ГЛАВА 4

Даты

На удивление, в JavaScript реализованы очень мощные механизмы обработки дат, упакованные в несколько старомодный объект `Date`. Как вы увидите, у объекта `Date` есть свои причуды и скрытые ловушки, вроде того что отсчет месяцев начинается с 0, а синтаксический анализ года выполняется в зависимости от регионального стандарта, выбранного на данном компьютере. Но научившись лавировать между этими подводными камнями, вы сможете выполнять множество полезных типовых операций, таких как подсчет количества дней между двумя датами, форматирование дат для вывода на экран, а также установка времени выполнения событий.

4.1. Получение текущих даты и времени

Задача

Получить текущие дату и время.

Решение

В JavaScript есть объект `Date` с хорошими возможностями для управления информацией о дате (и более скромными — для вычислительных операций с датами). При создании нового объекта `Date` в него автоматически заносятся текущие дата и время с округлением до ближайшей миллисекунды в меньшую сторону:

```
const today = new Date();
```

Теперь остается просто извлечь из объекта `Date` нужную информацию. Для решения этой задачи в объекте `Date` есть длинный список методов. Самые важные из них перечислены в табл. 4.1. Обратите внимание на то, что начало отсчета в разных методах может различаться: месяцы и дни недели отсчитываются от 0, а дни нумеруются начиная с 1.

Таблица 4.1. Методы обработки дат для получения различной информации о дате

Метод	Возвращает	Возможные значения
`getFullYear()`	Год	Четырехзначное число, например 2021
`getMonth()`	Номер месяца	Число от 0 до 11, где 0 соответствует январю
`getDate()`	День месяца	Число от 1 до 31
`getDay()`	День недели	Число от 0 до 6, где 6 соответствует воскресенью
`getHours()`	Время суток в часах	Число от 0 до 23
`getMinutes()`	Минуты	Число от 0 до 59
`getSeconds()`	Секунды	Число от 0 до 59
`getMilliseconds()`	Миллисекунды (тысячные доли секунды)	Число от 0 до 999

Вот пример вывода основной информации о текущей дате:

```
const today = new Date();

console.log(today.getFullYear());    // например, 2021
console.log(today.getMonth());       // например, 02 (март)
console.log(today.getDay());         // например, 0 (понедельник)

// Добавим небольшую обработку, чтобы при необходимости
// перед минутами выводились лидирующие нули и получалось
// двузначное число, как "05" в значении времени 4:05
const hours = today.getHours();
const minutes = today.getMinutes().toString().padStart(2, '0');
console.log('Time ' + hours + ':' + minutes);    // например, 15:32
```

Методы Date, перечисленные в табл. 4.1, существуют в двух версиях. В тех, которые представлены в таблице, используются локальные параметры времени. Второй набор методов имеет префикс UTC, например getUTCMonth() и getUTCSeconds(). В них применяется общемировой стандарт времени — всемирное координированное время (Coordinated Universal Time). Если нужно сравнить даты для разных часовых поясов или мест, где приняты разные соглашения о переходе на летнее время, необходимо задействовать UTC-методы. Внутри объекта Date всегда используется стандарт UTC.

Обсуждение

У объекта `Date()` есть несколько конструкторов. Как мы только что выяснили, пустой конструктор создает объект `Date` с текущими датой и временем. Но можно создать объект `Date` и для другой даты, указав год, месяц и день:

```
// 10 февраля 2021 года:
const anotherDay = new Date(2021, 1, 10);
```

Подчеркнем еще раз: помните о разных точках отсчета (месяцы начинаются с 0, а дни — с 1). Это значит, что переменная `anotherDay` в предыдущем примере соответствует 10 февраля, а не 10 января.

При желании можно указать в конструкторе `Date` до четырех дополнительных параметров — часы, минуты, секунды и миллисекунды:

```
// 1 февраля 2021 года, 9:30:
const anotherDay = new Date(2021, 1, 1, 9, 30);
```

Как вы узнаете далее в этой главе, у встроенного в JavaScript объекта `Date` есть ряд известных ограничений и несколько странностей. Если ваш код требует большого количества операций с датами, таких как вычисление диапазона дат, синтаксический анализ строк с датами или перенос дат между часовыми поясами, стоит использовать хорошо протестированную стороннюю библиотеку, такую как day.js (https://github.com/iamkun/dayjs) или date-fns (https://date-fns.org).

Читайте также

Если в вашей задаче есть даты, то вы, вероятно, захотите использовать их в вычислениях, как показано в рецепте 4.4. Также вас, возможно, заинтересует способ преобразования даты в форматированную строку (рецепт 4.6) или же преобразование строки с датой в соответствующий объект `Date` (рецепт 4.2).

4.2. Преобразование строки в дату

Задача

Есть информация о дате, представленная в виде строки. Мы хотим преобразовать ее в объект `Date`, чтобы потом манипулировать им в коде или вычислять даты.

Решение

Если повезет, то вам достанется строка с датой в стандартном формате метки времени ISO 8601 (вида 2021-12-17T03:24:00Z), которую можно передать в конструктор `Date` напрямую:

```
const eventDate = new Date('2021-12-17T03:24:00Z');
```

Буква `T` в этой строке отделяет дату от времени, а буква `Z` в конце строки показывает, что это универсальное время, заданное с использованием часового пояса UTC, что обеспечивает наилучшую совместимость на разных компьютерах.

Существуют и другие форматы, распознаваемые конструктором `Date` и методом `Date.parse()`. Но применять их настоятельно не рекомендуется, поскольку реализация этих форматов неодинаковая в разных браузерах. При рассмотрении

тестовых примеров может показаться, что они работают, однако в разных браузерах задействуются разные параметры, зависящие от региональных стандартов, такие как переход на летнее время, поэтому использование таких форматов вызовет проблемы.

Если дата представлена не в формате ISO 8601, необходимо исправить это вручную. Нужно извлечь из строки отдельные компоненты даты и передать их в конструктор `Date`. Для этого хорошо подойдут такие методы `String`, как `split()`, `slice()` и `indexOf()`, которые были подробно описаны в рецептах главы 2.

Например, если есть строка в формате `mm/dd/yyyy`, то можно использовать следующий код:

```
const stringDate = '12/30/2021';

// разбить строку на части по косым
const dateArray = stringDate.split('/');

// получить отдельные части даты
const year = dateArray[2];
const month = dateArray[0];
const day = dateArray[1];

// скорректировать номер месяца с учетом отсчета от 0
const eventDate = new Date(year, month-1, day);
```

Обсуждение

Конструктор объекта `Date` не особенно тщательно проверяет входные данные. Проверяйте их сами перед созданием объекта `Date`, поскольку он может посчитать приемлемыми значения, которые для вас неприемлемы. Например, он допускает перенос дней месяца (если передать конструктору день номер 40, то JavaScript просто перенесет эту дату на следующий месяц). Конструктор `Date` также принимает строки, которые могут по-разному преобразовываться в даты на разных компьютерах.

При попытке создать объект `Date` из строки, содержащей нецифровые данные, получим объект `Invalid Date`. Для проверки этого условия можно использовать функцию `isNaN()`:

```
const badDate = '12 bananas';

const convertedDate = new Date(badDate);

if (Number.isNaN(convertedDate)) {
    // Мы окажемся здесь, так как объект Date не создан
} else {
    // Если экземпляр объекта Data корректен, то мы окажемся здесь
}
```

Эта методика работает, поскольку по своей внутренней реализации объекты `Date` являются числами — этот факт подробно рассматривается в рецепте 4.4.

Читайте также

В рецепте 4.6 описывается обратная операция — преобразование объекта `Date` в строку.

4.3. Добавляем дни к дате

Задача

Вычислить дату, которая отстоит от другой даты на заданное число дней вперед или назад.

Решение

Получить номер сегодняшнего дня с помощью `Date.getDate()`, затем изменить его, применив `Date.setDate()`. Объект `Date` достаточно сообразителен, для того чтобы при необходимости перейти на следующий месяц или год:

```
const today = new Date();
const currentDay = today.getDate();

// Какой день наступит через три недели?
today.setDate(currentDay + 21);
console.log(`Three weeks from today is ${today}`);
```

Обсуждение

Метод `setDate()` не ограничен положительными числами. Если использовать отрицательное число, то можно получить дату в прошлом. Есть и другие методы типа `setXxx()` для изменения даты: например, `setMonth()` позволяет перейти на несколько месяцев вперед или назад, а `setHours()` — на несколько часов. Все эти методы, как и `setDate()`, позволяют перейти на следующий день, месяц или год. Например, если добавить к текущему времени 48 часов, то получим дату точно на двое суток позже.

Объект `Date` *изменяемый*, из-за чего его поведение выглядит весьма старомодным. Возможно, в будущих библиотеках JavaScript методы, подобные `setDate()`, станут возвращать новый объект `Date`. Но сейчас они изменяют *текущий* объект `Date`. Это происходит даже в том случае, если дата объявлена как константа. (Ключевое слово `const` не позволяет переменной ссылаться на другой объект `Date`, но не мешает изменить объект `Date`, на который указывает данная ссылка.) Для того чтобы гарантированно избежать потенциальных проблем, необходимо клонировать дату, прежде чем что-то с ней делать. Для этого воспользуйтесь методом `Date.getTime()`, чтобы получить количество миллисекунд, которое является внутренним представлением даты, и создайте на его основе новый объект:

```
const originalDate = new Date();

// Клонируем дату
const futureDate = new Date(originalDate.getTime());

// Изменяем клонированную дату
futureDate.setDate(originalDate.getDate()+21);
console.log(`Three weeks from ${originalDate} is ${futureDate}`);
```

Читайте также

В рецепте 4.5 показано, как вычислить временной интервал между двумя датами.

4.4. Сравнение дат и проверка двух дат на равенство

Задача

Убедиться, что два объекта `Date` описывают одну и ту же календарную дату, или определить, что одна дата является более ранней, чем другая.

Решение

Объекты `Date` можно сравнивать между собой точно так же, как и числа, с помощью операторов `<` и `>`:

```
const oldDay = new Date(1999, 10, 20);
const newerDay = new Date(2021, 1, 1);

if (newerDay > oldDay) {
    // Выражение истинно, так как newerDay идет после oldDay
}
```

Внутри объекта `Date` даты представлены в виде целых чисел. При использовании оператора `<` или `>` даты автоматически преобразуются в числа и сравниваются. При выполнении этого кода мы сравниваем значение `oldDay` в миллисекундах (943 074 000 000) со значением `newerDay` в миллисекундах (1 612 155 600 000).

Оператор равенства (`=`) работает по-другому. Он сравнивает не содержимое объектов, а ссылки на эти объекты. (Другими словами, два объекта `Date` равны между собой только в том случае, если мы сравниваем две переменные, которые ссылаются на один и тот же экземпляр.)

Для того чтобы убедиться, что два объекта `Date` описывают один и тот же момент времени, необходимо вручную преобразовать их в числа. Нагляднее всего это делается с помощью метода `Date.getTime()`, который возвращает число миллисекунд для даты:

```
const date1 = new Date(2021, 1, 1);
const date2 = new Date(2021, 1, 1);

// Результат равен false, так как это разные объекты
console.log(date1 === date2);

// Результат равен true, так как это одна и та же дата
console.log(date1.getTime() === date2.getTime());
```

Несмотря на название метода, getTime() возвращает не просто время, а число миллисекунд, точно соответствующее дате и времени объекта Date.

Обсуждение

По своему внутреннему представлению объект `Date` — это просто целое число. Точнее, это число миллисекунд, прошедших с 1 января 1970 года. Оно может быть положительным или отрицательным. Другими словами, с помощью объекта `Date` можно представлять даты, относящиеся как к глубокому прошлому (примерно с 271 821 года до н. э.), так и к далекому будущему (до 275 760 года). Это количество миллисекунд можно получить с помощью метода `Date.getTime()`.

Два объекта `Date` равны только в том случае, если они совпадают вплоть до миллисекунды. Если два объекта `Date` соответствуют одной и той же дате, но различаются по времени, то они не равны. Это может стать проблемой, так как не все учитывают, что в объекте `Date` хранится также информация о времени. Такая ошибка часто возникает при создании объекта `Date` для текущего дня (см. рецепт 4.1).

Чтобы избежать данной ошибки, нужно удалить информацию о времени с помощью метода `Date.setHours()`. Несмотря на свое название, метод `setHours()` принимает до четырех параметров, позволяя задавать часы, минуты, секунды и миллисекунды. Для того чтобы создать объект `Date`, содержащий только дату, все эти компоненты нужно установить равными 0:

```
const today = new Date();

// Создаем еще один объект Date с текущей датой
// День остался тот же, но время, возможно, изменилось на миллисекунду
const todayDifferent = new Date();

// Это может быть true или false в зависимости
// от факторов времени, которые вы не контролируете
console.log(today.getTime() === todayDifferent.getTime());

// Удаляем всю информацию о времени
todayDifferent.setHours(0,0,0,0);
today.setHours(0,0,0,0);

// Это всегда равно true, поскольку мы удалили время из обоих экземпляров
console.log(today.getTime() === todayDifferent.getTime());
```