Яндекс

Что интересного нам готовит W3C

Сергей Константинов

Web Standards Days, 7 декабря 2013

Обо мне

– В Яндексе с 2008 года
– Руковожу разработкой API Яндекс.Карт
– С июля 2013 года являюсь участником Технической архитектурной группы W3C www.w3.org/2001/tag/
– @blogovodoved

We are the world.

В последние годы появилось много разных стандартов

Но ядро ~~JavaScript~~ ECMAScript практически не затронуто

В 2014 всё изменится*

* Если успеем, конечно

Тема дня

– Promises
– Modules
– Service Workers
– @@create

Promises

Promises

Promises (aka Futures) в вебе уже давно
На всякий случай, напомню:

```
someOperation.then(
    successCallback,
    errorCalback
);
```

Promises

Существует несколько стандартов Promises:
A, A+, B, KISS, C, D, …

И множество реалиазаций:
Q, jQuery Deferred, Vow, …

(см. http://wiki.commonjs.org/wiki/Promises)

Promises

Черновик стандарта DOM Level 4 включает в себя Promises как примитив языка:

http://dom.spec.whatwg.org/#promises

https://github.com/domenic/promises-unwrapping/blob/master/README.md

Promise with resolver

В отличие от большинства существующих реализаций, в DOM Promises функции подтверждения и отклонения промиса отделены от самого промиса и сам промис неизменяем.

Promise with resolver

Vow:

```
function asyncOperation () {
   var p = vow.promise();
   …
   return p;
}

…
asyncOperation().fulfill('some value');
```

Promise with resolver

Теперь будет так:

```
function asyncOperation () {
    var p = new Promise(
                    function (resolve, reject) {
                        …
                        resolve('some value');
                        …
                    });
    return p;
}
```

Promises

Используйте промисы для:

– возврата статуса асинхронной операции

```
file.open('path').then(
    function (handle) {
        // do something valueable
    },
    function (error) {
        // handle error
    }
);
```

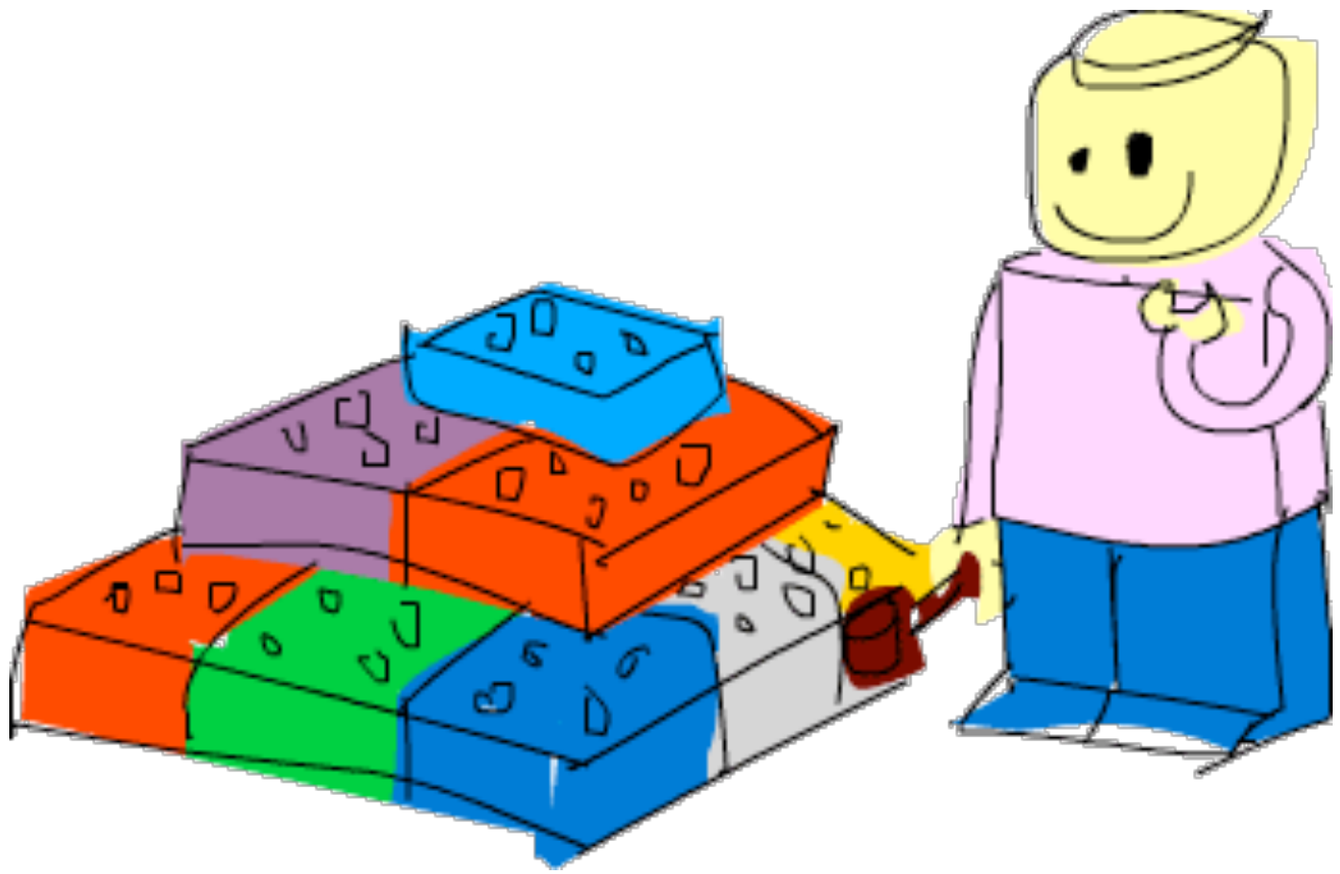
Promises

Не используйте промисы для:

– передачи больших объёмов данных,

– как замену событиям.

Модули



Модули

Долгое время в JavaScript/ECMAScript не было никакой модульности.

Это привело к бардаку в global и десяткам имплементаций (псевдо)модульных систем.

http://wiki.ecmascript.org/doku.php?id=harmony:modules

Модули

Теперь вы можете объявлять модули:

```
module 'math' {
    export function sum(x, y) {
        return x + y;
    }
    export var pi = 3.141593;
}
```

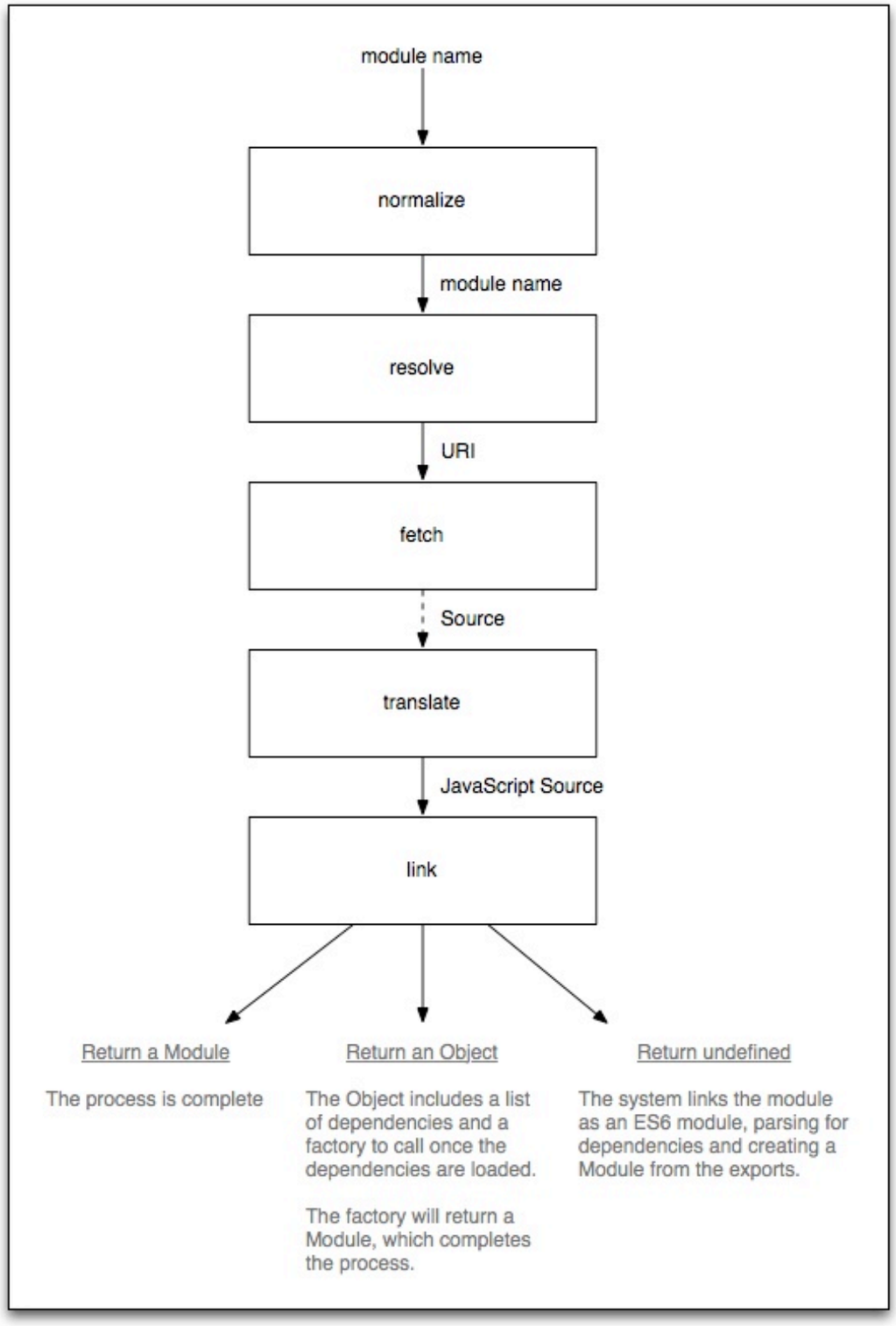
Модули

… импортировать модули:

```
import foo from "foo";
import {sum, pi} from 'math';
import "fs" as fs;
import { draw as drawShape } from 'shape';
import { draw as drawGun } from 'cowboy';
module YUI from 'http://developer.yahoo.com/
modules/yui3.js';
```

Конвейер компиляции модулей

– normalize
– resolve
– fetch
– translate
– link



Этап resolve

Вы можете задавать свои правила резолва имён модулей:

```
System.ondemand({
    "https://yandex.st/jquery/2.4/jquery.module.js": "jquery",
    "backbone.js": ["backbone/events", "backbone/model"]
});
```

Этап translate

Вы можете совершать операции над кодом модуля до его исполнения.

Например, проверить jshint-ом:

```
import { JSHINT } from "jshint";
import { options } from "app/jshintrc";
System.translate = function (source, options) {
  var errors = JSHINT(source, options),
      messages = [options.actualAddress];
  if (errors) {
    throw new SyntaxError(errors);
  }
  return source;
};
```

Этап translate

Вы можете совершать операции над кодом модуля до его исполнения.

Например, транслировать CoffeeScript

```
System.translate = function (source, options) {
  if (!options.path.match(/\.coffee$/)) {
    return;
  }
  return CoffeeScript.translate(source);
};
```

Этап link

Можно контролировать, что попадёт в модуль в качестве global, и можно выбрать, что экспортировать из модуля.

https://gist.github.com/wycats/51c96e3adcdb3a68cbc3

Конвейер компиляции модулей

В итоге, мы можем:

– писать и использовать модули;

– написать хук для текущих реализаций модулей (requirejs, amd);

– написать вообще свой кастомный Loader и/или отнаследоваться от системного.

Конвейер компиляции модулей

Мы **не** можем из коробки:

– грузить код on demand ☹

Service Workers (nee Navigation Controllers)

Service Workers – что это?

– Это возможность для определённого набора URL домена установить "контроллер", который будет обрабатывать все запросы с открытых браузером страниц, матчащихся на этот набор URL-ов

https://github.com/slightlyoff/ServiceWorker/blob/master/explainer.md

Service Worker

Инсталлируем:

```
navigator.registerServiceWorker(
   "/*", "/v1/ctrl.js"
).then(function (serviceWorker) {
   console.log("success!");
   // Чтобы использовать Service Worker сразу,
   // можно выполнить window.location.reload()
});
```

Service Worker – зачем?

Service Worker умеет:

– перехватывать запросы за сетевыми ресурсами и отвечать на них

```
this.addEventListener("fetch", callback);
```

– кэшировать ресурсы

```
var resources = new Cache(
        base + "/assets/v1/base.css"
    );
```

– редиректить запросы

– версионироваться.

Service Worker – зачем?

Service Worker имеет доступ:

– ко всем открытым в браузере страницам, на которые матчится

```
this.windows.forEach(function (w) {
   …
});
```

– ко всем разедляемым ресурсам: WebStorage, IndexedDB, cookies, …

Service Worker – зачем?

Service Worker научится (?):

– получать push уведомления;

– открывать новые страницы.

Магический символ

@@create

Наследование в JS

Как выглядит наследование в JS?

```
function ChildClass () {
    ParentClass.call(this);
}
ChildClass.prototype = new ParentClass();
// На самом деле не так, обычно делают магию
// чтобы не вызывать конструктор ParentClass
```

Наследование в JS

Что происходит при вызове new ChildClass()?

– создаётся пустой объект,

– записывается скрытая ссылка на ChildClass.prototype, если он есть,

– передаётся в качестве **this** в функцию-конструктор.

Наследование в JS

Но этот метод не работает, если в **this** должен быть не Object, а другой примитив языка, например, Array, Date, RegExp, …

```
// Так не работает
function ChildArray () {
    Array.call(this);
}
ChildArray.prototype = new Array();
```

Наследование в JS

Разрешение наследоваться от встроенных объектов приводит к неразрешимым противоречиям.

```
function ChildArray () {
    Array.call(this);
    Date.call(this);
    this.length = 100500;
    // И что мы получим в итоге? o_O
}
```

Наследование в JS

К тому же, в большинстве случаев внутренняя логика встроенных типов реализованы на C++ из соображений производительности.

Наследование в JS

Решение: отделить создание объекта (allocation) от его инициализации (initialization).

Функция-конструктор **инициализирует** созданный объект, а вот само создание спрятано за другой функцией.

Наследование в JS

Функция-аллокатор прячется за специальным символом @@create ("эт-эт-криэйт", ага).

```
// именно так, без кавычек
// возвращает инстанцию объекта
// для передачи в инициализирующий конструктор
Function[@@create]()
```

@@create – не единственный специальный символ, есть ещё несколько полезных штук.

http://people.mozilla.org/~jorendorff/es6-draft.html#sec-ecmascript-language-types-symbol-type

Наследование в JS

new ChildClass() теперь эквивалентен вот такой конструкции:

```
ChildClass.call(
    typeof ChildClass[@@create] != 'undefined' ?
        ChildClass[@@create]() :
        // Это некоторое упрощение ;)
        ObjectCreate(ChildClass.prototype)
);
```

http://people.mozilla.org/~jorendorff/es6-draft.html#sec-construct-argumentslist

Наследование в JS

Обещают поддержку наследования от:

– стандартных типов Array, Date, RegExp;

– TypedArray;

– новых ES6-типов Set, WeakSet, Map, WeakMap;

– и даже HTMLElement!

Наследование в JS

Вот так:

```
function ChildClass () {
    Array.call(this);
}
ChildClass[@@create] = Array[@@create];
ChildClass.prototype = new Array();
```

Главный вопрос:

– Когда?

Обещают весной

Обещают весной
(год не уточняют)

Обещают весной
(год не уточняют)
(но хотя бы при нашей жизни)

Яндекс

Сергей Константинов

Руководитель группы
разработки API Карт
W3C TAG Member

twirl@yandex-team.ru

Спасибо!