Глава 2

Типы данных (часть 1)

2.1. Строки

> Строки полезны для хранения данных, которые можно представить в текстовой форме.
>
> Mozilla Developer Network

2.1.1. Создание строки

Чтобы создать строку, достаточно поместить текст между парой одинаковых кавычек.

```
var emptyString = '';
```

В данном случае была создана пустая строка. Чтобы убедиться в этом, можно проверить свойство length, которое всегда содержит длину строки:

```
emptyString.length; // 0
```

Для создания строк можно использовать одинарные кавычки:

```
var russianString = 'строка текста';
russianString.length; // 13
```

А также — двойные:

```
var russianString = "строка текста";
russianString.length; // 13
```

2.1.2. Экранирование. Escape-последовательности.

Некоторые символы строки должны быть заэкранированны, то есть записаны как escape-последовательность, которая начинается с обратного слеша:

- Если строка создана при помощи одинарных кавычек, все символы одинарных кавычек в ней должны быть экранированы:

```
var escapeCodesString = 'a\'b' // a'b
escapeCodesString.length; // 3
```

- Символ обратного слеша экранируется всегда, чтобы можно было отличить его от экранирования какого-то другого символа:

```
var escapeCodesString = 'a\\b' // a\b
escapeCodesString.length; // 3
```

- Специальные символы записываются при помощи соответствующих им escape-последовательностей. Например, перевод строки записывается при помощи `\n`, а символ табуляции — при помощи `\t`:

```
var escapeCodesString = 'a\n\tb' // a
                                 //     b
escapeCodesString.length; // 4
```

2.1.3. Unicode-символы

Строки представляют собой последовательности unicode символов. Важно отметить, что это могут быть любые символы на любом языке:

```
var utf8String = ' español English  русский    ';
utf8String.length; // 53
```

При этом длина строки высчитывается корректно, какие бы символы из каких языков не использовались бы.

2.1.4. Неизменяемость строк

В JavaScript возможно обращение к символу по индексу:

```
var russianString = 'кот';
russianString[1]; // 'о'
```

При этом перезаписать по индексу символ другим нельзя, так как в JavaScript строки неизменяемые. Причем следующий код выполнится корректно, то есть не вызовет ошибку:

```
russianString[1] = 'и';
russianString; // 'кот'
```

Однако строка в результате не поменяется.

2.1.5. Практический пример работы со строками

Как известно, в Twitter существует ограничение размера сообщений в 140 символов. Пусть требуется реализовать аналогичное ограничение, а в случае, если строка имеет длину больше заданной, обрезать ее.

Например, задана следующая строка:

```
var longString = 'Очевидно проверяется, что математический анализ \
существенно масштабирует интеграл по поверхности, что неудивительно. \
Первая производная, очевидно, позиционирует ортогональный определитель.'
```

Метод slice(start [, end]) строки возвращает ее часть от позиции start до позиции end (но не включая ее). Если параметр end не указан, извлечение идет до конца строки.

Так можно решить поставленную задачу:

```
var shortString = longString;
if (longString.length > 140) {
  shortString = longString.slice(0, 139) + '…';
}
```

```
shortString; // 'Очевидно проверяется, что математический анализ
// существенно масштабирует интеграл по поверхности, что неудивительно.
// Первая производная, оч…'
shortString.length; // 140
```

Пусть теперь требуется определить, содержит ли строка определенный хеш-тег.

```
var tweet = 'PWA. Что это такое? Третий доклад на WSD в Питере Сергея ' +
  'Густуна #wstdays';
```

Для поиска хеш-тега в строке можно воспользоваться методом indexOf.

Метод indexOf(searchValue[, fromIndex]) возвращает индекс **первого** вхождения указанной в качестве аргумента подстроки или -1, если подстрока отсутствует. Параметр fromIndex необязательный и задает местоположение внутри строки, откуда начинать поиск.

В частности, так можно найти индекс первого вхождения подстроки с хеш-тегом в строке:

```
tweet.indexOf('#wstdays'); // 65
tweet.indexOf('#fronttalks'); // -1
```

В данном случае получается, что в сообщении есть хеш-тег #wstdays и нет #fronttalks.

2.1.6. Основые операции со строками

Полный список операций со строками доступен по следующей ссылке.

splice извлекает часть строки и возвращает новую строку.

indexOf возвращает индекс первого вхождения указанного значения.

toLowerCase Приведение строки к нижнему регистру.

trim Удалить пробельные символы с начала и с конца строки.

startsWith Проверяет, начинается ли строка с заданной подстроки.

2.2. Массивы

> Массивы являются спископодобными объектами, чьи прототипы содержат методы для операций обхода и изменения массива. Ни размер JavaScript-массива, ни типы его элементов не являются фиксированными.
>
> Mozilla Developer Network

2.2.1. Создание массива

Для создания массива используются квадратные скобки.

```
var emptyArray = [];  // Пустой массив
```

Как и у строк, у массивов есть свойство length — количество элементов в массиве.

```
emptyArray.length; // 0
```

Можно сразу создать массив из необходимых элементов:

```
var arrayOfNumbers = [1, 2, 3, 4];    // Массив чисел
arrayOfNumbers.length; // 4
var arrayOfStrings = ['a', 'b', 'c']; // Массив строк
arrayOfStrings.length; // 3
```

2.2.2. Основные операции с массивами

- Обращение к элементу массива по индексу производится при помощи квадратных скобок.
- Итерирование по массиву производится с помощью цикла for:

```
for (var i = 0; i < tweets.length; i++) {
  var tweet = tweets[i];
  // Что-то делаем с конкретным твитом
}
```

- Добавление элементов (в конец массива) производится при помощи метода push:

```
var emptyArray = [];
// Добавляем элементы
emptyArray.push('a');
emptyArray.push('b');
emptyArray; // ['a', 'b']
emptyArray.length; // 2
```

- Метод pop «выталкивает» последний элемент массива и возвращает его.

```
// Удаляем последний элемент
emptyArray.pop(); // 'b'
emptyArray; // ['a']

emptyArray.length; // 1
```

  Длина массива при этом уменьшается на 1.

- Метод concat позволяет объединять массивы. Важно отметить, что этот метод не изменяет исходные массивы, а вместо этого возвращает новый массив.

```
var concatedArray =
 ↪  arrayOfNumbers.concat(arrayOfStrings);

concatedArray; // [1, 2, 3, 4, 'a', 'b', 'c']

arrayOfNumbers; // [1, 2, 3, 4]
arrayOfStrings; // ['a', 'b', 'c']
```

- Метод splice(start, deleteCount, ...items) изменяет содержимое массива, удаляя существующие элементы и/или добавляя новые. Подробнее о нем пойдет речь позже.

- Метод slice([begin[, end]]) возвращает новый массив, содержащий копию части исходного массива. Если оба параметра не указаны, возвращается копия массива:

2.2.3. Практический пример работы с массивом

На примере массива из 12 строк (твитов) будут рассмотрены основные операции, которые можно производить с массивами. Пусть дан следующий массив:

```
var tweets = [
  'Я и IoT, пятый доклад на WSD в Питере Вадима Макеев
   ↪  #wstdays',
  'Вёрстка писем. Развенчиваем мифы. Четвёртый доклад на
   ↪  WSD в Питере Артура Коха #wstdays',
  'PWA. Что это такое? Третий доклад на WSD в Питере Сергея
   ↪  Густуна #wstdays',
```

```
  'Pokémon GO на веб-технологиях, второй доклад на WSD в
   ↪ Питере Егора Коновалова #wstdays',
  'Ого сколько фронтендеров. #wstdays',
  '<head> - всему голова, первый доклад на WSD в Питере
   ↪ Романа Ганина #wstdays',
  'Доброе утро! WSD в Питере начинается через 30 минут:
   ↪ программа, трансляция и хештег #wstdays',
  'Наглядная таблица доступности возможностей веб-платформы
   ↪ Пола Айриша: Can I use + StatCounter, от CSS до JS',
  'Node.js, TC-39 и модули, Джеймс Снел о проблемах Node.js
   ↪ с асинхронными модулями ES и вариантах выхода из
   ↪ ситуации',
  'Всегда используйте <label>, перевод статьи Адама
   ↪ Сильвера в блоге Академии HTML',
  'JSX: антипаттерн или нет? Заметка Бориса Сердюка на
   ↪ Хабре',
  'Как прятать инлайновые SVG-иконки от читалок, Роджер
   ↪ Йохансен объясняет, зачем это нужно'
];

tweets.length; // 12
```

Поиск в массиве твитов Чтобы найти в массиве все твиты с тегом #wstdays можно воспользоваться циклом for и проверять, содержится ли хеш тег в очередной строке:

```
var result = [];

for (var i = 0; i < tweets.length; i++) {
  var tweet = tweets[i];

  if (tweet.indexOf('#wstdays') !== -1) {
    result.push(tweet);
  }
}
```

Добавление элементов в копию массива Чтобы создать копию массива, можно использовать метод slice без параметров:

```
var tweetsWithAdv = tweets.slice();
```

Для изменения созданного массива (добавление рекламных твитов) используется метод splice:

```
tweetsWithAdv.splice(4, 0, 'Покупайте наших слонов!');
tweetsWithAdv.splice(9, 0, 'И натяжные потолки тоже у
 ↪  нас отличные!');

tweets.length; // 12
tweetsWithAdv.length; // 14
```

Постраничная навигация по твитам Если в методе slice указаны параметры, в результате получается «срез» массива. Это позволяет организовать постраничную навигацию по твитам:

```
var tweetsByPage = tweetsWithAdv.slice(5, 10);
```

2.2.4. Полезные функции для работы с массивами

Полный список функций для работы с массивами можно найти по следующей ссылке.

push Добавляет один или более элементов в конец массива и возвращает новую длину массива.

pop Удаляет последний элемент из массива и возвращает его.

concat Возвращает новый массив, состоящий из данного массива, соединённого с другим массивом.

splice Добавляет и/или удаляет элементы из массива.

slice Извлекает диапазон значений и возвращает его в виде нового массива.

sort на месте сортирует элементы массива и возвращает отсортированный массив.

every проверяет, удовлетворяют ли все элементы массива условию, заданному в передаваемой функции.

some проверяет, удовлетворяет ли хоть какой-нибудь элемент массива условию, заданному в передаваемой функции.

shift удаляет первый элемент из массива и возвращает его значение.

unshift добавляет один или более элементов в начало массива и возвращает новую длину массива.

2.3. Объекты

> Список, состоящий из пар с именем свойства и связанного с ним значения, которое может быть произвольного типа.
>
> Mozilla Developer Network

2.3.1. Создание объекта

Объект можно создать с помощью пары фигурных скобок.

```
var emptyObject = {};
```

Так будет создан пустой объект. Также можно создать объект с требуемым набором свойств:

```
var tweet = {
  createdAt: 'Sat Oct 01 12:01:08 +0000 2016',
  id: 782188596690350100,
  text: 'Я и IoT, пятый доклад на WSD в Питере Вадима Макеева
   ↪  #wstdays',
  user: {
    id: 42081171,
    name: 'Веб-стандарты',
    screenName: 'webstandards_ru',
    followersCount: 6443
  },
  hashtags: ['wstdays']
};
```

2.3.2. Основные операции с объектами

Обращение к свойствам объекта Получение значения свойства и присваивание ему значения производится через точечную нотацию:

```
emptyObject.propertyName = 'foo'; // { propertyName: 'foo' }
emptyObject.propertyName; // 'foo'
```

Удаление свойства объекта производится при помощи оператора delete:

```
delete emptyObject.propertyName; // {}
```

Важно понимать, что удаляется один конкретный ключ, а не весь объект.

Обращение к свойствам вложенных объектов также производится при помощи точки:

```
tweet.id; // 782188596690350100
tweet.user.screenName; // 'webstandards_ru'
```

Обращение к свойствам объекта при помощи квадратных скобок:

```
tweet['i' + 'd']; // 782188596690350100
```

Такой способ обращения к значению часто используется, если значение ключа нужно сначала динамически вычислить.

Итерирование по ключам объекта Специальный метод Object.keys объекта Object возвращает для переданного в качестве единственного аргумента объекта массив его ключей. Это позволяет итерировать по ключам объекта:

```
var keys = Object.keys(tweet);
keys; // ['createdAt', 'id', 'text', 'user', 'hashtags']

for (var i = 0; i < keys.length; i++) {
  var key = keys[i];
  var value = tweet[key];
  // Что-то делаем с ключом и со значением
}
```

Проверка наличия свойства у объекта Для проверки наличия свойства у объекта используется метод hasOwnProperty:

```
tweet.hasOwnProperty('text'); // true
tweet.hasOwnProperty('nonExistantProperty'); // false
```

2.4. Функции

> Именованый блок кода, который позволяет переиспользовать существующий код.
> Может иметь входные параметры и возвращать значение.
> Является объектом высшего порядка.
>
> Mozilla Developer Network

2.4.1. Декларация функции. Возвращаемое значение

Пример декларации функции, которая возвращает значение:

```
function getFollowersCount() {
  return 6443;
}
```

Функции также могут не возвращать значения:

```
function noop() {
}
```

В этом случае движок JavaScript неявно добавит возвращение значения undefined в конец тела функции:

```
function noop() {
  return undefined;
}
```

Все функции в JavaScript, таким образом, возвращают значение: либо то, которое было возвращено явно, либо undefined.

2.4.2. Декларация функции. Передача параметров

Функция может также принимать аргументы и работать с ними.

```
function getAuthor(tweet) {
  return tweet.user.screen_name;
}
```

Данная функция принимает в качестве параметра объект и возвращает вложенное свойство этого объекта.

Аргументы могут передаваться по значению и по ссылке в зависимости от того, какого они типа:

- Если в качестве аргумента передан примитив, то есть строка (string), логическое значение (boolean) или число (number), аргумент передается по значению.

```
tweet.user.followersCount; // 6443

function incrementFollowersCount(count) {
  count++; // 6444
}

incrementFollowersCount(tweet.user.followersCount);
tweet.user.followersCount; // 6443
```

Как видно из последнего примера, изменения, произведенные с примитивом внутри тела функции, не меняют значение исходной переменной.

- Все сложные типы данных: массивы, объекты, функции и так далее передаются по ссылке.

```
tweet.user; // { id: 42081171, name: 'Веб-стандарты',
// screenName: 'webstandards_ru', followersCount: 6443 }

function incrementFollowersCount(user) {
  user.followersCount++;
}

incrementFollowersCount(tweet.user);
tweet.user; // { id: 42081171, name: 'Веб-стандарты',
// screenName: 'webstandards_ru', followersCount: 6444 }
```

В этом случае изменения, произведенные с аргументом функции, приводят к изменению исходной (переданной) переменной и видны вне тела функции.

2.5. Функций в качестве аргументов

Функции — объекты высшего порядка. Они могут быть переданы в другие функции в качестве аргумента, а так же могут иметь личные свойства, как и другие объекты.

Во всех дальнейших примерах будет использоваться следующий массив твитов:

```
var tweets = [
  { hashtags: ['wstdays'], likes: 16, text: 'Я и IoT, пятый…' },
  { hashtags: ['wstdays', 'mails'], likes: 33, text: 'Вёрстка писем…' },
  { hashtags: ['wstdays'], likes: 7, text: 'PWA. Что это…' },
  { hashtags: ['wstdays', 'pokemongo'], likes: 12, text: 'Pokémon GO на…'
   ↪  },
  { hashtags: ['wstdays'], likes: 15, text: 'Ого сколько фронт…' },
  { hashtags: ['wstdays', 'html'], likes: 22, text: '<head> - всему…' },
  { hashtags: ['wstdays'], likes: 8, text: 'Доброе утро! WSD…' },
  { likes: 9, text: 'Наглядная таблица доступности…' },
  { hashtags: ['nodejs'], likes: 7, text: 'Node.js, TC-39 и модули,…' },
  { hashtags: ['html'], likes: 28, text: 'Всегда используйте <label>…' },
  { likes: 18, text: 'JSX: антипаттерн или нет?…' },
  { hashtags: ['svg'], likes: 19, text: 'Как прятать инлайновые…' }
];
```

Следует отметить, что хеш-теги твитов содержатся в качестве массива в виде отдельного поля объекта твита.

2.5.1. Итерация по массиву при помощи метода forEach

Метод forEach применяет передаваемую в качестве его аргумента функцию к массиву. Функция, передаваемая методу forEach, сама принимает два аргумента: элемент на текущем шаге итерации и текущий индекс.

Например, задача фильтрации твитов, которые содержат определенный хеш-тег, решается следующим образом:

```
var result;
// Теперь в result лежат отфильтрованные твиты
tweets.forEach(filterWithWstdaysHashtag);
// Выбираем только твиты с хештегом #wstdays
function filterWithWstdaysHashtag(tweet, index) {
  var hashtags = tweet.hashtags;
  if (Array.isArray(hashtags) &&
   ↪  hashtags.indexOf('wstdays') !== -1) {
    result.push(tweet);
  }
}
```

В последнем примере, чтобы проверить, содержится ли хеш тег в твите, сначала с помощью метода Array.isArray проверяется, что свойство tweet.hashtags является массивом.

```
Array.isArray(hashtags)
```

Поскольку логические операции в JavaScript ленивые, оставшаяся часть условия не выполняется, если hashtags не является массивом. Затем (уже известно, что hashtags — это массив) вызывается метод массива indexOf, который возвращает индекс первого вхождения элемента в массив (или -1, если такого элемента не нашлось):

```
Array.isArray(hashtags) && hashtags.indexOf('wstdays')
```

2.5.2. Фильтрация (filter) и отображение (map)

Метод filter создаёт новый массив со всеми элементами исходного массива, для которых функция фильтрации возвращает true.

Метод map Создаёт новый массив с результатами вызова указанной функции на каждом элементе данного массива.

Метод join объединяет все элементы массива в строку. В качестве аргумента передается разделитель.

С помощью методов filter и map операции над массивами могут обрабатываться в виде цепочки.

Например, в следующем примере сначала выделяются твиты, содержащие определенный хеш-тег (это реализуется с помощью метода filter и функции filterWithWstdaysHashtag), а затем каждый твит преобразуется в соответствующую ему HTML-строку (для этого используется метод map и функция render):

```
// Теперь в result лежит HTML с деревом твитов
var result = '<dl>' +
 ↪  tweets.filter(filterWithWstdaysHashtag)
                              .map(render)
                              .join('\n') + '</dl>';

// Выбираем только твиты с хештегом #wstdays
function filterWithWstdaysHashtag(tweet, index) {
  var hashtags = tweet.hashtags;
  return Array.isArray(hashtags) &&
   ↪  hashtags.indexOf('wstdays') !== -1;
}
```

```
// Превращаем массив объектов твитов в HTML-строки
function render(tweet, index) {
return '<dt>' + tweet.text + '</dt>' +
  '<dd>' + tweet.user + '</dd>' +
  '<dd>' + tweet.hashtags.join(', ') + '</dd>';
}
```

После того, как получен массив из HTML-строк, он объединяется в одну строку при помощи метода join (в качестве разделителя выбран символ новой строки). Получившаяся строка «оборачивается» в `<dl>...</dl>`.

2.5.3. Метод reduce для последовательной обработки массива

Для работы с массивами есть еще один полезный метод.

Метод reduce Применяет функцию к аккумулятору и каждому значению массива (слева-направо), сводя его к одному значению.

Результатом работы метода reduce является так называемый аккумулятор. На каждом шаге цикла функция обратного вызова (которая передается в качестве первого аргумента) должна возвращать обновленный аккумулятор.

В следующем примере показано вычисление полного количества лайков у всех твитов в массиве. Аккумулятором в данном случае является число лайков.

```
var likesCount = tweets.reduce(getTotalLikes, 0)

function getTotalLikes(acc, item) {
  return acc + item.likes;
}

likesCount; // 194
```

Несколько более сложный пример использования метода reduce для получения статистики хеш-тегов:

```
var hashtagsStat = tweets.reduce(flattenHashtags, [])
    // ['wstdays', 'wstdays', 'mails', 'wstdays',
     ↪  'wstdays', 'pokemongo',
    // 'wstdays', 'wstdays', 'html', 'wstdays', 'nodejs',
     ↪  'html', 'svg']
```

```
                                .reduce(getHashtagsStats, {});

function flattenHashtags(acc, item) {
  return acc.concat(item.hashtags || []);
}

function getHashtagsStats(acc, item) {
  if (!acc.hasOwnProperty(item)) {
    acc[item] = 0;
  }

  acc[item]++;

  return acc;
}

hashtagsStats; // { html: 2, mails: 1, nodejs: 1,
 ↪  pokemongo: 1, svg: 1, wstdays: 7 }
```

В данном случае используется цепочка из двух reduce. В первом reduce строится плоский массив со всеми хеш-тегами всех твитов. Этот массив используется во втором reduce для получения объекта, ключами которого будут имена хеш-тегов, а значениями — статистика по хеш-тегам.