第 1 章 爬虫初步

本章介绍爬虫的相关概念与基本运用，包括爬虫在大数据时代所占据的地位和发挥的作用，还有用于开发爬虫的语言 Python，以及在 Python 下的一些爬虫框架。

另外，爬虫涉及的技术领域很多，运用的技术也非常庞杂，从基本的网络访问到复杂的机器学习，可能会让初入门径者有望而却步的想法。为了能让读者对虫术保持一颗好奇的心，不至于被知识的海洋淹没，本章的最后一节提供了一份学习虫术的详细技术路线图，按照实际的学习与运用的阶段划分了需要涉猎与掌握的各种相关技术。

本章最后从实践出发，通过简单的爬虫浅尝一下虫术，希望通过实践能让读者对这门技术有一个基本的概念与认识。

1.1 爬虫与大数据

网络爬虫是一种伴随着互联网诞生与演化的“古老”的网络技术，我记得第一次听到“爬虫”这个词还是在早年的雅虎网站上（谷歌还没出生）。从那时起，搜索引擎就是运用网络爬虫在各类网络资源中爬取关键字或者一些简单的内容描述来建立网站索引目录的，爬虫技术的发展可以说是与互联网发展同步的。随着互联网进入大数据时代，爬虫技术迎来了一波新的振兴浪潮。

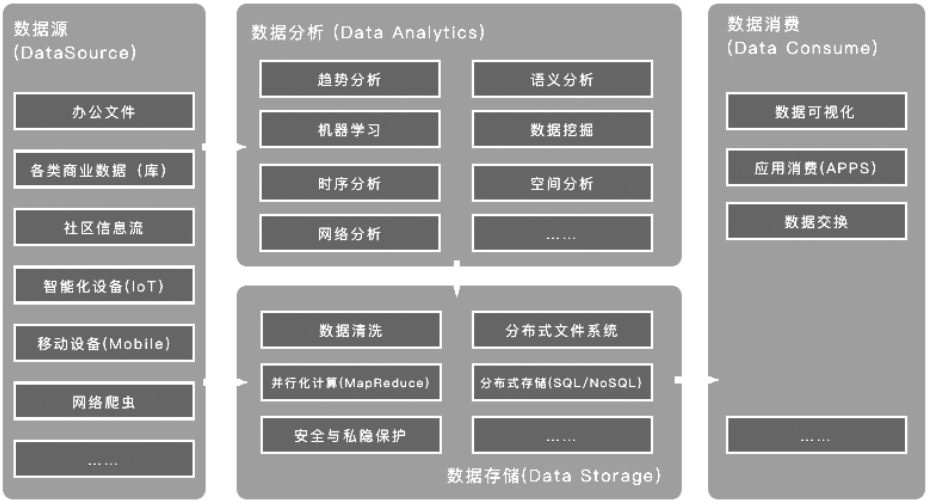
1.1.1 大数据架构

爬虫在大数据架构中有着举足轻重的位置，在对其进行详细讲述之前先来聊聊大数据。在

我的印象中，大数据（Big Data）的概念大约是从 2007 年前后开始兴起的，当时并不觉得这是很“高大上”的技术，毕竟我以前做的项目没有哪个是没有数据库的。由于我对各种技术都抱有强大的好奇心，所以从那时起，除了本职工作需要使用 SQLServer 和 Oracle，其他数据库也都喜欢研究一番，以至于对大数据的浅层次理解就是：

> 应用程序的运行数据多了自然就成为“大数据”了。

显然，这种理解是非常粗浅的，虽说数据量大是“大数据”应用的一种表面的特征，但这并不是全面的。因为“大数据”是一种采用多种技术对数据进行综合性处理的系统化简称。如果将大数据应用采用层次理论简单地分层展开，则会让我们有一个比较客观和感性的认识。下图是按照功能的作用与数据的供给对大数据应用进行分层。



大数据应用分为以下几个层次。

- 原始数据源（Data Source）——负责进行数据的采集、过滤去重、清洗、修复，按照实际应用需要对数据进行规范化处理，可以直接将数据提供给分析加工层进行深加工或将数据提供给持久层进行存储。
- 数据分析与加工层（Data Analytics）——负责对原始数据进行深加工，例如，文本语言分析、情感分析、传播路径分析、机器学习或人工智能化分析，并将数据分析结果提供给消费层使用。
- 存储层（Data Storage）——对结构化、非结构化数据进行存储。针对大型分散式应用还可以提供分布式的数据存储。
- 消费层（Data Consum）——负责对外部提供数据展现或者自动化数据供给，也就是多

维度的数据可视化和各种数据 API。

- 基础架构层（Infrastructure）——为大数据应用提供系统级的支持，例如，建立数据集群、虚化服务等。

这个层次图是我对大数据应用的一个笼统的概括，由于每个层次中的一个小部分都足以用一本书甚至是一个系列书的篇幅进行叙述，已经远远超出了本书的讨论范围。我画出此图的最重要的原因是希望读者能关注其中的“原始数据源”与“存储层”两个层次，它们除了是大数据架构中的重要组成部分，也是本书所讲述的重点——爬虫。

1.1.2 爬虫的作用与地位

在大数据架构中，数据收集与数据存储占据了极为重要的地位，可以说是大数据的核心基础。而爬虫技术在这两大核心技术层次中占有了很大的比例。为何有此一说？我们不妨通过一个实际应用场景来看看爬虫到底发挥了哪些作用？

在了解爬虫的作用之前，应该先了解其基本特性：

- **主动**——爬虫的重点在于“爬取”（Crawl），这是一种主动性的行为。换句话说，它是一个可以独立运行且能按照一定规则运作的应用程序。
- **自动化**——由于处理的数据可能很分散，数据的存留具有一定的时效性，所以它是一套无人值守的自动化程序。

企业内部爬虫

在我接近 20 年的 IT 从业生涯中，企业管理系统是我参与过的项目或产品中占比最大的。在这些项目与产品的开发过程中，我观察到很多企业内部其实有非常多的**数据处理**场景可以用爬虫技术进行处理，从而能以惊人的效率取代原有的人工化的操作。

以我近年来在电商企业内部所见为例，阿里巴巴（简称阿里）已显现出它在电子商务一统全球的实力与地位，几乎可以将电商与阿里之间划一个等号。阿里为各个店铺和商家提供了各种各样优秀的运营工具。我们会理所当然地认为电商企业内部的信息化管理程度一定很高，不是吗？然而事实恰恰相反，我见过的多数中小型的电商企业甚至是三板挂牌企业内部的信息化水平仍然非常落后，不少企业仍然依赖 Excel 这样基于大量人力为主导的表格处理。那么问题来了，为何阿里巴巴、京东这些电商平台已经提供了大量优质运营工具，而电商企业的信息化水平却很低，还需要靠劳动密集型的方式进行运营呢？首先，电商企业不会只在某一平台上开店，通常都会在多个平台同时开多个店铺以拓宽市场的销售渠道；其次，电商企业之间、电商与供货商之间缺乏统一的数据交换标准，通常只依赖于一些技术陈旧的ERP来维持日常的运营。

电商企业通常只能通过某一平台上提供的专用工具监测某些产品的价格波动和销售情况，

而无法全面、统一地了解他们所销售的产品在各大平台的具体表现如何。然而这样的需求很明显是迫切的，因为只有了解销售数据的变化才能实时调节销售的策略。我见过最多的做法就是企业安排一位专人从各大电商平台中导出运行的数据，然后合并到 Excel 中，再进行一番统计，手工做出各种统计报表作为分析依据，这种做法往往对某一个单品就得做一次！

其实，导致这种现象的原因有以下几点：

（1）缺乏统一的数据来源——这是不可调和的，因为电商运行的数据源本来就具有多样性。

（2）结构化数据与非结构化数据并存——企业间最常见的数据交互格式是 Excel，交互工具是微信和 QQ。

（3）一个数据存在多种时间版本——QQ 或者微信上的同一个文件修改多了且重复传会出现各种的 `data.xlsx、data(1).xlsx…data(n).xlsx`。

（4）数据结构可能存在随意性——Excel 文件内很少会看见用英文命名的列，甚至相同作用的列很有可能会采用不同的中文名。

（5）数据查找变得困难——在电商企业与供货商之间要找出某个时段相同的数据副本可能是一件极为可怕的事件。

我们不妨来大胆地假设一下，如果将这些事情换成让爬虫去处理，那么情况会变成什么样子呢？

（1）每天爬虫在一个固定的时间到淘宝、京东或者其他电商平台上自动下载商家当前的营业数据。

（2）完成爬取后将数据自动保存到数据库。

（3）从内网的某台 PC 的指定文件夹中下载每天从其他经销商发来的 Excel 文件，整理后保存到数据库。

（4）发现某些商品库存不足自动生成供货商规定格式的订货单，通过电子邮件发出。

（5）决策者（运营经理/老板）在手机或 PC 中通过数据可视化工具查看每天的数据统计结果，或者由爬虫系统直接生成统计报表发到他们的邮箱中。

此时你可能会产生这样的疑问：爬虫不是单单爬取数据的吗？为何还能处理这么多的事情呢？这还是爬虫的技术领域吗？答案是肯定的，上面这个例子是由我经历过的一个项目中的真实案例简化而来的，爬虫的这些行为融合了对爬取数据的后处理与 Python 自动化后得到的效果。其实爬虫能做到的事情可以更多，具体的实现与企业内部的实际需求相关。而在互联网中，它更像是一个具有“智能”的机器人。关于这些内容在本书后面的章节会有具体且详细的讲述。

企业内网爬虫只是互联网爬虫的一个小范围的应用，是爬虫技术与自动化技术的一种综合性应用，而且自动化技术的占比可能会比爬虫技术手段更多一些。

互联网爬虫

与企业爬虫相比，互联网爬虫就显得更为单一与常见，在这个数据唾手可得的时代，在数据中用爬虫淘金并不鲜见。如前面提及的搜索引擎本身就是“虫术大师”，只要是它们想爬的网站，几乎是没有爬不穿的。App Store 上最火的内容性 App 总是某些新闻类的聚合应用，大多数网站开发者都知道那只是一个聚合了各种新闻网站链接的综合性平台，它们的内容也是靠“放虫”才可能在各大新闻门户中获取第一手的新闻信息。更重要的是，这些新闻信息都是“免费”的，任何一个用户都可以轻易地从互联网上获取，这个用户当然也可以包括“虫子”。

互联网中存在大量如新闻资讯一类的免费内容，或是政府、企业、第三方机构、团体甚至个人共享的各种数据。例如，我们可以轻易地到气象局的网站上获取近十年某个地区的降雨量信息，或者从证券交易所获取当天各支股票的价格走势，又或者到微博上获知当天最具有传播性的某个事件的详情。换句话说，只要有清晰的目标数据源，只要你具有对数据源具有访问的权限，那么你也可以让爬虫为你代劳，一次性从数据源上获取所有你想要的数据。

要通过爬虫顺利地从互联网中爬取数据，那么就得了解这些数据的特质，然后采取针对性的手段才可能做到无往不利。一般来说，互联网中可爬取的数据可分为以下几种：

（1）一般性的网页——符合 W3C 规范的网页都可视为一种半结构化的内容，可以通过一些页面元素分析工具从网页中读取指定数据，由于网页开发的自由度极大，几乎没有哪个网站的结构是完全相同的。而且可变因素也很多，可能网页读取要通过权限的审查，或者网页由客户端的 JavaScript 进行绘制才能呈现最终效果，甚至网页可能来源于 CDN，其内容未必是最新的，只是某个网络缓存的副本，等等。不过不用担心，当你完全掌握了虫术，这一切对你将不再是阻挡。

（2）API 资源——API 资源是最适合爬取的数据源（没有之一），因为 RESTful API 都是结构化数据，会以 XML 或者 JSON 的形式进行调用或者返回，这些数据内容即使没有 API 说明手册一般也能读懂。

（3）文件资源——文件资源属于最麻烦的数据源了，除非爬取的文件是以结构化数据格式呈现的，否则作为自由文本，由于是非结构化的，我们需要对文本的内容进行一些后处理，要让爬虫“读懂”这些文本内容，再判断哪些内容是获取的目标。

（4）媒体资源——如图片和视频等，其爬取的动作基本与文件类似，只是由于图片与视频等资源一般来说都比较大，可能还需要对文件的元信息进行一些分析以判断其是否具有爬取的价值，以避免让爬虫过多地消耗不必要的网络流量与爬取时间。

1.1.3　Python与爬虫

本书所讲述的虫术都是基于 Python 的，读者可能会产生这样的疑问：“不可以用其他语言

来编写爬虫吗？”当然，所有的语言都可以编写爬虫，Python 只是其中的一种实现手段，但 Python 绝对是其中的佼佼者。

首先，Python 具有极为简单的语法与跨平台的支持，Windows、Linux、macOS 都完美支持 Python，甚至还可以将 Python 放到如 Arduino YUN、树莓派等智能设备的上位机上运行，这是除 C 语言外其他语言都无法企及的。

其次，Python 具有极其庞大的第三方资源库。在国外，Python 早已成为一门教学级和工具级的语言，除了软件开发的专业领域，也应用到了其他各类科学计算领域，而且通过社区，源自于各个科学领域的专用的资源包都可被共享。

再者，Python 具有脚本级的简单性。得益于编译器的能力，Python 的源代码可被编译为 C 语言的原生码运行，也就是说，Python 具有脚本的简洁易懂的特性，同时具有 C 语言一般的高效运行的效能。

最后，由于各科学领域的贡献与社区的积累，Python 拥有大量在大数据与科学计算方面既稳定又出色的工具库。例如，热门的爬虫框架、简单的 Web 开发框架、各种高等数学的计算函数、自然语言及语义分析工具等，而且大数据应用领域内的工具绝大多数都会首先支持 Python 的编程接口，这一切使得 Python 在大数据应用开发上具有综合性的优势。

综上所述，使用 Python 这门具有强悍综合能力、广泛工具支持、入门简单的语言，可以让我们在虫术这条路上走得更快、更远。

1.1.4 Python的网络爬虫框架

进入爬虫的世界后你会发现有两条路可以走，一条是“重新发明轮子”，一切都由你自己实现；另一条路则是使用别人的轮子，站人巨人的肩膀上看世界，站得高自然看得远。

为了学习底层实现，有必要从底层的源码入手，这样说并不代表我支持要走第一条路，我是个很懒的人，与其重新发明轮子不如在别人的基础上改良或者发展。当上层的功能堆积到一定程度时会自然组合成新的发明，因此我推荐使用已有的轮子，在别人的基础上发展，这样学得更快，而且更容易写出实用的东西。

主流框架

我们先从框架入手，看看现在 Python 世界中有哪些主流的网络爬虫框架。

- **PySpider**

PySpider（https://github.com/binux/pyspider）是国人做的一个爬虫架构的开源化实现，具有以下特性：

- Web 界面编写调试脚本，启停脚本，监控执行状态，查看活动历史，获取结果产出；

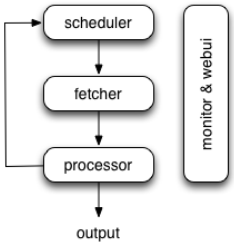
- 提供 SaaS 服务，可以在线提交部署；
- 支持 MySQL、MongoDB、SQLite；
- 原生支持抓取 JavaScript 的页面；
- 组件可替换，支持单机/分布式部署，支持 Docker 部署；
- 具有强大的调度控制；
- 灵活可扩展，稳定可监控。

这也是绝大多数 Python 爬虫的需求——定向抓取，结构化解析。但面对结构迥异的各种网站，单一的抓取模式并不一定能满足我们的要求，灵活的抓取控制是必需的。为了实现这个目的，单纯的配置文件往往不够灵活，于是，通过脚本去控制抓取是最后的选择。而去重调度、队列、抓取、异常处理、监控等功能作为框架，提供给抓取脚本，并保证灵活性。最后加上 Web 的编辑调试环境，以及 Web 任务监控，即组成了这套框架。

PySpider 的设计基础：以 Python 脚本驱动的抓取环模型爬虫。

- 通过 Python 脚本进行结构化信息的提取，follow 链接调度抓取控制，实现最大的灵活性。
- 通过 Web 化的脚本编写、调试环境。Web 展现调度状态，抓取环模型成熟稳定，模块间相互独立，通过消息队列连接，从单进程到多机分布式灵活拓展。

PySpider 的架构主要分为 scheduler（调度器）、fetcher（抓取器）和 processor（脚本执行），如下图所示。



- 各个组件间使用消息队列连接，除了 scheduler 是单点的，fetcher 和 processor 都是可以多实例分布式部署的。scheduler 负责整体的调度控制。
- 任务由 scheduler 发起调度，fetcher 抓取网页内容，processor 执行预先编写的 Python 脚本，输出结果或产生新的提链任务（发往 scheduler），形成闭环。
- 每个脚本可以灵活使用各种 Python 库对页面进行解析，使用框架 API 控制下一步抓取动作，通过设置回调控制解析动作。

PySpider 在最近几年的关注度颇高，主要得益其学习曲线比较平缓，代码的编写非常简单，而且容易理解，非常适合入门级和一些“短平快”的小型应用。

更高的易用性自然就会削弱其自定义的能力，在这方面 PySpider 的扩展能力是稍显不足的，而且对于千万级以上的增量式爬网，其去重（这两个概念在高阶虫术中会重点讲解）能力很弱，PySpider 更关注对数据入库的去重而缺失了对生成请求的 URL 的去重。而且其文档与资源相对其他老牌的爬虫框架要少，一旦出现问题，查找资料会显得困难重重。

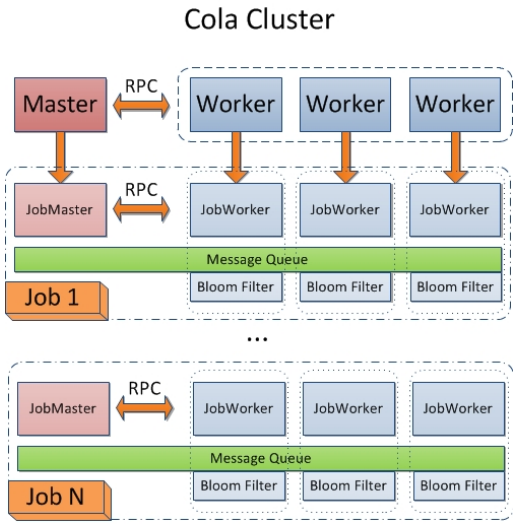
- **grap**

grab（http://docs.grablib.org/en/latest/#grab-spider-user-manual）是一个基于 pycurl/multicur 构建的网络爬虫框架。它是一个比较完善的爬虫框架，但相比于 PySpider，其学习曲线会显得陡峭一些，但工具与资源上的配置就与 PySpider 相差甚远。

- **Cola**

Cola（https://github.com/chineking/cola）是一个分布式的爬虫框架，用户只需编写几个特定的函数，而无须关注分布式运行的细节，任务会自动分配到多台机器上，整个过程对用户是透明的。

Cola 集群需要一个 Master 和若干个 Worker，每台机器只能启动一个 Worker。但是，集群不是必需的，在单机模式下也可以运行。Cola 集群集群如下图所示。



在 Cola 集群中，当一个任务被提交时，Cola Master 和 Worker 会分别启动 JobMaster 和 JobWorker。对于一个 Cola Job，当 JobWorker 启动完成后，会通知 JobMaster，JobMaster 等待所有 JobWorker 启动完成后开始运行 Job。在一个 Cola Job 启动时，会启动一个消息队列（Message Queue，主要操作是 put 和 get，Worker 抓取到的对象会被“put”到队列中，而要抓取新的对象时，只要从队列中获取即可），每个 JobWorker 上都存在消息队列节点，同时会有一个去重模块（bloom filter 实现）。

Cola 还不够稳定，目前处于持续改进的状态。而且 Cola 还没有在较大规模的集群上测试，但它也属于一个比较值得关注与学习的框架。

➢ **Scrapy**

一个基于 twisted 开发的可能是 Python 世界中最出名也是使用者最多的爬虫框架。同时它也是本书的主角，既然它是主角，那么在本书中的分量一定不少，这里容许我先卖个关子，后面自然对它有详尽的介绍。

其他

- Portia——基于 Scrapy 的可视化爬虫。地址为 https://github.com/scrapinghub/portia。
- Restkit——Python 的 HTTP 资源工具包。它可以让你轻松地访问 HTTP 资源，并围绕它建立的对象。地址为 https://github.com/benoitc/restkit。

1.1.5 虫术技术路线图

虫术的各个阶段涉及的技术非常庞杂，为了让读者有一个全面的认识，我特意将三个阶段中所要学习与使用的技术归纳成下图以作参考。