Расулев Тимур

Frontend. LeroyMerlin.

timur.rasulev@leroymerlin.ru

Модульный JS

история развития модульного javascript

Когда нет модульной архитектуры

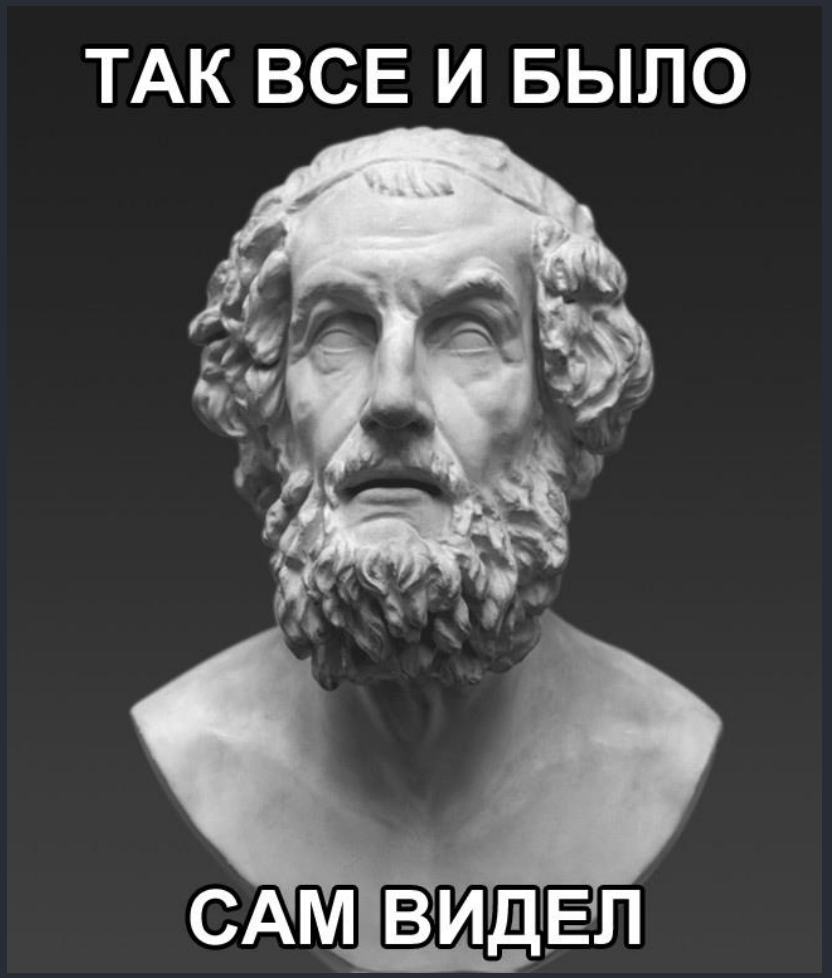
Зачем модули в javascript?

- Организация кода и архитектуры приложения
- Помогает добиться инкапсуляции
- Избежать конфликта имен функций и переменных
- Пользователи работают с публичным API

Когда есть модульная архитектура

История развития модульности в JS

Принято считать, что впервые паттерн 'Модуль' разработал Ричард Конфорд в 2003 году. В дальнейшем этот паттерн был популяризирован Дугласом Крокфордом в своих лекциях и курсах.



Замыкание + IIFE

Immediately-invoking function expressions (IIFE)

```
var module = (function() {
    var private = { obj: "Hi!" };

    return {
        hi: function() {
            console.log(private.obj);
        }
    };
})();

module.hi();
```

```
void function() {
    window.mathlib = window.mathlib || {};
    window.mathlib.sum = sum;

    function sum(...values) {
        return values.reduce((a, b) => a + b, 0)
    }
}();

mathlib.sum(1, 2, 3);
```

OOP

```
import Component from '../lib/component.js';
import store from '../store/index.js';

export default class Status extends Component {
    constructor() {
        super({
            store,
            element: document.querySelector('.js-status')
        });
    }

    render() {
        let self = this;
        let suffix = store.state.items.length !== 1 ? 's' : '';

        self.element.innerHTML = `${store.state.items.length} item${suffix}`;
    }
}
```

```
const statusInstance = new Status();

statusInstance.render();
```

RequireJS(AMD)

```
//modules/main.js
require(["moduleA"],function(moduleA){
    moduleA.init();
});
```

```
//modules/moduleA.js
define(["moduleB","moduleC"], function(moduleB, moduleC) {
    console.log("Function : moduleA");
    return {
        init: function() {
            var count = moduleB.getCount();
            if(count > 0){
                moduleC.saveCount();
                return true;
            }
            return false;
        }
    }
});
```

```
//modules/moduleB.js
define(function() {
    console.log("Function : moduleB.getCount");
    return {
        getCount: function() {
            var count = 100;
            return count;
        }
    }
});
```

```
//modules/moduleC.js
define(function() {
    console.log("Function : moduleC.saveCount");
    return {
        saveCount: function() {
            return true;
        }
    }
});
```

AngularJS(Dependency Injection)

```
module.factory('calculator', function(mathlib) {
    // …
});
```

```
module.factory('calculator', ['mathlib', function(mathlib) {
    // …
}]);
```

CommonJS(Node.JS)

```
var privateObj = {
    hi: 'Hi!'
};

var sayHi = function (greet) {
    return privateObj[greet];
};

module.exports.sayHello = sayHi;
```

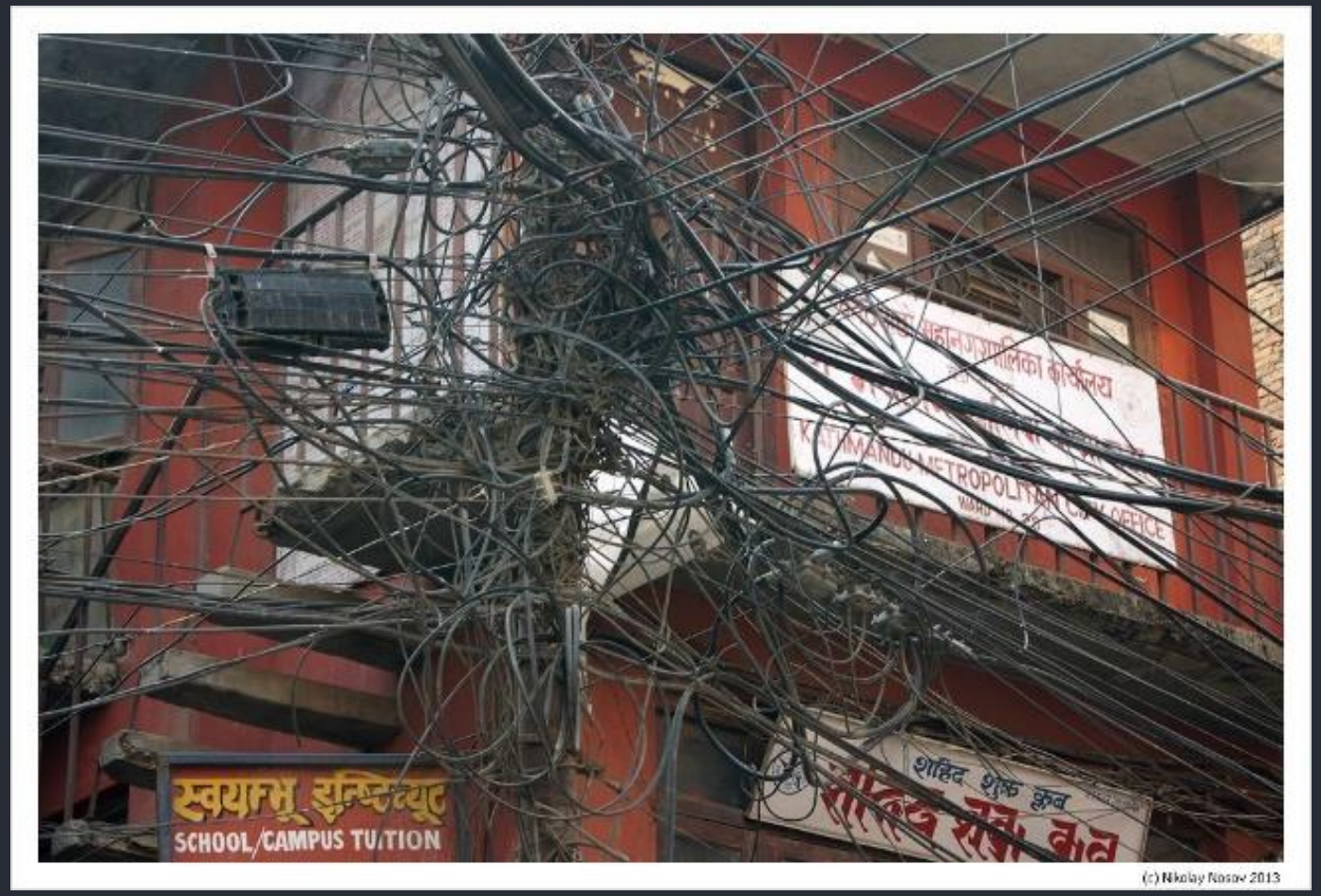
```
var sayHi = require('./lib/greeting').sayHello;
var greeting = sayHi('hi');

console.log(greeting);
```

ES6, import, Babel и Webpack

```
import mathlib from './mathlib';
import('./mathlib').then(mathlib => {
    // …
});
```

Сложность



- Принцип единственной ответственности
- API
- Управление состоянием, объект Store
- Publish/Subscribe паттерн

Принцип единственной ответственности

- Одна конкретная задача
- Что является единственной ответственностью компонента?
- Какую одну минимальную задачу он должен решить?

API

- Публичный интерфейс разрабатываемого модуля важен
- Хороший публичный интерфейс может прикрыть слабую реализацию
- Предоставлять доступ к базовым возможностям компонента
- Быть гибким к новым возможностям компонента

Управление состоянием, объект Store

- сложно организовать синхронизацию изменения состояния между компонентами
- сложно отслеживать и контролировать изменения
- Flux, Redux, Vuex, Mobx
- Состояние всего приложения хранится в дереве объектов внутри единственного хранилища

Publish/Subscribe (named events, mediator)

- Избавляемся от слишком тесной взаимосвязанности
- Более высокий уровень абстракции
- Независимые компоненты