

Типы данных, часть II

Баженова Анна

В предыдущих сериях:

- Строки, массивы, объекты и функции
- Отличия примитивных типов данных от объектов
- Основные методы для работы со строками и массивами

Сегодня в программе:

- Объекты и их методы
- Операции сравнения
- Даты
- Регулярные выражения

Объекты

```
const tweet = {  
  id: 782188596690350100,  
  text: 'Я и IoT, доклад на WSD Вадима Макеева #wstdays',  
  user: {  
    id: 42081171,  
    name: 'Веб-стандарты',  
    screenName: 'webstandards_ru',  
    followersCount: 6443  
  },  
  hashtags: ['wstdays']  
};
```

Методы объекта

```
const tweet = {  
  likes: 16,  
  getLikes: function () {  
    return this.likes;  
  },  
  setLikes: function (value) {  
    this.likes = parseInt(value) || 0;  
  }  
};
```

Методы объекта

```
const tweet = {  
  likes: 16,  
  getLikes: function () {  
    return this.likes;  
  },  
  setLikes: function (value) {  
    this.likes = parseInt(value) || 0;  
    return this;  
  }  
};
```

Методы объекта

```
tweet.getLikes(); // 16
```

```
tweet.setLikes(17) // { ... }  
    .getLikes();   // 17
```

Методы объекта

```
const tweet = {  
  user: {  
    followersCount: 6443,  
    ...  
  },  
  getFollowersCount: function() {  
    return this.user.followersCount;  
  }  
};
```

```
tweet.getFollowersCount(); // 6443
```


Преобразование объектов

Логическое преобразование

```
// Любой объект в логическом контексте - true  
if ({} ) {  
    console.log('true!'); // вывод сработает  
}
```

Строковое преобразование

```
const panda = {  
  face: '🐼'  
};
```

```
panda.toString(); // '[object Object]'
```

Строковое преобразование

```
const panda = {  
  face: '🐼',  
  toString: function () {  
    return '&#x1F43C;';  
  }  
};  
  
panda.toString(); // &#x1F43C;
```

Строковое преобразование

```
const array = [1, 2, 3];
```

```
// Встроенный метод
```

```
console.log(array.toString()); // 1,2,3
```

Приведение к строке

```
const panda = {  
  face: '🐼',  
  toString: function () {  
    return '&#x1F43C;';  
  }  
};
```

```
panda.toString(); // '&#x1F43C;'
```

```
String(panda); // '&#x1F43C;'
```

```
'' + panda; // '&#x1F43C;'
```

==

```
'1' == 1; // true ???
```

==

```
'1' == 1; // true ???
```

===

```
'1' === 1; // false
```

Абстрактный Алгоритм Эквивалентного Сравнения

Операция сравнения двух сложных типов
вернет истину только в том случае, если
внутренние ссылки обоих объектов ссылаются
на один и тот же объект в памяти

```
{ } === { } // false
```

object1 === object2 -> true ?

```
const obj1 = { value: 1 };
```

```
const obj2 = obj1;
```

```
obj1 === obj2; // ?
```

object1 === object2 -> true ?

```
const obj1 = { value: 1 };
```

```
const obj2 = obj1;
```

```
obj1 === obj2; // true
```

```
obj1 === { value: 1 }; // false
```

```
obj2 === { value: 1 }; // false
```

object1 === object2 -> true ?

```
const obj1 = { value: { a: 1 } };
```

```
const obj2 = obj1;
```

```
obj1 === obj2; // true
```

```
obj1.value.a = 2;
```

```
console.log(obj1); // { value: { a: 2 } }
```

```
obj1 === obj2; // ?
```

object1 === object2 -> true ?

```
const obj1 = { value: { a: 1 } };
```

```
const obj2 = obj1;
```

```
obj1 === obj2; // true
```

```
obj1.value.a = 2;
```

```
console.log(obj1); // { value: { a: 2 } }
```

```
obj1 === obj2; // true
```

```
console.log(obj2); // { value: { a: 2 } }
```

Скрытые методы

```
const panda = {  
  toString: function() { return '🐼'; }  
};
```

```
Object.keys(panda); // ['toString']
```

```
const emptyObject = {};
```

```
Object.keys(emptyObject); // []
```

```
typeof panda.toString === 'function'; // true
```

```
typeof emptyObject.toString === 'function'; // true
```

Объявление свойств объекта

`Object.defineProperty(obj, prop, descriptor)`

`obj` – объект, в котором объявляется свойство

`prop` – имя свойства, которое нужно объявить или модифицировать

`descriptor` – объект, который описывает поведение свойства

Объявление свойств объекта

```
const panda = {};
```

```
panda.color = 'black';
```

```
Object.defineProperty(panda, 'color', {  
  value: 'black',  
  writable: true,  
  enumerable: true,  
  configurable: true  
});
```


Объявление свойств объекта. `writable`

```
Object.defineProperty(panda, 'color', {  
  value: 'black',  
  
  writable: false // запретить редактирование  
});
```

```
panda.color; // black  
panda.color = 'white';  
panda.color; // black
```

Объявление свойств объекта. configurable

```
Object.defineProperty(panda, 'color', {  
  value: 'black',  
  
  configurable: false // запретить удаление  
});
```

```
panda.color; // black  
panda.hasOwnProperty('color'); // true
```

```
delete panda.color; // false
```

```
panda.color; // black  
panda.hasOwnProperty('color'); // true
```

Объявление свойств объекта. enumerable

```
const panda = {  
  color: 'black',  
  
  toString: () => '&#x1f43c;',  
};
```

```
Object.keys(panda); // ['color', 'toString']
```

Объявление свойств объекта. enumerable

```
const panda = {  
  color: 'black'  
};
```

```
Object.defineProperty(panda, 'toString', {  
  value: () => '&#x1f43c;',  
  enumerable: false // исключить toString из списка итерации  
});
```

```
Object.keys(panda); // ['color']
```

Значения параметров `writable`, `enumerable` и `configurable` по умолчанию — `false`.

Заморозка

```
const panda = {  
  color: 'black'  
};
```

```
Object.isFrozen(panda); // false
```

```
Object.getOwnPropertyDescriptor(panda, 'color');  
// { value: 'black',  
//   writable: true,  
//   enumerable: true,  
//   configurable: true }
```

Заморозка. freeze

```
Object.freeze(panda);

Object.isFrozen(panda); // true

Object.getOwnPropertyDescriptor(panda, 'color');
// { value: 'black',
//   writable: false,
//   enumerable: true,
//   configurable: false }

panda.color = 'white';

panda.color; // 'black'

delete panda.color; // false
```

Заморозка. freeze

```
Object.getOwnPropertyDescriptor(panda, 'color');  
// { value: 'black',  
//   writable: false,  
//   enumerable: true,  
//   configurable: false }
```

```
Object.defineProperty(panda, 'color', {  
  value: 'white'  
});  
// TypeError: Cannot redefine property: color
```


Заморозка. freeze

```
const panda = {  
  classification: {  
    kingdom: 'Animalia',  
    class: 'Mammalia'  
  }  
};
```

```
Object.freeze(panda);  
panda.classification.kingdom = 'Fungi';
```

```
console.log(panda); // {  
  //   classification: {  
  //     kingdom: 'Fungi',  
  //     class: 'Mammalia'  
  //   }  
  // }
```

Заморозка. `seal`

```
const panda = {  
  color: 'black'  
};
```

```
Object.seal(panda);
```

```
Object.getOwnPropertyDescriptor(panda, 'color');  
// { value: 'black',  
//   writable: true,  
//   enumerable: true,  
//   configurable: false }
```

```
panda.color = 'white';  
panda.color; // white
```

Объект Даты

```
// Создает объект с текущей датой в системном часовом поясе  
new Date(); // Mon Oct 08 2018 00:07:53 GMT+0500 (YEKT)
```

```
const date = '2018-10-08T14:07:27.362Z';  
// Пытаемся сконвертировать строку в дату  
new Date(date); // Mon Oct 08 2018 19:07:27 GMT+0500 (YEKT)
```

```
// Создаем дату из UNIX Timestamp  
new Date(1540300025000); // Tue Oct 23 2018 18:07:05 GMT+0500 (YEKT)
```

```
// Создаем дату из набора параметров  
new Date(2018, 9, 23, 18, 7, 5); // Tue Oct 23 2018 18:07:05 GMT+0500 (YEKT)
```

```
// Получаем UNIX Timestamp из даты  
(new Date(2018, 9, 23, 18, 7, 5)).valueOf(); // 1540300025000
```

```
// Получаем текущее значение UNIX Timestamp  
Date.now(); // 1539007957389
```

Объект Даты

```
const date = new Date(2018, 9, 23, 17, 50, 05);
```

```
date.getFullYear(); // 2018
```

```
date.getMonth(); // 9
```

```
date.getDate(); // 23
```

```
date.getHours(); // 17
```

```
date.getMinutes(); // 50
```

```
date.getSeconds(); // 5
```

```
date.setFullYear(2020);
```

```
date.setMonth(1);
```

```
date.setDate(25);
```

```
date.setHours(4);
```

```
date.setMinutes(7);
```

```
date.setSeconds(0);
```

Date

Relax 🐼

Регулярные выражения

- Имеют стандартный PCRE-синтаксис
- Реализованы отдельным объектом RegExp и интегрированы в методы строк

СИНТАКСИС

```
const regexp = new RegExp(pattern, flags);
```

```
const regexp = /pattern/; // без флагов
```

```
const regexp = /pattern/gi; // с флагами
```

Методы строк

```
tweet.text; // 'Node.js, и модули, Джеймс о проблемах Node.js  
            // #nodejs #modules'  
  
// Проверяем, содержится ли регулярное выражение в строке  
tweet.text.search(/js/); // 5  
tweet.text.search(/abc/); // -1  
  
// Находит первое совпадение в строке  
const result = tweet.text.match(/js/);  
result[0]; // js  
result.index; // 5  
result.input; // 'Node.js, и модули...' (вся поисковая строка)  
  
tweet.text.match(/abc/); // null
```


Флаги

g — глобальное сопоставление

i — игнорирование регистра при сопоставлении

m — многострочный режим

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
// Находит все совпадения в строке
```

```
tweet.text.match(/node/ig); [ 'Node', 'Node', 'node' ]
```

replace

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
tweet.text.replace(/node/gi, 'NODE');
```

replace

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
tweet.text.replace(/node/gi, 'NODE');
```

```
// 'NODE.js, и модули, Джеймс о проблемах NODE.js #NODEjs #modules'
```

replace

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
tweet.text.replace(/node/gi, 'NODE');
```

```
// 'NODE.js, и модули, Джеймс о проблемах NODE.js #NODEjs #modules'
```

```
function replacer(match, offset, str) {  
    return match + offset;  
};
```

```
tweet.text.replace(/node/gi, replacer);
```

```
// 'Node0.js, и модули, Джеймс о проблемах Node38.js #node47js #modules'
```

Классы символов

`\d` — любая цифра

`\s` — "пустой" символ (пробел, табуляция, перевод строки, ...)

`\w` — символ "слова" (цифра, буква, _)

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
const result = tweet.text.match(/\sN\w\we/);
```

```
result[0]; // ' Node'
```

```
result.index; // 37
```

Список классов

Классы СИМВОЛОВ

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
tweet.text.match(/\w\w\w\w.\w\w/gi);
```

```
// [ 'Node.js', 'Node.js', 'modules' ]
```

Классы СИМВОЛОВ

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
tweet.text.match(/\w\w\w\w\w.\w\w/gi);
```

```
// [ 'Node.js', 'Node.js', 'modules' ]
```

Точка – любой символ

```
tweet.text.match(/\w\w\w\w\w\.\w\w/gi);
```

```
// [ 'Node.js', 'Node.js' ]
```

Квантификаторы

$\{n\}$, $\{n,m\}$ — количество повторений

$+$ — один или более, $\{1,\}$

$?$ — ноль или один, $\{0,1\}$

$*$ — ноль или более, $\{0,\}$

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
tweet.text.match(/\w+\.\w+/g); // [ 'Node.js', 'Node.js' ]
```

```
tweet.text.match(/\w+\.\?\w+/g);
```

```
// [ 'Node.js', 'Node.js', 'nodejs', 'modules' ]
```

```
tweet.text.match(/\w{5,7}/g); // [ 'nodejs', 'modules' ]
```


Наборы символов

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

```
tweet.text.match(/#[nodemilsubj]+/g); // [ '#nodejs', '#modules' ]
```

```
tweet.text.match(/#[a-z]+/g); // [ '#nodejs', '#modules' ]
```

```
\d - [0-9]
```

```
\s - [\t\n\v\f\r ]
```

```
\w - [a-zA-Z0-9_]
```

```
tweet.text.match(/[#\w]+/g);
```

```
// [ 'Node', 'js', 'Node', 'js', '#nodejs', '#modules' ]
```

Наборы символов

```
tweet.text;
```

```
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'
```

^ - все, кроме перечисленных

```
tweet.text.match(/^[^s]+/g);
```

```
// [ 'Node.js,', 'и', 'модули,',
```

```
//   'Джеймс', 'о', 'проблемах',
```

```
//   'Node.js', '#nodejs', '#modules' ]
```

Скобочные группы

```
tweet.text;  
// 'Node.js, и модули, Джеймс о  
// проблемах Node.js #nodejs #modules'  
  
tweet.text.match(/\s[#\w]+/g);  
// [ ' Node', ' #nodejs', ' #modules' ]  
  
// Квантификатор применится ко всей скобке  
tweet.text.match(/\s([#\w]+)/g);  
// [ ' Node', ' #nodejs #modules' ]
```

Скобочные группы

```
tweet.text;  
// 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules'  
  
tweet.text.match(/\s[#\w]+/);  
// [ ' Node',  
// index: 37,  
// input: 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules' ]  
  
// Выделяем часть совпадения в отдельный элемент массива  
tweet.text.match(/\s(#[\w]+)/);  
// [ ' Node',  
// 'Node',  
// index: 37,  
// input: 'Node.js, и модули, Джеймс о проблемах Node.js #nodejs #modules' ]
```

Полезные ссылки

- [Understanding Date and Time](#)
- [Регулярные выражения для новичков](#)
- [Руководство по регулярным выражениям](#)
- [Сервис для проверки регулярных выражений](#)

Вопросы?