```
document.getElementById('cat-photo').onclick = function() {
  document.getElementById('cat-para').innerHTML += ' Meow!';
};
```

这里的第一行添加了一个事件处理程序，它会在单击猫咪照片时运行。这个事件处理程序代码的定义在第二行，它告诉浏览器在段落上添加文本Meow!。由于被定义为事件处理程序，代码只会在单击图片事件发生时才会运行。请注意，这个脚本通过ID（cat-photo和cat-para）分别引用了照片和段落。HTML元素可以被赋予ID，这使得我们能轻松地以编程方式来引用它们。只有把这些ID添加到HTML，脚本才能工作。下面是更新后的HTML，它引用了脚本（名为cat.js）并添加了所需的ID：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>A Cat</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <script src="cat.js"></script>
  </head>
  <body>
    <h1>Thoughts on a Cat</h1>
    <p id="cat-para">This is a cat.</p>
    <img id="cat-photo" src="cat.jpg" alt="cat photo">
  </body>
</html>
```

在脚本代码被保存为cat.js且HTML如上所示更新后，重新加载该页面并单击会在段落上附加Meow！的猫咪照片。如果我们多次单击该照片，最后就会得到如图12-7所示的内容。

图12-7 运行附加文本的JavaScript代码后的示例网页

注意

请参阅设计40用JavaScript更新网页。

JavaScript可以用于构建在网络浏览器上运行的完整的应用程序。前面的例子只是对其功能的一个尝试。规范ECMAScript对JavaScript进行了标准化。各种浏览器实现的脚本引擎都尝试遵守全部或部分的ECMAScript标准，该标准会定期更新。

12.2.4 用JSON和XML构造网络数据

网站并不是网络上唯一可用的内容类型。网络服务通过HTTP提供数据，并希望以编程方式与之交互。这与返回HTML（以及相关资料）的网站相反，那些网站的意图是通过网络浏览器供用户使用。大多数最终用户从不直接与网络服务交互，尽管我们使用的网站和应用程序通常是由网络服务提供支持的。

想象一下你运营着一个网站，上面有关于在你的城市演出的本地乐队的信息。这个网站包含每个乐队的简介——乐队成员、背景、乐队演出地点等。最终用户可以访问你的网站并轻松了解他们喜欢的音乐家的信息。现在，假设一位应用程序开发人员找到你，他想在自己的应用程序中包含出自你网站的最新信息。但是，这个应用程序有自己的呈现方式，且完全不同于你的网页——这名开发人员并不想只在应用程序中显示你的网页。他们需要一种方法来获取你网站的基础数据。他们可能会尝试编程读取你的网页并提取相关信息，但这个过程既复杂又容易出错，特别是如果你的网站布局还发生了变化。

你可以通过以下方式让这位开发人员的工作轻松很多：向他提供网络服务，以HTML之外的格式显示你网站的数据。尽管HTML确实提供了某种结构，但它的结构描述的是文档（标题、段落等），而对文档所引用的数据类型说明甚少。HTML对人类阅读者而言是有意义的，但对软件而言则是难以解析的。那么，网络服务应该用什么格式来表示乐队数据结构呢？当前网络服务使用的最常见的通用数据格式是XML和JSON。

扩展标记语言（eXtensible Markup Language，XML）自20世纪90年代以来就已经存在，它是一种在网络上交换数据的流行方法。和HTML一样，它是基于文本的标记语言，但它没有一组预定义的标签，XML允许自定义标

签来描述数据。在我们虚构的乐队信息服务中，我们可以定义一个<band>标签和一个<concert>标签。我们来看一下使用XML描述的假想乐队：

```
<band name="The Highbury Musical Club">
  <bandMembers>
    <member name="Jane Fairfax" instrument="Piano" />
    <member name="Emma Woodhouse" instrument="Guitar" />
    <member name="Harriet Smith" instrument="Percussion" />
    <member name="Frank Churchill" instrument="Vocals" />
  </bandMembers>
  <upcomingConcerts>
    <concert location="Donwell Abbey" date="August 14, 2020" />
    <concert location="Hartfield" date="November 20, 2020" />
  </upcomingConcerts>
</band>
```

如你所见，特定的XML标签和它们的属性是按照我们的需求量身定制的，而开始标签、结束标签和树形层次结构等通用结构则遵循了类似于HTML的模式。XML的灵活性（即标签可以任意定义）意味着XML的生产者和消费者都需要就预期标签及其含义达成一致。HTML也是如此，但在使用HTML时，是各方就标准达成一致。对于XML，只有通用格式是标准化的，而特定标签可以不同。

XML是在网络上共享数据的一种流行方法，很多网络服务使用XML作为它们呈现数据的主要方式。但是，XML很烦琐，正确地解析它可能会很棘手。

JavaScript对象表示法（JavaScript Object Notation，JSON）与XML一样，是一种以文本格式描述数据的方法。JSON避免使用标记标签，而是采用一种类似于描述对象的JavaScript语法的样式，由此得名。在JSON中，对象被放在大括号{和}中，数组（对象的集合）被放在方括号[和]中。它的语法比XML更简洁，这有助于减少在网络上传递的数据量。JSON的流行始于21世纪10年代，当时它开始取代XML成为新网络服务的首选数据格式。下面是用JSON描述的同一个假想乐队：

```
{
  "name": "The Highbury Musical Club",
  "bandMembers": [
    {
      "name": "Jane Fairfax",
      "instrument": "Piano"
    },


    {
      "name": "Emma Woodhouse",
      "instrument": "Guitar"
    },
    {
      "name": "Harriet Smith",
      "instrument": "Percussion"
    },
    {
      "name": "Frank Churchill",
      "instrument": "Vocals"
    }
  ],
  "upcomingConcerts": [
    {
      "location": "Donwell Abbey",
      "date": "August 14, 2020"
    },
    {
      "location": "Hartfield",
      "date": "November 20, 2020"
    }
  ]
}
```

XML和JSON都忽略多余的空格，所以就像使用HTML一样，我们可以删除多余的空格符、制表符和换行符，而不会影响到数据的解释。这样做会产生相当紧凑的数据呈现形式，特别是在使用JSON的时候。

XML和JSON不是用于在网络浏览器中直接呈现的格式。在某些浏览器中打开JSON和XML内容可能导致浏览器显示一些东西（可能是数据的轻量级格式化版本），但实际上JSON和XML并不打算被浏览器直接使用。它们意在被代码阅读，而这些代码又用数据做一些有用的事情。在我们的例子中，代码可能是智能手机应用程序，该程序显示正在附近演出的乐队的信息。此外，代码也可能是客户端JavaScript，它把JSON转换成HTML以供浏览器显示。

12.3 网络浏览器

现在我们已经介绍了用于描述网络的语言，接下来我们来看网络客户端的软件，即网络浏览器。第一个网络浏览器被称为WorldWideWeb（不要与本章主题搞混）。它由Tim Berners- Lee在1990年开发。第一个浏览器是第一个网络服务器CERN httpd的客户端。几年后，WorldWideWeb被Mosaic取代，Mosaic是一种帮助普及网络的浏览器。之后发布的主要浏览器是Netscape Navigator，它也有大量的追随者。1955年，微软发布了它们的第一款浏览器Internet Explorer，它是Netscape Navigator的直接竞争者，成了那个时代占主导地位的浏览器。如今，浏览器的格局发生了巨大的变化，在撰写本书时，占主导地位的浏览器是Google Chrome、Apple Safari和Mozilla Firefox。

12.3.1 渲染页面

现在让我们来看看网络浏览器渲染页面的过程。对网站的典型访问从对网站默认页面（如http://www.example.com/）的请求或者对该网站特定页面（如http://www.example.com/animals/cat.html）的请求开始。用户可以直接在地址栏中输入这个URL，也可以通过链接到达这个URL。不管是哪种情况，浏览器都会请求这个特定URL上的内容。假设这个URL是有效的且以网页形式呈现，那么服务器将使用HTML进行响应。

网络浏览器必须返回HTML，并生成网页的DOM表示。HTML可以包含对其他资源（比如图像、脚本和样式表）的引用。这些资源都有自己的URL，浏览器对每个资源发出单独的请求，如图12-8所示。

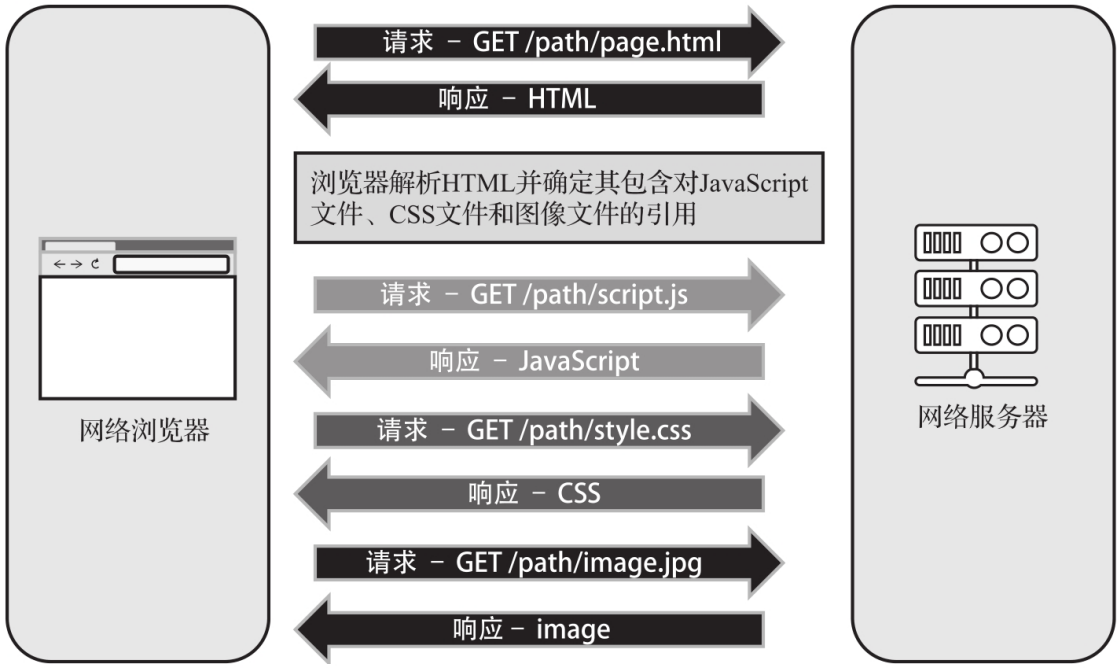


图12-8 网络浏览器请求页面及其引用的资源

在浏览器检索到该网页的各种资源后，它就会显示HTML，用指定的CSS来确定合适的表现形式。所有脚本都交给JavaScript引擎运行。JavaScript代码可以立即对页面进行修改，也可以注册事件处理程序，以便之后在某个事件发生时运行。JavaScript代码还可以从网络服务请求数据并使用该数据更新页面。

网络浏览器包含一个渲染引擎（用于HTML和CSS）、一个JavaScript引擎和一个把内容联系起来的用户界面。尽管用户界面提供了浏览器自身的外观和感觉（比如后退按钮和地址栏的外观），但决定网站表现形式与行为（包括网页如何布局以及如何响应输入）的却是渲染引擎和JavaScript引擎。由于每个渲染引擎和JavaScript引擎处理事情的方式略有差异，当用不同浏览器访问网页时，其外观和行为可能会有所不同。理想情况下，所有的浏览器应该以同样的方式呈现内容，这完全符合网站开发人员的预期，但情况并非总是如此。在撰写本书时，只有3款主要的渲染引擎处于积极开发过程中：WebKit、Blink和Gecko。

WebKit是Apple Safari浏览器的渲染引擎和JavaScript引擎。它也用于iOS App商店中的应用程序，因为苹果公司要求所有显示网络内容的iOS应用程序都使用这个引擎。Blink是WebKit的一个分支，它是Chromium开源项目的渲染引擎，该项目还包括了V8JavaScript引擎。Chromium是Google Chrome和Opera的基础。2018年12月，微软宣布Microsoft Edge浏览器也将基于Chromium，它选择停止开发自己的渲染引擎和JavaScript引擎。这就只剩下一个主流浏览器没有追溯到WebKit——Mozilla Firefox，它有自己的Gecko渲染引擎和SpiderMonkey JavaScript引擎。

注意

当开发人员复制了一个项目的源代码，然后对这个副本进行修改时，就会产生软件分支。这允许原始项目和分支项目作为各自独立的软件而共存。

12.3.2 用户代理字符串

网络浏览器正式的技术性术语是"用户代理"（user agent）。这个术语也可以用于其他软件（任何代表用户的软件），但这里我们专门讨论的是网络浏览器。这个术语出现在关于网络的技术文档中，尽管它很少在正式交流之外使用。在实践中，使用这个术语的一个地方是"用户代理字符串"。当浏览器向网络服务器提出一个请求时，它通常会包括一个被称为User-Agent的标题值，用于描述浏览器。例如，下面是Chrome（版本71）在Windows 10上发送的一个用户代码字符串：

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/71.0.3578.98 Safari/537.36
```

这似乎是矛盾的。它们究竟是什么意思？

Mozilla/5.0是早期网络的延续。Mozilla是Netscape Navigator的用户代理名称，许多网站专门在用户代理字符串中寻找"Mozilla"，作为把其网站的最新版本发送到浏览器的指示符。当时，其他浏览器也想要得到网站的最佳版本，所以它们把自己标识为Mozilla，尽管它们根本就不是Mozilla。快

进到现在，基本上每个浏览器都把自己标识为Mozilla，我们发现用户代理字符串的那个部分完全没有任何意义。

（Windows NT 10.0;Win64;x64）指定了浏览器运行的平台。

接下来是渲染引擎，本例中是AppleWebKit/537.36。如前所述，Chrome的Blink引擎是WebKit的分支，而且仍然把自己标识为WebKit。后面的文本（KHTML,like Gecko）只是对其的进一步阐述：KHTML是WebKit所基于的一个传统引擎。

现在，我们得到了实际的浏览器名称和版本：Chrome/71.0.3578.98。

最后，我们尴尬地提到苹果的浏览器Safari/537.36，其中包括了一些给予Safari特殊待遇的网站。通过包含这个文本，Chrome试图确保那些网站发给它的内容与Safari接收到的内容相同。

这是一种相当复杂的识别Chrome的方法，但其他浏览器也做着相同的事情以确保与各种网站的兼容性。这种复杂性是历史上在不同的浏览器和网站上分散功能的副作用，这些浏览器和网站想根据特定的浏览器发送其内容的定制版本。浏览器的发展使得当前浏览器功能的变化减少。但是，许多网站并没有发展，它们仍然为特定的浏览器发送定制的内容，这迫使现代浏览器继续欺骗旧网站，让其相信它们正在与不同的浏览器进行通信。

12.4 网络服务器

到目前为止，我们主要关注的是网络客户端使用的技术。网络浏览器使用三种常见语言：HTML、CSS和JavaScript。那么，网络服务器端呢？哪些语言和技术被用来为网络服务器提供支持？简单来说，所有的编程语言或技术都可以用于网络服务器，只要该技术可以通过HTTP进行通信并能以客户端理解的格式返回数据即可。

一般来说，网站被设计成静态的或动态的。静态网站返回提前构建的HTML、CSS或JavaScript。通常，这种网站的内容存储在服务器上的文件

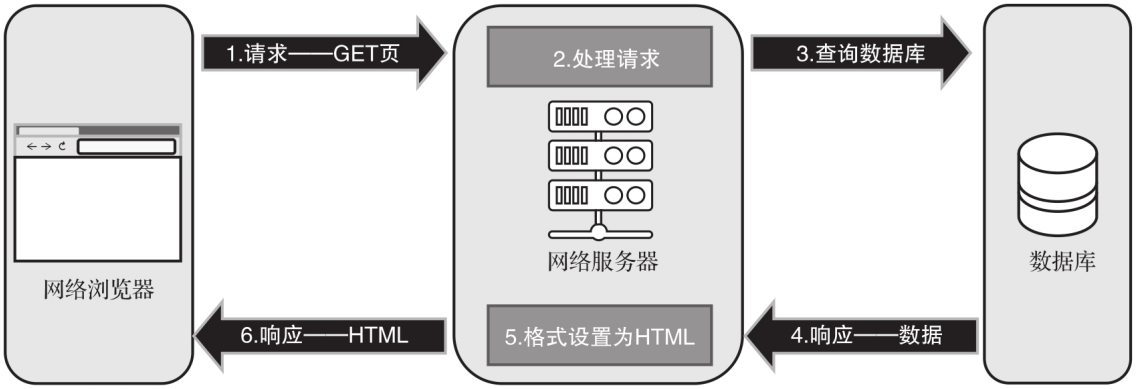
中，服务器只返回这些文件的内容而无须修改。这意味着任何所需的运行时处理都必须在浏览器运行的JavaScript中实现。动态网站在服务器上执行处理，当请求到来时生成HTML。

在网络发展的早期，几乎所有的东西都是静态的。页面是简单的HTML，几乎没有交互性。随着时间的推移，开发人员开始添加在网络服务器上运行的代码，允许服务器返回动态内容或接受用户上传的文件及提交的表单。这种趋势仍在持续，在服务器响应之前，请求要经过服务器端的处理已经成为一种普遍现象。

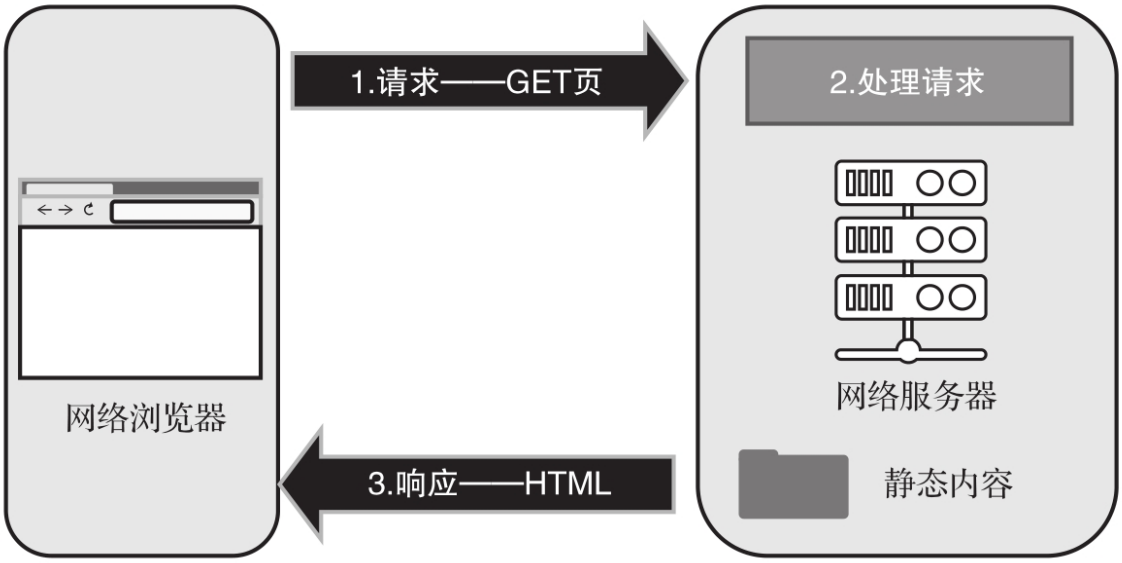
让我们来看看动态网站上的服务器端处理通常是如何工作的，如图12-9所示。假设图12-9中呈现的动态网站是一个博客。浏览器对博客文章发出请求。当网络服务器接收到这个对博客文章的请求后，它会读取被请求的URL并确定需要生成HTML。然后，服务器上的代码查询数据库（该数据库可能在这个服务器上，也可能在其他服务器上），检索相关博客文本数据，把文本格式设置为HTML，再用该HTML响应客户端。这个方法很有用，因为它允许把网站的内容与网站的代码分开管理，但动态网站也有一些缺点。服务器上日益增加的复杂性意味着更多的设置工作、更慢的运行时响应、服务器上潜在的沉重负载，以及在安全问题上不断增加的风险。

最近的一个趋势是尽可能回归静态网站。图12-10给出了静态网站上的页面请求过程。

如图12-10所示，与动态网站相比，静态网站的服务器端处理是简化了的。静态网站上的服务器端只返回与被请求URL相匹配的静态文件。内容是已经构建好的，服务器不需要检索原始数据并对其进行格式化。降低服务器端的复杂性通常意味着更简单、更快、更安全的网站。



▲图12-9 动态网站处理请求的典型过程



▲图12-10 静态网站处理请求的过程

一定要搞清楚，在这个上下文中，“静态”和“动态”是来自服务器的角度，不是用户的角度。静态网站的内容与服务器上文件的内容相同，而动态网络的内容是在服务器上生成的。这两个词不是在描述用户对网站的体验，例如，网站是否有交互性或者内容是否是自动更新的。这些体验可以在浏览器中用JavaScript来实现，有时与独立的网络服务一起使用，无论网站自身是静态的还是动态的。

如果你托管的是静态网站，那么你需要的就只有网络服务器软件，它能响应对静态文件的请求并提供这些文件的内容。无须自定义代码。许多软件包和在线服务都可以用于托管静态网站。一般，服务于静态网站的软件被配置成指向服务器上的文件目录，当对某个文件的请求到来时，服务器只需返回该文件的内容。例如，如果网站上的文件名为example.com，它位于服务器上的/websites/example目录中，那么对http://example.com/images/cat.jpg的请求就会映射到/websites/example/images/cat.jpg。网络服务器只需从本地目录中读取匹配的文件，并把文件中包含的字节返回给客户端即可。设计37到设计40开发的网站就是静态网站的一个例子。

如果你正在构建动态网站或网络服务，则可以使用现有软件来管理内容并提供动态网页，也可以编写自己的自定义代码来生成网络内容。假设你在编写自定义代码，你会发现，与网络开发的客户端相比，服务器端的情况完全不同。任何编程语言、操作系统和平台都可以用于网络服务器。只要网络服务器通过HTTP响应并以客户端理解的格式返回数据，那么就可以进行任意操作！客户端不关心用什么技术生成HTML或JavaScript，它只要求响应的格式是它能处理的。

由于对于客户端来说，服务器端用什么技术并不重要，因此对希望编写在服务器端运行的代码的开发人员来说就有了许多选择。客户端网络开发仅限于HTML、CSS和JavaScript三种，而服务器端网络开发则可以使用Python、C#、JavaScript、Java、Ruby、PHP等。服务器端网络开发常常包含某种数据库的接口开发。就像任何编程语言都可以用于服务器端网络开发一样，任何类型的数据库都可以用于服务器端网络开发。

12.5 总结

本章介绍了网络——一组分布式的、可寻址的、链接的资源，由HTTP通过互联网进行传递。你学习了如何用HTML构建网页、如何用CSS设置样式，以及如何用JavaScript编写脚本。我们研究了网络浏览器和网络服务器，浏览器被用于访问网络上的内容，服务器是托管网络资源的软件。第

13章将介绍现代计算机的一些趋势，你将有机会完成一个最终设计，它把本书中的各种概念联系在一起。

设计36：查看HTTP流量

本设计中，你将使用Google Chrome或Chromium来查看网络浏览器和网络服务器之间的HTTP流量。你可以使用Windows PC或Mac上的Chrome，也可以使用Raspberry Pi上的Chromium网络浏览器。下面的步骤假设你使用的是Raspberry Pi，不过Windows PC或Mac上的步骤与之类似，差别在于使用的是Chrome而不是Chromium。

1）如果你没有在Raspberry Pi上使用图形桌面，请现在进行切换。与前面的设计不同，本设计不能从终端窗口完成。

2）依次单击Raspberry（左上角的图标）→Internet→Chromium Web Browser。

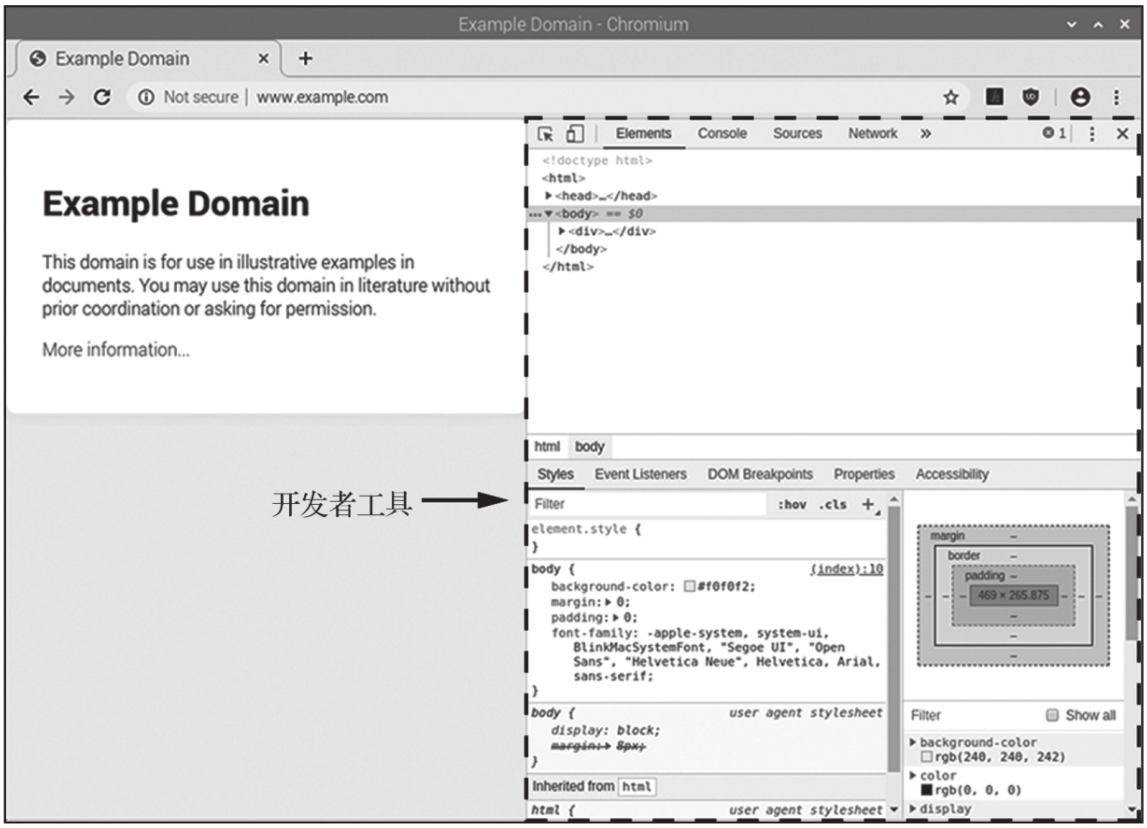
3）访问一个网站，比如http://www.example.com。

4）按<F12>键（或<Ctrl+Shift+I>快捷键）打开开发者工具（DevTools），如图12-11所示。

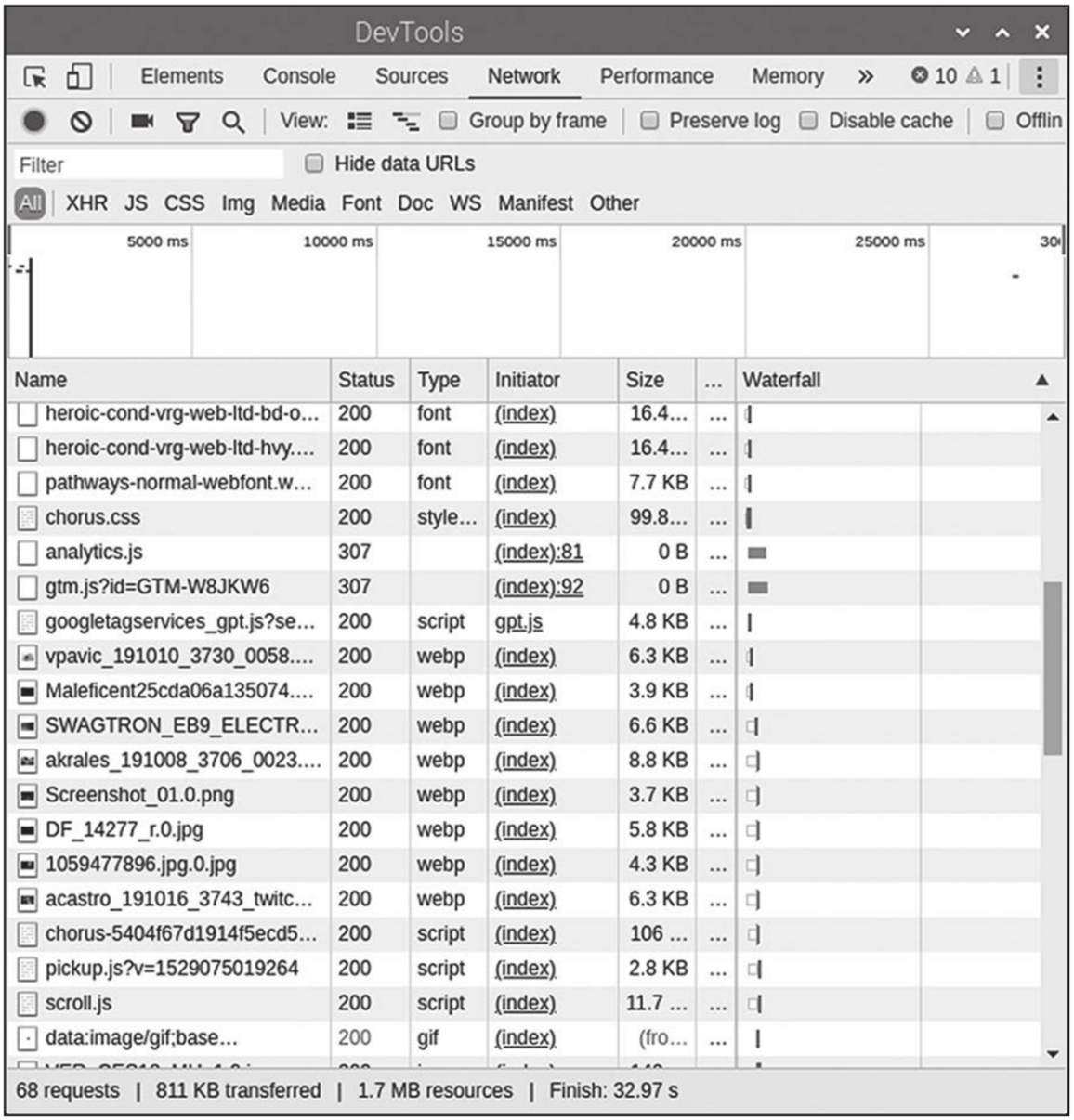
5）在DevTools菜单上选择Network菜单项。

6）按<F5>键（或单击重新加载图标）重新加载页面，你将看到用于加载当前访问页面的HTTP请求。

7）如果你实际使用的是www.example.com，你可能会看到一个非常无聊的请求。如果你想看到更有趣的东西，就访问更复杂的网站并查看网络请求，如图12-12所示。



▲图12-11 Chromium中的开发者工具



▲图12-12 一个示例：用Chromium中的DevTools显示网站的HTTP流量

8）每一行都代表对网络服务器的一个请求。你可以看到被请求的资源名称，请求的状态（200表示成功）等。

9）你可以单击每一行，查看请求的具体信息，比如标题和返回的内容。