |
|---|--|--------------------|-----------|-------|---|
| Front                                       |  | LocalAddress:Port  | Scheduler | Flags |   |
|   |  | RemoteAddress:Port |           |       |   |
|   |  | 172.16.0.50:80     | rr        |       |   |
|   |  | 172.16.0.13:80     | Route     | 1     | Z |
|   |  | 172.16.0.12:80     | Route     | 1     | Z |

图 A-12 LVS 实现基于轮询的负载调度

从图 A-12 中可以发现，用户请求被平均地分配给了后台的真实服务器，实现了基于轮询的负载均衡调度。这一结论也可以从真实服务器 squid-1 和 squid-2 的日志文件分析得出。图 A-13 和图 A-14 分别给出了两台真实服务器的访问记录。

|                |               |   |
|----------------|---------------|---|
| 172.22.215.570 | 0 172.16.0.30 | TCP MEM HIT/200 39B GET http://172.16.0.50/ |
| 172.22.223.249 | 0 172.16.0.30 | TCP MEM HIT/200 39B GET http://172.16.0.50/ |
| 172.22.223.191 | 0 172.16.0.30 | TCP MEM HIT/200 39B GET http://172.16.0.50/ |

图 A-13 squid-1 代理缓存服务器的 access.log 日志

|                    |               |   |
|--------------------|---------------|---|
| 172.22.225.4.613   | 0 172.16.0.30 | TCP MEM HIT/200 39B GET http://172.16.0.50/ |
| 172.22.223.300.892 | 0 172.16.0.30 | TCP MEM HIT/200 39B GET http://172.16.0.50/ |
| 172.22.223.191     | 0 172.16.0.30 | TCP MEM HIT/200 39B GET http://172.16.0.50/ |

图 A-14 squid-2 代理缓存服务器的 access.log 日志

从两台 Squid 代理缓存服务器记录的用户访问处理信息来看，负载均衡服务器在将用户请求转发给后台真实服务器时，其源地址仍旧保留为客户端 IP 地址，同时其目的地址也仍旧是虚拟 IP 地址，体现了 DR 负载均衡的特征。同时，从日志中记录的时间信息可以看出，在轮询调度算法的控制下，两台真实服务器是交替处理用户请求的。

在上述负载均衡功能演示和验证过程中，需要注意随时清除客户端浏览器上保存的 Web 页面内容，以避免演示效果出现偏差。另外，还要确认 LVS 负载均

衡服务器的防火墙设置是否允许接受来自客户端的 HTTP 访问请求。

## A.5 边缘节点七层负载均衡

CDN 试验床边缘节点中的七层负载均衡场景的搭建除了按照前文介绍的方法部署一台虚拟机名为 apache 的 Apache HTTP 服务器、两台 Squid 代理缓存服务器（虚拟机名称分别为 squid-3 和 squid-4）、Linux 客户端，还需要安装 KVM 虚拟机用于承载 Nginx 负载均衡服务器，具体的网络部署情况如图 A-15 所示。

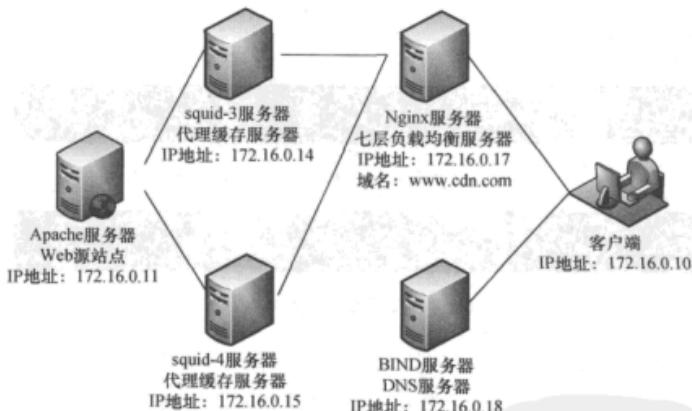


图 A-15 七层负载均衡环境的部署

为了体现出 Nginx 的七层负载均衡能力（例如对用户访问请求中的域名信息进行分析），需要为 Nginx 服务器配置相应的域名（www.cdn.com）并作为它被访问时进行用户请求负载调度的依据，因此部署环境中还包括了一台用于安装 BIND DNS 服务器的虚拟机，而该 DNS 服务器还将在下一节的“多边缘

节点负载均衡”中作为全局负载均衡服务器使用。

### A.5.1 BIND 域名服务器的安装与配置

为对后续的试验床搭建过程提供域名服务支持，需要首先创建名为 bind 的 KVM 虚拟机，并在其上部署 BIND DNS 服务器软件。

#### BIND 域名服务器的安装

BIND 的代码下载和编译安装的指令序列如下：

```
# cd /usr/local/src/  
# wget ftp://ftp.isc.org/isc/bind9/9.8.1-P1/bind-9.8.1-P1.tar.gz  
# tar -zxvf bind-9.8.1-P1.tar.gz  
# cd bind-9.8.1-P1  
# ./configure --prefix=/usr/local/bind  
# make  
# make install
```

需要注意的是，除了安装必需的 C/C++ 语言编译环境，BIND 中 DNSSEC 还需要 OpenSSL 库的支持，因此需要在运行 `./configure` 进行安装配置之前安装库文件并在配置参数中指明头文件的存放位置，或者按照配置错误信息中的提示在配置时增加相应的`--without` 选项。

#### BIND 域名服务器的配置

BIND 域名服务器的主程序是 `named`，与之相关的配置文件 `named.conf` 是由 `rndc` 工具管理并生成的。`rndc` 是 BIND 安装包提供的控制域名服务运行的工具，它能够在不中止 DNS 服务器工作的情况下更新数据并使得修改后的配置文件生效。`rndc` 与域名服务器的连接需要数字证书认证，当 `rndc` 在连接通道中发送命令时，必须使用经过域名服务器认可的密钥加密。为了生成双方都认可的密钥，需要使用 `rndc-confgen` 命令产生密钥和相应的配置，再把这些配置分别放入 `named.conf` 和 `rndc` 的配置文件 `rndc.conf` 中，具体的操作步骤如下：

```
# cd /usr/local/bind/etc/  
# /usr/local/bind/sbin/rndc-confgen > rndc.conf  
# cat rndc.conf > rndc.key  
# chmod 777 /usr/local/bind/var  
# tail -n10 rndc.conf | head -n9 | sed -e s/#\ //g > named.conf
```

在 BIND 域名服务器的配置文件 named.conf 中增加了密钥信息后，需要继续为其补充域名解析信息，执行：

```
# vi named.conf
```

打开该配置文件，并在其中增加以下语句（#后面的内容为相关语句的说明）：

```
options {  
    directory "/usr/local/bind/var";      #存放域名文件的绝对路径  
    pid-file "named.pid";  
    listen-on port 53 {any;};            #服务器监听的端口地址  
    allow-query {any;};                  #服务器允许的 DNS 查询客户端范围  
};  
zone "cdn.com" in {  
    type master;                      #cdn.com 域的正向解析  
    file "cdn.com.zone";              #设置为主域名服务器  
    allow-update {none;};              #cdn.com 域正向解析文件名  
};
```

从配置文件中可以看出，BIND 服务器配置中的 zone 段落用于定义域名解析的域，后续将编制专门的文件对域中的具体域名解析规则进行定义。同时，BIND 服务器支持主从方式的设置，由此也使得在域名解析文件中，需要定义主从服务器之间交互的规则。完成配置文件编辑后，可以执行如下指令序列，检查文件是否编制正确：

```
# cd /usr/local/bind/sbin/  
# ./named-checkconf
```

上述 named.conf 的内容只是针对 CDN 试验床所需的局域网内部域名解析

能力进行的最基本配置，通过配置 BIND 服务器还能够实现反向解析、IP 地址段解析等其他功能。完成 named.conf 文件的编制后，进入在配置文件 options 段落中定义的域名解析文件的存放目录/usr/local/bind/var 编制相关的域名解析文件。本实施指南中，最关键的是创建和编辑用于提供 cdn.com 域的域名正向解析文件，输入指令序列如下：

```
# cd /usr/local/bind/var/  
# vi cdn.com.zone
```

即可编制 cdn.com 域的正向解析文件，在文件中写入以下语句（#后面的内容为相关语句的说明）：

```
$TTL 3600;          # 定义域数据文件中各项记录的默认 TTL 值为 3600 秒  
@ IN SOA ns.cdn.com. admin.cdn.com. {  
    20111205 ;      # 利用修订配置文件的年月日信息标识配置文件版本  
    28800 ;         # 从域名服务器和主域名服务器进行同步的时间间隔  
    7200 ;          # 从域名服务器发生同步失败后进行重试的时间间隔  
    3600000;        # 从域名服务器发生同步失败后记录过期的时间约束  
    6400);          # 默认的最小 TTL 值（如果前面没有指定，则以此为准）  
IN NS ns.cdn.com. #NS 说明该主机为域名服务器（IN 前面必须留有空格）  
ns IN A 172.16.0.18 # 域授权服务器的 A 记录  
www IN A 172.16.0.17 # Nginx 负载均衡服务器的 A 记录
```

上述内容中的第 2 行中标识了一个域数据文件描述的开始，其中 IN 表明后续数据采用的是 Internet 标准，SOA（Start Of Authority）表示后面的记录是域授权服务器（即 ns.cdn.com）和管理员信箱（admin.cdn.com）。因为 BIND 服务器的部署通常需要主从配置，因此后续内容中提供了多个数字用于定义主从服务器之间的同步更新的配置。完成域数据文件编辑后，可以执行如下指令序列，检查文件是否编制正确：

```
# cd /usr/local/bind/sbin/  
# ./named-checkzone cdn.com /usr/local/bind/var/cdn.com.zone
```

如果 named.conf 和各个域数据文件编制无误，则可以在/usr/local/bind/sbin

目录下执行：

```
# ./named
```

启动域名服务。在 BIND 域名服务器运行过程中，因为 named.conf 中定义了服务器将对端口 53 进行监听，所以如果服务器上开启了 iptables 服务，则需要确认防火墙配置文件中是否已经打开了 UDP 53 端口。

为了检验 BIND 域名服务器是否运行正常，可以按照图 A-15 的要求配置 Nginx 负载均衡服务器的域名，并将 172.16.0.18 设置为它使用的 DNS 服务器。然后，在 BIND 域名服务器上执行如下指令序列：

```
# cd /usr/local/bind/sbin/  
# nslookup www.cdn.com
```

如果获得如图 A-16 所示的内容，则说明 BIND 服务器已经开始承担域名解析的工作。

```
Server: 172.16.0.10  
Address: 172.16.0.10#53  
  
Name: www.cdn.com  
Address: 172.16.0.17
```

图 A-16 BIND 域名服务器功能验证

### A.5.2 Nginx 负载均衡服务器的安装与配置

利用 virt-manager 工具创建名为 nginx 的 KVM 虚拟机，并在其上部署 Nginx 负载均衡服务器软件。

#### Nginx 负载均衡服务器的安装

Nginx 的代码下载和编译安装的指令序列如下：

```
# cd /usr/local/src/  
# wget http://nginx.org/download/nginx-1.1.10.tar.gz
```

```
# tar -zvxf nginx-1.1.10.tar.gz
# cd nginx-1.1.10
# ./configure --prefix=/usr/local/nginx
# make
# make install
```

需要注意的是，除了安装必需的 C/C++ 语言编译环境，Nginx 中的 `rewrite`、`gzip` 等模块还需要 PCRE（Perl Compatible Regular Expressions）库、`zlib` 库的支持，因此需要在运行 `./configure` 进行安装配置之前安装这些库文件并在配置参数中指明头文件的存放位置，或者按照配置错误信息中的提示在配置时增加相应的`--without` 选项。

Nginx 本身是一款 HTTP 服务器软件，因此在完成上述安装过程后，可以通过执行如下的指令启动 Nginx 服务并验证 Nginx 的安装是否成功：

```
# cd /usr/local/nginx/sbin/
# ./nginx
# curl -i http://localhost
```

上述指令执行完成后，目录 `/usr/local/nginx/logs` 下的日志文件 `access.log` 中将记录这次访问。在实际运行中，Nginx 需要对 80 端口进行监听，因此需要调整相应的防火墙配置。

### Nginx 负载均衡服务器的配置

Nginx 服务器的配置文件 `nginx.conf` 默认保存在 `/usr/local/nginx/conf` 目录下，其中定义了很多 HTTP 服务器功能和反向代理服务器功能的配置选项，利用这些选项可以实现负载均衡，例如按照用户请求中的域名访问信息，将其指向相应的位于边缘层的代理缓存服务器 IP 地址。执行如下指令序列：

```
# cd /usr/local/nginx/conf/
# vi nginx.conf
```

打开 `nginx.conf` 配置文件，将以下语句替换到文件中（#后面的内容为相关

## CDN 技术详解

语句的说明) :

```
worker_processes  1;
events {
    use epoll;          #设置工作模式为 epoll
    worker_connections 1024;
}
http {
    include mime.types; #设置 mime 类型
    default_type application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local]'
                    '"$request" $status $bytes_sent'
                    '"$http_referer" "$http_user_agent"';
    access_log  logs/access.log  main; #利用此前定义的main格式记录访问日志

    tcp_nopush      on;
    tcp_nodelay     on;
    keepalive_timeout 120;
    server_names_hash_bucket_size 64;

    upstream www.cdn.com {           #定义负载均衡服务器列表
        server 172.16.0.14:80;       #设置默认的轮询方式
        server 172.16.0.15:80;
    }

    server {                      #设置虚拟服务器
        listen  80;                #侦听 80 端口
        server_name www.cdn.com;
        charset utf8;
        access_log  logs/www.cdn.com.log  main;
        location / {
            proxy_pass    http://www.cdn.com;
            proxy_redirect off;
            proxy_set_header Host      $host;
            proxy_set_header X-Real-IP  $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
    }
}
```

```

}
}

```

从上述配置文件中的内容可以看出，Nginx 通过定义 `upstream` 结构，将其与提供给用户访问的域名地址和位于边缘层的代理缓存服务器相绑定，当 Nginx 服务器判别出用户访问数据包中的域名信息后，将按照一定的调度规则（例如轮询、主备等）将其指派到合适的代理缓存服务器上处理。完成配置文件的编辑后，可以执行如下指令序列，检查文件是否编制正确并按照新的配置重新启动 Nginx 服务：

```

# cd /usr/local/nginx/sbin/
# ./nginx -t
# ./nginx -s reload

```

### A.5.3 CDN 试验床七层负载均衡功能的演示和验证

在进行七层负载均衡功能的演示和验证前，需要按照图 A-15 的要求配置 Nginx 负载均衡服务器的域名，并将 Nginx 负载均衡服务器和客户端所需的 DNS 服务器的 IP 地址设置为 172.16.0.18，使它们能够得到 BIND 服务器的域名管理和解析信息。同时，按照前文介绍的 Squid 服务器的配置改写 `squid-3` 和 `squid-4` 的配置文件`/usr/local/squid/etc/squid.conf`，即在配置文件的最后补充以下语句：

```

http_port 80 accel vhost vport
cache_peer 172.16.0.11 parent 80 0
http_access allow all

```

就可以将 `squid-3` 和 `squid-4` 配置为 Web 源站点的代理缓存服务器。

在客户端浏览器的地址栏中输入 Nginx 负载均衡服务器对外提供的域名地址，即可观察到浏览器上出现用户需要访问的 Web 源站点提供的页面内容，说明用户访问请求已经由负载均衡服务器分派给了后端的代理缓存服务器进行处理，如图 A-17 所示。



图 A-17 Nginx 负载均衡服务功能演示

在演示中需要注意的是，Nginx 本身就具有 Web 内容的代理缓存功能但没有提供缓存清空机制，为了能够重点体现出它的负载均衡能力，就需要对其进行相应的配置以保证每次访问请求都能够被调度到后台的 Squid 代理缓存服务器上而不是直接在 Nginx 服务器本地响应，例如设置 Nginx 服务器缓存内容的过期时间或者借助于第三方模块完善 Nginx 的缓存管理功能（例如 `ngx_cache_purge` 模块）。本实施指南编制中，采用了适当增加 Web 源站点提供的网页的内容规模（例如插入图片），并在配置文件 `nginx.conf` 的相应 `server` 配置的 `location` 段落中对 `proxy_buffer_size`、`proxy_buffers`、`proxy_busy_buffers_size` 等参数进行调整的方法，使得 Nginx 缓存不能够保存完整的网页，从而每次都需要向后台服务器发出访问请求。图 A-18 和图 A-19 为代理缓存服务器的访问记录。

```

1323423725.154      16 172.16.8.17 TCP_MISS/200 41495 GET http://www.cdn.com/leno_
.jpg - FIRST_UP_PARENT/172.16.8.11 image/jpeg
1323423728.183      2 172.16.8.17 TCP_MISS/404 464 GET http://www.cdn.com/favico_
n.ico - FIRST_UP_PARENT/172.16.8.11 text/html
1323423929.926      6 172.16.8.17 TCP_MEM_HIT/200 41584 GET http://www.cdn.com/l_
ena.jpg - NONE/- image/jpeg
1323423932.964      4 172.16.8.17 TCP_MISS/404 464 GET http://www.cdn.com/favico_
n.ico - FIRST_UP_PARENT/172.16.8.11 text/html

```

图 A-18 Squid-3 代理缓存服务器的 access.log 日志

```

1323423724.879      7 172.16.8.17 TCP_MISS/200 438 GET http://www.cdn.com/ - FIR_
ST_UP_PARENT/172.16.8.11 text/html
1323423729.149      1 172.16.8.17 TCP_MISS/404 476 GET http://www.cdn.com/favico_
n.ico - FIRST_UP_PARENT/172.16.8.11 text/html
1323423929.667      8 172.16.8.17 TCP_MEM_HIT/200 447 GET http://www.cdn.com/ - 
HOME/- text/html
1323423929.931      4 172.16.8.17 TCP_MISS/404 476 GET http://www.cdn.com/favico_
n.ico - FIRST_UP_PARENT/172.16.8.11 text/html

```

图 A-19 Squid-4 代理缓存服务器的 access.log 日志

通过对图 A-18 和图 A-19 中所示的位于 nginx 服务器后台的 squid-3 和 squid-4 的访问日志文件进行分析，可以观察出，Nginx 负载均衡服务器将用户请求转发给了 Squid 代理缓存服务器，其中网页上的文本内容和图像内容被封装为不同的 HTTP 请求并以轮询的方式分发给后台服务器响应。

## A.6 多边缘节点负载均衡

CDN 试验床中的多边缘节点负载均衡场景的搭建利用了此前各个步骤中积累的服务器集群环境，具体的网络部署情况如图 A-20 所示。

多边缘节点负载均衡体现了全局负载均衡机制的工作情况，其中位于中心层的 Apache HTTP 服务器上仿真部署了两个 Web 源站点，分别具有域名 web-1.cdn.com 和 web-2.cdn.com，通过前端的 BIND 服务器可以进行基于域名的全局负载均衡。例如当用户访问 web-1.cdn.com 时，请求将被分发到边缘节点 1 进行处理，而当用户访问 web-2.cdn.com 时，请求将被分发到边缘节点 2

进行处理。

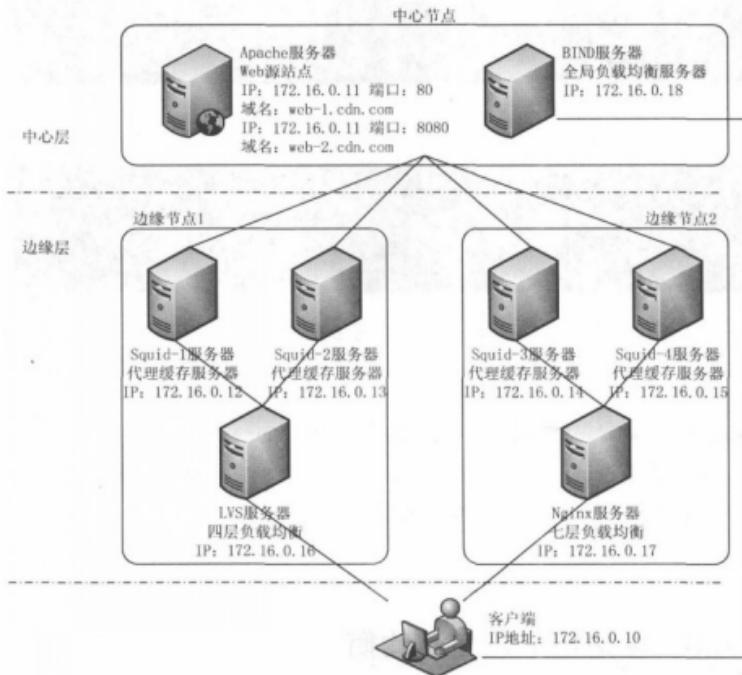


图 A-20 多边缘节点负载均衡环境的部署

### A.6.1 Apache 服务器和 BIND 服务器的配置

按照图 A-20 的要求，在此前集群中各台服务器的已有配置的基础上，需要在 Apache HTTP 服务器上部署两个 Web 站点并以不同的端口对外提供服务，并对 BIND 负载均衡服务器的域名解析进行配置，同时还要调整相应的代理缓存服务器的代理端口绑定。

### Apache HTTP 服务器的配置

通过配置试验床 Apache 服务器的虚拟主机（Virtual Host）机制，可以在同一 IP 地址的不同端口上提供多个域名并部署多个 Web 站点，执行如下指令序列打开配置文件：

```
# cd /usr/local/apache/conf/
# vi httpd.conf
```

将如下语句添加在配置文件的最后：

```
Listen 81                                *服务器侦听 81 端口
Listen 82                                *服务器侦听 82 端口
NameVirtualHost 172.16.0.11:81            *虚拟主机 1, Web 站点 1
NameVirtualHost 172.16.0.11:82            *虚拟主机 2, Web 站点 2
<VirtualHost 172.16.0.11:81>
  ServerName web-1.cdn.com                *虚拟主机 1 的域名
  DocumentRoot /usr/local/apache/htdocs/web-1  *虚拟主机 1 的网页存放位置
</VirtualHost>
<VirtualHost 172.16.0.11:82>
  ServerName web-2.cdn.com                *虚拟主机 2 的域名
  DocumentRoot /usr/local/apache/htdocs/web-2  *虚拟主机 2 的网页存放位置
</VirtualHost>
```

完成配置文件编辑后，进入 /usr/local/apache/bin 目录，执行如下指令：

```
# ./apachectl -t
# ./apachectl restart
```

以检查配置文件的语法正确性并重新启动 HTTP 服务。完成服务器配置后，还需要按照配置文件中要求的内容创建必要的网页文件目录，并编写相应的 Web 页面文件，执行如下指令序列：

```
# mkdir /usr/local/apache/htdocs/web-1
# vi index.html
```

在打开的文件中写入如下 web-1.cdn.com 将要提供的网页内容：

## CDN 技术详解

```
<html><body><h1>Web-1 works!</h1></body></html>
```

类似的，执行如下指令打开 `web-2.cdn.com` 将要提供的网页内容文件：

```
# mkdir /usr/local/apache/htdocs/web-2  
# vi index.html
```

在打开的文件中写入如下 `web-2.cdn.com` 提供的网页内容：

```
<html><body><h1>Web-2 works!</h1></body></html>
```

为了验证 HTTP 服务器的工作情况，在客户端浏览器的地址栏中分别输入两个 Web 站点的 IP 地址和端口，随后显示的网页内容分别如图 A-21 和图 A-22 所示。



图 A-21 Web 站点 1 的网页内容展示

### Squid 代理缓存服务器的配置

为了更好地体现出边缘节点间的负载均衡，需要将边缘节点中的 Squid 代理缓存服务器进行设置，使得边缘节点 1 中的 Squid 服务器集群被用做 Web 站

点 1 的代理缓存，边缘节点 2 中的 Squid 服务器集群被用做 Web 站点 2 的代理缓存。

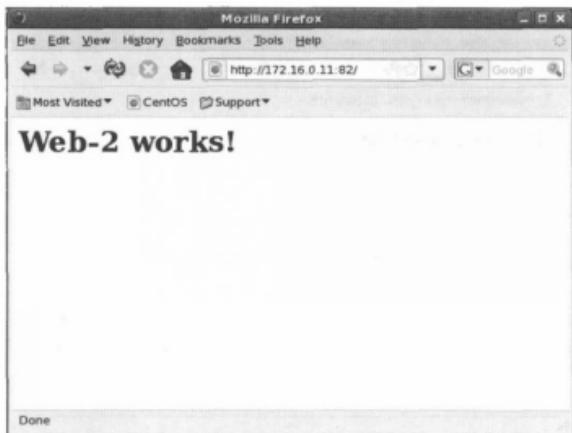


图 A-22 Web 站点 2 的网页内容展示

以边缘节点 1 的 squid-1 服务器为例，进入其 /usr/local/squid/etc/ 目录，打开 squid.conf 文件，将此前添加在文件最后的配置信息改写为如下语句：

```
http_port 80 accel vhost vport
cache_peer 172.16.0.11 parent 81 0
http_access allow all
```

上述语句的主要改变在于 cache\_peer 的定义，当 Squid 服务器的本地缓存内容不能满足 80 端口上的 HTTP 访问要求时，将向其父节点（即源 Web 站点）的 81 端口发起 HTTP 请求并接收数据，从而建立了和 Web 站点 1 的代理缓存关系。类似的，其他三台 Squid 服务器的配置也需要做相应调整。完成配置文件更新后，执行如下指令序列启动代理缓存服务：

```
# cd /usr/local/squid/sbin
# ./squid -k reconfigure
```

## CDN 技术详解

然后，在客户端浏览器地址栏中输入 Squid 服务器的 IP 地址，可观察代理缓存服务的工作情况，以 squid-1 为例，浏览器显示内容如图 A-23 所示。

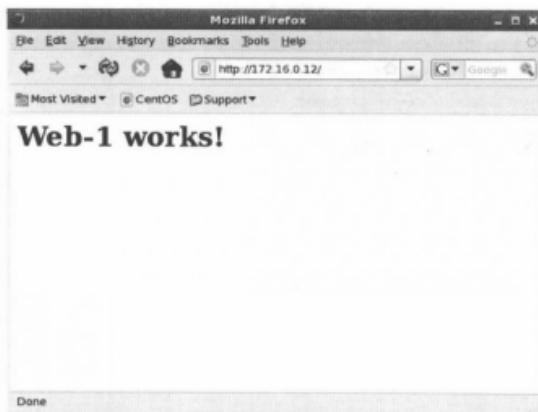


图 A-23 Squid 代理缓存 Web 站点 1 的网页内容展示

### BIND 负载均衡服务器的配置

BIND 服务器主要是通过域名解析的方式，根据解析规则把来自用户的域名访问请求转换为合适的 IP 地址并供客户端与之建立链接，因此它实现负载均衡的关键在于域名解析规则的建立。在 BIND 服务器的 cdn.com 域的域名解析文件 /usr/local/bind/var/cdn.com.zone 中增加对两个 Web 源站点域名的解析规则，修订后的文件内容如下（#后面的内容为相关语句的说明）：

```
$TTL 3600;
@ IN SOA ns.cdn.com. admin@cdn.com. (
    20111205 ;
    28800 ;
    7200 ;
    3600000;
    6400);
IN NS ns.cdn.com.
```

```

ns IN A 172.16.0.18
www IN A 172.16.0.11
web-1IN A 172.16.0.50 #将 web-1.cdn.com 与边缘节点 1 的 LVS 服务器虚拟 IP 绑定
web-2IN A 172.16.0.17 #将 web-2.cdn.com 与边缘节点 2 的 Nginx 服务器绑定

```

编制并保存域名解析文件后，在/usr/local/bind/sbin 目录下执行./named 启动域名服务。

## A.6.2 CDN 试验床多边缘节点负载均衡功能的演示和验证

按照图 A-20 的要求设置 LVS 负载均衡服务器和 Nginx 负载均衡服务器的域名配置，然后在客户端浏览器的地址栏中输入 web-1.cdn.com，即可观察到浏览器上显示的相应的 Web 站点 1 的网页内容，如图 A-24 所示。



图 A-24 通过域名访问 Web 站点 1 的网页内容展示

图 A-24 说明 BIND 域名服务器有效地实现了全局负载均衡，用户对 Web 站点 1 的访问请求已经在经过域名解析后转发至边缘节点 1，并进而由后台的代理服务器进行了处理和响应。另外，在边缘节点 1 中的前端 LVS 负载均衡服务器上也可以看到全局负载均衡的效果，在 LVS 服务器中运行如下指令：

## CDN 技术详解

```
# cd /usr/local/src/ ipvsadm-1.24/  
# ./ipvsadm -l
```

即可看到用户请求在被调度到边缘节点 1 后，LVS 负载均衡服务器将其分派到了后台的代理缓存服务器上，如图 A-25 所示。

| IP Virtual Server version 1.2.1 (size=4896) |                      |           |       |            |           |
|---|----------------------|-----------|-------|------------|-----------|
| Prot  | LocalAddress:Port    | Scheduler | Flags | Forward    | Weight    |
|   | > RemoteAddress:Port |           |       | ActionConn | InactConn |
| TCP   | 172.16.0.50:80       | rr        |       | Route      | 1         |
|   | > 172.16.0.13:80     |           |       | Route      | 1         |
|   | > 172.16.0.12:80     |           |       | Route      | 0         |

图 A-25 LVS 负载均衡服务器的调度记录

类似的，用户发起的对域名 web-2.cdn.com 的访问请求将被 BIND 全局负载均衡服务器在域名解析的基础上调度到边缘节点 2 上，再由边缘节点 2 中的各个组件进行处理，并最终返回给用户其所需的网页内容。

## A.7 小结

本实施指南利用虚拟化环境构建服务器集群，通过部署开源软件搭建 CDN 试验床。指南中全面地介绍了 CDN 系统实现中的关键技术，并对相关软件的安装配置和功能验证进行了分析和阐述。按照本实施指南的内容，读者可以逐步搭建一个具有完备功能的仿真 CDN 系统，有助于读者更深刻地领会 CDN 系统的技术本质和应用效果。

本实施指南中所列的内容主要是根据 CDN 系统的建设和运行的实际需求所遴选出的典型配置和操作。除此以外，CDN 试验床中的每款软件都具有丰富的功能和强大的性能。读者可以在阅读和学习本实施指南的基础之上，对 CDN 系统中感兴趣的技术要点进行重点研究和探索，以实现更灵活、更优化的 CDN 系统架构。

## 参考文献

- 
- [1] 雷葆华, 饶少阳, 江峰, 等. 云计算解码. 北京: 电子工业出版社, 2011.
  - [2] Scot Hull. *Content Delivery Networks: Web Switching for Security, Availability, and Speed*. Brandon A.Nordin, 2002.
  - [3] Rajkumar Buyya, Mukaddim Pathan. *Useful CDN Fundamental-Avtor neizvesten Content Delivery Network*. Springer-Verlag Berlin Heidelberg, 2008.
  - [4] [美] James F.Kurose, KeithW.Ross. 计算机网络——自顶向下的网络设计. 陈鸣译. 北京: 机械工业出版社, 2008.
  - [5] [美] Paul Albitz,Cricket Liu. DNS 与 Band. 雷迎春, 龚奕利译. 北京: 中国电力出版社, 2002.

## CDN 技术详解

- [6] 钟玉琢, 向哲, 沈洪. 流媒体和视频服务器. 北京: 清华大学出版社, 2003.
- [7] 张丽. 流媒体技术大全. 北京: 中国青年出版社, 2001.
- [8] RFC 3040, Internet 网复制和分类法, 2001.
- [9] RFC 2186, Internet Cache Protocol, 1997.
- [10] RFC 2756, Hyper Text Caching Protocol, 2000.
- [11] RFC 882, DOMAIN NAMES - CONCEPTS and FACILITIES, 1983.
- [12] RFC 883, DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION, 1983.
- [13] RFC 1034, Domain Names-Concepts and Facilities, 1987.
- [14] RFC 1035, Domain Names-Implementation and Specification, 1984.
- [15] RFC 2616, HTTP/1.1, 1999.
- [16] IETF draft-ietf-lisp-07 LISP, 2011.
- [17] Chinacache CDN 技术白皮书, 2005.
- [18] Squid 中文权威指南, 2005.
- [19] Blue coat 产品技术白皮书, 2008.
- [20] 面向站长和网站管理员的 Web 缓存加速指南,  
[http://www.chedong.com/tech/cache\\_docs.html](http://www.chedong.com/tech/cache_docs.html)
- [21] Caching Tutorial, [http://www.mnot.net/cache\\_docs/](http://www.mnot.net/cache_docs/)
- [22] Web 技术的发展与意义, [http://blog.163.com/web2009\\_study/blog/static/110314592200912733933660/](http://blog.163.com/web2009_study/blog/static/110314592200912733933660/)
- [23] 大话 session, <http://www.cnblogs.com/shoru/archive/2010/02/19/1669395.html>
- [24] Squid Release Notes 1.1, [http://www.linofee.org/~jel/proxy/Squid/rel-notes-1\\_1.html](http://www.linofee.org/~jel/proxy/Squid/rel-notes-1_1.html)

- 1.html #cache
- [25] Redhat 6.0 下的 Proxy Server, <http://teachers.wyes.tn.edu.tw/chris/os/linux/server/proxy/proxy.htm>
- [26] 广域数据服务-并非只是带宽问题, Riverbed 科技公司技术白皮书, 2007.
- [27] 深入讲解服务器集群技术,  
[http://www.bokee.net/company/weblog\\_viewEntry/2906086.html](http://www.bokee.net/company/weblog_viewEntry/2906086.html)
- [28] 架构基于 Linux 的服务器集群,  
<http://wenku.baidu.com/view/82ff94ff910ef12d2af9e77a.html>
- [29] 集群技术, 百度百科,  
<http://baike.baidu.com/view/4804677.htm>
- [30] RTSP 协议详解, 百度文库,  
<http://wenku.baidu.com/view/38819223192e45361066f5ad.html>
- [31] 邓亮. P2P 与 CDN 结合实现 IPTV 点播业务. 《电信快报》, 2007 年 01 期.
- [32] 包盛, 段保通, 邵锋军. 三网融合下基于云计算的实时转码技术研究和应用. 《电信科学》, 2011 年 03 期.
- [33] 云计算数据中心网络技术, 弯曲评论,  
<http://www.tektalk.org/2011/07/05/%E4%BA%91%E8%AE%A1%E7%AE%97%E6%95%BD%E6%8D%AE%E4%B8%AD%E5%BF%83%E7%BD%91%E7%BB%9C%E6%8A%80%E6%9C%AF/>
- [34] CDN 未来发展趋势 (翻译), <http://blog.c114.net/html/40/26740-53299.html>  
Ongoing Trends and Future Directions in CDNs, Mukaddim Pathan, <http://knol.google.com/k/mukaddim-pathan/ongoing-trends-and-future-directions-in/3uxfz2bu28zlw/2#>
- [35] 长肥管道, <http://www.penna.cn/blog/?p=234>
- [36] 适应长肥网络环境, <http://ziyoula.com/htmlfile/wljs/2011053282.html>

## CDN 技术详解

- [37] 雷葆华, 周开宇. P2P 技术的组网模式与业务模. 《电信科学》, 2007 年增刊.
- [38] 雷葆华, 饶少阳. 云计算对信息产业的影响和成功因素分析. 《电信技术》, 2011 年 01 期.
- [39] 康亮. HTTP Streaming 技术发展趋势. 《电信网技术》, 2011 年 06 期.
- [40] 《Internet study》, ipoque, 2007.
- [41] 《Internet study》, ipoque, 2008-2009.
- [42] 《P2P Survey》, ipoque, 2006.