6 JavaScript и TypeScript: часть 2

В этой главе будут представлены некоторые расширенные возможности JavaScript, применяемые в разработке Angular. Я объясню, как JavaScript работает с объектами (включая поддержку классов) и как функциональность JavaScript упаковывается в модули JavaScript. Также будут представлены некоторые возможности, предоставляемые компилятором TypeScript и не входящие в спецификацию JavaScript; они встречаются в некоторых примерах книги. В табл. 6.1 приведена краткая сводка материала главы.

Таблица 6.1. Сводка материала главы

Проблема	Решение	Листинг
Создание объекта с заданием его свойств	Используйте ключевое слово new или объектный литерал	1–3
Создание объекта по шаблону	Определите класс	4, 5
Наследование поведения от другого класса	Используйте ключевое слово extends	6
Группировка функциональности JavaScript	Создайте модуль JavaScript	7
Объявление зависимости в модуле	Используйте ключевое слово import	8–12
Объявление типов свойств, параметров и переменных	Используйте аннотации TypeScript	13–18
Определение множественных типов	Используйте объединенные типы	19–20
Группировка разнотипных значений	Используйте кортежи	21
Группировка значений по ключу	Используйте индексируемые типы	22
Управление доступом к методам и свойствам класса	Используйте модификаторы управления доступом	23

Подготовка примера

В этой главы мы продолжим использовать проект JavaScriptPrimer из главы 5. Никакие изменения не потребуются, а выполнение следующей команды из папки JavaScriptPrimer запустит компилятор TypeScript и сервер HTTP для разработки:

```
npm start
```

Откроется новое окно браузера с контентом, показанным на рис. 6.1.

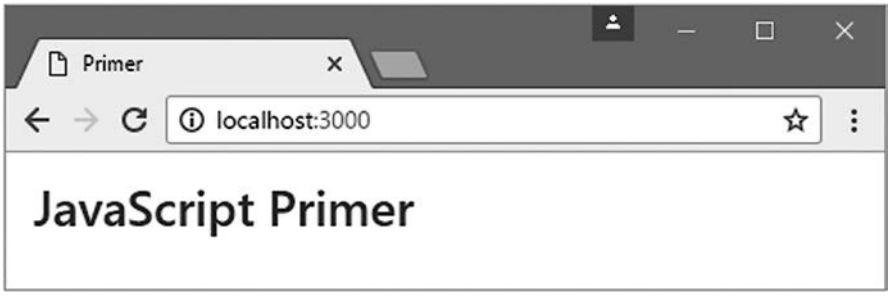

Рис. 6.1. Выполнение примера

Примеры этой главы используют консоль JavaScript в браузере для вывода сообщений. Взглянув на консоль, вы увидите следующий результат:

```
Total value: $2864.99
```

Работа с объектами

В JavaScript поддерживаются разные способы создания объектов. В листинге 6.1 представлен простой пример.

ПРИМЕЧАНИЕ

В некоторых примерах этой главы компилятор TypeScript выводит сообщения об ошибке. Примеры все равно работают, и на сообщения можно не обращать внимания; они появляются из-за того, что TypeScript предоставляет дополнительные возможности, которые будут описаны позже в этой главе.

Листинг 6.1. Создание объекта в файле primer.ts

```
let myData = new Object();
myData.name = "Adam";
myData.weather = "sunny";

console.log("Hello " + myData.name + ".");
console.log("Today is " + myData.weather + ".");
```

Объект создается вызовом `new Object()`, после чего результат (созданный объект) присваивается переменной с именем `myData`. После того как объект будет создан, его свойства определяются простым присваиванием значений:

```
...
myData.name = "Adam";
...
```

До выполнения этой команды у объекта не было свойства `name`. После выполнения команды такое свойство существует, и ему присвоено значение `Adam`. Чтобы

прочитать значение свойства, укажите имя переменной и имя свойства через точку:

```
...
console.log("Hello " + myData.name + ".");
...
```

Результат выглядит так:

```
Hello Adam.
Today is sunny.
```

Объектные литералы

Определить объект и его свойства можно за один шаг. Для этого используется формат объектных литералов, продемонстрированный в листинге 6.2.

Листинг 6.2. Использование формата объектного литерала в файле primer.ts

```
let myData = {
    name: "Adam",
    weather: "sunny"
};

console.log("Hello " + myData.name + ". ");
console.log("Today is " + myData.weather + ".");
```

Имя каждого определяемого свойства отделяется от его значения двоеточием (`:`), а свойства разделяются запятыми (`,`). Этот синтаксис приводит к тому же результату, что и предыдущий пример; на консоль будут выведены следующие сообщения:

```
Hello Adam.
Today is sunny.
```

Функции как методы

Одна из самых полезных возможностей JavaScript — добавление функций к объектам. Функция, определяемая для объекта, называется *методом*. В листинге 6.3 показано, как добавлять методы к объекту.

Листинг 6.3. Добавление методов к объекту в файле primer.ts

```
let myData = {
    name: "Adam",
    weather: "sunny",
    printMessages: function () {
        console.log("Hello " + this.name + ". ");
        console.log("Today is " + this.weather + ".");
    }
};
myData.printMessages();
```

В этом примере функция используется для создания метода с именем `printMessages`. Обратите внимание: для обращения к свойствам, определяемым объектом, используется ключевое слово `this`. При использовании функции как метода функции объект, для которого был вызван метод, неявно передается в специальной переменной `this`. Вывод листинга выглядит так:

```
Hello Adam.
Today is sunny.
```

Определение классов

Классы представляют собой шаблоны, используемые для создания объектов с идентичной функциональностью. Поддержка классов, недавно включенная в спецификацию JavaScript, должна была привести JavaScript в соответствие с другими популярными языками программирования; она широко применяется в разработке Angular. Листинг 6.4 показывает, как функциональность, определяемая объектом из предыдущего раздела, может выражаться с использованием класса.

Листинг 6.4. Определение класса в файле primer.ts

```
class MyClass {

    constructor(name, weather) {
        this.name = name;
        this.weather = weather;
    }

    printMessages() {
        console.log("Hello " + this.name + ". ");
        console.log("Today is " + this.weather + ".");
    }
}

let myData = new MyClass("Adam", "sunny");
myData.printMessages();
```

Классы JavaScript знакомы всем программистам, работавшим на любом из популярных языков (таких, как Java или C#). Ключевое слово `class` используется для объявления класса; за ним следует имя класса, в данном случае `MyClass`. Конструктор — функция, вызываемая при создании нового объекта на основе класса, — предоставляет возможность получить данные и провести инициализацию, необходимую для класса. В примере конструктор определяет параметры `name` и `weather`, которые используются для создания одноименных переменных. Переменные, определяемые таким образом, называются *свойствами* (properties).

Классы также могут содержать методы, которые определяются как функции, но без обязательного использования ключевого слова `function`. В этом примере представлен один метод с именем `printMessages`, который использует значения свойств `name` и `weather` для вывода сообщений на консоль JavaScript в браузере.

ПРИМЕЧАНИЕ

Классы также могут содержать статические методы, помеченные ключевым словом static. Статические методы принадлежат классу, а не объектам, созданным на основе этих классов. Пример статического метода встречается в листинге 6.14.

Ключевое слово `new` используется для создания объекта на основе класса:

```
...
let myData = new MyClass("Adam", "sunny");
...
```

Команда создает новый объект, при этом класс `MyClass` используется в качестве шаблона. `MyClass` используется в этой ситуации как функция, а переданные аргументы будут получены функцией-конструктором, определяемой классом. Результатом этого выражения является новый объект, который присваивается переменной с именем `myData`. После того как объект будет создан, вы сможете обращаться к его свойствам и методам через переменную, которой объект был присвоен:

```
...
myData.printMessages();
...
```

Пример выводит на консоль JavaScript в браузере следующий результат:

```
Hello Adam.
Today is sunny.
```

КЛАССЫ JAVASCRIPT И ПРОТОТИПЫ

Поддержка классов не изменяет основополагающий механизм работы с типами в JavaScript. Вместо этого классы просто предоставляют способ использования типов, более знакомый большинству программистов. Во внутренней реализации JavaScript продолжает использовать традиционную систему типов, основанную на прототипах. Например, код из листинга 6.4 можно записать так:

```
var MyClass = function MyClass(name, weather) {
    this.name = name;
    this.weather = weather;
}

MyClass.prototype.printMessages = function () {
    console.log("Hello " + this.name + ". ");
    console.log("Today is " + this.weather + ".");
};

var myData = new MyClass("Adam", "sunny");
myData.printMessages();
```

Классы упрощают разработку приложений Angular, поэтому в книге я использовал именно этот способ. Многие возможности, появившиеся в ES6, относятся к категории «синтаксических удобств»; иначе говоря, они упрощают понимание и использование некоторых аспектов JavaScript. У JavaScript есть свои странности, и многие из «синтаксических удобств» помогают разработчику избежать типичных ошибок.

Определение свойств с get- и set-методами

Классы JavaScript могут определять свойства в конструкторе; в результате создается переменная, которую можно читать и изменять в любой точке приложения. Свойства с get- и set-методами выглядят за пределами класса как обычные свойства, но они позволяют ввести дополнительную логику, которая может использоваться для проверки или преобразования новых значений или генерирования их на программном уровне, как показано в листинге 6.5.

Листинг 6.5. Get- и set-методы в файле primer.ts

```
class MyClass {
    constructor(name, weather) {
        this.name = name;
        this._weather = weather;
    }

    set weather(value) {
        this._weather = value;
    }

    get weather() {
        return `Today is ${this._weather}`;
    }

    printMessages() {
        console.log("Hello " + this.name + ". ");
        console.log(this.weather);
    }
}

let myData = new MyClass("Adam", "sunny");
myData.printMessages();
```

Get- и set-методы реализуются в виде функций, перед которыми ставится ключевое слово `get` или `set`. В классах JavaScript отсутствует понятие уровней доступа, а по общепринятой схеме имена внутренних свойств начинаются с символа подчеркивания (`_`). В листинге свойство `weather` реализуется set-методом, который обновляет внутреннее свойство с именем `_weather`, и get-методом, который встраивает значение `_weather` в строковый шаблон.

Пример выводит следующий результат на консоль JavaScript в браузере:

```
Hello Adam.
Today is sunny
```

Наследование

Классы могут наследовать поведение от других классов при помощи ключевого слова `extends`, как показано в листинге 6.6.

Листинг 6.6. Наследование в файле primer.ts

```
class MyClass {
    constructor(name, weather) {
```

```
        this.name = name;
        this._weather = weather;
    }

    set weather(value) {
        this._weather = value;
    }

    get weather() {
        return `Today is ${this._weather}`;
    }

    printMessages() {
        console.log("Hello " + this.name + ". ");
        console.log(this.weather);
    }
}

class MySubClass extends MyClass {

    constructor(name, weather, city) {
        super(name, weather);
        this.city = city;
    }

    printMessages() {
        super.printMessages();
        console.log(`You are in ${this.city}`);
    }
}

let myData = new MySubClass("Adam", "sunny", "London");
myData.printMessages();
```

Ключевое слово `extends` используется для объявления класса, от которого наследует определяемый класс; этот класс называется *суперклассом*, или *базовым классом*. В данном примере `MySubClass` наследует от `MyClass`. Ключевое слово `super` используется для вызова конструктора и методов суперкласса. `MySubClass` использует функциональность `MyClass` и дополняет ее поддержкой `city`. На консоль JavaScript в браузере выводится следующий результат:

```
Hello Adam.
Today is sunny
You are in London
```

Работа с модулями JavaScript

Модули JavaScript используются для управления зависимостями в веб-приложениях; это означает, что вам не придется управлять набором элементов `script` в документе HTML. Вместо этого загрузчик модулей сам определяет, какие файлы потребуются для выполнения приложения, загружает эти файлы и выполняет их в правильном порядке. В сложном приложении решать эту задачу вручную достаточно тяжело, и она особенно хорошо подходит для автоматизации.