Give You My Promise

随笔 Javascript ES6 Promise Markdown

What is Promise

Promise，允诺，许诺的意思，在Javascript中象征着希望与承诺，是ES6原生提供的对象。所谓Promise对象，就是代表了未来某个将要发生的事件（通常是一个异步操作）。它的好处在于，有了Promise对象，就可以将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数。此外，Promise对象还提供了一整套完整的接口，使得可以更加容易地控制异步操作。

在没有Promise的时代，我们可以想想这样一种场景：我们需要根据ajax请求返回的数据发送下一个请求，然后对其结果进行处理，因此，在第一个请求中我们传入一个回调函数，然后在这个回调函数中发送下一个请求再传入另一个回调函数，于是乎你会发现callback函数层层嵌套，到最后错综复杂难以维护。
再或者是如下这样(传说中的回调金字塔)：

```
loadImg('a.jpg', function() {
    loadImg('b.jpg', function() {
        loadImg('c.jpg', function() {
            console.log('all done!');
        });
    });
});
```

为了解决上面这种问题，Promise应运而生，专治各种不服。下面就来仔细谈谈怎么使用Promise。

How to use Promise

首先要了解所谓的`Promise/A+规范`：

- 一个promise可能有三种状态：等待（pending）、已完成（fulfilled）、已拒绝（rejected）
- 一个promise的状态只可能从“等待”转到“完成”态或者“拒绝”态，不能逆向转换，同时“完成”态和“拒绝”态不能相互转换
- promise必须实现then方法（可以说，then就是promise的核心），而且then必须返回一个

promise，同一个promise的then可以调用多次，并且回调的执行顺序跟它们被定义时的顺序一致

- then方法接受两个参数，第一个参数是成功时的回调，在promise由“等待”态转换到“完成”态时调用，另一个是失败时的回调，在promise由“等待”态转换到“拒绝”态时调用。同时，then可以接受另一个promise传入，也接受一个“类then”的对象或方法，即thenable对象

创建 Promise：

```
var promise = new Promise(function(resolve, reject) {
    // do a thing, possibly async, then…

    if (/* everything turned out fine */) {
        resolve("Stuff worked!");
    }
    else {
        reject(Error("It broke"));
    }
});
```

Promise 的构造器接受一个函数作为参数，它会传递给这个回调函数两个变量 resolve 和 reject。在回调函数中做一些异步操作，成功之后调用 resolve，否则调用 reject。

使用 Promise：

```
promise.then(function(result) {
    console.log(result); // "Stuff worked!"
}, function(err) {
    console.log(err); // Error: "It broke"
});
```

用于XMLHttpRequest：

```
function get(url) {
    return new Promise(function(resolve, reject) {
        var req = new XMLHttpRequest();
        req.open('GET', url);

        req.onload = function() {
            if (req.status == 200) {
                resolve(req.response);
            }
            else {
                reject(Error(req.statusText));
            }
        };

        req.onerror = function() {
            reject(Error("Network Error"));
        };

        req.send();
    });
}

get('story.json').then(function(response) {
    console.log("Success!", response);
}, function(error) {
    console.error("Failed!", error);
});
```

链式调用：

```
var promise = new Promise(function(resolve, reject) {
    resolve(1);
});

promise.then(function(val) {
    console.log(val); // 1
    return val + 2;
}).then(function(val) {
    console.log(val); // 3
});
```

其他API：

- Promise.cast(promise)
- Promise.cast(obj)
- Promise.resolve(thenable)

- Promise.resolve(obj)
- Promise.reject(obj)
- Promise.all(array)
- Promise.race(array)

是男人就要实现自己的Promise

在百行之内就可以实现一个Promise，所以，哪怕自己不动手实现，也要看别人是如何实现的吧？下面本人将带领大家看看别人家孩子的代码。

```
// status
var PENDING = 0,
    FULFILLED = 1,
    REJECTED = 2;

var Promise = function (fun) {
        var me = this,
            resolve = function (val) {
                me.resolve(val);
            },
            reject = function (val) {
                me.reject(val);
            };
        me._status = PENDING;
        me._onFulfilled = null;
        me._onRejected = null;
        (typeof fun === 'function') && fun(resolve, reject);
    }
```

一开始定义了三个状态量变量和一个Promise构造函数。看到这里，有没有发现一个很有意思的地方，`fun(resolve, reject)`是不是很绕？将内置的两个函数`resolve`，`reject`传给外面的函数调用，自己又调用外面传入的函数，好吧，本人特别不喜欢层层封装各种回调。

先实现几个基本简单的方法，都比较好懂啦：

```
var fn = Promise.prototype;

fn.isPromise = function (obj) {
    return obj instanceof Promise;
}

fn.resolve = function (obj) {
    var ret;
    if (!Promise.isPromise(obj)) {
        ret = obj;
        obj = new Promise();
    }
    setTimeout(function () {
        obj.resolve(ret);
    });
    return obj;
}

fn.reject = function (obj) {
    var ret;
    if (!Promise.isPromise(obj)) {
        ret = obj;
        obj = new Promise();
    }
    setTimeout(function () {
        obj.reject(ret);
    });
    return obj;
}
```

下面这个就是重点啦，再来个`then`：

```
fn.then = function (resolve, reject) {
    var pms = new Promise();
    this._onFulfilled = function (val) {
        var ret = resolve ? resolve(val) : val;
        if (Promise.isPromise(ret)) {
            ret.then(function (val) {
                pms.resolve(val);
            });
        }
        else{
            pms.resolve(ret);
        }
    };
    this._onRejected = function (val) {
        var ret = reject ? reject(val) : val;
        pms.reject(ret);
    };
    return pms;
}
```

我靠，一个基本的Promise就完成了，就是这么任性！
接下来就是补充其他函数啦，比如说all的实现：

```
fn.all = function (arr) {
    var pms = new Promise();
    var len = arr.length,
        i = -1,
        count = 0,
        results = [];
    while (++i < len) {
        ~function (i) {
            arr[i].then(
                function (val) {
                    results[i] = val;
                    if (++count === len) {
                        pms.resolve(results);
                    }
                },
                function (val) {
                    pms.reject(val);
                }
            );
        }(i);
    }
    return pms;
}
```

更多的这里就不再细说，总之，一个Promise这样就可以实现了。然后再随手一封装：

```
~function (global) {

    // Promise code ...

    global.Promise = Promise;
    try{
        module.exports = Promise;
    }
    catch (e){}
}(this);
```

最后的话就是，ES5,6,7提供了众多新特性和语法糖，Iterator，Generator啥的（python党好熟悉），有时间多多更深入的了解一下，这样才能做到自己各种写polyfill写插件写框架写库。越努力，越幸运。

Reference

[0]. ECMAScript 6入门
[1]. 如何实现一个ECMAScript 6 的promise功能
[2]. JavaScript Promise
[3]. JavaScript Promise启示录