ГЛАВА 1

Как работает Интернет

Мне встречалось не так много людей, которые по-настоящему знают, как работает Интернет, и я определенно не принадлежу к их числу.

Многим из нас достаточно простого набора ментальных абстракций, позволяющих пользоваться Интернетом ровно настолько, насколько это необходимо. Даже программисты могут пользоваться весьма ограниченными абстракциями, которых достаточно для решения их задач.

Из-за ограниченного объема книги и знаний автора эта глава также основывается на подобных абстракциях. В ней описывается механизм работы Интернета и веб-приложений в том объеме, который необходим для веб-скрапинга (и, возможно, немного больше).

Эта глава в некотором смысле откроет перед вами мир, где работают веб-скраперы: его обычаи, практики, протоколы и стандарты, к которым мы будем возвращаться на протяжении всей книги.

Если ввести URL в адресную строку браузера и нажать Enter, то как по волшебству появятся текст, изображения и мультимедийные объекты. И это волшебство наблюдают миллиарды разных людей каждый день. Они посещают те же сайты, используют те же приложения и часто получают медиаконтент и текст, созданные специально для них.

Все эти миллиарды людей используют различные типы устройств и приложений, написанных разными разработчиками из разных (часто конкурирующих!) компаний.

Удивительно, но не существует единого руководящего органа, который мог бы регулировать Интернет и координировать его развитие, обладая при этом какой-либо юридической силой. Вместо этого разные части Интернета управляются несколькими различными организациями, которые со временем развивались по принципах «от случая к случаю» и «по мере необходимости».

Конечно, если вы *откажетесь* соблюдать стандарты, принятые этими организациями, ваши веб-разработки могут просто... перестать работать. Например, ваш сайт не отображается в популярных браузерах, значит, люди не будут его посещать. Если данные, отправляемые вашим маршрутизатором, будут непонятны другим маршрутизаторам, то эти данные будут проигнорированы.

Веб-скрапинг — по сути, замена веб-браузера приложением собственной разработки. Поэтому так важно понимать стандарты и основы, на которых построены браузеры. Ваши будущие веб-скраперы должны поддерживать, а иногда даже нарушать обычаи и практики Интернета.

Передача данных в сети

На заре развития телефонии каждый телефонный аппарат был физически подключен проводом к центральному коммутатору. Решив позвонить другу, вы снимали трубку, просили оператора соединить вас, и тот физически (с помощью штекеров и разъемов) создавал прямое соединение между вашим телефоном и телефоном вашего друга.

Междугородние звонки стоили дорого, а установление соединения могло занять несколько минут. Чтобы позвонить из Бостона в Сиэтл, требовалось скоординировать действия операторов коммутаторов по всей территории США для создания единой огромной проводной цепи, напрямую соединяющей ваш телефон с телефоном другого абонента.

Сегодня нам не нужно временное выделенное соединение — мы можем совершать видеозвонки из дома в любую точку мира через устойчивую паутину проводов. Теперь не провод передает данные, они сами направляют себя, используя процесс, называемый *коммутацией пакетов*. Многие технологии внесли свой вклад в создание того, что мы называем Интернетом, но коммутация пакетов — это технология, положившая начало всему.

В сети с коммутацией пакетов отправляемое сообщение делится на упорядоченную последовательность пакетов, каждый из которых имеет адрес отправителя и получателя. Эти пакеты динамически пересылаются в тот или иной пункт назначения в сети на основе их адреса. Вместо того чтобы следовать по единственному выделенному соединению от получателя к отправителю вслепую, пакеты могут идти по любому пути, который выберет сеть. Фактически пакеты, составляющие одно сообщение, могут идти по разным маршрутам и прибывать в компьютер адресата не по порядку.

Если старые телефонные сети были подобны канатной дороге, доставляющей пассажиров из одного пункта на вершине холма в другой пункт внизу, то сети

с коммутацией пакетов подобны системе автомагистралей, по которым разные автомобили могут ехать в разные пункты назначения и обратно, пользуясь одними и теми же дорогами.

Современную сеть с коммутацией пакетов обычно описывают, используя модель взаимодействия открытых систем (Open Systems Interconnection, OSI), которая состоит из семи уровней маршрутизации, кодирования и обработки ошибок.

1. Физический уровень.
2. Канальный уровень.
3. Сетевой уровень.
4. Транспортный уровень.
5. Сеансовый уровень.
6. Уровень представления.
7. Прикладной уровень.

Большинство разработчиков веб-приложений почти не покидают прикладной уровень. Именно ему уделяется наибольшее внимание в данной книге. Однако, занимаясь веб-скрапингом, важно иметь хотя бы общее представление и о других уровнях. Например, сканирование отпечатков пальцев по протоколу TLS, которое будет рассмотрено в главе 17, представляет собой метод обнаружения веб-скрапинга, работающий на транспортном уровне.

Кроме того, знание особенностей организации и передачи данных на всех уровнях может помочь устранить ошибки в ваших веб-приложениях и веб-скраперах.

Физический уровень

На *физическом уровне* происходит передача информации по проводам Ethernet с помощью электрических сигналов. К важным особенностям относятся уровни напряжения, представленные кодами 1 и 0, частота их изменения, а также порядок интерпретации радиоволн, передающихся через устройства Bluetooth и Wi-Fi.

Этот уровень не требует программирования или цифровых инструкций и основан исключительно на физических и электрических стандартах.

Канальный уровень

На *канальном уровне* задается порядок передачи информации между двумя узлами локальной сети, например между компьютером и маршрутизатором. Определяются начало и конец передачи единого блока данных, и обеспечи-

вается исправление ошибок, если в процессе передачи данные были потеряны или искажены.

На этом уровне пакеты упаковываются в дополнительные «цифровые конверты», которые содержат информацию о маршрутизации и называются *фреймами*. Когда информация, содержащаяся во фрейме, становится не нужна, она разворачивается и отправляется по сети в виде пакета.

Важно отметить, что на канальном уровне все устройства в сети получают одни и те же данные — никакой коммутации или управления передачей пакетов адресатам на этом уровне не происходит. Однако устройства, не являющиеся получателями данных, обычно игнорируют их и ждут, пока не придет то, что предназначено для них.

Сетевой уровень

На *сетевом уровне* выполняется коммутация пакетов, и именно тут задействован Интернет. Происходит пересылка пакетов с вашего компьютера на маршрутизатор и далее на другие устройства, находящиеся за пределами локальной сети.

Сетевой уровень реализуется через интернет-протокол (Internet Protocol, IP) — часть широко известной связки протокола управления передачей и протокола Интернета (Transmission Control Protocol/Internet Protocol, TCP/IP). IP — это источник IP-адресов. Например, прямо сейчас мой IP-адрес в глобальной сети Интернет: 173.48.178.92. Это позволяет мне обмениваться данными с любыми другими имеющими IP-адрес компьютерами в мире.

Транспортный уровень

На *транспортном уровне* осуществляется подключение определенных сервисов или приложений, работающих на одном компьютере, к конкретным сервисам и приложениям, работающим на другом компьютере. Здесь также происходит исправление ошибок и выполняются повторные попытки передачи данных, если это необходимо.

TCP очень взыскателен и будет запрашивать недостающие пакеты в потоке до тех пор, пока все они не будут получены. TCP часто используется для передачи файлов, когда важно, чтобы все пакеты были доставлены корректно и из них получился правильно сформированный файл.

Протокол пользовательских дейтаграмм (User Datagram Protocol, UDP), наоборот, проигнорирует потерю пакетов, чтобы сохранить поток данных. Он часто

используется для видео- или аудиоконференций, где временное снижение качества передачи данных предпочтительнее, чем задержка в разговоре.

Поскольку у разных приложений на вашем компьютере могут быть разные требования к надежности данных (например, телефонный звонок во время загрузки файла), на транспортном уровне также используется номер порта. Операционная система назначает каждому приложению или сервису, работающему на компьютере, определенный номер порта, куда они отправляют и откуда получают данные.

Номер порта обычно записывается как число после IP-адреса, отделенное от последнего двоеточием. Например, запись 71.245.238.173:8080 говорит о том, что на компьютере с IP-адресом 71.245.238.173 работает приложение, которому операционная система назначила порт 8080.

Сеансовый уровень

На *сеансовом уровне* происходят открытие и закрытие сеанса между двумя приложениями. Поддержка сеансов позволяет получать информацию о том, какие данные были отправлены, а какие нет и с кем взаимодействует компьютер. Сеанс обычно остается открытым до тех пор, пока выполняются запросы, а затем закрывается.

Сеансовый уровень позволяет повторить передачу данных в случае кратковременного сбоя или отключения.

Сеансы на разных уровнях

Сеансы на сеансовом уровне модели OSI — это не то же, что сеансы веб-приложений, о которых обычно говорят веб-разработчики. Сеанс в веб-приложении — это понятие прикладного уровня, реализуемое программным обеспечением браузера.

Информация о сеансе прикладного уровня хранится в браузере, пока она востребована или пока пользователь не закроет окно браузера. На сеансовом уровне модели OSI сеанс обычно длится ровно столько, сколько необходимо для передачи одного файла!

Уровень представления

На *уровне представления* входящие данные преобразуются из строк символов в формат, понятный приложению. Здесь также происходят кодирование символов и сжатие данных. Уровень представления различает форматы входных данных, получаемых приложением (например, PNG или HTML), выполняет необходимые преобразования и передает преобразованные данные прикладному уровню.

Прикладной уровень

На *прикладном уровне* интерпретируются данные, закодированные на предыдущем уровне, и передаются приложению. Иными словами, уровень представления связан с преобразованием и идентификацией данных, а прикладной — с выполнением задач. Например, HTTP с его методами и статусами является протоколом прикладного уровня. Более банальные JSON и HTML (поскольку фактически являются типами файлов и определяют способ кодирования данных) — это протоколы уровня представления.

HTML

Основная функция веб-браузера — отображение документов HTML (HyperText Markup Language — язык разметки гипертекста). HTML-документы — это файлы с расширением `.html` или реже `.htm`.

Как и простые текстовые файлы, HTML-файлы содержат самые обычные текстовые символы, как правило, ASCII (см. раздел «Кодирование текста и глобальный Интернет» главы 10). Это означает, что их можно открыть и прочитать в любом текстовом редакторе.

Вот пример простого HTML-файла:

```
<html>
  <head>
    <title>A Simple Webpage</title>
  </head>
  <body>
    <!-- Это текст комментария, он не выводится браузером -->
    <h1>Hello, World!</h1>
  </body>
</html>
```

Файлы HTML — это особый тип файлов XML (Extensible Markup Language — расширяемый язык разметки). Строки, начинающиеся с `<` и заканчивающиеся `>`, называются *тегами*.

Стандарт XML определяет понятия открывающих, или *начальных*, тегов, таких как `<html>`, и закрывающих, или *конечных*, которые начинаются с пары символов `</`, например `</html>`. Между начальным и конечным тегами находится их *содержимое*.

Если тег не имеет содержимого, то он может быть записан в так называемой самозакрывающейся форме:

```
<p />
```

Тег может иметь атрибуты, которые записываются в форме `имяАтрибута="значение атрибута"`, например:

```
<div class="content">
  Lorem ipsum dolor sit amet, consectetur adipiscing elit
</div>
```

Здесь тег `div` имеет атрибут `class` со значением `main-content`.

HTML-*элемент* имеет начальный (открывающий) тег, иногда с дополнительными атрибутами, некоторое содержимое и конечный (закрывающий) тег. Элемент также может содержать другие элементы, которые в этом случае являются *вложенными*.

Стандарт XML определяет базовые понятия (тег, содержимое, атрибуты и их значения), а HTML — перечень допустимых тегов, их назначение, что они могут содержать и как должны интерпретироваться и отображаться браузером.

Например, стандарт HTML определяет использование атрибутов `class` и `id`, которые часто применяются для организации и управления отображением элементов HTML:

```
<h1 id="main-title">Some Title</h1>
<div class="content">
  Lorem ipsum dolor sit amet, consectetur adipiscing elit
</div>
```

Один и тот же класс может быть присвоен сразу нескольким элементам на странице, однако значения в атрибуте `id` должны быть уникальными для этой страницы. Таким образом, класс `content` могут иметь несколько элементов, но идентификатор `main-title` — только один.

Порядок отображения элементов HTML-документа на веб-странице полностью зависит от особенностей браузера. Если разные веб-браузеры будут отображать одни и те же элементы интерфейса по-разному, это приведет к несогласованности действий пользователей.

По этой причине HTML-теги должны быть согласованы и зафиксированы в едином стандарте. В настоящее время стандарт HTML контролируется Консорциумом Всемирной паутины (World Wide Web Consortium, W3C). Текущую спецификацию всех HTML-тегов можно найти по адресу https://html.spec.whatwg.org/multipage/.

Однако официальный стандарт W3C HTML — не лучший вариант для начинающих изучать HTML. Веб-скрапинг почти полностью заключается в чтении и интерпретации HTML-файлов, найденных в Интернете. Если вы никогда раньше не имели дела с HTML, я настоятельно рекомендую прочитать книгу *HTML & CSS: The Good Parts*[1], чтобы познакомиться с некоторыми наиболее распространенными тегами HTML.

---

[1] *Хеник Б.* HTML и CSS. Путь к совершенству. — СПб.: Питер, 2011.

CSS

Каскадные таблицы стилей (Cascading Style Sheets, CSS) определяют внешний вид HTML-элементов на веб-странице. Они задают, например, размещение, цвет, размер и другие свойства элементов, которые превращают скучную HTML-страницу со стилями, заданными браузером по умолчанию, в нечто более привлекательное для современного пользователя.

Для приведенного выше примера HTML:

```
<html>
  <head>
    <title>A Simple Webpage</title>
  </head>
  <body>
    <!-- Это текст комментария, он не выводится браузером -->
    <h1>Hello, World!</h1>
  </body>
</html>
```

вариант его оформления:

```
h1 {
  font-size: 20px;
  color: green;
}
```

Эта таблица стилей задает размер шрифта содержимого тега `h1` равным 20 пикселам и отображает его в зеленом цвете.

Часть `h1` в этой таблице стилей называется *селектором* или CSS-селектором. Он указывает, что стили, определяемые внутри фигурных скобок, будут применяться к содержимому любых тегов `h1`.

Селекторы CSS могут также применяться только к элементам с определенными атрибутами: `class` или `id`. Например, для разметки HTML:

```
<h1 id="main-title">Some Title</h1>
<div class="content">
  Lorem ipsum dolor sit amet, consectetur adipiscing elit
</div>
```

таблица стилей может быть следующей:

```
h1#main-title {
  font-size: 20px;
}

div.content {
  color: green;
}
```

Символ # уточняет, что определяемые стили должны применяться к элементу с атрибутом `id="main-title"`, а символ . указывает на значение атрибута `class`.

Если имя тега не имеет значения, то его можно полностью опустить. Например, следующая таблица стилей окрасит в зеленый цвет содержимое любого элемента с атрибутом `class="content"`:

```
.content {
  color: green;
}
```

Данные таблицы стилей могут содержаться либо в самом HTML, либо в отдельном CSS-файле с расширением `.css`. Таблица стилей в HTML-файле размещается внутри тегов `<style>` в заголовке HTML-документа:

```
<html>
  <head>
    <style>
      .content {
        color: green;
      }
    </style>
...
```

Чаще всего CSS импортируется в заголовок документа с помощью тега `link`:

```
<html>
  <head>
    <link rel="stylesheet" href="mystyle.css">
...
```

Занимаясь веб-скрапингом, вам нечасто придется писать таблицы стилей, чтобы сделать HTML красивым. Однако важно владеть синтаксисом и понимать, каким образом HTML-страница преобразуется с помощью CSS, чтобы видеть взаимосвязь между тем, что отображается в браузере, и тем, что написано в коде.

Например, вас может удивить элемент HTML, который не отображается на странице. Однако, прочитав код CSS, применяемый к элементу, вы можете увидеть определение стиля:

```
.mystery-element {
  display: none;
}
```

Это означает, что для атрибута элемента `display` установлено значение `none`, делая его невидимым.

Если вы никогда раньше не сталкивались с таблицами стилей, скорее всего, вам не потребуется глубоко их изучать, чтобы извлекать данные из Интернета, однако необходимо освоить их синтаксис и учитывать правила CSS, приведенные в этой книге.

JavaScript

Когда веб-сервер получает от клиента запрос на определенную веб-страницу, он выполняет некоторый код, чтобы создать ее и затем отправить пользователю. Этот *серверный код* может выполнять очень простые действия, такие как чтение статического HTML-файла с диска и его отправка, или очень сложные, написанные на Python (лучшем языке), Java, PHP или любом другом серверном языке программирования.

В итоге серверный код создает своеобразный поток данных, который отправляется браузеру и отображается на экране. Но что, если нужно выполнить некоторое действие, например изменить текст или перетащить элемент, без обращения к серверу и выполнения кода на его стороне? Для этого используется *клиентский код*.

Клиентский код — это любой код, который передается клиенту веб-сервером, но выполняется браузером. На заре Интернета (до середины 2000-х) клиентский код писался на нескольких языках. Возможно, вы помните Java-апплеты и Flash-приложения. Но со временем JavaScript стал единственным языком для реализации клиентского кода по одной простой причине: этот язык изначально поддерживался браузерами без дополнительного программного обеспечения, загружаемого и обновляемого отдельно (как, например, Adobe Flash Player).

JavaScript появился в середине 90-х как новая функция Netscape Navigator. Он был быстро адаптирован к Internet Explorer, что сделало его стандартом для обоих основных веб-браузеров того времени.

Несмотря на название, JavaScript не имеет почти ничего общего с Java — серверным языком программирования. Если не считать некоторого сходства в синтаксисе, это совершенно разные языки.

В 1996 году компании Netscape (где был создан JavaScript) и Sun Microsystems (создательница Java) заключили лицензионное соглашение, позволяющее Netscape использовать название JavaScript, что предполагало дальнейшее тесное сотрудничество в развитии двух языков (https://www.infoworld.com/article/2653798/javascript-creator-ponders-past--future.html). Однако это сотрудничество так и не состоялось, и с тех пор название языка JavaScript сбивает с толку.

Несмотря на то что изначально JavaScript был языком сценариев для уже несуществующего браузера, в настоящее время он превратился в один из самых популярных языков программирования в мире. Этой популярности поспособствовал тот факт, что на нем можно также писать серверный код, используя Node.js. Но основная причина его всеобщего признания, безусловно, заключается в том, что это единственный язык программирования, используемый на стороне клиента.

JavaScript встраивается в HTML-страницы с помощью тега `<script>`. Код на JavaScript можно вставить в форме содержимого:

```
<script>
  alert('Hello, world!');
</script>
```

Или же на него можно сослаться в отдельном файле, используя атрибут `src`:

```
<script src="someprogram.js"></script>
```

В отличие от HTML и CSS, вам, скорее всего, не нужно будет читать или писать код на JavaScript, занимаясь веб-скрапингом, но будет полезно хотя бы знать, как он выглядит. Иногда код может содержать полезные данные. Например:

```
<script>
  const data = '{"some": 1, "data": 2, "here": 3}';
</script>
```

Здесь объявляется JavaScript-переменная `data` с ключевым словом `const` (что означает «константа») и ей присваивается строка в формате JSON, содержащая данные, которые теоретически можно парсировать напрямую с помощью веб-скрапера.

JSON (JavaScript Object Notation — форма записи объектов JavaScript) — это текстовый формат, описывающий данные, который легко парсируется веб-скраперами и широко применяется в сети. Он будет рассмотрен подробнее в главе 15.

Иногда вы можете увидеть, что код на JavaScript запрашивает данные из стороннего источника:

```
<script>
  fetch('http://example.com/data.json')
    .then((response) => {
      console.log(response.json());
    });
</script>
```

Здесь код на JavaScript посылает запрос по адресу `http://example.com/data.json` и выводит полученный ответ в консоль (подробнее о консоли рассказывается в следующем разделе).

Изначально JavaScript создавался для реализации динамической интерактивности и анимации в статических веб-страницах. Однако сейчас динамическое поведение создается не только с помощью JavaScript. HTML и CSS тоже обладают некоторыми возможностями, позволяющими изменять содержимое страницы.

Например, поддержка анимации в CSS позволяет перемещать элементы, менять их цвет, размер и выполнять другие преобразования, когда пользователь выбирает элемент или наводит на него указатель мыши.

Понимание того, как связаны (часто в буквальном смысле) движущиеся части веб-сайта, может помочь вам избежать путаницы при поиске данных.