УДК 004.738.5+004.43
ББК 32.973.26-018.2
Д75

Дронов В. А.

Д75 Angular 4. Быстрая разработка сверхдинамических Web-сайтов на TypeScript и PHP. — СПб.: БХВ-Петербург, 2018. — 448 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-3334-8

Книга посвящена быстрой разработке сверхдинамических одностраничных Web-сайтов на основе популярного фреймворка Angular 4 и языка программирования TypeScript. Дан вводный курс TypeScript, описаны типизация, классы и интерфейсы, модификаторы доступа, динамические свойства и разбиение программного кода на модули. Рассказано о создании интерфейса сайта посредством компонентов, реализации его бизнес-логики с помощью служб, структурировании программного кода сайта с применением метамодулей. Рассмотрены средства маршрутизации и навигация по сайту. Описано взаимодействие с серверной частью сайта, выгрузка файлов, программирование на языке PHP с применением баз данных MySQL. Рассказано о программировании инструментов разграничения доступа, средствах анимации, написании сложных таблиц стилей на языке LESS, тестировании сайтов с применением отладчика Augury и публикации готовых сайтов. Рассмотрен процесс создания полнофункционального сайта, исходные коды которого доступны для загрузки с сайта издательства.

Для Web-программистов

УДК 004.738.5+004.43
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Марины Дамбиевой*

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-5-9775-3334-8

Оглавление

Введение

Когда-то, на заре развития Интернета, Web-сайты представляли собой наборы статических Web-страниц, т. е. обычных документов — точно таких же, как текстовый файл, набираемый в настоящий момент автором. Web-обозреватель загружал с сервера этот документ и выводил на экран его содержание. То была эра *статических сайтов*.

Динамические Web-сайты

Потом в Web-разработке случился коренной перелом: информация, публикуемая на сайтах, стала сохраняться в базах данных, а сами сайты превратились в наборы программ, извлекающих эту информацию и преобразующую ее в обычные Web-страницы. Возник принципиально новый тип сайтов — обрабатывающих информацию, полученную от посетителя, взаимодействующих с ним: поисковые и почтовые службы, интернет-магазины, блоги, социальные сети и др. Web-сайты стали *динамическими*.

Динамические сайты представляют собой набор программ, выполняющихся на серверном компьютере (*серверных программ*). Они генерируют обычные Web-страницы, которые и пересылаются Web-обозревателю. Страницы могут генерироваться на основании результатов обработки информации, полученной от посетителя сайта, или данных, извлеченных из информационной базы (как бывает чаще всего).

В 2000-е годы стали появляться так называемые *фреймворки* — программные библиотеки, реализующие типовые части функциональности среднестатистического сайта и предоставляющие их разработчикам сайтов в виде удобных программных инструментов. Фреймворки позволили программировать сайты максимально быстро и эффективно, не отвлекаясь на рутину.

Но у динамических сайтов есть серьезный недостаток. Поскольку все операции по генерированию Web-страниц выполняются на стороне сервера, серверный компьютер испытывает довольно высокую нагрузку. А если сайт оказывается популярным, и на него ежедневно приходят сотни тысяч или даже миллионы посетителей, в один не очень прекрасный момент времени нагрузка может оказаться непосильной.

У серверного компьютера на обслуживание сайта просто не хватит системных ресурсов, и сайт, на жаргоне пользователей Интернета, "рухнет".

В последнее время, в связи со взрывным ростом числа пользователей Интернета, проблема "обрушения" сайтов стала актуальной как никогда. И разработчики сайтов стали думать, как снизить нагрузку на многострадальные серверы...

Сверхдинамические, или одностраничные, Web-сайты

Идея, как уменьшить бремя серверов, буквально витала в воздухе. Если все операции по генерированию результирующей страницы выполняются на стороне сервера, рассуждали наиболее опытные и дальновидные специалисты по программированию, то мы можем радикально снизить нагрузку на сервер, перенеся бóльшую часть этих операций на сторону клиента, т. е. на Web-обозреватель. Все равно ему, кроме вывода страницы на экран, более особо нечем заняться...

Сказано — сделано! И сделано было следующим образом.

- На сервер были возложены задачи исключительно работы с базой данных: выборки из нее информации и записи в нее сведений, полученных от посетителя.
- Результаты работы серверных программ — выбранные из базы данные — стали пересылаться по Сети не в виде полноценных Web-страниц, а в особом компактном формате. Обычно используется формат *JSON* (JavaScript Object Notation, объектная нотация JavaScript); также возможна пересылка данных в виде обычного текста.
- Сам сайт делался на основе одной-единственной Web-страницы, на которой, в зависимости от ситуации, выводились различные материалы, составляющие сайт. (Их называют *экранами*, но чаще всего — просто страницами, по аналогии с "настоящими" Web-страницами. В этой книге используются оба названия.)
- Вся функциональность такого сайта обеспечивалась набором сложных Web-сценариев. Один из таких сценариев, получив данные от серверной программы, преобразует их в удобный для чтения вид и представляет посетителю в виде отдельного экрана. Другой сценарий собирает данные, занесенные посетителем в Web-форму, отправляет их серверной программе и выводит на другом экране результат их обработки. Третий обрабатывает щелчки на гиперссылках (навигацию) и, в зависимости от того, на какой гиперссылке был выполнен щелчок, выводит тот или иной экран сайта.

Сайты подобного рода получили название *сверхдинамических*, или *одностраничных*, поскольку фактически включают в свой состав всего одну полноценную Web-страницу.

Вот преимущества сверхдинамических сайтов перед динамическими:

- исключительно высокое быстродействие (поскольку страницы не загружаются с сервера целиком, что может занять продолжительное время, а мгновенно генерируются на стороне клиента);

- радикальное снижение нагрузки на сервер (за счет переноса большей части действий по формированию страниц на сторону клиента);
- простота реализации динамических и интерактивных эффектов (наподобие плавной смены одного экрана другим);
- компактность (как правило, сверхдинамические сайты состоят из значительно меньшего количества файлов, чем динамические).

Недостатками сверхдинамических сайтов можно считать несколько более долгую предварительную загрузку (поскольку приходится загружать весьма объемистые Web-сценарии, составляющие программное ядро сайта) и, самое главное, исключительную сложность программирования. И если первый недостаток малосущественен, то второй был способен быстро отравить жизнь даже опытному программисту...

Angular — сверхдинамический фреймворк

...пока не появился фреймворк Angular, реализованный в виде набора Web-сценариев, выполняющийся в среде Web-обозревателя и радикально упрощающий разработку сверхдинамических сайтов. Он предложил:

- разработку экранов одностраничного сайта путем их верстки, а не программирования. Говоря другими словами, чтобы создать экран, мы пишем привычный и компактный HTML- и CSS-код, а не километровую программу на JavaScript;
- разумеется, реализацию типовой функциональности сайта и предоставление ее в виде удобных программных инструментов. Так, Angular сам обеспечивает вывод данных на экран, проверку данных, занесенных в Web-формы, взаимодействие с сервером, навигацию, анимирование элементов и др.;
- разбиение функциональности сайта на отдельные независимые разделы, возможно, подгружаемые по запросу (что положительно сказывается на скорости первоначального открытия сайта);
- удобные средства для генерирования заготовок для программных модулей различных типов;
- средства для построения компактной эксплуатационной редакции сайта;
- высокое быстродействие и исключительную компактность создаваемых с его помощью сайтов.

Разве могли Web-разработчики, занимающиеся созданием одностраничных сайтов, пройти мимо такого сокровища?! Конечно же, нет.

Неудивительно, что до сих пор Angular входит в число наиболее популярных Web-фреймворков.

О чем эта книга?

Да-да, эта книга посвящена Angular и программированию сайтов с его применением! А также всем прочим Web-технологиям, которые обязательно понадобятся или

же окажутся полезными нам в нелегком, но увлекательном деле Angular-программирования. В их числе:

- TypeScript — язык для разработки Web-сценариев, расширяющий возможности JavaScript и устраняющий многие его недостатки; основной язык программирования для Angular. TypeScript-код компилируется в обычный JavaScript;
- LESS — язык для написания таблиц стилей, предоставляющий дополнительные возможности по сравнению с CSS и упрощающий написание сложного представления. LESS-код компилируется в обычный CSS;
- Augury — отладчик Angular-сайтов, выполненный в виде расширения Web-обозревателя Google Chrome.

А еще мы изучим разработку серверных программ, предназначенных для работы вместе с Angular. Эти программы мы будем писать под популярную платформу PHP. Что касается информационных баз, в которых будут находиться данные сайтов, то для их хранения мы выберем не менее популярный формат MySQL, поддерживаемый одноименной СУБД.

А в завершение, в качестве практики, мы создадим полнофункциональный сайт — интернет-представительство фирмы "Садовые растения", продающей садовые растения. Этот сайт можно свободно использовать как основу для разработки других, более сложных решений.

> ***Материалы полнофункционального сайта***
>
> Электронный архив с сайтом "Садовые растения" и дополнительными главами можно скачать с FTP-сервера издательства "БХВ-Петербург" по ссылке **ftp://ftp.bhv.ru/9785977533348.zip** или со страницы книги на сайте **www.bhv.ru** (*см. приложение*).

При работе над книгой автор использовал следующие версии программного обеспечения:

- Microsoft Windows 10, русскую 64-разрядную редакцию со всеми установленными обновлениями;
- OSPanel (ранее — OpenServer) 5.2.2, 64-разрядная редакция — пакет хостинга, включающий в свой состав Web-сервер Apache, платформу PHP и СУБД MySQL;
- Node.js 6.11.2, 64-разрядная редакция — программная платформа, под которой работает среда разработки, входящая в состав Angular;
- фреймворк Angular 4.4.4;
- отладчик Augury — 1.14.0.

Созданные сайты проверялись в Web-обозревателях Microsoft Internet Explorer 11, Microsoft Edge, Google Chrome и Mozilla Firefox актуальных версий.

Для написания программного кода применялся текстовый редактор Notepad++, который можно найти по интернет-адресу **https://notepad-plus-plus.org/**.

Типографские соглашения

В книге будут часто приводиться форматы написания различных конструкций, применяемых в языках TypeScript и LESS. В них использованы особые типографские соглашения, которые мы сейчас изучим.

> ***Внимание!***
>
> Все эти типографские соглашения применяются автором только в форматах написания языковых конструкций. В реальном программном коде они не имеют смысла.

- В угловые скобки (<>) заключаются и дополнительно выделяются курсивом наименования различных значений. В реальный код, разумеется, должны быть подставлены реальные значения. Например:

  ```
  class <имя класса>
  ```

  Здесь вместо подстроки *<имя класса>* должно быть подставлено реальное имя класса.

- В квадратные скобки ([]) заключаются необязательные фрагменты кода:

  ```
  get(<интернет-адрес серверной программы>
  [, <дополнительные параметры>])
  ```

  Здесь *дополнительные параметры* могут присутствовать, а могут и отсутствовать.

- Символом вертикальной черты (|) разделяются параметры или значения, из которых в коде должно присутствовать лишь одно:

  ```
  style(<описание стиля>|<массив описаний стилей>)
  ```

  Здесь допускается указание либо *описания стиля*, либо *массива описания стилей*.

- Слишком длинные, не помещающиеся на одной строке языковые конструкции, автор разрывал на несколько строк и в местах разрывов ставил знаки ↳:

  ```
  move_uploaded_file($_FILES["additionalpics"]↳
  ["tmp_name"][$i], $path2 . $rid . "." . $ext);
  ```

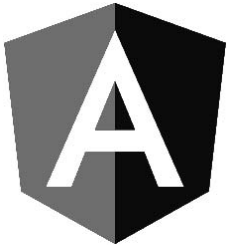
  Приведенный код разбит на две строки, но должен быть набран в одну. Символ ↳ при этом нужно удалить.

- Многоточием (. . .) помечены пропущенные по тем или иным причинам фрагменты кода. Обычно такое можно встретить в исправленных впоследствии фрагментах — приведены лишь собственно исправленные выражения, а оставшиеся неизмененными пропущены.

  Также многоточие используется, чтобы показать, в какое место должен быть вставлен вновь написанный код, — в начало исходного фрагмента, в его конец или в середину, между уже присутствующими в нем выражениями.

> ***Еще раз внимание!***
>
> Все приведенные здесь типографские соглашения имеют смысл только в форматах описания конструкций языков TypeScript и LESS. В коде примеров используется лишь знак ↳ и многоточие.

ЧАСТЬ I



Введение в Angular и язык программирования TypeScript

ГЛАВА 1

Установка Angular и создание нового проекта

Фреймворк AngularJS — предшественник Angular — поставлялся в виде набора обычных файлов сценария с расширением js, в которых хранился код как самого программного ядра фреймворка, так и дополнительных модулей, расширяющих его функциональность. Разработчик просто загружал нужные файлы, сохранял их в корневой папке разрабатываемого сайта и привязывал их к единственной странице этого сайта, применив HTML-тег `<script>`. Затем он писал код сайта, сохранял его в других файлах сценария, которые точно таким же образом привязывал их к странице, после файлов с кодом фреймворка.

Такой подход в программировании сайтов можно назвать традиционным и уж никак нельзя охарактеризовать, как удачный и удобный для разработчика хотя бы потому, что код файла оказывается разделенным на множество файлов, которые нужно не забыть привязать к странице и выстроить в определенном порядке. И если разработчик забудет выполнить привязку какого-либо файла, сайт работать не будет. Более того, часто далеко не сразу удается понять причину возникновения проблемы.

Вот поэтому в Angular, начиная с версии 2, его компания-разработчик Google пошла по другому пути. Можно сказать, что Angular представляет собой не просто фреймворк для быстрого создания сайтов, а настоящую среду разработки. Входящие в комплект этой среды библиотеки и утилиты делают львиную долю рутинной работы по программированию сайта самостоятельно и, в отличие от людей, не совершают при этом ошибок.

Установка необходимого ПО

Перед тем как начать разработку сайтов с применением фреймворка Angular 4, необходимо установить некоторое дополнительное программное обеспечение. Это не займет много времени.

Установка Node.js

Библиотеки и утилиты, составляющие пакет Angular, написаны на языке JavaScript и работают под управлением популярной программной платформы Node.js, также

разработанной компанией Google. Следовательно, первым программным пакетом, который мы установим на свой компьютер, станет Node.js.

Дистрибутив этой программной платформы можно найти по интернет-адресу **https://nodejs.org/en/download/**. Доступны как 32-разрядная, так и 64-разрядная редакции; следует загрузить и установить редакцию, соответствующую установленной редакции Windows.

Установка Node.js особых проблем не вызывает. По ее окончанию рекомендуется проверить, успешно ли установилась платформа, для чего достаточно запустить Командную строку Windows и набрать в ней команду, не забыв завершить ее нажатием клавиши <Enter>:

```
node -v
```

Если Node.js была установлена успешно, будет выведен номер ее версии. Так, у автора этой книги было выведено следующее:

```
v6.11.2
```

Установка Angular CLI

Теперь нам следует установить *Angular CLI* (Angular Command Line Interface — интерфейс командной строки Angular) — утилиту, входящую в состав Angular, запускаемую из Командной строки Windows и выполняющую различные действия: создание нового сайта, добавление в него новых программных модулей различных типов, запуск отладки и пр. Без этой утилиты мы не сможем даже создать новый "пустой" проект Angular, который станет нашим будущим сайтом.

Для установки Angular CLI следует набрать в Командной строке Windows такую команду:

```
npm install -g @angular/cli
```

Здесь:

- `npm` — исполняемый файл утилиты *npm* (Node.js Package Manager — диспетчер пакетов Node.js), входящей в состав Node.js и предназначенной для установки различных программ под эту платформу;
- `install` — команда, указывающая выполнить установку;
- `-g` — ключ, указывающий выполнить *глобальную установку*. В этом случае утилита станет доступной на уровне текущего пользователя Windows и может быть вызвана в любом месте файловой системы;
- `@angular/cli` — имя устанавливаемой утилиты Angular CLI, под которым она зарегистрирована в *репозитории* (хранилище дополнительных библиотек и программ) Node.js.

После нажатия клавиши <Enter> начнется процесс загрузки из репозитория Node.js дистрибутива как самой утилиты Angular CLI, так и всех используемых ею библио-

тек и их установка. Процесс этот длится довольно долго и на не очень производительных компьютерах может отнять до десяти минут, так что придется набраться терпения.

По окончании установки следует проверить, успешно ли она была выполнена. Для этого достаточно в той же Командной строке Windows запустить исполняемый файл утилиты Angular CLI, отдав команду:

```
ng
```

В ответ утилита выведет довольно длинный список поддерживаемых ею команд и их параметров. И это будет признаком правильной установки.

Создание нового проекта

Установив Node.js и Angular CLI и тем самым подготовив почву для работы, мы можем создать новый проект.

Проектом в терминологии Angular называется совокупность:

- файлов с исходным кодом сайта (файлов с программным кодом TypeScript, HTML- и CSS-файлов);
- всех прочих файлов, входящих в состав сайта (графических изображений, аудио- и видеофайлов и др.);
- всех необходимых для работы сайта библиотек, включая сам фреймворк Angular;
- программ и утилит, составляющих среду разработки Angular (компилятора TypeScript, сборщика проекта, отладочного Web-сервера и пр.);
- файлов, хранящих настройки проекта и отдельных утилит из состава среды разработки;
- вспомогательных файлов (например, файла readme.md, содержащего текст, который кратко описывает утилиту Angular CLI и некоторые из поддерживаемых ею команд).

Для простоты можно считать, что проект — это и есть разрабатываемый нами на основе Angular сайт.

Все файлы и папки проекта хранятся в отдельной папке, называемой *папкой проекта*.

Для создания нового проекта следует открыть Командную строку Windows, перейти в папку, в которой будет находиться папка проекта (саму папку проекта создавать необязательно — она будет создана самой Angular CLI), и отдать команду формата:

```
ng new <имя проекта>
```

Команда `new` утилиты Angular CLI создает новый проект с указанным *именем*. В ней можно указать следующие дополнительные ключи, перечислив их после *имени проекта*:

- `--skip-tests|-st` — не создавать тестовые модули (по умолчанию они создаются);
- `--directory|-dir` *<имя папки проекта>* — указывает имя папки проекта (по умолчанию имя папки проекта совпадает с именем самого проекта, указанным после команды `new`).

> ***НА ЗАМЕТКУ***
>
> Вообще-то, команда `new` утилиты Angular CLI поддерживает гораздо больше ключей. Наиболее полезные из них, не рассмотренные здесь, мы изучим позже, по мере ознакомления с книгой.

Пример команды, создающей новый проект с именем `ngtest` без генерирования тестовых модулей (все равно в этой книге они не рассматриваются и не используются):

```
ng new ngtest -st
```

Нажатие клавиши <Enter> запускает процесс создания проекта. Angular CLI сгенерирует файлы с исходным кодом, составляющим изначальный тестовый сайт, установит все необходимые библиотеки и программы. Это также может занять довольно много времени — до десяти минут на малопроизводительных компьютерах.

Содержимое проекта Angular

Далее перечислены наиболее полезные для нас файлы и папки, хранящиеся в папке проекта и составляющие сам проект. Для удобства читателей имена папок набраны в верхнем регистре, имена файлов — в нижнем.

- SRC — папка, хранящая весь исходный код сайта. Вот ее содержимое:
  - APP — папка с кодом всех компонентов, метамодулей, служб, директив, фильтров и классов, составляющих программный код сайта;
  - ASSETS — папка с файлами всех прочих типов, входящими в состав сайта: графическими изображениями, аудио- и видеороликами и пр.;
  - ENVIRONMENTS — папка с конфигурационными файлами, описывающими отладочную и эксплуатационную редакции проекта (о создании отладочной редакции сайта мы поговорим в *главе 13*, а о создании редакции эксплуатационной — в *главе 20*);
  - index.html — стартовая Web-страница, единственная страница, входящая в состав Angular-сайта;
  - favicon.ico — значок сайта;
  - styles.css — глобальная таблица стилей (подробнее о ней будет рассказано в *главе 7*);
  - main.ts — запускающий Web-сценарий, который будет выполнен сразу после открытия сайта (он будет рассмотрен в *главе 3*), произведет инициализацию и запустит сайт на выполнение;
  - polyfills.ts — Web-сценарий совместимости (о нем мы поговорим в конце этой главы);

- NODE_MODULES — папка, содержащая все используемые библиотеки и программы, которые входят в состав среды разработки Angular. Отметим, что в отличие от утилиты Angular CLI в их случае была выполнена *установка локальная*, непосредственно в папку проекта, вследствие чего данные библиотеки и программы доступны только внутри текущего проекта;
- E2E — папка с тестовыми модулями. Несмотря на то, что при создании проекта было указано не создавать тестовые модули, эта папка по какой-то причине все же была создана;
- .angular-cli.json — файл, хранящий конфигурацию Angular CLI применительно к текущему проекту (с этим файлом мы познакомимся в последующих главах);
- package.json — файл, хранящий описание проекта в формате Node.js, а также список всех дополнительных библиотек, используемых проектом;
- readme.md — файл с текстовым описанием текущего проекта и действий, которые можно произвести с ним посредством утилиты Angular CLI.

Прочие файлы, не перечисленные в этом перечне, задают конфигурацию различных программ, входящих в состав среды разработки Angular, или хранят мало интересные для нас служебные данные.

Отладочный Web-сервер Angular и его использование

В начале этой главы говорилось, что Angular — это не только и даже не столько фреймворк, сколько почти полноценная среда разработки. (Вот только текстового редактора в ее составе нет — нам придется применять стороннюю программу такого рода, например Notepad++.) В ее состав входит целый набор программ, в числе которых компилятор TypeScript, сборщик проекта и — внимание! — отладочный Web-сервер.

Отладочный Web-сервер Angular предназначен для отладки сайтов, разработанных с применением данного фреймворка. Он запускается посредством все той же утилиты Angular CLI, для чего нужно в Командной строке Windows обязательно перейти в папку проекта и отдать команду:

```
ng serve
```

Приняв команду `serve`, Angular CLI:

- откомпилирует TypeScript-код, хранящийся в программных файлах, в обычный JavaScript, непосредственно исполняемый Web-обозревателями;
- выполнит *сборку проекта*, т. е.:
  - объединит все уже откомпилированные файлы с программным кодом, входящие в состав проекта, а также код всех используемых библиотек, в том числе и самого Angular, в четыре файла сценария;
  - объединит все таблицы стилей, входящие в состав проекта, в одну.

Все эти действия направлены на сокращение количества загружаемых с Web-сервера файлов и, следовательно, на увеличение скорости загрузки сайта. Нужно отметить, что в качестве сборщика проектов используется популярный программный пакет Webpack.

Еще следует отметить, что все файлы сценария и таблица стилей, полученные в результате выполнения сборки проекта, хранятся исключительно в оперативной памяти компьютера. Это сделано для повышения быстродействия и исключения появления в составе проекта "мусорных" файлов;

- и, наконец, запустит отладочный Web-сервер, по умолчанию использующий TCP-порт 4200.

Для выполнения всех этих действий потребуется некоторое время. Сигналом готовности сайта к работе выступит появление в командной строке сообщения:

```
webpack: Compiled successfully.
```

После чего мы сможем открыть разрабатываемый сайт в любом поддерживаемом Web-обозревателе — Google Chrome, Opera, Mozilla Firefox или Microsoft Edge, набрав интернет-адрес **http://localhost:4200/**. (Как сделать Web-сайт кроссплатформенным, т. е. работающим во всех Web-обозревателях, мы узнаем в конце этой главы.)

Команда `serve` поддерживает большое количество командных ключей, из которых наиболее полезными для начинающих разработчиков являются:

- `--port|-p` *<номер TCP-порта>* — указывает номер TCP-порта, используемого отладочным Web-сервером (по умолчанию, как мы уже знаем, используется порт 4200);
- `--open|-o` — после запуска отладочного Web-сервера открывает сайт в Web-обозревателе, установленном в системе по умолчанию.

Любой новый проект Angular содержит код изначального тестового Web-сайта с минимальным содержанием. Так что мы можем сразу же проверить отладочный Web-сервер в действии.

В Командной строке перейдем в папку только что созданного проекта (предполагается, что до этого момента мы находились в папке, в которую вложена папка проекта), отдав команду:

```
cd ngtest
```

Запустим сервер:

```
ng serve
```

И, дождавшись окончания сборки проекта, откроем тестовый сайт в любом Web-обозревателе, перейдя в нем по интернет-адресу **http://localhost:4200/**. На рис. 1.1 показан тестовый сайт, открытый в Google Chrome.

Только вот заголовок страницы "Welcome to app!" выглядит как-то не очень вдохновляюще... Давайте исправим его. Откроем файл src\app\app.component.ts (путь

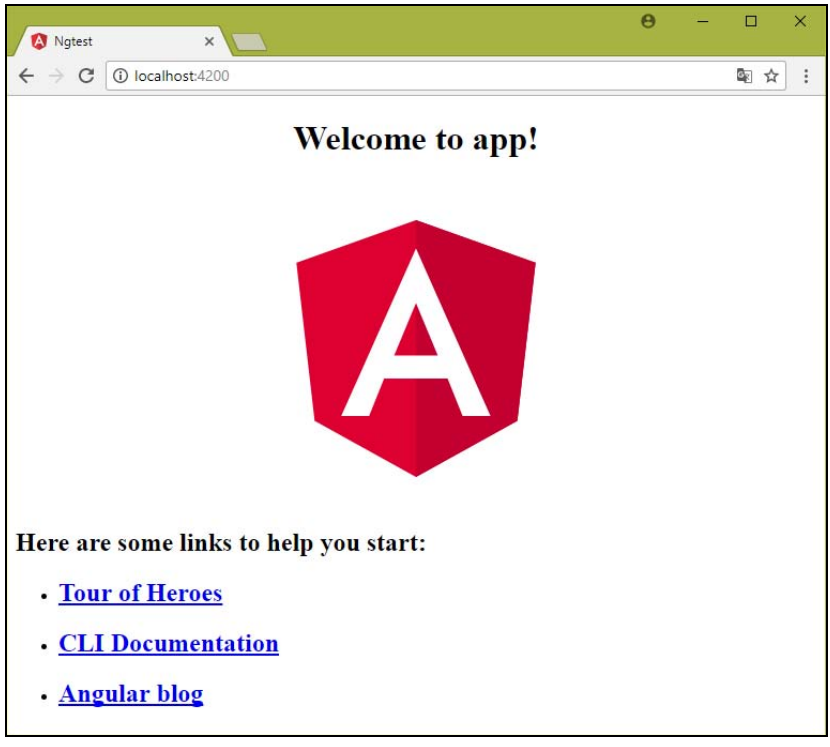


Рис. 1.1. Изначальный тестовый Web-сайт, открытый в Google Chrome

указан относительно папки проекта) в любом текстовом редакторе, например Notepad++, и отыщем в нем строчку:

```
title = 'app';
```

Надо полагать, что она указывает слово, которое будет выводиться на странице в заголовке первого уровня после фрагмента "Welcome to". Давайте изменим его на что-либо другое, например:

```
title = 'Angular 4';
```

Сохраним файл и переключимся на окно Командной строки, в которой мы запустили отладочный Web-сервер. Мы увидим, что, судя по выводящимся сообщениям, после изменения исходного кода сайта сборщик проекта выполняет повторную сборку. А как только он закончит, Web-обозреватель автоматически перезагрузит обновившийся сайт (рис. 1.2).

Как видим, Angular отслеживает изменение файлов с исходным кодом и сразу же после этого запускает повторную компиляцию и сборку проекта. А Web-обозреватель самостоятельно, по команде от отладочного Web-сервера обновляет сайт, что заметно ускоряет и упрощает труд программиста.

Для того чтобы завершить работу отладочного Web-сервера Angular, достаточно переключиться в окно Командной строки, в которой был запущен этот сервер, и нажать комбинацию клавиш <Ctrl>+<Break>. В окне появится текст, предлагающий

Рис. 1.2. Изначальный тестовый Web-сайт после правки

ввести букву "y" для завершения работы программы или "n", если нужно, чтобы она продолжила работу; введем букву "y" и нажмем клавишу <Enter>.

Как сделать Web-сайт кроссплатформенным

Изначальный тестовый сайт, сформированный в новом проекте, мы открывали в одном из поддерживаемых Web-обозревателей, к которым относятся программы Google Chrome, Opera, Mozilla Firefox и Microsoft Edge. Если открыть его в неподдерживаемом Web-обозревателе, скажем, Microsoft Internet Explorer, вместо страницы, показанной на рис. 1.1, мы увидим пустое окно (рис. 1.3).

Дело в том, что Angular при работе активно задействует программные инструменты, поддерживаемые только совместимыми Web-обозревателями; несовместимые же, наподобие Internet Explorer, их не поддерживают. Причем, как мы видим из рис. 1.3, эти инструменты являются ключевыми — в несовместимых Web-обозревателях сайт вообще не отображается.

Однако не все так плохо. Разработчики Angular предусмотрели средства сделать Web-сайты, созданные на основе этого фреймворка, кроссплатформенными, т. е. работающими в любых Web-обозревателях. И средства эти достаточно просты в использовании.

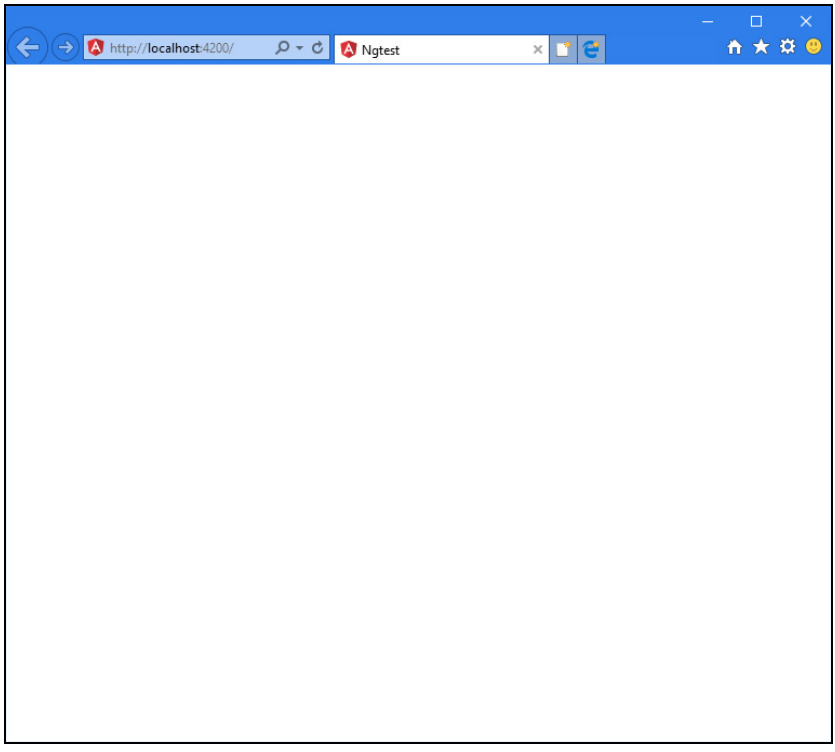


Рис. 1.3. При попытке открыть Angular-сайт в Internet Explorer мы получим пустое окно

Прежде всего остановим отладочный Web-сервер, если он еще запущен. Нам придется внести правки в один из программных файлов и, ко всему прочему, установить три дополнительные библиотеки, которые по какой-то непонятной причине не были установлены утилитой Angular CLI непосредственно при создании нового проекта. Поэтому будет лучше, если отладочный сервер пока отдохнет.

Проверим в Командной строке, перешли ли мы в папку проекта. Нам понадобится установить несколько дополнительных библиотек, а эту операцию следует выполнять, находясь в папке проекта.

Откроем программный файл src\polyfills.ts (путь указан относительно папки проекта). Мы уже знаем, что в нем хранится так называемый *Web-сценарий совместимости*, выполняющий подключение, или, как говорят TypeScript-программисты, импорт программных модулей, которые и обеспечивают сайту кроссплатформенность (об импорте программных модулей мы поговорим в *главе 2*).

Изначально подавляющее большинство этих модулей не импортируется (и, таким образом, сайт не является кроссплатформенным) — выражения, присутствующие в файле src\polyfills.ts и выполняющие импорт, закомментированы. Нам нужно их раскомментировать.

Сначала обратим внимание на следующий набор выражений:

```
// import 'core-js/es6/symbol';
// import 'core-js/es6/object';
```

```
// import 'core-js/es6/function';
// import 'core-js/es6/parse-int';
// import 'core-js/es6/parse-float';
// import 'core-js/es6/number';
// import 'core-js/es6/math';
// import 'core-js/es6/string';
// import 'core-js/es6/date';
// import 'core-js/es6/array';
// import 'core-js/es6/regexp';
// import 'core-js/es6/map';
// import 'core-js/es6/weak-map';
// import 'core-js/es6/set';
```

Они расширяют возможности объектов, встроенных в сам язык JavaScript, добавляя к ним инструменты, что активно использует в работе Angular. Следовательно, эти выражения нужно раскомментировать в любом случае:

```
import 'core-js/es6/symbol';
import 'core-js/es6/object';
. . .
```

Если в разрабатываемом сайте будут использованы графические изображения формата SVG, а в них будут применены директивы `ngClass` (о директивах расскажет *глава 3*, а о директиве `ngClass` — *глава 7*), следует найти выражение:

```
// import 'classlist.js';
```

раскомментировать его:

```
import 'classlist.js';
```

и установить пока что отсутствующую библиотеку classlist.js, отдав в Командной строке команду:

```
npm install --save classlist.js
```

Здесь нам все уже знакомо, кроме ключа `--save`. Он указывает утилите npm дописать устанавливаемую библиотеку в список дополнительных библиотек, хранящихся вместе с параметрами проекта в файле package.json. Если потом мы соберемся переносить наш проект на другой компьютер, этот файл пригодится нам, чтобы быстро установить все нужные библиотеки.

Если мы планируем использовать в разрабатываемом сайте средства анимации Angular, то потребуется раскомментировать выражение:

```
// import 'web-animations-js';
```

И, кроме того, нужно установить дополнительную библиотеку web-animations-js, устраняющую проблемы с анимацией в неподдерживаемых Web-обозревателях. Она устанавливается командой:

```
npm install --save web-animations-js
```

Если при выводе данных мы собираемся задействовать фильтры, выводящие значения денежных сумм, даты и времени соответственно указанным языковым настройкам, следует раскомментировать выражение:

```
// import 'intl';
```

Библиотека intl, обеспечивающая кроссплатформенность, устанавливается командой:

```
npm install --save intl
```

В этом случае также понадобится указать для сайта русские языковые настройки. Это можно сделать, отыскав выражение:

```
// import 'intl/locale-data/jsonp/en';
```

и немного изменив его:

```
import 'intl/locale-data/jsonp/ru-RU';
```

Это выражение укажет языковые настройки для русского языка и России.

Собственно, на этом все. Сохраним файл src\polyfills.ts и запустим отладочный Web-сервер. Как только завершится сборка проекта, откроем изначальный тестовый сайт в Internet Explorer. Мы сразу увидим, что принятые нами в плане обеспечения кроссплатформенности меры пошли на пользу, и Internet Explorer благополучно выведет сайт на экран, как это ранее сделал Chrome (см. рис. 1.2).

ГЛАВА 2

Язык программирования TypeScript

Ранее в этой книге неоднократно упоминался язык программирования TypeScript, на котором пишется программный код Web-сайтов, разрабатываемых на основе фреймворка Angular. Что он собой представляет? Почему в Angular-программировании предпочтительным языком выбран именно он? Почему, наконец, не использовать старый добрый JavaScript?

TypeScript — язык программирования, предназначенный для разработки Web-сценариев и всевозможных программ под платформу Node.js. Он основан на JavaScript, значительно расширяет его возможности и компилируется в обычный JavaScript-код, который, в свою очередь, исполняется Web-обозревателем.

Переход на TypeScript в мире Angular-программирования призван:

- упростить разработку за счет более простого синтаксиса объявления сложных структур данных;
- привести как терминологию языка, так и его синтаксис к стандартам, принятым в других объектно-ориентированных языках программирования (C++, C#, Delphi, Visual Basic, Java, PHP и др.);
- уменьшить количество ошибок исполнения за счет более строгого контроля типов.

Эта глава будет полностью посвящена языку TypeScript. Следует освоить его прежде, чем переходить к рассмотрению основных принципов Angular-программирования, в противном случае нам будет существенно сложнее понять их.

Не следует также рассматривать эту главу как исчерпывающее руководство по TypeScript — этот язык все же достаточно сложен и таит в себе много подводных камней. Мы изучим лишь необходимые основы.

Как уже говорилось ранее, TypeScript основан на JavaScript и, следовательно, очень похож на этот язык. В главе будут рассмотрены, по большей части, различия между этими двумя языками.

Программный код TypeScript хранится в обычных текстовых файлах в кодировке UTF-8. Файлы с TypeScript-кодом должны иметь расширение ts.

Работа с переменными. Типизация

Одна из ключевых особенностей TypeScript — строгая *типизация* переменных. Она заключается в том, что при объявлении переменной для нее указывается тип, после чего эта переменная получает возможность хранить значения только указанного в ее объявлении типа. Попытка присвоения переменной значения любого другого типа приводит к ошибке компиляции.

Помимо этого, имеются некоторые нововведения в плане объявления переменных и возможность создания настоящих констант.

Указание типа переменной

Тип переменной указывается в операторе объявления переменной, сразу после ее имени, и отделяется от него двоеточием. (Между двоеточием, именем переменной и наименованием типа для удобства чтения кода можно поставить пробелы.) После этого переменная сможет хранить только значения указанного типа.

Доступны для указания следующие типы:

- `string`, `number` и `boolean` — стандартные типы JavaScript: строковый, числовой и логический. Примеры:

```
// Объявляем три переменные строкового, числового и логического
// типов соответственно
var s: string = 'TypeScript';
var nn: number;
var flag: boolean = false;

// Присваиваем переменной nn число - успех
nn = 7;
// Пытаемся присвоить переменной flag строку - неудача: несоответствие
// типов
flag = 'JavaScript';
```

- `any` — переменная может хранить значения любых типов. Тот же самый эффект дает объявление переменной без указания типа. Пример:

```
// Объявляем две переменные, способные хранить значения любых типов,
// двумя разными способами
var a: any = 0;
var b = 'Visual Basic';

// Присваиваем этим переменных другие значения, относящиеся к разным
// типам, - успех
a = true;
b = {c: 234, d: 'Angular 4'};
```

- Если нужно, чтобы переменная могла принимать значения нескольких разных типов, эти типы следует перечислить, разделив их символами вертикальной чер-

ты (|). (Между символами вертикальной черты и наименованиями типов можно поставить пробелы.) Пример:

```
// Объявляем переменную, способную хранить значения числового и
// логического типов
var c: number | boolean;

// Присваиваем переменной число - успех
c = 3;
// Присваиваем переменной логическую величину - успех
c = true;
// Присваиваем переменной строку - неудача: несоответствие типов
c = 'TypeScript';
```

- Если переменная должна хранить массив значений определенного типа, используется следующий синтаксис указания типа: *<тип элементов массива>*[]. Пример:

```
// Объявляем массив чисел
var arrN: number[] = [1, 2, 3, 4];
```

- Если переменная должна также хранить значение null, следует либо добавить тип null в список допустимых типов, либо поставить в конце имени переменной вопросительный знак. Пример:

```
// Объявляем две переменные, способные хранить либо строку, либо null
// двумя способами
let x1: string | null = 'Строка 1';
let x2?: string = 'Строка 2';

// Присваиваем обеим переменным null - успех
x1 = null;
x2 = null;
```

Явное указание типа значения для объявляемой переменной позволяет исключить ряд ошибок, связанных с присвоением значения не той переменной или занесением в переменную не того значения. Правила TypeScript-программирования настоятельно рекомендуют указывать тип переменной при ее объявлении.

Объявление переменных

TypeScript привносит кое-что новое в саму операцию объявления переменных. Объявить переменную можно с применением трех различных операторов:

- var — этот оператор ведет себя так же, как и в JavaScript;
- let — ведет себя так же, как var, за следующими исключениями:
  - повторное объявление переменных с его помощью запрещено:

```
// Объявляем переменную - успех
let a;
```

```
// Пытаемся объявить ту же переменную повторно - неудача
let a;
// Пытаемся объявить ту же переменную повторно, но уже оператором
// var - неудача
var a;
```

- переменная доступна только в том блоке, в котором объявлена, и во вложенных в него блоках:

```
// Этот условный оператор будет выполнен в любом случае
if (true) {
  // Внутри блока объявляем две переменные: одну - оператором var,
  // другую - оператором let
  var a1 = 1;
  let a2 = 2;
}
// Вне блока пытаемся извлечь значение переменной a1 - успех
var b1 = a1;
// Пытаемся извлечь значение переменной a2 - неудача:
// такой переменной не существует
var b2 = a2;
```

- `const` — этот оператор ведет себя так же, как `let`, но значение переменной может быть присвоено лишь однажды. Фактически с его помощью мы можем создать *константу* — переменную с неизменяемым значением. Пример:

```
// Объявляем константу
const siteName: string = 'Садовые растения';
// Извлекаем значение константы
let s: string = 'Главная страница :: ' + siteName;
// Пытаемся изменить значение константы - неудача
siteName = 'Комнатные растения';
```

Рекомендуется по возможности объявлять переменные с помощью оператора `let`. Если же значение переменной после присвоения ни разу не изменяется, следует объявить ее как константу — оператором `const`.

Сложные типы данных

TypeScript предоставляет разработчику несколько разновидностей сложных типов данных. Рассмотрим их.

Типизированные экземпляры объектов. Псевдонимы типов

В JavaScript для хранения наборов взаимосвязанных величин часто используются экземпляры объекта `Object`. Чаще всего они создаются с помощью синтаксиса с фигурными скобками; например, так может выглядеть объявление экземпляра объекта `Object`, хранящего описание одного из садовых растений:

```
let obj = {
  id: 8,
  name: 'Гладиолус',
  price: 123.7
};
```

Мы можем объявить тип, указывающий набор свойств, которые должен содержать такой экземпляр объекта, имена и типы значений этих свойств, и указать его для объявляемой переменной, создав тем самым *типизированный экземпляр объекта*. Такой тип создается почти так же, как и сам экземпляр объекта, но вместо значений его свойств указываются их типы.

Вот пример, указывающий для объявляемой переменной тип экземпляра объекта, который содержит числовое свойство `id`, хранящее уникальный номер растения, строковое свойство `name`, хранящее ее название, и также числовое свойство `price`, где находится цена растения:

```
let obj: {
  id: number,
  name: string,
  price: number
};
```

Теперь мы можем присвоить этой переменной любой экземпляр объекта, имеющий заданный в типе набор свойств:

```
obj = {
  id: 8,
  name: 'Гладиолус',
  price: 123.7
};
. . .
obj = {
  id: 7,
  name: 'Саговник',
  price: 314.2
};
```

Но при попытке присвоить переменной экземпляр объекта, имеющий другой набор свойств:

```
obj = {
  cat_id: 2,
  cat_name: 'Розоцветные'
};
```

мы получим ошибку несоответствия типов.

Мы можем объявить своего рода переменную, присвоить ей объявленный тип и использовать "переменную" такого рода для ссылки на созданный тип. Такие "переменные", называемые *псевдонимами типов*, объявляются с применением оператора `type`. Пример:

```
type Plant = {
  id: number,
  name: string,
  price: number
};
let plant1: Plant, plant2: Plant;
```

Литеральные типы

Переменная *литерального типа* может принимать значения из ограниченного набора строк; при попытке присвоения ей значения, не входящего в этот набор, возникает ошибка.

Литеральный тип объявляется перечислением строк, входящих в набор, через символы вертикальной черты и указанием для него псевдонима. Например, так может выглядеть объявление литерального типа, включающего языки программирования, применяемые при написании Web-сценариев:

```
type PL = 'JavaScript' | 'TypeScript';
```

Теперь мы можем объявить переменную такого типа и присвоить ей любое из допустимых строковых значений:

```
let language: PL = 'TypeScript';
```

При попытке присвоить ей строку, не входящую в набор допустимых:

```
language = 'Python';
```

мы получим ошибку несоответствия типов.

Перечисления

Перечисления похожи на литеральные типы за тем исключением, что каждому значению, перечисленному в составе допустимых (*элементу* перечисления), соответствует не строковое, а числовое значение. Вследствие этого перечисления обрабатываются быстрее, чем литеральные типы.

Перечисление создается с применением синтаксиса следующего формата, использующего оператор `enum`:

```
enum <имя перечисления> {<элементы перечисления, записанные через запятую>}
```

Элементы перечисления записываются без кавычек.

Вот пример, объявляющий перечисление — набор языков программирования, применяемых при написании Web-сценариев:

```
enum PLs {JavaScript, TypeScript};
```

Создав тип-перечисление, мы можем объявить переменную этого типа:

```
let language: PLs;
```

Для доступа к отдельным элементам перечисления применяется тот же синтаксис, что и для доступа к свойствам экземпляра объекта: записывается имя перечисления, ставится точка, а за ней — имя нужного элемента. Вот пример:

```
language = PLs.JavaScript;
```

Само перечисление можно рассматривать как массив элементов, чьи индексы совпадают с числовыми значениями, заданными для отдельных элементов перечисления, а значения представляют собой строковые имена этих элементов. Стало быть, мы можем получить строку с именем элемента перечисления, указав этот элемент в квадратных скобках после имени самого перечисления. Так, после выполнения этого примера в переменной `s` окажется строка "TypeScript":

```
let s: string = PLs[PLs.TypeScript];
```

Однако на практике подобное действие приходится выполнять крайне редко. Поэтому наиболее часто применяются *перечисления-константы*, которые:

- не позволяют получить строковое имя элемента перечисления по самому этому элементу;
- но при этом не отнимают системных ресурсов и, следовательно, быстрее обрабатываются.

Перечисления-константы создаются указанием оператора `const` перед оператором `enum`.

Вот пример объявления и использования перечисления-константы:

```
const enum PLs2 {JavaScript, TypeScript};
let language2: PLs2 = PLs2.JavaScript;
```

Функции

Что касается работы с функциями, то TypeScript припас для нас много нововведений в этом плане. Это касается как типизации параметров функции и возвращаемого ей результата, так и объявления в функциях параметров, обрабатываемых особым образом.

Типизация параметров и результата функций. Тип данных *void*

Типизация в TypeScript касается не только переменных. Параметры, принимаемые функцией, равно как и возвращаемый ей результат, также могут быть типизированы. Более того, это настоятельно рекомендуется делать во избежание возникновения ошибок времени выполнения.

Тип параметра в объявлении функции записывается после имени этого параметра и отделяется от него двоеточием, как и в случае переменной. Тип возвращаемого результата ставится после закрывающей круглой скобки и также отделяется от нее двоеточием.

Вот пример объявления функции, принимающей числовой и логический параметры и возвращающей строковый результат:

```
function someFunc(n: number, flag: boolean): string { . . . }
```

А эта функция принимает единственный параметр, чье значение должно быть элементом объявленного нами ранее перечисления `PLs`, и возвращает либо строку, либо `null`:

```
function getLangOptions(lang: PLs): string | null { . . . }
```

Если функция не возвращает результата, в качестве типа возвращаемого значения следует указать тип `void` ("пустой"). Вот пример функции такого рода:

```
function doAllWork(): void { . . . }
```

Необязательные параметры и параметры по умолчанию

На практике часто используются функции, принимающие параметры особого рода:

- *необязательные* — если в вызове функции не задан соответствующий параметру аргумент, параметр получит значение `null` или `undefined`. Разумеется, функция должна правильно обработать такой случай;
- *по умолчанию* — если в вызове функции не указан соответствующий такому параметру аргумент, параметр получит заданное в объявлении функции значение по умолчанию.

Необязательный параметр объявляется так же, как и переменная, способная хранить значение `null`, — указанием в конце его имени вопросительного знака. Вот пример функции, принимающей два параметра, причем второй помечен как необязательный:

```
function someFunc2(n: number, flag?: boolean): string {
  if ((flag === undefined) || (flag === null)) {
    // Второй аргумент не был указан при вызове функции.
    // Обрабатываем такой случай нужным образом.
  }
  . . .
}
```

Эту функцию можно вызвать как с указанием второго аргумента, так и не указывая его:

```
// Вызываем функцию someFunc2, указав оба аргумента
let s1: string = someFunc2(1, false);
// Вызываем функцию someFunc2, указав всего один аргумент
let s2: string = someFunc2(3);
```

Для того чтобы объявить параметр по умолчанию, достаточно в объявлении функции просто присвоить ему нужное значение по умолчанию. Пример функции, принимающей параметр по умолчанию:

```
function getLangOptions2(lang: PLs = PLs.TypeScript): string | null { . . . }
```

Эта функция может быть вызвана с указанием аргумента или без каких-либо аргументов; в последнем случае единственный параметр получит значение по умолчанию — элемент `TypeScript` перечисления `PLs`:

```
let s1: string = getLangOptions2(PLs.JavaScript);
let s2: string = getLangOptions2();
```

Функции, принимающие произвольное количество параметров

Довольно часто также применяются функции, принимающие произвольное количество параметров.

Объявление такой функции должно включать один параметр. Он в качестве значения получит массив всех аргументов, что были указаны при вызове функции. Для этого параметра обязательно следует указать тип массива, а перед его именем поставить три точки (*параметр-массив*).

Вот пример объявления функции, что принимает произвольное количество строковых параметров и возвращает строку, составленную из их значений:

```
function concat(...strs: string[]): string {
  let s = '';
  for (var i = 0; i = strs.length; i++) {
    s += strs[i];
  }
  return s;
}
```

Попробуем вызвать ее, указав разное количество аргументов:

```
let s1: string = concat('Java', 'Script');
// Результат: 'JavaScript'
let s2: string = concat('P', 'H', 'P');
// Результат: 'PHP'
let s3: string = concat();
// Результат: '' (пустая строка)
```

Мы даже можем объявить функцию, принимающую некоторое количество обязательных параметров. В таком случае обязательные параметры в объявлении указываются перед параметром-массивом.

Вот объявление функции, принимающей один обязательный параметр и произвольное количество необязательных:

```
function concat2(prefix: string, ...strs: string[]): string {
  let s = prefix;
  for (var i = 0; i = strs.length; i++) {
    s += strs[i];
  }
  return s;
}
```

Попробуем вызвать ей:

```
let s4: string = concat2('Язык ', 'C', '+', '+');
// Результат: 'Язык C++'
```

Функции-стрелки

Функция-стрелка — это просто более компактный синтаксис для написания обычной анонимной функции JavaScript. Этот синтаксис выглядит следующим образом:

```
([<список параметров, перечисленных через запятую>]) => {<тело функции>}
```

Как видим, *список параметров* и *тело функции* разделяют знаки "равно" и "больше", образующие нечто напоминающее стрелку. Отсюда функция такого рода и получила свое название.

Пример функции-стрелки, принимающей два параметра, возвращающей частное от их деления и сохраненной в переменной (не забываем, что функция-стрелка анонимна, и если мы не сохраним ее в переменной, то не сможем использовать):

```
let f1 = (a, b) => { return a / b; };
```

И параметры функции-стрелки, и возвращаемый ей результат можно и нужно типизировать. Пример:

```
let f1 = (a: number, b: number): number => { return a / b; };
```

Вызвать функцию-стрелку можно так же, как и обычную именованную функцию:

```
let r1 = f1(4, 2);
```

Также поддерживается упрощенный формат записи функций-стрелок:

```
([<список параметров, перечисленных через запятую>]) => <выражение>
```

Такая функция вернет значение, полученное в результате вычисления *выражения*. Отметим, что *выражение* записывается без фигурных скобок. Пример:

```
let f1 = (a: number, b: number): number => a / b;
```

Функции-стрелки часто применяются в качестве аргументов других функций или методов.

Классы и объекты

Но наиболее значительно отличия TypeScript от JavaScript проявляются в плане обработки самых сложных типов данных: объектов и их экземпляров. Отличия проявляются настолько сильно, что в TypeScript для таких типов даже введена совершенно отдельная терминология.

- То, что в JavaScript называется объектом, в TypeScript носит название *класса*.
- Привычные нам по JavaScript экземпляры объектов в TypeScript называются *объектами*.