我们可以把互联网比作一张大网，而爬虫（即网络爬虫）便是在网上爬行的蜘蛛。如果把网的节点比作一个个网页，爬虫爬到这就相当于访问了该页面，获取了其信息。可以把节点间的连线比作网页与网页之间的链接关系，这样蜘蛛通过一个节点后，可以顺着节点连线继续爬行到达下一个节点，即通过一个网页继续获取后续的网页，这样整个网的节点便可以被蜘蛛全部爬行到，网站的数据就可以被抓取下来了。

爬虫概述

简单来说，爬虫就是获取网页并提取和保存信息的自动化程序，下面概要介绍一下。

获取网页

爬虫首先要做的工作就是获取网页，这里就是获取网页的源代码。

源代码里包含了网页的部分有用信息，所以只要把源代码获取下来，就可以从中提取想要的信息了。

前面讲了请求和响应的概念，向网站的服务器发送一个请求，返回的响应体便是网页源代码。所以，最关键的部分就是构造一个请求并发送给服务器，然后接收到响应并将其解析出来，那么这个流程怎样实现呢？总不能手工去截取网页源码吧？

不用担心，Python 提供了许多库来帮助我们实现这个操作，如 urllib、requests 等。我们可以用这些库来帮助我们实现 HTTP 请求操作，请求和响应都可以用类库提供的数据结构来表示，得到响应之后只需要解析数据结构中的 Body 部分即可，即得到网页的源代码，这样我们可以用程序来实现获取网页的过程了。

提取信息

获取网页源代码后，接下来就是分析网页源代码，从中提取我们想要的数据。首先，最通用的方法便是采用正则表达式提取，这是一个万能的方法，但是在构造正则表达式时比较复杂且容易出错。

另外，由于网页的结构有一定的规则，所以还有一些根据网页节点属性、CSS 选择器或 XPath 来提取网页信息的库，如 Beautiful Soup、pyquery、lxml 等。使用这些库，我们可以高效快速地从中提取网页信息，如节点的属性、文本值等。

提取信息是爬虫非常重要的部分，它可以使杂乱的数据变得条理清晰，以便我们后续处理和分析数据。

保存数据

提取信息后，我们一般会将提取到的数据保存到某处以便后续使用。这里保存形式有多种多样，如可以简单保存为 TXT 文本或 JSON 文本，也可以保存到数据库，如 MySQL 和 MongoDB 等，还可保存至远程服务器，如借助 SFTP 进行操作等。

自动化程序

说到自动化程序，意思是说爬虫可以代替人来完成这些操作。首先，我们手工当然可以提取这些信息，但是当量特别大或者想快速获取大量数据的话，肯定还是要借助程序。爬虫就是代替我们来完成这份爬取工作的自动化程序，它可以在抓取过程中进行各种异常处理、错误重试等操作，确保爬取持续高效地运行。

能抓怎样的数据

在网页中我们能看到各种各样的信息，最常见的便是常规网页，它们对应着 HTML 代码，而最常抓取的便是 HTML 源代码。

另外，可能有些网页返回的不是 HTML 代码，而是一个 JSON 字符串（其中 API 接口大多采用这样的形式），这种格式的数据方便传输和解析，它们同样可以抓取，而且数据提取更加方便。

此外，我们还可以看到各种二进制数据，如图片、视频和音频等。利用爬虫，我们可以将这些二进制数据抓取下来，然后保存成对应的文件名。

另外，还可以看到各种扩展名的文件，如 CSS、JavaScript 和配置文件等，这些其实也是最普通的文件，只要在浏览器里面可以访问到，就可以将其抓取下来。

上述内容其实都对应各自的 URL，是基于 HTTP 或 HTTPS 协议的，只要是这种数据，爬虫都可以抓取。

JavaScript 渲染页面

有时候，我们在用 urllib 或 requests 抓取网页时，得到的源代码实际和浏览器中看到的不一样。

这是一个非常常见的问题。现在网页越来越多地采用 Ajax、前端模块化工具来构建，整个网页可能都是由 JavaScript 渲染出来的，

也就是说原始的 HTML 代码就是一个空壳，例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>This is a Demo</title>
</head>
<body>
<div id="container">
</div>
</body>
<script src="app.js"></script>
</html>
```

body 节点里面只有一个 id 为 container 的节点，但是需要注意在 body 节点后引入了 app.js，它便负责整个网站的渲染。

在浏览器中打开这个页面时，首先会加载这个 HTML 内容，接着浏览器会发现其中引入了一个 app.js 文件，然后便会接着去请求这个文件，获取到该文件后，便会执行其中的 JavaScript 代码，而 JavaScript 则会改变 HTML 中的节点，向其添加内容，最后得到完整的页面。

但是在用 urllib 或 requests 等库请求当前页面时，我们得到的只是这个 HTML 代码，它不会帮助我们去继续加载这个 JavaScript 文件，这样也就看不到浏览器中的内容了。

这也解释了为什么有时我们得到的源代码和浏览器中看到的不一样。

因此，使用基本 HTTP 请求库得到的源代码可能跟浏览器中的页面源代码不太一样。对于这样的情况，我们可以分析其后台 Ajax 接口，也可使用 Selenium、Splash 这样的库来实现模拟 JavaScript 渲染。

后面，我们会详细介绍如何采集 JavaScript 渲染的网页。本节介绍了爬虫的一些基本原理，这可以帮助我们在后面编写爬虫时更加得心应手。