Современная Front-End разработка

Объектно-ориентированное программирование в JavaScript

Павел Бекетов
pavel.beketov@simbirsoft.com

Работа с DOM

Есть такой фрагмент html:

```
<html>
    <head>
        <title>Title</title>
    </head>

    <body>
        <button>Click!</button>
    </body>
</html>
```

Из JavaScript кода мы имеем полный доступ к нему с помощью глобального объекта document:

```
> document.documentElement
<- <html>
     ▼<head>
        <title>Title</title>
      </head>
     ▼<body>
        <button>Click!</button>
      </body>
    </html>
> document.body
<- ▼<body>
      <button>Click!</button>
    </body>
```

```
document.body.children
[   <button>Click!</button>]
```

Работа с DOM

Из JS мы можем менять аттрибуты элементов:

```
document.body.children[0].className = "btn btn-default";
document.body.children[0];
  <button class="btn btn-default">Click!</button>
```

```
document.body.children[0].style.width = "300px"
"300px"
```

```
document.body.children[0];
  <button class="btn btn-default" style="width: 300px;">Click!</button>
```

Вешаем события на кнопку:

```
document.body.children[0].onclick=function(){console.log(this)};
function (){console.log(this)}
```

```
  <button class="btn btn-default" style="width: 300px;">Click!</button>
```

Важно: нельзя повесить несколько событий onclick на элемент, они будут переопределяться.

DOM события

На элемент лучше вешать события с помощью метода addEventListener(type, eventListener), порядок вызова слушателей определяется их объявлением:

```
> document.body.children[0]
      .addEventListener("click", function() {
          console.log("event1");
      });
<- undefined
   event1
> document.body.children[0]
      .addEventListener("click", function() {
          console.log("event2");
      });
<- undefined
   event1
   event2
```

Удаляются события с помощью метода removeEventListener(type, eventListener), где eventListener – ранее привязанная функция:

```
> function listener() {
      console.log("listener");
  }

  document.body.children[0]
      .addEventListener("click", listener);
<- undefined
   event1
   event2
   listener
> document.body.children[0]
      .removeEventListener("click", listener);
<- undefined
   event1
   event2
```

DOM события

В функцию eventListener передается аргумент event, содержащий информацию по событию, а также ряд методов:

```
> function listener(event) {
      console.log(event.target);
      event.stopPropagation();
  }

  document.body.children[0]
      .addEventListener("click", listener);

  document.body.children[0]
      .addEventListener("click", listener);
<· undefined
```

```
<button>Click!</button>
```

Фунцкия event.stopPropagation используется, чтобы остановить вызов последующих слушателей

Поиск элементов в DOM

Для поиска элементов в DOM дереве можно пользоваться функциями поиска:

- document.getElementById(id) – возвращает элемент с заданным id или null, если элемент не найден
- document.getElementsByClassName(class) – возвращает массив элементов с заданным именем класса, можно использовать несколько имен:
- document.getElementsByClassName('red test') – вернет все элементы с классами “red” и “test”
- document.querySelector(selectors) – возвращает первый попавшийся элемент, удовлетворяющий селектору, либо null и document.querySelectorAll(selectors) – возвращает массив, удовлетворяющих элементов:
- document.querySelectorAll(“.red.test”) - результат аналогичен предыдущему примеру с className

Изменение и добавление элементов в DOM

Создание новых элементов:

```html
<ul id="list">
    <li>Первый элемент</li>
</ul>
```

```js
var list = document.getElementById('list');

// новый элемент
var li = document.createElement('li');
li.innerHTML = 'Новый элемент списка';

// добавление в конец
list.appendChild(li);
```

Клонирование элементов:

```
var clone = document.querySelector("button").cloneNode(true);
undefined
clone;
  <button>Click!</button>
```

NOTES:

- Клонированные элементы не добавляются в DOM, чтобы это сделать, используется appendChild()
- Свойства добавленные через JS-код(например onclick) не копируются

AJAX

Объект XMLHttpRequest (или, как его кратко называют, «XHR») дает возможность из JavaScript делать HTTP-запросы к серверу без перезагрузки страницы.

```
// 1. Создаём новый объект XMLHttpRequest
var xhr = new XMLHttpRequest();

// 2. Конфигурируем его: GET-запрос на URL 'posts.json'
xhr.open('GET', 'posts.json', false);

// 3. Отсылаем запрос
xhr.send();

// 4. Если код ответа сервера не 200, то это ошибка
if (xhr.status != 200) {
  // обработать ошибку
  console.log( xhr.status + ': ' + xhr.statusText );
} else {
  // вывести результат
  console.log( xhr.responseText ); // responseText -- текст ответа.
}
```

AJAX

Предыдущий вариант использования делает синхронный запрос, весь JavaScript "подвиснет", пока запрос не завершится, поэтому почти всегда используется асинхронная версия XHR:

```
var xhr = new XMLHttpRequest();

xhr.open('GET', 'posts.json', true);

xhr.send(); // (1)

xhr.onreadystatechange = function() { // (3)
    if (xhr.readyState != 4) return;

    if (xhr.status != 200) {
        console.log(xhr.status + ': ' + xhr.statusText);
    } else {
        console.log(xhr.responseText);
    }
};

console.log('Загрузка'); // (2)
```

AJAX

Состояния запроса(readyState):

- 0; // начальное состояние
- 1; // вызван open
- 2; // получены заголовки
- 3; // загружается тело (получен очередной пакет данных)
- 4; // запрос завершён

Работа с заголовками:

- setRequestHeader(name, value) – задает заголовок запроса с именем name
- getResponseHeader(name) – возвращает заголовок ответа по имени
- getAllResponseHeaders() - возвращает все заголовки ответа

Наследование в JavaScript

На уровне языка реализовано наследование на прототипах. С помощью некоторых трюков можно сделать наследование на классах, объявить приватные свойства объекта и многое другое.

Создание объекта. Функция-конструктор

Любая функция может создать объект:

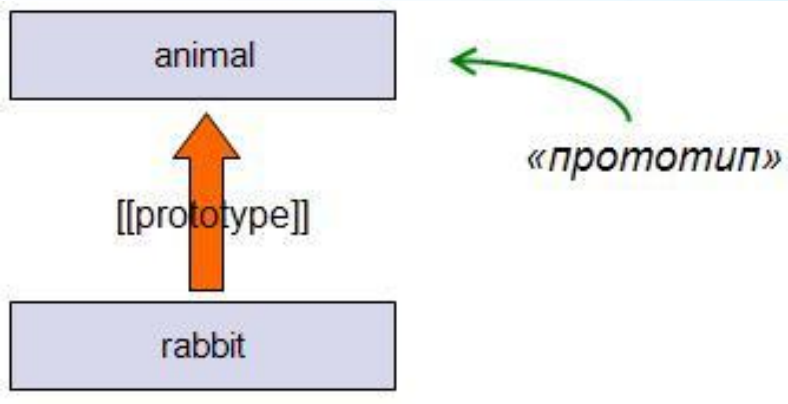
```
function Animal(name) {
    this.name = name;
    this.canWalk = true;
}

var animal = new Animal("Medved");
```

Класс объекта определяется функцией, которая его создала. Для проверки принадлежности классу есть оператор instanceof:

```
alert(animal instanceof Animal)  // => true
```

Наследование через прототип

Реализуется наследование через неявную(внутреннюю) ссылку одного объекта на другой, который называется его прототипом и в спецификации обозначается [[prototype]]. Это свойство обычно скрыто от программиста. Также существует свойство с похожим названием prototype (без квадратных скобок) - оно вспомогательное и указывает, откуда брать прототип при создании объекта.
Когда вы ставите функции Animal свойство Animal.prototype = XXX - вы этим декларируете: "все новые объекты класса Animal будут иметь прототип XXX".



Ссылка [[prototype]] работает так:

1) Любое запрошенное свойство ищется сначала в rabbit
2) Если свойство там не найдено, то оно ищется в rabbit.[[prototype]], т.е в animal

Наследование через прототип

```
function Rabbit(name) {
    this.name = name;
}

// все объекты, созданные Rabbit
// будут иметь прототип (наследовать) animal
Rabbit.prototype = animal;

var big = new Rabbit('Big');
var small = new Rabbit('Small');

console.log(big.name);  // Big
console.log(small.name); // Small

console.log(big.canWalk); // true
```

Расширяем прототип функции-конструктора:

```
Animal.prototype.walk = function(n) {
    this.distance = n;
    console.log(this.distance);
};
```

Наследование на классах. Функция extend

В качестве решения для полноценного наследования, используется функция extend, которая является почти стандартом:

```
function extend(Child, Parent) {
    var F = function() { };
    F.prototype = Parent.prototype;
    Child.prototype = new F();
    Child.prototype.constructor = Child;
    Child.superclass = Parent.prototype;
}
```

Вызов родительских методов

С помощью функции extend легко получить доступ к родительским методам:

```
function Rabbit(args)  {
    //КАКОЙ ТО КОД
    Rabbit.superclass.constructor.apply(this, arguments);
    //ЕЩЕ КАКОЙ ТО КОД
}

//использование метода родителя
Rabbit.superclass.method.apply(this, args);
```

Вызов родительских методов

Обязательно следить за контекстом, в данном случае использование this вместо Rabbit приведет к ошибке:

```
function foo() {}
foo.prototype.identify = function() {
    return "I'm a foo";
};

function bar() {}
extend(bar, foo);
bar.prototype.identify = function() {
    return "I'm a bar and " +
        this.constructor.superclass.identify.apply(this, arguments);
};

function zot() {}
extend(zot, bar);
zot.prototype.identify = function() {
    return "I'm a zot and " +
        this.constructor.superclass.identify.apply(this, arguments);
};

var f = new foo();
console.log(f.identify()); // "I'm a foo"

var b = new bar();
console.log(b.identify()); // "I'm a bar and I'm a foo"

var z = new zot();
console.log(z.identify()); // stack overflow
```

Использование прототипов для расширения функционала

Задача: нужно логировать время старта и время окончания AJAX запросов по URL.

```
var basicXMLHttpRequest = {};
basicXMLHttpRequest.tempOpen = XMLHttpRequest.prototype.open;
basicXMLHttpRequest.tempSend = XMLHttpRequest.prototype.send;

var network = {};//используем словарь, чтобы хранить URLы

//Interceptor for start of HTTP requests
XMLHttpRequest.prototype.open = function(method, url) {
    basicXMLHttpRequest.tempOpen.apply(this, arguments);
    this.url = url;
    network[url] = {};
    network[url].start = new Date();
};

//Interceptor for send of HTTP requests
XMLHttpRequest.prototype.send = function(a, b) {
    basicXMLHttpRequest.tempSend.apply(this, arguments);

    //Adding a listener of returning of response
    this.addEventListener('readystatechange', function() {
        if (this.readyState !== 4) return;

        var url = this.url;
        network[url].time = new Date() - network[url].start;
    }, false);
};
```

Домашнее задание

Примечание: для работы с файлами нужно запустить браузер со специальным параметром например для chromium:
chromium --allow-file-access-from-files

Задание

Создаем два файла: posts.json и users.json в корне проекта и заполняем их тестовыми данными, модели:

```
"post": {
    "_id": int,
    "content": string,
    "date": date,
    "author": int(_id $user),
    "profile": int(_id $user)
}

"user": {
    "_id": int,
    "email": String,
    "firstName": String,
    "lastName": String,
    "thirdName": String,
    "mobile": String,
    "facebook" : String,
    "twitter": String,
    "city": String
}
```

Домашнее задание

Необходимо обернуть XMLHttpRequest в сервис httpService, который экпортирует метод get(url, callback). Пример использования:

```
var posts = null;

function getPosts() {
    httpService.get('posts.json', function(data) {
        posts = data;
    });
}
```

Реализовать модуль userService. Экпортирует фукнцию getUserById, которая возвращает id
Создать функцию конструктор Post, интерфейс:

```
// создает элемент используя шаблон
// заполняет поля автора, даты и сообщения поста
// сохраняет шаблон внутри объекта Post
// НЕ добавляет элементы в DOM
// template - div элемент, который содержит три поля с классами author, date, content:
// к примеру:
// <div>
//    <p class="author"></p>
//    <p class="date"></p>
//    <p class="content"></p>
// </div>
render: function(template) {}
//Добавляет себя в потомки parent, добавляется после element
//Возвращает истину в случае успеха
//Если шаблон еще не задан, то ничего не делает и возвращает ложь
insertAfter: function(parent, element) {}

//принимает модель post задает поля нового объекта как в модели
function Post(post) {};
```

Домашнее задание

Используя созданные классы, вывести содержимое файла posts.json на странице Ленты, т.е. на каждую запись в файле выводить отдельный пост.

Вопросы?