

— JAVASCRIPT —

JavaScript модули. Форматы. Загрузчики. RequireJS.

МОДУЛИ

Паттерн «Модуль»

Модуль

это переиспользуемая часть кода, содержащая в себе детали реализации и предоставляющая открытое API, что позволяет легко загрузить её и использовать в другом коде.

В идеале, модули JavaScript позволяют:

абстрагировать код,

передавая функциональные возможности сторонним библиотекам, так что нам не придётся разбираться во всех сложностях их реализации;

инкапсулировать код,

скрывая его внутри модуля, если не хотим, чтобы его изменяли;

переиспользовать код,

избавляясь от необходимости писать одно и то же снова и снова;

управлять зависимостями,

легко изменяя зависимости без необходимости переписывать наш код.

МОДУЛИ

ПАТТЕРН

МГНОВЕННО ВЫЗЫВАЕМАЯ ФУНКЦИЯ
(Immediately Invoked Function Expressions)

Немедленно вызываемая функция (IIFE)

анонимная функция, которая вызывается сразу после объявления. Функция окружена скобками.

```
(function(){  
    // ...  
})();
```

МОДУЛИ

Immediately Invoked Function Expressions

```
// Объявление функции
function(){
  console.log('test');
}
```

Помещение функции в скобки делает это функциональным выражением:

```
// Функциональное выражение
(function(){
  console.log('test');
})

// returns function test(){ console.log('test') }
```

```
// Мгновенное объявление функции
function(){
  console.log('test');
}()
// Uncaught SyntaxError: Unexpected token )
```

Функциональное выражение возвращает нам функцию, поэтому тут же можно к ней обратиться:

```
// Мгновенно вызываемая функция
(function(){
  console.log('test');
})();
// пишет 'test' в консоли и возвращает undefined
```

МОДУЛИ

Паттерн выявления модуля (Revealing Module)

Паттерн выявления модуля схож с IIFE,
но здесь присваивается возвращённое значение переменной:

```
// Объявление модуля как глобальной переменной
var singleton = function(){

    // Внутренняя логика
    function sayHello(){
        console.log('Hello');
    }

    // Внешнее API
    return {
        sayHello: sayHello
    }
}()
```

```
// Access module functionality
singleton.sayHello();
// Hello
```

```
// Объявление модуля как глобальной переменной
var Module = function(){

    // Внутренняя логика
    function sayHello(){
        console.log('Hello');
    }

    // Внешнее API
    return {
        sayHello: sayHello
    }
}
```

```
var module = new Module();

module.sayHello();
// Hello
```

— Форматы модулей —

Асинхронное определение модуля

AMD - Asynchronous Module Definition или AMD

Формат AMD используется в браузерах и применяется для определения модулей функцию define

```
//Вызов функции define с массивом зависимостей
//и фабричной функцией
define(['dep1', 'dep2'], function (dep1, dep2) {

    //Определение модуля с помощью возвращаемого значения
    return function () {};

});
```

Универсальное определение модуля (UMD)

Формат UMD может быть использован как в браузере, так и в Node.js.

Формат модулей ES6

В ES6 JavaScript уже поддерживает нативный формат модулей. Он использует токен **export** для экспорта публичного API модуля и токен **import** для импорта частей, которые модуль экспортирует

CommonJS

Формат CommonJS применяется в Node.js и использует для определения зависимостей и модулей require и module.exports

```
var dep1 = require('./dep1');
var dep2 = require('./dep2');

module.exports = function(){
    // ...
}
```

System.registerA

Формат System.register был разработан для поддержки синтаксиса модулей ES6 в ES5

Загрузчики модулей

Загрузчик модуля интерпретирует и загружает модуль, написанный в определённом формате, в время выполнения (в браузере).
Распространённые — **RequireJS** и SystemJS.

Загрузчик модуля запускается в среде исполнения:

- вы загружаете загрузчик модуля в браузере;
- вы сообщаете загрузчику, какой главный файл приложения запустить;
- модуль скачивает и интерпретирует главный файл приложения;
- загрузчик модулей скачивает файлы по мере необходимости.

RequireJS

- *www/*
 - *index.html*
 - *js/*
 - *app/*
 - *app.js*
 - *lib/*
 - *jquery.js*
 - *canvas.js*
 - *init.js*
 - *require.js*

```
requirejs.config({  
    // базовый url, относительно которого будет идти поиск модулей  
    baseUrl: 'js/app',  
    // используем для маппинга/соответствия путей и модулей  
    // для библиотек, либо глобальных компонентов  
    // если путь НЕ начинается с "/", то поиск в baseUrl  
    // если с "/" - относительно файла подключения require.js  
    // может содержать url на сторонний ресурс ('http:// ...')  
    paths: {  
        jquery: '/js/lib/jquery-3.2.1.min',  
        app: 'app',  
        radio: 'radio'  
    }  
});  
  
requirejs(['app'], function (app) {  
    app.init();  
});
```

```
<script data-main="js/init.js" src="js/require.js"></script>
```


RequireJS

Если модуль не имеет зависимостей, и это просто набор пар имя / значение

```
define({  
  color: "black",  
  size: "unysize"  
});
```

Если модуль имеет зависимости, первым аргументом должен быть массив имен зависимостей, а второй аргумент должен быть функцией определения.

```
define(["models/TaskModel", "views/TaskView"],  
  function(TaskModel, TaskView) {  
    //return an object to define the module.  
    return {  
      color: "blue",  
      size: "large",  
      init: function() {  
        this.model = new TaskModel();  
        this.view = new TaskView(this.model);  
      }  
    }  
  }  
);
```

Если модуль не имеет зависимостей, но ему нужно использовать функцию для выполнения некоторой работы по настройке

```
define(function () {  
  // выполнить некоторую работу по настройке  
  
  return {  
    color: "black",  
    size: "unysize"  
  }  
});
```

RequireJS

модуль как функция

```
define(["my/cart", "my/inventory"],
  function(cart, inventory) {
    //return a function
    //It gets or sets the window title.
    return function(title) {
      return title ? (window.title = title) :
        inventory.storeName + ' ' + cart.name;
    }
  }
);
```

```
define(function () {
  var TaskView = function (model) {
    this.model = model;
  };

  TaskView.prototype = {

    render: function (index) {
      var wrapper = document.createElement('div');
      wrapper.classList.add('field');
      return wrapper;
    }
  };
  return TaskView;
});
```

```
define(function () {

  var Radio = function () {
    this.topics = {};
  };

  Radio.prototype = {
    on: function (topic, listener) {
      // ...
    },

    trigger: function (topic, data) {
      // ...
    }
  };

  var radio = new Radio();
  return radio;
});
```

— JAVASCRIPT —

JavaScript модули. Форматы. Загрузчики. RequireJS.