Содержание

Об авторах

Эндрю Ринингсланд – разработчик и журналист, в последние десять лет он большую часть времени занимался созданием интерактивного контента для таких изданий, как Financial Times, Times, Sunday Times, Economist и Guardian. За три года работы в газетах Times и Sunday Times он участвовал в различных проектах – от написания некрологов о смерти таких личностей, как Нельсон Мандела, до резонансных расследований типа допингового скандала – крупнейшей в истории утечки данных об анализах крови спортсменов. В настоящее время является старшим разработчиком группы интерактивной графики в редакции Financial Times.

Выражаю благодарность своей восхитительной подруге Наоми, которая бог знает сколько раз подбадривала и нахваливала меня, пока я работал над книгой. Спасибо также всем членам группы D3.js Slack, оказывавшим мне всемерную поддержку, а особенно завсегдатаям канала #v4-migration, которые помогли освоиться с бесчисленными изменениями в версии 4.

Свизек Теллер, автор книги «Data Visualization with d3.js» – технарь по призванию. В 21 год он основал свой первый стартап, а теперь, в поисках очередной достойной идеи, трудится в роли специалиста по всем уровням веб-разработки, отдавая предпочтение внештатному сотрудничеству со стартапами на ранних этапах становления. Во время, свободное от кодирования, он ведет блог, пишет книги или выступает на различных мероприятиях в Словении и соседних странах, мечтая сделать доклад на какой-нибудь крупной международной конференции. В ноябре 2012 года он начал писать книгу «Почему программисты работают по ночам» и задался целью улучшить жизнь разработчиков, где бы они ни жили.

Я благодарен @gandalfar и @robertbasic, которые подначивали меня по ходу работы и выступали в роли подопытных свинок для проверки моих примеров. Также выражаю искреннюю признательность всем членам группы @psywerx, которые не дали мне спятить и создали один из лучших в мире наборов данных.

О рецензенте

Герардо Фуртадо – преподаватель биологии, популяризатор науки, автор книги по эволюционной биологии, опубликованной Бразильским университетом, а также нескольких художественных произведений.

Его второе академическое пристрастие – визуализация данных, именно оно в конечном итоге и привело его к D3.js, самой мощной (и комплексной) JavaScript-библиотеке для этой цели.

Предисловие

Здравствуйте, читатели! В этой книге вы познакомитесь с основами одной из самых распространенных и мощных библиотек визуализации данных, но не только. К концу нашего совместного путешествия вы приобретете все навыки, необходимые настоящему специалисту по D3, и сможете решить любую задачу: от создания визуализации с нуля до запуска ее на сервере и написания автоматизированных тестов. Тех, чьи знания JavaScript несколько подзамшели, ждет приятный сюрприз – эта книга располагает к использованию самых последних добавленных в язык средств, и мы всегда объясняем, почему это круто и чем отличается от «старой школы».

Краткое содержание книги

В главе 1 «Первое знакомство с D3, ES2017 и Node.js» рассматриваются новейшие средства для создания визуализации данных с помощью D3.

Глава 2 «Начала DOM, SVG и CSS» содержит обзор базовых веб-технологий, используемых в D3.

Глава 3 «Геометрические примитивы в D3» посвящена определению и созданию базовых геометрических элементов, из которых составляются визуализации данных.

В главе 4 «Извлечение пользы из данных» вы научитесь преобразовывать данные, так чтобы D3 могла их визуализировать.

Глава 5 «Все для удобства пользователя» покажет вам, как дополнить визуализацию средствами анимации и интерактивности.

Глава 6 «Иерархические макеты в D3» посвящена иерархической компоновке, которая поднимет ваши навыки владения D3 на новый уровень, где для создания сложных диаграмм применяются повторно используемые шаблоны.

В главе 7 «Прочие макеты» обсуждаются неиерархические макеты, позволяющие ускорить создание диаграмм других типов.

В главе 8 «Использование D3 на сервере с применением Canvas, Koa 2 и Node.js» описаны создание и развертывание веб-службы на базе сервера Node.js, которая отрисовывает визуализации D3 с помощью библиотек Koa.js и Canvas.

В главе 9 «Обретение уверенности в своих визуализациях» показано, как улучшить качество кода за счет проверки соблюдения стандартов кодирования, статической проверки типов и автоматизированного тестирования проектов.

В главе 10 «Проектирование хорошей визуализации данных» сопоставляются различные подходы к визуализации и предлагается набор рекомендаций.

Что необходимо для чтения книги

Вам понадобится компьютер, на котором можно запустить Node.js. В первой же главе мы обсудим, как его установить, а работать он может практически на любой машине, хотя для разработки несколько лишних гигабайтов памяти не помешает. Для нескольких примеров построения карт нужны солидные вычислительные ресурсы, но большинство компьютеров, выпущенных после 2014 года, с такой нагрузкой справится.

Понадобится также последняя версия браузера; я предпочитаю Chrome и разрабатывал примеры в нем, однако Firefox тоже годится. Можете также попробовать Safari, Internet Explorer/Edge, Opera или любой другой браузер, но, на мой вкус, инструменты разработчика в Chrome самые лучшие.

На кого рассчитана эта книга

На разработчиков веб-приложений, авторов интерактивных новостей, специалистов по анализу и обработке данных и всех, интересующихся интерактивным представлением данных в вебе с помощью библиотеки D3. Предполагается знакомство с основами JavaScript, но предварительный опыт визуализации данных или работы с D3 не потребуется.

Графические выделения

В этой книге тип информации обозначается шрифтом. Ниже приведено несколько примеров с пояснениями.

Фрагменты кода внутри абзаца, имена таблиц базы данных, папок и файлов, URL-адреса, данные, которые вводит пользователь, и адреса в Твиттере выделяются следующим образом: «Если в сообщении говорится что-то типа '`Command not found`', проверьте, все ли правильно установлено, и убедитесь, что Node.js включен в переменную среды `$PATH`».

Кусок кода выглядит так:

```
"babel": {
  "presets": [
    "es2017"
  ]
},
"main": "lib/main.js",
"scripts": {
  "start": "webpack-dev-server --inline",
},
```

Входная и выходная информация командных утилит выглядит так:

```
$ brew install n
$ n lts
```

Новые термины и **важные фрагменты** выделяются полужирным шрифтом. Например, элементы графического интерфейса в меню или диалоговых окнах выглядят в книге так: «Мы в основном будем пользоваться вкладками **Elements** и **Console**: вкладка **Elements** предназначена для просмотра DOM, а **Console** – для ввода JavaScript-кода и анализа ошибок».

Предупреждения и важные примечания выглядят так.

Советы и рекомендации выглядят так.

Отзывы

Мы всегда рады отзывам читателей. Расскажите нам, что вы думаете об этой книге – что вам понравилось или, быть может, не понравилось. Читательские отзывы важны для нас, так как помогают выпускать книги, из которых вы черпаете максимум полезного для себя.

Чтобы отправить обычный отзыв, просто пошлите письмо на адрес feedback@packtpub.com, указав название книги в качестве темы. Если вы являетесь специалистом в некоторой области и хотели бы стать автором или соавтором книги, познакомьтесь с инструкциями для авторов по адресу www.packtpub.com/authors.

Поддержка клиентов

Счастливым обладателям книг Packt мы можем предложить ряд услуг, которые позволят извлечь из своего приобретения максимум пользы.

Загрузка кода примеров

Вы можете скачать код примеров к этой книге из своей учетной записи на сайте http://www.packtpub.com. Если книга была куплена в другом месте, зайдите на страницу http://www.packtpub.com/support, зарегистрируйтесь, и мы отправим файлы по электронной почте.

Для скачивания файлов с кодом выполните следующие действия:

1) зарегистрируйтесь или зайдите на наш сайт, указав свой адрес электронной почты и пароль;
2) наведите мышь на вкладку **SUPPORT** в верхней части страницы;
3) щелкните по ссылке **Code Downloads & Errata**;
4) введите имя книги в поле **Search**;
5) выберите интересующую вас книгу;
6) с помощью выпадающего меню укажите, где вы приобрели книгу;
7) нажмите **Code Download**.

Загруженный файл можно распаковать, воспользовавшись последними версиями программ:

- WinRAR / 7-Zip для Windows;
- Zipeg / iZip / UnRarX для Mac;
- 7-Zip / PeaZip для Linux.

Код к этой книге имеется также на странице сайта GitHub по адресу https://github.com/PacktPublishing/D3.js-4.x-Data-Visualization. По адресу https://github.com/PacktPublishing/ размещены также код и видео к другим книгам из нашего обширного каталога. Полюбопытствуйте!

Опечатки

Мы проверяли содержимое книги со всей тщательностью, но какие-то ошибки все же могли проскользнуть. Если вы найдете в нашей книге ошибку, в тексте или в коде, пожалуйста, сообщите нам о ней. Так вы избавите других читателей от разочарования и поможете нам сделать следующие издания книги лучше. При обнаружении опечатки просьба зайти на страницу http://www.packtpub.com/support, выбрать книгу, щелкнуть по ссылке **Errata Submission Form** и ввести информацию об опечатке. Проверив ваше сообщение, мы поместим информацию об опечатке на нашем сайте или добавим ее в список замеченных опечаток в разделе **Errata** для данной книги.

Список ранее отправленных опечаток можно просмотреть, выбрав название книги на странице http://www.packtpub.com/books/content/support. Запрошенная информация появится в разделе **Errata**.

Нарушение авторских прав

Незаконное размещение защищенного авторским правом материала в Интернете – проблема для всех носителей информации. В издательстве Packt мы относимся к защите прав интеллектуальной собственности и лицензированию очень серьезно. Если вы обнаружите незаконные копии наших изданий в любой форме в Интернете, пожалуйста, незамедлительно сообщите нам адрес или название веб-сайта, чтобы мы могли предпринять соответствующие меры.

Просим отправить ссылку на вызывающий подозрение в пиратстве материал по адресу copyright@packtpub.com.

Мы будем признательны за помощь в защите прав наших авторов и содействие в наших стараниях предоставлять читателям полезные сведения.

Вопросы

Если вас смущает что-то в этой книге, вы можете связаться с нами по адресу questions@packtpub.com, и мы сделаем все возможное для решения проблемы.

Глава 1

Первое знакомство с D3, ES2017 и Node.js

Библиотека **Data-Driven Documents (D3)**, разработанная Майком Бостоком (Mike Bostock) и сообществом D3 в 2011 году, пришла на смену прежней библиотеке Бостока Protovis. Она поддерживает отрисовку данных с точностью до пикселя благодаря абстрагированию вычисления масштабов и осей с помощью простого предметно-ориентированного языка (DSL) и благодаря использованию идиом, понятных всякому, кто работал с популярной JavaScript-библиотекой jQuery. В D3, как и в jQuery, нужно сначала выбрать множество элементов, а затем применить к ним операции с помощью цепочки функций-модификаторов. В контексте визуализации данных такой декларативный подход оказывается гораздо проще многих других имеющихся инструментов. На официальном сайте по адресу https://d3js.org/ приведено много примеров, демонстрирующих мощь D3, но поначалу разобраться в них нелегко. Прочитав эту книгу, вы будете знать D3 в достаточной степени, чтобы понять примеры и приспособить их к своим нуждам. Если вы хотите познакомиться с разработкой D3 поближе, скачайте исходный код с сайта GitHub по адресу https://github.com/d3.

В этой главе мы заложим фундамент для выполнения представленных в книге примеров. Я объясню, как писать код на языке ECMAScript 2017 (ES2017) – последней и самой передовой версии JavaScript – и как с помощью программы Babel транслировать его на язык ES5, который понимает любой современный браузер. Затем мы познакомимся с основами D3 v4 на примере создания простой диаграммы.

Что такое D3.js?

Благодаря точному контролю и элегантности D3 заслуженно считается одной из самых мощных библиотек визуализации с открытым исходным кодом. Но это также означает, что для простых задач – изображения одного-двух линейных графиков – она не слишком подходит; в этом случае лучше взять какую-нибудь библиотеку для рисования графиков. Кстати, многие из них сами основаны на D3. Обширный список опубликован по адресу https://github.com/sorrycc/awesome-javascript#data-visualization.

D3 построена на принципах функционального программирования, которое ныне переживает второе рождение в сообществе JavaScript. Эта книга не о функциональном программировании, но многое покажется знакомым тем, кто уже сталкивался с этими принципами. Если вы не из их числа или привыкли к объектно-ориентированному программированию, не расстраивайтесь, я буду объяснять все необходимое по ходу дела и надеюсь, что раздел о функциональном программировании в начале главы 4 поможет вам понять, почему эта парадигма так полезна, особенно в задачах визуализации данных и конструирования приложений.

Что случилось с классами?

Во втором издании этой книги было много примеров использования механизма классов, появившихся в языке ES2015. Но теперь мы используем *фабричные функции*, а ключевое слово `class` ни разу не встречается. С чего бы это?

В ES2015 классы на самом деле были *синтаксическим сахаром* поверх фабричных функций, т. е. в конечном счете компилировались в фабричные функции. Хотя с помощью классов можно повысить уровень организации сложного кода, в итоге они просто скрывают происходящее внутри. К тому же использование объектно-ориентированных парадигм, в т. ч. классов, идет вразрез с одним из самых действенных и элегантных аспектов языка JavaScript – полноправными функциями и объектами. Ваш код станет проще и изящнее, если вы будете придерживаться функциональных парадигм, да и примеры, созданные сообществом D3, читать будет проще, поскольку в них классы почти никогда не используются.

Против использования классов можно привести еще много аргументов, для изложения которых здесь просто нет места. Рекомендую обратиться к великолепной серии статей Эрика Эллиота «The Two

Pillars of JavaScript» по адресу www.medium.com/javascript-scene/the-two-pillars-of-javascript-ee6f3281e7f3.

Что нового в версии D3 v4?

Одно из главных событий, случившихся после выхода предыдущего издания этой книги, – выпуск версии 4.

Из многочисленных изменений самым важным является полный пересмотр пространства имен D3. Это означает, что ни один пример, приведенный в этой книге, не будет работать в версии D3 3.x, а примеры из второго издания книги «Learning D3.js Data Visualization» не будут работать в D3 4.x. Это, пожалуй, самое страшное, что жестокий м-р Босток мог учинить над такими авторами учебников, как я (шучу, шучу). Но шутки в сторону – из-за этого многие примеры кода, разработанные сообществом D3, потеряли актуальность и могут даже показаться странными тому, для кого эта книга стала первым знакомством с библиотекой. Поэтому так важно проверять, для какой версии D3 написан пример, – если для 3.x, то стоит поискать аналогичный пример для 4.x, чтобы не пасть жертвой когнитивного диссонанса.

Обычно в примерах версия D3 указывается в тегах `script` в начале кода. Если вы видите такой код:

```
<script src="https://d3js.org/d3.v3.min.js"></script>
```

то пример написан для версии 3.x. А если такой:

```
<script src="https://d3js.org/d3.v4.min.js"></script>
```

то это более современный пример, рассчитанный на версию 4, т. е. вы движетесь в правильном направлении.

С этим связано также разбиение D3 на много библиотек меньшего размера (микробиблиотек). Вы можете пойти по одному из двух путей:

- работать с D3 как с единой библиотекой (монобиблиотекой) так же, как в версии 3;
- включать в проект лишь отдельные компоненты D3 (микробиблиотеки).

В этой книге принят первый подход. Использование микробиблиотек заметно усложнило бы изучение D3, хотя при этом уменьшается размер конечного комплекта файлов, который должны будут загрузить пользователи для просмотра вашей графики. Тем не менее я буду отмечать, в каком пакете находится та или иная функциональность,

а вы, когда получше освоитесь с D3, сможете перейти на микробиблиотеки, вместо того чтобы включать все подряд, нужное и ненужное.

Что такое ES2017?

Одно из главных изменений в этой книге с момента ее первого издания – упор на современный JavaScript, в данном случае ES2017. Эта версия, ранее называвшаяся ES6 (Harmony), – крупный шаг в развитии языковых средств JavaScript, позволивший реализовать новые паттерны, которые упрощают восприятие кода и повышают его выразительность. Если вы писали на JavaScript раньше и примеры в этой главе кажутся вам странными, значит, вы имели дело с прежним, более распространенным синтаксисом ES5.

Но не переживайте! На привыкание к новому синтаксису не уйдет много времени, а я уж постараюсь объяснять новые языковые возможности по мере надобности. Хотя поначалу кривая обучения может показаться довольно крутой, в конце вы будете писать код куда лучше и окажетесь на переднем крае современной разработки на JavaScript.

Отличное введение во все новшества, которые принес нам ES2015-17, можно найти в руководстве, составленном авторами программы Babel.js (https://babeljs.io/docs/learn-es2015/), которой мы будем постоянно пользоваться на страницах этой книги.

Прежде чем двигаться дальше, я хотел бы развеять некоторые мифы о том, чем на самом деле является ES2017. Первоначально версии стандартов ECMAScript (сокращенно ES) нумеровались последовательно: ES4, ES5, ES6, ES7. Но начиная с ES6 этот порядок был изменен, поскольку важно было отразить тот факт, что новые стандарты выходят ежегодно, чтобы не отставать от современных тенденций разработки. Так что текущий стандарт относится к 2017 году. Важной вехой стала версия ES2015, которая примерно соответствует ES6. Стандарт ES2016, ратифицированный в июне 2016 года, основывается на предыдущем стандарте с добавлением нескольких исправлений и двух новых возможностей. ES2017 пока находится на стадии проекта, т. е. новые предложения рассматриваются и разрабатываются – до ратификации, которая произойдет в течение 2017 года. Эта книга писалась, когда новые средства еще не были утверждены, и, возможно, они даже не войдут в стандарт ES2017, так что с их официальным включением, быть может, придется подождать до выхода следующей версии стандарта.

Но это не причина для беспокойства, поскольку мы все равно используем Babel.js для компиляции всего кода на ES5, чтобы он одинаково работал в Node.js и в браузере. Для полноты картины я буду указывать, в какой версии стандарта появилось то или иное средство (например, модули были включены в ES2015), но, говоря о JavaScript, я всегда имею в виду современный JavaScript безотносительно к номеру спецификации ECMAScript.

Запускаем Node и Git из командной строки

В этой книге я постараюсь не выказывать предпочтений какому-либо конкретному редактору или операционной системе (хотя сам использую Atom в macOS X), но все-таки какие-то предварительные условия должны быть соблюдены.

Первое из них – Node.js. В настоящее время Node широко применяется для веб-разработки и, по сути дела, представляет собой средство для выполнения JavaScript-кода из командной строки. Ниже я покажу, как написать серверное приложение для Node, а пока займемся просто установкой Node и `npm` (замечательного менеджера пакетов для Node).

Если вы работаете в Windows или macOS X без Homebrew, то воспользуйтесь установщиком по адресу https://nodejs.org/en/. Если в вашей системе на базе macOS X имеется Homebrew, то я рекомендую вместо этого установить пакет `n`, который позволяет без труда переключаться с одной версии Node на другую:

```
$ brew install n
$ n lts
```

Пользователям Windows знак `$` может показаться непонятным. В операционных системах на базе UNIX обычный пользователь видит в командной строке приглашение `$`, а суперпользователь *root* – приглашение `#`. Включая знак `$`, я показываю, что вы должны выполнять команды от имени обычного пользователя, а не суперпользователя.

В любом случае по завершении проверьте результат, выполнив такие команды:

```
$ node --version
$ npm --version
```

Если будут напечатаны версии `node` и `npm`, значит, все хорошо.

Я работаю соответственно с версиями 6.5.0 и 3.10.3. Ваша версия может отличаться, но важно, чтобы версия Node была не ниже 6.0.0.

Если в сообщении говорится что-то типа `Command not found`, проверьте, все ли правильно установлено, и убедитесь, что Node.js включен в переменную среды `$PATH`.

В этой книге мы с помощью Babel и Webpack будем преобразовывать наш изысканный модульный современный код на JavaScript в нечто, что могут выполнить даже самые захудалые, побитые молью браузеры (ау, Internet Explorer 9!).

Для начала создадим файл `package.json`, в котором будут храниться версии всех нужных нам зависимостей. Для этого создадим новую папку и выполним команду `npm init`:

```
$ mkdir d3-projects
$ cd d3-projects
$ npm init -y
```

Флаг `-y` означает, что `npm init` должна использовать параметры по умолчанию, не задавая никаких вопросов.

Затем с помощью `npm` установим все необходимое:

```
$ npm install "babel-core@^6" "babel-loader@^6" "babel-preset-es2017@^6"
"babel-preset-stage-0@^6" "webpack@^2" "webpack-dev-server@^2" css-loader
style-loader json-loader --save-dev
```

Эта команда установит версию 2 Webpack, версию 6 Babel и комплект начальных установок и дополнительных модулей для того и другого. Имена и номера версий программ будут записаны в файл `package.json`, так что для повторной установки понадобится всего лишь ввести команду

```
$ npm install
```

Еще установим D3:

```
$ npm install d3 --save
```

Далее нужно создать конфигурационный файл для Webpack. Не буду объяснять, что означает каждая директива, все комментарии вы сможете найти в репозитории кода к этой книге. Просто сохраните показанный ниже код в файле `webpack.config.js`:

```
const path = require('path');
module.exports = [{
```

```
  entry: {
    app: ['./lib/main.js'],
  },
  output: {
    path: path.resolve(__dirname, 'build'),
    publicPath: '/assets/',
    filename: 'bundle.js',
  },
  devtool: 'inline-source-map',
  module: {
    rules: [{
      test: /.js?$/,
      exclude: /(node_modules|bower_components)/,
      loader: 'babel-loader',
    }, {
      test: /.json$/,
      loader: 'json-loader',
    }, {
      test: /.css$/,
      loader: 'style-loader!css-loader',
    }],
  },
}];
```

Напоследок отредактируем файл `package.json`, добавив несколько аббревиатур, которые упростят нам жизнь. После строки, начинающейся словом `name`, вставьте такие строки:

```
"babel": {
  "presets": [
    "es2017"
  ]
},
"main": "lib/main.js",
"scripts": {
  "start": "webpack-dev-server --inline",
},
```

Все это необходимо, если вы начинаете новый проект с нуля.

Можно вместо этого клонировать репозиторий книги из GitHub. GitHub – это место, где размещается большая часть всего открытого и иного кода в мире. Я не только положил туда примеры и тестовые данные, но и поработал над конфигурацией. Далее я буду исходить из предположения, что вы клонировали репозиторий, создав копию на своей машине. Для этого выполните такие команды:

```
$ git clone https://github.com/aendrew/learning-d3-v4
$ cd learning-d3-v4
```

В результате будут клонированы среда разработки и все примеры в каталоге `learning-d3-v4/`, после чего мы сделаем этот каталог текущим и установим все зависимости с помощью `npm`.

Есть еще один вариант: разветвить репозиторий на GitHub и клонировать свое ответвление вместо моего. Это позволит публиковать результаты своего труда в облаке, упростит получение помощи от коллег, даст возможность отображать завершенные проекты на страницах GitHub и даже отправлять исправления и дополнения в родительский проект. Что, в свою очередь, будет способствовать улучшению будущих изданий этой книги. Чтобы разветвить проект `aendrew/learning-d3-v4`, нажмите кнопку «fork» на сайте GitHub и замените строку `aendrew` в приведенном выше фрагменте кода своим именем пользователя GitHub.

Каждая глава книги хранится в отдельной ветке. Для переключения между главами выполните команду вида

```
$ git checkout chapter1
```

Вместо `1` укажите номер интересующей вас главы. Но пока останемся в главной ветке. Чтобы вернуться к ней, введите такую команду:

```
$ git stash save && git checkout master
```

В главной ветке вы будете писать свой код по мере работы над книгой. Установить зависимости все равно нужно, сделаем это прямо сейчас:

```
$ npm install
```

Весь исходный код, с которым вы будете работать, находится в папке `lib/`. Обратите внимание, что там имеется только файл `main.js`; почти всегда мы будем иметь дело именно с ним, поскольку `index.html` – всего лишь минимальный контейнер, в котором отображается результат нашей работы. Ниже он приведен целиком, и это первый и последний раз, когда в книге встретится HTML-код:

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="utf-8">
  <title>Learning Data Visualization with D3.js</title>
  </head>
  <body>
  <script src="assets/bundle.js"></script>
  </body>
</html>
```

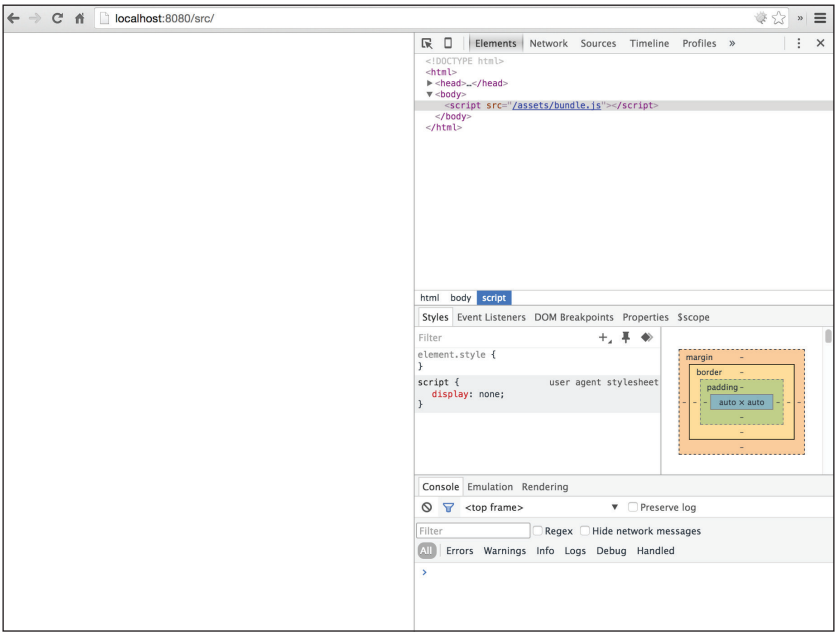
Имеется также таблица стилей в файле `styles/index.css`; пока она пуста, но скоро мы начнем ее заполнять.

Следующий шаг – запустить сервер разработки:

```
$ npm start
```

Эта команда запускает сервер Webpack, который преобразует наш новомодный код на JavaScript стандарта ES2017 в совместимый с ES5 и отправляет его браузеру.

Теперь введите в адресной строке Chrome (или другого браузера, я не привередливый – лишь бы только не Internet Explorer!) строку `localhost:8080` и откройте консоль разработчика (*Ctrl+Shift+J* в Linux и Windows, *Option+Command+J* – в Mac). Появятся пустая страница и пустая консоль JavaScript с приглашением для ввода кода:



Краткое введение в инструменты разработчика Chrome

Инструменты разработчика Chrome – неоценимое средство для разработки веб-приложений. В большинстве современных браузеров

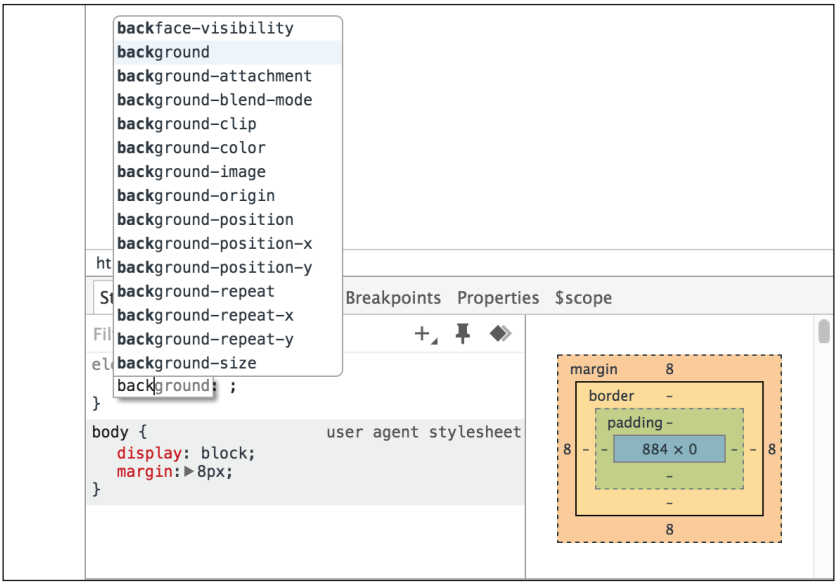
есть нечто подобное, но мы для простоты ограничимся только тем, что предлагает Chrome. Вы вполне можете выбрать другой браузер. Особенно хорош Firefox Developer Edition и говорят – да-да, ребята у меня за спиной, я слышу, – что Opera тоже неплох!

Мы в основном будем пользоваться вкладками **Elements** и **Console**: вкладка **Elements** предназначена для просмотра DOM, а **Console** – для ввода JavaScript-кода и анализа ошибок. Остальные шесть вкладок нужны для более крупных проектов.

С помощью вкладки **Network** можно узнать, сколько времени ушло на загрузку файлов, и изучить Ajax-запросы. Вкладка **Profiles** позволяет профилировать JavaScript-код. Вкладка **Resources** полезна для исследования данных на стороне клиента, а вкладки **Timeline** и **Audits** – когда в программе имеется глобальная переменная, приводящая к утечке памяти, и вы пытаетесь понять, почему вдруг Chrome «отъел» 500 МБ оперативной памяти. Мне доводилось пользоваться всеми этими вкладками в ходе разработки с использованием D3, но, пожалуй, наибольшую пользу они все-таки приносят при создании крупных веб-приложений с применением таких каркасов, как React и Angular.

Но главное, что нас будет интересовать, – это вкладка **Sources**, на которой показаны все исходные файлы, загруженные веб-страницей. Мало того, что мы видим, какой код на самом деле загружается, так еще эта вкладка содержит полнофункциональный отладчик JavaScript, которым решаются воспользоваться немногие простые смертные. Описывать технику отладки программы скучно, и вообще, это не тема данной главы, но все же расстановка точек останова вместо включения бесконечных предложений `console.log`, для того чтобы понять, что делает программа, – умение, которое принесет вам немало пользы в будущем. Хорошее руководство имеется на странице https://developers.google.com/web/tools/chrome-devtools/javascript/breakpoints.

Однако в основном вы будете использовать Инструменты разработчика для исследования CSS-стилей в правой части вкладки **Elements**. Здесь можно узнать, какие правила CSS влияют на стилизацию элемента, это позволяет отыскать зловредные правила, которые все портят. Можно также отредактировать CSS и сразу же увидеть результат своих действий.



Неизбежный пример – столбчатая диаграмма

Никакое введение в D3 нельзя считать полным без примера простой столбчатой диаграммы. В контексте D3 они играют ту же роль, что программа «Здравствуй, мир» в языках программирования, при этом 90% любого рассказа о данных можно свести к демонстрации подходящей диаграммы или графика. Хорошие примеры можно найти в статьях из *Financial Times* или *Economist* – там часто изложение завершается итоговым линейным графиком или гистограммой. Поскольку по роду деятельности я занимаюсь разработками для отдела новостей (раскрою секрет: я работаю в отделе интерактивной графики редакции *Financial Times*), многие мои примеры так или ина-

че связаны с текущими событиями или темами, для подачи которых полезна визуализация данных. Сообщество поддержки новостных редакций приложило немало усилий для создания среды, в которой процветает D3, и для любого целеустремленного журналиста уверенное владение инструментами типа D3 становится все более важным профессиональным навыком.

Нашим первым набором данных будут результаты подсчета голосов на референдуме по брекситу, опубликованные избирательной комиссией Великобритании. Мы построим столбчатую диаграмму, отражающую явку по регионам. Источник данных – http://www.electoralcommission.org.uk/find-information-by-subject/elections-and-referendums/past-elections-and-referendums/eu-referendum/electorate-and-count-information.

Мы создадим по одному столбику для каждого региона Великобритании. Первый шаг – настроить простой контейнер, который затем будет заполнен с помощью нашего прелестного JavaScript-кода. Можно либо сразу перейти к коду, либо кое-что подготовить, чтобы потом было проще. Для начала двинемся по первому пути. Скоро на вас обрушится целая лавина новомодного JavaScript-кода, поэтому пока постараемся не усложнять.

Откройте файл `lib/main.js` и добавьте в него свою первую строку, имеющую отношение к библиотеке D3:

```
const chart = d3.select('body')
  .append('svg')
  .attr('id', 'chart');
```

Здесь мы выбираем HTML-элемент `<body>`, добавляем в его конец элемент `<svg>` и назначаем ему идентификатор `#chart`. Эта схема будет встречаться нам очень часто.

Прежде чем двигаться дальше, обратим внимание на первую конструкцию современного JavaScript:
Ключевое слово `const` используется для определения переменной, которая не будет существенно изменяться. Говоря *существенно*, я имею в виду, что как-то изменяться она все-таки может (например, можно добавить элементы в массив или модифицировать объект), но попытка присвоить ей другое значение приведет к исключению. Исключение возбуждается и при попытке использовать переменную до ее объявления. В отличие от других языков, в JavaScript константы ограничены областью видимости текущей функции (они не являются глобальными, если специально не сделать их таковыми). Это весьма полезно в функциональных программах, поскольку

позволяет предотвратить странные ошибки, вызванные поднятием переменных (необычная особенность JavaScript, заключающаяся в том, что переменные интерпретируются в начале замыкания функции, а не там, где они фактически определены). Дополнительные сведения о `const` имеются по адресу http://mdn.io/const.
Еще один способ определения переменных в JavaScript дает ключевое слово `let`, которое действует как `var`, но подобно `const` имеет блочную область видимости, т. е. видимость переменной ограничена блоком, предложением или выражением, в котором она используется. Это тоже помогает предотвратить трудные для отладки ошибки. Подробнее см. http://mdn.io/let.

Когда употреблять одно, а когда другое? Ключевое слово `const` лучше использовать, если вы не собираетесь присваивать переменной другое значение, а `let` – если собираетесь. Я стараюсь не переприсваивать значения переменным, поэтому обычно использую `const`. Хотя в современном JavaScript употребление ключевого слова `var` не запрещено, делать этого не стоит – используйте `let` или `const`, отдавая предпочтение `let`, если не уверены, что правильно в данной ситуации.

Пора! Откроем браузер, не забыв предварительно запустить сервер разработки (для его запуска выполните `npm start`), и перейдем по адресу `http://localhost:8080`.

Uncaught Error: Cannot find module “d3”

М-да. Нехорошо вышло...

Ошибка связана с тем, что мы еще не импортировали D3. Если вам уже доводилось работать с D3, то, наверное, вы привыкли, что она присоединяется к глобальному объекту `window`. Именно это происходит, когда файл d3.js включается с помощью тега `<script>`. Но мы этого делать не будем, а воспользуемся новой функциональностью ES2015 – импортом модулей!

Вернемся к файлу `main.js`. В самом начале добавьте строку

```
import * as d3 from 'd3';
```

Разберемся, что это значит. Все предложения импорта должны находиться в начале файла (никаких потайных вызовов `require()` внутри функций в стиле Node.js!). Это открывает возможность для статического анализа, так что новые инструменты JavaScript оказываются более эффективными. Каждое такое предложение начинается ключевым словом `import`.

Дальше может следовать фигурная скобка. Модуль ES2015 допускает экспорт двух видов:

- **именованный**: в этом случае в фигурных скобках указываются имена конкретных экспортируемых членов (хотя их можно переименовывать);
- **по умолчанию**: такой член в модуле может быть только один, при импорте его можно назвать как угодно. Пример будет показан ниже.

В предложении выше мы импортировали все входящие в D3 микробиблиотеки в пространство имен `d3`.

Вернитесь в браузер и перейдите на вкладку **Elements**: вы увидите новый элемент SVG с идентификатором `#chart` в конце страницы. Это уже прогресс!

Загрузка данных

Вернемся к `main.js`. Нам нужно как-то получить данные, и ниже я покажу, как это сделать правильно, а пока последуем по старому, трудному и неприятному пути – воспользуемся объектом `XMLHttpRequest`:

```
const req = new window.XMLHttpRequest();
req.addEventListener('load', mungeData);
req.open('GET', 'data/EU-referendum-result-data.csv');
req.send();
```

Здесь мы создаем новый объект `XMLHttpRequest`, просим его загрузить файл из каталога `data` и передаем полученные данные функции `mungeData()`, которую скоро напишем.

Вы обратили внимание на использование уродливого ключевого слова `new`? И на то, что для объявления переменной понадобились четыре строки? И на то, что ответ обрабатывается функцией обратного вызова? Бр-р-р! Но ничего, в последующих главах мы все исправим. Единственное преимущество такого подхода – в том, что он работает практически в любом браузере без всяких полифилов[1]. Но есть множество способов сделать это лучше, и мы рассмотрим их в главе 4.

В загруженном CSV-файле каждому избирательному округу Великобритании соответствует одна строка, содержащая разнообразные данные: процент проголосовавших за каждый вариант, явка избирателей, количество недействительных или испорченных бюллетеней и т. д. Мы преобразуем этот файл в массив объектов, представляющих средний процент проголосовавших за выход в более крупных территориальных образованиях.

---

[1] См. https://ru.wikipedia.org/wiki/Полифил. – *Прим. перев.*

Пора уже написать функцию `mungeData()`. Для преобразования строки CSV-файла в объект мы воспользуемся функцией `d3.csvParse()` (из микробиблиотеки `d3-dsv`), а затем применим функции из микробиблиотеки `d3-array` для обработки этих объектов:

```
function mungeData() {
  const data = d3.csvParse(this.responseText);
  const regions = data.reduce((last, row) => {
    if (!last[row.Region]) last[row.Region] = [];
    last[row.Region].push(row);
    return last;
  }, {});
  const regionsPctTurnout = Object.entries(regions)
    .map(([region, areas]) => ({
      region,
      meanPctTurnout: d3.mean(areas, d => d.Pct_Turnout),
    }));

  renderChart(regionsPctTurnout);
}
```

Да, кстати, вот и еще одно новшество ES2015! Вместо того чтобы раз за разом набирать `function() {}`, мы теперь можем определить анонимную функцию, написав просто `() => {}` . Во-первых, мы сэкономили пять ударов по клавишам, а во-вторых, *двойная стрелка* не привязывает значение `this` к какому-то другому объекту. Поскольку мы применяем функциональный стиль программирования, это не столь важно, но если бы мы использовали классы, то это существенно упростило бы жизнь. Подробнее см. http://mdn.io/Arrow_functions.

Преобразование данных состоит из трех шагов.

1. Сначала преобразуем данные в массив объектов с помощью функции `d3.csvParse()` и присвоим результат переменной `data`.
2. Затем преобразуем массив в объект, в котором ключами являются названия регионов, а значениями – массивы входящих в регион избирательных округов.
3. И наконец, `Object.entries` преобразует объект в многомерный массив, состоящий из элементов, содержащих пары ключ-значение, который затем можно редуцировать в новый объект, содержащий названия регионов и среднюю процентную явку, вычисленную по всем входящим в регион избирательным округам.

Вы, наверное, обратили внимание на необычную сигнатуру функции `Array.prototype.map`:

```
.map(([region, areas]) => {
```

Здесь используется новая возможность ES2015 – *деструктурирующее присваивание*, – чтобы сопоставить элементу массива временное имя. Обычно сигнатура обратного вызова имеет вид:

```
function(item, index, array) {}
```

Но поскольку мы знаем, что `item` – массив с двумя элементами, то можем поименовать каждый из них, упростив тем самым чтение кода (в этот раз мы не использовали аргументов `index` и `array`, но если бы они понадобились, то мы поместили бы их после деструктурирующей конструкции).

И напоследок преобразованные данные передаются еще не написанной функции `renderChart()`, которую мы добавим ниже.

Можно также просто добавить к написанному выше коду такие строки:

```
const regionsPctTurnout = d3.nest()
  .key(d => d.Region)
  .rollup(d => d3.mean(d, leaf => leaf.Pct_Turnout))
  .entries(data);
```

Функция `d3.nest()` входит в состав микробиблиотеки `d3-collection`, которая будет рассмотрена в главе 4. D3 – беспристрастная библиотека в том смысле, что одну и ту же задачу можно решить разными способами, и зачастую среди них нет единственно правильного. Я буду демонстрировать различные подходы, а вы уж сами выбирайте, какой вам больше нравится.

Двенадцать (плюс-минус) столбиков

Настало время изобразить данные на экране.

Добавим в файл `main.js` новую функцию `renderChart()`:

```
function renderChart(data) {
  chart.attr('width', window.innerWidth)
    .attr('height', window.innerHeight);
}
```

Она всего лишь устанавливает размеры ранее созданной переменной `chart` равными размерам окна. Мы почти у цели, держитесь крепче!

Но сначала нужно определить масштабы, которые описывают, как D3 отображает данные на значения пикселей. Иначе говоря, масштаб – это просто функция, которая отображает входную область определения в выходную область значений. На случай, если это труд-

но запомнить, я беззастенчиво украл упражнение из великолепного пособия Скотта Мюррея по масштабам, приведенного в книге «Interactive Data Visualization for the Web»:

Когда я говорю «входная», ты говоришь «область определения».
Когда я говорю «выходная», ты говоришь «область значений».
Понятно? Поехали.
Входная! Область определения!
Выходная! Область значений!
Входная! Область определения!
Выходная! Область значений!
Запомнил? Ну и отлично.

Глупо, конечно, но я часто ловлю себя на том, что бормочу эти строки, когда сроки поджимают, а я поздно ночью работаю над какой-нибудь диаграммой. Попробуйте сами!

Далее добавим в `renderChart()` такой код:

```
const x = d3.scaleBand()
  .domain(data.map(d => d.region))
  .rangeRound([50, window.innerWidth - 50])
  .padding(0.1);
```

Теперь масштаб `x` – функция, которая отображает область определения, состоящую из названий регионов, в область значений от `50` до ширины окна просмотра минус `50` с промежутками, которые определяются функцией `.padding()` с аргументом 0.1. В результате мы создали зонный масштаб, который похож на обычный с тем отличием, что выходная область разделена на секции. Мы еще поговорим о масштабах ниже.

В этом примере задана фиксированная ширина полей 50, которая передается масштабам и другим функциям. Произвольное число, зашитое в код, часто называют *магическим числом*, понимая под этим, что всякий читающий код видит какое-то непонятное значение, при котором код работает, как по волшебству. Это плохо, никогда так не поступайте – такой код трудно читать, а кроме того, если вы захотите изменить значение, то придется найти все места, где оно встречается. Я лишь хотел привлечь ваше внимание к этому факту. Далее мы будем определять подобные величины более разумным способом, следите за новостями!

Все еще внутри `renderChart()` определим еще один масштаб `y`:

```
const y = d3.scaleLinear()
  .domain([0, d3.max(data, d => d.meanPctTurnout)])
  .range([window.innerHeight - 50, 0]);
```

Масштаб `y` отображает линейную область определения (от нуля до максимального значения данных, которое возвращает функция `d3.max`) в область значений от `window.innerHeight` (минус поле высотой `50` пикселей) до `0`. Инвертировать диапазон обязательно, потому что D3 считает, что верхней точке графика соответствует значение `y=0`. Обнаружив, что диаграмма D3 нарисована вверх ногами, попробуйте инвертировать область значений в одном из масштабов.

Теперь определим оси. Добавьте в конец `renderChart` следующий код:

```
const xAxis = d3.axisBottom().scale(x);
const yAxis = d3.axisLeft().scale(y);
```

Мы сказали, какой масштаб использовать на каждой оси при нанесении делений и по какую сторону от оси располагать метки. D3 сама решит, сколько нарисовать делений, где они должны находиться и как их пометить. Поскольку большинство элементов D3 одновременно является объектами и функциями, мы можем изменить внутреннее состояние обоих масштабов, не присваивая результата никакой переменной. Область значений `x` – дискретный список, а область значений `y` – диапазон от `0` до максимального значения в наборе данных.

Теперь нарисуем оси:

```
chart.append('g')
  .attr('class', 'axis')
  .attr('transform',
    &grave;translate(0, ${window.innerHeight - 50})&grave;)
  .call(xAxis);
```

Еще одно новшество ES2015! Аргумент атрибута `transform` заключен в обратные апострофы (`), это шаблонная литеральная строка. Она отличается от обычной строки в двух отношениях: внутри допускаются символы новой строки, а благодаря синтаксической конструкции `${}` мы можем вычислить и подставить в строку произвольное JavaScript-выражение. В данном случае просто выводится значение `window.innerHeight`, но, в принципе, это может быть любое выражение, возвращающее нечто, похожее на строку, например `Array.prototype.join`, – тогда будет подставлено содержимое всего массива. Удобно, ничего не скажешь.

Мы поместили в диаграмму элемент `g`, сопоставили элементу `axis` CSS-класс и с помощью атрибута `transform` перенесли его в левый нижний угол диаграммы.

Наконец, мы вызвали функцию `xAxis`, предоставив D3 возможность сделать все остальное.

Вторая ось рисуется так же, только аргументы другие:

```
chart.append('g')
  .attr('class', 'axis')
  .attr('transform', 'translate(50, 0)')
  .call(yAxis);
```

Итак, метки расставлены, осталось только нарисовать данные:

```
chart.selectAll('rect')
  .data(data)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.region))
  .attr('y', d => y(d.meanPctTurnout))
  .attr('width', x.bandwidth())
  .attr('height', d => (window.innerHeight - 50) - y(d.meanPctTurnout));
```

Кода много, но смысл его очень прост:

- для каждого прямоугольника (`rect`) в графике загрузить данные;
- обойти данные;
- для каждого элемента добавить прямоугольник;
- задать атрибуты прямоугольника.

Не обращайте внимания, что в начальный момент нет ни одного прямоугольника; здесь мы по существу создаем выделение, привязанное к данным, и производим над ним операции. Могу понять, какое странное чувство возникает, когда оперируешь несуществующими элементами (когда я изучал D3, это стало для меня главным камнем преткновения), но это идиома, полезность которой станет ясна позднее, когда мы начнем добавлять и удалять элементы в соответствии с изменяющимися данными.

Масштаб `x` помогает вычислять горизонтальные позиции, а функция `bandwidth()` дает ширину столбика. Масштаб `y` вычисляет вертикальные позиции, и мы вручную получаем высоту каждого столбика от `y` до нижней оси. Отметим, что когда значения атрибута для каждого элемента разные, атрибут задается в виде функции (`x`, `y` и `height`), в противном случае – в виде значения (`width`).

Теперь добавим «бантик» – заставим каждый столбик вырастать из горизонтальной оси. Добро пожаловать в мир анимации!

Модифицируйте только что добавленный код, как показано ниже; изменившиеся строки выделены полужирным шрифтом:

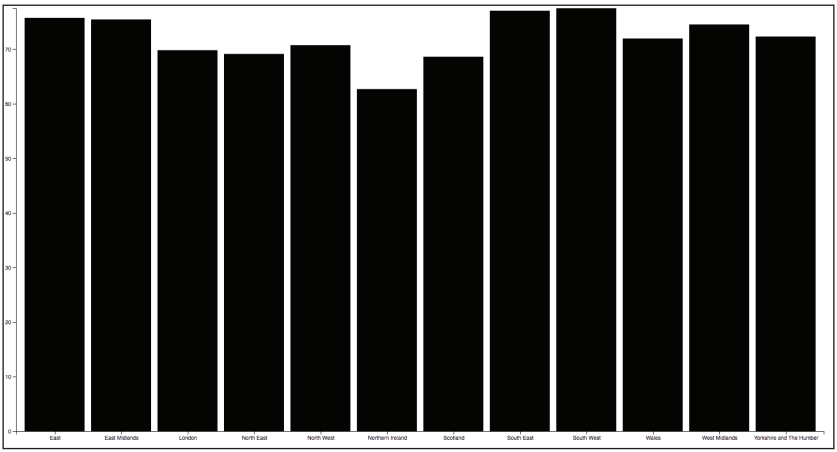
```
chart.selectAll('rect')
  .data(data)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', d => x(d.region))
.attr('y', window.innerHeight - 50)
  .attr('width', x.bandwidth())
  .attr('height', 0)
    .transition()
    .delay((d, i) => i * 20)
    .duration(800)
    .attr('y', d => y(d.meanPctTurnout))
    .attr('height', d =>
      (window.innerHeight - 50) - y(d.meanPctTurnout));
```

Разница в том, что мы статически сместили все столбики вниз (`window.innerHeight - 50`), присвоив им нулевую высоту, а затем инициировали анимацию функцией `.transition()`. После этого мы определяем нужный нам переход.

Прежде всего мы хотим отложить переход каждого столбика на 20 миллисекунд относительно предыдущего, задав аргумент `i*20`. Большинство обратных вызовов D3 возвращает некоторое значение (привязанное к элементу, обычно оно обозначается `d`) и индекс (порядковый номер обрабатываемого элемента, обычно обозначаемый `i`), а аргумент `this` при этом ссылается на текущий выбранный элемент DOM. Если бы мы использовали классы, то этот последний факт был бы важен, т. к. в противном случае мы работали бы в контексте объекта `rect` типа `SVGElement`, а не в том, который нам действительно нужен. Но поскольку мы в большинстве случаев пользуемся фабричными функциями, знать, какой контекст связан с `this`, не так важно.

В результате гистограмма постепенно появляется слева направо, а не возникает сразу, как черт из табакерки. Далее мы говорим, что каждая анимация должна длиться чуть меньше секунды – `.duration(800)`. А в конце определяем конечные значения анимированных атрибутов – `y` и `height` – такие же, как в предыдущем коде. Обо всем остальном позаботится D3.

Сохраните файл и обновите окно браузера. Если все пойдет по плану, то должна появиться такая диаграмма:



Как видим, явка на референдуме по вопросу выхода из ЕС была довольно высокой, а самой высокой она оказалась на юго-западе. Прикиньте – мы только что решили задачу из области журналистики данных! На случай, если у вас не получилась такая диаграмма, напомним, что полностью код доступен по адресу http://github.com/aendrew/learning-d3-v4/tree/chapter1.

Но кое-чего нам пока не хватает, главным образом CSS-стилей элементов SVG.

Можно было бы просто добавить CSS в HTML-код, но тогда пришлось бы открывать этот мерзкий файл `index.html`. К тому же кому захочется писать HTML-код, когда изучаешь новомодный JavaScript?

Прежде всего создадим файл `index.css` в каталоге `styles/`:

```
html, body {
  padding: 0;
  margin: 0;
}

.axis path, .axis line {
  fill: none;
  stroke: #eee;
  shape-rendering: crispEdges;
}

.axis text {
  font-size: 11px;
}
```

```
.bar {
  fill: steelblue;
}
```

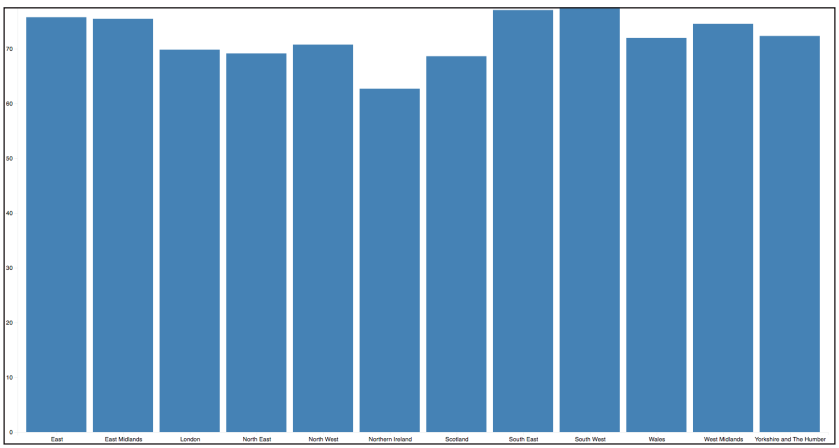
Затем добавим такую строку в начало `main.js`:

```
import * as styles from 'styles/index.css';
```

Чума, правда?! Тегов `<style>` вообще не понадобилось!

Отметим, что если в `require()` или `import` упомянуто что-то, не являющееся JS-файлом, значит, это результат работы загрузчика Webpack. Хотя автор – большой поклонник Webpack, на самом деле мы всего лишь импортируем стили в `main.js` посредством Webpack, а не затребуем их глобально с помощью тега `<style>`. Это хорошо, поскольку вместо загрузки на сервер десятков файлов на этапе развертывания законченного кода мы развертываем только один оптимизированный комплект. Можно также ограничить область действия правил CSS тем местом, где они включаются. Есть и другие полезные трюки, о которых можно прочитать по адресу https://github.com/webpack-contrib/css-loader.

Глядя на этот CSS, мы понимаем, зачем нужно было добавлять классы к фигурам. Теперь в процессе стилизации мы можем ссылаться на элементы напрямую. Мы сделали оси тонкими, раскрасили их светло-серым цветом и уменьшили шрифт меток. Сами столбики стали голубыми. Сохраните файл и обновите страницу. Вот и готова наша первая диаграмма, созданная с помощью D3:



Рекомендую поэкспериментировать со значениями `.width` и `.height`, чтобы составить представление о возможностях D3. Обратите внимание, как все масштабируется под любой размер, хотя больше мы никакого кода не изменяли. Фантастика!

Резюме

В этой главе вы узнали, что такое библиотека D3, и вкратце познакомились с философией ее работы. Вы настроили свой компьютер для создания прототипов и экспериментов с визуализацией. Далее в книге мы будем предполагать, что среда уже настроена.

Мы проработали простой пример и, пользуясь простейшими средствами D3, создали анимированную гистограмму. Мы узнали о масштабах и осях, о том, что вертикальная ось направлена вниз, что любое свойство, заданное в виде функции, заново вычисляется для каждого элемента данных. Для придания привлекательности изображению мы воспользовались комбинацией CSS и SVG. Мы также познакомились с возможностями ES2017, Babel и Webpack и установили Node.js. Вот какие мы молодцы!

Но главное – в этой главе мы получили в свое распоряжение базовые инструменты, с помощью которых можно самостоятельно поиграть с D3.js. Пробуйте – и вам воздастся! Не бойтесь что-то сломать – вы всегда сможете восстановить код главы в исходном состоянии, выполнив команду `$ git reset --soft origin/chapter1`, где вместо `1` нужно подставить нужный номер главы.

В последующих главах мы рассмотрим все вышеизложенное более глубоко, обращая особое внимание на взаимодействие DOM, SVG и CSS между собой. В этой главе был представлен обширный материал; если что-то осталось непонятным, не переживайте. Просто наберитесь сил для следующей главы, и постепенно все станет на свои места.