第1章 引言

1.1 起源

首先，坦白地说，开发 Fiddler Web 调试器时，并没有什么伟大的愿景或期望——要做一个全世界最受欢迎的调试代理。它只是由具体需求触发，应运而生。我从未打算构建一个功能如此灵活而强大的复杂平台。因为它的复杂性，我不得不花费九个月的时间来写这本书，为的就是介绍如何充分利用 Fiddler。我们真的做到了！

在深入介绍技术细节之前，首先分享一下 Fiddler 背后的故事。

1999 年春天，我还是马里兰大学的学生，得到了微软的一个新团队的程序经理（Program Manager）职位的面试机会。在最后一轮面试中，面试官的第一个问题是“HTTP 是如何工作的？”我对此只是略知皮毛，因此给了个不完整也不太准确的回答，但还不至于让自己很难堪。从那个暑假开始，我的工作是参与第一版 SharePoint 的开发。我偶尔会使用 Microsoft Network Monitor （NetMon）查看网络数据流。Microsoft Network Monitor（NetMon）是一个功能强大的数据包探嗅器（packet sniffer），但很原始很难用。2001 年暑假刚开始，我正式加入微软公司，工作职位是 Office Clip Art organizer 客户端和网站的程序经理。

当时，我所在团队的大多数开发和测试人员都不熟悉 Web 开发，他们之前主要是用 C 和 C++实现本地运行的应用。很快，调试过程过于繁琐这个问题就凸显出来——很多同事都不愿意使用 NetMon。我甚至看到一些开发人员用如下方式调试 HTTP 请求——把鼠标停留在 Visual Studio 的某些变量上，查看十六进制形式的原始数据流，如图 1-1 所示。

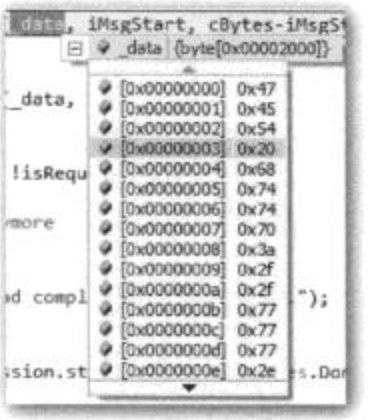

图 1-1

在过去几年，我编写了一些小工具，因此有信心实现一个使得 Web 调试变得简单的工具。最初的方案是基于已有的 C++代理服务器，对其做些修改，从而可以把 HTTP 流输出到系统控制台，如图 1-2 所示。

图 1-2

这种方案说它多差都不为过——这个代理无法处理安全数据流或认证协议。非文本形式的内容的显示也是个问题——可笑的是，该工具还会把二进制内容当成 ASCII 码显示。老式控制台用户可能还记得八进制的 0x07 代表的是字符“bell”，因此当控制台显示 0x07 时，系统就会发出声响。因此，这个调试代理发布后，由于测试人员使用时会遇到二进制数据流，Office Online 团队的走廊就会不断响起像拉斯维加斯的赌场那样的声响。

虽然有烦人的缺陷，但这个工具还是很受欢迎，这激发我开始考虑下一个版本。我使用 Borland Delphi 快速实现了一个小演示程序，Borland Delphi 是我当时工作时最常用的开发工具。其彩色 UI 界面是 Fiddler 的最终外观的基础，如图 1-3 所示。

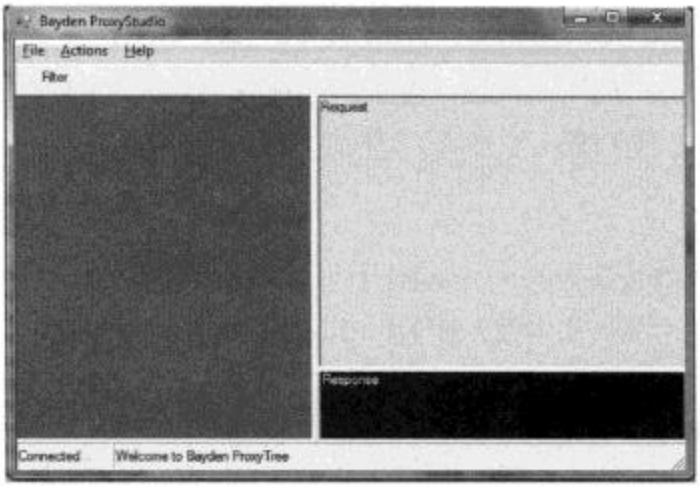

图 1-3

然而，在考虑了使用本地代码编写代理服务器所涉及的安全和内存管理问题后，我决定使用 C#语言来实现新版本，该语言是由 Visual Studio 团队开发的，我一个很好的朋友加入了该团队。于我而言，通过.NET 从零开始实现 HTTP 代理服务器面临两大挑战：一是我不是很了解 HTTP 是如何工作的；二是我不会用 C#。

幸运的是，花钱买几本书，以及利用大量的周末闲余时间，我很快克服了这两个不足。其中有两本书是我的良师益友：《HTTP: The Definitive Guide》和《C# Cookbook》。通过一章章的学习，我了解了 HTTP 是如何工作，以及如何用 C# .NET 编程，也开始慢慢地实现 Fiddler。大约半年后，我完成了 Fiddler 的第一个版本，如图 1-4 所示。

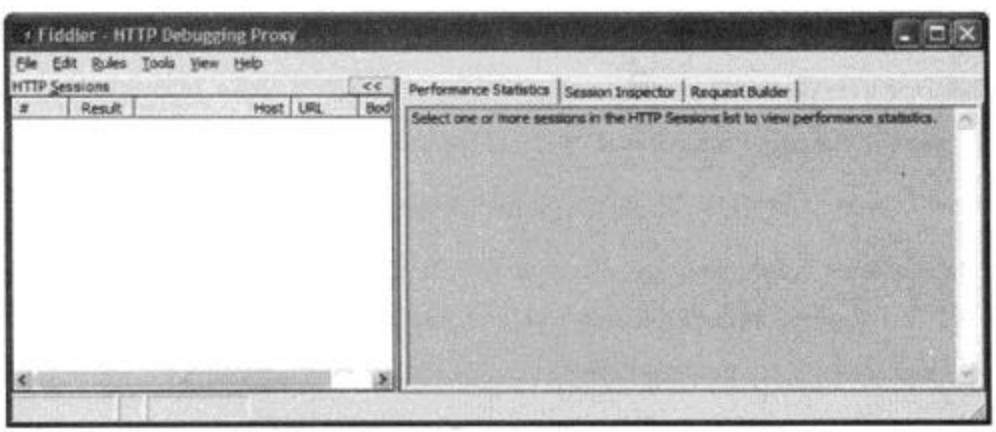

图 1-4

同样，这个版本也存在很多不足（错误也是不计其数），但是同事们都积极采用了它，因

为之前的版本不断地发出蜂鸣声，他们电脑的音箱都快崩溃了。

随后几年，Fiddler 有了很大改进，它包含两个扩展模块，一是自动生成和修改响应的机制，二是支持加密的（HTTPS）数据流、FTP 和 HTML5 WebSockets。

关于本书

历经九年以及无数次版本更新，Fiddler 已经发展成一个强大的工具和平台，可以执行各种任务。它有一个可扩展性相当强大的模型，以及一个组件开发者社区，他们又进一步增强了 Fiddler 作为性能、安全和负载测试工具的价值。从过去几年的电子邮件咨询、在线讨论小组以及无数次的会议来看，大多数用户只是用到了 Fiddler 很少的一部分功能。我开始意识到如果有关于该工具的完整指南，很多用户可以更好地利用它。本书就是由此而来。

作为 Fiddler 的作者，我发现写这本书既简单又具有挑战性。其简单性在于我非常了解 Fiddler，比如其底层实现，而且对于一些晦涩的细节查看源代码就可以了解。其挑战性在于每当我选择写某个有趣的场景或功能时，将迫使我对其进行深入的思考。通常情况下，我写着写着就“跑偏”了，转而写代码对 Fiddler 进行改进，改进之后，原来的主题通常只需要很少的篇幅来介绍甚至不再需要介绍。最后，我重写了本书的大部分内容和 Fiddler 工具本身。这个过程很漫长，但工具和书之间彼此也相得益彰。

本书的出版时间和 Fiddler 2.4.0.0 版本的发布时间基本一致，大致都在 2012 年初夏。如果你使用的是更高版本的 Fiddler，你会发现一些细微差别，但其核心概念一致。

本书几乎涵盖了 Fiddler 和 FiddlerCore 的每个主题，但它不是关于 HTTP、SSL、HTML、Web Services 或深入了解 Fiddler 所需的很多其他主题的教程。如果你想深入了解网络协议，我建议你查看以下资料，我在开发 Fiddler 的过程中一直在参考这些资料：

- Hypertext Transfer Protocol -- HTTP/1.1，网址为 http://www.ietf.org/rfc/rfc2616.txt；
- David Gourley 的《HTTP: The Definitive Guide》；
- Balachander Krishnamurthy 和 Jennifer Rexford 的《Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement》；
- Stephen A. Thomas 的《SSL & TLS Essentials: Securing the Web》。

关于本书的阅读方式，你可以从头到尾阅读本书，也可以通过目录和索引来查找自己感兴趣的话题。我建议你阅读本书的所有章节，即使其中某些章节可能和你不相关，因为每章都包含了一些独有的建议和技巧。

我很推崇从下一节开始阅读，它介绍了一些术语和基础概念，可以帮助你更好地了解Fiddler以及本书。

希望你喜欢本书！

1.2 快速入门

本节将介绍Fiddler的一些基础知识，这些内容可帮助你入门并为本书后续的学习奠定基础。

1.2.1 基本概念

Fiddler是一款基于Windows系统的专用代理服务器软件。本地运行的程序，如Web浏览器、Office应用程序以及其他客户端应用，可以把HTTP和HTTPS请求发送给Fiddler，Fiddler通常把这些请求转发给Web服务器。然后，服务器把这些请求的响应返回给Fiddler，Fiddler再把这些响应转发给客户端。

几乎所有使用网络协议的程序都支持代理服务器，因此Fiddler几乎适用于所有应用。当启动Fiddler来捕获请求和响应时，Fiddler会自动注册为Windows Internet (WinINET)网络服务代理，并请求所有应用把请求发送给它，如图1-5所示。

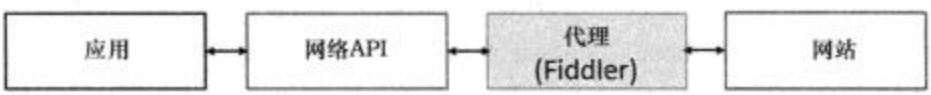

图1-5

有些应用不会自动识别Windows网络配置，要想让Fiddler捕获其数据流，需要手工配置这些应用。也可以对Fiddler进行配置，使它能够支持更罕见的场景，包括服务器到服务器（如Web Services）和设备到服务器（如iPad或Windows手机）的数据流。默认情况下，Fiddler被设计成能够自动挂接到（chain to）先前已经部署的各种上游代理（upstream proxy），通过这种方式，Fiddler能够在已经使用了代理服务器的网络环境中正常工作。

Fiddler可以捕获所有的本地数据流，而且支持很多过滤器（filter）。这一特性使得用户既可以隐藏不感兴趣的数据流，也可以高亮显示（使用颜色或字体选项）感兴趣的数据流。过滤器的过滤条件可以是数据流的来源（如特定的客户端进程），也可以是数据流本身的某些特征（如该数据流所绑定的主机名或服务器返回内容的类型）。

Fiddler 的扩展模型功能丰富，既有简单的 FiddlerScript，也包含强大的功能插件（Extensions），这些扩展可以使用任意一种.NET 语言来开发。Fiddler 还支持几种特殊的扩展类型，其中最流行的是 Inspector（稽查员）。顾名思义，Inspector 的特点在于支持查看单个请求或响应。Inspector 可以做成支持所有响应类型（如 HexView Inspector），也可以定制为只支持特定类型（如 JSON Inspector）。如果你是开发人员，可以通过类库 FiddlerCore 把 Fiddler 的核心代理引擎植入到自己的应用中。

Fiddler 可以解密 HTTPS 数据流，使用“中间人（man-in-the-middle）”[1]解密技术来显示和修改这些在网络上不可读的请求。要做到没有安全告警的无缝调试，可以把 Fiddler 的根证书（root certificate）安装到系统或 Web 浏览器的受信任的证书库中。

一个 Web 会话（Web Session）代表客户端和服务器之间的一个事务。Fiddler 界面中左侧边栏的 Web 会话列表中的一个条目就是一个会话。一个会话对象包含一个请求和一个响应，分别表示客户端发送给服务器的数据以及服务器返回给客户端的数据。会话对象还会维护一组标志位（Flag），用于保存会话的元数据以及在处理该会话过程中记录的时间戳对象（Timer）。

代理服务器并非局限于查看网络流量——顾名思义，Fiddler 这个名字的含义在于它可以“随意玩弄（fiddle）”发送出去的请求和接收进来的响应。可以设置请求或响应断点以手工修改数据流。到达设置的断点后，Fiddler 会暂停会话，以便用户手工修改请求和响应。Fiddler 还支持在其内部运行脚本或扩展来改写数据流。默认情况下，Fiddler 是在缓冲模式（buffering mode）下工作的，即 Fiddler 是接收到服务器的完整响应之后才将响应的内容返回给客户端。如果启用流模式（streaming mode），Fiddler 会将从服务器端接收到的响应立即返回给客户端。因此，流模式下不支持对服务器的响应进行修改。

Fiddler 捕获的会话信息可以保存到会话归档（Session Archive Zip，SAZ)文件中，以方便后续查看。这种格式的压缩文件中包含了完整的请求和响应、标志位、时间戳对象以及其他元数据。非技术人员可以使用一个轻量级的捕获工具 FiddlerCap 来生成 SAZ 文件，再把这些文件提供给专业人员进行分析。Fiddler 支持扩展组件 Exporter，它支持把捕获到的会话以其他工具支持的格式来保存，从而保证了 Fiddler 和这些工具的互通性。同样，Fiddler 也支持扩展组件 Importer，它支持 Fiddler 加载以其他格式存储的数据流，包括很多浏览器开发工具所使用的 HTTP Archive（HAR）格式。

---

[1] 译注：“中间人”这个说法是借用了网络上常见的“中间人攻击（Man-in-the-Middle Attack）”（参考 http://baike.baidu.com/view/1531871.htm），指 Fiddler 在两台通信的计算机之间充当“中间人”角色。

1.2.2 使用场景

我最经常遇到的一些问题类似于："我可以使用 Fiddler 来完成<X>吗"？Fiddler 适用于很多场景中。但是，对于一些场景，Fiddler 是不合适的。绝大多数情况下，人们所使用的是 Fiddler 的少数几个功能。以下将简单说明 Fiddler 所适用以及不适用的场景。

Fiddler 支持功能的不完整列表

- 查看几乎所有的浏览器、客户端应用或服务之间的 Web 数据流。
- 手动或自动修改任意的请求或响应。
- 解密 HTTPS 数据流以便查看和修改。
- 归档捕获到的数据流，支持在不同的计算机上加载这些数据。
- 给客户端应用"回放（play back）"先前捕获到的响应，即使当前服务器处于脱机状态。
- 绝大多数 PC 和各种设备之间的 Web 数据流的调试，包括 Mac/Linux 系统、智能手机和平板电脑。
- 挂接到（chain to）上游代理服务器，包括 TOR 网络[1]。
- 作为反向代理运行，在不需要重新配置客户端计算机或设备的情况下，在服务端捕获数据流。
- 随着基于 FiddlerScript 或.NET 可扩展模型实现的新功能的不断增加，Fiddler 将变得更加强大。

Fiddler 不支持功能的不完整列表

Fiddler 是一个非常灵活的工具，但某些功能目前还不支持。

- 调试非网络协议数据流。
  - ★ Fiddler 支持 HTTP、HTTPS 和 FTP 数据流以及相关的协议，如 HTML5 WebSockets 和 ICY 流。
  - ★ Fiddler 无法监测或修改基于其他协议的数据，如 SMTP、POP3、Telnet、IRC 等。
- 处理超大请求和响应。

[1] 译注：TOR 是专门防范对流量探嗅分析的软件项目。它通过由遍及全球的中继所组成的分布式网络转发通信，可以实现匿名访问网络。

- ★ 超过 2GB 的请求，Fiddler 无法处理。
- ★ 超过 2GB 的响应，Fiddler 的处理能力有限。
- ★ Fiddler 使用系统内存和页面文件（pagefile）来保存会话数据。保存大量的会话或超大的请求和响应会导致性能急剧下降。

- “神奇”地修复网站的错误（bug）。
  - ★ Fiddler 可以用来协助识别网络问题，但通常不能独立修复这些 bug。我已经收到过无数个电子邮件：“怎么回事？我已经安装了 Fiddler，但我的网站怎么还有很多 bug？！”

好了，完成了 Fiddler 的入门，接下来让我们一起深入探索它吧！

第2章 探索 Fiddler

2.1 入门

Fiddler 的官方网站为 http://getfiddler.com，强烈建议通过站点下载 Fiddler，因为有些恶意网站对其进行了重新打包，会附带安装一些其他软件（如广告软件或浏览器工具栏）。

2.1.1 系统需求

从 Windows XP 到 Windows 8，所有版本的 Windows 操作系统都支持 Fiddler。安装 Fiddler 唯一需要的就是预先安装 Microsoft .NET Framework 2.0 或更高版本。Windows Vista 或更高版本的系统中已经自带安装了.NET Framework，在 Windows XP 中可以使用 WindowsUpdate 来安装它。如果你的系统中只安装了.NET 4（以 Windows 8 系统为例，大多没有安装.NET 2.0、3.0 或 3.5），需要下载 Fiddler 4 包，它可以在.NET v4 CLR 上运行。和大部分.NET 程序一样，Fiddler 在 32 位操作系统中会以 32 位模式运行，而在 64 位操作系统下会以 64 位模式运行。Fiddler 在 64 位操作系统中的运行效果最好，即使这些系统的内存小于 4GB。

虽然不是必须，但是安装 Internet Explorer 9 或更高版本的 IE 后， Fiddler 可以提供更多的功能，因此强烈建议安装。特别说明一下，当安装了 IE 9 及更高版本的浏览器时，Fiddler 的 WebView Inspector 可以显示媒体类型，无需配置就可以查看发送到 http://localhost 的数据流。此外，修改 IE 的配置之后，就可以通过 http 请求头中的 X-Download-Initiator 来查看请求的来源。

安装 Fiddler 的基本功能需要 5MB 左右的磁盘空间，而安装最常用的插件也需要占用 5MB 左右的磁盘空间。内存不低于 512MB 的系统就可以运行 Fiddler，但当运行于内存大于 2GB 的系统中时，其性能会显著提高。

2.1.2 安装 Fiddler

Fiddler 的安装很简单——首先是接受最终用户许可协议条款（简单而言，协议的内容就是：一是不要做违法的事；二是作者不提供任何保证），然后选择 Fiddler 的安装目录。建议使用默认的文件夹作为安装路径——因为有些 Fiddler 组件会以 Fiddler 被安装在默认路径下为前提，如果 Fiddler 没有被安装在默认路径下，该组件将无法正常安装。Fiddler 的默认安装路径是 %ProgramFiles%\Fiddler2，在大多数系统中就是 C:\Program Files\Fiddler2\，而在 64 位的系统中则是 C:\Program Files (x86)\Fiddler2\。如果操作系统支持 64 位，Fiddler 本身会运行在 64 位模式下，而它之所以被安装在 32 位的 Program Files 文件夹中，通常仅仅是由于安装程序是 32 位的。

当成功地安装了 Fiddler 之后，会打开一个 Web 页面，显示使用 Fiddler 的一些关键信息。Fiddler 工具的启动图标会被安装在开始菜单（Start Menu）和浏览器的菜单栏中，在这两个地方都可以启动它。此外，你还可以通过快捷键“Windows+R”调出运行框，通过在其中输入 fiddler2 来启动。

权限和 XCOPY 部署

安装 Fiddler 需要管理员权限，因为它会更新计算机文件夹和注册表位置。如果没有管理员权限，可以采用常见的 XCOPY 部署（XCOPY Deployment）方式，即在另一台机器上安装 Fiddler，然后把它的%ProgramFiles%\Fiddler2\文件夹复制到目标机器或 USB 存储中。虽然采用这种方式时，Fiddler 的某些功能不可用（比如 IE 工具栏按钮和用于 Firefox 的 FiddlerHook 扩展插件），Fiddler 本身还是可以正常工作的。

2.1.3 更新 Fiddler

Fiddler 在启动时，会自动查询是否有新版本。当发现有新版本时，你会看到如图 2-1 所示的更新提示。

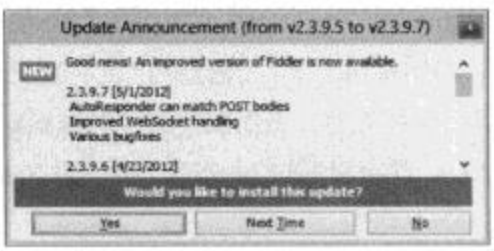

图 2-1

单击 Yes 按钮，Fiddler 会启动浏览器，下载最新的安装程序。下载完成后，关闭正在运行的 Fiddler，手动安装新下载的程序。如果单击的是 Next Time（下一次安装）这个按钮，下

一次启动 Fiddler 时，它会自动下载并安装最新版本。如果点击的是 No，弹出的对话框就会关闭，不会安装新的版本；下一次启动 Fiddler 时，这个提示还会弹出。

强烈建议安装最新版本的 Fiddler，因为每次更新通常都会增加新功能、提高性能以及修复 bug。当前，Fiddler 更新时不会自动对扩展进行更新，因此你需要不时地去登录那些扩展插件开发者的 Web 站点上自行检查扩展是否有新版本。

管理员如果禁止用户使用 Help > Check for Updates 命令，可以通过修改注册表的方式实现。打开 Windows 注册表编辑器，在 HKLM\SOFTWARE\Microsoft\Fiddler2 中新加一条记录，类型为 REG_STRING，名称为 BlockUpdateCheck，值为 True。

2.1.4 卸载 Fiddler

可以使用控制面板中的“添加/删除程序”来卸载 Fiddler。卸载后，系统并不会被清理干净，因此卸载无法解决配置问题。当遇到这些问题时，查看本书附录 A 以得到更多信息。

2.2 FIDDLER 用户界面

Fiddler 用户界面看起来非常复杂，因为它包含了大量的 Web 数据流信息，而且提供了很多自定义功能。

Fiddler 窗口的左边是 Web Sessions 列表，如图 2-2 所示其右边是各个选项视图（Views），显示在 Web Sessions 列表中选中的 Session 的信息。在窗口的最上方，主菜单栏（Main Menu）和工具栏（Toolbar）提供了各种操作的快捷入口。在窗口的最下方是状态栏（Status Bar），其中显示的是一些关键信息以及重要的命令。在 Web Sessions 列表下方的是一个很小的命令行窗口 QuickExec，它支持快速过滤和命令调用功能。

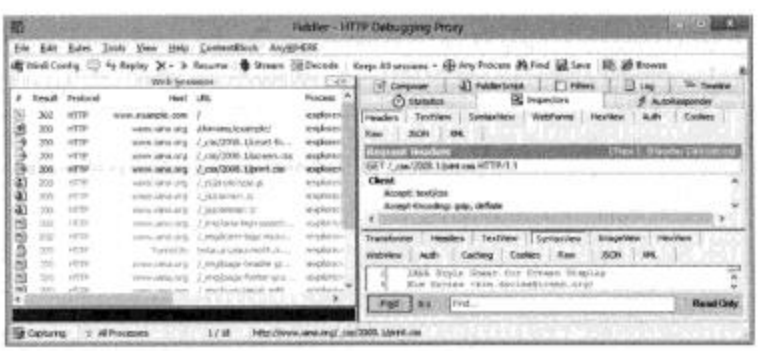

图 2-2

小贴士：

- 很多 UI 控制键包含上下文菜单。经常右击这些控制键会有很多收获。
- 如果鼠标箭头变成手势，说明可以点击该 UI 控制键。
- 支持标准的键盘快捷方式（如 CTRL+C 表示拷贝，CTRL+A 表示全部选中，CTRL+G 表示跳到特定的行号，F3 用于搜索）。
- 当有弹出的消息窗口时，按下 CTRL+C 会把文本拷贝到剪贴板。
- 按下 ESC 键可以关闭对话框或搜索框。
- 在启动 Fiddler 时，同时按下 SHIFT 键，可以把 Fiddler 的 UI 重新设置为默认形式。
- 双击选项卡标题，可以把该选项从视图中删除。
- 双击 splitter，可以最大化显示该 splitter 右侧的区域。

2.2.1　Web Sessions 列表

Web Sessions 列表是 Fiddler 中最重要的部分——它显示了 Fiddler 所捕捉到的每个 Session 的简短的摘要信息。Fiddler 中的大部分操作都是通过选中 Web Session 列表中的一个或多个项来启动的。要选中多个 Session，需要同时按住 CTRL 或 SHIFT 键，然后再点击想要选中的 Session。双击或按下 Enter 键会打开选中 Session 的默认 Inspectors。Inspectors 启动后，Fiddler 会自动选择最适合显示选中的 Session 的请求和响应的 Inspector。

Web Session 列表栏中包含了一些重要信息，具体如下：

- # – Fiddler 为 Session 生成的 ID。
- Result – 响应状态码。
- Protocol – 该 Session 使用的协议（HTTP/HTTPS/FTP）。
- Host – 接收请求的服务器的主机名和端口号。
- URL – 请求 URL 的路径、文件和查询字符串。
- Body – 响应体中包含的字节数。
- Caching – 响应头中 Expires 和 Cache-Control 字段的值。
- Content – Type – 响应的 Content-Type 头。
- Process – 数据流对应的本地 Windows 进程。

- Custom – FiddlerScript 所设置的 ui-CustomColumn 标志位的值。
- Comments –通过工具栏 Comment 按钮设置的注释信息。

可以通过拖拽来调整 Web Sessions 列表中的各个列标题，改变它们的大小和顺序；单击列标题，fiddler 会按该列的值对 Web Sessions 中的项进行排序。使用 FiddlerScript、Extensions 或 QuickExec 可以添加新字段。

理解不同图标和颜色的含义

Web Sessions 列表中的每行记录的默认文本色彩体现了 HTTP 状态（红色表示错误，黄色表示认证）、数据流类型（灰色表示 CONNECT）、响应类型（紫色表示 CSS、蓝色表示 HTML、绿色表示 script、灰色表示图像）。在 FiddlerScript Session 的 ui-color 标志位中可以修改字体颜色。

每行都有一个指向 Session 进度、请求类型或响应类型的快捷图标，图标及其含义见表 2-1。

表 2-1　　Web Sessions 图标

	正在向服务器发送请求
	正在从服务器下载响应
	请求停止于断点处，允许对它进行修改
	响应停止于断点处，允许对它进行修改
	请求使用的是 HEAD 或 OPTIONS 方法，或者返回 HTTP/204 状态码。通过 HEAD 方法和 OPTIONS 方法，客户端无需下载内容就可以获取目标 URL 或服务器的信息。HTTP/204 状态码表示没有指定 URL 的响应体
	请求使用了 POST 方法向服务器发送数据
	响应是 HTML 内容
	响应是图像文件
	响应是脚本文件
	响应是 CSS 文件
	响应是 XML 格式
	响应是 JSON 格式
	响应是音频文件
	响应是视频文件

续表

	响应是 Silverlight 小程序
	响应是 Flash 小应用程序
	响应是字体文件
	响应 Content-Type 没有专用的图标
	请求使用 CONNECT 方法。使用该方法构建传送加密的 HTTPS 数据流的通道
	Session 封装了 HTML5 WebSocket 连接
	响应是 HTTP/3xx 类重定向
	响应是 HTTP/401 或 HTTP/407，要求客户端进行认证；或响应为 HTTP/403，表示访问被拒绝
	响应包含 HTTP/4xx 或 HTTP/5xx 错误状态码
	Session 被客户端应用、Fiddler 或服务器中止。客户端的浏览器在下载一个页面的过程中，用户导航到另一个页面时，通常会发生这种情况。客户端浏览器取消所有正在处理的请求，因此最后 Session 的响应状态是 Aborted
	该响应属于 HTTP/206 响应类型。该响应通常在客户端对目标 URL 执行 Range 请求获取部分内容时发生
	响应状态是 HTTP/304，表示客户端缓存的副本已经是最新的了
	Web Session 是没有加锁的，因而可以修改已经处理完成的 Session

快捷键

Web Session 列表支持的快捷键如表 2-2 所示。

表 2-2　　Web Session 快捷键

Spacebar	在视图中激活并显示当前 Session
CTRL+A	选中所有 Session
ESC	取消选中所有 Session
CTRL+I	反向选中；取消选中的 Session，选中之前未选中的 Session
CTRL+X	删除所有 Session（和 fiddler.ui.CtrlX.KeepMarked 相关的）
Delete	删除选中的 Session

续表

Shift+Delete	删除所有未选中的 Session
R	重新执行当前请求
SHIFT+R	多次重复执行当前请求（次数在后续的提示框中给出）
U	无条件地重新执行当前请求，发送不包含 If-Modified-Since 和 If-None-Match 的请求头
SHIFT+U	无条件地多次重复执行当前请求（执行次数在后续的提示框中给出）
P	选中触发该请求的父请求。根据 HTTP 请求头的 Referer 值确定父请求
C	选中该响应触发的所有所有子请求。根据 HTTP 请求头的 Referer 字段的取值或者标示重定向的 Location 字段的取值判断
D	选中和当前 Session 使用了相同请求方法和 URL 的所有“重复”请求
ALT+Enter	查看当前 Session 的属性
SHIFT+Enter	在新的 Fiddler 窗口中启动该 Session 的 Inspectors
Backspace or “Back” mouse button	激活之前选中的 Session
Insert	切换是否用红色粗体标记选中的 Session
CTRL+1 CTRL+2 CTRL+3 CTRL+4 CTRL+5 CTRL+6	把选中的 Session 分别用粗体红色、蓝色、金色、绿色、橙色或紫色表示
M	给选中的 Session 添加描述

2.2.2 Web Session 上下文菜单

右击 Web Sessions 列表顶端的标题栏，显示包含两个选项的上下文菜单，如表 2-3 所示。

表 2-3　　Web Session 上下文菜单

所有列可见	把每个列的宽度重新设置为至少 30 个像素，确保所有列都是可见的
如何配置列...	打开一个帮助窗口，说明了如何在 Web Session 列表中添加新列

右击 Web Session 列表，会出现一个很大的上下文菜单。在该上下文菜单中，很多选项只有在选中了 Web Session 列表中一个或多个 Session 时才可见。FiddlerScript 会对该菜单进行扩展，因此它总是包含一些额外的命令。

AutoScroll Session List 选项用于决定 Fiddler 是否会自动把新增的 Session 添加到 Web Session 列表中。

Copy 菜单项用于复制在 Web Sessions 列表中选中的 Session 的信息，如表 2-4 所示。

表 2-4　　Copy 菜单项

Just Url	把选中的 Session 的 URL 列表拷贝到剪贴板，每行一个 url。当光标定位在 Web Session 列表时，快捷键 CTRL+U 可完成此功能
This column	拷贝菜单所在列的文本。把选中的 Session 的这一列的文本添加到剪贴板中，每行一个 Session
Terse summary	把选中 session 的简要说明复制到剪贴板中。简要说明中包括请求方法、URL、响应的状态码以及状态信息。如果响应是 HTTP/3xx 重定向，文本中也会包括响应头中 Location 字段的内容。当光标位于 Web Session 列表时，按下 CTRL+SHIFT+T 键也会执行该命令
Headers only	把 Session 的请求头复制到剪贴板中。既可以以纯文本格式，也可以以 HTML 格式复制，所以，在仅处理纯文本（如 notepad）的编辑器和支持富文本（如 Word）的编辑器中，复制结果会不同。由于这是“默认”命令，双击子菜单 Copy 命令会复制请求头。如果光标定位在 Web Session 列表中，按下 CTRL+SHIFT+C 键也会执行该命令
Session	把整个 Session 列表复制到剪贴板。支持以纯文本和 HTML 这两种格式进行复制，所以当粘贴到只支持纯文本的编辑器和支持富文本的编辑器中时，显示结果会有差别。当光标定位在 Web Session 列表时，还可以按下 CTRL+SHIFT+S 键来执行该命令
Full Summary	把 Web Session 列表中显示的信息复制到剪贴板。各个列通过 tab 分隔，这便于你把信息粘贴到 Excel 或其他编辑器中。当光标定位在 Web Session 列表时，按下 CTRL+C 键会执行该命令

Save 子菜单中包含了用于把流量保存到文件中的一些选项，如表 2-5 所示。

表 2-5 Save 菜单项

All Sessions	把 Web Session 列表中的所有 session 都复制到 SAZ 文件中	
...and Open as Local File	把选中 Session 的 response bodies 分别保存到独立的文件中，然后根据响应的文件类型找到对应的 handler 来打开文件。如果操作该选项时按住 CTRL 键，Windows 会提示你使用哪个应用来打开该文件	
Selected Sessions	In ArchiveZip	把 Web Session 列表中选中的 Session 保存到 SAZ 文件中
	As Text	把选中的所有 Session 一起保存到一个文本文件中
	As Text (Headers only)	把选中 Session 的请求头和响应头一起保存到一个文本文件中
Request	Entire Request	把选中 Session 的请求头和请求体分别保存到独立的文件中
	Request Body	把选中的 Session 的请求体保存到独立的文件中
Response	Entire Response	把选中 Session 的响应头和响应体保存到独立的文件中 如果需要创建的响应文件后续可以通过 AutoRe-sponder 进行重放，可以启用该选项
	Response Body	把选中 session 的请求体保存到独立的文件中。如果你希望在另一个程序单独打开响应体（如图像），而不受其 HTTP 响应头干扰，可以启用该选项

Remove 有三个子菜单可以用于从 Web Session 列表中删除全部、选中的或未选中的 Sessions。当光标定位在 Web Session 列表时，按下 CTRL+X、 Delete 或 Shift+Delete 键可以分别激活这几个命令。

Comment...菜单命令支持为一个或多个选中的 Web Session 增加或修改注释。

Mark 子菜单支持选择一种颜色来标记 Web Session 列表中的 Session。Session 的字体会根据选项相应加粗和着色。Unmark 选项会取消对选中的 Session 进行加粗，并把颜色恢复为默认颜色。

Replay 子菜单提供命令支持播放当前选中的请求，如表 2-6 所示。

表 2-6 Replay 子菜单

Reissue Requests	把选中的请求以原来的形式重新发送。如果执行该命令时也按下了 Shift 键，Fiddler 会弹出一个对话框，可以输入希望执行的次数。把光标置于 Web Session 列表中并按下 R 键就会执行该命令

续表

Reissue Unconditionally	如果需要无条件地反复发送选中的请求，增加请求头 If-Modified-Since 和 If-None-Match，可以告诉服务端不要返回 HTTP/304 响应。如果在调用该命令时也按下 SHIFT 键，Fiddler 会弹出对话框，提示输入希望该请求被执行的次数。把光标定位到 Web Session 列表中并按下 U 键也会执行该命令
Reissue and Edit	把选中的请求以原来的形式重新发送，在每个新的 Session 中设置请求断点，在请求被发送到服务器之前，使用 Fiddler 的 Inspector 修改请求
Revisit in IE	在 IE 中把每个选中的请求导航到请求 URL。注意 IE 在导航时总是使用 GET 方法，而且使用自己的 header 和 cookie，不管 session 中提供的是什么 HTTP 方法和请求头

Select 子菜单支持使用当前选中的 Session 来选择其他 Session，如表 2-7 所示。

表 2-7　　Select 子菜单

Parent Request	Parent Request 选项会尝试使用请求的 Referer 头和请求 ID 来查找当前请求的来源 Session。举个例子，假设你通过 JavaScript 执行该命令，选中的 Session 通常会是下载该 JavaScript 文件的 HTML 页面。当光标定位在 Web Session 列表时，按下 P 键可以执行该命令
Child Requests	Child Requests 选项会使用该请求的 URL 和请求 ID 来查找请求。例如，如果你在 HTML Session 中执行该命令，通过该选项选中的 Session 通常是 HTML 标记中的 CSS、JS 和图像文件的访问请求。当光标定位在 Web Session 列表时，按下 C 键也会执行该命令
Duplicate Requests	选中 Web Session 列表中和当前 Session 的 URL 和 HTTP 方法相同的所有 Session

Compare 命令只有当 Web Session 列表中仅选中两个 Session 时才可用。当选中该命令，会把选中的这两个 Session 保存到临时文件中，然后启动配置好的比较工具，从而比较请求和响应。

COMETPeek 命令会保留正在执行的响应的“快照”，在响应完成前就可以查看部分内容。当 Web 应用采用 COMET 模式以流式向客户端返回数据时，可以使用该命令。由于“流式”的含义就是永不结束，只有当服务端停止连接后，Fiddler 才会返回响应。

Abort Session 命令会终止正在执行的请求，中断客户端和服务器之间的连接。

Clone Response 命令只有当 Web Session 列表中选中两个 Session，并且其中一个 Session 在断点处中止，而另一个 Session 已经运行完时才可用。该命令会把已经完成的 Session 的响应拷贝给暂停运行的 Session。借助这个功能可以复制之前捕捉到的（或修改的）响应并返回给后续的请求。

Unlock for Editing 菜单命令会释放某个选中的 Session，支持通过 Inspector 编辑已经执行完的 Session 的请求和响应。当光标定位在 Web Session 列表，可按下 F2 键来执行该命令。

Inspect in New Window 命令会打开一个 Session Inspector 窗口，从而使你在独立窗口中查看 Session 的请求、响应和属性。

Properties...命令会打开 Session 属性窗口，显示当前选中的 Session 的信息，包括计时器、Session 标志位以及请求如何被转发的等信息。

2.3 FIDDLER 主菜单

主菜单被设计为可以启动几乎所有的 Fiddler 功能。菜单系统可以通过 FiddlerScript 或 Extensions 进行扩展和增强，本节将要探讨 Fiddler 本身默认的菜单命令。

2.3.1 File 菜单

File（文件夹）菜单中的命令主要支持完成通过 Fiddler 来启动和停止 Web 流量的捕获（capture），也可以加载或存储捕获的流量。

Capture Traffic 是个开关，可以控制是否把 Fiddler 注册为系统代理。当把 Fiddler 注册为系统代理时，所有依赖于 WinINET 代理的应用（如 IE 浏览器和很多其他浏览器）会把 Web 请求发送给 Fiddler。Firefox 的 FiddlerHook 附加组件也可以识别该菜单选项。即使 Fiddler 没有被注册为系统代理，Fiddler 仍然可以显示和处理其接收到的任何请求。

Load Archive 用于重新加载之前捕获的以 SAZ 文件格式保存的流量。

Save 子菜单中的选项支持以多种方式把流量保存到文件中；该菜单选项和 Web Session 列表中的上下文菜单中的选项相同。

Import Sessions...支持导入从其他工具捕获的流量，也支持导入以其他格式存储的流量。

Export Sessions 菜单支持把 Fiddler 捕捉到的 Session 以多种文件格式保存。在其子菜单中可以选择 All Sessions...或 Selected Sessions...。

Exit 菜单命令会取消把 Fiddler 注册为系统代理，并关闭工具栏。

2.3.2　Edit 菜单

Edit 菜单中的绝大多数命令都需要作用于 Web Session 中当前选中的 Session，因此除非选中一个或多个 Session，否则大多数命令都不可用。

Copy 下面的几个子菜单分别支持复制选中 Session 的某些信息。该菜单中的命令和 Session 列表的右键菜单中的命令相同。

Remove 下面的子菜单分别支持从 Web Session 列表中删除全部、选中的或未选中的 Session。当光标在 Session 列表中时，CTRL+X、Delete 或 Shift+Delete 键组合可以分别激活这几个命令。

Select All 命令选中 Sessions 列表的所有内容。光标置于 Session 列表中，并按下 CTRL+A 键也可以激活该命令。

Paste Files as Sessions 命令会基于剪贴板上的内容，生成一个或多个模拟的 Web Session。

如果剪贴板包含文本串，而且在前 64 个字符中含有 DataURI，可以通过标识解析出该 DataURI，创建新的 Session 并添加到 Web Session 列表中。当你通过 Fiddler 检查包含 DataURI 编码的图像的标识，并且想要看看图片看起来如何时，该功能很有用。只需要从标识中拷贝 DataURI，使用 Paste Files as Sessions 命令生成新的 Session，然后再在 ImageView 中查看该 Session 即可。

如果剪贴板中包含的是一张二进制的图片（比如，使用屏幕截图热键 Alt+PrintScrn 截取的激活窗口的一个截图），将会根据这张图片创建一个虚拟的 Session，并将其添加到 Session 列表中。

如果剪贴板包含一个或多个文件（比如是在 Windows Explorer 中复制的），每个文件可以触发生成一个虚拟的 Session 并添加到 Web Session 列表中。当你想要创建一个准备发送给别人的 SAZ 文件，并且想要添加一个没有在网络中实际传输过的文件时，就可以使用该功能。举个例子，利用 Fiddler 捕获了一个 ASPX 页面的内容后，Web 开发人员可以借助这个功能将页面的源代码以一个虚拟 Session 的形式添加到 Session 列表中，并一起以 SAZ 的格式导出。

Mark 子菜单支持选择一种颜色来标记 Web Session 列表中选中的 Session。Session 的字体是粗体的，并根据你选中的颜色进行着色。Unmark 选项会取消选中粗体，并重新设置成默认

的字体颜色。

Unlock for Editing 菜单命令会对某个选中的 Web Session 解锁，支持使用 Inspectors 编辑 Session 的所有请求和响应。你可以把光标定位在 Web Session 列表中，并按下 F2 键来激活命令。

Find Sessions...命令会打开 Find Session 窗口，搜索捕获到的数据流。你可以按下 CTRL+F 键来激活该命令。

2.3.3 Rules 菜单

Rules 菜单很容易扩展，其中的大多数命令是通过 FiddlerScript 文件生成的。

Hide Image Requests 触发器控制是否在 Web Session 列表中显示图像类 Session。

Hide CONNECTs 触发器控制是否在 Web Session 列表中显示使用 CONNECT 请求方法的 Session。客户端通过 CONNECT 方法构建到服务器的“原始”连接，发送 HTTPS 或 WebSocket 流量请求。

Automatic Breakpoints 子菜单控制 Fiddler 是否会自动在 Before Requests 或 After Responses 处断点。Ignore Image 触发器控制这些断点是否作用于图片请求。

Customize Rules...菜单命令会使用配置的脚本编辑器打开当前的 FiddlerScript 文件。

如果选中 Require Proxy Authentication 菜单项，所有未提交 Proxy-Authorization 请求头的请求会返回 HTTP/407 响应，要求客户端安装证书。该规则可以用于测试 HTTP 客户端，确保这些规则在有证书的客户端服务器上可以正常工作。

如果选中 Apply GZIP Encoding 菜单项，只要请求包含具有 gzip 标识的 Accept-Encoding 请求头，就会对除了图片以外的所有响应使用 GZIP HTTP 进行压缩。该规则用于测试使用 GZIP 选项支持的客户端是否真正对内容进行压缩。该选项还支持性能调优，并且计算传输的压缩后的数据流的字节数。

如果选中 Remove All Encodings，会删除所有请求和响应的 HTTP 内容编码和传输编码。该功能也可以通过 Fiddler 工具栏中的 Decode 按钮来调用。

该菜单中的其他命令都是用 FiddlerScript 实现的，因此菜单中的命令和默认的 FiddlerScript 命令不同。你可以通过点击 Rules 菜单中的 Customize Rules...选项来查看这些命令的实现（并添加自己的实现）。

Hide 304s 选项会隐藏包含 HTTP/304 Not Modified 状态的响应的所有 Session。

Request Japanese Content 选项会把所有请求的 Accept-Encoding 请求头设置或替换成 ja 标识，表示客户端希望响应以日语形式发送。

User-Agents 子菜单支持把所有请求的 User-Agent 请求头设置或替换成指定值。你可以在提供的值中进行选择，也可以使用菜单底部的 Custom...选项指定想要的选项值。

Performance 子菜单

Performance 子菜单提供影响 Web 性能的简单选项。

如果选中 Simulate Modem Speeds 选项，它会设置所有后续 Session 的 Flag。把 request-trickle-delay 标志位设置成 300，所有上传数据将延迟 300 毫秒/KB。同理，如果把 response-trickle-delay 标志位设置成 150，会使所有下载数据延迟 150 毫秒/KB。

如果选中 Disable Caching 选项，会删除所有 If-None-Match 和 If-Modified-Since 请求头，并添加 Pragma：no-cache 请求头。选中该选项还会删除响应中的所有 Expires 头，并把 Cache-Control 响应头设置成 no-cache。该选项无法阻止浏览器重用在选用该选项之前所缓存的响应。在选中该选项后，为了得到最佳结果，最好清空浏览器中的缓存（CTRL+SHIFT+ DELETE）。

Show Time-to-Last-Byte 会在 Web Session 列表的 Custom 列中显示 Fiddler 接收到所有响应所花费的时间，以微秒表示。

同样，Show Response Timestamp 选项会在 Web Session 列表中 Custom 列显示，Fiddler 接收到服务器的所有响应的时间戳。

Cache Always Fresh 选项会自动响应所有包含 HTTP/304 响应的有条件 HTTP 请求，表示客户端缓存是最新的。当访问的站点无法正确地设置缓存失效日期时，该选项可以极大地提高性能。尽管有该选项，但在浏览器中按下 CTRL+F5 键仍可以保证重新从服务器加载数据，因为浏览器会对要求强制更新的请求忽略 If-Modified-Since 和 If-None-Match 头。

2.3.4　Tools 菜单

Fiddler Options...打开 Fiddler Options 窗口。

WinINET Options...打开 IE 的 Internet Options 窗口。

Clear WinINET Cache 选项会清空 IE 和很多其他应用中所使用的 WinINET 缓存中的所有文件。

Clear WinINET Cookies 选项会清空 IE 和很多其他应用所发送的 WinINET cookie。Session 的 Cookies 还是保持不变。

TextWizard...选项会启动 TextWizard 窗口，支持对文本进行编码和解码。

Compare Sessions 选项只有当选中 Web Session 列表中的两个 Session 时才有效。当点击 Compare Sessions 选项，会使用内容区分工具来比较两个 Session。

HOSTS...选项会打开 Fiddler 的 Host Remapping 工具。

2.3.5 View 菜单

Squish Session List 控制 Web Session 列表是否水平收缩，以便有更大的空间来查看 Inspectors 和其他选项卡。按下 F6 键会触发该设置。

Stacked Layout 列表对 Fiddler 用户界面重新组织，使得 Web Session 列表显示在选项卡最上方，如图 2-3 所示。如果你添加很多定制列，希望有更多空间来查看这些列时，该功能就很有用。

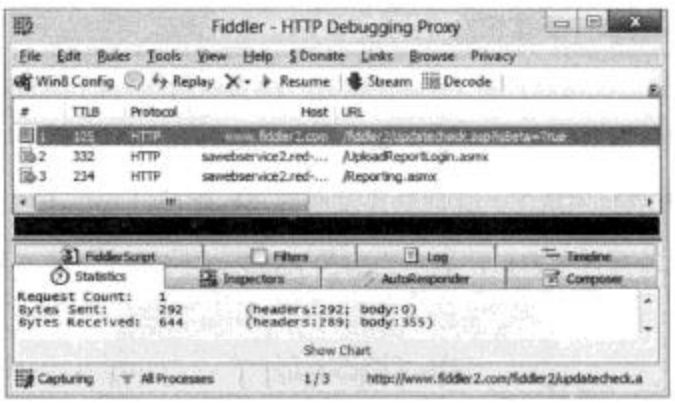

图 2-3

Show Toolbar 控制 Fiddler 工具栏是否可见，如图 2-4 所示。

Statistics 项会激活 Statistics 选项卡；按下 F7 键可以触发该命令。

图 2-4

Inspectors 项会激活 Inspectors 选项卡；按下 F8 键会触发该命令。

Composer 项会激活 Composer 选项卡；按下 F9 键会触发该命令。

Minimize to Tray 或按下 CTRL+M 键可以最小化 Fiddler 到系统托盘中。

Stay on Top 选项会强制 Fiddler 运行在所有窗口上方。

AutoScroll Session List 选项控制当添加新的 Session 时，Fiddler 是否会自动滚动到 Web Session 列表的底部。

Refresh 选项和 F5 键都用于刷新 Inspectors 或 Statistics 选项卡中当前选中的 Session 的信息。

2.3.6　Help 菜单

点击 Fiddler Help 菜单项会打开 Web 浏览器，跳转到 Fiddler 的帮助页面。F1 是这个菜单项的快捷键。

Fiddler Community Discussions 菜单项用于打开浏览器，跳转到 Fiddler 的讨论组，当前是通过 Google Groups 发起。

访问 HTTP References 项会打开包含了各种参考文档的页面，包括 RFC2616。

如果选中 Troubleshoot Filters... 选项，流量就会以显眼的字体在前端显示，否则会被隐藏起来。Comments 列中会给出实现隐藏的过滤规则。如果你发现某个流量不在 Fiddler 的 Web Session 列表中，可以尝试一下这个命令。

Check for Updates...选项会连接到 Web 服务，检查当前运行的 Fiddler 是否是最新版的。如果不是最新版，可以选择立即安装最新版，也可以选择在下一次启动时安装。

Send Feedback 选项会生成 Email 信息并发送到我的邮箱。

About Fiddler 选项会打开窗口，显示当前的 Fiddler 版本，如图 2-5 所示。

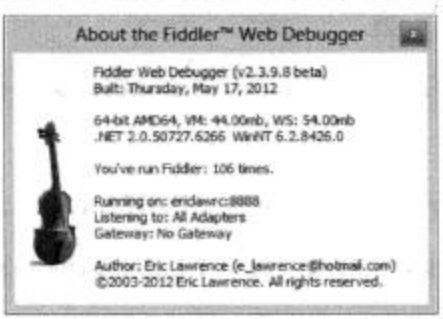

图 2-5

Fiddler 的 About Box

窗口的顶端给出了 Fiddler 的一些基本信息，包括版本号、是否是 beta 版以及何时编译的。

第二部分说明了 Fiddler 是以 32 位还是 64 位运行，以及当前使用了多少虚拟内存（Virtual Memory）和工作集（Working Set）。同时，给出了 Microsoft .NET 的版本以及 Windows 操作系统的版本信息。

第三部分给出了 Fiddler 被启动的次数。

第四部分中介绍了 Fiddler 当前使用的主机名和端口号。Listening to 表示 Fiddler 注册的网络连接。Gateway 表示上游代理服务器的信息。

最后一部分给出的是联系信息和版权信息。

按下 Esc 键或空格键都可以关闭窗口。按下 CTRL+C 键可以拷贝全部文本；用鼠标选中期望的文本子集，再按下 CTRL+C 键可以仅拷贝选中的部分。

2.4 FIDDLER 的工具栏

Fiddler 工具栏提供了常见命令和设置的快捷方式，如图 2-6 所示。

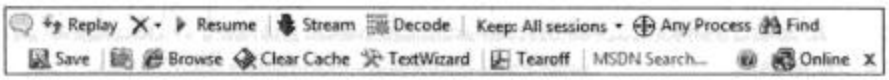

图 2-6

下面介绍 Fiddler 的按钮和功能，见表 2-8。

表 2-8 Fiddler 按钮

Comment	点击该按钮可以为所有选定的 Session 添加 Comment。Comment 是 Web Session 列表中的一列
Replay	点击该按钮可以向服务器重新发送选中的请求。点击该按钮并同时按下 CTRL 键会重新发送请求，而不包含任何条件请求（Conditional Request）头（如 If-Modified-Since 和 If-None-Match）。点击该按钮的同时按下 SHIFT 键会弹出提示对话框，要求指定每个请求应该被重新发送的次数
Remove	显示从 Web Session 列表中删除 Session 的选项菜单： • Remove all 用于删除列表中的所有 Session • Images 用于删除图像类 Session • CONNECTs 用于删除所有 CONNECT 通道 • Non-200s 用于删除所有 HTTP 响应码不是 200 的 Session • Non-Browser 用于删除不是由 Web 浏览器发出的所有请求 • Complete and Unmarked 用于删除所有状态为正常结束（Done）或异常终止（Aborted）以及没有被标记且 Comment 列没有内容的 Session • Duplicate response bodies 用于删除没有响应体或者响应体的内容在更早的 Session 中已经被接收到的 Session

续表

Resume	恢复执行在 Request 或 Response 断点处暂停的所有 Session
Stream	打开 Stream 开关，取消所有没有设置中断的响应的缓存
Decode	打开 Decode，会对请求和响应中的所有 HTTP 内容和传输编码进行解码
Keep: *value*	Keep 下拉选项框用于选择在 Web Session 列表中保存多少 Session。达到 Keep 选项中的指定计数值后，Fiddler 会删掉老的 Session，把列表中的 session 数限定为设置的期望值。未完成的 Session 以及包含注释、标记或 Inspector 窗口中处于打开状态的 Session 不会被删除
Process Filter	在应用中拖拽 Process Filter 图标会创建一个过滤器，它会隐藏选中进程的以外的所有流量。右击 Process Filter 图标会清除之前设置的过滤器
Find	打开 Find Sessions 窗口
Save	把所有的 Session 保存到 SAZ 文件中
Camera	把当前桌面的屏幕截图以 JPEG 格式添加到 Web Session 列表中
Browse	如果选中了一个 Session，会在 IE 中打开目标 URL。如果没有选中任何 Session 或选中了多个 Session，在 IE 中打开 about:blank
Clear Cache	清除 WinINET 缓存。按下 CTRL 键并点击该按钮还会清除 WinINET 中保存的永久 cookie
TextWizard	打开文本编码/解码小工具，以使文本在多种编码间转换
Tearoff	新建一个包含了所有 View 的新窗口，同时 Web Session 列表扩展为充满整个 Fiddler 主窗口
MSDN Search	在 MSDN 的 Web Content 区域中进行搜索
Help	打开 Fiddler 的帮助窗口
Online Indicator	指示系统当前是在线的还是离线的。如果在线，把光标停留在该按钮上方会显示包含本地计算机的主机名和 IP 地址的提示。双击该按钮会打开系统的 Network Connections 控制面板
X	删除工具栏。如果要恢复该工具栏，可以点击 View > Show Toolbar

把光标悬停在工具栏的任何元素上，都会显示一条提示信息，简要说明元素的功能。按

下 ALT 键可以将工具栏元素拖动到新位置，从而重新组织工具栏。但是，这些变化无法保存，而且重新启动 Fiddler 时需要重新设置。

如果你在小显示器中使用 Fiddler，缩短的工具栏可能会导致无法显示最右方下拉菜单中的某些命令。将 fiddler.ui.toolbar.ShowLabels 设置为 false，工具栏就会不再显示工具的名称，这样可以减小工具栏的宽度。

Fiddler 的状态栏

Fiddler 主窗口最下方的是状态栏，由一组面板组成，显示了 Fiddler 的一些配置信息。点击其中一些面板可以快速更改配置。面板从左到右如表 2-9 所示。

表 2-9　　Fiddler 状态栏

Capturing Indication	指示 Fiddler 是否被配置为系统代理。点击面板可以切换状态
Process-based Filter	显示 Fiddler 当前正在捕获的流量的进程类型。点击面板可以显示进程类型的过滤选项菜单
Breakpoint Indicator	指示中断影响的类型，可能的取值有：全部请求、全部响应、无。点击面板可以在这几种类型间快速切换
Session Counter	Web Session 列表的条目数。选中一个或多个 Session，显示的是选中的 Session 数以及总 Session 数，如“2 / 5”
Status Information	默认情况下，显示选中的第一个 Session 的 URL。该面板还可以显示操作结果的概要信息，比如何时加载或保存 SAZ 文件

2.5 QUICKEXEC

Web Sessions 列表下的 QuickExec 对话框中提供了常见操作的快捷方式。Fiddler 处于活动状态时，使用 Alt+Q 快捷键可以把光标定位到 QuickExec 对话框；如果 Fiddler 没有处于活动状态，则需要先使用 CTRL+ALT+F 键激活 Fiddler 窗口。

当光标定位在 QuickExec，按下 CTRL+I 键会把选中的第一个 Session 的 URL 插入到 Web Session 列表中。你还可以从 Web Sessions 列表中拖动/释放一个或多个 Session，把 URL 插入到 QuickExec 对话框；你也可以从文件系统中将一个或多个文件拖入到文件路径中。

QuickExec 选择命令

QuickExec 支持基于指定的搜索条件快速选择感兴趣的数据流。键入选择命令后，如果搜索结果非空，按下 Enter 键后，光标会定位到 Web Session 列表中对应的 Session，见表 2-10。

表 2-10　　QuickExec 选中命令

命　令	动　作	实　例
?search	选择 URL 中包含了指定文本的 Session。 这是 QuickExec 对话框中唯一具有即写即搜（*find-as-you-type*）特性的搜索功能。对于前缀为?的搜索，按下回车键会把光标定位到 Web Session 列表中的结果上。 而所有其他搜索，按下 Enter 键只是启动搜索	? example.com/pathchars
select *type*	选择响应头的 Content-Type 中包含有选中 *type* 的 Session	select css s elect image/jp
select *header-or-flag value*	选中在指定的 *Header or SessionFlag 的取值中*包含指定 *value* 字符串的 Session，其中值匹配时大小写不敏感。 星号前面不带有反斜线时，是一个通配符，表示任何值。*表示匹配星号本身	select ui-comments slow select ui-bold * select ui-backcolor red select ui-comments * select @Request.Accept html select @Response.Set-Cookie domain
>size	选中响应大小超出给定字节的 Session。 注意：字符“k”会被转换成“000”，这样就可以很容易地以 KB 甚至 MB 为单位进行设置	>40000000 >4000k >4KK
<size	选择响应大小小于指定 *size* 字节的 Session。 注意：字符“k”代表“000”	<5k
@host	选中请求头的 Host 中包含了指定 *host* 的 Session	@example.com @.gov
=ResponseCode	选中响应状态码等于给定值的 Session	=200 =404
=Method	选中请求的 HTTP 方法是给定值的 Session	=GET =POST

除了支持数据流的选择，QuickExec 对话框还支持一些其他命令：

cols add *flagname* cols add *title flagname*	给 Web Session 列表添加新的字段（只属于该实例）。参数 title 是可选的；如果该字段不存在，也可以用 flagname 命名。flagname 既可以是 Session Flag，也可以是请求头或响应头的字段名称。 添加一个字段后，后续的所有 Session 都会包含这个字段。要更新已经完成的 Session，需选中这些 Session 并按下 F5 键	cols add x-clientIP cols add Server @Response.Server cols add Accept @Request.Accept
!listen *port* [*SubjectCN*]	在指定端口启动一个新的代理监听器对象。这个监听器的 Session 会被添加到 Web Session 列表中并被自动配置为允许远程连接。 如果有 CN 参数，该监听器上的所有接入连接会自动触发 HTTPS 握手；Fiddler 会提供证书，包含指定的 *SubjectCN*。 当把 Fiddler 作为 HTTPS 网站的反向代理时，该功能很有用	!listen 8889 !listen 4443 secure.example.com
!dns *hostname* !nslookup *hostname*	为指定的 *hostname* 执行 DNS 查询（或者在 Fiddler 的 DNS 缓存中查询），在 Log 选项卡中显示结果	!dns www.example.com
prefs show [*substring*]	将所有的 Fiddler 配置选项显示在消息对话框中。 如果提供子字符串 *substring*，只显示包含了该子串的设置	prefs show prefs show composer
prefs remove *name*	清除包含了特定名称 *name* 的选项。下一次查询时，该选项会使用默认值	prefs remove fiddler.ui.font.size
prefs set *name value*	更新或创建包含指定名称 *name* 和值 *value* 的选项	prefs set fiddler.differ.UltraDiff False
prefs log	把 Fiddler 配置的所有选项写到 Log 选项卡，以便拷贝	
!spew	触发“DebugSpew”模式，它主要用于调试 Fiddler 本身的问题。在 DebugSpew 模式下，Fiddler 会将详细的日志信息（包括所有原始的数据流）发送到系统的调试端口。可以使用 SysInternals 的 DebugView 工具获取这些信息。在这种模式下，Fiddler 也会在 Log 窗口中记录一些额外的信息	
about:cache	在 Log 选项卡中显示 Fiddler 的监听端口以及连接复用缓存和 DNS 缓存的内容	

续表

about:config	显示 Fiddler 的选项配置窗口，它列出所有的选项及其值
about:connectoids about:network	在 Log 窗口中显示基于 WinINET 的所有连接的信息
toolbar	按下该按钮，如果 Fiddler 工具栏之前是隐藏的，会重新显示
tearoff	将 Inspectors 从 Fiddler 的主窗口中删除，转而在浮动窗口中显示

如果在 QuickExec 对话框中输入的命令不在上述的内置命令列表中，则它可能是需要使用 FiddlerScript 和 Fiddler 扩展进行响应的命令。

默认的 FiddlerScript 命令

Fiddler 默认的 FiddlerScript 文件中包含了很多实用的命令，这些命令可以在 QuickExec 对话框中启动，见表 2-11。

表 2-11　　FiddlerScript 命令

命　令	动　作	实　例
bold *urltext*	加粗显示 URL 中包含了指定文本的后续 Session。键入不带参数的 bold，会取消加粗显示	bold uploaddata.asp
bps *status*	为响应码是指定值的 Session 创建响应断点。键入不带参数的 bsp 时，取消断点	bps 404
bpm *method*	为 HTTP 方法匹配给定值的 Session 创建请求断点。键入不带参数的 bpm 时，取消断点	bpm POST bpm OPTIONS
bpu *urltext*	为 URL 包含指定文本的 Session 创建请求断点。键入不带参数的 bpu 时，取消断点	bpu uploaddata.asp
bpafter *urltext*	为 URL 包含指定文本的 Session 创建响应断点。键入不带参数的 bpafter 时，取消该断点	bpafter dodownload.cgi
nuke	清空 WinINET 缓存和 cookie	nuke
tail ####	截断 Web Session 列表，使得它包含的 Session 数不大于指定数目	tail 200
log string	向 Log 选项发送指定的文本或宏	log "At this point, requests start failing" log @Log.Save

续表

命　令	动　作	实　例
cls	清空 Web Session 列表	cls
dump	保存所有捕获的流量到\Captures\文件夹中的 dump.saz 文件中	dump
g	当前在断点处暂停的所有 Session 恢复执行	g
help	显示 QuickExec 的在线帮助	help
urlreplace *oldtext newtext*	对后续请求的 URL 进行字符串替换操作。键入不带参数的 urlreplace 时，可以终止替换	urlreplace jQuery.min jQuery.dbg
overridehost *oldhost newhost*	请求重定向，将请求发送到不同的主机。键入不带参数的 overridehost 时，停止重定向	overridehost production.example.com dev.internal.example.com
start	激活捕获模式，把 Fiddler 注册为系统代理	start
stop	关闭捕获模式，Fiddler 不再作为系统代理	stop
keeponly *MIMEtype*	删除 Web Session 列表中响应不带有给定 MIMEtype 的所有 Session	keeponly video/
quit	退出 Fiddler。该命令会退出以激活自动执行工具 ExecAction.exe，请求 Fiddler 清除自己的临时文件然后退出	quit

编辑 FiddlerScript 可以查看 FiddlerScript 命令的整个列表，并添加自己的命令，只需要点击 Rules>Customize Rules 选项，把滚动条拉到 OnExecAction 方法。

扩展还可以给 QuickExec 选项框添加其他命令——可以查看所安装的扩展的文档来获取更多信息。

2.6 应用热键

QuickExec 提供了强大的键盘支持，不过 Fiddler 还支持其他热键。

Fiddler 注册了系统级别的热键 CTRL+ALT+F，无论当前活动的应用是什么，都可以激活 Fiddler。在选项 Tools > Fiddler Options > General 中可以修改这个热键。

除了这个全局热键，启动 Fiddler 后，还可以应用很多其他热键，具体如表 2-12 所示。

表 2-12　　应用热键

ALT+Q	把光标定位在 QuickExec 对话框
CTRL+R	打开 FiddlerScript 规则编辑器
CTRL+E	打开 TextWizard
CTRL+Down	选中 Web Session 列表中的下一个 Session
CTRL+Up	选中 Web Session 列表中的上一个 Session
CTRL+T	激活 TextView Inspectors
CTRL+H	激活 HeaderView Inspectors
CTRL+0	把字体大小设置为 8.25pt（默认值）
CTRL+Plus	字体大小增加 1pt（最多到 32pt）
CTRL+Minus	字体大小减少 1pt（最少到 7pt）
CTRL+M	最小化 Fiddler
CTRL+SHIFT+DEL	清除 WinINET 缓存
F12	开关，把 Fiddler 注册为系统代理或者从系统代理处注销

2.7　统计选项卡

Fiddler 的统计选项卡中显示了当前 Session 的基本信息。在选项卡的最上方显示的是文本信息，最下方是个饼图，按 MIME 类型显示流量。

当选中多个 Session 时，显示的数据如表 2-13 所示。

表 2-13　　统计选项卡

Request Count	选中的 Session 数
Unique Hosts	流量流向的独立目标主机数。如果所有选中的流量都发送到相同的服务器上，就不会显示该字段
Bytes sent	HTTP 请求头和请求体中向外发送的字节总数。在本行后面的括号中分别给出了头和 body 各自的字节数
Bytes received	HTTP 请求头和请求体中接收到的所有字节数。在全部计数后面的括号中给出了请求头和请求体各自的字节数
Requests started at	Fiddler 接收到的第一个请求的第一个字节的时间点
Responses completed at	Fiddler 发送到客户端的最后一个响应的最后一个字节的时间点

续表

Sequence (clock) duration	第一个请求开始到最后一个响应结束之间的“时钟时间”
Aggregate session duration	所有选中的 Session 从请求到响应之间的时间的和。因为 Session 通常是并行运行的，这个结果可能比“时钟时间”长。同理，因为这些选中的 Session 从开始到结束时间之间也可能包含空闲时间，该时间总和也可能比第一次请求和最后一次请求之间的“时钟时间”短得多
DNS Lookup time	所有选中 Session 解析 DNS 所花费的时间的总和。如果所有的请求都命中了 DNS 缓存或者所有的连接都已经建立，就不需要 DNS 解析，因而不会显示该字段
TCP/IP Connect duration	所有选中 Session 建立 TCP/IP 连接所花费的时间总和。如果所有请求都是在已经建立的连接上，就不需要进行 TCP/IP 连接，因而不会显示该字段
HTTPS Handshake duration	所有选中 Session 在 HTTPS 握手上所花费的时间总和。如果所有的 Session 都是 HTTP，不需要解密，或者所有的请求都发生在已经建立的安全信任的连接上，则不存在 HTTPS 握手，因而不会显示该字段
Response Codes	选中 Session 中各个 HTTP 响应码的计数
Response Bytes by Content-Type	选中 Session 中响应的各个 Content-Type 的字节数。在选项卡的最下方也用饼图显示了这个信息
Estimated Performance	选中的流量在不同语种（locale）地区和连接方式下所需时间的初步估计。该值是根据选中 Session 的个数和大小计算的。实际网络性能会受到很多因素的影响，因而该估值可能很不准确

如果只选中了一个 Session，会显示该 Session 的计时器。如果选中的 Session 是在前一天开始记录的，则会给出日期——这对于分析自动化日志工具捕捉到的 SAZ 文件很有帮助。

如果选中的是单个 CONNECT 通道，会显示在该通道上发送和接收的字节数（除非该通道配置为 HTTPS 加密方式，在这种情况下，需要在 Web Session 列表中已解密的 HTTPS Session 中查看字节数信息）。

选项卡的最下方是个饼图，默认情况下不显示该饼图。点击 Show Chart 链接会显示选中 Session 的饼图。会出现一张关于选中数据流的 headers 以及 MIME 类型的饼图；饼图的切片是各个 MIME 类型以及 headers 的字节数。

点击左下方的“Copy this chart”链接会把饼图作为位图拷贝到剪贴板，进而粘贴到报表

或演示中，如图 2-7 所示。

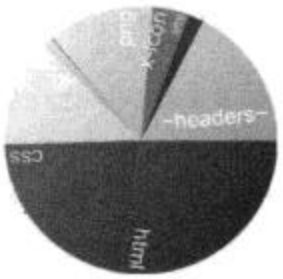

图 2-7

2.8　FILTERS 选项卡

Filters 选项卡提供了一种“即指即点”的方式，可以很方便地将简单的过滤规则应用到正在捕捉的数据流上。FiddlerScript 可以模拟选项卡上的所有功能（通常更准确或强大），但是对于简单的任务，Filters 选项一般就足够满足过滤需求。

选中 Filters 选项卡左上方的 Use Filters 复选框后，就可以使用其中随后给出的过滤器对流量进行过滤了。选中 Use Filters 复选框后，对于选定的 Session，其下方的选项依次控制：

- 是否隐藏显示
- 是否在 Web Session 列表中添加标识
- 是否设置断点用于人工调试
- 是否阻断发送
- 是否自动修改其数据头

Fiddler 还会为隐藏的 Session 提供代理功能，即使在 Web Session 列表中没有显示这些 Session。

选项卡的右上方的 Actions 按钮支持把当前选中的过滤器作为过滤集，加载之前保存的过滤集，并对之前捕捉到的数据流应用当前过滤规则。Help 选项会弹出关于过滤选项卡的帮助信息如图 2-8 所示。

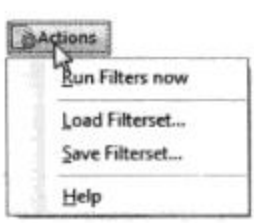

图 2-8

每组过滤选项的介绍如下。

Hosts 主机

Hosts 框提供根据主机名过滤的功能。

Zone Filter 下拉框支持只显示内网（如不带“.”的主机名）或互联网（如带“.”的主机名）的数据流，如图 2-9 所示。当调试某个区域的网站引用其他区域的 Web 文档时，这个选项很有用。

图 2-9

Host Filter 下拉框支持标识或隐藏在随后的文本框中指定的域名下的数据流，如图 2-10 所示。

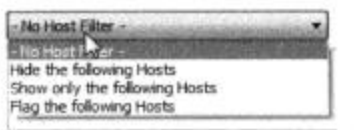

图 2-10

注意，文本框不会自动通过通配符匹配子域名。这意味着如果你设置了“只显示如下 Hosts”并且在列表中只有 fiddler2.com，那么将无法看到 www.fiddler2.com 网站下的数据流。为了查看 fiddler2.com 域名下所有的数据流，需要添加通配符*：*.fiddler2.com。在通配符方式下就可以包含如 test.fiddler2.com 和 sub.fiddler2.com 这样的网站。如果你想查看根目录 fiddler2.com 下的数据流，可以把通配符改成*fiddler2.com 的形式——这样就可以包含所有域名以 fiddler2.com 结束的数据流，前面不需要加点。如果有多个 host，可以使用分号分隔。注意：如果文本框的背景是黄色的，说明修改还没有生效。点击文本框外面的任何地方，修改会生效到列表中。

客户端进程

进程过滤器控制显示哪个进程的数据流。正在运行的应用进程和 Fiddler 在相同的主机时，Fiddler 才能判断出是哪个进程发出的哪个请求。

下拉框 Show only traffic from 的列表中包含了系统中当前正在运行的所有进程；如果选中某个进程，就会只显示该进程下的数据流。

Show only Internet Explorer 选项只显示进程名称以 IE 开头或请求的 User-Agent 头包含 compatible; MSIE 的数据流。

Hide traffic from Service Host 选项会隐藏来自进程 svchost.exe 的数据流，svchost.exe 进程是个系统进程，会同步 RSS Feeds 以及执行其他后台网络活动。

请求头 Request Headers

通过这些选项，你可以添加或删除 HTTP 请求头，也可以标识包含某些请求头的请求。

Show only if url contains 选项框支持基于 URL 隐藏某些请求。可以使用前缀 EXACT 来限定大小写敏感，如下：

```
EXACT:example.com/q=Case+Sensitive+String
```

也可以使用正则表达式：

```
REGEX:(?insx).*\.(gif|png|jpg)$ #only show image requests
```

Flag requests with header 选项支持指定某个 HTTP 请求头名称，如果在 Web Session 列表中存在该请求头，会加粗显示该 Session。

Delete request header 选项支持指定某个 HTTP 请求头名称，如果包含该请求头，会删除该请求头。

Set request header 选项支持创建一个指定了名称和取值的 HTTP 请求头，或将 HTTP 请求头更新为指定取值。

断点 Breakpoints

断点选项框支持对包含给定属性的请求或响应设置断点。

Break request on POST 选项会为所有 POST 请求设置断点。

Break request on GET with QueryString 选项会为所有方法为 GET 且 URL 中包含了给定查询串的请求设置断点。

Break on XMLHttpRequest 选项会对所有能够确定是通过 XMLHttpRequest 对象发送的请求设置断点。由于从数据流上无法判断该请求是否通过 XMLHttpRequest 对象发送，因此该功能是通过查找请求头是否为 X-Requested-With（由 jQuery 框架添加）实现的。它还会检查请求头是否为 X-Download-Initiator，在 IE10 及更高的版本中可以配置在请求头中包含 X-Download-Initiator。

Break response on Content-Type 选项会为所有响应头 Content-Type 中包含了指定文本的响应设置响应断点。

响应状态码 Response Status Code

通过这些选项，你可以基于响应状态码过滤 Session。

Hide success 选项会隐藏状态码在 200 到 299 之间（包含 200 和 299）的响应。这些状态码用来表示请求成功。Hide non-2xx 选项会隐藏状态码不在 200 到 299 之间的响应。

Hide Authentication demands 选项会隐藏状态码为 401 和 407 的响应，这些响应需要用户进一步确认证书。

Hide redirects 选项会隐藏对请求进行重定向的响应。

Hide Not Modified 选项会隐藏条件状态请求中状态码为 304 的响应，表示客户端缓存的实体是有效的。

响应类型和大小

通过这些选项，你可以控制在 Web Sessions 列表中显示哪些类型的响应，并阻塞符合某些条件的响应。

Type dropdown 列表支持隐藏响应不是指定类型的 Session。

- Show all Content-Types：不过滤。
- Show only IMAGE/*：隐藏 Content-Type 头不是图像类型的 Session。
- Show only HTML：隐藏 Content-Type 头不是 HTML 类型的 Session。
- Show only TEXT/CSS：隐藏 Content-Type 头不是 text/css 的 Session。
- Show only SCRIPTS：隐藏 Content-Type 头不是脚本类型的 Session。
- Hide IMAGE/*：隐藏 Content-Type 头是图像类型的 Session。

Hide smaller than 选项隐藏响应体小于指定的字节数的响应。Hide larger than 选项隐藏响应体大于指定字节数的响应。

Time HeatMap 复选框会基于服务器返回给定响应所需要的时间为每个 Session 设置背景颜色（通过 Timers 对象计算：ServerDoneResponse - FiddlerBeginRequest）。不超过 50 毫秒的响应会以绿色显示；50 毫秒到 300 毫秒之间的响应不着色；300 毫秒到 500 毫秒之间的响应以黄色显示；超出 500 毫秒的响应以红色显示。

如果返回的响应头指定 Content-Type 为脚本，Block script files 选项就会返回 HTTP/404 响应。如果返回的响应头指定 Content-Type 为图像，Block image files 选项就会返回 404 响应。如果返回的响应头指定 Content-Type 为 Adobe Flash (application/x-shockwave-flash)，Block SWF 选项就会返回 404 响应。

如果返回的响应头指定 Content-Type 为 CSS，Block CSS files 选项会返回 404。

响应头 Response Headers

通过这些选项，你可以添加或删除 HTTP 响应头，或者为包含特定响应头的响应添加标识。

Flag responses that set cookies 选项会以粗体显示所有响应头包含 Set-Cookie 的响应。

Flag responses with header 选项支持指定 HTTP 响应头名称，如果响应中该 HTTP 头存在，该 Session 在 Web Session 列表中会以粗体显示。

Delete response header 选项支持指定 HTTP 响应头名称，如果存在该响应头名称，会从响应头中删去。

Set response header 选项支持创建或更新 HTTP 响应头，你可以自己设置取值。

2.9　TIMELINE 时间轴选项卡

该选项卡支持使用“瀑布”模型查看 1 到 250 个选中的 Session，这对于性能分析和理解请求之间的关联是很有用的。选项卡的主体内容就是数据流视图。页面的上方是标题，显示时间轴模式（默认情况下是“传输时间轴”（Transfer Timeline））。点击右上方的帮助链接，会打开关于该功能的帮助对话框。

在选项卡中任意位置右击，可以看到上下文菜单，其中包括表 2-14 所示的选项。

表 2-14　　上下文菜单项

AutoScale Chart	如果选中该选项，会水平调节图形宽度，使得整个图形适配选项卡宽度，不需要水平滚动条
Copy Chart	点击该选项会把图形以位图格式拷贝到剪贴板，以便拷贝到其他文档中
Mode (dropdown)	Mode 下拉条控制图形如何显示： • Timeline：通过时间轴线条显示每个 Session，用彩色条表示时间段 • Client Pipe Map：显示每个时间轴的客户端进程和 Fiddler 之间的连接。多个 Session 之间重用的连接会以多种彩色条显示 • Server Pipe Map：显示每个时间轴的 Fiddler 和上游服务器之间的连接。多个 Session 之间重用的连接会以多种彩色条显示 Client Pipe Map 和 Server Pipe Map 模式显示客户端和服务器之间如何复用连接，对判断性能瓶颈有帮助

时间轴选项卡并不显示任何 CONNECT 通道，因为该通道的数据流可能是模糊的，也可能是由一个或多个解密的 HTTPS Web Session 项跟踪。

2.9.1 模式：Timeline（时间轴）

在时间轴模式下，每行显示一个 Session，最左边的是从 URL 中抽取出来的文件名。鼠标放在任意一个条形栏的上方时，下面的状态栏中就会显示该项的更多信息。双击该项会显示这个 Session 的信息；在双击的同时按下 SHIFT 键会在新的窗口显示该 Session 的信息。

条形栏的颜色是由响应的 MIME 类型决定的：浅绿色表示图像，深绿色表示 JavaScript，紫色表示 CSS，蓝色表示其他类型，如图 2-11 所示。

图 2-11

传输条形栏（transfer bar）在客户端开始向 Fiddler 发送请求（Timers.ClientBeginRequest）时开始绘制。当发送回客户端的响应（Timers.ClientDoneResponse）收到后完成绘制。如果条形栏是用阴影线表示，而不是平滑的，表示 HTTP 响应在 Fiddler 端被缓存起来了，而不是流式地转发给客户端。缓存会改变瀑布模型，如图 2-12 所示，在该 Session 的页面完成之前，不会开始下载图片。

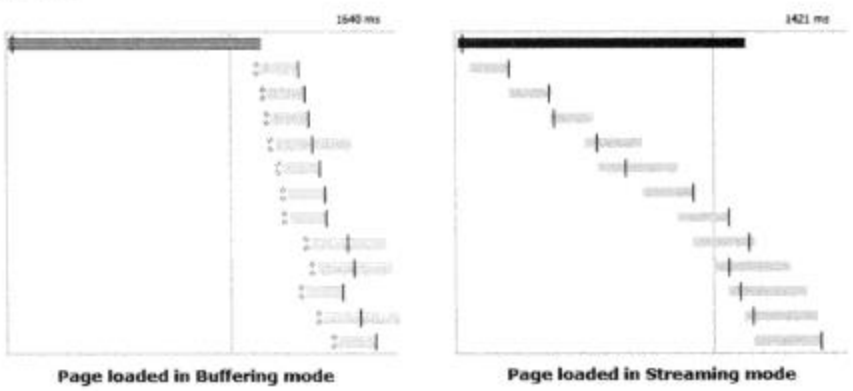

图 2-12

条形栏中黑色的垂直线表示接收到服务端响应（Timers.ServerBeginResponse）的第一个

字节的时间。条形栏前面的两个小圆圈表示 Session 是否是复用连接进行传输。绿色圆圈表示连接是复用的，而红色圆圈表示连接是新创建的。上方的圆圈表示客户端到 Fiddler 之间的连接；下方的圆圈表示 Fiddler 到服务器的连接。

条形栏后面的红色 X 图标表示服务器发送了请求头 Connection: close（对于 HTTP/1.0 类型的响应，是指发送请求头 Connection: Keep-Alive 失败），阻止后续请求重用该连接。灰色箭头图标表示服务器响应是重定向的（302）。红色！图标表示服务器返回了错误码（4xx 或 5xx）。

2.9.2　模式：Client Pipe Map（客户端管道映射）

在 Client Pipe Map 模式下，时间轴显示一个客户端的接入连接请求。图表的左侧是连接标识符，显示了进程名称、进程 ID 以及客户端端口号。例如，iexplore:1364(p14421)表示客户端是 IE，进程编号是 1364，使用端口 14421 连接 Fiddler。多个 Session 之间重用的连接也会显示为多个条形栏，如图 2-13 所示。

图 2-13

2.9.3　模式：Server Pipe Map（服务端管道映射）

在 Server Pipe Map 模式下，时间轴表示 Fiddler 向服务器发出的连接，每行一个。在图表的左侧是连接标识符，表示 Fiddler 端的端口号和目标主机名。例如，p14357->twimgs.com 表示 Fiddler 使用端口 14357 创建了到 twimgs.com 的 80 端口的连接。多个 Session 之间重用的连接会显示为多个，如图 2-14 所示。

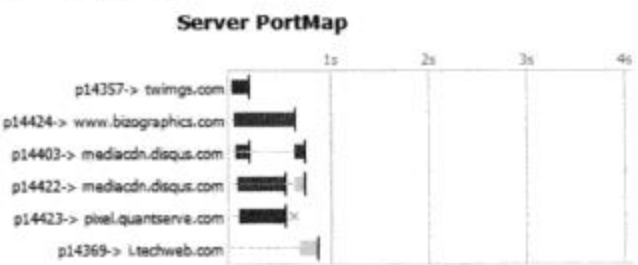

图 2-14

2.9.4 使用时间轴进行性能分析

时间轴提供了让应用使用网络的富信息视图表示。从图上可以很容易地识别出响应慢的请求（条形栏更长），请求由于连接限制带来的瓶颈（每组 6 个请求），以及连接不必关掉的情况（红色 X 图标）。通过该信息，你可以调整应用，更好地对请求进行排序，从而提高网络性能。

2.10 自动响应（AUTORESPONDER）选项卡

自动响应选项卡提供 Fiddler 最强大的一些功能。它支持创建规则，可以在响应请求时自动触发，常见例子是返回之前捕捉的响应，而不需要访问服务器。

在自动响应选项卡区域的上方是一组选项，能够控制 AutoResponder 的行为，而该选项卡的绝大部分区域是匹配接收到的 URL 请求的 Match Conditions 和 Actions。

Enable automatic responses 复选框控制是否激活 AutoResponder 选项卡。如果没有选中该选项，该选项卡上的其他选项就不可选。

Unmatched requests passthrough 选项控制当 Session 不匹配任何给定的规则时会发生什么。如果选中该选项，不匹配的请求会正常发送到服务器，就像 AutoResponsder 功能不存在。如果没有选中该选项，Fiddler 会为所有和该规则完全不匹配的 HTTP 请求生成 HTTP/404 Not Found 响应。如果客户端发送的是条件请求，其中包含请求头 If-None-Match 或 If-Modified-Since，那么 AutoResponder 会返回响应 HTTP/304 Not Modified。

Enable Latency 选项控制匹配某个规则的请求是立即执行，还是延迟 Latency 字段中所指定的毫秒数。如果没有选中该选项，Latency 字段就不会显示。使用这个选项可以更准确地模拟现实中的服务器响应，取消该选项可以提升性能。

Import 按钮支持导入之前捕获的 SAZ 文件；导入文件中的每个 Session 会被用于规则列表中生成新的规则。你还可以导入 FARX 文件，它包含从 AutoResponder 选项卡导出的规则。

选项卡的中心是个规则列表。第一列是匹配条件（Match Condition），该字段的内容用于判断接收到的请求是否匹配该规则。Match Condition 前面的复选框控制规则是否选中。第二列说明了规则匹配时所需要采取的操作。Action Text 可以指定要返回的本地文件名，也可以指定其他行为类型。

当选中某个规则时，在下方的 Rule Editor 选项框中可以调整规则的 Match Condition 和

Action 文本。Test 链接支持根据选中的样本 URI 测试 Match Condition；如果你善于使用正则表达式，该功能也会很有用，如图 2-15 所示。

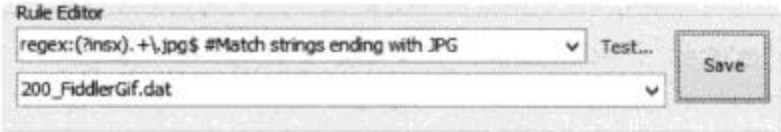

图 2-15

在对 Match Condition 或 Action Text 做出任何改变后，点击 Save 按钮能够更新规则。如果你选中了多个规则，Rule Editor 会隐藏 Match Condition 选项框，并且可以立即更新所有选中规则的 Action Text 选项，如图 2-16 所示。该功能支持多个规则使用相同的响应。

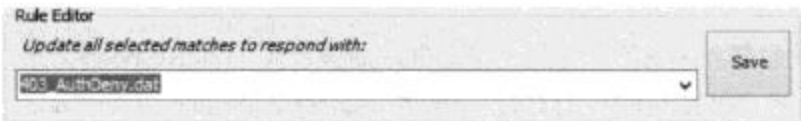

图 2-16

2.10.1　指定匹配条件

默认情况下，Match Condition 是对 URL 执行大小写不敏感的匹配操作。因此，如果你希望定制一个规则，匹配 URL 中不包含单词 fuzzle 的请求，只需要在文本框中键入 NOT:fuzzle。只要请求的 URL 中不包含文本 fuzzle（大小写不敏感），请求就匹配以下任意规则：

```
http://fuzzle.bunnies.com
http://bunnies.fuzzle.net
http://example.com/Fuzzle/
http://example.com/search?q=FuZzLe
```

匹配条件*可以匹配所有接收到的请求。

指定前缀 NOT:，该规则会匹配 URL 不包含给定字符串的所有请求，匹配时大小写不敏感。因此，如果希望定制一个规则，表示 URL 不包含单词 fuzzle 的所有请求，只需要在选项框中输入 NOT:fuzzle。除非在请求的 URL 中找到文本 fuzzle（大小写不敏感），该规则就可以匹配。

指定前缀 EXACT:，可以要求对目标 URL 大小写敏感。因此，规则 EXACT:http://example.com/a/ 可以匹配：

```
http://example.com/a/
```

...但是它不匹配以下 URL：

```
https://example.com/a/
http://example.com/b/
http://example.com/a/otherstuff
```

指定 REGEX:前缀，可以对请求的 URL 执行正则匹配。表达式是通过.NET 正则表达式引擎来匹配的。

已经有很多书探讨如何编写强大的正则表达式；本书不再赘述。建议参考 http://fiddler2.com/r/?RegExHelp 入门。常见的一些正则表达式规则如下。

表 2-15

规则	匹配项
REGEX:.+	任何包含一个或多个字符的 URL
REGEX:.+\.jpg.*	任何在一个或多个字符后至少包含.jpg 的 URL
REGEX:.+\.jpg$	任何以.jpg 结束的 URL
REGEX:.+\.(jpg|gif|png)$	任何以.jpg、.gif 或.png 结束的 URL
REGEX:^https.+$	以 https 开头的任意 URL
REGEX:(?insx).*\.(jpg|gif|png)$	任何以.jpg、.gif 或.png 结束的 URL，大小写不敏感

可以用在声明前面包含表达式的方式来为正则表达式指定选项（如大小写敏感）。选项字符串（?insx）的功能很实用：它会启动大小写敏感限制，要求显示捕获组，支持单行语义，支持使用#符号添加注释。对于正则表达式，注释并不是所谓的“一次编写，永不阅读”。如果不包含有意义的注释，你可能会很快忘掉自己精心编写的正则表达式到底要做什么。

匹配请求体

在某些情况下，对于很多不相关的操作，网站可能会使用相同的 URL，用请求体来指定希望执行的操作。在匹配条件前面添加前缀 URLWithBody 可以扩展 Match Condition，使其能够与 POST 或 PUT 请求体执行匹配操作。使用这种前缀时，模式串中第一个空白字符之前的内容将作为匹配请求 URL 的 Match Condition，而字符串的其他部分则作为请求体的 Match Condition。应该尽可能具体化 Match Condition 的 URL 部分，从而将强迫 AutoResponder 匹配请求体的次数减至最少。如果请求不包含请求体，则不会匹配任何 URLWithBody 规则。

在匹配条件（Match Condition）中，可以为请求 URL 和请求体分别指定前缀 EXACT:、

NOT:和 REGEX:。例如：

```
URLWithBody:upload.php TextToFindInBody
URLWithBody:login.php EXACT:Action=Login
URLWithBody:ping.php NOT:POST Data I Do Not Care About
URLWithBody:EXACT:https://example.com/upload.php REGEX:^.+TextToFind.*$
URLWithBody:REGEX:^.+/upload.php.*$ REGEX:^.+TailOfPOST$
```

记住，很多 Web Form 的 POST 会对请求体文本进行编码，因此应该确保匹配条件满足这种编码。例如，要匹配如下 POST：

```
POST http://www.enhanceie.com/sandbox/FileForm.asp HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 54

2=This+is+some+text&fileentry2=&_charset_=windows-1252
```

你的匹配条件应该这样写：

```
URLWithBody:/sandbox/FileForm.asp This+is+some+text
```

2.10.2　指定 Action Text

Action Text 是说明当满足匹配条件时，AutoResponder 如何工作。可能是返回内容、对请求重定向或执行一些其他操作，如表 2-16 所示。

任何 Action 都是 Final 或 Non-Final 类型。Non-Final 类型的 Action 的请求可以匹配多个规则。只要规则满足指定的 Final Action，匹配过程就会退出，该 Session 不会进一步执行规则匹配。规则是以出现的顺序持续匹配的，因此应该根据需求对规则顺序进行调整。

表 2-16　　Action Text

Action Text	说明	Final?
filename	将 *filename* 的内容作为响应	yes
http://*targetURL*	返回 *targetURL* 的内容作为响应。该 Action 可以有效地把请求重定向到不同的 URL，而不需要通知客户端应用	yes
*redir:http://*targetURL*	返回指向目标 URL 的 HTTP/307 重定向响应。使用前缀*redir 可以确保客户端知道请求将发送到哪里，从而可以发送正确的 cookie	yes
*bpu	设置请求断点	no
*bpafter	设置响应断点	no
*delay:####	延迟向服务器发送指定请求的毫秒数	no

续表

Action Text	说明	Final?
*flag:*flagname*=*value*	把 Session Flag 设置为指定值。例如，使用*flag:ui-bold=1 会在 Web Session 列表中加粗显示该项。把值置为空就相当于删除指定的 Session Flag	no
*drop	不发送响应，马上关闭客户端连接。从 TCP/IP 层角度看，该连接是优雅断开的，向客户端返回 FIN	yes
*reset	不发送响应，马上关闭客户端连接。从 TCP/IP 层角度看，该连接是粗暴断开的，向客户端返回 RST	yes
*exit	停止正在处理的规则	yes

AutoResponder 选项卡经常被用于从磁盘读取本地文件。当加载本地文件时，Fiddler 会扫描该文件，查看文件是否以一组 HTTP 头开头。如果是以 HTTP 头开头，在响应时会返回这些 HTTP 头，把剩下的文件内容作为响应体。如果不是，Fiddler 会自动生成默认的响应头，把全部文件内容作为响应体。如果返回本地文件，在 Web Session 列表中会以薰衣草背景来显示 Session。

2.10.3 对 Action Text 应用正则表达式

Fiddler 的 AutoResponsder 选项卡支持使用正则表达式，用匹配条件中的文本替换 Action Text（Action 文本）。举个例子，以下规则：

匹配文本	Action Text
REGEX:.+/assets/(.*)	http://example.com/mockup/$1

会把请求 http://example.com/assets/Test1.gif 匹配替换成 http://example.com/mockup/Test1.gif。

以下规则：

匹配文本	Action Text
REGEX:.+example\.com.*	http://proxy.webdbg.com/p.cgi?url=$0

会重写接收到的 URL，因而所有包含 example.com 的 URL 都会作为 URL 参数传给页面 proxy.webdbg.com。

很多网站在生产服务器上提供“不完整”或“缩略版”的脚本，但是为了调试方便，还维护原始脚本。比如，不完整的脚本在如下 URL：

```
http://example.com/main/js/main_c.js
```

而完整版的脚本在：

```
http://example.com/main/js/main.js
```

那么可以写个正则表达式，把不完整的脚本替换成原始版本，如下：

匹配文本	Action Text
REGEX:(?insx)^http://example.com(?'path'.+)_c\.js$	http://example.com${path}.js

该规则会捕捉到 path 的变量中指定的文件路径，使用该变量替换 Action Text 中的${path}。除了改变 URL 的主机或路径，还可以创建类似的规则，可以修改、添加或删除查询字符串参数。

该替换功能很智能，当把 URL 替换成文件路径时，可以把斜杆替换成反斜杆，因此以下规则：

Match Text	Action Text
REGEX:(?insx).+/assets/(?'fname'[^?]*).*	C:\users\ericlaw\desktop\${fname}

会把 http://example.com/assets/img/1.png?bunnies 替换成 C:\users\ericlaw\desktop\img\1.png。

2.10.4　拖放支持

AutoResponsder 选项卡支持使用拖放功能，轻松创建 AutoResponsder 规则。通过该功能，可以拖放 Windows Explorer 下的所有文件或目录，为这些文件自动生成规则。此外，还可以从 Web Session 列表中拖放 Session，从而重用之前捕捉到的响应。可以选中规则并按下 Enter 键或选中规则上下文菜单中的 Edit Response 项，修改规则所保存的响应。

规则（Rules）列表视图的上下文菜单提供表 2-17 所示的命令。

表 2-17　　规则列表视图的菜单

Remove	删除选中的规则。此外，选中一个或多个规则，按下 Delete 键
Promote Rule	把列表中的规则向前移动一个位置。也可以选中一个规则，并按下 Plus 键
Demote Rule	把列表中的规则向后移动一个位置。也可以选中一个规则，并按下 Minus 键

续表

Set Latency...	弹出对话框，要求指定规则匹配的延迟毫秒数。例如，如果指定 50 和一个文件名，Fiddler 在把选中的文件作为响应返回前，会等待 50 毫秒。 可以在前面指定+或-符号，以调整当前值而不是指定绝对值
Clone Rule	复制当前规则，并附加到列表中
Edit Response	对于以文件形式备份的规则，打开文件修改它。对于在之前捕捉的响应中备份的规则，打开独立的 Inspector 窗口，支持修改响应
Generate File	对于在之前捕捉的响应中备份的规则，把响应保存到文件中，调整规则指向那个文件。当你需要使用独立的编辑器（如 Expression Web 或 Visual Studio）编辑响应时，使用该命令
Edit File With...	弹出对话框，可以选择应用，编辑指定响应。使用独立的编辑器来修改响应，而且该编辑器不是处理保存响应文件类型的默认程序
Open URL	使用默认的 Web 浏览器打开在匹配条件（Match Condition）中指定的 URL
Export All...	弹出对话框，把当前规则保存成 FARX 文件。后期可以导入该文件，以重新加载规则

2.10.5 FARX 文件

Fiddler AutoResponder XML (FARX)文件包含一组 AutoResponsder 规则。当前的 AutoResponder 规则集在退出 Fiddler 时会自动保存，重新启动 Fiddler 时会重新加载。自动保存的文件名是 AutoResponder.xml，如果使用 Export（导出）命令保存独立文件，会包含.farx 文件扩展名。

FARX 文件包含完整的规则集，包括之前捕捉到的响应，该响应可以用于重新播放。二进制响应是 base64 编码和压缩方式，可以减少文件大小，但是 AutoResponder 规则的大集合（或包含大响应的规则的小集合）会导致 FARX 文件变得很大。

因为默认的 AutoResponsder 规则集是在 Fiddler 启动时自动重新加载的，大的默认 FARZ 文件会导致 Fiddler 启动变慢。要避免启动慢的问题，可把大的规则集导出到选中的 FARX 文件中，在关闭 Fiddler 之前从默认列表删除这些规则。

2.11 TEXTWIZARD

当和 Web 内容交互时，通常使用一种或多种格式对文本进行编码。TextWizard 支持快速

把文本转换成常用格式，或把常用格式转换成文本。

可以通过点击 Tools > TextWizard 或按下 CTRL+E 键打开 TextWizard，如图 2-17 所示。可以同时打开多个 TextWizard 拷贝。

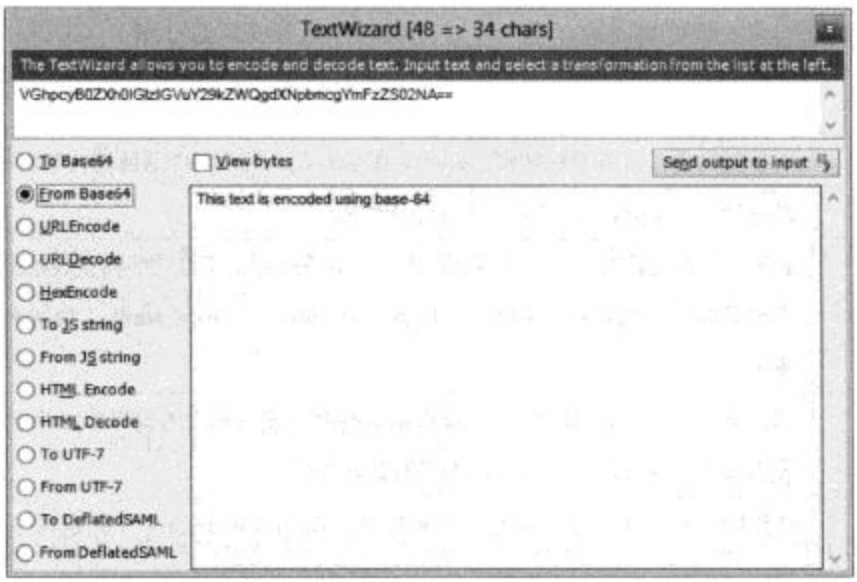

图 2-17

TextWizard 上方是个输入框（Input box），可以在该输入框输入或粘贴文本。在 TextWizard 左下方是一组转换按钮，可以选择某种转换方式，生成输出文本，在右下方的输出框（Output box）以只读方式显示。

可用的转换方式列表是固定的，每次只能应用一种转换方式。如果想要“挂接”多种转换方式，可使用 Send output to input 按钮，把最后一个操作的输出拷贝到输入文本框，然后选中要应用的下一步转换。

工具打开后，如果系统剪贴板的文本是 32KB 或更少，会自动复制到输入框中。当更新输入框的文本时，会立即应用选中的转换方式。标题栏给出了输入的字符数以及生成的输出字符数。

如果任何转换会生成空字符（会终止字符串），TextWizard 会以 Unicode 替换符替换空格（0xFFFD, ◆）。

复选框 View bytes 支持以十六进制形式查看输出，当解析无法以文本形式显示二进制内容时会很有用。

Send output to input 按钮支持把输入文本替换成输出文本，当挂接一系列的转换时会使用该功能。

表 2-18 是可用的转换方式列表。

表 2-18 可用转换方式

To Base64	把输入的字符串转换成 UTF8，然后使用 Base64 编码，把结果编码成 7 位的 ASCII 码字符串
From Base64	把输入的字符串从 7 位的 ASCII 码解析成字节数组，然后把结果解释成 UTF8 字符串
URLEncode	在输入字符串上应用 URL 编码规则
URLDecode	从输入字符串上对 escape 序列进行解码
HexEncode	把输入字符串的每个字符转换成以字符%开头的十六进制形式
To JS string	把\替换成\\，Carriage 替换成\r，换行符替换成\n，"替换成\"，超出 ASCII 127 的字符以\u*XXXX* 表示，其中 *XXXX* 是 Unicode 编码
From JS string	To JS string 操作的反操作
HTML Encode	使用 HTML 实体对输入字符串进行编码，如<编码成<
HTML Decode	HTML Encode 操作的反操作
To UTF-7	把输入字符串转换成以字节序标识符开头的 UTF-7 字符串
From UTF-7	把输入字符串转换成 UTF-7 字符串
To Deflated-SAML	把文本转换成 UTF-8 字节，并使用 DEFLATE 压缩算法，把结果字符串转换成 Base64，对最后结果进行 URL 编码
From Deflated-SAML	把字符串以 URLDecodes 解析成 Base64，转换成字节数组，对数组进行解压缩，把结果数组转换成 UTF-8 字符串

字符编码

默认情况下，TextWizard 会使用 UTF8 在字符和字节之间相互转换。绝大多数应用会使用 UTF8 作为标准的文本编码方式，但是如果你的站点使用的是另一种编码方式，可以设置 Preference，选择需要的文本编码。

设置 Preference fiddler.textwizard.InputEncoding，指定应该使用哪个字符集对%编码的字符进行解码。设置 Preference fiddler.textwizard.OutputEncoding，指定应该使用哪个字符集进行

%编码。提供的字符串值必须是.NET 框架能够识别的编码名，有效的编码名在以下链接可以找到：

```
http://fiddler2.com/r/?EncodingNames.
```

2.12　COMPOSER 选项卡

Composer 选项卡支持手动构建和发送 HTTP、HTTPS 和 FTP 请求。此外，还可以从 Web Session 列表中拖拽 Session，把它放到 Composer 选项卡中，把该 Session 的请求复制到用户界面。点击 Execute 按钮，把请求发送到服务器端。

Composer 选项卡是由三个子选项卡组成的：Parsed、Raw 和 Options 选项卡。下面将按序解释每个选项卡，为了便于说明，从右开始。

请求选项

Options 选项卡提供的选项如表 2-19 所示。

表 2-19　　Options 选项

Inspect Session	请求执行后，Inspectors 选项卡会被激活，可以查看请求的结果
Fix Content-Length header*	该选项控制 Composer 是否会自动添加或修改 Content-Length 请求头，表示请求体的大小。 在很多情况下，缺少适当的 Content-Length 头的请求会 hang 住或导致 HTTP 响应出错
Follow Redirects*	该选项控制 Composer 是否会自动使用响应的 Location 头，遵循 HTTP/3xx 重定向。 如果选中该选项，Composer 在失败之前最多会执行 10 次重定向
Automatically Authenticate*	该选项控制 Composer 是否会自动响应服务器的 HTTP/401 或 HTTP/407 认证需求。 如果选中该选项，会使用 Fiddler 所运行的账户的 Windows 证书自动响应这些问题。 要提供不同的证书集，设置 Preference fiddler.composer.AutoAuthCreds。 如果服务器需要的证书和提供方的证书不同，请求会失败，通常会返回响应 HTTP/403
Tear off button	该按钮会从主 Fiddler 窗口删除 Composer，并把它作为独立的悬浮窗口打开。 如果选中 Inspect Session 选项，该选项就非常有用，因为它支持同时查看 Composer 和 Inspector 选项卡

以*标识的选项只适用于使用 Parsed 选项卡发送的请求；从 Raw 选项卡发送的请求不支持这些选项。

Raw 请求

Raw 选项卡提供简单的文本框，可以在该文本框中输入合适的 HTTP 请求。如果请求格式不对（比如忘记在请求头后截断 CRLF），点击 Execute 按钮不会发起请求。

Raw 选项卡很少使用——绝大多数请求应该使用 Parsed 选项卡。

Parsed 请求

Parsed 选项卡支持为每个请求组件使用独立的输入框构建请求。

选项卡的上方是三个输入框：第一个输入框支持指定 HTTP 方法（如 POST）；第二个输入框支持指定请求的绝对 URL（必须以 http://、https://或 ftp://开头）；第三个输入框支持指定 HTTP 版本号（通常是 HTTP/1.1）。

在最上方的线下是两个大的文本区：最上方的输入框支持编辑请求头。最下方的输入框支持编辑请求体。如果当前选中的 HTTP 方法不支持请求体（如 GET 方法），当在请求体输入框输入文本时，该输入框会显示红色。

发送序列化请求

在某些场景中，可以发送多个请求，这些请求除了某个数字外，其他完全相同。例如，当要下载一系列顺序命名的图像时，每个下载请求只有文件名不同。Composer 可以生成一系列按序编号的请求——只需要在 URL 中数字出现的位置以#符号表示即可，如图 2-18 所示。

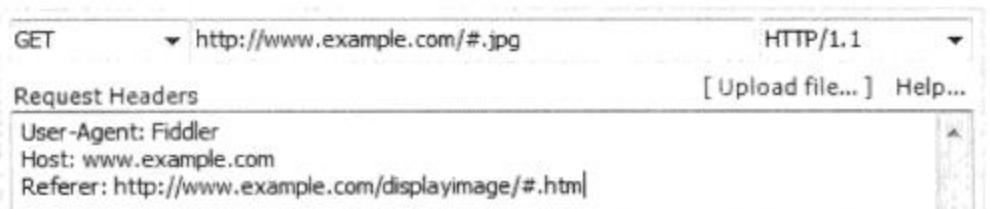

图 2-18

当请求执行后，Fiddler 会弹出对话框，要求输入第一个数字，如图 2-19。可以随便输入一个数字（如 8），如果所有的数字都必须是相同位数的数字，就在数值之前以 0 填充。如要确保所有的 URL 数字都包含两个整数，则应该是 08：

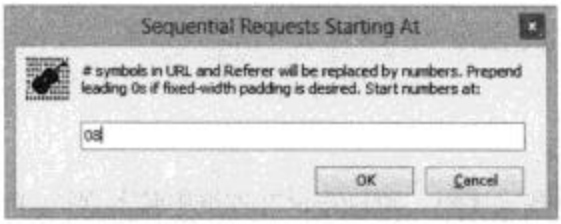

图 2-19

然后，会弹出对话框，输入一个数字，表示 Fiddler 要停止发送请求，如图 2-20 所示。

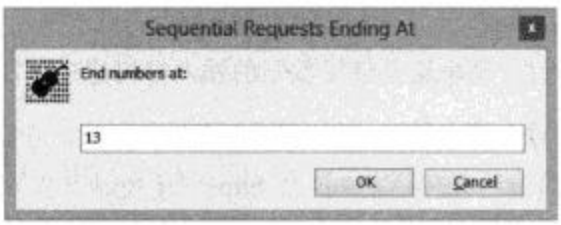

图 2-20

提供了开始和结束数字后，Fiddler 会发送指定区间的请求序列，如图 2-21 所示。

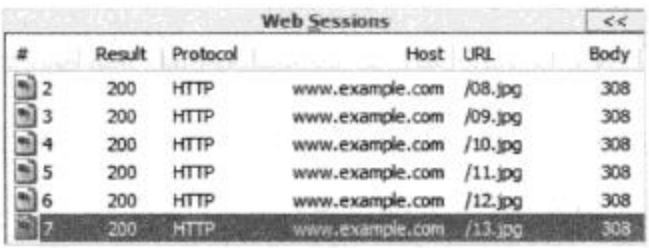

图 2-21

有些服务器不会返回响应，除非包含期望的 Referer 头。在请求的 Referer 头前面包含#，表示 Fiddler 要用当前的请求号替换该字符。

序列请求功能只有在使用 Parsed 选项卡时才可用；如果请求是以 Raw 选项卡编写，#会作为纯文本处理。

文件上传请求

可以通过点击选项卡右上方的上传文件（Upload File）链接创建文件上传，会弹出 Select File for Upload 文件选择窗口。如果请求方法是 PUT，只能选择一个文件；如果请求方法是 POST，可以选择多个文件。

选中上传的文件后，Composer 可以创建包含适当格式的请求；当请求执行时，请求体中

的所有@INCLUDE 引用会替换成指定文件的内容。

在 HTTP 中，文件上传是使用 PUT 或 POST 方法执行的。当使用 PUT 方法执行文件上传时，请求体通常包含文件的原始内容，如图 2-22 所示。

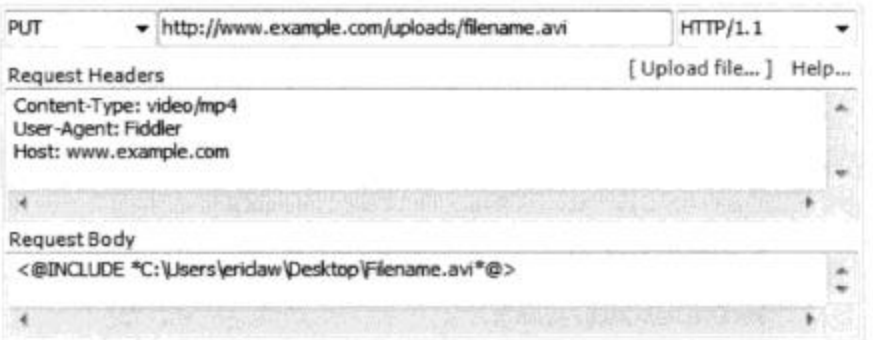

图 2-22

相比之下，使用 POST 方法上传通常会使用 Content-Type: multipart/form-data 对请求体进行格式化，如图 2-23 所示。

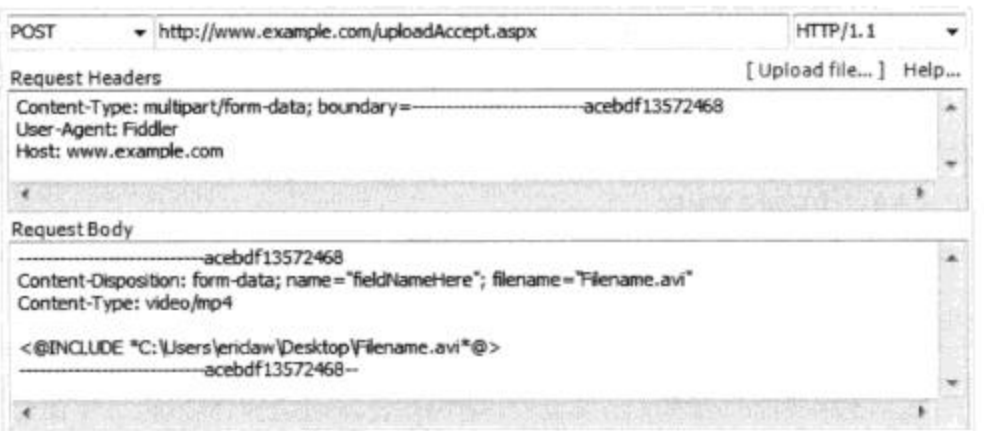

图 2-23

对于 POST 上传方式，需要编辑 Name 属性，确保它和服务器的表单字段名称匹配。

自动请求断点

在某些情况下，Fiddler 的请求 Inspector 的编辑体验比 Composer 选项卡更好。点击 Execute 按钮的同时，按下 Shift 键，可以在新的请求上设置断点。在断点处，新的 Session 会立即暂停，并激活请求 Inspector。这样，可以在新的请求发送到服务器之前，使用 Fiddler Inspector 进行修改。

2.13　Log 选项卡

Log 选项卡收集日志消息字符串，这些字符串是由扩展、FiddlerScript 或 Fiddler 本身生成的。Fiddler 会记录应用事件（如当保存或加载 SAZ 文件）以及系统事件（如系统的网络连接丢失或恢复）的响应通知。

在当前的 Fiddler 版本中，WebSocket 数据流也可以在 Log 选项卡上显示，如图 2-24 所示。

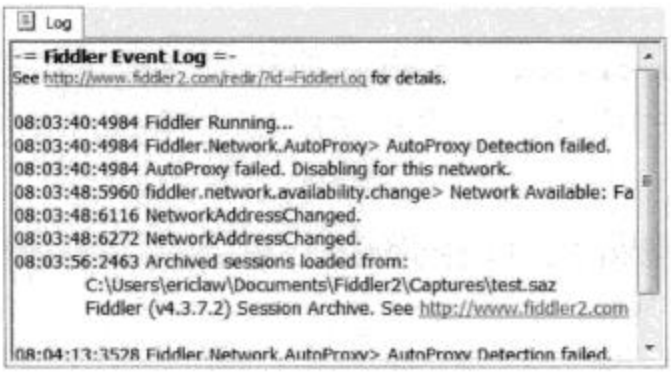

图 2-24

右击 Log 文本框，会显示上下文菜单，提供和日志交互的一些基本命令：

Copy	把选中的文本复制到剪贴板
Save...	把当前日志保存到磁盘文件中。可以是纯文本格式，也可以是格式化的富文本格式（.rtf）
Clear	清空 Log 中的所有文本

Log 选项卡支持简单的宏命令，可以从 QuickExec 文本框中调用这些命令。输入 log @Log.Clear 可清空日志。输入 log @Log.Save 可在 Web Session 列表中生成新的 Session，该 Session 的响应体包含 Log 选项卡的文本。输入 log "@Log.Erport\"filename\""可把 Log 选项卡的文本保存到指定的文件。文件名以.rtf 结尾，可将文件保存为富文本格式，保存大小和权重，或以文件名以.txt 结尾保存为纯文本。

2.14　Find Session 窗口

Fiddler 的 Find Session 窗口支持搜索捕捉到的请求和响应，选中包含感兴趣的文本的

Session，如图 2-25 所示。通过 Edit 菜单选项，或者在 Fiddler 中按下快捷键 CTRL+F，可打开 Find Session 窗口。

图 2-25

最上方的 Find 输入框支持指定要搜索的文本。该搜索框的下拉列表会显示历史搜索项，在输入时会自动补全。

Options 选项框控制如何搜索。Search 下拉选项框支持指定搜索选项，包括 Requests and responses（默认值）、Requests only、Responses only 和 URLs only。除非选择 URLs only，否则可使用 Examine 下拉条指定是否希望搜索 Session 的 Headers、Bodies 或是两者都搜索（默认值）。

在下拉选项框下方是一组复选框。Match case 选项框表示搜索是大小写敏感的。Regular Expression 表示 Fiddler 应该把搜索文本作为正则表达式处理。Search binaries 选项支持 Fiddler 在 Session 内部搜索，其中 Content-Type 头表示请求体或响应体的二进制类型，如音频、视频、Flash 对象等。Decode compressed content 选项支持 Fiddler 从搜索的请求和响应中删除 HTTP 内容和传输编码。该选项会严重影响搜索速度，从请求体和响应体中删除的编码不可恢复。当打开 Find Session 窗口时，如果在 Web Session 列表中打开多个 Session，就会激活（而且默认选中）Search only selected sessions 选项框。Select matches 选项会使得 Fiddler 自动选中包含搜索的文本的所有 Session。Unmark old results 选项会对之前搜索结果中所有高亮的 Session 取消高亮显示。Result Highlight 下拉选项框支持选择背景颜色，所有包含搜索文本的 Session 都会应用该背景颜色。如果取消选中 Unmark old results 选项，每次执行 Find Sessions 操作时，会循环使用不同的颜色。也就是说，当执行多次搜索时，每次搜索会用不同的颜色显示。

配置了搜索后，按下 Find Sessions 按钮可以执行搜索。Find Sessions 窗口会关闭，在 Web Session 列表中会高亮显示匹配的搜索结果（如果选中该选项）。Fiddler 的状态栏会给

出匹配数。

按下 Cancel 或点击 Escape 可以关闭窗口，不执行搜索。

2.15　Hosts Remapping 工具

Hosts Remapping（主机重映射）工具可以很方便从一台主机重路由到另一台主机，重写主机名及其 IP 地址列表之间普通的 DNS 关联。要启用该功能，可在 Fiddler 的 Tools 菜单点击 HOSTS...项，会弹出 Host Remapping 窗口，如图 2-26 所示。

图 2-26

可以通过 Enable remapping 复选框启用或禁用该功能。在文本框中，输入重写列表，每行一条。在第一列中输入新的主机或 IP 地址、一个或多个空白符，最后是要重写的主机名。每行如果以#号开头，表示该行是注释。

可以使用窗口最下方的链接，导入系统的 HOSTS 文件（%SYSTEMROOT%\System32\drivers\ etc\hosts）。

和 Windows 的 HOSTS 文件不同，该功能不需要指定新目标的 IP 地址；相反地，可以指定一台主机。需要的话，还可以指定目标端口号。

以下规则：

```
127.0.0.1:8088 meddler
```

会发送 http://meddler/ 到 http://127.0.0.1:8088/的所有请求。

通过 Host Remapping 工具从一个主机名重新路由到另一个主机名的 Session 在 Web Session 列表中会以淡蓝色表示。重路由的 HTTPS Session 包含 Session 标志位 X-IgnoreCert CNMismatch 和 X-OverrideCertCN，以避免出现“证书名不匹配”错误。

第3章 技巧和概念

3.1 使用 Fiddler 重定向数据流

Fiddler 提供了三种改变请求目标的方法。这些方法对于客户端和服务器有不同的含义，了解其中的区别可以帮助你更自如地做出选择。

方法 1 – 重写

重写（rewrite）是指使用 Fiddler 修改请求头的内容（目标 URL 或者请求头的 HOST 字段），导致本来是发送给服务器 A 的请求发送到服务器 B。

客户端不知道其发送的请求发生了改变，因此该请求所包含的任何 cookie 还将属于原始的 URL，而不是新的 URL。同样，如果你在浏览器的地址栏或 DOM 树中查看该资源的 URL，你会发现还是原始的 URL，而不是重写生成的新 URL。

因为请求的 URL 和 HOST 字段都已经修改过了，所以重写后的请求可以安全地发送到上游代理服务器而不会发生任何问题。目标服务器在接收到的请求中可以查看请求头的 HOST 字段。

如果请求是通过 HTTPS 和加密方式发送的，你可能希望控制客户端计算机接收到的证书，以避免出现“证书名称不匹配”这类错误。要做到这一点，可以使用会话的 X-OverrideCertCN 标记来设置 Fiddler 将要发送给客户端的 Subject CN（Certificate Name）。

方法 2 – 重路由

重路由（reroute）是指不修改请求，把本该发送给服务器 A 的请求发送给服务器 B。和“重写”不同，Fiddler 在连接服务器之前，发现该请求被标识为需要重路由，因此，它就不会连接到该请求的 HOST 字段所指定的主机，而是连接到这个会话的 X-OverrideHost 标志中所

指定的新目标主机。

客户端不知道请求发生了变化，因为请求中的 Host header、URL 和 Cookie 中保存都还是原始主机的信息，而不是重路由到的目标主机的信息。重路由请求只有在 Fiddler 本身能够确定该请求要发送的目标 IP 地址时才能够正常工作。如果上游使用了网关代理服务器，它将无法处理 Fiddler 的重路由请求，转而使用请求中的原始 HOST 字段进行路由。因此，这种情况下需要使用 Host Remapping 工具设置标志位 X-OverrideHost 和 bypassGateway 的值，以确保重路由的请求可以绕过任何上游网关代理。

当 Web 服务器接收到请求时，请求头中 Host 字段的内容还是原始的服务器主机名，因为请求虽然做了重路由，但并没有被修改。服务器可能会拒绝该请求，返回“No such host is known.”的 HTTP/400 错误信息，这取决于服务器的配置。

当 HTTPS 请求被重路由到另一台服务器上，客户端或 Fiddler 会给出类似“证书名称不匹配”的错误信息。举个例子，当使用下面这个 FiddlerScript 对服务器做重映射时：

```
if (oSession.HTTPMethodIs("CONNECT") && oSession.HostnameIs("example.com"))
{
  oSession["X-OverrideHost"] ="www.fiddler2.com";
}
```

加载 https://example.com/时会弹出证书不匹配的警告，显示服务器提供的是 www.fiddler2.com 的证书，而不是 example.com 的证书，如图 3-1 所示。

图 3-1

可以通过设置标志位 X-IgnoreCertCNMismatch，避免弹出证书错误消息：

```
if (oSession.HTTPMethodIs("CONNECT") &&
  oSession.HostnameIs("example.com"))
{
  oSession["X-OverrideHost"] ="www.fiddler2.com";

  // Set flag to suppress Fiddler's HTTPS Name Mismatch errors
```

```
    oSession["X-IgnoreCertCNMismatch"] = "no worries mate";
}
```

如果目标站点使用的是过期或自签名的测试证书，而你希望 Fiddler 能够忽略会话（session）所有的证书错误警告时，可以设置标志位 X-IgnoreCertErrors。

Host Remapping（主机名重映射）工具会自动设置 X-IgnoreCertCNMismatch 标志。当忽略证书名称不匹配的错误时，虽然浏览器地址栏显示的是 https://example.com/，而实际看到的却是 https://www.fiddler2.com/的页面。

方法 3 – 重定向

重定向（redirect）是指通过 Fiddler 给客户端返回 HTTP/307 重定向响应，使得客户端使用新的 URL 重新发送请求。因为负责重新发送请求的是客户端，所以它知道请求目标发生了变化，因而在发送新的请求时会包含新的 Host 头和 URL。客户端会给目标 URL 发送合适的 cookie，如果请求是通过 HTTPS 发送的，客户端期望目标服务器为新的目标 URL 使用合适的证书。因为发送的新请求所包含的是新的目标 URL 及其匹配的 Host，这种请求能够安全地通过上游网关代理发送到期望的服务器上。

重定向请求的功能

Fiddler 支持通过以下几个功能实现上述三种基础技术：

1）通过 Tools 菜单中的 HOSTS...命令，可以把数据流从一个 IP 地址重路由到另一个 IP 地址。

2）通过 FiddlerScript 或扩展，设置 Session 对象的 X-OverrideHost 标志，可以把数据流从一个 IP 地址重路由到另一个 IP 地址。

3）通过 FiddlerScript 或扩展，设置 Session 对象的 host 属性或 fullUrl 属性，把一个 URL 的请求重写到另一个 URL。

4）通过 FiddlerScript 或某个扩展的 utilCreateResponseAndBypassServer 方法，可以把一个 URL 重定向到另一个 URL。

5）通过 AutoResponder，在 Action 文本框中输入新的 URL，可以把一个 URL 的请求重写到另一个 URL。

6）通过 AutoResponder，在 Action 文本框中输入前缀*redir:，可以把一个 URL 的请求重定向到另一个 URL。

7）通过 AutoResponder，在 Action 文本框中输入*flag: X-OverrideHost=targethost，会把一个 URL 的请求重路由到另一个 URL。

3.2　Session 比较

调试任务中最常见的一种情形是确定为什么某个请求成功而另一个请求失败。要完成这种任务，需要依靠具有 Session 比较功能的强大工具的支持。

要比较两个 Session，只需要在 Web Sessions 列表中选中这两个 Session，然后从上下文菜单中选择 Compare 按钮（或者按下 CTRL+W 键）。执行对比命令时，那两个选中的 Session 会被保存到临时文件中，然后启动文件比较工具，并把两个临时文件的路径传递给它，如图 3-2 所示。

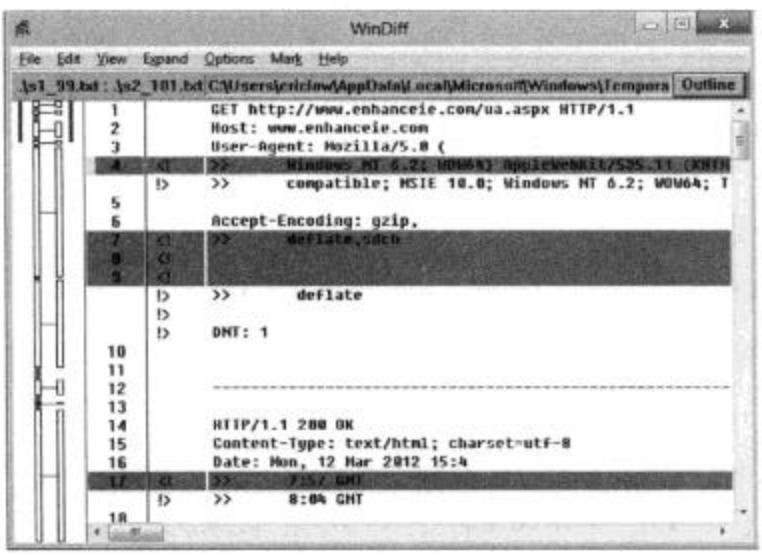

图 3-2

Fiddler 没有自带文件对比工具，不过很多开发者安装了 WinDiff、Odd、Beyond Compare、WinMerge 或其他类似的工具。如果你已经安装了 WinDiff 工具，把它拷贝到\Program Files\Fiddler2 文件夹下，或者把它拷贝到系统的某个 PATH 路径下（如 C:\Windows\ System32），这样 Fiddler 就可以找到它。如果你没有安装任何文件对比工具，可以从 http://WinMerge.org 下载 WinMerge。

如果你希望在 Fiddler 中使用的比较工具不是 WinDiff，使用 QuickExec 框或 about:config 标签页设置三个选项偏好（Preferences）后，就可以达成所愿了。首先，提供该工具的路径：

```
PREFS SET fiddler.config.path.differ "C:\Program Files (x86) \Microsoft Visual Studio 11.0\Common7\IDE\devenv.exe"
```

然后，配置期望安装的工具的命令行参数：

```
PREFS SET fiddler.differ.Params "/diff \"{0}\" \"{1}\""
```

你还可以设置“alternate（其他选项）”命令行，当你在调用文件比较功能时，按住 SHIFT 键就会执行这个命令：

```
PREFS SET fiddler.differ.ParamsAlt "/diff \"{0}\" \"{1}\""
```

借助 alternate 命令行功能，可以使用文件对比工具的多种模式。例如，WinDiff 支持-p 命令行标志位，会在每个比较的第一个符号处断点。当你在点击“Compare”选项时按住 SHIFT 键，Fiddler 会把-p 标志位传递给 WinDiff，在每个断点处会新起一行，如图 3-3 所示。这种方式对于查看行长度很长的文本很有用（如 URL 的查询字符串）。

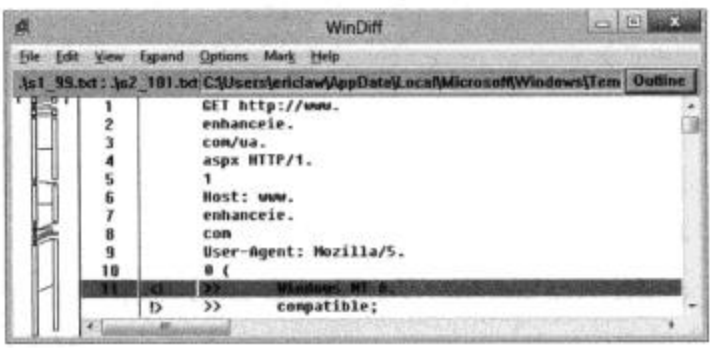

图 3-3

UltraDiff

默认情况下，Fiddler 与 WinDiff 这样基于行的比较工具类似，会对临时文件重新格式化，以便查看。这个功能是通过 UltraDiff 实现的，它会改变两个文件中的请求头的顺序，因此在两个 Session 中值完全相同的 headers 会显示在最前面，然后是取值不同的 headers，最后面显示的是完全不同的 Session 的 headers。UltraDiff 对请求的 URL 重新组织，从而保证即使两个 URL 只有微小的区别（比如 URL 长 512 个字符，在该 URL 中间有个字符不同），这种区别在显示上可以很容易看出来。

如果你希望 Fiddler 在保存请求的临时文件中不要修改其 headers 和 URL，可以把 fiddler.differ.ultradiff 选项设置为 false。

同时比较多个 Sessions

如果你想同时比较多个 Sessions（甚至是整个 SAZ 文件），应该使用扩展工具 Traffic

Differ，这个工具属于扩展，会在第 6 章中详细介绍。该扩展支持对 Web Sessions 列表进行比较，而且支持对列表中的每个 Session 做文本比较。

3.3　断点调试

在传统的调试器中，你可以选择程序执行时在哪个指令处中断，这些点就是断点。当程序在断点处停止时，你可以查看程序的状态，修改执行的内存或数据流，以改变程序的行为。

Fiddler 为 Web 请求提供了类似的，基于断点的调试功能。Session 在执行过程中有两个可能中断的时间点：

1）从客户端读到请求后，在请求被发送给服务器之前。.

2）在服务器发回响应后，在响应返回给客户端之前。

这两个断点称为请求断点（request breakpoints）和响应断点（response breakpoints）。

当程序在请求中断点停止执行时，你可以任意修改该请求，包括 URL、headers 或 body。你还可以选择不发送该请求给服务器，自定义响应返回给客户端。大多数情况下，客户端应用不知道 Fiddler 修改了它的请求。

同样，当程序在响应断点停止执行时，你可以任意修改响应的内容，包括 headers 或 body。你还可以自己生成新的响应，取代从服务器端接收到的响应。在响应断点，你“也可以”修改任意客户端发送的请求，但是由于该请求已经发送给服务器了，这些修改只在 Fiddler 中生效——服务器永远都看不到你做的这些修改。

设置断点

Fiddler 支持多种断点设置方式，但实质上，它们都是通过修改 Session 的两个标志位 X-BreakRequest 和 X-BreakResponse 实现的。

通过访问菜单 Rules > Automatic Breakpoints 或点击 Fiddler 状态栏左侧的第 3 个面板，都可以很容易地对所有的请求或响应设置断点。这种设置全局性断点方式的缺点在于你很可能会因为要手工越过那些不感兴趣的中断点而感到厌烦。

相反地，你可以采用 Filters 或 AutoResponder 标签来更精确地定位断点。Filters 标签支持对以下几种类型的 Session 设置断点：1）使用 POST 方法；2）请求中包含查询字符串；3）通过 XMLHttpRequest 方式发送；4）返回特定的 Content-Type。使用 AutoResponder 标签，可以

对 URL 能够匹配指定文本或正则表达式的各种请求设置请求断点或响应断点。比如要创建请求断点，可以设置规则的 Action 文本为*bpu；要创建响应断点，可以把文本设置为*bpafter。

如果你想要定义更复杂的断点条件，可以使用菜单栏 Rules > Customize Rules，在 FiddlerScript 的 OnBeforeRequest、OnPeekAtResponseHeaders 或 OnBeforeResponse 方法中设置 X-BreakRequest 或 X-BreakResponse 标志位。

在默认的 FiddlerScript 文件中，通过 QuickExec 框可以很容易地设置单个请求断点和单个响应断点。只需要输入：

```
bpu TextFromURL
```

就可以为 URL 包含指定文本的所有会话设置响应断点。

```
bpafter TextFromURL
```

可以为 URL 包含指定文本的所有会话设置响应断点。

你还可以使用特定的 HTTP 方法，为任意请求设置请求断点：

```
bpm POST
```

也可以为所有返回特定 HTTP 状态码的响应设置响应断点：

```
bps 307
```

键入任何一种不含参数的断点命令（bpu、bpafter、bpm 或 bps），将会清除该断点。

重新编译 FiddlerScript（或重新启动 Fiddler）也会清除 QuickExec 所设置的断点。

通过 Inspectors 修改

当程序在断点处停止后，可以在 Inspectors 中编辑请求或响应。正如在 Inspectors 那一章所描述的，很多 Inspectors 支持 Read/Write 功能，你可以使用这些功能来改变请求和响应的 header 或 body。Inspector 会自动提交在程序中断时所执行的修改。

除了使用 Inspectors，你还可以复制一份之前捕获到的响应给中断了的 Session。当在 Web Sessions 列表中选中两个 Session，其中一个 Session 在断点处中断，另一个 Session 已经完成，这时就可以通过上下文菜单中的 Clone Response 命令来复制响应，它会把已经完成的 Session 的响应拷贝给中断的 Session。借助这个功能，可以很容易地复制一份之前捕获的（或修改的）响应给后面的请求。

Breakpoint Bar（断点栏）

当 Session 停止执行时，在 Request Inspectors 和 Response Inspectors 之间会出现一个红色

的 Breakpoint Bar。它由两个按钮和一个下拉框组成，如图 3-4 所示。

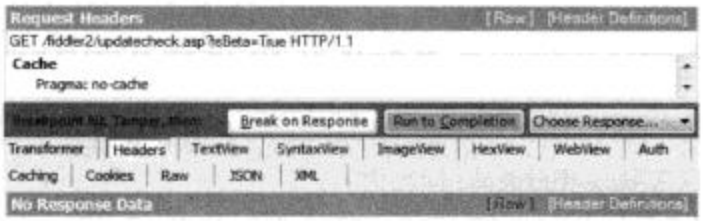

图 3-4

只有当程序在请求的断点处停止时，黄色的 Break on Response 按钮会处于可点击状态。当点击该按钮，它会为当前的 Session 设置响应断点，然后使程序恢复执行，向服务器发送（可能修改过的）请求。点击绿色的 Run to Completion 按钮后，程序会继续执行直至程序结束，不会在接收到响应时中断。

在 Breakpoint Bar 右侧的下拉框中，可以选择特定的响应模板或文件作为该请求的响应，如图 3-5 所示。

从下拉框中选中某个文件后，它会马上被加载到响应的 Inspectors（观察窗口）中，以便在返回给客户端之前进一步修改响应的具体内容（但不需要改变磁盘文件）。如果你在请求断点处使用下拉框中选中的内容作为响应，该请求就不会继续发送给服务端，因为 Fiddler 会返回响应。

同时恢复多个 Session

如果设置的断点目标不够明确，可能会存在很多断点请求，如图 3-6 所示你希望尽快越过这些断点继续执行。

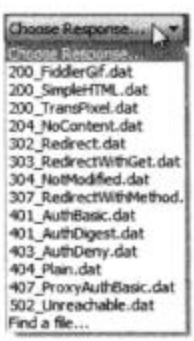

图 3-5

图 3-6

点击 Fiddler 工具栏的 Resume 按钮或在 QuickExec 输入框中键入 g，所有终端的 Sessions 就会立即开始继续执行。

第4章 配置 Fiddler 和客户端

4.1 Fiddler 选项

点击 Tools > Fiddler Options，会弹出 Fiddler 选项窗口，其中包含控制 Fiddler 行为方式的很多基本设置。该选项窗口中的很多设置需要重启 Fiddler 才能够生效，尽管某些修改会马上生效，或者在切换到 Fiddler 的 Capture Traffic 选项时会生效。调整好选项后，点击窗口下方的 OK 按钮可以保存修改，点击 Cancel 按钮则会放弃修改。点击下方的 Help 链接可以查看选项的帮助信息。

4.1.1 常用选项

常用选项卡可以对 Fiddler 的常用操作设置进行调整。

Check for updates on startup 选项控制 Fiddler 在启动时连接 Web 服务检查是否有新版本的 Fiddler。没有选中这个选项时，点击 Fiddler 帮助菜单中的 Check for Updates...选项，可以手工检查是否有新版本的 Fiddler。

Participate in the Fiddler Improvement Program 选项只在某些版本的 Fiddler 中可用（目前只有 Beta 版本）。选中这个选项时，Fiddler 会将用户的数据匿名上传到 Web 服务器上。数据中包括系统信息和用户使用 Fiddler 的情况，但不包含个人隐私信息。该数据将用作改进 Fiddler 的参考。

Show a message when HTTP protocol violations are encountered 选项决定 Fiddler 在遇到错误的数据流时是否会弹出消息框。协议错误对客户端和服务器都会有影响，因此在调试自己的客户端和网站时，可以打开这个选项。在 Web Session 列表中会使用黄色背景标记出错误的数据流，该 Session 的属性中包含了一份出现了协议错误的消息的副本。

Enable IPv6 (if available)选项控制 Fiddler 是否可以监听和尝试连接 IPv6 终端。即使没有选中这个选项，如果 DNS 返回的地址不是 IPv4 地址，Fiddler 还是会尝试把地址当作 IPv6 进行连接。

Map requests to originating application 选项控制 Fiddler 是否需要确定请求来自哪个本地运行的程序。本地进程在 Web Session 列表中的 Process 列中显示，并保存在 Session 的 X-Process-Info 标志中。当查询 Windows 获取网络连接列表以及与该列表相关的各个进程时，该功能才会生效。有些请求无法成功地映射回原始的进程，值得一提的是，在外围设备或远程客户端计算机上发起的请求相关的进程信息也无法获取。

Automatically stream audio & video 选项用于确定对于 MIME 类型为 audio/*和 video/*的响应，数据是否自动发送到客户端，而不需要在 Fiddler 内部缓存。当设置了该选项后，就不能再用 Fiddler 的 Inspectors 或 FiddlerScript 来修改音频或视频响应了。设置这个选项的优点在于在客户端浏览器或应用中可以即时“回放”流媒体响应。要在 Fiddler 中流式显示所有响应，可以点击 Fiddler 工具栏中的 Stream 按钮。

Systemwide Hotkey 用于设置快捷键，当 Fiddler 最小化或者处在非活动状态时，快捷键可以恢复并激活 Fiddler。

4.1.2 HTTPS 选项

HTTPS 选项卡用于控制 Fiddler 与监测安全传输的数据流相关的设置。

Capture HTTPS CONNECTs 选项决定 Fiddler 是否会注册为可以接收 HTTPS 请求的系统代理。如果没有选中该选项，其他所有的 HTTPS 选项都不可用。

Decrypt HTTPS traffic 选项决定 Fiddler 是否解析使用 CONNECT 通道发送的 HTTPS 请求和响应。点击复选框附近的“Learn more...”链接，会弹出一个帮助页面，解释数据流解密时是如何工作的。当你第一次选中解密选项，Fiddler 会生成一个新的自签名的证书，询问你是否想对 Windows 进行配置以信任这个根证书。

当选中解密按钮，会显示一个下拉框，可以选择默认解析哪些数据流。包含的选项如下：

- All processes：对所有数据流都做解析。
- Browsers only：只有当进程名是 Web 浏览器时，才做解析。
- Non-Browsers：只有当进程名不是 Web 浏览器时，才做解析。

- Remote clients：只有当进程名未知时（该请求来自远程计算机或设备），才做解析。

Decryption Process Filter 对于避免解析不想要的数据流是非常有用的——比如你可能不关心类似 Dropbox 这样的文件同步程序所发出的后台请求，那么可以使用该选项轻松跳过对这种数据流的解析。

Ignore server certificate errors 复选框控制服务器提供了无效的安全证书时，Fiddler 是否应该弹出警告。如果选中该选项，Fiddler 在连接 HTTPS 服务器时，会忽略遇到的所有证书错误。如果没有选中该选项，Fiddler 在连接 HTTPS 网站时，如果遇到证书错误，会弹出告警。这种错误对于客户端应用是“透明”的，因为 Fiddler 总是能生成有效的安全证书。一般而言，不应该选中该选项，除非你确定自己是在可信的网络环境中，该网络中确实存在无效的服务器证书（比如在自签名的开发服务器上）。

在 Skip decryption for the following hosts 选项框中可以指定对选中的服务器的数据流不做解析。该功能可以避开不想解析的数据流。例如，你可能不关心连接 Exchange RPC-over-HTTPS 终端的 HTTPs 请求，该过滤器可以屏蔽这种数据流。如果是个列表，使用分号进行分隔，* 作为通配符。举个例子，要屏蔽网站 example.com 及其子目录下的所有数据流以及 fiddler2.com 的数据流，可以把过滤器指定为 fiddler2.com;*example.com。

Export Root Certificate to Desktop 按钮会把 Fiddler 在 Windows 证书库中的根证书拷贝到桌面的 FiddlerRoot.cer 文件中。该根证书文件也可以拷贝到另一个设备或放置到 Firefox 或 Opera 的受信任证书库中。

Remove Interception Certificates 按钮只有在 Decrypt HTTPS traffic 复选框没有被选中时才有效。点击该按钮，会从 Windows 证书库中删除 Fiddler 根证书以及与其关联的服务器证书。证书被删除之后，如果你再次执行 HTTPS 解析，Fiddler 会生成新的根证书并弹出对话框询问是否要信任新的根证书。

4.1.3 扩展选项

扩展选项卡中提供了控制 FiddlerScript 的选项，同时也列出了可加载的所有扩展的列表。

Automatically reload script when changed 选项可控制 Fiddler 发现文件变化时，是否自动重新加载 FiddlerScript CustomRules.js 文件。

Editor 选项框可控制使用哪个文本编辑器来编辑 FiddlerScript。点击“...”按钮会弹出可选择的编辑器列表。

References 选项框可指定 FiddlerScript 所依赖的.NET 程序库。在介绍 FiddlerScript 的一节中可以了解到更多这方面的内容。

Extensions 选项框显示了除 Inspectors 和 Transcoders 以外的所有扩展。你可以复制该文本框中的信息用于生成 bug 报表或支持查询。

Find more extensions 链接会打开 Fiddler 组件的目录页面。

4.1.4　连接选项

连接选项卡中包含了用于控制 Fiddler 代理的配置选项。

Fiddler listens on port 选项框设置 Fiddler 监听 Web 数据流的端口号。推荐使用默认端口号（8888），除非该端口已经被占用。

Copy Browser Proxy Configuration URL 的功能是拷贝代理的配置脚本（Configuration Script）的 URL，配置客户端的代理时，可以直接把这个 URL 粘贴上。该功能很少使用。

Capture FTP requests 复选框控制 Fiddler 是否注册成可处理 FTP 请求的系统代理。默认情况下，该功能没有打开。

Allow remote computers to connect 控制 Fiddler 是否处理来自其他计算机或设备的 HTTP 请求。选中该选项，Windows 8 Metro 风格的应用才能连接到 Fiddler。修改该选项后，需要重启 Fiddler 才能生效。在重启过程中，防火墙软件可能会弹出对话框，让你确任是否允许 Fiddler 接收进入的入请求。

在不可信的网络环境中，不要启用该选项。如果有黑客知道你运行着 Fiddler，就可以通过你的 Fiddler 发送他的数据流，而该数据流在其他机器看来像是来自你的计算机。

Reuse client connections 选项可以控制 Fiddler 是否重用跟客户端之间连接（客户端保持活动状态）。同样，Reuse connections to servers 选项框可以控制 Fiddler 是否会重用与服务器之间的连接（服务端保持活动状态）。出于性能考虑，应该使用连接重用选项；在解决故障时可取消。

Chain to upstream gateway proxy 选项控制当启动 Fiddler 作为上游网关代理时，是否会在 IE 内使用代理配置。如果不选中该选项框，就相当于告诉 Fiddler“忽略系统默认的代理设置，

把请求都直接发送给 Web 服务器。”

点击 Show Gateway Info 可显示所有已知上游网关代理的信息。

Act as system proxy on startup 选项控制 Fiddler 在启动时是否会把自己注册成系统代理。IE 和很多其他应用默认会使用系统代理，当该系统代理变化时，会弹出提示信息。

Monitor all connections 复选框控制 Fiddler 是否注册为处理所有 WinINET 连接的代理。如果你是通过 VPN、RAS 或拨号连接，应该选中该选项，因为 WinINET 会对所有激活的这类连接使用代理。

Use PAC Script 复选框控制 Fiddler 注册为什么模式的系统代理。一般而言，当选中时，Fiddler 会告诉浏览器使用代理配置脚本（可以通过点击 Copy Browser Proxy Configuration URL 链接得到该脚本），而不是使用默认配置 127.0.0.1:8888。该功能很少使用，只有当尝试捕获 IE8 以及更低版本的回调数据流（比如 http://localhost 或 http://127.0.0.1）时才用的到；当配置了默认的固定代理时，这种数据流不会发送到 Fiddler，但是当使用 PAC 脚本时，会发送这些数据流。

WinINET Connections 列表显示该机器的网络配置。如果你通过拨号网络或 VPN 连接到互联网，希望 Fiddler 自动连接到某个非局域网的连接时，可以使用该功能。

对于 IE should bypass Fiddler 列表，当 Fiddler 注册为系统代理时，这个列表可控制最终哪个请求通过 Fiddler 发送。注意，该列表通常只有 IE 这样的 WinINET 客户端才能够识别。默认的<-loopback>标记告诉 IE9 及更高版本应该发送回调请求给 Fiddler；如果没有该标记（对于 IE8 以及更低版本的浏览器），对于回调数据流，WinINET 会自动过滤。在选项框中可以输入域名列表，通过分号分隔，支持*这个通配符。举个例子，如果你希望 example.com 网站的所有的数据流（包括回调数据流，除了安全的数据流外）以及 fiddler2.com 的所有数据流都通过 Fiddler，可以输入如下字符串：

```
<-loopback>;https://example.com;*.fiddler2.com
```

4.1.5 外观选项

外观选项卡包含控制 Fiddler 显示的选项。

Font Size 选项框用于设置 Fiddler 中的文本大小。要应用变化后的字体大小，需要重新启动 Fiddler。

Set Readonly Color 按钮支持为只读的文本框选择背景颜色。设计该选项的原因是 Windows 默认的颜色是灰色，但是灰色和黑色间的区分度较小，不容易读取。要应用这些选

项变化，也需要重新启动 Fiddler。

Hide Fiddler when minimized 复选框支持最小化 Fiddler 时，在系统托盘中显示 Fiddler 图标，而不是在任务栏中显示。

当选中 Always show tray icon 复选框时，Fiddler 托盘中的图标会一直显示在系统托盘中。

Use SmartScroll in Session List 选项用于当 Web Session 列表中增加新的 Session 时，可以控制 Fiddler 的行为。如果没有选中该选项，Fiddler 会自动滚动到新增 Session 处（如果选中 View > AutoScroll Session List）。

如果没有选中 Use SmartScroll in Session List，Fiddler 只会在 Web Session 列表已经滚动到最下方时，才会显示最新的 Web Session。选中该选项可以避免在查看和捕获数据流时，因滚动条滚到其他地方而找不到最新的 Session 列表。

如果选中 Reset Session ID counter on CTRL+X 选项，当按下快捷键 CTRL+X 或点击菜单命令 Remove > All Sessions 清空 Web Session 列表时，Session ID 号会重新从 1 开始。如果没有选中该选项，Session ID 初始值为 1，然后会不断递增，直到重新启动 Fiddler。

4.2　HEADER 编码设置

通过网络传输文本时，客户端和服务器必须采用相同的数据转换协议，按照预定的规则对文本字符串和网络上传输的数据进行转换。HTTP 编码规范使用 header Content-Type 中的 charset 属性来标识请求和响应所使用的字符集。

遗憾的是，HTTP 本身在 HTTP 请求头中没有提供统一的机制来表示非 ASCII 码。相反，HTTP 规范中指出只能使用 ASCII 码。有些客户端和服务器遵循该协议（对非 ASCII 码进行编码），但也有些客户端和服务器会发送和接收非 ASCII 码。在某些情况下，会存在客户端应用、中间件（如 Fiddler 这样的代理）和服务器（如 Apache 或 IIS）之间会存在字节和字符之间的转换不一致的问题。

为了支持绝大多数的客户端和服务器，Fiddler 统一把 HTTP 头看作 UTF-8 编码。某些语言风格的 locale（尤其是在亚洲），使用的是不同的编码（如 EUC-CN）。可以手工配置 Fiddler，在 HKCU\Software\Microsoft\Fiddler2 中创建新的 HeaderEncoding 注册字符串，可以为请求头设置不同的默认编码。字符串名称必须是 Encoding，这样才可以被.NET 框架识别；在 http://fiddler2.com/r/?EncodingNames 中可以找到有效的字符串值。

对 HeaderEncoding 的任何修改都需要重新启动 Fiddler 才能够生效。

4.3 PREFERENCES（偏好）

在 Preferences 中，Fiddler 支持很多名称/值对设置，可以作用于 Fiddler 中的很多扩展以及其他组件。很多 Preferences 可以通过 Fiddler 用户界面中的菜单或复选框进行调整，但有些 Preferences 不支持这种修改方式，只能通过 Preferences 系统来查看或修改。

在 Web Sessions 列表下方的 QuickExec 选项框中，可以使用 PREFS 命令和 Preferences 进行交互。输入 prefs log 会把所有 Preferences 都记录到 Fiddler 的 Log 选项卡中。你可以输入 prefs set [prefname] [prefvalue]来创建或更新 Preference 值。如果取值中包含空格，可以用双引号把它括起来。输入 prefs remove [prefname]会删除 Preference。下次 Fiddler 查询该 Preference 的值时，会提示该 Preference 已经不存在，并使用该 Preference 的默认值。输入 prefs show [partialname]会显示名称中包含了所提供的字符串的 Preferences 列表（Preference 的名称是大小写不敏感的）。

在 QuickExec 选项框中输入 about:config，可以查看所有配置的 Preferences；该操作会创建或激活 Fiddler UI 界面中的 about:config 选项卡，并在其中按照字母顺序列出所有 Preferences，如图 4-1 所示。

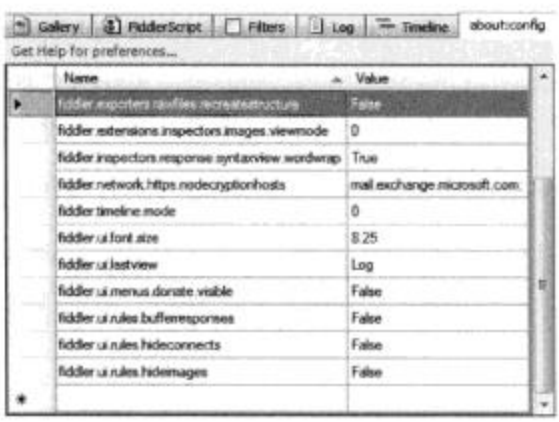

图 4-1

选中某个 Preference 所在的行并按下 Delete 键，就可以删除该 Preference。在单元格中按下 CTRL+C 可以复制该单元格中的值，或选中某一行并按下 CTRL+C，会同时复制名称及其值。可以点击 Value 列来修改某个 Preference 的值并输入新值。也可以点击最后一行（在左边有*号标识），为 Preference 增加新的行，并输入新的名称和值。

附录 D 中给出了 Fiddler 支持的所有 Preferences 以及 Fiddler 扩展。

4.4　配置客户端

作为代理服务器，Fiddler 只能看到发送给它的数据流。默认情况下，大多数应用程序的网络流会自动发送给 Fiddler，因为应用在运行时，Fiddler 会注册成默认的 Windows 系统代理。在某些情况下，如果 Fiddler 没有注册成默认的系统代理，可以对客户端应用或 Fiddler 进行配置来捕获这些数据流。作为代理服务器，Fiddler 还可以接收远程计算机和支持代理服务器的网络设备的数据流（如支持 WiFi 的平板电脑或手机）。

本节将介绍配置 Fiddler 来捕获数据流的各种方式。

4.4.1　捕获浏览器的数据流

在 Windows 上运行的 IE、Chrome 和 Safari 浏览器会自动把数据流发送给默认的系统代理，这意味着 Fiddler 可以自动捕获这些浏览器的数据流。

对 Firefox 和 Opera 浏览器进行简单配置，就可以让它们把数据流发送给 Fiddler。

Firefox

当前版本的 Fiddler 可以安装 Firefox 的一个插件——FiddlerHook，使得 Fiddler 也可以捕获 Firefox 的数据流。需要使用 Firefox's Tools>Add-ons > Extensions 来安装这个插件。当安装完成后，FiddlerHook 会在 Firefox 浏览器的 Tools 菜单中增加一个 Monitor with Fiddler 命令。

该菜单项命令支持把 Fiddler 设置成 Firefox 默认的代理，可以直接启动 Fiddler。

如果选中 Fiddler 菜单栏 Monitor 下的 Show StatusBar item 命令，如图 4-2 所示。而且选中了 Firefox 浏览器的 View>Toolbar>Add-on Bar 选项，Firefox 状态栏的最右下角会显示 FiddlerHook 的当前状态。

点击这个状态栏面板，会弹出一个菜单，从中可以修改 FiddlerHook 的设置。此外，还提供了一键清除 Firefox 的缓存和 Cookie 的菜单项，如图 4-3 所示。

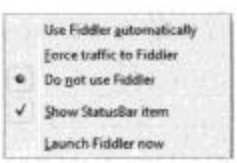

图 4-2

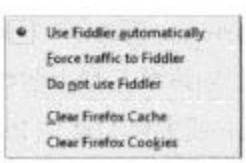

图 4-3

当启用 FiddlerHook 时，你还可以在 Firefox 的工具栏面板中增加一个按钮，可以用其启动 Fiddler。右击 Firefox 工具栏，选中 Customize，如图 4-4 所示。把 Fiddler 按钮拖到你希望的位置即可。

如果没有启用或无法使用 FiddlerHook 插件，可以重新配置 Firefox 浏览器以使用 Fiddler。

在 Firefox 4 以及更新的版本中，点击 Tools > Options 会打开选项窗口。点击最上方的 Advanced 图标，然后再点击 Network 选项卡。在 Network 选项卡中，点击 Connections 框中的 Settings 按钮。选中列表中的 Use system proxy settings 选项，如图 4-5 所示。完成这些操作后，Firefox 会使用和 Fiddler 无关的、系统默认的代理。

图 4-4

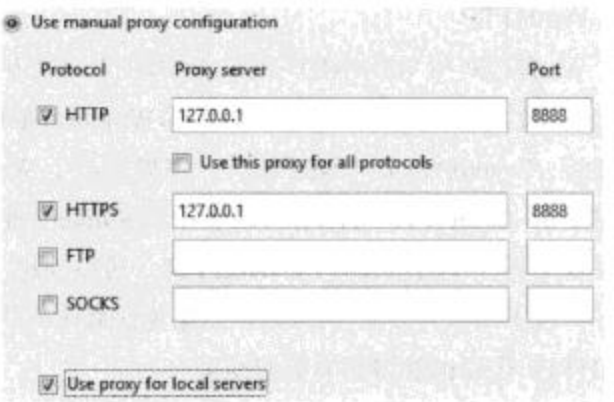

图 4-5

Opera

点击左上方的 Opera 按钮，点击 Settings，然后点击 Preferences。点击选项卡 Advanced，然后点击左侧列表中的 Network 命令。点击 Proxy Servers 按钮，把 HTTP 和 HTTPS 代理服务器字段设置成 127.0.0.1，把 Port 均设置成 8888。

其他浏览器

对于运行在 Windows 上的其他小众浏览器，大多数会自动采用默认的代理设置。对于那些没有自动采用默认代理设置的，一般而言，与 Opera 的设置步骤类似——使用浏览器提供的 UI 界面，把 HTTP 和 HTTPS 代理设置成 Fiddler 的终端，即 127.0.0.1:8888。

此外，你还可以配置很多浏览器，以使用代理配置脚本。在捕获数据流时，Fiddler 会自动生成这种配置脚本并进行更新。使用选项卡 Tools > Fiddler Options > Connections 中的链接 Copy Browser Proxy Configuration URL，把代理的配置脚本 URL 拷贝到剪贴板中，可以方便地粘贴到浏览器的代理设置中。

4.4.2　从其他应用中捕获数据流

Windows 中最常用的 HTTP / HTTPS / FTP 网络协议栈是 WinINET，IE 以及无数其他客户端应用都使用该协议栈。这些应用通常可以和 Fidder 无缝集成，因为 Fiddler 启动后，默认情况下会自动设置 WinINET 代理。

但是，Windows 中还有几种不是基于 WININET 的网络协议栈。一些应用程序，如 Firefox，有自带的网络组件。在某些情况下，这些组件会在启动时自动设置为使用 WinInet 代理，而在另外一些情况下，必须手动配置。

WinHTTP

WinHTTP 与 WININET 类似，区别在于它是专为 Windows 服务进程以及一些其他需要在后台静默行动的场景设计的。使用它的 Windows 组件包括 BITS（后台智能传输服务）、Windows Update、CryptoAPI 证书验证码等。如果在基于 WinHTTP 的应用程序启动之前，Fiddler 已经被启动了，WinHTTP 可能会自动连接到 Fiddler，连接与否主要依赖于 WinHTTP 的使用方式。如果应用程序中会直接使用 WinHTTP，可以使用 WinHttpOpen 或 WinHttpSetOption 这两个 API 将请求直接转发给 Fiddler。如果无法修改应用程序的代码，使用命令行工具修改默认的 WinHTTP 代理也能得到类似的结果。

在 Windows XP 和 Windows 2003 或更早的版本中，命令如下：

```
Proxycfg -p http= 127.0.0.1:8888;https = 127.0.0.1:8888
```

在 Windows Vista 和更高版本中，命令有所不同，如下：

```
Netsh winhttp set proxy 127.0.0.1:8888
```

如果按照上述步骤修改了 WinHttp 的默认设置，关闭 fiddler 之后，应该手动复原代理的设置，否则，基于 WinHTTP 的应用程序很可能无法连接到网络。

.NET 框架

Microsoft 的.NET 框架使用 System.Net 集实现 HTTP/HTTPS 和 FTP 协议。在某些情况下，应用启动时，基于.NET 的应用会自动采用 WinINET 代理，因此在应用启动之前先启动 Fiddler 可以确保 Fiddler 能够捕捉到应用的数据流。

否则，也可以按照如下方式设置代理，把应用代码修改为临时指向 Fiddler：

```
GlobalProxySelection.Select = new WebProxy("127.0.0.1", 8888);
```

此外，你也可以在各个 WebRequest 对象中手动指定代理：

```
objRequest = (HttpWebRequest)WebRequest.Create(url);
objRequest.Proxy= new WebProxy("127.0.0.1", 8888);
```

如果你无法访问应用的源代码，可以在应用程序的配置中指定代理。编辑（或创建）应用文件夹中的 yourappname.exe.config 文件，添加或更新以下部分：

```
<configuration>
  <system.net>
    <defaultProxy>
      <proxy
        usesystemdefault="true"
        bypassonlocal="false"
        proxyaddress="http://127.0.0.1:8888"
      />
     </defaultProxy>
    </
  system.net>
</configuration>
```

除了修改应用程序的清单外，还可以通过文件 machine.config 中的 configuration 部分调整机器配置。更新文件 machine.config 的优点在于该变化会在所有账号下的应用程序的.NET 代码中生效，包括运行在 IIS 上的 ASP.NET 所使用的服务的账号下运行的应用。

不管如何设置代理，.NET 总是可以绕过 Fiddler，直接处理指向本机的 URL（即“回路”）。调试回路数据流需要额外的配置，在本章后面会具体介绍。

Java

一些 Java 运行时环境（JRE）会自动采用 WinINET 代理。否则，如何设置默认的代理服务器取决于你使用的是 JRE 版本。有一个版本的 JRE 中包括了 Windows Control Panel 的小程序（applet），可以选择代理。使用下面这两个命令可以修改 JRE 的设置：

```
jre -DproxySet=true -DproxyHost=127.0.0.1 -DproxyPort=8888 MyApp
jre -DproxySet=true -Dhttp.proxyHost=127.0.0.1 -Dhttp.proxyPort=8888
```

如果上述两种方式都不能正常工作，可以查看 JRE 的说明文档。

PHP / CURL

在 PHP 中通过 CURL 发送网络请求时，要想将请求转发给 Fiddler 代理，可在发送 Web 请求前添加下面这行代码，其中$ch 是函数 curl_init()所返回的 handle：

```
curl_setopt($ch, CURLOPT_PROXY, '127.0.0.1:8888');
```

4.4.3　通过服务捕获数据流

Fiddler 只会注册为当前用户的系统代理。系统服务是以不同的账户运行的，包括 ASP.NET 和 IIS Worker Processes，因而这些服务在默认情况下不会向 Fiddler 发送数据。

把 Fiddler 配置为不同系统账户代理的方法不一，不同场景有不同的配置方式。举个例子，要捕捉 ASP.NET 网页的 Web 服务调用，可以编辑 ASP.NET 安装文件 web.config 或 machine.config，或修改代码，通过.NET 框架配置一节中所描述的步骤，手动指定 Web 代理。

对于使用其他网络栈的 Windows 服务（如 Java 或 WinHTTP），需要查找这些 Web 服务的配置并手工配置代理。

4.4.4　捕捉“回路”数据流

当服务器、客户端以及 Fiddler 运行在同一台机器上时，一般需要采取特殊的配置才能捕捉到客户端应用发送到服务器的流量。这种数据流称为回路（loopback）数据流，因为这些数据流一直没有离开过本机。回路数据流通常包括发送到回路地址 127.0.0.1, [::1]和域名 localhost 的数据，有时也包括发送到当前计算机的 TCP/IP 地址的数据。

对于回路数据流，有三种场景会出现问题：

1）很多客户端应用和框架通过硬编码的方式避免代理服务器处理回路数据流。

2）在 Windows Vista 以及更新的版本中，当客户端和服务器位于同一台计算机上时，HTTP 身份验证行为不一致。

3）在 Windows 8 中，Metro 风格的应用无法连接到运行在应用程序包以外的回路监听器（如 Fiddler）。

以下我们将分别探讨这些问题。

绕过回路

很多客户端应用自动绕过回路数据流代理，认为由于代理服务器的地址和客户端应用设置的地址不同，远程代理服务器无法处理流向“127.0.0.1”的数据流。然而，这些客户端应用没有考虑到代理服务器和客户端应用（如 Fiddler）运行在同一台机器上的场景，在这种情况下，代理服务器和客户端的地址是相同的。

在 IE 9 版本之前，通过 WinINET 网络栈（如 IE、Office 等使用该栈）发送的所有数据流

如果是发送到 127.0.0.1、[::1]或主机名 localhost，会自动绕过代理服务器。当前的 Opera 版本也是如此，Firefox 的 No Proxy For 默认情况下也是设置为 localhost 和 127.0.0.1。微软的.NET 框架从第一版到第四版的回路请求也会绕过代理。

如果想要使用上述某种客户端连接到运行在本机的服务器上，客户端会忽略代理设置，直接把数据流发送给服务器。

在 Fiddler 中，要捕捉这种数据流，最简单的方式是改变请求的主机名，让客户端无法识别自己是否是在向回路地址发送数据。一种简单实用的方式是使用当前计算机的 DNS 主机名（比如 http://mymachinename），但是这需要了解当前计算机的主机名，对于构建能够在多台机器上工作的测试用例会变得很复杂。

为了弥补这个缺陷，Fiddler 支持设置虚拟主机名，可为回路地址设置别名：

不要用下述方式	应该用下述方式
127.0.0.1	ipv4.fiddler
[::1]	ipv6.fiddler
localhost	localhost.fiddler

使用虚拟主机名时，客户端会把数据流发送给 Fiddler，Fiddler 在请求 URL 和 Host 头中会自动替换为正确的主机名。

在 IE9 中，WinINET 增加了新的功能，支持客户端配置 WinNET，通过代理接收回路数据流。为了配置成这种行为，需要对 Proxy Bypass List 进行配置，使其包含特殊的标记<-loopback>；Fiddler 默认会自动设置这个标记，从而可以很容易地查看发送到本地计算机的数据流。从 Windows 8 开始，WinHTTP 栈也能够识别<-loopback>标识符。期望 Opera 和.NET 框架在以后的版本中也能够支持把回路数据流发送给代理。

回路身份验证

Windows Vista 及更高版本的系统会保护用户免受回路身份验证(loopback authentication)攻击。在这种攻击中，客户端以为是和远程服务器进行身份验证，而实际上是和本机进行身份验证。这种攻击通常是要提升指定的应用程序在本机运行的优先级。当 Windows 遇到这种需要对本机进行身份验证的异常情况时，该身份验证请求会被阻塞。

为了取消回路保护，可按照 http://support.microsoft.com/kb/926642 中的描述进行设置，设置的内容如下：DisableLoopbackCheck=1。

被 Metro 风格的 Windows 8 应用阻塞的回路

Windows 8 推出了一种新的应用隔离技术，名为 AppContainer。新的“Metro 风格应用”会使用该技术，包括 Metro 风格模式的 IE，以及从 Windows 商店中下载的应用。为防止独立的 AppContainer 进程和其他进程（大部分优先级更高）通信，会通过防火墙阻止 Metro 风格的应用向运行在本机的服务器发起网络连接。不管应用什么主机名，都会被阻塞——连接 http://localhost、http://127.0.0.1 和 http://machinename 都会失败。终端用户可以在调试或开发时使用工具 CheckNetIsolation.exe 对单个进程解除这个限制。通过新版的 Visual Studio 来调试应用时，也支持临时连接到回路服务器。

默认情况下，如果你在 Windows 8 上运行 Fiddler，并启动了 Metro 风格的浏览器和应用，你会发现应用无法连接到任何网站，包括互联网上的网站。这是由于 Fiddler 运行在本机上，防火墙会阻止应用把数据流发送给它。因为其配置的代理服务器是 Fiddler，应用会认为网络连接不可用，放弃请求。

为了解决这个问题，我创建了一个图形化工具，名为 EnableLoopback，它支持对一个或多个 AppContainers 取消回路限制。Fiddler 版本 4 的用户会发现默认情况下已经安装了 EnableLoopback.exe 工具，点击 Fiddler 工具栏左侧的 Win8 Config 按钮可以启动这个工具，如图 4-6 所示。

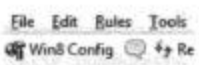

图 4-6

Fiddler 版本 2 的用户可以从 http://fiddler2.com/r/?Win8EL 下载 EnableLoopback.exe 这个工具。调整 Loopback Exemptions（回路免除）列表需要有管理员权限。当程序启动时，会弹出对话框，要求确认可以以管理员身份运行。

当启动 EnableLoopback.exe 工具后，它会在列表中列出本地所有的 AppContainers。可以分别调整每个 AppContainer，也可以使用该工具上方的按钮对所有的 AppContainers 做免除设置。修改完成后，点击 Save Changes 按钮，可以把这些修改应用到 Windows 窗口，如图 4-7 所示。

图 4-7

在免除 app 的 AppContainer 的回路限制后，就可以使用 Fiddler 调试这些 APP 的数据流了，与普通桌面应用一样。

4.4.5 在 Max OSX 上运行 Fiddler

作为 Windows 应用，Fiddler 本身无法运行在 Mac OSX 上。然而，VMWare Fusion 或 Parallels Desktop 这样的虚拟化产品使得 Fiddler 这样的 Windows 应用也可以在 Mac 的虚拟机上运行，如图 4-8 所示。

图 4-8

要在 Parallels 下运行 Fiddler，只需要对配置做的少量修改。安装 Parallels，重新配置，把 Windows 虚拟机的 Hardware > Network 1Type 设置成使用 Bridged Network 模式，如图 4-9 所示。

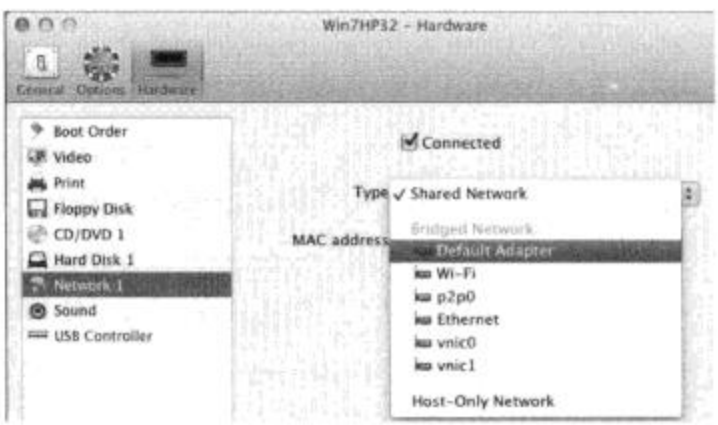

图 4-9

按照上述方式配置之后，Mac 就可以向虚拟机发送网络数据流了。重新启动虚拟机，并在虚拟机中的 Windows 操作系统上安装 Fiddler。在配置 Fiddler 时，点击 Tools > Fiddler Options > Connections 选项，选中 Allow remote computers to connect。设置完成后，需重启 Fiddler 才能生效。可能需要重新配置防火墙，使它允许向 Fiddler 进程发送连接请求。执行完以上步骤后，Fiddler 就可以接收 Mac 上发送过来的数据流了。现在，需要手动配置 Mac，把 Web 数据流发送给运行在虚拟机上的 Fiddler。

图 4-10

首先，需要了解虚拟机的 IP 地址。把鼠标悬停在 Fiddler 工具栏右侧的 Online 标识符上。工具栏提示框会显示分配给虚拟机的 IP 地址，如图 4-10 所示。

然后，点击 Apple Menu 和 System Preferences。点击 Network 图标，再点击 Advanced 按钮。点击 Proxies 选项卡。选中 Web Proxy（HTTP）选项和 Secure Web Proxy（HTTPS）选项，输入虚拟机的 Ipv4 地址；指定代理运行在 8888 端口，如图 4-11 所示。

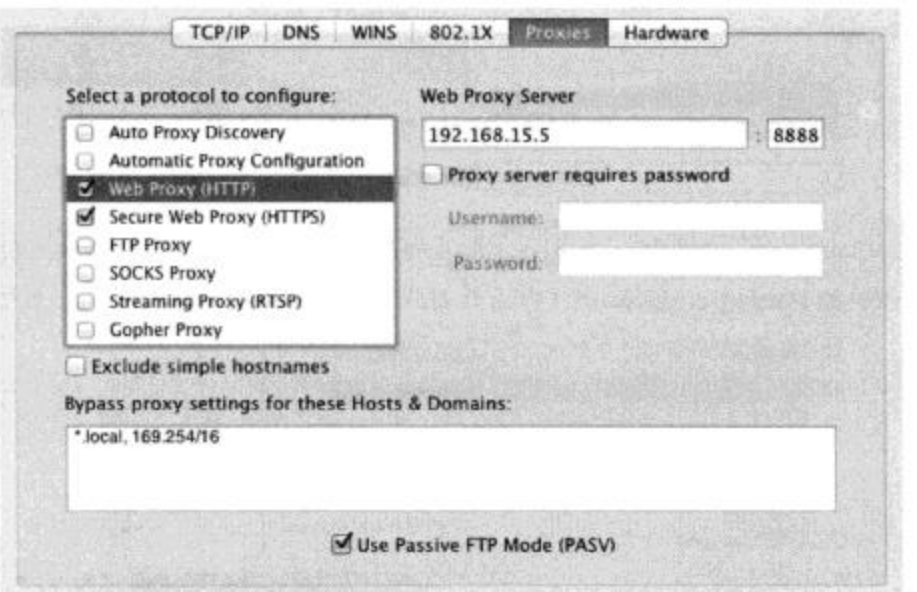

图 4-11

配置完 Mac 代理后，Fiddler 会捕捉 Safari 和其他应用的数据流。当你不再使用 Fiddler 时，应该返回到 OSX System Preferences 选项，取消代理设置。

4.4.6 从其他计算机捕捉数据流

只要应用支持代理服务器，就可以通过 Fiddler 捕捉运行在其他计算机上的这个应用的所有数据流。实际上，这就是代理服务器的工作方式。为了捕捉这种数据流，首先需要配置 Fiddler，然后配置其他计算机。

要配置 Fiddler，点击选项 Tools > Fiddler Options > Connections，选中 Allow remote computers to connect。修改完成后，重新启动 Fiddler，配置即会生效。还需要重新配置防火墙，允许与 Fiddler 进程建立网络连接并发送数据。

还需要验证客户端机器能够成功访问 Fiddler，不会被防火墙拦截：[打开浏览器，访问 http://*FiddlerMachineIP*:8888 时也不会出现问题]。如果你看到的是一个内容为“Fiddler Echo Service”的 Web 页面，说明客户端和 Fiddler 之间可以通信。

确定网络正常之后，下一步是在另一台计算机上配置代理，使用“从其他应用捕捉数据流”一节中所描述的方法来配置。其中一个区别是地址，不能再用 127.0.0.1:8888 作为代理的地址，应该使用 *IPOfTheFiddlerPC*:8888。举个例子，另一台计算机的 Internet Explorer > Tools > Internet Options > Connections > LAN Settings 的屏幕显示可能如图 4-12 所示。

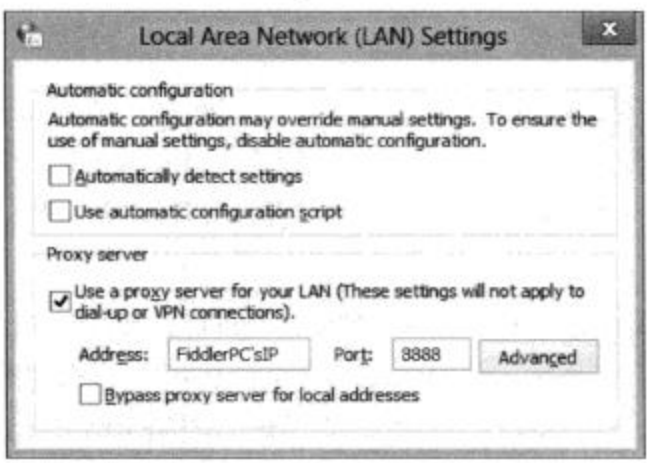

图 4-12

温馨提示：

- 当数据流是从另一台机器发送来的时，Fiddler 的根据进程名称进行过滤（如“只显示 Web 浏览器数据流”）的方式就无法正常工作，因为无法获取到远程数据流的进程名称。

- 如果你打算解析和 Fiddler 运行在不同计算机上的数据流，可以查看“HTTPS 配置”那一章，其中介绍了配置步骤。
- 当你关闭 Fiddler，停止调试时，需要清除另一台计算机上的代理设置。否则，它后续的请求都会失败，因为 Fiddler 已经不再运行了。

4.4.7　从设备捕捉数据流

很多支持 Wi-Fi 或 Ethernet 的计算设备都可以配置成把数据流发送给 Fiddler 代理。这些设备包括：iOS 设备（如 iPhone、iPad 和 iPod Touch）、很多 Android 设备以及所有 Windows Phone 和 Windows RT 设备。

在前面的章节中介绍了如何配置 Fiddler，使它支持远程计算机的数据流。下一步，最好应测试设备和 Fiddler 之间的网络连接——通过设备上的浏览器访问 http://*FiddlerMachineIP*: 8888，查看该设备的数据流是否在 Fiddler 的 Web Session 列表中。你可能需要断开设备的 3G 连接，从而确保该数据流只能是从 WiFi 上发送过来。如果在你的设备上没有看到数据流，而是显示连接错误，就需要配置 WiFi。常见的错误有如下两种：

1）WiFi Isolation

2）IPSEC

很多家庭路由器支持 Wi-Fi Isolation 功能。启动该功能后，连接 Wi-Fi 的设备无法连接直接绑定到路由器的 Ethernet 端口的设备。有些路由器还支持阻塞某个 Wi-Fi 客户端连接到另一个 Wi-Fi 客户端。如果你的路由器支持这种功能，则需要禁用这些功能，以便设备可以连接到 Fiddler 上。

出于安全考虑，有些企业网络会使用 IPSEC。IPSEC 会阻止从一个非 IPSEC 客户端到另一个 IPSEC 客户端之间的连接。绝大多数设备不支持 IPSEC，因此如果你需要从这种设备上发送数据流，必须把运行 Fiddler 的机器配置成“IPSEC Boundary Exception”，从而支持输入数据流不走 IPSEC。在很多网络中，这种特殊配置只有网络管理员能够完成。

当设备成功连接到 Fiddler 后，需要将它配置为 HTTP 和 HTTPS 数据流的代理。

Apple iOS 代理设置

要在 iPhone、iPad 或 iPod 上访问代理，点击主页的“设置”图标。在设置列表中，选中 General，然后从列表中选中 Network。点击 WiFi 并按下 Wi-Fi 网络名右侧的蓝色箭头，对其进行配置。点击 HTTP Proxy 中的 Manual 选项。在 Server 框中，输入 Fiddler 的 IP 地址或主

机名。在 Port 输入框中，提供 Fiddler 所监听的端口号。应将 Authentication 保持在 Off 状态，如图 4-13 所示。

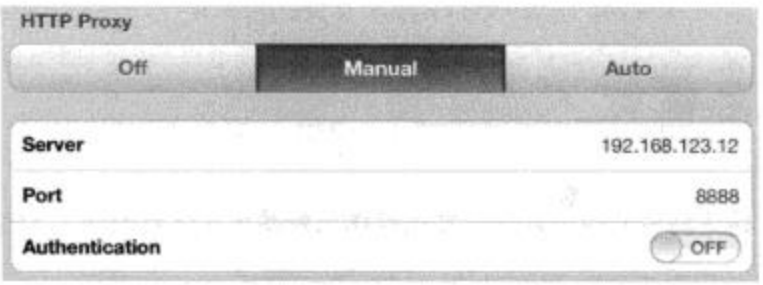

图 4-13

Windows Phone 代理设置

要在 Windows Phone 上设置代理，应打开“Setting”应用，点击 Wi-Fi 选项。按下 WiFi 网络名并点击“Edit”。把“Proxy”开关打开，在 Server/ORL 框中输入 Fiddler 的 IP 地址或主机名在 Port 框中。提供 Fiddler 所监听的端口。“Proxy authentication”保持 Off 状态。

Windows RT 代理设置

运行在 ARM 操作系统上的新版本的 Windows 8 系统，最近重命名为 Windows RT，它只能运行 Metro 风格的应用。这一特性导致 Fiddler 无法在这种设备上直接运行。然而，你可以配置 Windows RT 设备，把数据流发送到传统的 Windows 8 台式机或笔记本上。在 Windows 8 启动屏幕中，输入 Proxy，点击 Settings 选项，会显示 Configure Proxy Server 选项。点击该选项，会在桌面上弹出一个 Internet Properties 对话框。点击 LAN Settings 按钮，调整其内容如图 4-14 所示，在 Address 选项框中输入 Fiddler PC 的 IPv4 地址。

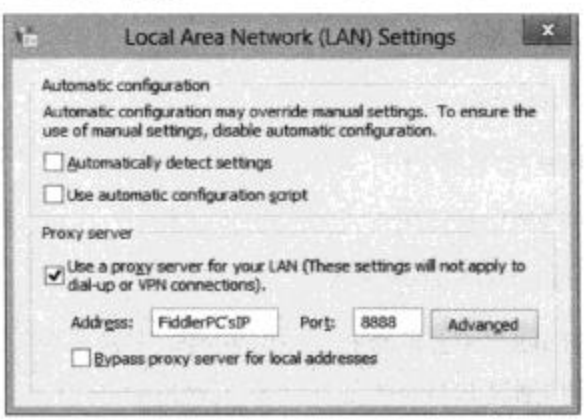

图 4-14

其他设备

绝大多数支持 Wi-Fi 的设备都能够使用 Settings 小应用程序配置代理。然而，只有极少数的设备，如当前版本的 Amazon 的 Kindle Fire，不支持配置代理。要在这些设备中使用 Fiddler，需要对设备进行“越狱”，从而能够访问某些设置选项，或者以反向代理模式使用 Fiddler，在本章的后面将介绍这些内容。

温馨提示：

- 如果没有捕获到所有的数据流，首先确保已经禁用了设备的 3G 功能，强制所有的数据流都使用 Wi-Fi 网络。
- 大多数设备需要特殊配置才能支持 Fiddler 对 HTTPS 数据流进行解密。在“HTTPS 解密”一节中将介绍配置的细节。

4.4.8　使用 Fiddler 作为反向代理

在某些情况下，你可能希望使用 Fiddler 捕捉数据流，但又无法配置客户端让其使用 Fiddler 代理服务器。为了满足这一需求，Fiddler 可以作为“反向代理”。在反向代理配置中，Fiddler 运行在服务器上，把接收到的请求转发到不同的端口甚至不同的计算机上。

举个例子，假设你的网站运行在名为 WEBSERVER 的服务器的 80 端口。使用 Kindle Fire 连接该网站，在 Kindle Fire 上你无法配置 Web 代理。你希望捕捉网站上表单的数据流，以及服务器的响应，可以如下操作：

1）在 WEBSERVER 上启动 Fiddler，运行在默认的 8888 端口。

2）点击选项 Tools > Fiddler Options，确保复选框 Allow remote computers to connect 被选中。需要的话，重新启动服务器。

3）选中 Rules > Customize Rules。

4）在 OnBeforeRequest 处理程序中，添加一行新的代码：

```
if (oSession.HostnameIs("webserver") oSession.host = "webserver:80";
```

5）在 Kindle，打开 http://webserver:8888。

以上配置完成后，在 Fiddler 中会显示来自表单的请求。该请求会从 8888 端口转发给 Web 服务器所运行的 80 端口。响应通过 Fiddler 发送回设备，而设备不知道 80 端口原始的数据是什么。

你也可以把 Fiddler 配置成反向代理，而不需要修改客户端应用的端口号。为了实现这一

目标，需要重新配置 Web 服务器软件和 Fiddler。首先，重新配置 Web 服务器，使它监听一个新的端口。例如，如果 Web 服务器当前运行在 80 端口，必须重新配置成运行在 81 端口。然后，使用 Tools > Fiddler Options > Connections 中的选项，把 Fiddler 配置成在 80 端口监听。

作为 HTTPS 的反向代理

作为反向代理运行的一个问题是客户端永远都无法知道数据流是从代理服务器发送过来的。这意味着如果客户端向 Fiddler 发送 HTTPS 请求，它不会先建立 CONNECT 连接，而是直接开始 HTTPS 握手协议。Fiddler 却期望客户端发送 HTTP 请求，因而会认为二进制的 HTTPS 握手是无效的数据流，放弃连接。

该问题可以通过为 Fiddler 创建额外的网络监听器来接收 HTTPS 连接的方式来解决。在 Session 列表下方的 WuickExec 对话框中，输入

```
!listen 444 WebServer
```

该命令会在 444 端口创建一个新的网络监听器，它期望所有接收到的请求都是从 HTTPS 握手开始。Fiddler 会作为服务器，对于监听命令的第二个参数所指定的任何主机名，返回授权证书。在以上的例子中，该证书会匹配任何发送给 https://WebServer 的请求。

在建立了安全的连接后，处理程序 OnBeforeRequest 会把接收到的安全请求转发给真正的服务器。

4.4.9 挂接到上游代理服务器

默认情况下，Fiddler 会使用当前的系统代理设置，并使用这些设置作为所有发出请求的默认的上游代理，如图 4-15 所示。

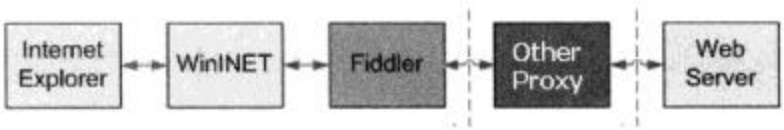

图 4-15

Fiddler 支持所有类型的代理设置，包括手工指定代理、代理脚本或自动检测（WPAD）代理。还支持所有的代理免除列表。

系统默认的代理设置（可以查看 IE 的 Tools > Internet Options > Connections > LAN Settings）默认情况下是作为上游网关使用的。即使你不是按照通常的方式使用 IE，或者使用拨号连接/VPN 连接这种有独代理设置的方式，它们的上游网关仍然是使用默认的系统代理。

如果你想阻止 Fiddler 自动挂接到系统的默认代理，可在菜单 Tools > Fiddler Options > Connections 中取消选中 Chain to upstream gateway proxy 选项。

需要的话，你还可以使用 FiddlerScript 重载所有 Session 的默认网关代理。要做到这一点，需要设置标志位 X-OverrideGateway。如果把标志位的值设置成 DIRECT，那么 Fiddler 会绕过网关，直接把请求发送给目标服务器。如果把值设置成 *address:port* 的形式，如 myproxyserver:80，Fiddler 会为该 Session 使用指定的代理。

4.4.10　挂接到 SOCKS/TOR

默认情况下，Fiddler 希望标志位为 X-OverrideGateway 的代理可以使用 CERN 代理协议，几乎所有的代理都使用该协议。然而，还存在一种不那么流行的代理标准，称为 SOCKS。在 SOCKS 协议中，客户端会向代理发送二进制头，它给出了 TCP/IP 连接应该使用的目标地址。当 SOCKS 代理确定请求的连接通道已经生成后，客户端会通过通道发送 Web 数据流。

企业虚拟网络软件（VPN）有时也使用 SOCKS 协议，SOCKS 也可以用来连接 TOR Project。TOR Project 是全球代理网络，旨在为用户提供匿名网络访问功能。TOR 网络请求在全球之间发送，期望阻止网络监听器定位请求的来源。SOCKS 协议版本 4a 还支持代理服务器执行 DNS 查询，增强了用户隐私。当通过 SOCKS v4a 连接 TOR 时，DNS 解析会在云而不是本地计算机上执行。

除了增强用户隐私，TOR 网络的另一个有趣之处在于可以像来自不同的地方那样体验网站。举个例子，在 TOR 网络中，我访问了一个广告页面，而广告是荷兰语，因为请求在 TOR 网络的出口点是阿姆斯特丹。

当设置 X-OverrideGateway 标志位时，需要使用前缀 socks=表示 Fiddler 在访问上游服务器时应该使用 SOCKS v4a 协议。例如，TOR 安装程序通过 SOCKS 代理的 8118 端口 Polipo 设置 TOR 的入口点。可以把下面代码添加到 FiddlerScript 的 OnBeforeRequest 方法中，把请求通过 TOR 网络路由到 test.example.com：

```
if (oSession.HostnameIs("test.example.com")) {
   oSession["x-OverrideGateway"] = "127.0.0.1:8111";
}
```

如果你希望通过 TOR 发送所有的 Web 数据流，可以为每个 Session 强制设置 X-OverrideGateway 标志位。

4.4.11 VPN、Modem 和网络共享

当你在 Windows 中建立 VPN、3G 共享连接或电话拨号连接时，WinINET 会让所有的请求使用连接代理。要确保 Fiddler 可以捕捉到这种数据流，需在 Tools > Fiddler Options > Connections 中选中复选框 Monitor all connections。

一些不常见的网络软件产品会在 WinINET 层以外运行。当通过这种软件连接时，Fiddler 就无法看到这些软件的 Web 数据流，因为捕捉和路由这些数据流所使用的机制超出了 WinINET 的控制范畴。

4.4.12 DirectAccess

当前的 Windows 版本支持 DirectAccess 技术（http://technet.microsoft.com/ en-us/network/dd420463），DirectAccess 支持远程访问企业网络，而不需要建立 VPN。DirectAccess 集成在 WinINET 中，因此 DirectAccess 的请求都不会被发送到默认的系统代理上。

遗憾的是，该行为意味着在使用 DirectAccess 时，Fiddler 无法捕获其数据流。在注册表中可以更新各个 DirectAccess 配置项，指定期望的目标代理服务器，但是通过这种机制调试会非常麻烦。相比之下，当在这种环境中使用 Fiddler 时，工程师通常会使用 Remote Desktop 访问运行在企业网络中的桌面 PC，可以在这台 PC 上运行 Fiddler 和客户端应用。

4.5 内存使用和 Fiddler 的“位数”

Fiddler 会把请求和响应的完整内容保存在内存中，这表示 Fiddler 在运行时可能会消耗大量内存（RAM）。操作系统的内存管理器就是用于管理应用的内存，确保即使 Fiddler 使用了大量的内存，很少访问的对象会被换出并放到位于磁盘上的页面文件中。

但是，即使有很多内存和磁盘空间，Fiddler 还是有可能显示警告信息：

```
Exception of type 'System.OutOfMemoryException' was thrown.
  at System.IO.MemoryStream.set_Capacity(Int32 value)
  at System.IO.MemoryStream.EnsureCapacity(Int32 value)
  at System.IO.MemoryStream.Write(Byte[] buffer, Int32 offset, Int32 count)
  at Fiddler.Session.Execute(Object objThreadstate)
```

该消息有时会有误导性，因为系统真的没有内存——相反地，这意味着内存管理器无法找到足够大的连续的地址空间分区来保存请求或响应。当下载大文件时（比如超过 100MB 的视频文件）经常会出现这个问题。不管有多少 RAM，32 位的进程是局限于 2GB 的地址空间。

该地址空间的每个对象会导致“碎片”，导致无法通过把可用的内存分块的方式来存储大对象，如图 4-16 所示，因为碎片太小了，无法在连续的分块保存整个响应。

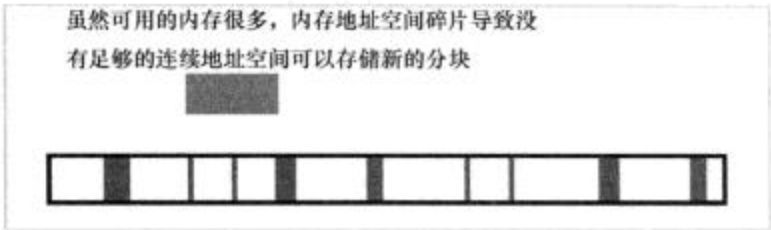

图 4-16

在 32 位的计算机上，如果在 Fiddler 的 Web Session 列表中包含数千个 Session，碎片会导致无法存储小至几 MB 的响应。可以通过周期性清空 Web Session 列表来避免这个问题。此外，通过工具栏上的 Keep Only 对话框可以自动截断 Web Session 列表，生成固定值到可用磁盘空间大小的 Session。

当 Fiddler 运行在 64 位的 Windows 版本上时，很少会遇到超出内存的错误，因为 64 位的地址空间非常大，不可能会填满它或因为碎片导致无法存储大的 Session。然而，即使在 64 位的计算机上，由于在.NET 框架的底层限制，每个请求和响应还是限制为 2GB。

在 32 位的计算机上，当下载大文件时，可以通过在 FiddlerScript 的 OnPeekAtResponse Headers 函数中添加以下代码避免内存越界的错误。以下代码片段会导致超过 5MB 的文件以流式发送到客户端，Fiddler 不会保存副本：

```
// This block enables streaming for files larger than 5mb
if (oSession.oResponse.headers.Exists("Content-Length"))
{
   var sLen = oSession.oResponse["Content-Length"];
   if (!isNaN(sLen)) {
    var iLen = parseInt(sLen);
    if (iLen > 5000000) {
      oSession.bBufferResponse = false;
      oSession["ui-color"] = "yellow";
      oSession["log-drop-response-body"] = "save memory";
    }
  }
}
```

如果是基于 FiddlerCore 构建或编写 Fiddler 扩展，可以使用类似的逻辑：

```
FiddlerApplication.ResponseHeadersAvailable += delegate(Fiddler.Session oS)
  {
```

```
    // This block enables streaming for files larger than 5mb
    if (oS.oResponse.headers.Exists("Content-Length"))
    {
      int iLen = 0;
      if (int.TryParse(oS.oResponse["Content-Length"], out iLen))
      {
        // File larger than 5mb? Don't save its content
        if (iLen > 5000000)
        {
          oS.bBufferResponse = false;
          oS["log-drop-response-body"] = "save memory";
        }
      }
    }
};
```

默认情况下，Fiddler 在 64 位版本的 Windows 上总是以 64 位模式运行。在很少情况下，你可能更希望它在 32 位模式下运行。举个例子，如果你使用的扩展依赖的本地二进制程序只有 32 位形式的模块，就必须以 32 位模式（如 Silverlight 4）运行，或者你的 FiddlerScript 依赖于其他的模块没有 64 位的版本。举个例子，Microsoft.Jet.OLEDB.4.0 数据库供应商习惯于编写 Microsoft Access .MDB 数据库文件，该文件没有 64 位形式。要强迫 Fiddler 在 64 位的 Windows 下以 32 位模式运行，可使用 Fiddler 安装文件夹下的 ForceCPU.exe 工具。

4.6 缓存和流式数据流

要支持修改请求和响应，可以对 Fiddler 进行配置，使它把请求和响应消息发送到目标机器之前，可以完全缓存它们。

4.6.1 请求缓存

当客户端连接到 Fiddler，Fiddler 会从客户端读取整个 HTTP 请求。如果设置了断点，请求会暂停，支持通过 Inspector 干预。请求继续执行后，会建立服务器连接，Fiddler 会把整个请求传递给服务器。

Fiddler 不支持任何机制以“流式”将 HTTP 请求发送到服务器，因为是从客户端读取请求——请求在发送前总是完全缓存起来。HTML5 WebSockets 是这个规则的一个特例——WebSocket 消息是双向流式发送的。

4.6.2　响应缓存

把请求发送到服务器后，Fiddler 会读取其响应。有些常见的 Web 场景（尤其是流式的音频和视频文件）是受到响应缓存的负面影响，因此 Fiddler 支持流式响应。流式响应无法使用 Fiddler 的 Inspcetor 进行修改。默认情况下，只有音频响应和视频响应配置成流式处理。你可以配置所有的响应使用 Fiddler 工具栏的触发开关配置所有的响应，或者可以通过 FiddlerScript 设置 Session 的 bBufferResponse 属性为 false，基于响应选择性地进行流式处理。

当对响应支持流式处理后，从服务器端读取的每个数据分块会马上传递给客户端应用。默认情况下，如果客户端应用关闭到 Fiddler 的连接，Fiddler 还是会读取服务端的响应，支持收集整个响应。如果你希望 Fiddler 在客户端断开连接时停止下载，只需要设置偏好 fiddler.network. streaming.AbortIfClientAborts 为 true。

对于某个响应，如果取消缓存，FiddlerScript 的 OnBeforeResponse 方法会在响应返回给客户端之后运行。这种行为（可能令人惊讶）支持不会修改响应的操作（如把响应体以日志形式写到数据库中），而对于所有 Session 都能够正常工作。如果你的代码需要确定某个给定的响应是否是流式的，它可以测试 Session 的 BitFlags：

```
bool bWasStreamed = oSession.isFlagSet(SessionFlags.ResponseStreamed));
```

如果你需要在响应体发送到客户端之前修改它，首先必须在处理程序 OnBeforeRequest 或 OnPeekAtResponseHeaders 中设置 oSession.bBufferResponse=true，取消流式处理。

4.6.3　COMET

Fiddler 的 Inspector 在从服务器端读取完响应数据后，只会显示响应体。但是，如果服务器的响应一直都不结束，会发生什么情况呢？这种响应在流式接收数据的电台中发生，在使用称为 COMET 的 Web 编程技术的站点上也会发生。

有了 COMET，服务器会使用“悬挂框架”（hanging frame）或其他机制，通过 HTTP 长连接，根据客户端需要把数据推送到客户端。HTML5 引入了一种类似的机制，称为“Server Sent Events”，它的工作机制和 COMET 相同。因为服务器的 COMET 响应都不会真正“结束”，在该连接上返回的数据对于 Fiddler Inspector 通常是不可见的，直至连接关闭。

要支持查看这样的数据，Fiddler 在 Web Session 的上下文菜单中提供了 COMETPeek 命令。当调用该命令，Fiddler 会“快照”正在执行的响应，可以从服务器查看读取的部分响应。

Fiddler 默认的对响应进行缓存的行为严重影响了依赖于 COMET 的站点。这是因为在

COMET 机制下，服务器的响应永远都不会结束，因此 Fiddler 对 COMET 数据流进行缓存通常只会导致 Web 应用终止。遗憾的是，COMET 响应没有以任何方式标记，因此如果不想全局支持流式选项，需要手动从缓存中删除这种响应。

4.7 HTML5 WEBSOCKETS

HTML5 规范引入了 WebSockets，该技术支持客户端和服务器之间的实时 socket 通信。要创建 WebSocket，首先，客户端建立到服务器的 HTTP(S)连接。其次，客户端和服务器握手，同意在该连接上为后续的数据流使用 WebSocket 协议。对于纯文本和安全的 WebSockets，分别使用 ws://和 wss:// URI schemes，虽然初始的握手协议是在 HTTP 或 HTTPS 上执行的。

要确保通过代理（如 Fiddler）发送的 WebSocket 数据流畅通无阻，首先客户端应该建立到代理的 CONNECT，请求到目标服务器的通道。如果 WebSocket URI 使用的是加密的 wss:// 协议，就执行 HTTPS 握手；如果使用的是未加密的 ws://协议，就忽略这一步。

然后，客户端使用新建立的连接发送 HTTP GET 请求，请求头 Upgrade 表示切换到 WebSocket 协议。如果服务器统一更改协议，它会发送 HTTP/101 Switching Protocols 响应。通过 Web Session 列表和 Inspector，可以很容易观察到这个过程，如图 4-17 所示。

图 4-17

握手后，客户端和服务器可以以任何顺序互相发送 WebSocket 消息。和 HTTP 请求响应模式不同，服务器会发送 WebSocket 消息给客户端，无需客户端先发送 WebSocket 消息给服务器。因为升级到 WebSocket 协议后，在连接上不再交换 HTTP 消息，在 Web Session 列表以

及请求或响应 Inspectors 中不会看到 WebSocket 消息。

但是，Fiddler 无法解析 WebSocket 消息，在以后的 Fiddler 版本中会引入新的 Inspector 类型，支持显示和修改 WebSocket 数据流。现在，客户端和服务器的 WebSocket 消息是在 Log 标签中显示的。这些消息是以纯文本形式解析和显示的。

```
Upgrading Session #47 to websocket
[WebSocket #47] Server->Client (12 bytes)
TYPE: TEXT.
MESSAGE: connected,
FLAGS: 10000001 DATA: 10 bytes.
------------------------------

[WebSocket #47] Client->Server (14 bytes)
TYPE: TEXT.
MESSAGE: timer,
FLAGS: 10000001 DATA: 6 bytes, masked using KEY: BD-36-84-D5.
------------------------------

 [WebSocket #47] Server->Client (23 bytes)
TYPE: TEXT.
MESSAGE: time,2012/4/30 14:50:38
FLAGS: 10000001 DATA: 23 bytes.
------------------------------

 [WebSocket #47] Client->Server (8 bytes)
TYPE: CLOSE.
CLOSE REASON: 1001
FLAGS: 10001000 DATA: 2 bytes, masked using KEY: 25-A2-56-AE.
------------------------------
```

防止透明代理或其他定义良好但是有 bug 的中间件带来的攻击，客户端的 WebSocket 消息通常是对消息文本根据掩码（masking key）进行异或运算得出的掩码文本，客户端运行的 JavaScript 脚本看不到该掩码。当以日志形式记录客户端的 WebSockets 消息时，Fiddler 通常会对掩码文本进行解密，以便于阅读。

由于标准的 finalize 方法和浏览器都会实现该规范，将有更多的网站会基于 WebSockets 构建。到时，Fiddler 会进行更新，从而更好地支持新协议。

4.8　Fiddler 和 HTTPS

当在 Fiddler 上访问 HTTPS 网站时，默认情况下看到的内容不多。我们看不到期望的请

求和响应列表，可以看到一个或多个 CONNECT 通道，如图 4-18 所示。

#	Result	Protocol	Host	URL	Body
1	200	HTTP	Tunnel to	www.fiddler2.com:443	0
2	200	HTTP	Tunnel to	ssl.google-analytics.com:443	0

图 4-18

HTTPS 协议是在 HTTP 请求和底层的 TCP/IP 网络连接之间添加一层加密的（SSL 或 TLS）连接，请求在该加密的连接上发送。由于使用了加密的协议，可以防止网络中间件或观察者查看或修改 HTTP 数据流。

你可能会惊讶地发现通过合理配置 Fiddler，可以查看和修改 HTTPS 数据流。Fiddler 通过“中间人”的方式实现到 HTTPS 的连接，这意味着当和客户端对话时，Fiddler 会“假装”自己是服务器；而和服务器对话时，Fiddler 会“假装”自己是客户端。

HTTPS 协议的设计就是通过使用数字证书来认证 HTTPS 服务器（也可以选择客户端），从而阻止攻击。当客户端从服务器接收到证书，它会通过挂接到客户端和操作系统都信任的 Root Certification Authority，验证证书本身是值得信任的。因为通常是在本地计算机上运行 Fiddler，可以重新配置浏览器或操作系统，使它信任 Fiddler 的根证书。所以，客户端应用检测到数据流是由 Fiddler 生成的根证书保护时，就不会弹出信息。

要在 Fiddler 中启用 HTTPS 数据流解密，可点击 Tools > Fiddler Options。在 HTTPS 选项卡上，选中 Decrypt HTTPS traffic 对话框，如图 4-19 所示。

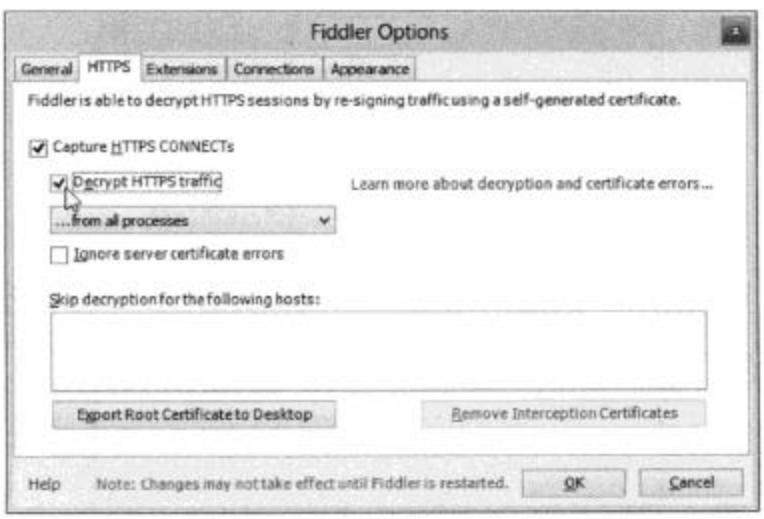

图 4-19

当你启用 HTTPS 解密，Fiddler 会生成自签名的根证书以及匹配的私有密钥。Fiddler 会为每个要访问的安全站点使用该根证书生成 HTTPS 服务器证书[也称为“终端实体”（End Entity）证书]。

信任 Fiddler 根证书

生成根证书后，Fiddler 会弹出对话框，让你选择是否将其添加到 Windows 的受信任的根证书权威库中。把根证书添加到受信任的库中可以使得浏览器和其他应用认为 Fiddler 后面生成的 HTTPS 服务器证书是有效的。它可以阻止浏览器显示警告信息，避免应用由于“信任错误”而导致连接失败。

Fiddler 和 Windows 会弹出警告信息，为不了解受信任证书含义的用户进行说明，如图 4-20 所示。

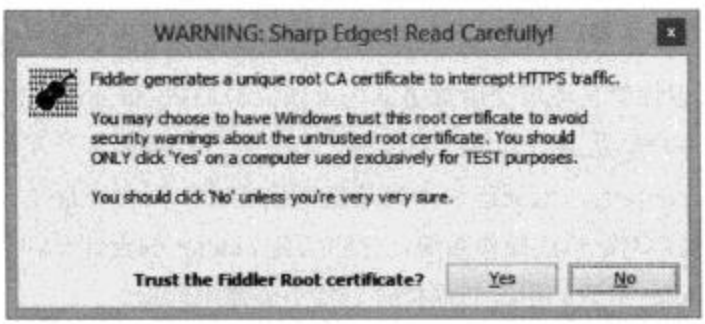

图 4-20

如果点击 Yes，Windows 会弹出对话框，要求确定以下变化，如图 4-21 所示。

图 4-21

这些警告信息故意说得很“吓人”，但是实际的风险是非常小的。每个 Fiddler 根证书都是为每台计算机唯一生成的，这样确保其他 Fiddler 用户没有相同的根证书，从而提高安全性。因此，根证书不会因为在本地计算机上运行的恶意软件而被滥用。如果你的计算机已经被恶意软件感染，那就要考虑更大的问题。

在 Windows 8 上的计算机范围的信任

在 Windows 8 上，Metro 风格的应用不会信任 Fiddler 的根证书，除非该根证书是在计算机信任的根证书库中。因此，在执行完前面几个步骤把证书添加到用户信任的根证书库中后，Fiddler 会启动管理程序，把根证书添加到计算机库中，如图 4-22 所示。

图 4-22

如果点击 Yes 启动该工具，将确定操作，如图 4-23 所示。

图 4-23

点击一系列的弹出对话框后，会安装 Fiddler 的根证书。当 Fiddler 解密数据流时，依赖 Windows 证书库的应用不会再显示安全错误。

要删除所有 Fiddler 生成的证书，应取消选中 Decrypt HTTPS traffic 复选框，然后按下 Remove Interception Certificates 按钮。

手动信任 Fiddler 根证书

如果你希望手动信任 Fiddler 根证书，可启动 certmgr.msc，并把信任证书从 Personal 文件夹拖拽到 Trusted Root Certification Authorities 文件夹。如果希望基于整个计算机做这个修改，执行以下步骤：

1）在 Personal 文件夹中右击 DO_NOT_TRUST_FIDDLERROOT 证书，选择 All Tasks > Export。

2）把根证书以 DER Encoded X509 Binary 形式导出成桌面 FiddlerRoot.cer 文件。

3）启动 mmc.exe。

4）点击 File > Add/Remove Snap-In。

5）选中 Certificates 并按下 Add 键。

6）当弹出对话框 This snap-in will always manage certificates for，选中 Computer Account。

7）点击 Local Computer，然后点击 Finish，再点击 OK。

8）打开节点 Certificates (Local Computer)。

9）右击文件夹 Trusted Root Certificate Authorities，选择 All Tasks > Import。

10）选中在第 2 步中导出的文件，并导入它。

其他 HTTPS 选项

在关闭 Fiddler Options 窗口之前，考虑使用下拉条配置使用哪个进程解析数据流，如图 4-24 所示。

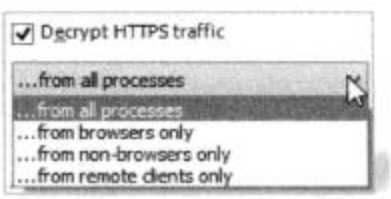

图 4-24

如果你只计划解析浏览器数据流，那么选中该选项，避免解析其他不感兴趣的应用数据流。除了保存 CPU 循环和内存，选中该选项可以防止不是以标准方式使用 HTTPS 的应用所引发的问题（如 Outlook 使用 RPC-over-HTTPS 通道连接 Exchange 服务器）以及不关心 Fiddler 在 Windows 信任库（如 Dropbox）中的根证书所引发的问题。

你还可以使用文本框列出不需要解密的 HTTPS 数据流的服务器。举个例子，我通过以下设置防止对 Outlook Web Access 和 Dropbox 数据流进行解密，如图 4-25 所示。

通过分号可以解析列表域名，使用*作为通配符。

图 4-25

4.9 为 HTTPS 解密配置客户端

虽然绝大多数应用（IE、Microsoft Office、Chrome、Safari 等）使用 Windows 证书库来验证证书链，但一些应用会维护自己的证书库。比如，Java 运行时环境通常有自己的证书库，而 Firefox 和 Opera 浏览器又分别维护自己的证书列表。

要配置这样的客户端信任 Fiddler 生成的证书，首先必须获取 Fiddler 的根证书作为.CER 文件。可以有两种方式实现这一目标：

1）点击 Fiddler Options 窗口的 HTTPS 选项卡的 Export Root Certificate to Desktop 按钮。

2）在浏览器中访问 http://127.0.0.1:8888/，点击链接 FiddlerRoot Certificate，把证书下载为.CER 文件。

在有了根证书文件后，便可以将其加入到应用的信任证书列表中。

4.9.1 浏览器

Firefox

在 Firefox 中，点击 Tools > Options。点击 Advanced 按钮，然后切换到 Encryption 选项卡。点击 View Certificates 可以打开 Certificate Manager。点击选项卡 Authorities，并点击 Import 按钮。选中文件 FiddlerRoot.cer，点击 Open。选中复选框 Trust this CA to identify websites，按下 Ok。Firefox 会信任 Fiddler 生成的 HTTPS 服务器证书。

Opera

在 Opera 浏览器中，点击 Opera > Settings > Preferences。点击 Advanced 选项卡，点击列表中的 Security。点击 Manage Certificates。点击 Authorities 选项卡，点击 Import...按钮。选中 FiddlerRoot.cer 文件，点击 Open。点击 Install 按钮，点击 Ok 按钮，确定你信任该证书。Opera 会信任 Fiddler 生成的 HTTPS 服务器证书。

跨计算机的场景

如果配置 Fiddler 作为其他计算机的 HTTPS 数据流代理，必须手动配置其他计算机，使它信任 Fiddler 服务器的根证书。可以从 http://*FiddlerMachineName*:8888/页面下载根证书，然后手动配置 certmgr.msc 文件，使它信任根证书。

重要提示

每个 Fiddler 根证书对每台计算机是唯一的。如果客户端 PC 之前已经生成了 Fiddler 根证书，该根证书会在另一台计算机上通过 Fiddler 干扰代理数据流。因为不同计算机上生成的根证书不匹配（其主题内容相同，但是公有密钥不同）。这种不匹配性在建立 HTTPS 连接时会带来致命错误——即使客户端已经被配置为信任根证书。要避免这个问题，必须删除已有的 Fiddler 根证书。

4.9.2　HTTPS 和设备

Windows 手机

为了使 Windows Phone 设备信任 FiddlerRoot 证书，在 Windows 手机上必须安装该证书。要实现这一点，应先从设备打开 http://*FiddlerMachineName*:8888/，下载根证书。打开 FiddlerRoot.cer 文件，然后在弹出的 Install certificate?屏幕上，点击 Install 按钮。

Android and iOS

Fiddler 默认的证书生成器（Certificate Maker）是基于命令行工具 makecert.exe。几乎所有的 Windows 客户端都接受该工具生成的证书，大部分其他平台也接受这类证书。但是，Apple iOS 设备（包括 iPad、iPhone 和 iPod）都要求根证书和服务器证书包含 makecert.exe-生成的证书中所没有的其他元数据。一些 Android 发布也有类似的需求。

要使 Fiddler 生成的证书和这些设备兼容，应从以下站点下载 Fiddler 插件 Certificate Maker：

```
http://fiddler2.com/r/?FiddlerCertMaker
```

该插件需要的运行环境为 Windows Vista 或以上的版本。它会把 Fiddler 中默认的证书生成代码替换成基于开源的 Bouncy Castle 图形化库的版本。默认情况下，插件会生成 iOS 兼容的证书，并且考虑一些 Preferences，以支持广泛的平台兼容。

举个例子，至少一种 Android 平台要求服务器证书在 EKU 上不应该有 Critical 约束。可以配置证书生成器插件，通过 QuickExec 设置偏好，从而删除约束：

```
prefs set fiddler.certmaker.bc.ee.criticaleku false
```

该插件生成的服务器证书并没有放在 Windows 证书库中，而只是保存在内存中。每次 Fiddler 重新启动，就会生成新的服务器证书。而 makecert.exe 工具所生成的证书则分别添加到个人证书库中，只有当按下 Remove Interception Certificates 按钮后才会删除。

配置了 Fiddler 生成和设备兼容的证书后，就应该配置设备信任 Fiddler 的根证书。要实现这一点，应该按照本书前面章节所描述的方法配置设备，使它能够支持 Fiddler 作为代理。然后，在设备上打开 http://ipv4.fiddler:8888/，并下载根证书。打开文件 FiddlerRoot.cer，如图 4-26 所示。

按下 Install 按钮。你会看到警告信息，可以按下 Install 按钮进行确认，如图 4-27 所示。

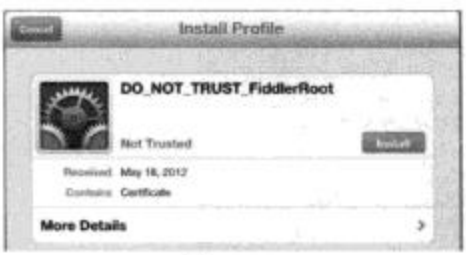

图 4-26

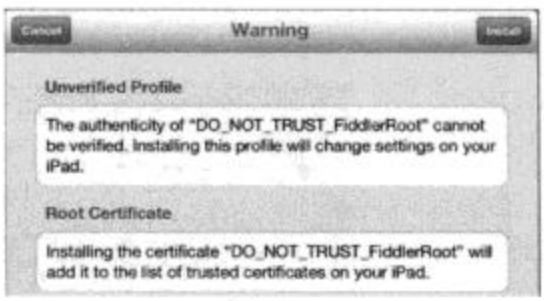

图 4-27

安装了 Fiddler 的根证书后，当 Fiddler 解析数据流时，设备浏览器和应用不会再弹出根证书错误的信息。

如果你希望从设备上卸载根证书，可打开 Settings 应用，点击 General，然后滚动鼠标到最下方的 Profiles。选中 DO_NOT_TRUST_FiddlerRoot，并按下 Remove 按钮。

4.9.3 HTTPS 服务器有 Bug

有少数 HTTPS 服务器没有正确实现 TLS 握手协议，当 Fiddler 和这些服务器握手时会失败。当 Web 浏览器和服务器连接时，会出现同样的连接失败，但是浏览器可以自动回退到使用老版本的握手协议，默默地处理失败问题。

Fiddler 可以配置成为服务器提供特定的协议版本，从而适应有 bug 的实现。在 FiddlerScript 中，在 Main()函数中可以添加如下代码，限制 Fiddler 在建立 HTTPS 连接时只提供 SSLv3。

```
CONFIG.oAcceptedServerHTTPSProtocols =
    System.Security.Authentication.SslProtocols.Ssl3;
```

此外，可以限制协议基于每一个请求。在 OnBeforeRequest 函数中，添加如下代码：

```
if (oSession.HTTPMethodIs("CONNECT") &&
    oSession.HostnameIs("buggy.example.com"))
{
   oSession["x-OverrideSslProtocols"] = "ssl3.0";
}
```

通过只提供 SSLv3，你可以调整那些没有正确支持扩展或 TLS 协议其他功能的服务器。

4.9.4　证书生效

当 Fiddler 连接到 HTTPS 服务器时，它会验证服务器证书，确保证书是有效的，而且包含目标站点的主机名。默认情况下，会展示所有的证书错误供你决定如何继续，如图 4-28 所示。

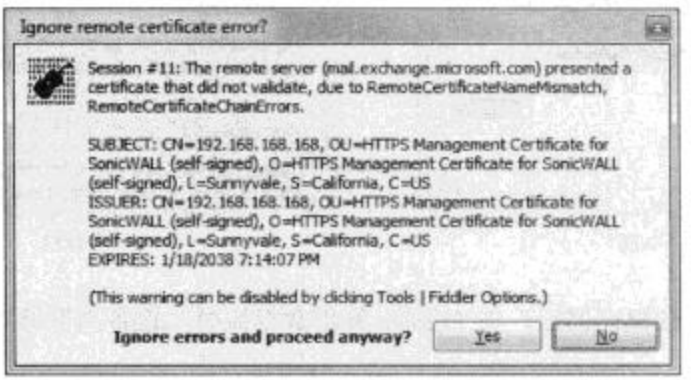

图 4-28

如果选中 No，该连接就会被丢弃。如果选择 Yes，证书错误会被忽略，连接会正常使用。Fiddler 会缓存你做出的决定，直到重新启动。因为 Fiddler 会生成它自己的证书，使得来自客户端的连接变安全，客户端应用不会意识到服务器证书是无效的。

4.9.5　Certificate Pinning

有少数 HTTPS 客户端应用支持“Certificate Pinning”功能，该功能是指对客户端应用进行硬编码，从而只接收指定的证书。即使操作系统完全信任连接使用的证书所挂接到的根服务器，这种应用会拒绝接收不是自己期望的证书。

比如，一些 Twitter 和 Dropbox 应用包含该功能，Windows 8 Metro 应用可以选择性地要求特定证书，而不是依赖系统的受信任根证书库。

当 Certificate-Pinned 应用通过 CONNECT 通道和 Fiddler 进行 HTTPS 握手时，它会检查响应的证书，并且在发现 Fiddler 生成的证书时，会拒绝发送其他请求。

遗憾的是，没有通用的方案解决这个问题，你能够做到的最多是通过在 CONNECT 通道上设置 x-no-decrypt Session 标志位，免除对应用的数据流进行解密。该标志位会阻止 Fiddler 解密通道中的数据流，因而流向 Fiddler 的数据流是未加密的。

4.10 Fiddler 和 FTP

与 HTTP 类似，文件传输协议（FTP）是基于 TCP/IP 的协议，用于传输文件，但是它比 HTTP 协议出现得早，并且和 HTTP 协议不兼容。但是，协议依然是个例外，Fiddler 只能查看基于 HTTP 的协议。这是因为当客户端浏览器或应用是配置成通过 CERN 形式的代理（如 Fiddler）来代理 FTP 数据流时，客户端在发送给代理之前会把 FTP 下载请求转化成 HTTP GET 请求。

客户端的期望是代理会作为网关，把每个 FTP 到 HTTP 的请求转化成 FTP 请求，该请求会通过 FTP 协议被发送到原始服务器。只有 HTTP 请求的 URL 和 Authorization 头是用于 FTP 请求，其他请求头都会被忽略。

Fiddler 可以通过三种方式处理 FTP 数据流：

1）作为 HTTP 到 FTP 的网关。

2）通过挂接到上游 CERN 形式的代理服务器（如 Microsoft ISA），该服务器会作为 HTTP 到 FTP 的网关。

3）通过 Fiddler AutoResponder（或其他功能）来响应请求。

要配置 Fiddler 注册为系统的 FTP 代理，点击 Tools > Fiddler Options。在 Connections 选项卡里，选中 Capture FTP requests 选项，再重启 Fiddler。

如果 Fiddler 接收到的 FTP 请求没有发送到上游网关代理，而且 AutoResponder 规则没有处理该请求，Fiddler 会尝试使用 FTP 协议连接到服务器。Fiddler 会重新格式化 FTP 服务器的响应作为 HTTP 响应，并返回给客户端。当作为 HTTP 到 FTP 的网关时，会存在一些局限性。例如，不会设置 Session 的 Timers 对象。Fiddler 不支持 FTP 流式响应——完整的 FTP 响应在转换之前会被缓存，并返回给客户端。

4.11　Fiddler 和 Web 认证

绝大多数公开的 Web 站点使用 HTML 表单进行认证——会弹出对话框，要求客户输入用户名和密码。如果提供的证书有效，客户端会得到登录 Cookie，该 Cookie 会发送给所有后续的请求。Fiddler 可以很容易查看和提交这种 Cookie。

但是，HTTP 和 HTTPS 提供两种原生机制进行客户端认证：

- HTTP 身份认证
- HTTPS 客户端证书

Fiddler 可以使用以上任何一种认证方式与服务器通信。

4.11.1　HTTP 身份认证

使用 HTTP 身份认证，客户端会发送 HTTP 请求，服务器响应包含需要证书的 HTTP/401 或 HTTP/407。HTTP/401s 是由需要身份认证的服务器发送的，而 HTTP/407s 是由代理服务器发送的。

HTTP 通常有四种身份认证机制：

- Basic
- Digest
- NTLM
- Negotiate

在 Basic 认证（RFC 2617）机制中，客户端在 Authorization 请求头中提供用户名和密码。客户端证书通常基于 64 位编码，利用 Auth Inspector 可以轻松解码。Basic 认证机制是原始的，显然在未加密的 HTTP 连接中使用是不安全的。

在 Digest 认证机制中（它也在 RFC 2617 中描述），服务器会通过 WWW-Authenticate 响应头向客户端提出质询。客户端会结合质询数据和用户密码计算消息摘要，向服务器证明自己知道密码。该认证机制比 Basic 强大，但是由于服务器端对该机制支持有限以及各种客户端 bug，该机制很少见。

NTLM 认证机制很少在公共 Internet 中使用，但是在基于 Windows 的 Intranet 中却普遍使

用。当服务器向客户端发送 HTTP/401 响应，客户端会重新发送请求，表示其支持 NTLM 认证，然后服务器提出质询。客户端通过该质询和证书生成对该质询的回复。如果质询回答正确，服务器就返回请求的资源。该模式意味着当产生 NTLM 认证时，客户端会在资源完全以 HTTP/200 返回之前，接收两个 HTTP/401 质询。

与 Basic 和 Digest 认证不同，NTLM 机制往往不是基于每一个请求。相反地，NTLM 机制是基于每一个连接的，表示会使用某个认证证据在相同的连接上认证所有后续的请求。这是你经常会看到在第一次加载内网站点时会产生一系列 HTTP/401s，而在后期的页面加载中却看不到任何后续认证质询的原因。

Negotiate 认证机制（有时称为“集成的 Windows 认证”）是“封装式”协议，其底层使用 NTLM 协议或 Kerberos 协议。在 HTTP 层，Negotiate 与 NTLM 很类似，每次请求需要 1 到 3 次循环。

当在代理服务器上使用NTLM或Negotiate机制时，客户端应用要求该代理添加Proxy-Support: Session-Based-Authentication 响应头。该响应头表示代理理解不同的客户端之间必须共享认证的连接。Fiddler 会自动添加该响应头。

4.11.2 Fiddler 中的自动身份认证

Fiddler 可以自动对使用 Digest、NTLM 和 Negotiate 协议的服务器进行身份认证。当Fiddler自动进行身份认证时，发送请求的客户端无法看到中间生成的HTTP/401或HTTP/407响应。这是因为 Fiddler 本身会处理这些响应，回答服务器的质询，只有当认证完成后才向客户端返回数据。

要使 Fiddler 自动响应服务器或网关代理的认证质询，应设置 Session 的 X-AutoAuth 属性为纯文本证书字符串。如果使用（默认）值，Fiddler 会使用运行 Fiddler 的 Windows 登录用户账户的证书：

```
static function OnBeforeRequest(oSession: Session)
{
   // To use the current Fiddler user's credentials:
   if (oSession.HostnameIs("ServerThatDemandsCreds")) {
      oSession["x-AutoAuth"] = "(default)";
   }

   // or, to use explicit credentials...
   if (oSession.HostnameIs("ServerUsingChannelBinding")) {
      oSession["x-AutoAuth"] = "redmond\\ericlaw:MyP@$$w0rd";
```

```
}
//
```

如果 Fiddler 是配置成从其他设备或用户账户中接收请求，使用默认值会造成安全隐患。因为这些请求会通过运行 Fiddler 的账号的证书来进行身份认证。

如果在 Options 子选项卡中选中了 Automatically Authenticate（自动身份认证）对话框，Composer 选项卡会自动把 x-AutoAuth 标志位的值设置成 Preference fiddler.composer.AutoAuthCreds 的值。在需要 HTTP Basic 认证的场景下，只需要产生需要的 username:password 字符串，使用 TextWizard 进行 base64 编码。使用 base64 编码的证书字符串，把它添加到 Authorization 或 Proxy-Authorization 请求头。请求头可以使用 Filters 选项卡、FiddlerScript 选项卡或 Composer 选项卡的 Request Headers 对话框添加对外的请求。

4.11.3　身份认证问题

因为 Fiddler 作为代理并提供强大的 HTTPS 解密功能，所以 Fiddler 在运行时有时会影响认证。认证时常见的三个问题是 Channel-Binding Tokens、WinHTTP 证书发布和 Loopback 认证保护。以下将逐一描述。

Channel-Binding

NTLM 身份认证机制的缺点之一在于会遭受“直通（passthrough）”攻击。在这种攻击中，客户端会被引导到恶意网站，该网站借用不知情的第三方服务器提出认证质询。客户端如实发送对认证质询的响应，该恶意网站就使用客户端的质询响应发送给被攻击的第三方服务器进行身份认证。通过攻击，攻击者可以使用客户端证书从被攻击的服务器窃取数据。

要解决这个问题，提出了 Channel-Binding 的概念。Channel-Binding 把认证质询响应绑定到底层连接，从而不能在另一个连接上重用质询响应。通常，把证书绑定到当前 HTTPS 连接的方式可以正常工作。Channel-Binding 是通过 IIS 的 Extended Protection 选项实现的。

Channel-Binding 给 Fiddler 带来了问题，因为客户端把证书通过 Fiddler 绑定到客户端，因此当 Fiddler 把这些证书转发给真实的服务器时，服务器会拒绝。要解决这个问题，可以配置 Fiddler 本身和服务器身份认证，不需要客户端参与。因为 Fiddler 自己生成质询响应，使用 Channel-Binding 信息和服务器自己的连接匹配，服务器会接收该证书。要配置 Fiddler 为客户端完成任何。使用 4.11.2 节所描述的 X-AutoAuth 标志位来完成。

WinHTTP 证书发布策略

WinHTTP 网络栈不支持安全区的概念，这表示当涉及 Fiddler 这样的代理时，它有时会拒绝响应认证质询。当使用 Microsoft Office 客户端应用从 SharePoint 站点下载文档时，可能会发生这种问题。可以配置 Fiddler 在客户端通过 X-AutoAuth 标志位认证来解决这个问题。此外，查看 http://support.microsoft.com/kb/956943 可以了解更多如何在注册表中修改 AuthForwardServerList 的信息。

回路保护（Loopback Protection）

Windows 还会保护用户避免“回路认证”攻击。在这种攻击中，客户端认为它是和远程服务器进行认证，但实际上是和本地计算机认证。这种攻击通常是为了提升低级别进程的运行权限，使其成为高级别的本地进程。当 Windows 遇到未期望的和本地计算机认证，会阻塞该请求。

要取消回路保护，可以参考 http://support.microsoft.com/kb/926642 中描述的，设置标志位 DisableLoopbackCheck=1。

4.11.4 HTTPS 客户端证书

除了基于 HTML 表单和 HTTP 认证协议，在高安全的网站中可以使用第三种认证类型。HTTPS 客户端证书是非常严格的认证表单，用于对安全性要求特别高的场合（如银行和文档签名）。当使用客户端证书认证时，客户端为服务器提供客户端证书，通过密码加密保证用户的身份。

客户端证书认证的关键目的之一是要阻止网络中间件（如 Fiddler）滥用客户端证书。即使客户端应用把证书发送给 Fiddler，Fiddler 也无法成功地重用该证书，响应服务器的需求，因为客户端永远都不会为 Fiddler 提供私钥。

要解决这个缺陷，当和服务器握手时，可以直接提供任何客户端证书和私钥给 Fiddler。默认情况下，如果服务器弹出对话框，要求客户端提供证书，Fiddler 会查看%USERPROFILE%\Documents\Fiddler2\目录下的 ClientCertificate.cer 文件，使用该证书来响应服务器的认证要求。

在某些情况下，你可以使用不同的客户端证书进行安全连接。要实现这一点，在连接到需要安全认证的服务器的 CONNECT 通道上使用 https-Client-Certificate 属性来指定证书所在的路径。例如，你可以写如下代码：

```
static function OnBeforeRequest(oSession: Session)
{
```

```
    if (oSession.HTTPMethodIs("CONNECT")) {
     if (oSession.HostnameIs("exampleA")) {
       oSession["https-Client-Certificate"] = "C:\\certs\\CertA.cer";
     }
     else
     if (oSession.HostnameIs("exampleB")) {
       oSession["https-Client-Certificate"] = "C:\\test\\CertB.cer";
     }
    }
//...
```

.CER 文件不包含和证书的公钥相关的私钥。相反，.CER 文件只是作为 Windows 个人证书库（certmgr.msc）的引用。Windows 证书库保存了和证书相关的私钥以及需要的发布版本。当插入 Smartcard 时，在 Smartcard 上保存的客户端证书会自动出现在个人证书库中，如图 4-29 所示。

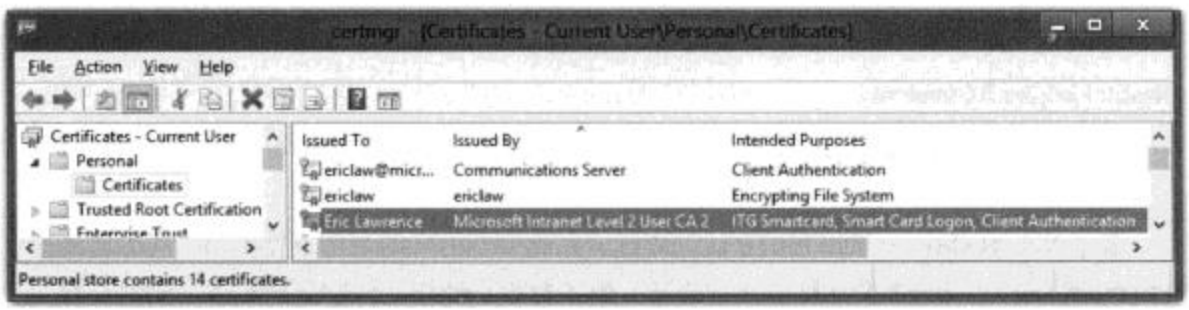

图 4-29

当插入 Smartcard 时，可以从 certmgr.msc 中导出.CER 文件，就像其他客户端证书那样使用它。注意，Smartcard 必须保留，以便 Fiddler 可以使用它获取私钥。

如果想要的证书还没有安装 Windows 个人证书库（比如只有.pfx 文件），你必须先把它导入到证书库中，然后导出成.CER 文件。安装了证书后，只需要右击证书，选择 All Tasks > Export...，把.CER 文件保存到默认的 ClientCertificate.cer 路径或保存到 https-Client-Certificate 标志位所指定的路径。

第5章 Inspectors

5.1 概览

Fiddler 的 Inspectors 用于显示 Web Session 列表中选定 session 的请求和响应。Inspectors 可以出现在两个地方：一个是 Fiddler 主窗口的 Inspectors tab 选项卡，另一个是独立的 Inspect Session 窗口。Inspect Session 窗口需要在 Web Session 的上下文菜单中输入命令才能打开。

在 Inspectors 选项卡中，Request Inspectors 在面板顶部，Response Inspectors 在面板底部。点击 Inspector 名称（如 HexView）可以对 Inspector 进行切换，如图 5-1 所示。

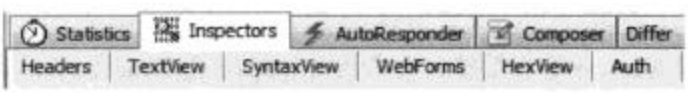

图 5-1

第一次在 Web Session 列表中双击某个 Session 或按下 Enter 键，就会激活 Inspectors 选项卡。系统会对每个请求和响应的 inspector 进行轮询，确认选中的请求和响应是否合适。举个例子，ImageView Inspector 会对 image/*类型返回“高分”，而对文本类型返回“低分”。相反地，TextView Inspector 对文本类型返回高分，而对二进制类型返回低分。

对于某个 Session，返回最高分的 inspectors 会被激活。要强制 Fiddler 总是激活某个 Request Inspector，可以设置 fiddler.ui.inspectors.request.alwaysuse 为该 Inspector 选项卡的标题。要强制 Fiddler 总是激活某个 Response Inspector，则需要设置 fiddler.ui.inspectors.response.alwaysuse 的取值为该 Inspector 选项卡的标题。

面板上方和下方之间使用一条很细的蓝色分割线进行分割；可以使用鼠标移动这条线并调整面板的大小。双击蓝色分割线会最大化 Response Inspectors 面板，因为这是大多数用户经常使用的面板，如图 5-2 所示。

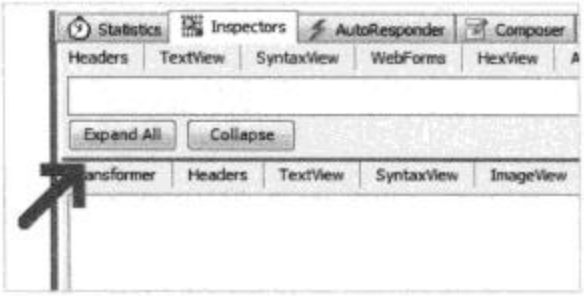

图 5-2

为了扩大显示区域，可以让 Inspector 选项卡从主窗口中独立出来，可以点击工具栏上的 Tearoff 按钮，也可以在 QuickExec 对话框中键入 tearoff 命令实现这一目的。关闭 Inspector 窗口，就会返回主窗口的默认位置。

右击单个请求和响应的 Inspector 选项卡会显示包含两个选项的菜单：Inspector Properties 和 Hide Inspector。第一个菜单显示 Inspector 提供的自身信息，而第二个菜单会从 Inspectors 选项卡中删除这个 Inspector。为了在后期恢复隐藏的 Inspector，可以编辑 fiddler.inspectors.hidelist preference，并重新启动，或者在启动 Fiddler 时按下 SHIFT 键。

默认情况下，Inspectors 是以只读（ReadOnly）模式分析某个 session，除非该 Session 当前停止在某个断点处。除此以外，只要 Session 处于未锁定（Unlocked）状态，就可以在 Edit 菜单中输入命令对该 Session 进行编辑。大多数 Inspector 在 ReadOnly 模式下会以特定的背景色来显示，当允许编辑时，会显示另一种颜色（通常是白色）。默认的 ReadOnly 颜色是淡蓝色，可以通过 Tools > Fiddler Options > Appearance 选项卡对其进行修改。

本章的最后将介绍 Fiddler 中的所有 Inspector。

5.2　授权和认证（AUTH）

类型	请求&响应
允许编辑	否

Fiddler 的 Auth Inspector 会解释请求头和响应头的授权（Authorization）和认证（Authentication）相关的内容。一般而言，服务器返回 HTTP/401 和 HTTP/407 响应头时，说明它们需要认证证书，后续的请求需要提供这些证书。

举个例子，假设你选中 Rule 菜单中的“Require Proxy Authentication”规则，Fiddler 会返回

HTTP/407 响应，要求用户为每个请求提供认证证书。该命令是通过请求头 Proxy-Authenticate 提供的，Auth Response Inspector 的显示如图 5-3 所示。

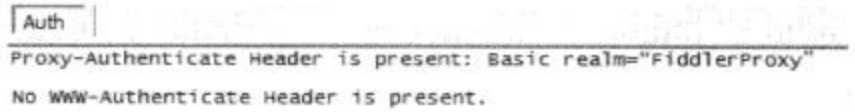

图 5-3

用户在浏览器的 Authentication 对话框中输入用户名 *UserName* 和密码 *SecretKey* 后，浏览器的后续请求中会提供包含了用户认证证书的 Proxy-Authorization 请求头。因为这个场景所使用的认证机制是 HTTP Basic，认证证书是以 base64 编码的。Auth Request Inspector 会自动对字符串进行解码，并以纯文本形式显示，如图 5-4 所示。

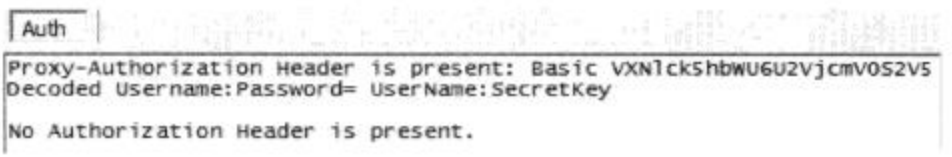

图 5-4

当然，大多数 Web 站点会使用更严格的认证方式，如使用 HTTP Digest、Windows NTLM 或 Negotatiate 机制。在 Windows 网络中，Negotatiate 认证机制被普遍使用。

Fiddler 的 Auth inspector 内置了解析 NTLM 数据块的机制，会用下面的方式来显示数据块中所包含的信息：

```
    -[NTLM Type3: Authentication]-----------------------------
    Provider: NTLMSSP
    Type: 3
    OS Version: 6.2:8329
    Flags: 0xa2888205
        Unicode supported in security buffer.
        Request server's authentication realm included in Type2 reply.
        NTLM authentication. Negotiate Always Sign. Negotiate NTLM2 Key.
    Target Information block provided for use in calculation of the NTLMv2 response.
    Supports 56-bit encryption. Supports 128-bit encryption.
lmresp_Offset: 134; lmresp_Length: 24; lmresp_Length2: 24
ntresp_Offset: 158; ntresp_Length: 396; ntresp_Length2: 396
Domain_Offset: 88; Domain_Length: 14; Domain_Length2: 14
User_Offset: 102; User_Length: 14; User_Length2: 14
Host_Offset: 116; Host_Length: 18; Host_Length2: 18
msg_len: 554
Domain: REDMOND
```

```
User: ericlaw
Host: ERICLAWT8
lm_resp: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
nt_resp: 42 40 AE AB 3E D7 7F 02 B5 F9 76 D7 C9 5E 6B 82 01 01 00 00 00 00 00 00
         05 96 B2 D4 01 23 CD C1 D3 D9 8A 33 47 87 40 3D 00 00 00 00 02 00 52 ...
```

Auth Inspector 当前无法解析 Kerberos 认证消息，但是可以显示解码后的质询和响应内容。

5.3 缓存（CACHING）

类型	只响应
允许编辑	否

Caching Response Inspector 会查看 HTTP 响应头，以确定选中的响应在 HTTP 规则下是否可以缓存，如果可以缓存，则还需明确其缓存时间。它会通过响应头的 Cache-Control、Expires、Pragma、Vary、ETag、Age 和 Last-Modified 信息来检查响应是否有效。

有些浏览器（包括 IE）支持在 HTML 文档中使用 META HTTP-EQUIV 标记缓存信息。Caching Inspector 会扫描包含 HTML Content-Type 的响应，显示标记中的任何 HTTP-EQUIV 或 PRAGMA 指令。

对于没有显式指定是否可缓存以及缓存生命周期的响应，Caching Inspector 会通过请求头的 Last-Modified 标签以及 RFC2616 中提出的算法，启发式计算缓存的生命周期，如图 5-5 所示。

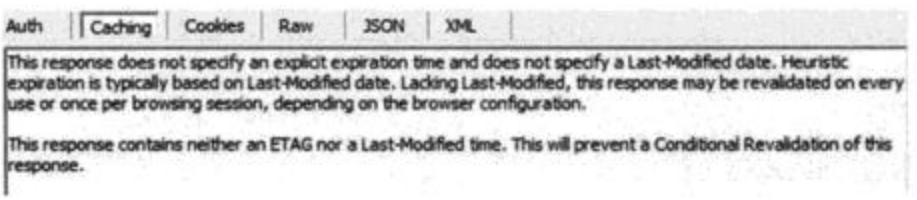

图 5-5

通过以下链接你能了解更多浏览器如何利用缓存的内容：http://fiddler2.com/r/?httpperf。

5.4 COOKIES

类型	请求&响应
允许编辑	否

Cookies Inspector 会显示任何发送出去的请求头 Cookie 和 Cookie2 中的内容，并显示接收到的响应头中的 Set-Cookie、Set-Cookie2 和 P3P。注意，请求头 Cookie2 和 Set-Cookie2 不经常使用，它们在 IE 以及一些其他浏览器中不被支持。

Cookie 的显示形式是基本的信息（在 Headers inspector 中可以看到相同的信息）。这个 inspector 的值的确定，需要检查 P3P 响应头；如果该响应头存在，则确定它的 Cookie 是否已经缓存。P3P（Platform for Privacy Preferences）是标准的服务器和客户端间的通信方式，客户端可以确定如何使用包含 P3P 请求头的 Cookie。

```
Set-Cookie: ASPSESSIONIDCCBTDCRD=CIFKIKJDFMJFFODAJPFMFKGN; path=/
P3P: CP="ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo
          OUR SAMo CNT COM INT NAV ONL PHY PRE PUR UNI"
```

Inspector 可以解析 P3P 头的 CP（Compact Policy）字符串中的标识符，其含义如图 5-6 所示。

图 5-6

Inspector 会进一步对字符串进行检查，确定 IE 使用的默认的隐私设置是否认为该 Cookie 是“可接受的”。值得一提的是，有些网站（如 Facebook、Google 等）会发送非法的 P3P 声明，以屏蔽浏览器的隐私功能。在 P3P 规则下，无法识别的标记（比如以上提到的非法声明）会被忽略。

Cookies Request Inspector 显示发出的 Cookie 头的大小，帮助删除超大 Cookie 或减少这些 Cookie 大小以提升网络性能。

Cookies Response Inspector 标志出一些常见的问题。例如，IE 会拒绝为所有主机名包含下划线的服务器设置 Cookie，Inspector 发现这种情况时会给出提示。

5.5 HEADERS

类型	请求&响应
允许编辑	是

每个 HTTP 请求都是以纯文本形式的请求头开始，描述客户端需要什么资源和操作。请求的第一行（“Request Line”）包含三个值：HTTP 方法（如“GET”或“POST”）、请求的 URL 路径（如“/index.htm”）以及 HTTP 版本号（如“HTTP/1.1”）。Request Line 下方是一行或多行，其中包含用于描述请求和客户端的元数据的键值对，如 User-Agent 和 Accept-Language。

类似地，所有的 HTTP 响应都是以纯文本响应头开始，它描述了请求的结果。响应的第一行（“Status Line”）包含 HTTP 版本号（如“HTTP/1.1”）、响应状态码（如“200”）以及响应状态文本（如“OK”）。Status Line 下方是一行或多行，其中包含用于描述响应和服务器的元数据的名称值对，比如响应文件的长度、类型以及响应如何缓存的信息。

Headers Inspector 支持查看请求和响应的 HTTP 头，在 Request Line 或 Status Line 下方以树形视图的形式显示名称值对。HTTP 头的名称值对是基于功能进行分组的，然后根据名称按字母序排序。分组只是为了阅读方便，不会发送到网络。对于请求头，分组包括[Cache, Client, Entity, Transport, Cookies/Login, Miscellaneous]。对于响应头，分组包括[Entity, Transport, Cookies/Login, Security, Miscellaneous]。

默认情况下，Headers Inspector 的内容是只读的，无法对其进行编辑。在只读模式下，树形视图和 Raw Headers 对话框的背景色是只读颜色（淡蓝色）。当选中在某个断点处暂停的 Web Session，或者选中 Edit 菜单中的 Unlock for Editing 选项时，Headers Inspector 会以 Edit 模式显示。在 Edit 模式下，背景颜色是默认的窗口颜色（白色），此时，可以对 headers 的内容进行编辑。

点击 Inspector 右上方的 Raw 超链接，会显示纯文本形式的 header，header 就是以纯文本形式发送到网络上的。点击链接 Header Definitions，可以查看关于常见 HTTP header 及其使用方式的帮助。

由于 Windows 树形视图控制存在局限性，因此只能显示 header 的名称和值的前 260 个字符。要查看长度超过 260 个字符的 header，可以选中该 header，按下 Enter 或 F2 键，或右击 header 并选中 View Header。Header Viewer 窗口会以只读模式打开，可以查看其全名和值。在

窗口的标题栏中会显示值的长度（以字符数表示）。

虽然 Inspector 处于只读（ReadOnly）模式，但为了便于阅读，所有的 Cookie 请求头都会分解成名称-值对。在编辑（Edit）模式下，任何 Cookie 头都在单独一行中显示，发送给服务器时也是每行一个 Cookie 头。

5.5.1 上下文菜单

右击 Header 列表会显示包含以下选项的菜单，如表 5-1 所示。

表 5-1 Header 菜单

View Header	打开选中 header 的 Header Viewer 窗口。
Edit Header	（只以编辑模式显示）打开选中 header 的 Header Editor 窗口。
Copy Header	把整个选中的 header 拷贝到剪贴板。
Copy Value only	把选中的 header 的值拷贝到剪贴板。
Send to TextWizard	把选中的 header 的值拷贝到 TextWizard，以便对编码过的文本进行解码。
Add Header	在 Edit 模式下，创建新的 header。
Remove Header	在 Edit 模式下，删除选中的 header。
Paste Headers	在 Edit 模式下，尝试基于剪贴板的信息添加新 header。
Lookup Header	打开包含选中 HTTP header 的信息的 Web 页面。

5.5.2 快捷键

选中 Web Session 列表中的 Session 后，按下 CTRL+H 键可以激活请求和响应的 Header Inspectors。在 Inspector 中，可以使用表 5-2 所示的快捷键：

表 5-2 快捷键

CTRL+C	把选中的 header 拷贝到剪贴板。
CTRL+SHIFT+C	把选中 header 的值拷贝到剪贴板。
F2 或 Enter	打开选中 header 的 Header Viewer 或 Header Editor 窗口。
CTRL+V	在 Editing 模式下，基于剪贴板上的文本，添加一个或多个新 header。
Insert	在 Editing 模式下，创建新的空白 header。
Delete	在 Editing 模式下，删除选中的 header。

5.5.3　编辑

当 Inspector 处在 Edit（编辑）模式下时，把 headers 发送到服务器或客户端之前可以对它进行修改。如果想一次性修改全部 headers，可以点击 Inspector 右上方的 Raw 链接。此时会显示 Raw Headers 窗口，可以保存新的 header 集。

此外，还可以通过编辑 Inspector 上方的文本内容，直接修改 Request 或 Status Line 的内容。要编辑某个名称-值对的 header，可以选中它，按 F2 键或点击 Enter，Header Editor 窗口就会出现。要添加新的 Header，按 Insert 键；会出现新的 header，并自动打开 Header Editor。

你可以在对话框中更新新 header 的名称-值对，或者点击窗口下方的 Header Templates 按钮，从常用的 header 中选择名称-值对，如图 5-7 所示。

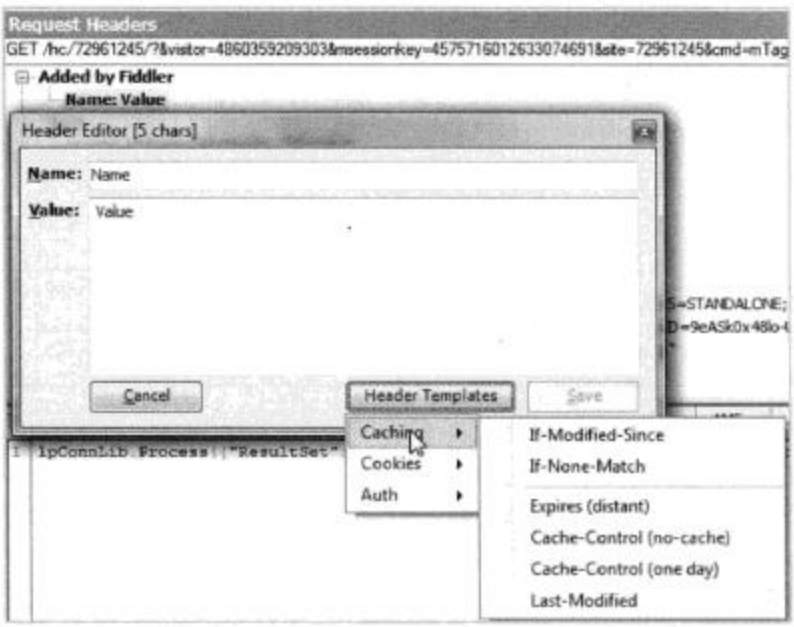

图 5-7

5.6　HEXVIEW

类型	请求&响应
允许编辑	是

HexView Inspector 支持在十六进制编辑框中查看请求和响应的 header 和 body。当内容是二进制形式时，该功能很有用。

该 Inspector 包含 HexEdit 控制栏，提供三个列。最左侧的列用浅灰色显示，表示相邻列的字节流的十六进制地址。中间列是各个字节流的十六进制形式。最右侧的列以 ASCII 文本形式显示这些字节流。如果 HexView 被配置成显示 header，请求头会是蓝色的，响应头是绿色的，请求体用黑色显示。

Inspector 的最下方是状态栏，包含三个面板。第一个面板显示的是当前光标在字节流中的位置，其值有十进制和十六进制两种格式。如果 Inspector 需要显示 header 字节，当光标进入 body 部分的字节时，会自动把偏移重置为 0。

如果在控制栏中选中了一个或多个字节，中间面板显示的是当前选中字节的长度。

右侧面板显示了当前的模式：Read only、Overwrite 或 Insert。除非 Session 是暂停在断点处或由于编辑而处于未锁定状态，否则 Inspector 都会处于 Read only 模式。在编辑模式下，按下 Insert 键，可以在重写和插入新的字节之间切换。

HexView Inspector 的上下文菜单支持表 5-3 中的选项：

表 5-3 HexView 菜单

Insert File Here...	在 Edit 模式下，在当前光标位置插入选中文件的内容。
Select Bytes...	弹出对话框，要求你输入要选择的字节数，然后从当前光标位置开始按指定数量选中相应字节。如果你想指定十六进制形式的字节数，在最前面键入$字符。 提示：如果你想查看 HTTP 响应的 Chunked 编码块，该功能就很有用，因为每个块的长度使用十六进制数来指定。
Save Selected Bytes...	当选中一个或多个字节时，该项可将选中的字节保存到指定文件中。
Goto Offset...	支持把光标移到选定的位置。 输入+或-字符，可以指定从当前光标位置开始的相对偏移，而不是从内容最开始的偏移。如果你想指定十六进制形式的字节数，在最前面键入$字符。
Find Bytes...	可以指定要搜索的十六进制内容。该搜索从当前光标所在的位置开始。按下 F3 键，在每次匹配后可以继续向下搜索。
Find String...	支持指定要搜索的字符串（字节会被当做 UTF-8 字符处理）。搜索时从当前光标所在的位置开始。按下 F3 键，在每次搜索匹配后可以继续向下搜索。

续表

Show Headers	该复选框用于控制是否显示 header 字节。如果没有选择该复选框，只会显示 body 字节。 提示：在当前版本的 Fiddler 中，当 HexView Inspector 处于 Edit 模式时，该选项会自动取消选中，因为它不支持编辑 header。
Set Bytes per Line…	支持指定每行显示多少个字节。0 表示让 HexView 自动基于可用的宽度选择字节数（每行最少四个字节）。

5.7 IMAGEVIEW

类型	响应
允许编辑	否

ImageView Inspector 支持以图片形式显示响应的内容。该 Inspector 既支持常见的 Web 图片格式，包括 JPEG、PNG、GIF，也支持一些不常见的格式，比如游标、图片、位图、EMF/WMF 和 TIFF 格式。该 Inspector 不支持 SVG 图；如果安装了 IE9 或更新版本，可通过 WebView Inspector 查看 SVG 格式的响应。

左侧的灰色面板中显示的是当前选中图片的信息，包括其大小（以字节数表示）、像素尺寸以及文件格式。在灰色面板底部是一个下拉列表，可以控制图像的缩放：

- No scaling：图片以原始大小显示。
- Autoshrink：自动缩小尺寸比显示区域大的图像。
- Scale to fit：按显示区域大小自适应调整，将大图像缩小，将小图像放大。

在这个 Inspector 的上下文菜单中，可以以位图方式复制图片到剪贴板，或使用当前时间作为文件名的一部分，快速将图像文件保存到 Windows 桌面。它还提供了一个选项，把图像作为 DataURI 进行复制，DataURI 是一种文本格式，可以嵌入到 HTML 或样式表中，在新一代的浏览器（IE8+）中支持。如果结果 URI 的长度超出 32kb，Fiddler 会弹出警告信息，提示 IE8 不支持大于 32kb 的 URI，需要 IE9 或更新版本的浏览器来渲染图片。菜单中的最后一个选项支持改变背景的颜色（通常是淡蓝色），当查看小图片或透明的图片时，它可以提供有用的对比度。

中键点击图片会把该图片复制到桌面文件夹中。双击图片会打开一个全屏视图。在全屏

视图中，包含表 5-4 功能：

表 5-4 全屏视图功能

键盘或鼠标动作	功能
Enter 或 Z	在全屏和实际大小之间切换
H	水平旋转图像
V	垂直旋转图像
R	把图像顺时针旋转 90 度
鼠标上滚	全屏显示
鼠标下滚	实际大小显示
Escape	退出全屏显示

为了更快速地查看大量的图像响应，可以使用扩展组件 GalleryView。

5.8 JSON

类型	请求&响应
允许编辑	否

JSON Inspector 会把选中的请求或响应体解释成 JSON（JavaScript Object Notation）格式的字符串，以树形视图显示 JSON 对象节点。如果请求或响应体不是 JSON 格式，树形图会是空的。和大多数 Inspector 不同，JSON Inspector 可以渲染数据，即使请求或响应是压缩格式或使用了 HTTP Chunked 编码；你不需要删除编码来显示数据内容。

很多 Content-Type 指定是 JSON 的响应实际上并不是 JSON 格式。相反，它们是 JSONP，包含单个函数调用的 JavaScript 文件，函数的一个参数是 JSON 格式的字符串。JSON Inspector 能够处理很多类型的 JSONP，它会忽略前面的函数调用，删除后面的括号和分号。但是，有些 JSONP 格式不正确，比如名称/值对没有加引号：JSON Inspector 无法处理这种错误格式，会拒绝解析内容。

上下文菜单支持两个选项：Copy 用于把选中的节点复制到剪贴板（或按下 CTRL+C）；Send to TextWizard 用于把选中的节点内容发送到 TextWizard 窗口，对其进行编码和解码处理。

页脚的 Expand All 按钮会展开树的所有节点，而 Collapse 按钮会收回所有节点。如果请求体包含的节点数少于 2000，JSON 树会自动扩展。出于性能方面的考虑，对于大文档，你应该手动展开树形图。

5.9 RAW

类型	请求&响应
允许编辑	是

Raw Inspector 支持查看完整的请求和响应，包括文本格式的 header 和 body。“raw（原始）”这个词容易让人误解，因为 Fiddler 对请求和响应的字节进行了解析；如果需要查看网络流的纯粹的“raw”视图，可以使用数据包嗅探器工具，如 Microsoft Network Monitor （NetMon）或 Wireshark。

该 Inspector 的绝大部分区域是一个大文本块，它以文本形式显示 header 和 body 信息，使用的字符集通过 header、字节序标记或嵌入的 META 声明标记。在文本区域按下 CTRL+G 键可以把鼠标移动到指定的行号。右击文本区域，弹出的上下文菜单中提供了标准的“剪切”、“复制”和“粘贴”选项。该上下文菜单还提供相应选项，可以将当前选中的文本发送到 TextWizard 工具。菜单中还有两个复选框，一个用于控制是否启用自动换行（Word Wrap），另一个用于控制是否启用自动截断（AutoTruncate）功能。

Inspector 下方是个条形栏，它支持一些其他功能。首先，搜索框支持选中和内容匹配的文本。搜索文本区分大小写，不支持正则表达式。按下搜索框中的向上或向下箭头可以让文本区域滚动（以便查看搜索结果的上下文）。输入过程中会实时选择匹配项，如果找到匹配项，搜索框会显示为绿色；如果没有找到匹配项，搜索框会显示为红色。按下 Enter 或 F3 可以跳到下一个匹配项。按下 CTRL+Enter 键会高亮显示所有匹配的内容。

View in Notepad 按钮会把文本内容保存到临时文件，并打开文本编辑器查看文件。文本编辑器由选项中的 fiddler.config.path.texteditor 项控制；默认使用系统自带的记事本程序（notepad.exe）。

Inspector 会用 Unicode 码（◆）替换所有的 null 字节，因此，这个 Inspector 也可以查看二进制的响应内容，即使 HexView 更适用于这个任务。因为在文本框中显示大的二进制格式的 body 需要占用大量的 CPU 时间和内存，所以这个 Inspector 被配置成对大的响应进行自动

截断。触发截断的阈值是由 Content-Type 以及以下四个选项控制的：

- fiddler.inspectors.request.raw.truncatebinaryat
- fiddler.inspectors.request.raw.truncatetextat
- fiddler.inspectors.response.raw.truncatebinaryat
- fiddler.inspectors.response.raw.truncatetextat

默认情况下，二进制形式的 Content-Types 在 128 个字节处截断，文本形式的 Content-Types 在 262144 字节处截断。可以通过上下文菜单关闭截断功能。

5.10 SYNTAXVIEW

类型	请求&响应
允许编辑	是

SyntaxView Inspector 支持根据多个规则高亮显示多种类型的请求和响应体文本。当读 HTML、XML、CSS 和 JavaScript 时，这个功能是非常有用的。Inspector 通过 Content-Type 头判断应该使用哪种高亮规则。

SyntaxView Inspector 是 Fiddler 中最有用的 Inspectors 之一，但是由于它太大了，因此，不包含在默认的安装包中。Fiddler 当前的安装包大约 750KB，如果包含 Syntax Highlighting 扩展，其大小几乎会翻一倍。为了确保 Fiddler 更新的简洁性，SyntaxView 扩展需要单独下载，下载地址为 http://fiddler2.com/r/?SYNTAXVIEWINSTALL，如图 5-8 所示。

图 5-8

在 Inspector 下方是个条形栏，它提供了几个额外的信息和功能。第一个是个文本，它给出了光标当前所在的行和列。下一个是 QuickFind 框，它支持内部搜索。接下来是个按钮，它提供了高级的“查找并替换”功能。在条形栏的外侧是一个文本，显示 Inspector 当前是只读模式还是编辑模式。

QuickFind 框支持正则表达式，只需要在搜索串前面加上前缀 REGEX：字符串，剩余部分会被解释成正则表达式。

除了标准的“剪切”、“复制”、“粘贴”、“撤消”和“重新执行”命令，SyntaxView 的 Context 菜单中还提供了表 5-5 所示功能：

表 5-5　　SyntaxView 的功能

Send to TextWizard	把当前选中的文本发送给 TextWizard 窗口以进行编码和解码
Format XML	尝试以 XML 格式解析文档。如果解析成功，文本会以缩进的 XML 树形图重新格式化。即使 Inspector 是在只读模式下，该选项也是可用的，但是只有在编辑模式下才会一直使用重新格式化功能。
Format Script/JSON	尝试以 JavaScript/JSON 格式解析文档。文本会以标准的 JavaScript 缩进格式显示。即使 Inspector 是在只读模式下，该选项也是可用的，但是只有在编辑模式下才会使用重新格式化功能。
Find...	打开“查找和替换”对话框，它提供了很多查找和替换选项。
Word Wrap	该复选框可以控制文本是否自动换行。
Editor Options...	打开一个包含了很多高级文本显示选项的选项窗口。

5.11　TEXTVIEW

类型	请求&响应
允许编辑	是

TextView Inspector 支持以文本形式查看请求体和响应体。TextView Inspector 会在第一个空字节处截断，因此不适合用于显示二进制内容。

Inspector 的主体是个大的文本区域，显示请求体或响应体的文本，其使用的字符集是通过 header、字节序标记或嵌入 META 标记的声明来检测的。在文本区域按下 CTRL+G 键，可

以把光标移动到指定的行号。右击时，会弹出一个菜单，提供标准的“剪切”、“复制”和“粘贴”功能。此外，还提供了一个控制自动换行功能的复选框。该菜单中还提供了可以把当前选中的文本发送到 TextWizard 工具中的功能。

Inspector 的下方是个条形栏，提供了一些其他信息和功能。第一栏中记录了光标当前所在的行和列（以 Line:Column 格式显示）。第二栏以偏移/总数（Offset/Total）形式显示当前字符在全部内容中的偏移。第三个选项框显示的是当前选中的字符数。

下面是个搜索框，支持在内容中选择文本。搜索文本是大小写敏感的，不支持正则表达式。按下搜索框中的向上或向下箭头可以让文本区滚动（目的是为了方便用户在正文中查看搜索结果）。输入的过程中，程序就会执行匹配，如果找到了匹配项，搜索框会显示为绿色；如果没有找到匹配项，搜索框显示为红色。按下 Enter 或 F3 键可以跳到下一个匹配项。按下 CTRL+Enter 键会高亮显示所有的匹配项。

View in Notepad 按钮会把文本内容保存到临时文件中，并在文本编辑器中打开这个文件。文本编辑器是由 fiddler.config.path.texteditor 来控制的，默认是记事本（notepad.exe）。

点击条形栏右侧的...按钮，会把内容保存到临时文件中，并弹出 Windows 的 Open With 对话框，从中选择一个应用程序来处理这个文件。

5.12 TRANSFORMER

类型	响应
允许编辑	总是

5.12.1 响应的编码的一些背景知识

HTTP 规范为响应定义了很多可以提高性能的编码方式：

- 压缩（Compression）算法（如 DEFLATE）可以应用于 HTTP 体中，能够降低在网络上实际传输的字节数。文本类型的 Content-Types，如 HTML、Script 和 CSS，压缩率可以达到 80%。
- 分块传输编码（Chunked Transfer-Encoding）支持在传输 body 前不需要预先设置 body 的长度。一般而言，body 的长度是作为 Content-Length 头发送的，但是预先计算 body 的大小可能需要很多时间和内存，尤其是当 body 的内容基于数据库查询或其他操作

的情况下生成时更是如此。

如果没有分块传输编码，服务器返回的响应的大小是不确定的时候，必须发送 Connection: close 头，当响应结束时关闭连接。这种机制与 HTTP 的 Keep-Alive 机制是冲突的，会造成性能下降，出现新的性能瓶颈点。

分块的原理是通过发送一个或多个块的数据，每个数据块前面包含一个以十六进制数表示的长度值。接收到数据块中长度值的取值为 0 时，说明数据传输完成。下面是一个分块形式的响应的例子：

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

2b
This is a response which has been delivered
21
using HTTP Chunked encoding. To r
38
educe overhead, the chunks should be larger than those in
0c
this exampl
18
e; 2kb is a common size.
0
```

5.12.2　使用 Transformer 添加或删除编码方式

编码会使得查看 body 内容变得复杂。Transformer Inspector 支持向响应中添加或从响应中删除基于 HTTP 的编码。在 Transformer 选项卡的上方列出了 body 当前的大小；在添加或删除编码时，可以留意这个数值，如图 5-9 所示。

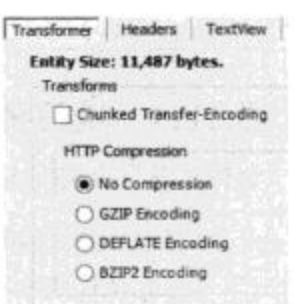

图 5-9

Chunked Transfer-Encoding 复选框从响应中添加或删除 Chunked Transfer-Encoding。选中或者取消这个复选框可以应用或删除编码，然后添加或删除 Transfer-Encoding 和 Content-Length 响应头。启用分块后，会关闭该复选框下方的 HTTP 压缩选项框。要改变压缩方式，首先必须删除分块方式，然后调整 HTTP 压缩方式。

HTTP Compression 选项框中的单选按钮支持对 body 进行压缩或解压缩：

- No compression：body 是未压缩的。
- GZIP：body 使用 GZIP 方式压缩。该编码内部使用的也是 DEFLATE，但是二进制格式稍有些不同。
- DEFLATE：Body 是通过 DEFLATE（RFC1951）算法压缩的。
- BZIP2：body 是通过 BZIP2 算法压缩的。BZIP2 算法的压缩率通常比 GZIP 或 DEFLATE 要高，代价是消耗的 CPU 时间更多。目前仅有 Google 的 Chrome 浏览器支持该格式，而其他主流的浏览器或服务器都不支持。Fiddler 提供这个算法主要是为了进行比较，以及用于测试客户端在遇到不能识别的编码格式时的处理行为。

选中单选按钮就可以对 body 应用或删除某个算法，并添加、更新或删除 Content-Encoding 头以及调整 Content-Length 头。

借助这个 Inspector，可以了解到这些 HTTP 压缩算法应用到 body 内容时的效率如何——对于大多数文本类型，压缩大约会减少 body 大小的 80%。和大多数的 Inspector 不同，Transformer Inspector 支持修改 ReadOnly（完成的）响应，而不需要提前解锁它。有了这个功能，可以更方便地删除 Session 的编码，以便查看。

当前，Transformer Inspector 只能用于处理响应。很少会对请求使用 HTTP 编码，而且大多数客户端或服务器也不支持这些编码。

5.12.3 删除编码的其他方式

因为大多数 Fiddler Inspectors 无法正常处理编码过的内容，当选中某个编码过的 Session 时，在 Inspector 列表的上方会出现一个黄色按钮。点击这个按钮会从请求或响应中删除所有的编码，如图 5-10 所示。

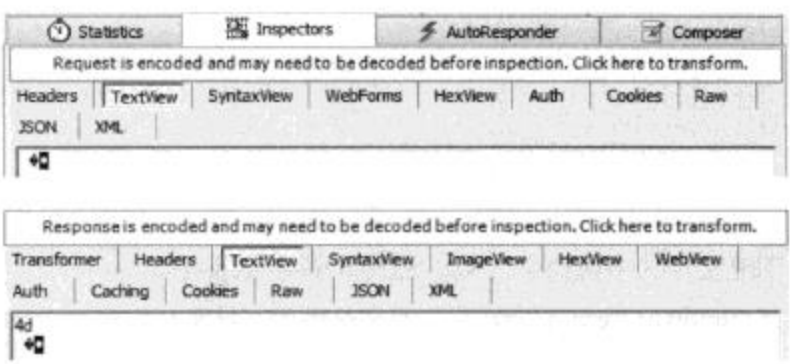

图 5-10

也可以右击 Web Session 列表中的 Session，从上下文菜单中选择 Decode Selected Sessions，可以同时从请求和响应中删除编码。同样，启用工具栏中的 Decode 选项也会自动删除所有编码，因为 Fiddler 是从客户端读取请求，从服务器读取响应。

5.13 WEBFORMS

类型	请求
允许编辑	是

WebForms Request Inspector 会解析请求的查询字符串和请求体，查看是否包含 HTML 格式的数据内容。如果找到 HTML 格式的数据，会对它进行解析，在网格视图中显示名称/值对。例如，以下请求：

```
POST /sandbox/FileForm.asp?Query=1 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: www.fiddler2.com
Content-Length: 54

2=Data%3e123&fileentry2=a%2etxt&_charset_=windows-1252
```

会显示如图 5-11 所示的内容：

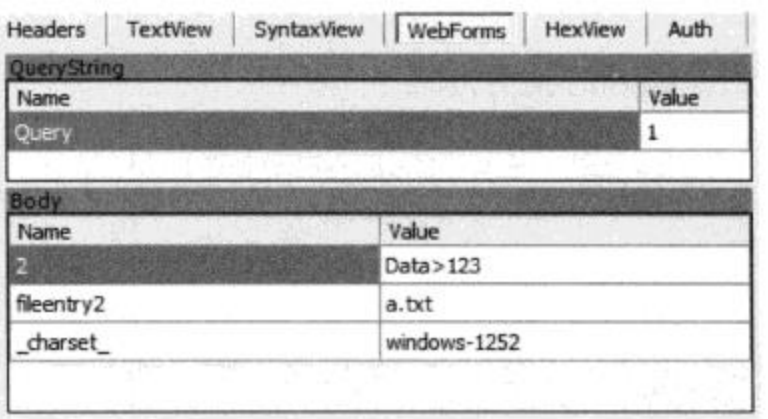

图 5-11

该 Inspector 最适用于大多数简单的 Web 格式所使用的 application/x-www-form- urlencoded 形式的数据。支持 multipart/form-data 格式，通常是用于文件上传，在显示功能中的使用时受限的。要修改文件上传，应该使用 HexView Inspector。

5.14 WEBVIEW

类型	响应
允许编辑	否

WebView Response Inspector 支持查看 Web 浏览器控制条中的响应，它可以快速预览某个给定响应在浏览器中是如何显示的。Web 浏览器控制条被配置成渲染响应时阻止其他下载，从而避免把 Web Session 列表都搅合在一起——这意味着大多数的图片、样式和对象不会显示在内容中。此外，也会阻止脚本和导航，对 HTML 页面只提供预览方式。

除了纯 XHTML 和 HTML，对于 IE8 或更新的版本，WebView Inspector 还可以渲染一些其他的多媒体类型。对于 IE8，Inspector 可以渲染所有小于 24KB 的二进制图片（png、jpg、gif 等）。存在大小限制的原因是该 Inspector 使用 Data URI 渲染图片，而 IE8 的 Data URI 长度限制是 32KB，相当于 24KB 的二进制形式。

对于 IE9，该 Inspector 可以显示超过 1.5GB 大小的图片。此外，它还可以显示 SVG 文档，支持为 WOFF、TTF 和 EOT 字体文件、MP3 音频文件、h264 视频文件生成预览页面。举个例子，查看 WOFF 文件时显示如图 5-12 所示的内容如下：

图 5-12

当预览音频或视频文件时，在选项卡的最上方有一个“AutoPlay”复选框。当选中

“AutoPlay”功能时，多媒体文件在加载后会自动开始回放。如果没有选中该功能，多媒体文件在加载后不会播放，只有点击播放按钮后，才会开始播放。

5.15　XML

类型	请求&响应
允许编辑	否

XML Inspector 会把选中的请求和响应体解释成 XML 格式的字符串，显示 XML 文档节点的树形图。如果 body 不是合法的 XML 格式，树形图会是空的。

每个 XML 元素都表示成树形图中的一个节点，元素的属性在元素名称后以方括号的形式表示。

和其他的 Inspector 不同，即使请求或响应是压缩的或应用了 HTTP Chunked Encoding，XML Inspector 仍然可以对其数据进行渲染并显示，显示内容时不需要删除编码。

树形视图的上下文菜单有两个选项：Copy 用于把选中的节点复制到剪贴板（或按下 CTRL+C 键）；Send to TextWizard 用于把选中节点的内容发送到 Textwizard 窗口进行编码或解码。

点击最下方的 Expand All 按钮可以展开树形图的所有节点；而点击 Collapse 按钮则可以收起树形图的所有节点。如果 XML 树包含的节点数少于 2000 时会自动展开；由于性能原因，当文档很大时，树不会被自动展开，而需要手动展开。

第6章 扩展

6.1 概览

Fiddler 提供了丰富的扩展模型，开发人员可以通过这些能够轻松安装的插件来增强 Fiddler 的功能。

6.1.1 流行的第三方扩展

独立开发者们已经构建了很多 Fiddler 扩展，下面这个地址中列出了部分扩展：http://getfiddler.com/addons。到编写本书时为止，已经有了一批非常流行的第三方扩展，这些扩展有效地增强了 Fiddler 在对 Web 应用进行性能测试和安全测试方面的功能。

性能扩展组件

Fiddler 本身已经提供很多重要的性能分析和优化功能，然而，扩展给 Fiddler 带来了更强大的功能。

- neXpert 性能报告生成器——它是微软在线服务测试团队开发的一款扩展，neXpert 专注于性能优化，可以对 Web 站点进行评估并生成报表，会指出问题并给出解决方案。许可方式：免费软件。
- StresStimulus——这款负载能力测试扩展支持对 Web 站点的承载能力进行测试并记录测试过程中的一些关键数据，使用这个扩展可以评估一个网站可以为多少个并发用户提供服务。许可方式：免费试用。

安全扩展组件

Fiddler 支持很多种安全测试。网络安全专家们构建了一些强大的安全方面的扩展组件，可以帮助新手发现并解决安全问题。

- Watcher——由 Casaba Security 团队开发。Watcher 是一种“被动安全审计器”，它可以监测浏览器和网站的交互。该工具会侦听请求和响应，标记出潜在的安全漏洞。专业的安全渗透（security penetration）测试人员使用该工具来评估主要站点。许可方式：开源软件。
- x5s——Casaba Security 团队开发的另一个组件，x5s 可以评估网站漏洞，包括由于字符集相关的问题导致的跨站脚本错误。许可方式：开源软件。
- intruder21——该组件支持对 Web 应用程序执行模糊测试（fuzz-testing）。确定了 Fiddler 接收的目标请求后，该扩展会生成模糊负载，并针对网站施加这些负载。许可方式：免费软件。
- Ammonite——该组件监测常见的网站漏洞，包括 SQL 注入、操作系统命令注入、跨站脚本运行、文件夹带（file inclusion）以及缓冲区溢出。许可方式：免费试用。

6.1.2　我创建的扩展

本章的剩余部分将介绍我自己开发的一些最有用的扩展，这些扩展都可以免费从 http://getfiddler.com/addons 中免费下载。

有些扩展对于大多数 Fiddler 用户都有用，在 Fiddler 安装包中没有包含它们主要是为了减小安装文件的大小。其他扩展只在某些不太常见的场景下有用，通过附加组件模型提供这些功能可以避免 Fiddler 过分“膨胀”——同时也确保了 Fiddler 的附加组件模型足够强大，可以满足开发社区的需求。

大部分扩展的源代码是开源的，希望扩展 Fiddler 满足自己需求的开发人员可以以此为参考样例。

6.2　JAVASCRIPT FORMATTER

安装了 JavaScript Formatter 后，可以右击任意包含 JavaScript 响应的 Session，选中 Make JavaScript Pretty。该命令会重新格式化 JavaScript，从而极大提高其可读性，特别是对压缩过的 JavaScript 更为有用。举个例子，对于图 6-1 所示的这行 JavaScript 代码：

```
1 /* Copyright (C) 2012 Microsoft Corporation */(function(){var b=window,e=
  b.jQuery,f=b.Debug,g=b.wLive={Core:{},Controls:{}},a=b.$Config;a.
  handlerBaseUrl=a.handlerBaseUrl||"";if(!a.ed){var d=document.domain,c=d.
  split(".");a.sd=c.length===1?"":"."+c[c.length-2]+".com";a.mkt=a.mkt||"
  na";a.prop=a.prop||"X";if(typeof window.SymRealWinOpen!=="undefined")
  window.open=window.SymRealWinOpen}})();(function(){var a=window,e=a.
```

图 6-1

重新格式化提高可读性后的形式如图 6-2 所示：

```
/* Copyright (C) 2012 Microsoft Corporation */
(function() {
    var b = window, e = b.jQuery, f = b.Debug, g = b.wLive = {
        Core: {}, Controls: {}
    },
    a = b.$Config;
    a.handlerBaseUrl = a.handlerBaseUrl || "";
    if (!a.sd) {
        var d = document.domain, c = d.split(".");
        a.sd = c.length === 1 ? "" : "." + c[c.length - 2] + ".com"
    }
    a.mkt = a.mkt || "na";
    a.prop = a.prop || "X";
    if (typeof window.SymRealWinOpen !== "undefined")
        window.open = window.SymRealWinOpen
})();
```

图 6-2

要对所有脚本响应启用自动格式化，选中 Fiddler 的 Rules 菜单中的 Make JavaScript Pretty 选项即可。因为对 JavaScript 的格式化是在下载过程中执行的，所以客户端只会看到格式化后的形式，如果你正在使用浏览器的脚本调试工具，这个功能就会很有用。

还可以通过 X-Format-JS session 标记，手动控制 JavaScript 格式化程序是否对响应进行格式化。如果这个标记取值为 0，该扩展组件就不会对响应体进行格式化，即使在 Rules 菜单中选中了 Make JavaScript Pretty 选项。将该标记设置成其他任何值都会对响应体做格式化，即使在 Rules 菜单中没有选中 Make JavaScript Pretty 选项。

JavaScript Formatter 扩展和极少数响应不兼容，对于这些响应，该格式化程序会无法执行。在两种场景下可能会出现这个问题：一是响应头中声明是 JavaScript MIME 类型，但内容实际上并不是真正的 JavaScript（如一些 Google 属性）；二是如果使用（非常模糊的）JavaScript 续行功能，格式化过程可能会出现问题。续行方式支持脚本开发人员通过反斜杠结束一行，JavaScript 引擎会自动连接其下一行。JavaScript Formatter 解析器无法识别该功能，因而无法正确地对这种行进行格式化。

6.3 GALLERY

Gallery 扩展组件用于显示选中 Session 中的所有图片。如果你希望和大量的图片交互或者希望基于返回的图片快速选中某个 Session，该功能就非常有用。

该功能只有几个选项，显示在选项卡的最上方，如图 6-3 所示。

图 6-3

默认情况下，每个图片都是 150×150 像素的缩略图，而且小于 10kb 的图片会被忽略。点击链接 Filter sessions，会选中被显示图片的 Session，取消选中没有显示的图片或者图片被从视图中删除的 Session。点击链接 Help 会显示文本，解释扩展的功能。

把鼠标悬停在缩略图上会给出提示显示 Session ID、URL、图片大小以及图片格式，如图 6-4 所示。

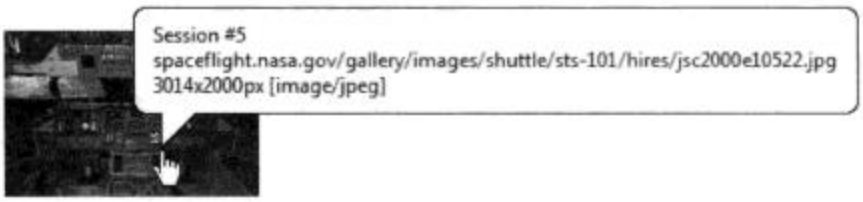

图 6-4

中键点击缩略图会从 Gallery 视图中删除被点击的图片。右击缩略图会从 Gallery 中切换出来，使用 Inspector 查看选中图片所在的 Session。如果在右击或双击时按住 Shift 键，Inspectors 会在弹出窗口中打开。

全屏视图

双击缩略图会进入全屏视图模式。如果图片所在的 Session 包含 Comment，会在屏幕下方以标题形式显示该评论。

在全屏模式下，可以使用鼠标或键盘来控制视图：

- 点击中键或按下 Escape 键可以退出全屏模式。
- 要处理下一个图片，左击，按空格键，按向右的箭头，或按 Page Down 键。要回到前一张图片，右击，按 Shift+Spacebar 键，按向左的箭头或按 Page Up 键。
- 按下 Z 键、Enter 键或通过鼠标滚动都可以切换缩放级别（实际尺寸大小、调整到合适尺寸）。如果图片以实际尺寸大小显示，而且图片大于屏幕尺寸，用鼠标可以移动图片。
- 按下 1～9 之间的任意数字键，可以启动幻灯片模式自动播放图片，数字表示播放每

张图片所停留的时间。按 0 键会取消幻灯片自动播放。把幻灯片和自动显示注释的功能结合起来，可以把 Fiddler 作为一个基本的照片演示工具。

- Fiddler 支持各种选项，可以临时改变图片的显示方式。按 H 键可以水平翻转所显示的图片，按 V 键可以垂直翻转图片。按 R 键可以顺时针将图片旋转 90°，按 I 键可以切换图片颜色。按 S 键可以把图片转换成深褐色，按 G 键会转换成灰色。按 C 键可以将图片在绿色和灰色之间切换。按 E 键可以把图片的非红色像素转换成灰色；该操作很慢，对于大图片可能需要数秒的时间。按 U 键会撤消对图片应用的所有修改。
- 按 Delete 键会从 Gallery 中删除图片，并跳到下一张图片。

6.4 CONTENT BLOCKER

Content Blocker（内容拦截器）扩展组件可以向客户端返回 HTTP/404 响应，从而拦截从下载中选中的内容。该功能可以用来模拟测试内容被广告拦截器或IE8 以及更新版本中提供的访问跟踪保护功能拦截时 Web 应用程序的行为。

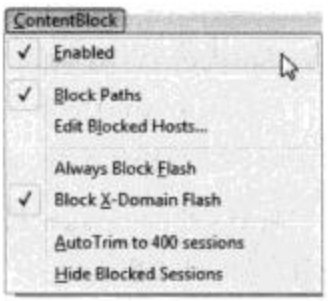

图 6-5

要安装该扩展，先从 Fiddler 网站下载它，并把 DLL 拷贝到目录\Program Files\Fiddler2\Scripts 文件夹中。安装完扩展后，在 Fiddler 的主菜单中会新增一个 ContentBlock 菜单，如图 6-5 所示。

ContentBlock 菜单中提供了表 6-1 所示的几个简单选项。

表 6-1 ContentBlock 菜单项

Enabled	如果选中，会启动拦截。不选中时，就不会拦截任何内容
Block Paths	如果选中，URI 的路径中包含/ad 的请求都会被拦截
Edit Blocked Hosts	点击该选项，会新打开一个窗口，可以编辑拦截主机列表，所有来自这些主机的请求都会被拦截。主机名必须使用完整名称，主机之间使用分号分隔
Always Block Flash	选中时，所有的 flash 都会被拦截，无论来自哪里
Block X-Domain Flash	选中时，拦截所有跟所在网页不是同一个主域的 flash。来源网页根据 Referer 判断

续表

AutoTrim to 400 sessions	选中时，扩展会自动对 Web Sessions 列表进行截断，控制其中包含的 session 的数量不超过 400。Fiddle 工具栏上的“keep only”下拉菜单的功能与此相同
Hide Blocked Sessions	选中时，被拦截的 session 会变成隐藏状态，不再在 Web Sessions 列表中显示

右击 Web Session 列表的某个 Session，然后在上下文菜单中选中 Block this Host，可以很方便地把这个 session 的主机（host）添加到拦截主机列表中。

开发该扩展是为了演示如何使用 Fiddler 扩展模型（Fiddler 下载中包含其源代码），因此它提供的功能在 Fiddler 其他组件提供的功能中也会提供。

6.5　TRAFFIC DIFFER

Traffic Differ 扩展组件可以对比捕获的两套流量的不同之处。这个功能可能很有用，比如有个客户下载某个 SAZ 文件遇到了问题，而下载另一个 SAZ 文件却能工作良好。

可以用 Traffic Differ 加载这两个文件，然后整体比较二者。另外，你还可以从当前的 Web Session 列表拖拽任意 Session 到 Session 列表，并比较这些 Session。

两个并排的 Session 列表会显示每个 Session 的状态码、URL、响应大小以及响应体的哈希值，如图 6-6 所示。

图 6-6

要比较两次捕获的不同 Session，在每个列表中分别选中一个 Session，然后点击 Compare Selected Session 按钮。此时会启动文件比较工具，比较请求和响应头以及 body 中的文本内容。

6.6 FIDDLERSCRIPT 编辑器

Syntax Highlighting（语法高亮）扩展组件包会在主 Fiddler UI 中添加新的 FiddlerScript 选项卡，并提供功能类似的独立的应用程序 Fiddler2 ScriptEditor。它还包含一个 SyntaxView Inspectors，在之前的 Inspector 一章中提到过。

6.6.1 FiddlerScript 选项卡

借助 FiddlerScript 选项卡，可以直接在 Fiddler UI 中轻松地查看和更新 FiddlerScript 规则。在选项卡的上方是一些 UI 控制项，参见表 6-2。

表 6-2 UI 控制项

Save Script 按钮	将修改保存到脚本文件中
Go to 下拉框	该下拉列表中给出了 FiddlerScript 中的主要方法。在下拉列表中选中一项，会滚动到选定的方法处。 选中“*FiddlerScript Reference”选项，将打开 FiddlerScript 网页版帮助页面. 隐藏功能：双击“Go to”文本标签本身，脚本源代码会折叠到定义处
Find...查找框	在脚本中搜索指定的文本
ClassView 按钮	这个按钮可以打开或者关闭 FiddlerScript 的 ClassView 侧边栏

位于这些控制项下方的是可识别语法的源代码编辑器，其中会显示当前打开的 FiddlerScript 文本。这个编辑器提供了上下文敏感的代码补全功能，在键入时会以弹出窗口的方式显示可用的属性、变量和方法，如图 6-7 所示。

图 6-7

在左侧空白处有一个黄色条形栏，标识修改过的代码，如图 6-8 所示。

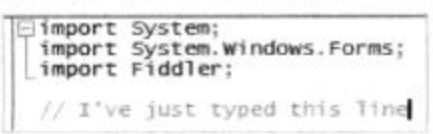

图 6-8

修改保存之后，条形栏会成变绿色，如图 6-9 所示。

```
import System;
import System.Windows.Forms;
import Fiddler;

// I've just typed this line
```

图 6-9

若编译脚本时遇到脚本错误，FiddlerScript 选项卡会被激活，并滚动到错误处。

6.6.2　ClassView 侧边栏

在 ClassView 侧边栏中可以查看脚本的主要对象、属性和方法。注意，ClassView 不显示所有可用的功能，只显示脚本最常用的功能。

在树形图中点击某项，侧边栏的顶部就会显示出该项的描述信息。在某项上按 CTRL+C 键会把该项名称拷贝到剪贴板，从而很容易地将其插入到脚本中，如图 6-10 所示。

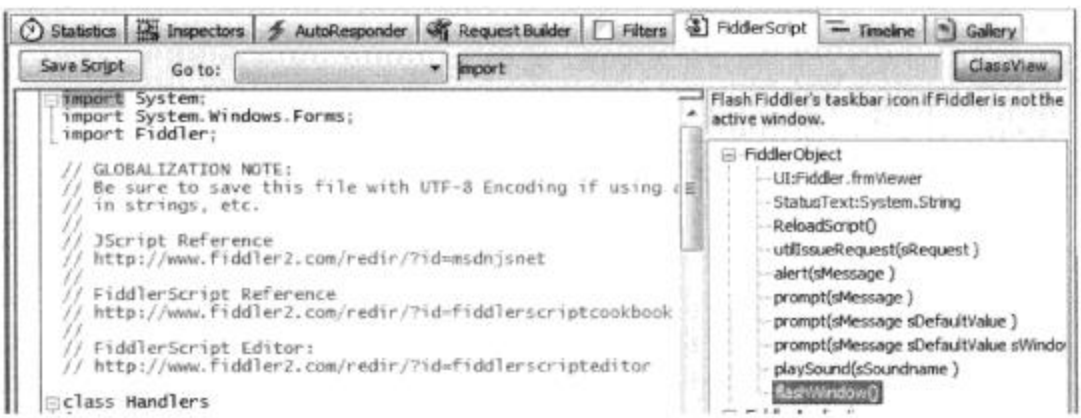

图 6-10

侧边栏的项显示的时候是根据类型着色的：

颜色	类型
黑色	方法
蓝色	属性
绿色	变量
紫色	事件

6.6.3 Fiddler2 ScriptEditor

在 Fiddler 中点击 Rules > Customize Rules 或按下 CTRL+R 键会打开 Fiddler2 ScriptEditor 程序。按下 Windows+R 可以打开 Windows 的 Run 窗口，在其中输入 fse2 可以打开 Fiddler2 ScriptEditor 程序。

ScriptEditor 是独立运行的，它提供的功能和 FiddlerScript 选项卡相同，可以很方便地运行在自己的窗口中。和 FiddlerScript 选项卡的不同之处在于，ScriptEditor 是独立运行的应用程序，它不会注意脚本编译错误，因此当 Fiddler 编译用户脚本出现问题时，它不会给出提示信息。

利用 Go 菜单中提供的命令，可以快速导航到 FiddlerScript 中重要的方法，并滚动到指定行。

View 菜单中提供的选项，可以调整源代码的显示方式，包括改变字体大小、显示行号、展开和折叠方法块。它还包含切换功能，可以显示或隐藏 ClassView Explorer 侧边栏。

Insert 菜单提供了很多预先创建好的片段，可以把这些片段添加到脚本中。要使用这些脚本片段，可以把光标放到你想要添加片段的代码处，然后点击这个菜单项。

举个例子，把光标放在 Handlers 类的左大括号里面，如图 6-11 所示。

然后点击 Insert > Context Menu Item，如图 6-12 所示。

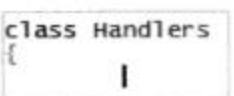

图 6-11

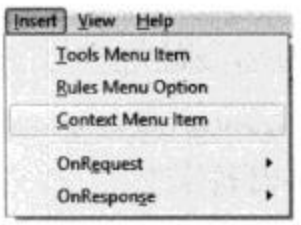

图 6-12

编辑器会插入 ContextAction 模板块，在 Web Session 的上下文菜单中添加新的菜单项，如图 6-13 所示。

图 6-13

6.7 SAZCLIPBOARD

SAZClipboard 是个简单的扩展组件，它可以把.SAZ 文件加载到其所在窗口的独立 Session 列表中。在剪贴板的窗口和 Fiddler 用户界面主窗口之间可以相互拖拽任意 session。如果使用 Composer 或 AutoResponder 特性，该功能就很方便，Composer 和 AutoResponder 都支持拖拽 Session 以便重用，如图 6-14 所示。

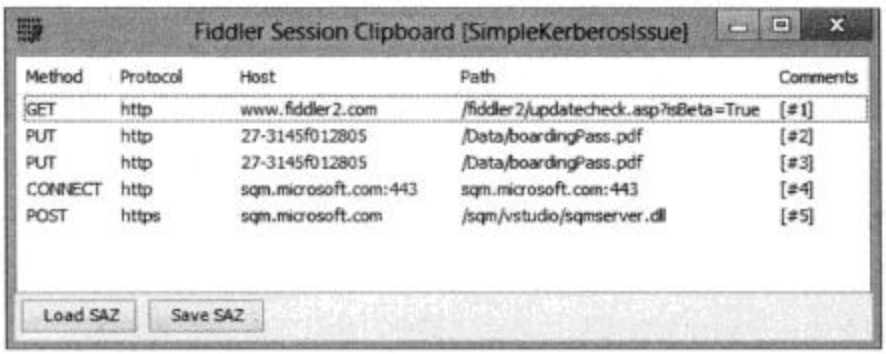

图 6-14

要使用 SAZClipboard 扩展，点击 Fiddler Tools 菜单中的 New SAZClipboard...。

可以拖拽 Web Session 列表到 SAZClipboard，或者直接使用下方的按钮加载 SAZ 文件。

6.8 ANYWHERE

大多数现代浏览器都支持地理位置定位，借助这个功能，JavaScript 可以确定用户的真实地理位置。大多数浏览器是通过查询操作系统，获取用户附近的 WiFi 接入点列表，把该列表提交到某个 Web 服务，该服务会把附近的接入点映射成经度/维度坐标，从而估测用户的真实位置。Web 服务把坐标值返回给浏览器，浏览器又返回给 JavaScript。

Fiddler 的 AnyWHERE 扩展可以修改 Web 服务返回的地理位置信息，从而“欺骗”浏览器，使得其获取到的位置和你的实际位置不符。AnyWHERE 扩展窗口支持键入当前位置，或从全球地理位置列表中选择一个你感兴趣的位置，如图 6-15 所示。

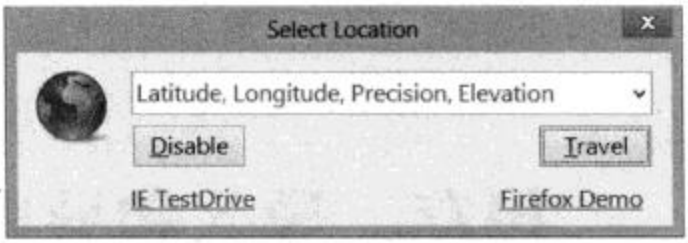

图 6-15

因为浏览器的地理定位 Web 服务查询是通过 HTTPS 传输的，所以要使该附件组件正常工作，必须启用 HTTPS 解密功能。该组件支持 IE9+、FF4、Chrome 和 Opera，但是如果浏览器的地理定位功能不是基于 Web 服务查询，该组件也会无法正常工作。例如，在包含 GPS 的 Windows 8 设备上运行 IE10 时，Windows 会使用 GPS 而不是 Web 服务来确定地理位置。

Fiddler 下载包中提供了该扩展的源代码。

第7章 保存、导入和导出数据流

7.1 Session 的 ARCHIVE ZIP（SAZ）文件

Fiddler 默认的保存格式是 Session Archive Zip (SAZ)文件。SAZ 文件是简单的 ZIP 压缩的存档文件，是通过 Fiddler 能够理解的方式构建的。把.SAZ 文件重新命名成.ZIP，便可以通过 Windows Explorer 或 WinZip 查看 SAZ 文件，看它是如何工作的。

SAZ 格式是 Fiddler 支持的最接近“无损”格式的压缩文件——它包含 Fiddler 捕捉到的每个 Session 的 header 和 body，以及元数据（包括注释、颜色标识以及计时信息）。因为 SAZ 文件包含所有的数据流，该文件会变得非常庞大，尤其是当它包含图片、音频和视频响应时。相反地，HTML、Script 和 CSS 文件压缩效果比较好，其压缩率通常是 5:1。可以把几 GB 的数据以及成千上万的 Session 保存成 SAZ 文件，但是由于这么大的文件难以加载和传输，因此通常需要使用 Fiddler 的过滤功能，对所有以给定 SAZ 文件保存的数据流进行最小化。

还可以在后期把 SAZ 文件重新加载到 Fiddler，查看其包含的 Session；这些 Session 会通过浅灰色背景色显示以便于辨别。要注意一点，Fiddler 在加载 SAZ 文件时，不会使用 Session 原有的 Session ID；当加载 SAZ 文件时，每个 Session 获取到从当前索引开始的新的 ID。在和其他人共享 SAZ 文件之前，使用 Comments 按钮或 Edit > Mark 菜单，可以标识任何感兴趣的 Session，如图 7-1 所示。

如果 Fiddler 重新加载不包含注释的 Session，它会自动添加包含原始 Session ID 的注释，如图 7-2 所示。

提示：

AutoResponder 选项卡会导入 SAZ 文件，重放其包含的响应，模拟在 SAZ 文件中捕捉到的原始数据流。

图 7-1

图 7-2

保护 SAZ 文件

SAZ 文件包含所有捕捉到的 Web 数据流，如果 Fiddler 捕捉到这些数据，可能包含敏感信息（如用户名、密码、cookie 以及账号信息）。因此，只应该在信任者和组织之间共享 SAZ 文件。

如果某个 SAZ 文件只能保存在某个不受保护的位置或者在不安全的连接中传输，可以对内容进行加密再保存，使得其他用户无法查看。要实现这点，先在弹出的 Save Session Archive 对话框中的 Save as type 下拉条中选中 Password-Protected SAZ 选项，如图 7-3 所示。

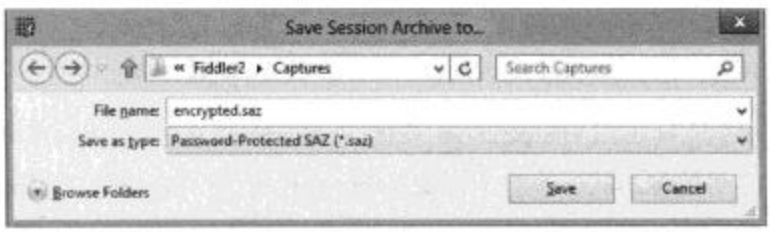

图 7-3

选中以加密格式保存后，需要输入密码，如图 7-4 所示。

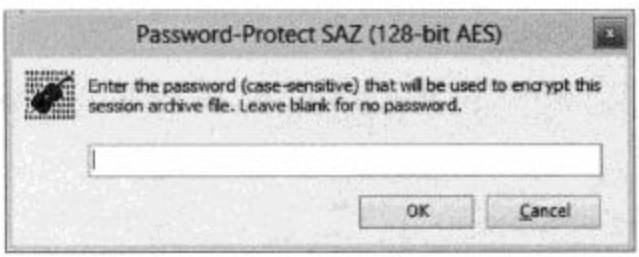

图 7-4

通过密码保护的 SAZ 文件是使用 128 位的 AES 加密方法进行加密的，AES 是一种强加密格式，HTTPS 数据流和政府、军队也使用该格式进行加密。当然，要使文件保密，必须对密码进行保密。如果和其他人共享该文件，必须通过安全的方式告诉对方密码。不要忘记密码——如果忘了密码，就没有什么方式可以重新打开 SAZ 文件（只能通过蛮力猜测）。

Preference fiddler.saz.AES.Use256Bit 可以设置成 true，让 Fiddler 使用 256 位 AES 而不是 128 位 AES 进行加密。此外，遗留的 CONFIG.bUseAESForSAZ 属性可以设置成 false，配置 Fiddler 使用遗留的 PKZIP 模糊加密方式，该加密方法速度快，但是对文件内容保护力较差。

7.2 FIDDLERCAP

SAZ 文件已经被证明对于捕捉和重现 Web 应用问题是非常有用的，我很快就发现，很多 Web 开发人员请求从新的 PC 用户捕捉 SAZ 文件。为了使得该请求更有效，我创建了 FiddlerCap。FiddlerCap 是轻量级的工具，用于支持非技术用户捕捉 SAZ 文件，使得后期专业人员可以通过 Fiddler 调试器进行分析。

FiddlerCap 提供简单的用户界面和流式工作流来捕捉 Web 数据流。和 Fiddler 本身不同，它无法修改数据流，而且不包含扩展机制。类似 Fiddler，可以通过 FiddlerCap 收集通过该技术开发的任何应用产生的 Web 数据流。FiddlerCap 是基于 FiddlerCore 库创建的，它使用了 FiddlerCore 代理实例，在 8889 端口上运行，可以捕捉 Web 数据流。

可以通过访问网站 http://fiddlercap.com 安装 FiddlerCap，该网站还提供如何使用该工具捕捉 Web 数据流的简明教程。默认情况下，安装 FiddlerCap 时会在桌面上生成文件夹，这不需要管理员权限。为使得全世界的用户可以方便地使用它，FiddlerCap 已经提供了西班牙语、法语、意大利语、日语、葡萄牙语和俄语版本。

完成安装后，该工具会自动打开。窗口会被划分成三个部分：捕捉（Capture）、捕捉选项（Capture Options）和工具（Tools），如图 7-5 所示。

图 7-5

FiddlerCap 窗口右下方的Details按钮能够用于对窗口进行扩展，显示简单的捕捉到的数据流的 Session 列表。Session 的内容无法查看，但是可以选中某个 Session，按下 Delete 键删除它。

7.2.1 Capture 窗口

Capture 窗口提供收集捕捉所需要的最小的控制项集合。按下 Start Capture 按钮，开始捕捉 Web 数据流。此时会打开新的浏览器窗口，重现该问题，虽然 FiddlerCap 会从任何需要系统代理的进程流捕捉数据流——而不仅仅是从单个浏览器窗口。用于 Firefox 的 FiddlerHook 扩展对于 FiddlerCap 不可用。要捕捉 Firefox 的网络数据流，打开浏览器的 Tools > Options > Advanced > Network > Connection Settings 选项，选中 Use System Proxy Settings 选项。

当问题重现后，点击 Stop Capture 按钮，结束数据流捕捉。最后，点击 Save Capture 按钮，选择.SAZ 文件的保存位置。默认情况下，SAZ 文件会保存到桌面。要通过密码对 SAZ 文件进行加密，选中 Save as type 下拉框中的 Password-Protected SAZ 选项，如图 7-6 所示。

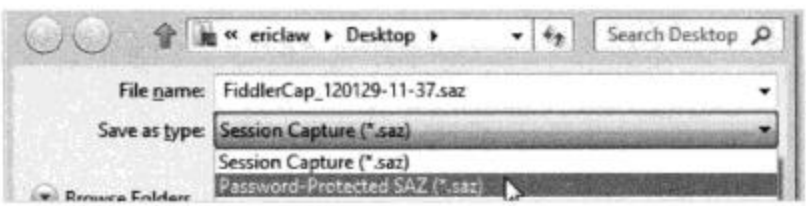

图 7-6

7.2.2 Capture Options 窗口

Capture Options 窗口控制捕捉时使用的选项。

Store binaries 复选框（默认不会选中）控制在捕捉时是否会保存二进制响应体（如图片、音频、视频和应用/octet 流下载）。忽略这些响应体（只保存响应头）可以极大降低最终的 SAZ 文件大小，虽然问题是否重现依赖于下载的内容，应该支持保存二进制选项。

Decrypt HTTPS traffic 复选框（默认不会选中）控制 FiddlerCap 是否会对 HTTPS 数据流解密。如果选中该复选框，会显示一条解释信息：

```
    --------------------------
    A note about HTTPS Decryption
    --------------------------
    HTTPS decryption will enable your debugging buddy to see the raw traffic sent
via the HTTPS protocol.
```

```
    This feature works by decrypting SSL traffic and reencrypting it using a locally
generated certificate. FiddlerCap will generate this certificate and remove it when
you close this tool.

    You may choose to temporarily install this certificate in the Trusted store
to avoid warnings from your browser or client application.
    -------------------------
```

看完该信息后，会显示 Windows Security 提示框，支持信任 FiddlerCap 根证书，如图 7-7 所示。

图 7-7

关闭 FiddlerCap 时，它会自动删除在捕捉数据流时生成的所有证书，弹出对话框让用户确定从受信任的证书库中删除根证书。

Store cookies and POSTs 选项（默认是选中的）会控制 FiddlerCap 是否保存 POST 请求体以及 HTTP 请求头的 Cookie、Set-Cookie、Set-Cookie2、Authorization 和 Proxy-Authorization。取消该选项可以在一定程度上减少捕捉量，并且减少在捕捉中保存的隐私信息量。即使没有选中该选项，捕捉的数据流显然可能包含其他隐私信息，因此只能在信任方之间共享。

Automatically discard traffic after # minutes 选项用于控制在 SAZ 文件中保存几分钟的数据流。当尝试捕捉间歇性发生的数据流时，该选项很有用。可以忽略后台运行的 FiddlerCap，老的数据流会周期性自动过期，减少捕捉的数据量以及使用的内存。如果遇到某个希望捕捉

的问题，可以只保存最后几分钟的数据流。

7.2.3 Tools 窗口

Tools 窗口提供的选项有助于重现问题。Clear Cookies 按钮会清空 IE 以及其他基于 WinINET 的所有持久 Cookie。Clear Cache 按钮会从 IE/WinINET 缓存清空所有的缓存文件，确保 FiddlerCap 可以查看所有的响应，而不是从本地缓存获取。

Clear Capture 按钮只有在 FiddlerCap 开始捕捉时才生效。它会立刻清除所有之前捕捉到的 Session。

Snapshot 按钮会对 FiddlerCap 正在运行的监视器进行屏幕截图，并添加到捕捉的 JPEG 格式的图像中。在新的 Session 中会保存该屏幕截图，其 URL 包含当前时间戳，格式为 http://localhost/Screenshot_*h-mm-ss*.jpg。

按下 Flag 按钮会弹出对话框，可以为捕捉到的数据流添加注释，如图 7-8 所示。

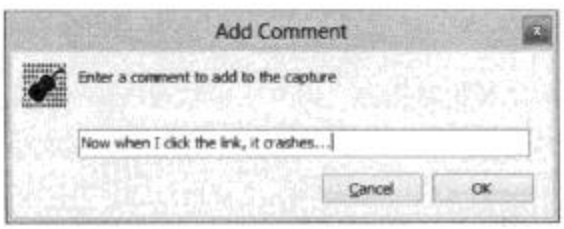

图 7-8

输入的注释文本会保存成新的 Session 的 HTTP 响应体，其 URL 是 http://USERCOMMENT/MARKER。

7.3 Fiddler 的 Viewer 模式

通常情况下，每次只能执行一个 Fiddler 实例；当 Fiddler 已经打开一个实例时，如果想打开一个新的实例，只会激活已有的 Fiddler 实例。

但是，有时再打开一个独立的 Fiddler 实例查看 SAZ 文件或生成请求是必要的。要实现这一点，Fiddler 支持通过两种方式打开新的 Viewer 实例，一是传递命令行参数-viewer，二是在 Windows Explorer 中右键点击 SAZ 文件，从上下文菜单中选中 Open in Viewer，如图 7-9 所示。

Viewer Mode 实例在状态栏中会以 Viewer 模式图标 Viewer Mode 显示，而且可以通过工具栏

前面的文本来表示。如果没有加载任何 SAZ 文件，文本会显示 FiddlerViewer，如图 7-10 所示。

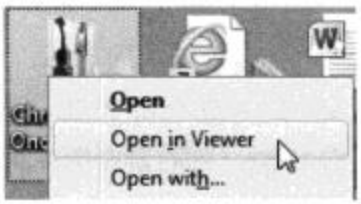

图 7-9

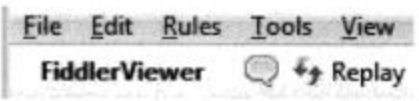

图 7-10

可以通过点击文本，输入新的标题来标识该 Viewer 实例，如图 7-11 所示。

如果把 SAZ 文件加载到实例中，文本会变成 SAZ 文件名称，显示文件的全路径的工具栏提示，如图 7-12 所示。

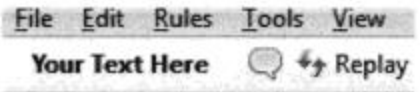

图 7-11

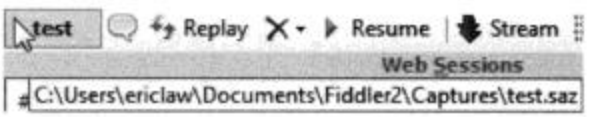

图 7-12

Fiddler Viewer 实例无法捕捉数据流，而且大多数在 Viewer 模式下所做的用户界面变化（如隐藏工具栏或对 Web Session 列表中的列重新排序）在关闭实例时会被丢弃。

7.4　导入和导出 Session

除了 Fiddler 原始的 SAZ 格式，Fiddler 调试器还提供丰富的导入导出功能（即“Transcoders”），从而可以和其他工具共享捕捉到的网络流。除了 Fiddler 绑定的 Transcoders，扩展机制支持其他开发人员添加其他格式。

Fiddler 的文件夹菜单提供导入导出功能。Import Sessions...命令支持从其他格式加载 Session，而 Export Sessions 子菜单支持导出所有的 Session 或只在 Web Session 列表中选中的 Session，如图 7-13 所示。

图 7-13

7.4.1 导入格式

当前的 Fiddler 版本包括两种导入格式：HTTPArchive 和 F12 NetXML。HTTPArchive 格式（HAR; http://groups.google.com/group/http-archive-specification）是有损的基于 JSON 的格式，很多工具支持它，包括 Firebug、Chrome 浏览器的开发者工具以及 HTTPWatch。支持 1.1 版本和 1.2 版本。

F12 NetXML 格式和 IE 9 开发者工具的 Network 选项卡导出的格式非常类似。它实际上就是 HTTPArchive 文件，这种文件通过 XML 而不是 JSON 编码。该格式不是故意要和微软的 IE 导出格式不同，而是因为原始的 HTTPArchive 格式是通过 XML 语法定义的。在说明的最上方，只是简单提了一下应该使用 JSON 格式。Fiddler 是唯一较流行的可读取 NetXML 格式的工具；可以通过 Fiddler 加载 NetXML，保存成更常见的格式（如 HAR）。注意，这两种格式都是有损压缩格式，因此有些数据会丢失（特别地，不会显示大的二进制图像）。

选中了要导入的格式后，系统会弹出对话框要求你选择要导入的文件。Fiddler 会解析选中的文件，其数据流会添加到 Web Session 列表。

开发人员都可以添加其他导入格式。目前，我在开发可以导入 Wireshark .pcap 格式和 NetMon .cap 格式文件的导入程序。

7.4.2 导出格式

Fiddler 提供了很多导出功能，支持导出数据流给其他工具或应用使用。本章的剩余部分将会描述 Fiddler 本身支持的导出格式。

HTML5 AppCache 声明文件

HTML 引入了应用缓存（Application Cache）的概念，它支持 Web 开发人员提供页面声明文件（manifest），可以缓存这些文件以供离线使用。声明文件指定浏览器应该将哪个资源下载到 AppCache 中，而哪些资源应该一直从网络获取。

声明文件只是文本文件，可以通过选中的文本编辑器创建它们。但是，该过程可能会很繁琐，需要自动化解决方案。Fiddler 的 HTML5 AppCache Manifest Exporter 使得生成声明文件变得很简单：

1）清空浏览器缓存。

2）启动 Fiddler。

3）在浏览器加载 web 站点。

4）在 Fiddler 中点击 File > Export Sessions > All Sessions....。

5）在 Select Format 窗口中，选中 HTML5 AppCache Manifest。

6）点击 Next。在 Adjust AppCache Manifest 窗口中，选中不希望在声明文件（manifest）的 CACHE 部分包含的资源；把这些资源添加到声明文件的 NETWORK 部分，如图 7-14 所示。

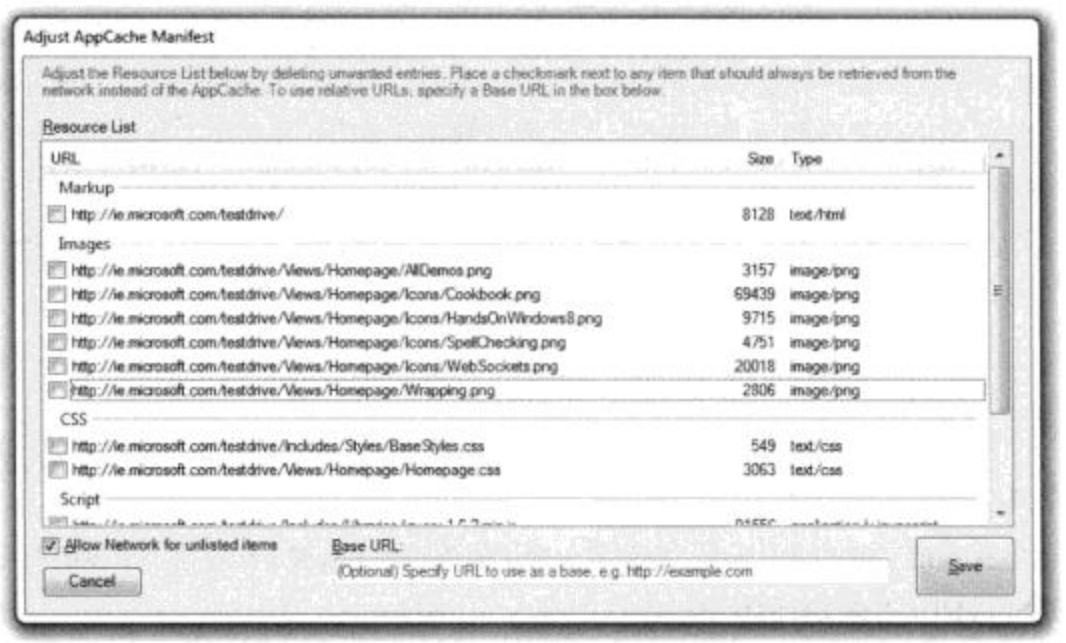

图 7-14

7）可以通过最下方的文本框，如果资源 URL 和声明相关，指定一个 Base URL。例如，在上面的例子中，我会把声明文件放在目录 http://ie.microsoft.com/testdrive 下，因而会使用它作为 Base URL。

8）点击 Save 按钮，在文本编辑器中生成并显示声明，如图 7-15 所示。

9）如果对生成的声明满意，就把它保存到 Web 服务器的合适位置。确保将 Web 服务器配置为返回包含 Content-Type: text/cache-manifest 的声明文件类型。

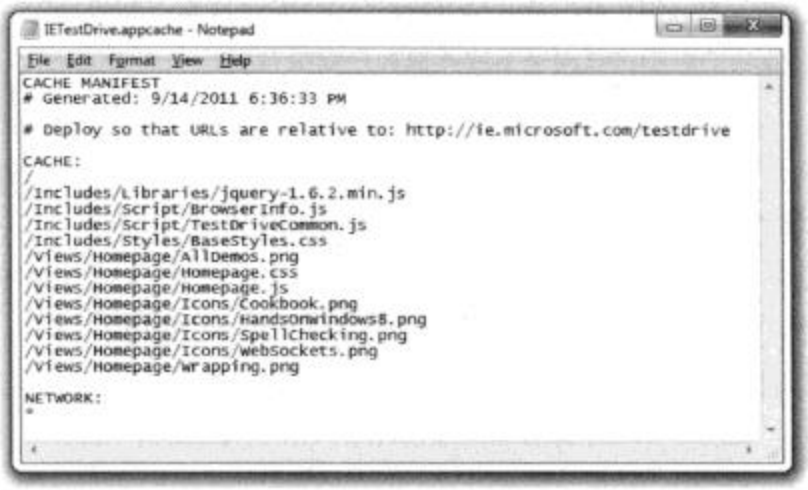

图 7-15

10）在 Web 页面，确保页面会以标准模式运行（如使用 HTML5 文档格式），在指向应用的声明文件的 HTML 元素中添加 Manifest 属性，如图 7-16 所示。

```
<!DOCTYPE html>
<html manifest="app.manifest" >
<head>
```

图 7-16

记住，当浏览器使用 AppCached 内容，会从缓存中重用这些资源，而不会从网络获取（直到缓存失效）。因为内容是本地缓存的，在 Fiddler 中，你不会看到重新加载该内容的请求。如果你希望使用 Fiddler 修改这部分内容，需要清空浏览器缓存，从而强制浏览器在下一次使用时重新从网络加载内容。

HTTPArchive v1.1 和 v1.2

HTTPArchive 格式是一种 JSON 格式，它包含 Web Session 的 header、body 和时间信息。1.1 版本和 1.2 版本之间的主要区别是新版本支持包含小的二进制 body。

默认情况下，Fiddler 会保存最多 1MB 大小的文本类型的 body，而二进制类型的 body 最多可以保存 32768 字节；可以通过设置相应的 preference 对这些值进行修改。

```
fiddler.importexport.HTTPArchiveJSON.MaxTextBodyLength
fiddler.importexport.HTTPArchiveJSON.MaxBinaryBodyLength
```

如果 body 太长了，它不会以文件形式保存，相反地，在 Session 中会添加一条注释，表示忽略该 body。

MeddlerScript

Meddler (http://webdbg.com/Meddler/)是一个 HTTP 数据流生成工具，我开发它是为了支

持创建小的、自包含的 HTTP 数据流。Meddler 本质上是个“脚本 Socket”，它支持非常底层的 socket 操作，该功能对于调试浏览器或其他客户端很有用。

当你把 Web Session 导出到 MeddlerScript 时，生成的.MS 文件可以加载到 Meddler 中，从而可以“重放”数据流。从概念上讲，这和使用 Fiddler 的 AutoResponder 选项卡重放之前从 SAZ 文件捕捉到的数据流非常类似，但是 Meddler 脚本更便于实现自动化，而且支持在底层做一些调整（如“在发送 header 和发送 body 的第一个字节之间等待 300ms，然后每隔 10s 以每个分块 2048 字节写 body”），而如果使用 Fiddler 本身则很难实现这一点。

除非你是构建浏览器或其他的 HTTP 客户端，否则导出 MeddlerScript 很可能对你没有什么用处。

原始文件（Raw Files）

Raw Files Exporter 支持把每个响应体保存到磁盘文件。当把媒体内容导入到磁盘时，该功能非常有用。举个例子，假设你浏览了照片网站，在 Web Session 列表中收集了一批照片。你可以使用 Fiddler 的过滤器和其他功能，只选中感兴趣的照片（如超过 20k 的 JPG 文件）。然后，可以选择 File > Export Sessions > Selected Sessions，在 Select Export Format 选项框选中 Raw Files。

File Exporter 窗口支持配置导出过程，如图 7-17 所示。

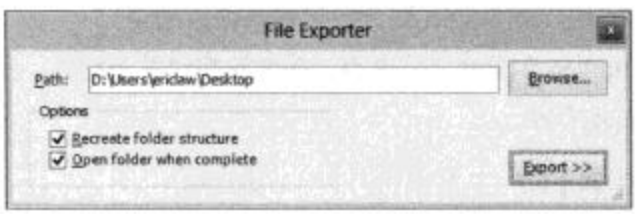

图 7-17

Path 选项框支持选中要创建的新文件夹所在的根路径。该文件夹会自动命名为\Dump-*MonthDay-Hour-Minute-Second*\。

Options 选项框包含两个复选框。Recreate Folder Structure 会为每个资源基于主机名和文件路径创建子文件夹。通过它模拟网站在本地磁盘的层次结构。

Open folder when complete 选项会打开 Windows Explorer，显示导出的文件所在的根目录。

点击 Export >>按钮可以启动导出。

除了导出媒体文件，Raw Files Exporter 还可以方便地为捕捉到的 Web 站点创建本地磁盘的镜像文件。然后，可以拖拽文件夹内容到 AutoResponsder 选项卡，Fiddler 会基于接收到的

请求重放本地内容镜像。当你想用其他工具（如 Expression Web 或 Visual Studio）在本地编辑 HTML 内容时，该功能对于调试 Web 站点是非常方便的。

Visual Studio WebTest

生成 Visual Studio .WebTest 文件后，便可以使用 Visual Studio Web Test 产品重新提交之前捕捉到的请求，该功能可以用于功能测试或本地测试。在 Visual Studio 2008 以及更新版本中提供了 Visual Studio Web Test 工具。

WCAT Script

诚如该工具的 Web 站点所描述的：

```
Web Capacity Analysis Tool (WCAT) is a lightweight HTTP load generation
tool designed to measure the performance of a web server within a controlled
environment. WCAT can simulate thousands of concurrent users making requests
to a single web site or multiple web sites. The WCAT engine uses a simple script
to define the set of HTTP requests to be played back to the web server.
```

Fiddler 的 WCAT Exporter 可以很容易为 WCAT 生成请求脚本，可以在服务器重新执行这些请求，校验其处理负载的能力。

可以在 http://fiddler2.com/r/?WCAT 下载 32 位或 64 位的 WCAR 安装文件。

第8章 FiddlerScript

8.1 使用 FiddlerScript 扩展 Fiddler

最早版本的 Fiddler 没有任何扩展功能，用户只能使用其提供的基础功能。很快我就意识到自己永远都无法满足 Fiddler 用户为解决各种问题而对该工具提出的各种需求。

Fiddler 早期版本的局限性主要在于只提供了一个过滤器（Hide Images），流量高峰期间很容易出现过载。我计划在 Fiddller 中创建一个提供过滤功能的用户界面，包含很多下拉列表框和文本框，用户基于其中提供的布尔值来过滤流量。这种方法存在两个明显问题：首先，它需要我做很多烦人的 UI 开发；其次，我相信很多用户还是会不满意。高级用户觉得使用 UI 控件构建复杂的查询很繁琐，而新用户在尝试创建涉及嵌套 AND、OR 和 NOT 操作符的复杂查询时很容易感到迷茫。

幸运的是，“懒惰”往往会孕育出更好的工程思想。我当时想了一晚：“在 Fiddler 代码中构建复杂的过滤表达式对我来说轻而易举。是不是也可以让用户自己写代码实现过滤？”这个想法让我茅塞顿开，很兴奋——开始想到的是使用.NET 框架在应用中构建脚本引擎，而且把应用的对象模型开放给脚本引擎也很容易。在 2003 年年底给 Fiddler 添加了几行代码后，它的功能就变得强大多了。

虽然 Fiddler 在过去 9 年中引入了不少新功能，包括 Fiddler 扩展和很多内置的过滤功能，但 FiddlerScript 引擎依然是 Fiddler 的一个不可或缺的功能特性。在学习 FiddlerScript 的过程中，相信你会获益良多，对 Fiddler 的了解也会更上一层楼。

8.1.1 关于 FiddlerScript

Fiddler 在处理每个 Session 时，脚本文件 CustomRules.js 中的方法都会运行，该脚本使得