

Broadview®
www.broadview.com.cn

深入分析

Java Web

技术内幕

许令波◎著

书山有路勤为径

学海无涯苦作舟



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

<http://www.phei.com.cn>

深入分析Java Web 技术内幕

许令波 著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书围绕 Java Web 相关技术从三方面全面深入地进行阐述。首先介绍前端知识，主要介绍 Java Web 开发中涉及的一些基本知识，包括 Web 请求过程、HTTP 协议、DNS 技术和 CDN 技术。其次深入介绍 Java 技术，包括 I/O 技术、中文编码问题、Javac 编译原理、class 文件结构解析、ClassLoader 工作机制及 JVM 的内存管理等。最后介绍 Java 服务端技术，主要包括 Servlet、Session 与 Cookie、Tomcat 与 Jetty 服务器、Spring 容器、Ibatis 框架和 Velocity 框架等原理介绍。本书不仅介绍这些技术和框架的工作原理，而且结合示例来讲解，通过通俗易懂的文字和丰富生动的配图，让读者充分并深入理解它们的内部工作原理，同时还结合了设计模式来介绍这些技术背后的架构思维。

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

深入分析 Java Web 技术内幕 / 许令波著. —北京：电子工业出版社，2012.9

ISBN 978-7-121-17990-7

I. ①深… II. ①许… III. ①JAVA 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字（2012）第 194519 号

策划编辑：刘 皎

责任编辑：董 英

印 刷：北京丰源印刷厂

装 订：三河市鹏成印业有限公司

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：787×980 1/16 印张：29 字数：746 千字

印 次：2012 年 9 月第 1 次印刷

印 数：4000 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。

专 家 点 评

这是一本有关 Java 的书，里面讲述的大量基础知识对前端开发工程师也非常有帮助。比如中文编码章节，作者以一个实践者的身份详细阐述了编码问题的方方面面。总之，这是一本用心的书，是实践者的思考和总结。国内目前很少看到这类书籍，强烈推荐从事 Web 开发工作的人员购买阅读并实践之。

——王保平，开源前端类库 KISSY、SeaJS 作者

作者在淘宝做了很多 Java Web 方面的改造项目，在 Java Web 的相关技术上有深入的掌握，并积累了丰富的经验。在这本书中作者不仅向读者展示了这类大改造项目所需的知识，还展示了 Java Web 更为全景的技术知识体系，值得 Java Web 开发人员阅读。

——林昊，淘宝资深技术专家、China OSGi User Group 总监

从第一次拜读相关内容开始，就可以感觉到作者并不是简简单单地讲述一个技术或者概念，他的分析和讲解十分深入，并且可以很好地聚焦读者的思路，尤其是在 Java Web、Servlet 规范及字符串处理方面，都有很优秀的内容。在众多向 developerWorks 投稿的国内作者中，无论从文章的质量看，还是从内容的选题方向看，作者的文章都可称是上乘之作。同时，他的多篇文章还得到了广大网站读者的好评，其访问量、评分及评论的数量均名列前茅。

深入分析 Java Web 技术内幕

——刘达，developerWorks 中国 Java 专区编辑、技术工程师

读者热评

——摘自 developerWorks 上读者对作者文章的评价

相当不错，读完之后颇有顿悟的感觉~~

——lnwazg

看过 how tomcat works，但是有些东西还是没有弄明白，看了你的这篇介绍，虽然不敢说弄明白了，但是至少让我对 tomcat 工作机制及内部实现有了更进一步的了解！

——android007

总结得非常好，以前看了很多遍源代码，也没这样讲的易懂。

——birds

头一次看着这么全的编解码分析，谢谢分享。

——chenxh

文章相当不错。启动 Servlet 的重要的步骤相当不错。

——RecallYatou

这篇对 Spring 分析的文章太经典了。

——BradyZhu

文章写得很深刻，一直在关注你写的东西。

——61WC_agan_tomsong

很详细，案例和图解释得也很到位，感谢分享。

——lyron

最近才读完了 iBatis 的源代码，也很想写一篇文章的冲动，不过看了此篇，感觉没有

必要了。作者的技术水平和写作水平都令人佩服。

——527*****@QQ.com

推 荐 序

经过 10 多年的发展，Java Web 从开发框架到社区都已经很成熟，在这些成熟的框架、工具的帮助下，开发人员的效率得到了很大的提高，但也造成了在原理性、整体性上的相对欠缺，很多人往往知其然、不知其所以然，特别是在解决一些系统问题的时候，不能很好地举一反三。

举个例子，我看到一些开发人员在使用 Web 框架后，基于约定的方法进行业务的代码实现，但不清楚自己写的代码是如何被调用执行的，如果他很清楚 Servlet 规范，以及看过容器的大致实现过程，对解决问题是很有帮助的。

许令波是我认识的一位很关注原理细节的工程师，同时很乐于分享，会把工作中使用到的技术进行分析并写成文章，分享给大家。他写的这本书中涉及的技术正是他自己在实际工作中遇到的问题的学习过程和解决过程的总结。是总结技术所涉及的知识，更是总结如何分析和解决问题的思路，以及这些技术背后的原理，让你知其所以然。

本书中的内容涉及从 HTTP、Servlet、模板渲染、数据层到容器、JVM 等 Java Web 开发的各个方面，这些问题是许令波在日常工作中经常遇到的，我想也是所有 Java Web 开发人员都会遇到的。本书最大的特点就是让 Java Web 开发人员对整个开发过程所涉及技术能有一个完整的脉络图，从前端浏览器到 Java 技术再到 Java 服务端技术，还介绍了实现这些技术用到的设计模式。不仅详细总结了这些技术的工作原理，而且也结合了很多实际案例来进行阐述，将复杂难懂的技术原理通过时序图和架构图的方式展现出来，更加便于读者理解。可以说掌握了本书的知识，就可以成为一个合格的 Java Web 开发人员。

本书文笔流畅，图表清晰易懂，值得推荐给 Java Web 开发人员作为进阶学习的参考书。

吴泽明

淘宝网产品技术部研究员
杭州，2012.7.23

序

回想起我第一次接触电脑的时间应该是在 10 年前了，记得当时连怎么开电脑都不会，当时感觉电脑真是一个让人着迷的东西，但是那时别说拥有一台电脑，就算是能玩上电脑也算是一件奢侈的事情了。人总是有好奇心的，而我也因为追随着这份好奇和电脑一起走过了将近 10 年的光阴，也是这份好奇让我接触了电脑，认识了电脑，到现在了解了电脑。但是到目前为止我仍然有很多好奇的东西，所以我将一直求解下去。

回想我开始学习编程的时候，在大学期间开始构建自己的第一个网页到第一个网站，其中的复杂程度真是难以想象，要构建一个网页，需要学习当时的“网页三剑客”，页面布局需要学习 Dreamweaver、图片处理需要学习 Fireworks、动画制作需要学习 Flash。有时候为了一个导航栏甚至通宵达旦。还有要自己搭建一个本地服务器，要学习 IIS、Apache 等。当时的我竟然能够一个人完成这一系列的事情，现在想想还真是有点佩服自己。

现在回想一下当时自己的学习过程，真是走了很多弯路，浪费了很多时间。当时的学习就像是在一个陌生的城市找路一样，不知道如何才能到达目的地，只能边走边问别人，这个人告诉你一点，那个人告诉你一点，一点一点往前走。但是虽然在往前走，但走的路并不是最近的，甚至有人指的方向是错的。当时缺少一个总揽全局的地图，所以不能画出一条最优的路。虽然走了很多弯路，但是这种不断自学的过程还是大大地提升了我的学习能力，这种好的自学能力也在我以后的学习工作中起到了关键作用。

IT 行业知识变化很快，需要不断地学习新东西，所以学习知识的能力比掌握知识本身更重要。这也是目前大公司招聘标准中很重要的一条。记得当时我的老人在招聘我进入淘宝时，面试时就问我如何学习一门新技术的问题。当你在学习的过程中碰到了很多难题，然后克服这些难题，很多这种过程积累起来就是你无形的宝贵财富。因为你遇到的问题肯定也是其他人遇到的问题，从发现问题、分析问题再到解决问题的过程远比这个问题本身

更有价值。

爱因斯坦说过：“发现问题比解决问题更重要。”对 IT 人员来说，发现 Bug 和重现 Bug 比解决这个 Bug 更有难度。这就好比一个外国地问周总理中国有多少厕所，总理回答说只有两个厕所：男厕所和女厕所。但是，什么人在什么时间、什么地点需要上厕所，考虑这样的情形恐怕需要多少厕所就很难计算了。同样电脑中也只有 0 和 1 两个选择，电脑中的程序也同样如此，每写一行代码就能增加甚至一个数量级的出错的概率。但是我们还是要学习如何避免出现 Bug，这就要求我们能有总理的看问题的思维，将复杂的问题简单化，发现问题背后的本质，找到解决问题的背后的一些通用逻辑，按照这种思路来解决问题可能会让你事半功倍。

如何让学习知识的过程事半功倍，尤其是我们程序员如何做到，从我这么多年的学习过程来说，有一些经验可以分享给大家，这也是我写这本书的初衷，我真正想分享的不是我掌握的知识，而更多的是我如何学习这个知识的过程，或者说我是如何学习这个知识的，以及我对这些知识的一些总结和提炼。

虽然要掌握整个 Web 开发中涉及的所有知识是一件非常困难的事情，尤其是要掌握这些知识的实现原理，不仅知其然还要知其所以然。所以掌握学习它们的方法至关重要。如何快速高效地阅读它们的源码，有很多同学看到我在 developerWorks 发表的文章时来信问我如何阅读各种框架的源码，很多同学都说不知道从哪里入手来看。其实，当你掌握了一些技巧，加上你的一点耐心，这并不是很难的。

本书虽然介绍了很多开源框架，但是始终都在告诉你如何才能更深入和简单地掌握这个框架，告诉你学习的方法，而并不是告诉你这个框架有哪些类、怎么用这些零碎的知识。打个比喻，本书并不是告诉你 $1+1=2$ ， $1+2=3$ ， $2+2=4$ 这个结果，然后你可以根据这个方式得出 $1+1+2=4$ ，你要计算其他数必须根据它给你的公式才能计算，而是告诉你加减乘除的算法规则，然后你可以根据这个规则自己做运算了。

另外本书为什么要选择介绍 Web 开发中这些技术的实现原理，因为只有你掌握它们的实现原理，你才能够快速地解决一些意想不到的问题。例如，当你理解了 ClassLoader 的工作机制后，遇到 ClassNotFoundException 时，你就能快速地判断，到底为什么会报这个错误，可能是哪个地方出错导致的。

另外还有一个很重要的原因是，如果你很想进入淘宝、腾讯、百度这样的大型互联网企业工作，不掌握本书讲到的这些技术的实现原理，是很难通过技术面试的。因为面试官不仅希望你会用这些技术，还要求你说出个所以然来。所以掌握这些技术的实现原理可以为你的职业发展提供更好的机会。

本书组织结构

本书从结构上主要分为三个部分：第一部分为基础知识，主要介绍 Java Web 开发中涉及的一些基本知识，如一次 HTTP 请求是什么样的，HTTP 协议本身是如何工作的；第二部分将深入介绍 Java 技术，帮助读者了解 Java 是如何工作的，在会用的基础上进一步理解 Java；第三部分是 Java 服务端技术，主要介绍 Web 服务器的处理流程，包括 Servlet 容器的工作原理和 Web 框架是如何运转的，也就是从 Web 服务器接收到请求到返回请求这个过程中涉及的知识。

除了介绍在 Java Web 开发中用到的框架或系统外，本书还会在介绍这些框架的同时介绍这个框架用到的经典设计模式，因为只有掌握了这些设计模式才能更好地理解这些框架的设计原理。另外用具体的示例来讲解设计模式也能让我们更好地理解设计模式在框架设计中的作用。

目标读者

如果你是学校刚毕业的学生或者刚刚准备学习 Web 开发并且不知道如何入手的人，那么这本书比较适合你；如果你已经工作 1~2 年，已经熟悉了 Java Web 开发的基本流程并且想进一步提高自己，那么这本书更适合你。

如果你已经知道了如何去学习 Java Web 开发技术，正准备入门去实际开发，也就是说你是一个开发新手，那么这本书不太适合你。但是当你知道了如何去开发一个 Web 应用想知道它们如何工作时，欢迎你再回来看看本书，它能帮助你进一步提高。

总的来说，本书适合以下读者群：

- ◎ 对 Web 技术感觉迷茫，不知道如何开始学习，对整个 B/S 工作机制不了解的同学

可以参考本书。

- ◎ Java 技术爱好者，想深入学习 Java 技术内部实现细节的人。
- ◎ 有一定开发基础，但是不了解 Web 中一些容器和框架的内部工作原理的人。

- ◎ 大型互联网工程师，对性能优化和分布式数据管理有兴趣，这里介绍了淘宝的一些实践经验。
- ◎ 开源代码爱好者，喜欢研究开发代码的 Coder 可以从本书中找到一些如何分析源码的方法。

本书不会教你如何开发 Web 应用程序，也不会介绍 Struts、Spring、iBatis 等框架如何使用。我想这些框架的使用参考手册图书市场上有一大把，本书没有必要再重复介绍。但是如果你已经掌握如何使用并且不满足只会使用，想知道它们是如何工作的，想打开这些黑盒子，想以后告诉人家这些黑盒子里到底有些什么东西，对每种技术有强烈的好奇心，如果你是这样的人，本书值得你拥有。

本书特点

- ◎ 本书按照通常的学习习惯设计，为你展示从浏览器发起请求到浏览器最终显示出页面整个过程，让你对 Web 开发的整个过程有个总体的理解。
- ◎ 本书虽然讲解的都是比较深入的技术，但是实践示例和比较恰当的比喻将帮你更好地理解。
- ◎ 本书将结合淘宝网中真实使用示例应用程序来讲解技术，让读者有更好的直观认识。

读者讨论

在本书出版后的任何时间有任何问题，你都可以通过 xulingbo0201@163.com 发送邮件给我，或者到 <http://xulingbo.net> 上向我提交你的建议和想法，我会对所有问题给予回复。

致谢

感谢我的父母，在我高考失败后仍然给我机会让我选择自己想做的事，支持我选择了自己喜欢的电脑行业，在家庭并不富裕的情况下给我配置了第一台电脑，让我有机会继续

追求自己的梦想，是你们的支持和鼓励让我在做自己一直喜欢做的事。

感谢我的莹，从大学就一直陪伴在我身边，有你在我身边是我一直不断努力的最大动力，在本书写作过程中，你完成初稿的审阅工作，同时也给了我很多鼓励和建议。

感谢电子工业出版社的策划编辑刘皎和责任编辑董英，你们严谨认真的工作态度让我非常敬佩。

感谢吴泽明（范禹）老大为本书写的序，不仅带我进入淘宝，而且也一直帮助我持续进步，感谢王保平、林昊和刘达在繁忙的工作中为我写推荐语。

感谢在本书写作过程中提出宝贵意见的同事们，他们的花名是：小凡、小邪、丹臣、哲别、景升、文通、向飞、凌弃、路奇、济城、大仁、常彬、旭天、韩章、小赌、雁声、索尼、凤豪、柳擎、华黎、空望、嗷嗷、渐飞、普智、胜衣、叔度、文景、撒迦、狄龙、祝幽、单通、承泽等。

感谢 developerWorks 上所有给我提出问题和建议的网友们。

许令波

2012.7

目 录

第 1 章	深入 Web 请求过程	1
1.1	B/S 网络架构概述	2
1.2	如何发起一个请求	4
1.3	HTTP 协议解析	6
1.3.1	查看 HTTP 信息的工具	8
1.3.2	浏览器缓存机制	9
1.4	DNS 域名解析	12
1.4.1	DNS 域名解析过程	12
1.4.2	跟踪域名解析过程	15
1.4.3	清除缓存的域名	18
1.4.4	几种域名解析方式	19
1.5	CDN 工作机制	20
1.5.1	CDN 架构	20
1.5.2	负载均衡	21
1.6	总结	24
第 2 章	深入分析 Java I/O 的工作机制	25
2.1	Java 的 I/O 类库的基本架构	25
2.1.1	基于字节的 I/O 操作接口	26
2.1.2	基于字符的 I/O 操作接口	27
2.1.3	字节与字符的转化接口	28
2.2	磁盘 I/O 工作机制	29
2.2.1	几种访问文件的方式	29
2.2.2	Java 访问磁盘文件	33

2.2.3	Java 序列化技术	34
2.3	网络 I/O 工作机制	36
2.3.1	TCP 状态转化	37
2.3.2	影响网络传输的因素	39
2.3.3	Java Socket 的工作机制	39
2.3.4	建立通信链路	40
2.3.5	数据传输	41
2.4	NIO 的工作方式	41
2.4.1	BIO 带来的挑战	41
2.4.2	NIO 的工作机制	42
2.4.3	Buffer 的工作方式	45
2.4.4	NIO 的数据访问方式	47
2.5	I/O 调优	49
2.5.1	磁盘 I/O 优化	49
2.5.2	TCP 网络参数调优	50
2.5.3	网络 I/O 优化	52
2.6	设计模式解析之适配器模式	56
2.6.1	适配器模式的结构	56
2.6.2	Java I/O 中的适配器模式	57
2.7	设计模式解析之装饰器模式	57
2.7.1	装饰器模式的结构	58
2.7.2	Java I/O 中的装饰器模式	58
2.8	适配器模式与装饰器模式的区别	59
2.9	总结	59
第 3 章	深入分析 Java Web 中的中文编码问题	60
3.1	几种常见的编码格式	60
3.1.1	为什么要编码	60
3.1.2	如何“翻译”	61
3.2	Java 中需要编码的场景	63
3.2.1	I/O 操作中存在的编码	63

3.2.2	内存操作中的编码	65
3.3	Java 中如何编解码	66
3.3.1	按照 ISO-8859-1 编码	68
3.3.2	按照 GB2312 编码	69
3.3.3	按照 GBK 编码	70
3.3.4	按照 UTF-16 编码	70
3.3.5	按照 UTF-8 编码	71
3.3.6	UTF-8 编码代码片段	71
3.3.7	几种编码格式的比较	73
3.4	Java Web 中涉及的编解码	73
3.4.1	URL 的编解码	75
3.4.2	HTTP Header 的编解码	78
3.4.3	POST 表单的编解码	78
3.4.4	HTTP BODY 的编解码	79
3.5	JS 中的编码问题	80
3.5.1	外部引入 JS 文件	80
3.5.2	JS 的 URL 编码	81
3.5.3	其他需要编码的地方	83
3.6	常见问题分析	83
3.6.1	中文变成了看不懂的字符	83
3.6.2	一个汉字变成一个问号	84
3.6.3	一个汉字变成两个问号	84
3.6.4	一种不正常的正确编码	85
3.7	总结	86
第 4 章	Javac 编译原理	87
4.1	Javac 是什么	88
4.2	Javac 编译器的基本结构	88
4.3	Javac 工作原理分析	90
4.3.1	词法分析器	91
4.3.2	语法分析器	98

4.3.3	语义分析器	103
4.3.4	代码生成器	113
4.4	设计模式解析之访问者模式	116
4.4.1	访问者模式的结构	117
4.4.2	Javac 中访问者模式的实现	118
4.5	总结	119
第 5 章	深入 class 文件结构	120
5.1	JVM 指令集简介	120
5.1.1	类相关的指令	122
5.1.2	方法的定义	123
5.1.3	属性的定义	124
5.1.4	其他指令集	125
5.2	class 文件头的表示形式	133
5.3	常量池	137
5.3.1	UTF8 常量类型	140
5.3.2	Fieldref、Methodref 常量类型	141
5.3.3	Class 常量类型	141
5.3.4	NameAndType 常量类型	142
5.4	类信息	142
5.5	Fields 和 Methods 定义	143
5.6	类属性描述	147
5.7	Javap 生成的 class 文件结构	148
5.7.1	LineNumberTable	150
5.7.2	LocalVariableTable	151
5.8	总结	153
第 6 章	深入分析 ClassLoader 工作机制	154
6.1	ClassLoader 类结构分析	155
6.2	ClassLoader 的等级加载机制	156
6.3	如何加载 class 文件	159
6.3.1	加载字节码到内存	159

6.3.2	验证与解析	161
6.3.3	初始化 Class 对象	161
6.4	常见加载类错误分析	161
6.4.1	ClassNotFoundException	161
6.4.2	NoClassDefFoundError	162
6.4.3	UnsatisfiedLinkError	163
6.4.4	ClassCastException	164
6.4.5	ExceptionInInitializerError	165
6.5	常用的 ClassLoader 分析	166
6.6	如何实现自己的 ClassLoader	170
6.6.1	加载自定义路径下的 class 文件	170
6.6.2	加载自定义格式的 class 文件	172
6.7	实现类的热部署	174
6.8	Java 应不应该动态加载类	176
6.9	总结	177
第 7 章	JVM 体系结构与工作方式	178
7.1	JVM 体系结构	178
7.1.1	何谓 JVM	178
7.1.2	JVM 体系结构详解	181
7.2	JVM 工作机制	183
7.2.1	机器如何执行代码	183
7.2.2	JVM 为何选择基于栈的架构	184
7.2.3	执行引擎的架构设计	185
7.2.4	执行引擎的执行过程	186
7.2.5	JVM 方法调用栈	191
7.3	总结	195
第 8 章	JVM 内存管理	196
8.1	物理内存与虚拟内存	197
8.2	内核空间与用户空间	198
8.3	Java 中哪些组件需要使用内存	199

8.3.1	Java 堆	199
8.3.2	线程	199
8.3.3	类和类加载器	200
8.3.4	NIO	200
8.3.5	JNI	201
8.4	JVM 内存结构	201
8.4.1	PC 寄存器	202
8.4.2	Java 栈	202
8.4.3	堆	203
8.4.4	方法区	203
8.4.5	运行时常量池	204
8.4.6	本地方法栈	204
8.5	JVM 内存分配策略	204
8.5.1	通常的内存分配策略	205
8.5.2	Java 中内存分配详解	205
8.6	JVM 内存回收策略	210
8.6.1	静态内存分配和回收	210
8.6.2	动态内存分配和回收	211
8.6.3	如何检测垃圾	211
8.6.4	基于分代的垃圾收集算法	213
8.7	内存问题分析	222
8.7.1	GC 日志分析	222
8.7.2	堆快照文件分析	225
8.7.3	JVM Crash 日志分析	225
8.8	实例 1	231
8.9	实例 2	233
8.10	实例 3	235
8.11	总结	240
第 9 章	Servlet 工作原理解析	241
9.1	从 Servlet 容器说起	241

9.1.1	Servlet 容器的启动过程	242
9.1.2	Web 应用的初始化工作	245
9.2	创建 Servlet 实例	247
9.2.1	创建 Servlet 对象	248
9.2.2	初始化 Servlet	248
9.3	Servlet 体系结构	250
9.4	Servlet 如何工作	253
9.5	Servlet 中的 Listener	255
9.6	Filter 如何工作	257
9.7	Servlet 中的 url-pattern	259
9.8	总结	260
第 10 章	深入理解 Session 与 Cookie	261
10.1	理解 Cookie	262
10.1.1	Cookie 属性项	262
10.1.2	Cookie 如何工作	263
10.1.3	使用 Cookie 的限制	266
10.2	理解 Session	267
10.2.1	Session 与 Cookie	267
10.2.2	Session 如何工作	268
10.3	Cookie 安全问题	271
10.4	分布式 Session 框架	272
10.4.1	存在哪些问题	272
10.4.2	可以解决哪些问题	273
10.4.3	总体实现思路	273
10.5	Cookie 压缩	278
10.6	表单重复提交问题	280
10.7	总结	281
第 11 章	Tomcat 的系统架构与 设计模式	282
11.1	Tomcat 总体设计	282
11.1.1	Tomcat 总体结构	283

11.1.2	Connector 组件	289
11.1.3	Servlet 容器 Container	294
11.1.4	Tomcat 中的其他组件	305
11.2	Tomcat 中的设计模式	305
11.2.1	门面设计模式	305
11.2.2	观察者设计模式	307
11.2.3	命令设计模式	309
11.2.4	责任链设计模式	310
11.3	总结	312
第 12 章	Jetty 的工作原理解析	313
12.1	Jetty 的基本架构	313
12.1.1	Jetty 的基本架构简介	313
12.1.2	Handler 的体系结构	315
12.2	Jetty 的启动过程	316
12.3	接受请求	317
12.3.1	基于 HTTP 协议工作	317
12.3.2	基于 AJP 工作	319
12.3.3	基于 NIO 方式工作	322
12.4	处理请求	323
12.5	与 Jboss 集成	326
12.6	与 Tomcat 的比较	327
12.6.1	架构比较	327
12.6.2	性能比较	328
12.6.3	特性比较	328
12.7	总结	329
第 13 章	Spring 框架的设计理念与 设计模式分析	330
13.1	Spring 的骨骼架构	330
13.1.1	Spring 的设计理念	331
13.1.2	核心组件如何协同工作	332
13.2	核心组件详解	333

13.2.1	Bean 组件	333
13.2.2	Context 组件	335
13.2.3	Core 组件	336
13.2.4	Ioc 容器如何工作	338
13.3	Spring 中 AOP 特性详解	348
13.3.1	动态代理的实现原理	348
13.3.2	Spring AOP 如何实现	351
13.4	设计模式解析之代理模式	354
13.4.1	代理模式原理	354
13.4.2	Spring 中代理模式的实现	354
13.5	设计模式解析之策略模式	357
13.5.1	策略模式原理	357
13.5.2	Spring 中策略模式的实现	358
13.6	总结	358
第 14 章	Spring MVC 工作机制与 设计模式	360
14.1	Spring MVC 的总体设计	360
14.2	Control 设计	365
14.2.1	HandlerMapping 初始化	366
14.2.2	HandlerAdapter 初始化	368
14.2.3	Control 的调用逻辑	369
14.3	Model 设计	370
14.4	View 设计	371
14.5	框架设计的思考	373
14.5.1	为什么需要框架	373
14.5.2	需要什么样的框架	373
14.5.3	框架设计的原则	374
14.5.4	“指航灯”	374
14.5.5	最基本的原则	374
14.6	设计模式解析之模板模式	375
14.6.1	模板模式的结构	375

14.6.2	Spring MVC 中的模板模式示例	376
14.7	总结	377
第 15 章	深入分析 Ibatis 框架之系统 架构与映射原理	378
15.1	Ibatis 框架主要的类层次结构	378
15.2	Ibatis 框架的设计策略	379
15.3	Ibatis 框架的运行原理	381
15.4	示例	383
15.5	Ibatis 对 SQL 语句的解析	385
15.6	数据库字段映射到 Java 对象	386
15.7	示例运行的结果	388
15.8	设计模式解析之简单工厂模式	388
15.8.1	简单工厂模式的实现原理	388
15.8.2	Ibatis 中的简单工厂模式示例	389
15.9	设计模式解析之工厂模式	390
15.9.1	工厂模式的实现原理	390
15.9.2	Ibatis 中的工厂模式示例	391
15.10	总结	392
第 16 章	Velocity 工作原理解析	394
16.1	Velocity 总体架构	395
16.2	JJTree 渲染过程解析	398
16.2.1	#set 语法	402
16.2.2	Velocity 的方法调用	403
16.2.3	#if、#elseif 和 #else 语法	406
16.2.4	#foreach 语法	407
16.2.5	#parse 语法	409
16.3	事件处理机制	410
16.4	常用优化技巧	413
16.4.1	减少树的总节点数量	413
16.4.2	减少渲染耗时的节点数量	413
16.5	与 JSP 比较	414

16.5.1	JSP 渲染机制	414
16.5.2	Velocity 与 JSP	420
16.6	设计模式解析之合成模式	420
16.6.1	合成模式的结构	420
16.6.2	Velocity 中合成模式的实现	421
16.7	设计模式解析之解释器模式	422
16.7.1	解释器模式的结构	422
16.7.2	Velocity 中解释器模式的实现	423
16.8	总结	423
第 17 章	Velocity 优化实践	424
17.1	现实存在的问题	424
17.2	优化的理论基础	425
17.2.1	程序语言的三角形结构	425
17.2.2	数据结构减少抽象化	426
17.2.3	简单的程序复杂化	426
17.2.4	减少翻译的代价	427
17.2.5	变的转化为不变	427
17.3	一个高效的模板引擎的实现思路	427
17.3.1	vm 模板如何被编译	429
17.3.2	方法调用的无反射优化	436
17.3.3	字符输出改成字节输出	439
17.4	优化的成果	440
17.4.1	char 转成 byte	440
17.4.2	无反射执行	441
17.5	其他优化手段	442
17.6	总结	442

第 1 章

深入 Web 请求过程

随着 Web 2.0 时代的到来，互联网的网络架构已经从传统的 C/S 架构转变到更加方便快捷的 B/S 架构，B/S 架构大大简化了用户使用网络应用的难度，这种人人都能上网、人人都能使用网络上提供的服务的方法也进一步推动了互联网的繁荣。

B/S 架构带来了两方面好处：

- ⊙ 客户端使用统一的浏览器（**Browser**）。由于浏览器具有统一性，它不需要特殊的配置和网络连接，有效地屏蔽了不同服务提供商提供给用户使用服务的差异性。另外，最重要的一点是，浏览器的交互特性使得用户使用它非常简便，且用户行为的可继承性非常强，也就是用户只要学会了上网，不管使用的是哪个应用，一旦学会了，在使用其他互联网服务时同样具有了使用经验，因为它们都基于同样的浏览器操作界面。
- ⊙ 服务端（**Server**）基于统一的 HTTP 协议。和传统的 C/S 架构使用自定义的应用层协议不同，B/S 架构使用的都是统一的 HTTP 协议。使用统一的 HTTP 协议也为服务提供商简化了开发模式，使得服务器开发者可以采用相对规范的开发模式，这样可以大大节省开发成本。由于使用统一的 HTTP 协议，所以基于 HTTP

协议的服务器就有很多，如 Apache、IIS、Nginx、Tomcat、JBoss 等，这些服务器可以直接拿来使用，不需要服务开发者单独来开发。不仅如此，连开发服务的通用框架都不需要单独开发，服务开发者只需要关注提供服务的应用逻辑，其他一切平台和框架都可以直接拿来使用，所以 B/S 架构同样简化了服务器提供者的开发，从而出现了越来越多的互联网服务。

B/S 网络架构不管对普通用户的使用还是对服务的开发都带来了好处，为互联网的主要参与者、服务使用者和服务开发者降低了学习成本。但是作为互联网应用的开发者，我们还是要清楚，从用户在浏览器里单击某个链接开始，到我们的服务返回结果给浏览器为止，这个过程中到底发生了什么、这其中还需要哪些技术来配合。

所以本章将为你描述这一过程的工作原理，它将涉及浏览器的基本行为和 HTTP 协议的解析过程、DNS 如何解析到对应的 IP 地址、CDN 又是如何工作和设计的，以及浏览器如何渲染出返回的结果等。

1.1 B/S 网络架构概述

B/S 网络架构从前端到后端都得到了简化，都基于统一的应用层协议 HTTP 来交互数据，与大多数传统 C/S 互联网应用程序采用的长连接的交互模式不同，HTTP 协议采用无状态的短连接的通信方式，通常情况下，一次请求就完成了一次数据交互，通常也对应一个业务逻辑，然后这次通信连接就断开了。采用这种方式是为了能够同时服务更多的用户，因为当前互联网应用每天都会处理上亿的用户请求，不可能每个用户访问一次后就一直保持住这个连接。

基于 HTTP 协议本身的特点，目前的 B/S 网络架构大都采用类似于图 1-1 所示的架构设计，既要满足海量用户的访问请求，又要保持用户请求的快速响应，所以现在的网络架构也越来越复杂。

当一个用户在浏览器里输入 `www.taobao.com` 这个 URL 时，将会发生很多操作。首先它会请求 DNS 把这个域名解析成对应的 IP 地址，然后根据这个 IP 地址在互联网上找到对应的服务器，向这个服务器发起一个 `get` 请求，由这个服务器决定返回默认的数据资源给访问的用户。在服务器端实际上还有很复杂的业务逻辑：服务器可能有很多台，到底指定哪台服务器来处理请求，这需要一个负载均衡设备来平均分配所有用户的请求；还有请求

的数据是存储在分布式缓存里还是一个静态文件中，或是在数据库里；当数据返回浏览器时，浏览器解析数据发现还有一些静态资源（如 CSS、JS 或者图片）时又会发起另外的 HTTP 请求，而这些请求很可能会在 CDN 上，那么 CDN 服务器又会处理这个用户的请求，大体上一个用户请求会涉及这么多的操作。每一个细节都会影响这个请求最终是否会成功。

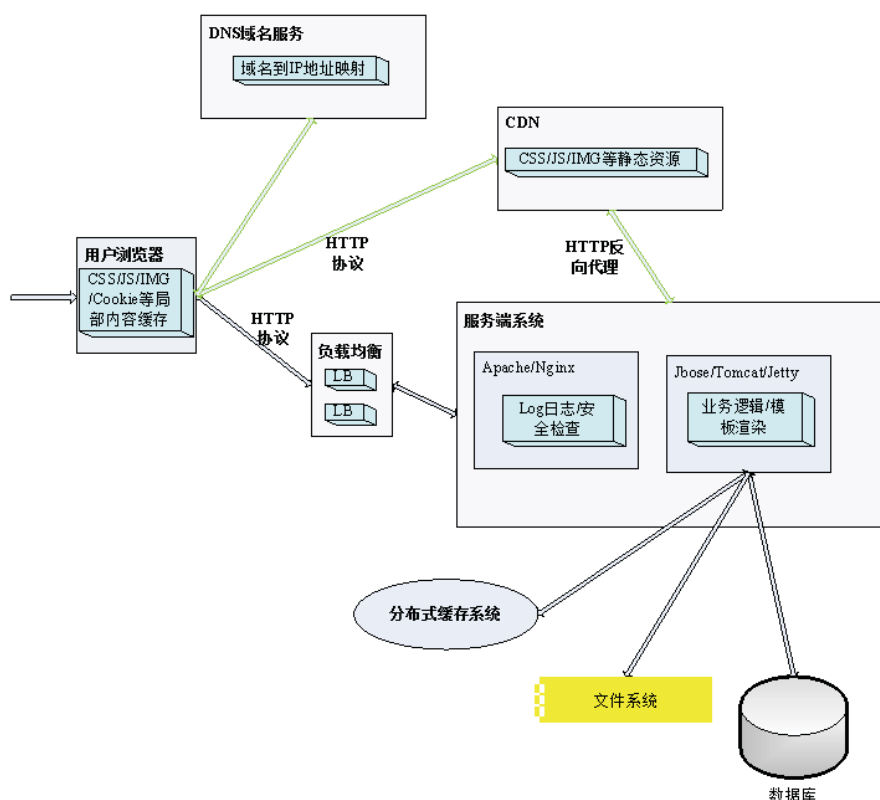


图 1-1 CDN 架构图

不管网络架构如何变化，但是始终有一些固定不变的原则需要遵守。

- ① 互联网上所有资源都要用一个 URL 来表示。URL 就是统一资源定位符，如果你要发布一个服务或者一个资源到互联网上，让别人能够访问到，那么你首先必须要有一个世界上独一无二的 URL。不要小看这个 URL，它几乎包含了整个互联

网的架构精髓。

- ◎ 必须基于 HTTP 协议与服务端交互。不管你要访问的是国内的还是国外的数据、是文本数据还是流媒体，都必须按照套路出牌，也就是都得采用统一打招呼的方式，这样人家才会明白你要的是什么。
- ◎ 数据展示必须在浏览器中进行。当你获取到数据资源后，必须在浏览器上才能恢复它的容貌。

只要满足上面的几点，一个互联网应用基本上就能正确地运转起来了，当然这里面还有好多细节，这些细节在后面将分别进行详细讲解。

1.2 如何发起一个请求

如何发起一个 HTTP 请求？这个问题似乎既简单又复杂，简单是指当你在浏览器里输入一个 URL 时，按回车键后这个 HTTP 请求就发起了，很快你就会看到这个请求的返回结果。复杂是指能否不借助浏览器也能发起请求，这里的“不借助”有两层含义，一是指能不能自己组装一个符合 HTTP 协议的数据包，二是处理浏览器还有哪些方式也能简单地发起一个 HTTP 请求。下面就按照这两层含义来解释如何发起一个 HTTP 请求。

如何发起一个 HTTP 请求和如何建立一个 Socket 连接区别不大，只不过 `outputStream.write` 写的二进制字节数据格式要符合 HTTP 协议。浏览器在建立 Socket 连接之前，必须根据地址栏里输入的 URL 的域名 DNS 解析出 IP 地址，再根据这个 IP 地址和默认 80 端口与远程服务器建立 Socket 连接，然后浏览器根据这个 URL 组装成一个 `get` 类型的 HTTP 请求头，通过 `outputStream.write` 发送到目标服务器，服务器等待 `inputStream.read` 返回数据，最后断开这个连接。

当然，不同浏览器在如何使用这个已经建立好的连接，以及根据什么规则来管理连接，有各种不同的实现方法。一句话，发起一个 HTTP 请求的过程就是建立一个 Socket 通信的过程。

既然发起一个 HTTP 连接本质上就是建立一个 Socket 连接，那么我们完全可以模拟浏览器来发起 HTTP 请求，这很好实现，也有很多方法实现，如 `HttpClient` 就是一个开源的通过程序实现的处理 HTTP 请求的工具包。当然如果你对 HTTP 协议的数据结构非常熟悉，

你完全可以自己再实现另外一个 `HttpClient`，甚至可以自己写个简单的浏览器。

下面是一个基本的 `HttpClient` 的调用示例：

```
HttpClient httpClient = createHttpClient();
    PostMethod postMethod;
    String domainName = Switcher.domain;
    postMethod = new PostMethod(domainName);
    postMethod.addRequestHeader("Content-Type", "application/x-www-form-
urlencoded; charset=GBK");
    for (FilterData filterData : filterDatas) {
        postMethod.addParameter("ip", filterData.ip);
        postMethod.addParameter("count", String.valueOf(filterData.count));
    }
    try {
        httpClient.executeMethod(postMethod);
        postMethod.getResponseBodyAsString();
    } catch (Exception e) {
        logger.error(e);
    }
}
```

处理 Java 中使用非常普遍的 `HttpClient` 还有很多类似的工具，如 Linux 中的 `curl` 命令，通过 `curl + URL` 就可以简单地发起一个 HTTP 请求，非常方便。

例如，`curl "http://item.taobao.com/item.htm?id=1264"` 可以返回这个页面的 HTML 数据，如图 1-2 所示。

```
[junshan@v101055.sqa.cm4 admin]$ curl "http://item.taobao.com/item.htm?id=12643598611" >/dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 149k    0 149k    0    0   3819k    0  --:--:-- --:--:-- --:--:-- 48.3M
```

图 1-2 HTTP 请求返回的 HTML 数据

也可以查看这次访问的 HTTP 协议头的信息，加上 `-I` 选项，如图 1-3 所示。

```
[junshan@v101055.sqa.cm4 admin]$ curl "http://item.taobao.com/item.htm?id=12643598611" -I
HTTP/1.1 200 OK
Date: Sat, 25 Feb 2012 08:33:06 GMT
Server: Apache
at_autype: 5_33483998
at_cat: item_50010850
X-Category: /cat/16
at_nick: %E5%A4%A9%E9%B9%85%E7%BA%B5%E6%A8%AA
at_itemid: 12643598611
at_isb: 0
Set-Cookie: cookie2=f050a98bd189c08c09e46c6d934b60b3; Domain=.taobao.com; Path=/; HttpOnly
Set-Cookie: _tb_token_=ee5ea36ee3366; Domain=.taobao.com; Path=/; HttpOnly
Set-Cookie: t=06f338afdea3d0c1f0306defa65fe4d3; Domain=.taobao.com; Expires=Fri, 25-May-2012 08:33:06 GMT; Path=/
Set-Cookie: v=0; Domain=.taobao.com; Path=/
Content-Language: zh-CN
Vary: Accept-Encoding
Connection: close
Content-Type: text/html; charset=GBK
```

图 1-3 HTTP 协议头的信息

还可以在访问这个 URL 时增加 HTTP 头，通过 -H 选项实现，如图 1-4 所示。

```
</script>[junshan@v101055.sqa.cm4 admin]$ curl "http://switch.taobao.com:9999/repository.htm"
[junshan@v101055.sqa.cm4 admin]$ curl -I "http://switch.taobao.com:9999/repository.htm"
HTTP/1.1 302 Moved Temporarily
Date: Sat, 25 Feb 2012 08:39:31 GMT
Server: Apache
Last-Modified: Thu, 01 Jan 1970 00:00:00 GMT
Location: https://ark.taobao.org:4430/arkserver/Login.aspx?app=http%3A%2F%2Fswitch.taobao.com%
Vary: Accept-Encoding
Content-Type: text/html; charset=GBK
```

图 1-4 访问 URL 时增加 HTTP 头

因为缺少 Cookie 信息，所以上面的访问返回 302 状态码，必须增加 Cookie 才能正确访问该链接，如下所示：

```
[junshan@v101055.sqa.cm4 admin]$ curl -I "http://switch.taobao.com:9999/
repository.htm" -H "Cookie:cna=sd0/BjeZulwCAfIdAHkzZZqC; _t_track=121.0.29.
242.1320938379988839;"
HTTP/1.1 200 OK
Date: Sat, 25 Feb 2012 08:41:20 GMT
Server: Apache
Last-Modified: Thu, 01 Jan 1970 00:00:00 GMT
Vary: Accept-Encoding
Content-Type: text/html; charset=GBK
```

1.3 HTTP 协议解析

B/S 网络架构的核心是 HTTP 协议，掌握 HTTP 协议对一个从事互联网工作的程序员来说非常重要，也许你已经非常熟悉 HTTP 协议，这里除了简单介绍 HTTP 协议的基本知识外，还将侧重介绍实际使用中的一些心得，后面将以实际使用的场景为例进行介绍。

要理解 HTTP 协议，最重要的就是要熟悉 HTTP 协议中的 HTTP Header，HTTP Header 控制着互联网上成千上万的用户的数据的传输。最关键的是，它控制着用户浏览器的渲染行为和服务器的执行逻辑。例如，当服务器没有用户请求的数据时就会返回一个 404 状态码，告诉浏览器没有要请求的数据，通常浏览器就会展示一个非常不愿意看到的该页面不存在的错误信息。

常见的 HTTP 请求头和响应头分别如表 1-1 和表 1-2 所示，常见的 HTTP 状态码如表 1-3 所示。

表 1-1 常见的 HTTP 请求头

请 求 头	说 明
Accept-Charset	用于指定客户端接受的字符集
Accept-Encoding	用于指定可接受的内容编码，如 Accept-Encoding:gzip.deflate
Accept-Language	用于指定一种自然语言，如 Accept-Language:zh-cn
Host	用于指定被请求资源的 Internet 主机和端口号，如 Host:www.taobao.com
User-Agent	客户端将它的操作系统、浏览器和其他属性告诉服务器
Connection	当前连接是否保持，如 Connection: Keep-Alive

表 1-2 常见的 HTTP 响应头

响 应 头	说 明
Server	使用的服务器名称，如 Server: Apache/1.3.6 (Unix)
Content-Type	用来指明发送给接收者的实体正文的媒体类型，如 Content-Type:text/html;charset=GBK
Content-Encoding	与请求报头 Accept-Encoding 对应，告诉浏览器服务端采用的是什么压缩编码
Content-Language	描述了资源所用的自然语言，与 Accept-Language 对应
Content-Length	指明实体正文的长度，用以字节方式存储的十进制数字来表示
Keep-Alive	保持连接的时间，如 Keep-Alive: timeout=5, max=120

表 1-3 常见的 HTTP 状态码

状 态 码	说 明
200	客户端请求成功
302	临时跳转，跳转的地址通过 Location 指定
400	客户端请求有语法错误，不能被服务器识别
403	服务器收到请求，但是拒绝提供服务
404	请求的资源不存在
500	服务器发生不可预期的错误

要看一个 HTTP 请求的请求头和响应头，可以通过很多浏览器插件来看，在 Firefox 中有 Firebug 和 HttpFox，Chrome 自带的开发工具也可以看到每个请求的请求头信息（可用 F12 快捷键打开），IE 自带的调试工具也有类似的功能。

1.3.1 查看 HTTP 信息的工具

有时候我们需要知道一个 HTTP 请求到底返回什么数据，或者没有返回数据时想知道是什么原因造成的，这时我们就需要借助一些工具来查询这次请求的详细信息。

在 Windows 下现在主流的浏览器都有很多工具来查看当前请求的详细 HTTP 协议信息，如在 Firefox 浏览器下，使用最多的是 Firebug，如图 1-5 所示。



图 1-5 Firefox 浏览器下 HTTP 协议的信息

还有一个 HttpFox 工具提供的信息更全，如图 1-6 所示，所有 HTTP 相关信息都可以一目了然。

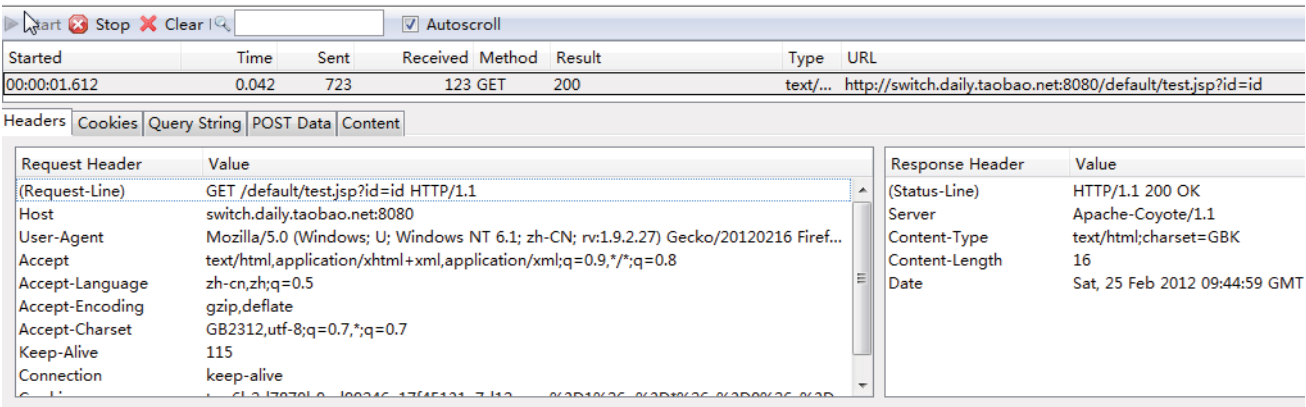


图 1-6 HttpFox 工具显示的 HTTP 协议的信息

Chrome 浏览器下也有一些类似的工具，如 Google 自带的调试工具，同样可以查看到这次请求的相关信息，如图 1-7 所示。

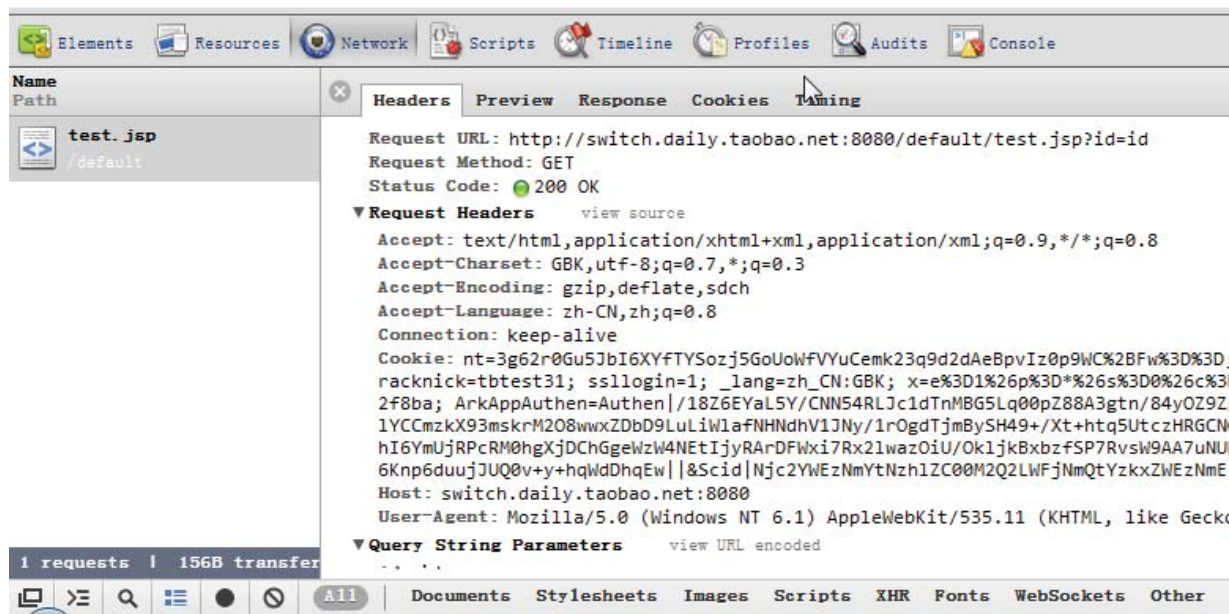


图 1-7 Google 自带的调试工具显示的 HTTP 协议的信息

Chrome 下也有类似的 Firebug 工具，但是还不够完善。

IE 从 7.0 版本开始也提供了类似的 HTTP 调试工具，如自带的开发人员工具可以通过 F12 键打开，HttpFox 插件也有 IE 版本，读者可以试着安装一下。

1.3.2 浏览器缓存机制

浏览器缓存是一个比较复杂但是又比较重要的机制，在我们浏览一个页面时发现异常的情况下，通常考虑的就是是不是浏览器做了缓存，所以一般的做法就是按 Ctrl+F5 组合键重新请求一次这个页面，重新请求的页面肯定是最新的页面。为什么重新请求就一定能够请求到没有缓存的页面呢？首先是在浏览器端，如果是按 Ctrl+F5 组合键刷新页面，那么浏览器会直接向目标 URL 发送请求，而不会使用浏览器缓存的数据；其次即使请求发送到服务端，也有可能访问到的是缓存的数据，比如，在我们的应用服务器的前端部署一个缓存服务器，如 Varnish 代理，那么 Varnish 也可能直接使用缓存数据。所以为了保证用户能够看到最新的数据，必须通过 HTTP 协议来控制。

当我们使用 Ctrl+F5 组合键刷新一个页面时，在 HTTP 的请求头中会增加一些请求头，

它告诉服务端我们要获取最新的数据而不是缓存。

如图 1-8 所示，这次请求没有发送到服务端，使用的是浏览器缓存数据，按 Ctrl+F5 组合键刷新后，如图 1-9 所示。

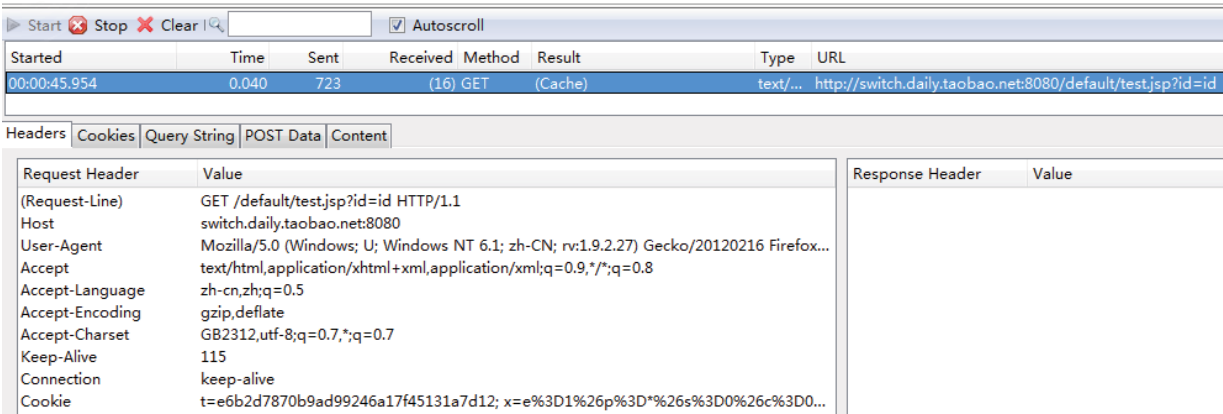


图 1-8 HTTP 的请求头返回缓冲数据

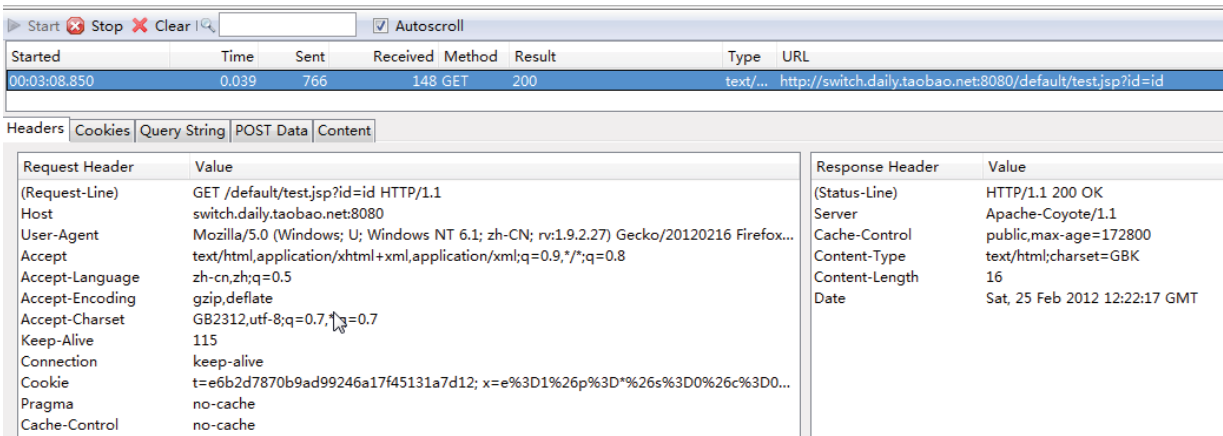


图 1-9 按 Ctrl+F5 组合键刷新页面时 HTTP 的请求头返回数据

这次请求时从服务端返回的数据，最重要的是在请求头中增加了两个请求项 **Pragma:no-cache** 和 **Cache-Control:no-cache**。为什么增加了这两项配置项，它们有什么作用？

1 . Cache-Control/Pragma

这个 HTTP Head 字段用于指定所有缓存机制在整个请求/响应链中必须服从的指令，如果知道该页面是否为缓存，不仅可以控制浏览器，还可以控制和 HTTP 协议相关的缓存或代理服务器。HTTP Head 字段有一些可选值，这些值及其说明如表 1-4 所示。

表 1-4 HTTP Head 字段的可选值

可 选 值	说 明
Public	所有内容都将被缓存，在响应头中设置
Private	内容只缓存到私有缓存中，在响应头中设置
no-cache	所有内容都不会被缓存，在请求头和响应头中设置
no-store	所有内容都不会被缓存到缓存或 Internet 临时文件中，在响应头中设置
must-revalidation/proxy-revalidation	如果缓存的内容失效，请求必须发送到服务器/代理以进行重新验证，在请求头中设置
max-age=xxx	缓存的内容将在 xxx 秒后失效，这个选项只在 HTTP 1.1 中可用，和 Last-Modified 一起使用时优先级较高，在响应头中设置

Cache-Control 请求字段被各个浏览器支持得较好，而且它的优先级也比较高，它和其他一些请求字段（如 Expires）同时出现时，Cache-Control 会覆盖其他字段。

Pragma 字段的作用和 Cache-Control 有点类似，它也是在 HTTP 头中包含一个特殊的指令，使相关的服务器来遵守，最常用的就是 Pragma:no-cache，它和 Cache-Control:no-cache 的作用是一样的。

2 . Expires

Expires 通常的使用格式是 Expires: Sat, 25 Feb 2012 12:22:17 GMT，后面跟着一个日期和时间，超过这个时间值后，缓存的内容将失效，也就是浏览器在发出请求之前检查这个页面的这个字段，看该页面是否已经过期了，过期了就重新向服务器发起请求。

3 . Last-Modified/Etag

Last-Modified 字段一般用于表示一个服务器上的资源的最后修改时间，资源可以是静态（静态内容自动加上 Last-Modified 字段）或者动态的内容（如 Servlet 提供了一个 getLastModified 方法用于检查某个动态内容是否已经更新），通过这个最后修改时间可以判断当前请求的资源是否是最新的。

一般服务端在响应头中返回一个 Last-Modified 字段，告诉浏览器这个页面的最后修改时间，如 Last-Modified: Sat, 25 Feb 2012 12:55:04 GMT，浏览器再次请求时在请求头中增加一个 If-Modified-Since: Sat, 25 Feb 2012 12:55:04 GMT 字段，询问当前缓存的页面是

否是最新的，如果是最新的就返回 304 状态码，告诉浏览器是最新的，服务器也不会传输新的数据。

与 Last-Modified 字段有类似功能的还有一个 Etag 字段，这个字段的作用是让服务端给每个页面分配一个唯一的编号，然后通过这个编号来区分当前这个页面是否是最新的。这种方式比使用 Last-Modified 更加灵活，但是在后端的 Web 服务器有多台时比较难处理，因为每个 Web 服务器都要记住网站的所有资源，否则浏览器返回这个编号就没有意义了。

1.4 DNS 域名解析

我们知道互联网都是通过 URL 来发布和请求资源的，而 URL 中的域名需要解析成 IP 地址才能与远程主机建立连接，如何将域名解析成 IP 地址就属于 DNS 解析的工作范畴。

可以毫不夸张地说，虽然我们平时上网感觉不到 DNS 解析的存在，但是一旦 DNS 解析出错，可能会导致非常严重的互联网灾难。目前世界上的整个互联网有几个 DNS 根域名服务器，任何一台根服务器坏掉后果都会非常严重。

1.4.1 DNS 域名解析过程

图 1-10 是 DNS 域名解析的主要请求过程实例图。

如图 1-10 所示，当一个用户在浏览器中输入 `www.abc.com` 时，DNS 解析将会有将近 10 个步骤，这个过程大体描述如下。

当用户在浏览器中输入域名并按下回车键后，第 1 步，浏览器会检查缓存中有没有这个域名对应的解析过的 IP 地址，如果缓存中有，这个解析过程就将结束。浏览器缓存域名也是有限制的，不仅浏览器缓存大小有限制，而且缓存的时间也有限制，通常情况下为几分钟到几小时不等，域名被缓存的时间限制可以通过 TTL 属性来设置。这个缓存时间太长和太短都不好，如果缓存时间太长，一旦域名被解析到的 IP 有变化，会导致被客户端缓存的域名无法解析到变化后的 IP 地址，以致该域名不能正常解析，这段时间内有可能有一部分用户无法访问网站。如果时间设置太短，会导致用户每次访问网站都要重新解析一次域名。

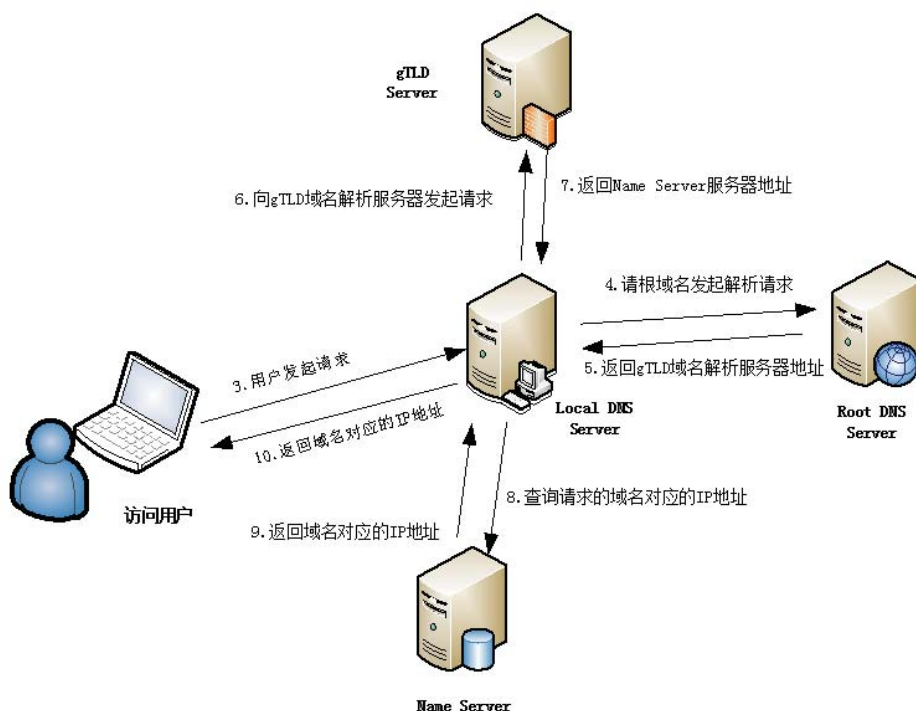


图 1-10 DNS 域名解析

第 2 步，如果用户的浏览器缓存中没有，浏览器会查找操作系统缓存中是否有这个域名对应的 DNS 解析结果。其实操作系统也会有一个域名解析的过程，在 Windows 中可以通过 `C:\Windows\System32\drivers\etc\hosts` 文件来设置，你可以将任何域名解析到任何能够访问的 IP 地址。如果你在这里指定了一个域名对应的 IP 地址，那么浏览器会首先使用这个 IP 地址。例如，我们在测试时可以将一个域名解析到一台测试服务器上，这样不用修改任何代码就能测试到单独服务器上的代码的业务逻辑是否正确。正是因为有这种本地 DNS 解析的规程，所以黑客就有可能通过修改你的域名解析来把特定的域名解析到它指定的 IP 地址上，导致这些域名被劫持。

这导致早期的 Windows 版本中出现过很严重的问题，而且对于一般没有太多电脑知识的用户来说，出现问题后很难发现，即使发现也很难自己解决，所以 Windows 7 中将 `hosts` 文件设置成了只读的，防止这个文件被轻易修改。

在 Linux 中这个配置文件是 `/etc/named.conf`，修改这个文件可以达到同样的目的，当

解析到这个配置文件中的某个域名时，操作系统会在缓存中缓存这个解析结果，缓存的时间同样是受这个域名的失效时间和缓存的空间大小控制的。

前面这两个步骤都是在本机完成的，所以在图 1-10 中没有表示出来。到这里还没有涉及真正的域名解析服务器，如果在本机中仍然无法完成域名的解析，就会真正请求域名服务器来解析这个域名了。

第 3 步，如何、怎么知道域名服务器呢？在我们的网络配置中都会有“DNS 服务器地址”这一项，这个地址就用于解决前面所说的如果两个过程无法解析时要怎么办，操作系统会把这个域名发送给这里设置的 LDNS，也就是本地区的域名服务器。这个 DNS 通常都提供给你本地互联网接入的一个 DNS 解析服务，例如你是在学校接入互联网，那么你的 DNS 服务器肯定在你的学校，如果你是在一个小区接入互联网的，那这个 DNS 就是提供给你接入互联网的应用提供商，即电信或者联通，也就是通常所说的 SPA，那么这个 DNS 通常也会在你所在城市的某个角落，通常不会很远。在 Windows 下可以通过 ipconfig 查询这个地址，如图 1-11 所示。

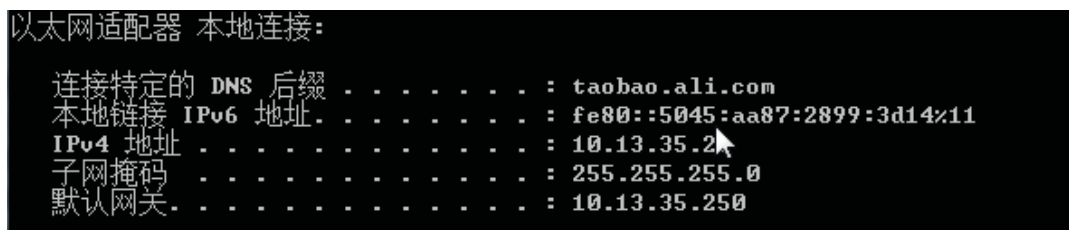


图 1-11 在 Windows 中查询 DNS Server

在 Linux 下可以通过如下方式查询配置的 DNS Server，如图 1-12 所示。

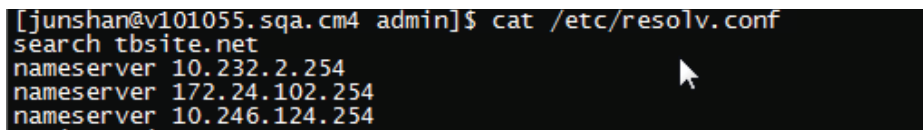


图 1-12 在 Linux 中下查询 DNS Server

这个专门的域名解析服务器性能都会很好，它们一般都会缓存域名解析结果，当然缓存时间是受域名的失效时间控制的，一般缓存空间不是影响域名失效的主要因素。大约 80% 的域名解析都到这里就已经完成了，所以 LDNS 主要承担了域名的解析工作。

第 4 步，如果 LDNS 仍然没有命中，就直接到 Root Server 域名服务器请求解析。

第 5 步，根域名服务器返回给本地域名服务器一个所查询域的主域名服务器（gTLD Server）地址。gTLD 是国际顶级域名服务器，如 .com、.cn、.org 等，全球只有 13 台左右。

第 6 步，本地域名服务器（Local DNS Server）再向上一步返回的 gTLD 服务器发送请求。

第 7 步，接受请求的 gTLD 服务器查找并返回此域名对应的 Name Server 域名服务器的地址，这个 Name Server 通常就是你注册的域名服务器，例如你在某个域名服务提供商申请的域名，那么这个域名解析任务就由这个域名提供商的服务器来完成。

第 8 步，Name Server 域名服务器会查询存储的域名和 IP 的映射关系表，正常情况下都根据域名得到目标 IP 记录，连同一个 TTL 值返回给 DNS Server 域名服务器。

第 9 步，返回该域名对应的 IP 和 TTL 值，Local DNS Server 会缓存这个域名和 IP 的对应关系，缓存的时间由 TTL 值控制。

第 10 步，把解析的结果返回给用户，用户根据 TTL 值缓存在本地系统缓存中，域名解析过程结束。

在实际的 DNS 解析过程中，可能还不止这 10 个步骤，如 Name Server 也可能有多级，或者有一个 GTM 来负载均衡控制，这都有可能影响域名解析的过程。

1.4.2 跟踪域名解析过程

在 Linux 和 Windows 下都可以用 nslookup 命令来查询域名的解析结果，如图 1-13 所示。

```
[junshan@v101055.sqa.cm4 admin]$ nslookup
> www.taobao.com
Server:      10.232.2.254
Address:     10.232.2.254#53

Non-authoritative answer:
www.taobao.com canonical name = www.gslb.taobao.com.
Name:   www.gslb.taobao.com
Address: 115.238.23.251
Name:   www.gslb.taobao.com
Address: 115.238.23.241
>
```

图 1-13 用 nslookup 查询域名解析结果

在 Linux 系统中还可以使用 dig 命名来查询 DNS 的解析过程，如下所示：

```
[junshan@v101055.sqa.cm4 admin]$ dig www.taobao.com

; <<>> DiG 9.3.6-P1-RedHat-9.3.6-4.P1.el5 <<>> www.taobao.com
;; global options: printcmd
;; Got answer:
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16903
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;www.taobao.com.                IN      A

;; ANSWER SECTION:
www.taobao.com.      1542    IN      CNAME   www.gslb.taobao.com.
www.gslb.taobao.com.  130     IN      A       115.238.23.251
www.gslb.taobao.com.  130     IN      A       115.238.23.241

;; AUTHORITY SECTION:
gslb.taobao.com.     70371   IN      NS       gslbns3.taobao.com.
gslb.taobao.com.     70371   IN      NS       gslbns1.taobao.com.
gslb.taobao.com.     70371   IN      NS       gslbns2.taobao.com.

;; ADDITIONAL SECTION:
gslbns1.taobao.com.  452     IN      A       121.0.23.218
gslbns2.taobao.com.  452     IN      A       115.124.17.70
gslbns3.taobao.com.  452     IN      A       110.75.3.193

;; Query time: 5 msec
;; SERVER: 10.232.2.254#53(10.232.2.254)
;; WHEN: Sun Feb 12 19:19:05 2012
;; MSG SIZE rcvd: 201
```

结果的第 1 行输出了当前 Linux 的版本号，第 2 行说明可以增加可选参数 `printcmd`，如果加上 `printcmd`，打印出来的结果如下：

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 58602
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;printcmd.                IN      A

;; AUTHORITY SECTION:
.      10800   IN      SOA      a.root-servers.net. nstld.
verisign-grs.com. 2012021200 1800 900 604800 86400

;; Query time: 208 msec
```

```
;; SERVER: 10.232.2.254#53(10.232.2.254)
;; WHEN: Sun Feb 12 19:20:59 2012
;; MSG SIZE rcvd: 101
```

“QUESTION SECTION”部分表示当前查询的域名是一个 A 记录，“ANSWER SECTION”部分返回了这个域名由 CNAME 到 www.gslb.taobao.com，返回了这个域名对应的 IP 地址。

还可通过增加+trace 参数跟踪这个域名的解析过程，如下所示：

```
[junshan@v101055.sqa.cm4 admin]$ dig www.taobao.com +trace

; <<>> DiG 9.3.6-P1-RedHat-9.3.6-4.P1.el5 <<>> www.taobao.com +trace
;; global options: printcmd
.                449398 IN      NS      k.root-servers.net.
.                449398 IN      NS      l.root-servers.net.
.                449398 IN      NS      m.root-servers.net.
.                449398 IN      NS      a.root-servers.net.
.                449398 IN      NS      b.root-servers.net.
.                449398 IN      NS      c.root-servers.net.
.                449398 IN      NS      d.root-servers.net.
.                449398 IN      NS      e.root-servers.net.
.                449398 IN      NS      f.root-servers.net.
.                449398 IN      NS      g.root-servers.net.
.                449398 IN      NS      h.root-servers.net.
.                449398 IN      NS      i.root-servers.net.
.                449398 IN      NS      j.root-servers.net.
;; Received 272 bytes from 10.232.2.254#53(10.232.2.254) in 0 ms

com.             172800 IN      NS      a.gtld-servers.net.
com.             172800 IN      NS      b.gtld-servers.net.
com.             172800 IN      NS      c.gtld-servers.net.
com.             172800 IN      NS      d.gtld-servers.net.
com.             172800 IN      NS      e.gtld-servers.net.
com.             172800 IN      NS      f.gtld-servers.net.
com.             172800 IN      NS      g.gtld-servers.net.
com.             172800 IN      NS      h.gtld-servers.net.
com.             172800 IN      NS      i.gtld-servers.net.
com.             172800 IN      NS      j.gtld-servers.net.
com.             172800 IN      NS      k.gtld-servers.net.
com.             172800 IN      NS      l.gtld-servers.net.
```

```

com.                172800  IN      NS      m.gtld-servers.net.
;; Received 492 bytes from 193.0.14.129#53(k.root-servers.net) in 607 ms

taobao.com.         172800  IN      NS      ns1.taobao.com.
taobao.com.         172800  IN      NS      ns2.taobao.com.
taobao.com.         172800  IN      NS      ns3.taobao.com.
;; Received 134 bytes from 192.5.6.30#53(a.gtld-servers.net) in 250 ms

www.taobao.com.     1800    IN      CNAME   www.gslb.taobao.com.
gslb.taobao.com.    86400   IN      NS      gslbns2.taobao.com.
gslb.taobao.com.    86400   IN      NS      gslbns3.taobao.com.
gslb.taobao.com.    86400   IN      NS      gslbns1.taobao.com.
;; Received 169 bytes from 110.75.1.19#53(ns1.taobao.com) in 0 ms

```

上面清楚地显示了整个域名是如何发起和解析的,从根域名(.)到 gTLD Server(.com.)再到 Name Server (taobao.com.) 的整个过程都显示出来了。还可以看出 DNS 的服务器有多个备份,可以从任何一台查询到解析结果。

1.4.3 清除缓存的域名

我们知道 DNS 域名解析后会缓存解析结果,其中主要在两个地方缓存结果,一个是 Local DNS Server, 另外一个用户的本地机器。这两个缓存都是 TTL 值和本机缓存大小控制的,但是最大缓存时间是 TTL 值,基本上 Local DNS Server 的缓存时间就是 TTL 控制的,很难人工介入,但是我们的本机缓存可以通过如下方式清除。

在 Windows 下可以在命令行模式下执行 `ipconfig /flushdns` 命令来刷新缓存,如图 1-14 所示。



```

C:\Windows\System32>ipconfig /flushdns

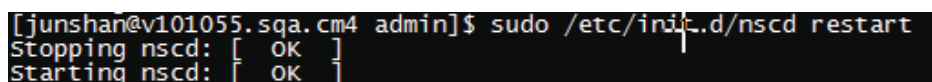
Windows IP 配置

已成功刷新 DNS 解析缓存。

```

图 1-14 在 Windows 下刷新 DNS 缓存

在 Linux 下可以通过 `/etc/init.d/nscd restart` 来清除缓存,如图 1-15 所示。



```

[junshan@v101055.sqa.cm4 admin]$ sudo /etc/init.d/nscd restart
Stopping nscd: [ OK ]
Starting nscd: [ OK ]

```

图 1-15 在 Linux 下清除 DNS 缓存

重启依然是解决很多问题的第一选择。

在 Java 应用中 JVM 也会缓存 DNS 的解析结果，这个缓存是在 `InetAddress` 类中完成的，而且这个缓存时间还比较特殊，它有两种缓存策略：一种是正确解析结果缓存，另一种是失败的解析结果缓存。这两个缓存时间由两个配置项控制，配置项是在 `%JAVA_HOME%\lib\security\java.security` 文件中配置的。两个配置项分别是 `networkaddress.cache.ttl` 和 `networkaddress.cache.negative.ttl`，它们的默认值分别是 -1（永不失效）和 10（缓存 10 秒）。

要修改这两个值同样有几种方式，分别是：直接修改 `java.security` 文件中的默认值、在 Java 的启动参数中增加 `-Dsun.net.inetaddr.ttl=xxx` 来修改默认值、通过 `InetAddress` 类动态修改。

在这里还要特别强调一下，如果我们需要用 `InetAddress` 类解析域名时，一定要是单例模式，不然会有严重的性能问题，如果每次都创建 `InetAddress` 实例，每次都要进行一次完整的域名解析，非常耗时，这点要特别注意。

1.4.4 几种域名解析方式

域名解析记录主要分为 A 记录、MX 记录、CNAME 记录、NS 记录和 TXT 记录。

- ◎ A 记录，A 代表的是 Address，用来指定域名对应的 IP 地址，如将 `item.taobao.com` 指定到 `115.238.23.241`，将 `switch.taobao.com` 指定到 `121.14.24.241`。A 记录可以将多个域名解析到一个 IP 地址，但是不能将一个域名解析到多个 IP 地址。
- ◎ MX 记录，表示的是 Mail Exchange，就是可以将某个域名下的邮件服务器指向自己的 Mail Server，如 `taobao.com` 域名的 A 记录 IP 地址是 `115.238.25.245`，如果 MX 记录设置为 `115.238.25.246`，是 `xxx@taobao.com` 的邮件路由，DNS 会将邮件发送到 `115.238.25.246` 所在的服务器，而正常通过 Web 请求的话仍然解析到 A 记录的 IP 地址。
- ◎ CNAME 记录，全称是 Canonical Name（别名解析）。所谓的别名解析就是可以为一个域名设置一个或者多个别名。如将 `taobao.com` 解析到 `xulingbo.net`，将 `srcfan.com` 也解析到 `xulingbo.net`。其中 `xulingbo.net` 分别是 `taobao.com` 和 `srcfan.com` 的别名。前面的跟踪域名解析中的“`www.taobao.com. 1542 IN CNAME www.gslb.taobao.com`”就是 CNAME 解析。

- ⊙ NS 记录，为某个域名指定 DNS 解析服务器，也就是这个域名有指定的 IP 地址的 DNS 服务器去解析，前面的“`gslb.taobao.com. 86400 IN NS gslbns2.taobao.com.`”就是 NS 解析。
- ⊙ TXT 记录，为某个主机名或域名设置说明，如可以为 `xulingbo.net` 设置 TXT 记录为“君山的博客|许令波”这样的说明。

1.5 CDN 工作机制

CDN 也就是内容分布网络（Content Delivery Network），它是构筑在现有 Internet 上的一种先进的流量分配网络。其目的是通过在现有的 Internet 中增加一层新的网络架构，将网站的内容发布到最接近用户的网络“边缘”，使用户可以就近取得所需的内容，提高用户访问网站的响应速度。有别于镜像，它比镜像更智能，可以做这样一个比喻：CDN = 镜像(Mirror)+ 缓存(Cache)+ 整体负载均衡(GSLB)。因而，CDN 可以明显提高 Internet 中信息流动的效率。

目前 CDN 都以缓存网站中的静态数据为主，如 CSS、JS、图片和静态页面等数据。用户在从主站服务器请求到动态内容后再从 CDN 上下载这些静态数据，从而加速网页数据内容的下载速度，如淘宝有 90% 以上的数据都是由 CDN 来提供的。

通常来说 CDN 要达到以下几个目标。

- ⊙ 可扩展（Scalability）。性能可扩展性：应对新增的大量数据、用户和事务的扩展能力；成本可扩展性：用低廉的运营成本提供动态的服务能力和高质量的内容分发。
- ⊙ 安全性（Security）。强调提供物理设备、网络、软件、数据和服务过程的安全性，（趋势）减少因为 DDoS 攻击或者其他恶意行为造成商业网站的业务中断。
- ⊙ 可靠性、响应和执行（Reliability、Responsiveness 和 Performance）。服务可用性，能够处理可能的故障和用户体验的下降，通过负载均衡及时提供网络的容错机制。

1.5.1 CDN 架构

通常的 CDN 架构如图 1-16 所示。

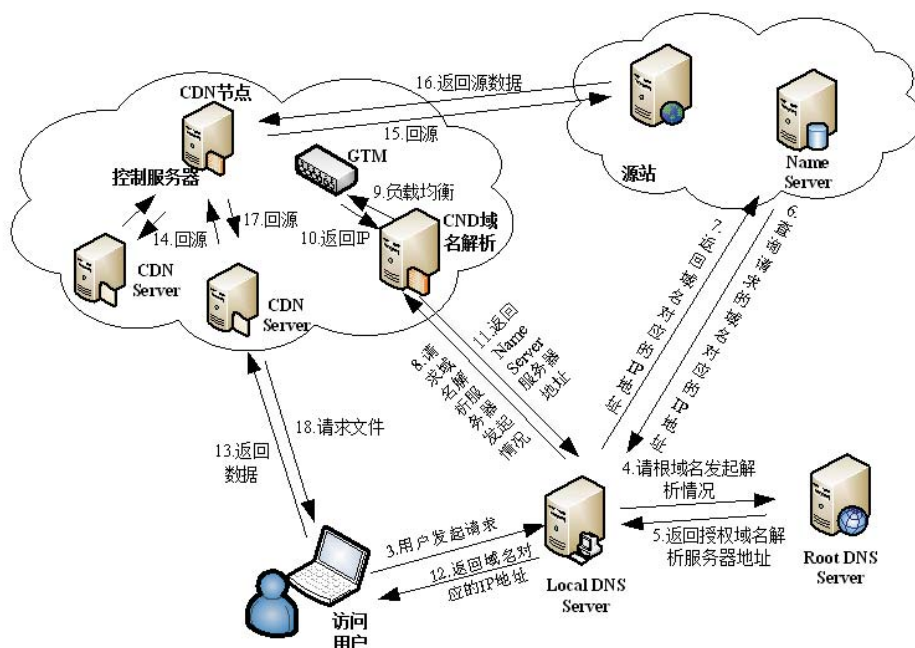


图 1-16 CDN 架构

如图 1-16 所示，一个用户访问某个静态文件（如 CSS 文件），这个静态文件的域名假如是 `cdn.taobao.com`，那么首先要向 Local DNS 服务器发起请求，一般经过迭代解析后回到这个域名的注册服务器去解析，一般每个公司都会有一个 DNS 解析服务器。这时这个 DNS 解析服务器通常会把它重新 CNAME 解析到另外一个域名，而这个域名最终会被指向 CDN 全局中的 DNS 负载均衡服务器，再由这个 GTM 来最终分配是哪个地方的访问用户，返回给离这个访问用户最近的 CDN 节点。

拿到 DNS 解析结果，用户就直接去这个 CDN 节点访问这个静态文件了，如果这个节点中所请求的文件不存在，就会再回到源站去获取这个文件，然后再返回给用户。

1.5.2 负载均衡

负载均衡（Load Balance）就是对工作任务进行平衡、分摊到多个操作单元上执行，如图片服务器、应用服务器等，共同完成工作任务。它可以提高服务器响应速度及利用效

率，避免软件或者硬件模块出现单点失效，解决网络拥塞问题，实现地理位置无关性，为用户提供较一致的访问质量。

通常有三种负载均衡架构，分别是链路负载均衡、集群负载均衡和操作系统负载均衡。所谓链路负载均衡也就是前面提到的通过 DNS 解析成不同的 IP，然后用户根据这个 IP 来访问不同的目标服务器，如图 1-17 所示。

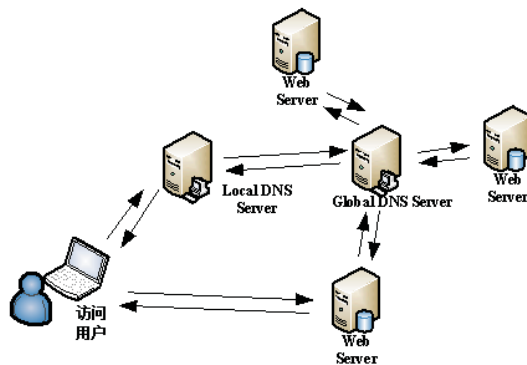


图 1-17 链路负载均衡

负载均衡是由 DNS 的解析来完成的，用户最终访问哪个 Web Server 是由 DNS Server 来控制的，在这里就是由 Global DNS Server 来动态解析域名服务。这种 DNS 解析的优点是用户会直接访问目标服务器，而不需要经过其他的代理服务器，通常访问速度会更快。但是也有缺点，由于 DNS 在用户本地和 Local DNS Server 都有缓存，一旦某台 Web Server 挂掉，那么很难及时更新用户的域名解析结构。如果用户的域名没有及时更新，那么用户将无法访问这个域名，带来的后果非常严重。

集群负载均衡是另外一种常见的负载均衡方式，它一般分为硬件负载均衡和软件负载均衡。硬件负载均衡一般使用一台专门硬件设备来转发请求，如图 1-18 所示。

硬件负载均衡的关键就是这台价格非常昂贵的设备，如 F5，通常为了安全需要一主一备。它的优点很显然就是性能非常好，缺点就是非常贵，一般公司是用不起的，还有就是当访问量陡然增大超出服务极限时，不能进行动态扩容。

软件负载均衡是使用最普遍的一种负载方式，它的特点是使用成本非常低，直接使用廉价的 PC 就可以搭建。缺点就是一般一次访问请求要经过多次代理服务器，会增加网络

延时。它的架构通常如图 1-19 所示。

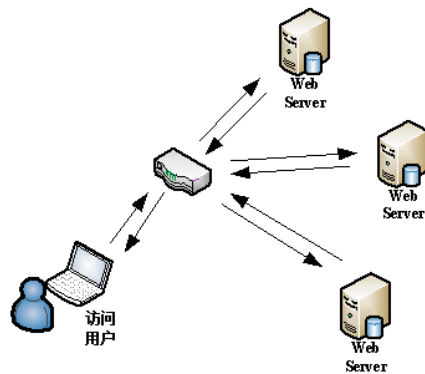


图 1-18 硬件负载均衡

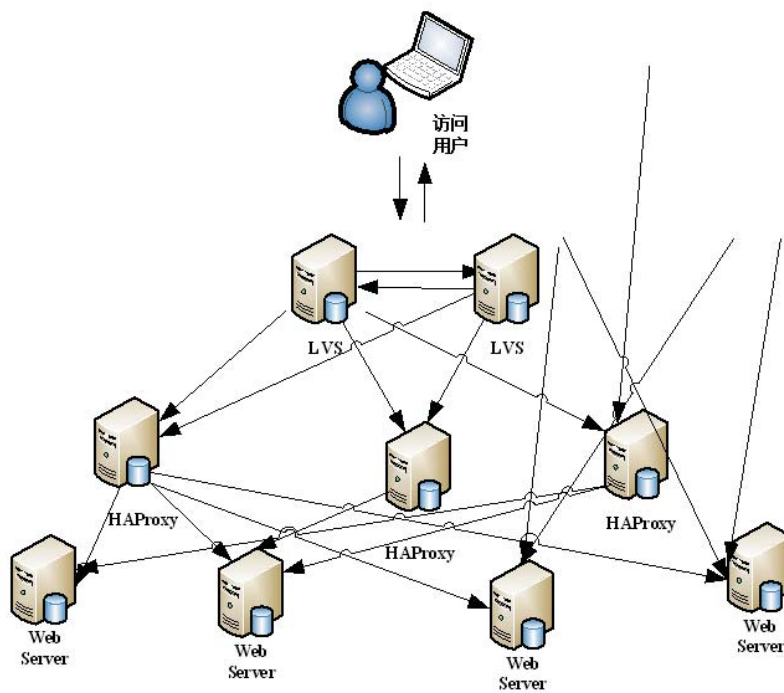


图 1-19 软件负载均衡

图 1-19 中上面两台是 LVS，使用四层负载均衡，也就是在网络层利用 IP 地址进行地址转发。下面三台使用 HAProxy 进行七层负载，也就是可以根据访问用户的 HTTP 请求头来进行负载均衡，如可以根据不同的 URL 来将请求转发到特定机器或者根据用户的 Cookie 信息来指定访问的机器。

最后一种是操作系统负载均衡，就是利用操作系统级别的软中断或者硬件中断来达到负载均衡，如可以设置多队列网卡等来实现。

这几种负载均衡不仅在 CDN 的集群中能使用，而且在 Web 服务或者分布式数据集群中同样也能使用，但是在这些地方后两种使用得要多一点。

1.6 总结

本章主要介绍了前端的一些基本知识，包括在用户端发起一个请求时，这个请求都经过了哪些服务单元，进行了哪些处理。本章可以帮助我们对 B/S 网络架构有个整体的认识。