如果你看到了本课时，那么恭喜你已经学完了本专栏课程的所有内容，爬虫的知识点很复杂，一路学过来相信你也经历了不少坎坷。

本节课我们对网络爬虫所要学习的内容做一次总结，这里面也是我个人认为爬虫工程师应该具备的一些技术栈，由于专栏篇幅有限，肯定不可能把所有的知识点都覆盖到，但基础知识都已经涵盖了，下面我会把网络爬虫的知识点进行总结和梳理，如果你想深入学习网络爬虫的话可以参考。

网络爬虫的学习关系到计算机网络、编程基础、前端开发、后端开发、App 开发与逆向、网络安全、数据库、运维、机器学习、数据分析等各个方向的内容，它像一张大网一样把现在一些主流的技术栈都连接在了一起。正因为涵盖的方向多，因此学习的东西也非常零散和杂乱。

## 初学爬虫

一些最基本的网站，往往不带任何反爬措施。比如某个博客站点，我们要爬全站的话就顺着列表页爬到文章页，再把文章的时间、作者、正文等信息爬下来就可以了。

那代码怎么写呢？用 Python 的 requests 等库就够了，写一个基本的逻辑，顺带把一篇篇文章的源码获取下来，解析的话用 XPath、BeautifulSoup、PyQuery 或者正则表达式，或者粗暴的字符串匹配把想要的内容抠出来，再加个文本写入存下来就可以了。

代码也很简单，就只是几个方法的调用。逻辑也很简单，几个循环加存储。最后就能看到一篇篇文章被我们存到了自己的电脑里。当然如果你不太会写代码或者都懒得写，那么利用基本的可视化爬取工具，如某爪鱼、某裔采集器也能通过可视化点选的方式把数据爬下来。

如果存储方面稍微扩展一下的话，可以对接上 MySQL、MongoDB、Elasticsearch、Kafka 等来保存数据，实现持久化存储。以后查询或者操作会更方便。

反正，不管效率如何，一个完全没有反爬的网站用最基本的方式就可以搞定。到这里，你可以说自己会爬虫了吗？不，还差得远呢。

## Ajax、动态渲染

随着互联网的发展，前端技术也在不断变化，数据的加载方式也不再是单纯的服务端渲染了。现在你可以看到很多网站的数据可能都是通过接口的形式传输的，或者即使不是接口那也是一些 JSON 数据，然后经过 JavaScript 渲染得出来的。

这时候，你要再用 requests 来爬取就不行了，因为 requests 爬下来的源码是服务端渲染得到的，浏览器看到页面的和 requests 获取的结果是不一样的。真正的数据是经过 JavaScript 执行得出来的，数据来源可能是 Ajax，也可能是页面里的某些 Data，也可能是一些 iframe 页面等，不过大多数情况下可能是 Ajax 接口获取的。

所以很多情况下需要分析 Ajax，知道这些接口的调用方式之后再用程序来模拟。但是有些接口带着加密参数，比如 token、sign 等，又不好模拟，怎么办呢？

一种方法就是去分析网站的 JavaScript 逻辑，死抠里面的代码，研究这些参数是怎么构造的，找出思路之后再用爬虫模拟或重写就行了。如果你解析出来了，那么直接模拟的方式效率会高很多，这里面就需要一些 JavaScript 基础了，当然有些网站加密逻辑做得太厉害了，你可能花一个星期也解析不出来，最后只能放弃了。

那这样解不出来或者不想解了，该怎么办呢？这时候可以用一种简单粗暴的方法，也就是直接用模拟浏览器的方式来爬取，比如用 Puppeteer、Pyppeteer、Selenium、Splash 等，这样爬取到的源代码就是真正的网页代码，数据自然就好提取了，同时也就绕过分析 Ajax 和一些 JavaScript 逻辑的过程。这种方式就做到了可见即可爬，难度也不大，同时模拟了浏览器，也不太会有一些法律方面的问题。

但其实后面的这种方法也会遇到各种反爬的情况，现在很多网站都会去识别 webdriver，看到你是用的 Selenium 等工具，直接拒接或不返回数据，所以你碰到这种网站还得专门来解决这个问题。

## 多进程、多线程、协程

上面的情况如果用单线程的爬虫来模拟是比较简单的，但是有个问题就是速度慢啊。

爬虫是 I/O 密集型的任务，所以可能大多数情况下都在等待网络的响应，如果网络响应速度慢，那就得一直等着。但这个空余的时间其实可以让 CPU 去做更多事情。那怎么办呢？多开一些线程吧。

所以这个时候我们就可以在某些场景下加上多进程、多线程，虽然说多线程有 GIL 锁，但对于爬虫来说其实影响没那么大，所以用上多进程、多线程都可以成倍地提高爬取速度，对应的库就有 threading、multiprocessing 等。

异步协程就更厉害了，用 aiohttp、gevent、tornado 等工具，基本上想开多少并发就开多少并发，但是还是得谨慎一些，别把目标网

站搞挂了。

总之，用上这几个工具，爬取速度就提上来了。但速度提上来了不一定都是好事，反爬措施接着肯定就要来了，封你 IP、封你账号、弹验证码、返回假数据，所以有时候龟速爬似乎也是个解决办法？

## 分布式

多线程、多进程、协程都能加速，但终究还是单机的爬虫。要真正做到规模化，还得靠分布式爬虫来搞定。

分布式的核心是什么？资源共享。比如爬取队列共享、去重指纹共享，等等。

我们可以使用一些基础的队列或组件来实现分布式，比如 RabbitMQ、Celery、Kafka、Redis 等，但经过很多人的尝试，自己去实现一个分布式爬虫，性能和扩展性总会出现一些问题。不少企业内部其实也有自己开发的一套分布式爬虫，和业务更紧密，这种当然是最好了。

现在主流的 Python 分布式爬虫还是基于 Scrapy 的，对接 Scrapy-Redis、Scrapy-Redis-BloomFilter 或者使用 Scrapy-Cluster 等，它们都是基于 Redis 来共享爬取队列的，多多少少总会遇到一些内存的问题。所以一些人也考虑对接到其他的消息队列上面，比如 RabbitMQ、Kafka 等，可以解决一些问题，效率也不差。

总之，要提高爬取效率，分布式还是必须要掌握的。

## 验证码

爬虫时难免遇到反爬，验证码就是其中之一。要会反爬，那首先就要会解验证码。

现在你可以看到很多网站都会有各种各样的验证码，比如最简单的图形验证码，要是验证码的文字规则的话，OCR 检测或基本的模型库都能识别，你可以直接对接一个打码平台来解决，准确率还是可以的。

然而现在你可能都见不到什么图形验证码了，都是一些行为验证码，如某验、某盾等，国外也有很多，比如 reCaptcha 等。一些稍微简单一点的，比如滑动的，你可以想办法识别缺口，比如图像处理比对、深度学习识别都是可以的。

对于轨迹行为你可以自己写一个模拟正常人行为的，加入抖动等。有了轨迹之后如何模拟呢，如果你非常厉害，那么可以直接去分析验证码的 JavaScript 逻辑，把轨迹数据录入，就能得到里面的一些加密参数，直接将这些参数放到表单或接口里面就能直接用了。当然也可以用模拟浏览器的方式来拖动，也能通过一定的方式拿到加密参数，或者直接用模拟浏览器的方式把登录一起做了，拿着 Cookies 来爬也行。

当然拖动只是一种验证码，还有文字点选、逻辑推理等，要是真不想自己解决，可以找打码平台来解析出来再模拟，但毕竟是花钱的，一些高手就会选择自己训练深度学习相关的模型，收集数据、标注、训练，针对不同的业务训练不同的模型。这样有了核心技术，也不用再去花钱找打码平台了，再研究下验证码的逻辑模拟一下，加密参数就能解析出来了。不过有的验证码解析非常难，以至于我也搞不定。

当然有些验证码可能是请求过于频繁而弹出来的，这种如果换 IP 也能解决。

## 封 IP

封 IP 也是一件令人头疼的事，行之有效的方法就是换代理了。代理有很多种，市面上免费的，收费的太多太多了。

首先可以把市面上免费的代理用起来，自己搭建一个代理池，收集现在全网所有的免费代理，然后加一个测试器一直不断测试，测试的网址可以改成你要爬的网址。这样测试通过的一般都能直接拿来爬取目标网站。我自己也搭建过一个代理池，现在对接了一些免费代理，定时爬、定时测，还写了个 API 来取，放在了 GitHub 上：https://github.com/Python3WebSpider/ProxyPool，打好了 Docker 镜像，提供了 Kubernetes 脚本，你可以直接拿来用。

付费代理也是一样，很多商家提供了代理提取接口，请求一下就能获取几十几百个代理，我们可以同样把它们接入到代理池里面。但这个代理服务也分各种套餐，什么开放代理、独享代理等的质量和被封的概率也是不一样的。

有的商家还利用隧道技术搭建了代理，这样代理的地址和端口我们是不知道的，代理池是由他们来维护的，比如某布云，这样用起来更省心一些，但是可控性就差一些。

还有更稳定的代理，比如拨号代理、蜂窝代理等，接入成本会高一些，但是一定程度上也能解决一些封 IP 的问题。

## 封账号

有些信息需要模拟登录才能爬取，如果爬得过快，目标网站直接把你的账号封禁了，就没办法了。比如爬公众号的，人家把你 WX 号封了，那就全完了。

一种解决方法就是放慢频率，控制节奏。还有一种方法就是看看别的终端，比如手机页、App 页、wap 页，看看有没有能绕过登录的方法。

另外比较好的方法，就是分流。如果你的号足够多，建一个池子，比如 Cookies 池、Token 池、Sign 池等，多个账号跑出来的 Cookies、Token 都放到这个池子里，用的时候随机从里面获取一个。如果你想保证爬取效率不变，那么 100 个账号相比 20 个账号，对于每个账号对应的 Cookies、Token 的取用频率就变成原来的了 1/5，那么被封的概率也就随之降低了。

## 奇葩的反爬

上面说的是几种比较主流的反爬方式，当然还有非常多奇葩的反爬。比如返回假数据、返回图片化数据、返回乱序数据，等等，那都需要具体情况具体分析。

这些反爬也得小心点，之前见过一个反爬直接返回 `rm -rf /` 的也不是没有，你要是正好有个脚本模拟执行返回结果，后果自己想象。

## JavaScript 逆向

说到重点了。随着前端技术的进步和网站反爬意识的增强，很多网站选择在前端上下功夫，那就是在前端对一些逻辑或代码进行加密或混淆。当然这不仅仅是为了保护前端的代码不被轻易盗取，更重要的是反爬。比如很多 Ajax 接口都会带着一些参数，比如 sign、token 等，这些前文也讲过了。这种数据我们可以用前文所说的 Selenium 等方式来爬取，但总归来说效率太低了，毕竟它模拟的是网页渲染的整个过程，而真实的数据可能仅仅就藏在一个小接口里。

如果我们能够找出一些接口参数的真正逻辑，用代码来模拟执行，那效率就会有成倍的提升，而且还能在一定程度上规避上述的反爬现象。但问题是什么？比较难实现啊。

Webpack 是一方面，前端代码都被压缩和转码成一些 bundle 文件，一些变量的含义已经丢失，不好还原。然后一些网站再加上一些 obfuscator 的机制，把前端代码变成你完全看不懂的东西，比如字符串拆散打乱、变量十六进制化、控制流扁平化、无限 debug、控制台禁用等，前端的代码和逻辑已经面目全非。有的用 WebAssembly 等技术把前端核心逻辑直接编译，那就只能慢慢抠了，虽然说有些有一定的技巧，但是总归来说还是会花费很多时间。但一旦解析出来了，那就万事大吉了。

很多公司招聘爬虫工程师都会问有没有 JavaScript 逆向基础，破解过哪些网站，比如某宝、某多、某条等，解出来某个他们需要的可能就直接录用你。每家网站的逻辑都不一样，难度也不一样。

## App

当然爬虫不仅仅是网页爬虫了，随着互联网时代的发展，现在越来越多的公司都选择将数据放到 App 上，甚至有些公司只有 App 没有网站。所以数据只能通过 App 来爬。

怎么爬呢？基本的就是抓包工具了，Charles、Fiddler 等抓到接口之后，直接拿来模拟就行了。

如果接口有加密参数怎么办呢？一种方法你可以边爬边处理，比如 mitmproxy 直接监听接口数据。另一方面你可以走 Hook，比如上 Xposed 也可以拿到。

那爬的时候又怎么实现自动化呢？总不能拿手来戳吧。其实工具也多，安卓原生的 adb 工具也行，Appium 现在已经是比较主流的方案了，当然还有其他的某精灵都是可以实现的。

最后，有的时候可能真的就不想走自动化的流程，我就想把里面的一些接口逻辑抠出来，那就需要搞逆向了，IDA Pro、jdax、FRIDA 等工具就派上用场了，当然这个过程和 JavaScript 逆向一样很痛苦，甚至可能得读汇编指令。

## 智能化

上面的这些知识，都掌握了以后，恭喜你已经超过了百分之八九十的爬虫玩家了，当然专门搞 JavaScript 逆向、App 逆向的都是站在食物链顶端的人，这种严格来说已经不算爬虫范畴了。

除了上面的技能，在一些场合下，我们可能还需要结合一些机器学习的技术，让我们的爬虫变得更智能起来。

比如现在很多博客、新闻文章，其页面结构相似度比较高，要提取的信息也比较类似。

比如如何区分一个页面是索引页还是详情页？如何提取详情页的文章链接？如何解析文章页的页面内容？这些其实都是可以通过一些算法来计算出来的。

所以，一些智能解析技术也应运而生，比如提取详情页，我的一位朋友写的 GeneralNewsExtractor 表现就非常好。

假如说有一个需求，需要爬取一万个新闻网站数据，要一个个写 XPath 吗？如果有了智能化解析技术，在容忍一定错误的条件

下，完成这个就是分分钟的事情。

总之，如果我们能把这一块也学会了，我们的爬虫技术就会如虎添翼。

## 运维

这块内容也是一个重头戏。爬虫和运维也是息息相关的。比如：

- 写完一个爬虫，怎样去快速部署到 100 台主机上运行起来。
- 怎么灵活地监控每个爬虫的运行状态。
- 爬虫有处代码改动，如何去快速更新。
- 怎样监控一些爬虫的占用内存、消耗的 CPU 状况。
- 怎样科学地控制爬虫的定时运行。
- 爬虫出现了问题，怎样能及时收到通知，怎样设置科学的报警机制。

这里面，部署大家各有各的方法，比如可以用 Ansible。如果用 Scrapy 的话有 Scrapyd，然后配合上一些管理工具也能完成一些监控和定时任务。不过我现在用的更多的还是 Docker + Kubernetes，再加上 DevOps 一套解决方案，比如 GitHub Actions、Azure Pipelines、Jenkins 等，快速实现分发和部署。

定时任务大家有的用 crontab，有的用 apscheduler，有的用管理工具，有的用 Kubernetes，我的话用 Kubernetes 会多一些了，定时任务也很好实现。

至于监控的话，也有很多，专门的爬虫管理工具自带了一些监控和报警功能。一些云服务也带了一些监控的功能。我用的是 Kubernetes + Prometheus + Grafana，什么 CPU、内存、运行状态，一目了然，报警机制在 Grafana 里面配置也很方便，支持 Webhook、邮件甚至某钉。

数据的存储和监控，用 Kafka、Elasticsearch 个人感觉也挺方便的，我主要用的是后者，然后再和 Grafana 配合起来，数据爬取量、爬取速度等等监控也都一目了然。

## 法律

另外希望你在做网络爬虫的过程中注意一些法律问题，基本上就是：

- 不要碰个人隐私信息。
- 规避商业竞争，看清目标站点的法律条款限制。
- 限制并发速度，不要影响目标站点的正常运行。
- 不要碰黑产、黄赌毒。
- 不随便宣传和传播目标站点或 App 的破解方案。
- 非公开数据一定要谨慎。

更多的内容可以参考一些文章：

- https://mp.weixin.qq.com/s/aXr-ZE0ZifTm2h5w8BGh_Q
- https://mp.weixin.qq.com/s/zVTMQz78L16i7j8wXGjbLA
- https://mp.weixin.qq.com/s/MrJbodU0tcW6FRZ3JQa3xQ

## 结语

至此，爬虫的一些涵盖的知识点也就差不多了，通过梳理发现计算机网络、编程基础、前端开发、后端开发、App 开发与逆向、网络安全、数据库、运维、机器学习都涵盖到了？上面总结的可以算是从爬虫小白到爬虫高手的路径了，里面每个方向其实可研究的点非常多，每个点做精了，都会非常了不起。

最后，感谢你学习我的课程，希望你在学习过程中有所收获。