JavaScript 类型简介

JavaScript (/lib/tag/JavaScript) 2016-04-15 18:44:35 发布

对于 JavaScript 类型，可以简单地概括为：相对于强类型语言来说，它是弱（松散）类型的语言；有基本类型和引用类型，他们是区别是一个有固定空间存在于栈内存中，一个没有固定空间保存在堆内存中并且在栈内存中保存了一个指向实现位置的指针。市面上很多书都有不小的篇幅在讲。这篇文章会讲几个方面，这些方面可能会需要你对 JavaScript 已经有了一些简单的了解，特别是 JavaScript 的类型。如果还不一解，可以随手拿起一本关于 JavaScript 的书翻翻，再来看本文。

一、基本类型与引用类型

- 基本类型：Undefined / Null / Boolean / Number / String / Symbol
- 引用类型：Object / Array / Function / Date / RegExp ...
类型只是枚举了一部分，有些特殊的像 WeakMap 大家都可以在 MDN JavaScript (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide) 上找到。

二、JavaScript 类型的判断

在 JavaScript 有两个 operator 可以用以判断类型。他们是 typeof 和 instanceof ，不过圈子很小，它们混的可不是那么好，是出了名的不靠谱。少数情况也是对的，很多情况下是不靠谱的。看看就知道了：

```
// 靠谱的时候：
typeof 'sofish' // string
new String('sofish') instanceof String // true

// 不靠谱的时候：
typeof [] // object
typeof null // object
'sofish' instanceof String // false
```

呃～ 可能很多初学的 JavaScript 程序员会因此爆粗口。还大部分人在需要用 JS 的时候已经有了 jQuery 等这样的库，他们都做了封装，让你可以方便地检测类型。当然，事实上要检测也不麻烦，因为那句「在 JavaScript 中，一切都是对象」，当然像很多文档中说到的， undefined 其实和 NaN , Infinity 都只是一个全局属性。你大概知道就可以了。但「对象」可以帮到我们：

```
/* 检测对象类型
 * @param: obj {JavaScript Object}
 * @param: type {String} 以大写开头的 JS 类型名
 * @return: {Boolean}
 */
function is(obj, type)  {
  return Object.prototype.toString.call(obj).slice(8, -1) === type;
}
```

这样的话，我们就可以利用 is 这个函数来帮我们搞定类型判断了，并且这个简单的函数有很好的兼容性，可以用到你的项目中去。情况如：

```
is('sofish', 'String') // true
is(null, 'Null') // true
is(new Set(), 'Set') // true
```

三、JavaScript 类型的转换

在 JavaScript 中，变量（属性）的类型是可以改变的。最常看到的是 String 与 Number 之间的转换。为何 1 + '2' 是 12 呢？这里面有必要理解一下 + 号 operator，它是一个数学运算符，同时也是 JavaScript 中的字符串连字符。所以新手会经常会看到一个有趣的现象，当使用 + 号的时候有时计算出来的不是想要的，而用 - 号却总能得到「正确」的答案。

```
1 + '2' // '12'
1 + (+'2') // 3
1- '2' // -1
```

这里面其实就是因为 + 的双重角色导致的。在上面的代码中，可以注意到第二条表达式在 String 前面运用了一个 + 号，强制把它的类转换为 Number 。而对于 JavaScript 的类型转换理解，大多数情况下，只要理解 + 具有双重角色就可以了。其他的可以理解类，类似都是可以用赋值/重载来修改的，甚至包括 Error：

```
var err = new Error();
console.log(err instanceof Error); // true

err = 'sofish';
console.log(err); // 'sofish'
```

四、JavaScript 引用类型

这一点是本文的一个难点。相对于基本类型，引用类型可以为其添加属性和方法；引用类似的值是一个引用，把一个引用类型的值赋给一个变量，他们所指向的是同一存储在堆内存中的值。变量（属性）可以重载，但复制会是一件很有趣的事情，后面我们会详细来说。

1. 添加属性和方法

下面的代码我们将会看到，假设我们对一个基本类似赋值，它并不会报错，但在获取的时候却是失效的：

```
var arr = [1,2,3];
arr.hello = 'world';
console.log(arr.hello); // 'world'

var str = 'sofish';
str.hello = 'world';
console.log(str.hello); // undefined
```

2. 引用类型值的操作

由于引用类型存储在栈内存中的是一个引用，那么当我们指向的同一个原始的值，对值的操作将会影响所有引用；这里有一个例是，重新赋值（并非对值的直接操作）会重新创建一个对象，并不会改变原始值。比如：

```
var arr = [1,2,3], sofish = arr;
sofish.push('hello world');
console.log(arr); // [1, 2, 3, 'hello world']

// 非相同类型
sofish = ['not a fish']; // 当 sofish 类似改变时，不会改变原始值
console.log(arr);// [1, 2, 3, 'hello world']
```

3. 引用类型值的复制

对原始值的操作会影响所有引用，而这不一定是我们想要的，有时候我们需要复制一个全新的对象，操作的时候不影响其他引用。而一般情况也，像 Date / Function / RegExp ... 都很少有具体的操作，主要是像 Array 和 Object 会有添加项、属性等操作。所以我们主要需要理解的是如何复制 Array 和 Object 对象。

3.1 数组的复制

在 Array 对象中，存在 slice 方法返回一个截取的数组，在 ES5 中 map / filter 等也返回一个新的数组，那么我们可能利用这个方法来进行复制。

```
var arr = [1, 2, 3];
var sofish = arr.slice();

// 对新的数组进行操作并不会影响到原始数组
sofish.push('hello world');
console.log(arr); // [1, 2, 3]
```

3.2 对象的复制

在 Array 的复制中我们使用的是 slice 方法，实际上对于 Array 和 Object 中都可以利用 for ... in 循环来进行遍历并赋值来进行复制。

```
var obj = { name: 'sofish' }, sofish = {}, p;
for (p in obj) sofish[p] = obj[p];

// 对新的对象操作并不会影响原始值
sofish.say = function() {};
console.log(obj); // { name: 'sofish' }
```

3.3 Shadow / Deep Copy

像上面的操作，就是我们常说的浅拷贝（Shallow Copy）。不过在 Array 和 Object 都可以有多层（维），像这样的拷贝只考虑到最上面一层的值，在可能存在的值中的 Array 和 Object 都还是指向了原始对象。比如：

```
var arr = [1, { bio: 'not a fish' } ], sofish = [], p;
for(p in arr) {
  sofish[p] = arr[p];
}

// 对 `sofish` 中包含的对象 `cat` 的操作会影响原始值
sofish[1].bio = 'hackable';
console.log(arr);//  [1, cat: { bio: 'hackable' } ]
```

那么如何做呢？来一个 copy() 函数解决这个问题：

```
/* 复制对象
 * @param: obj {JavaScript Object} 原始对象
 * @param: isDeep {Boolean} 是否为深拷贝
 * @return: {JavaScript Object} 返回一个新的对象
 */
function copy(obj, isDeep) {
    var ret = obj.slice ? [] : {}, p;
    // 配合 is 函数使用
    if(!isDeep && is(obj, 'Array')) return obj.slice();
    for(p in obj) {
        var prop = obj[p];
        if(!obj.hasOwnProperty(p)) continue;
        if(is(prop, 'Object') || is(prop, 'Array')) {
          ret[p] = copy(prop, isDeep);
        } else {
            ret[p] = prop;
        }
    }
    return ret;
}
```

这样，我们就可以通过 copy(obj, isDeep) 函数来复制一个 Array 或者 Object 。可以测试一下：

```
var arr = [1, {bio: 'not a fish'}];
var sofish = copy(arr);

// 浅拷贝对于第一层的操作不影响原始值，但影响第二层
sofish.push('cat');
console.log(arr); //  [1, {bio: 'not a fish'}]
sofish[1].bio = 'hello world';
console.log(arr) //  [1, {bio: 'hello world'}]

// 深拷贝则不会影响原始值
sofish = copy(arr, 1);
sofish[1].bio = 'foo or bar'
console.log(arr); // [1, {bio: 'hello world'}]
```

来自： https://fe.ele.me/javascript-lei-xing-2/ (https://fe.ele.me/javascript-lei-xing-2/)

同类热门经验

1. Node.js 初体验 (/lib/view/open1326870121968.html)
2. JavaScript开发规范要求 (/lib/view/open1352263831610.html)
3. 使用拖拉操作来自定义网页界面布局并保存结果 (/lib/view/open1325064347889.html)
4. Nodejs入门学习，nodejs web开发入门，npm、express、socket配置安装、nodejs聊天室开发 (/lib/view/open1329050007640.html)
5. 利用HTML5同时上传多个文件 - resumable.js (/lib/view/open1327591300671.html)
6. nide：一个不错的Node.js开发工具IDE (/lib/view/open1325834128750.html)

阅读目录

open.com/doc/view/4a20dd6c60dd4a18b792e52d8c9ba389)
- javascript特效.chm (http://www.open-open.com/doc/view/34ee1d76364e4d5baf6ee17d53c501b2)
- Javascript 特效.chm (http://www.open-open.com/doc/view/e6033d8fc81c407f84e9e1eeecf01eb1)
- ArcGIS API for JavaScript & HTML5 应用开发.pdf (http://www.open-open.com/doc/view/ab32a438126c49838315f2f6230723d0)
- javascript-示例.chm (http://www.open-open.com/doc/view/1509663a4c0c4cb28e7368935731120f)
- JavaScript 语法.ppt (http://www.open-open.com/doc/view/5a2c78897e2c400d81b32959f6b5ef7b)
- JavaScript 语法概述.pdf (http://www.open-open.com/doc/view/fa02ef9a7b424936aca500f85f1b6961)
- JavaScript实例集.pdf (http://www.open-open.com/doc/view/c905690079c84dbd86de59a81ea235f7)
- The Future of JavaScript.pdf (http://www.open-open.com/doc/view/e3a2ad797831420b9ff803fdda21a3d4)
- JavaScript-极速快感.pdf (http://www.open-open.com/doc/view/d63ebea1701246cbbec4ab6fb5bd8f25)
- 《精通JavaScript+jQuery》.pdf (http://www.open-open.com/doc/view/89f5cf57aff4498cb5f105891004b30a)
- JavaScript 技巧大全.doc (http://www.open-open.com/doc/view/f2d7af4c2e82486db0415e9e4011587f)
- javascript学习笔记.pdf (http://www.open-open.com/doc/view/e5d6d9ff118d4f8fb5d3795aa8bda6e7)
- JavaScript速查手册.pdf (http://www.open-open.com/doc/view/aa2b272c93fc4f9c94cc9a3b721b52ec)
- JavaScript 学习笔记.doc (http://www.open-open.com/doc/view/114b3b45a64e4796acc8a53d99a573b9)

- 总结的JavaScript数据类型判定 (/lib/view/open1436767987943.html)
- Javascript数据类型转换规则 (/lib/view/open1436687950974.html)
- JavaScript的隐式类型转换浅析 (/lib/view/open1423986626452.html)
- JavaScript 代码的静态类型检查工具：Flow (/lib/view/open1416359257086.html)
- Nashorn - 在JDK 8中融合Java与JavaScript之力 (/lib/view/open1419901231265.html)
- JavaScript图表图形框架 -- GOJS简介 (/lib/view/open1435023502544.html)
- （Frontend Newbie）JavaScript基础之常见数据类型 (/lib/view/open1451787151417.html)

- Javascript闭包学习 (http://www.open-open.com/solution/view/1321491048452)
- 76个JavaScript教程资源免费下载 (http://www.open-open.com/solution/view/1372818526987)