Vylaďte si aplikaci!
Návrhové vzory v JS

Marek Fojtl <marek.fojtl@firma.seznam.cz>, programátor senior UI

SEZNAM.CZ

Co po dnešku budu vědět?

- co je to **Immediately-invoked function expression** a jaké je její použití
- jak vytvářet **objekty**, realizovat **dědičnost** a vytvořit **singleton**
- co je to **ES6** a jak řeší absenci klíčového slova **class** v JS
- jak řešit implementační rozdíly mezi prohlížeči za pomocí **polyfillů**
- jak se vyhnout **callback hell**
- jak zkrotit **asynchronní kód**
- **MVC** a jeho zástupci

Immediately-invoked function expression

Nejčastější zápis

```
<script type="text/javascript">
    var elm = document.querySelector("#id-elementu");
    var calendar = new Calendar(elm);

    console.log(typeof window.elm); // object
    console.log(typeof window.calendar); // object
</script>
```

- **globální prostor DOMu** se zanáší různými atributy
- můžete **přepsat atribut** někoho jiného, **který se** později **bude někde používat**
- **obtížnější ladění chyb**

Nyní trochu lépe

```
<script type="text/javascript">
    var initCalendar = function() {
        var elm = document.querySelector("#id-elementu");
        var calendar = new Calendar(elm);
    };

    initCalendar();

    console.log(typeof window.elm); // undefined
    console.log(typeof window.calendar); // undefined
    console.log(typeof window.initCalendar); // function
</script>
```

A nejlépe

```
<script type="text/javascript">
    (function(win) {
        var elm = document.querySelector("#id-elementu");
        var calendar = new Calendar(elm);
    })(window); // Immediately-invoked function expression

    console.log(typeof window.elm); // undefined
    console.log(typeof window.calendar); // undefined
</script>
```

Objekty
Konstrukční funkce
Dědičnost
Singleton

Sestavit strom, zplodit objekt?

- objekty lze vytvořit třemi způsoby:

```
var obj1 = new Object();
var obj2 = {};
var obj3 = Object.create(Object.prototype);

console.log(obj1 instanceof Object); // true
console.log(obj2 instanceof Object); // true
console.log(obj3 instanceof Object); // true
```

- v JS je vše **objekt**
- všechny objekty jsou potomky objektu **Object**
- všechny objekty dědí základní metody a atributy z **Object.prototype**

Class? To neznám! … Zatím.

```
var Person = function(name) {
    this.name = name;
};

Person.prototype.getName = function() {
    return name;
};

var person = new Person("John Doe");
console.log(person.getName()); // "John Doe"
```

- v JS se pro vytváření tříd (vzorů) používají konstrukční funkce a prototypy
- konstrukční funkci voláme s operátorem **new**

Zajímavost - simulace new

```
// nasledujici kod popisuje temer to stejne, co se deje, kdyz se pouzije operator new

function SimulateNew(func, params) {
    // vytvori se prazdny objekt
    var o = {};
    // skryta vlastnost __proto__ zacne ukazovat na prototype funkce
    o.__proto__ = func.prototype;
    // zavolame konstrukcni funkci v kontextu nove vytvoreneho objektu
    func.apply(o, params);
    // vratime vytvoreny objekt
    return o;
}

// misto var person = new Person("John Doe"); pak lze napsat jako:
var person = SimulateNew(Person, ["John Doe"]);
console.log(person instanceof Person); // true
```

Dědičnost

```
var Person = function(name) {
    this.name = name;
};

Person.prototype.getName = function() {
    return name;
};

var Man = function(name) {
     // zavolame kons. funkci rodice v kontextu podedene tridy
    Person.apply(this, arguments);
};

// naklonujeme metody a atributy z Person do Man
Man.prototype = Object.create(Person.prototype);

var man = new Man("Marek");
console.log(man.getName()); // Marek
console.log(man instanceof Person && man instanceof Man); // true
```

Singleton

```
(function(win) {
    var instance = null;

    var App = function() {
        throw new Error("Can not call singleton with new!");
    };

    App.getInstance = function() {
        if (instance) { return instance; }
        instance = Object.create(App.prototype);
        instance.init();
        return instance;
    };

    App.prototype.init = function() {
        this.inited = true;
    };

    win.App = App;
})(window);
```

ECMAScript 6

Co je to ECMAScript?

- dále jen ES
- **skriptovací jazyk normovaný neziskovou organizací ECMA International**
- autoři interpreterů jazyka Javascript z něj vychází a při implementaci se řídí jeho specifikací
- v moderních prohlížečích je momentálně implementován převážně **ES5.1**
- **připravuje se ES6**, která zejména přidává **podporu zápisu pro psaní tříd a modulů**

ES6 Classes

```
class Person {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
}

class Man extends Person {
    constructor() {
        super();
    }
}
```

Lze to použít již nyní?

- **ANO!**
- podobně jako na CoffeScript, TypeScript, LiveScript, … existuje i na **ES6 transpiler** pro **překlad do ES5 kódu**, kterému dnešní prohlížeče rozumí
- https://github.com/google/traceur-compiler

Polyfilly

Prohlížeč mě nezná! ☹

- **polyfill** v JS **je doplnění standardizované funkcionality** (nebo API), kterou prohlížeč sám nemá
- na rozdíl od abstrakce (viz např. $.bind("…")) **pak může uživatelův kód používat standardní API** a nestarat se o podporu prohlížečů či proprietární rozhraní
- implementace polyfillu se řídí striktně specifikací, neměly by se do něj přidávat vlastní funkcionalita nebo vylepšení

Příklad

```
if (!Array.prototype.forEach) {
    Array.prototype.forEach = function(cb, _this) {
        var len = this.length;
        for (var i=0;i<len;i++) {
            if (i in this) {
                cb.call(_this, this[i], i, this);
            }
        }
    }
}

if (!Array.forEach) {
    Array.forEach = function(obj, cb, _this) {
        Array.prototype.forEach.call(obj, cb, _this);
    }
}
```

Callback hell

Seznamte se, Callback Hell

```
var Tooltip = function(elm) {
    this._dom            = {};
    this._dom.target     = elm;
    this._dom.container  = null;
    this._init();
};

Tooltip.prototype._init = function() {
    var that = this;
    that._dom.container  = document.createElement("span");
    that._dom.container.classList.add("tooltip");
    that._dom.target.addEventListener("mouseover", function() {
        setTimeout(function() {
            that._dom.container.classList.add("visible");
        }, 1000);
    });
};
```

Seznamte se, Function.bind

```
var Tooltip = function(elm) {
    . . .
    this._init();
};

Tooltip.prototype._init = function() {
    . . .
    this._dom.target.addEventListener("mouseover", this._delayedShow.bind(this));
};

Tooltip.prototype._delayedShow = function() {
    setTimeout(this._show.bind(this), 1000);
};

Tooltip.prototype._show = function() {
    this._dom.container.classList.add("visible");
};
```

Ještě lépe

```
var Tooltip = function(elm) {
    . . .
    this._delayedShow    = this._delayedShow.bind(this);
    this._show           = this._show.bind(this);
    this._init();
};

Tooltip.prototype._init = function() {
    . . .
    this._dom.target.addEventListener("mouseover", this._delayedShow);
};

Tooltip.prototype._delayedShow = function() {
    setTimeout(this._show, 1000);
};

Tooltip.prototype._show = function() {
    this._dom.container.classList.add("visible");
};
```

Promise

Slibem neurazíš!

- **problém asynchronního kódu spočívá v tom**, **že výsledek funkce není v době volání ještě znám**, ale interpreter na něj nečeká a provádí další příkazy
- řešení existuje v podobě callbacku, který funkci předáme a ta ho zavolá, až bude výsledek známý
- co když ale v zpožděné části funkce **nastane výjimka**?
- jak zřetězit **více asynchronních volání**?
- co když je funkce **asynchronní podmíněně**?

- nejvyšší čas sáhnout po návrhovém vzoru **Promise**

Až budu, dám Ti vědět!

```
var Loader = function() {
    this._xhr = null;
};

Loader.prototype.load = function() {
    return new Promise(function(resolve, reject) {
        this._xhr = new XMLHttpRequest();
        this._xhr.open(...);
        this._xhr.onreadystatechange = this._onload.bind(this, resolve, reject);
        this._xhr.send();
    }.bind(this));
};

Loader.prototype._onload = function(resolve, reject) {
    if (this._xhr.readyState != 4) { return; }
    resolve(this._xhr.responseText);
};

var loader = new Loader();
loader.load().then(console.log);
```

A to je jako všechno?

- **nikoliv!**
- metoda then vrací další **Promise**
- lze tedy psát Loader.load().then(…).then(…) a vytvořit tak řetěz asynchronních volání, tzv. **synchronizace asynchronního kódu**
- **nová Promise**, vrácená then, **bude resolvnuta hodnotou vrácenou callbackem**, **kterou může být další Promise**

- ??? o čem to ten člověk mluví?

- pochopení **Promise** jde zahranici našeho povídání, **mrkněte na**
- **http://www.html5rocks.com/en/tutorials/es6/promises/**

MVC frameworky

Kam čert nemůže, nastrčí MVC

- **M**odel - **V**iew - **C**ontroller
- tento návrhový vzor pochází již z roku 1979 (autor Trygve Reenskaug)
- má svoje místo na desktopech, mobilních aplikacích (iOS), a serverech, tak proč ho nepoužít i v prohlížeči
- Nejsilnější motivace:

  vytvářet elementy pomocí document.createElement je a připínat je do DOMu **je opruz**

  psaní low-level logiky aplikace je opruz

MVC přichází

- a s nimi i šablony a samostatné šablonovací systémy (Handlebars)
- píše se rok 2008 a na svět přichází doposud méně známý JavascriptMVC
- konec roku 2009 vše mění, vzniká nejznámnější MVC framework s názvem **AngularJS**, po té přichází **Knockout**, následován **Backbone** a v roce 2011 doplní fantastickou čtyřku i **Ember**
- výše uvedené frameworky získávají silné povědomí u vývojářů
- mají společné paradigma (MVC), ale jsou dosti členité (komplexnost, modularita, práce se šablonami)
- vyvolávají *flame*, jestli MVC v prohlížeči **ano - ne**

AngularJS

- právem nazývaný **super-heroic framework**
- šablonovací jazyk založený na speciálních **HTML atributech a značkách s prefixem ng** (ng-app, ng-controller, ng-submit, …)
- má **obousměrný data-binding**
- závádí pojmy jako **scope, controller, direktiva, filtr a service**
- je komplexní - nabízí router, práci s requesty, promise, dekorátory, css animace a vše, co je potřebné k vývoji single page aplikace
- existuje do něj nepřeberné množství modulů včetně lokalizačních knihoven
- používáme na http://zbozi.cz a http://sreality.cz
- další služby se chystají

Tak trochu jiný ReactJS

- vyvinuli vývojáři **Facebooku**
- první veřejně dostupná verze z roku **2013**
- v **MVC** zastupuje **V**
- závádí **JSX** - syntaxe podobná XML
- lze psát i bez **JSX**, ale **je to nepohodlné**
- je potřeba transpiler - **JSXTransformer** - pro překlad do JS
- pracuje s tzv. **virtuálním DOMem**
- **funguje i na serveru** (node.js)
- vytváří prostor pro tvorbu **izomorfních aplikací** - ale to je zas jiná pohádka :-)

A to je vše přátelé!

Dotazy?

...najdu tam, co neznám!

Marek Fojtl, programátor senior UI