模块化开发

笔记本：	好程序员		
创建时间：	2016/10/11 10:04	更新时间：	2016/10/17 16:30
作者：	【千锋互联】JS高级讲师--尹涛		

什么是模块化？

以前我们可能会这样做代码分工

```
modules.js -----
        var moduleA = {......}
        var moduleB = {......}
        var moduleC = {......}
```

然后每个页面都引入JS

```
<script src="modules.js"></script>
```

不管这个页面用到了几个模块，你都要把整个文件加载进来，于是我们就想，不如分开吧
然后我们的代码变成了这样:

```
moduleA.js -----
    var moduleA = {......}
moduleB.js -----
    var moduleB = {......}
```

```
moduleC.js -----
    var moduleC = {.....}
```

但是，由于各个模块之间存在相互引用的依赖关系，所以我们这样引入JS：

```
<script src="jquery.js"></script>
<script src="moduleB.js"></script>
<script src="moduleC.js"></script>
<script src="moduleA.js"></script>
```

假设A引用了C，而C引用了B，ABC里面都用到了JQ，你必须小心翼翼的确保它们的加载顺序是正确的

于是require.js出现了
按照requireJS当中的规范要求，你只需加载一个文件就可以而，但同时需要带上一个配置文件config.js
第一步：

```
<script data-main ="config.js" src ="require.js"  ></script>
```

这个require.js你可以从官网下载，这个config.js你必须自己完成。不用担心，它写起来非常简单:

```
config.js -----
        requirejs.config({
            baseUrl : "js",
            paths : {
```

```
        "jquery": "jquery/jquery.1.11.3"
    }
});
```

baseUrl的意思就是说你所有的路径都是基于它基础上的。
看的出来，我们正在配置一个文件的路径
"jquery": "jquery/jquery.1.11.3"
左边蓝色的是我们定义的模块名称，右边红色则是文件路径，注意文件没有后缀名，因为在require看来所有的模块都是JS

紧接着，我们可以定义自己的模块文件了，**moduleB.js**

首先mouduleB.js也是一个模块，但是我们必须对这个JS改造一下，让它符合require的规范要求，才能使用
moduleB.js -----

```
define(["jquery"],function(jq){
    //根据设定，模块B需要使用Jquery
    //define函数中的第一个参数，用来表明，我们即将引用哪个模块。
    //由于之前我们已经定义过了jquery模块，所以这里可以直接使用
    //注意，你如果不写声明，这里是用不了jQuery的
    //你如果注意到第一个参数是一个数组，那就应该猜到，其实我们可以一次引用好多模块
    //jq变量就是jquery，你如果不习惯，把名字改回$就是了

    return {
        start: function(){
            jq("#box").show(1000);
            console.log("模块B提供的start方法");
```

```
            //为什么这里要return呢？ 因为我们定义的模块要被其他人使用
            //那又为什么要return对象呢？为了更符合模块化思维以及面相对象的原则
            //我们返回一个对象而不是函数，使用会更方便（待会讲完怎么调用你就知道了）
        }
    }
});
```

好了，模块B定义完了，我们可以来试试。当然，用之前别忘了修改配置

```
config.js -----
    requirejs.config({
        baseUrl : "js",
        paths : {
            "jquery": "jquery/jquery.1.11.3",
            "moduleb" : "ms/moduleB"
        }
    });
注意：配置路径时，排名不分先后，顺序可以任意。
```

我们有一个index.html页面，包含一个index.js文件。

```
index.js -----
    require(["config",function(){ //config默认从项目根目录找，由于我们的config.js就是放在根目录下的
        //只有先引用config，才能引用配置好的所有模块
```

```
        require(["jquery","moduleb"],function($,mb){
            //我们引用两个模块jquery、moduleb
            mb.start(); //mb就是我们刚才在模块B中返回的对象
        })
    })
```

根据我们一开始的设定(A引用了C，而C引用了B，ABC里面都用到了JQ)
和上面的讲解，你能否自己完成moduleC.js和moduleA.js？

```
moduleC.js -----
    define(["moduleb","jquery"],function(mb,$){
        return {
            .......
        }
    });

moduleA.js -----
    define(["modulec","jquery"],function(mc,$){
        return {
            .......
        }
    });
```

别忘了配置：

```
config.js -----
..............
```

```
    paths: {
        "jquery": "jquery/jquery.1.11.3",
        "modulea" : "ms/moduleA",
        "moduleb" : "ms/moduleB",
        "modulec" : "ms/moduleC"
    }
..............
```

一些问题：

问：是不是所有要加载的模块，都必须写define函数，为什么jquery就没写，一样可以加载？

谁说JQ没写呢？这是jquery的部分源代码。

```
......
if ( typeof define === "function" && define.amd ) {
    define( "jquery", [], function() {
        return jQuery;
    });
}
......
```

所以，所有的JS都必须遵循require的规范，才能够正确的加载

问：我自己写了一大堆的工具函数，请问怎么加载？

首先，工具函数也可以用对象来表示

```
common.js -----
    define(function(){
        return {
            getStyle : function(){ ..... },
            randomColor : function(){ ..... }
        }
    });
```

当然如果你比较任性，非要写成这个样子：

```
common.js -----
    function getStyle(){ ..... }
    function randomColor(){ ..... }
```

解决办法也是有的:

请百度requireJS， shim配置

问：有一些插件，本身没有导出任何对象或核心函数，只是对某个框架的扩展，例如 **my.jquery.scroll.js** 该怎么用？

官方文档给出的答案是使用shim配置

```
config.js -----
    require.config({
        baseUrl: "js",
        paths: {
            "jquery" :  "jquery.1.11.3",
```

```
            "jquery.scroll" : "my.jquery.scroll"
        },
        shim: {
            "jquery.scroll": {
                deps: ["jquery"],
                exports: "jQuery.fn.scroll"
            }
        }
    });
```

这是第一种方式, **deps**表示它依赖了哪个模块，**exports**表示它输出哪个函数，函数名字要和插件代码里保持一致

使用方式：

```
index.js -----
    require(["config",function(){
        require(["jquery","jquery.scroll"],function($){
            $("#box").scroll(500);
        })
    })
```

==========

第二种配置方式：

```
config.js -----
    require.config({
        baseUrl: "js",
        paths: {
            "jquery" :  "jquery.1.11.3",
            "jquery.scroll" : "my.jquery.scroll"
        },
```

```
        shim: {
            "jquery.scroll":  ["jquery"]
        }
    });
```

使用方法同上

But! 官方文档又说，这样使用可能导致错误！ 请参考原文：

> **"shim"配置的重要注意事项:**
>
> - *shim配置仅设置了代码的依赖关系，想要实际加载shim指定的或涉及的模块，仍然需要一个常规的require/define调用。设置shim本身不会触发代码的加载。*
> - *请仅使用其他"shim"模块作为shim脚本的依赖，或那些没有依赖关系，并且在调用define()之前定义了全局变量(如jQuery或lodash)的AMD库。否则，如果你使用了一个AMD模块作为一个shim配置模块的依赖，在build之后，AMD模块可能在shim托管代码执行之前都不会被执行，这会导致错误。终极的解决方案是将所有shim托管代码都升级为含有可选的AMD define()调用。*

所以，终极解决方案是，把插件改成define调用.....

问：如果我用到了两个框架，Jquery和Zepto，他们的核心函数都叫$ ，那该怎么办呢？

```
require(["jquery","zepto"],function(jq,zepto){
    // 改个名字就不冲突了
```

```
})
```

说一说好处：

讲了这么多，requirejs所倡导的模块化开发，好处在哪里呢？

1 你有没有发现，整个项目当中，再也没有出现一个全局变量？
2 你有没有发现，你再也没有考虑过加载顺序的问题？
3 你有没有发现，即使两个框架名字冲突了也没关系？
4 而且你肯定没有发现，所有的JS文件的加载过程，已经变成了异步。
5 最后，你还发现，用了requireJS，你的代码想不写成面相对象都难？？

关于AMD和CMD

总在网上看大家讨论AMD和CMD，到底是什么东西呢？
AMD就是require所倡导的模块化开发的方式

遵循AMD规范的还有一个叫CommonJS的，不过还是和requireJS有些区别，主要针对NODE后端开发

另外还有一个大名鼎鼎的SeaJS，作者是淘宝的玉伯，于是诞生了CMD规范
根据作者自己的介绍，SeaJS各方面都比requireJS强大
这个就不做评价了

至于区别呢？

AMD推崇的是依赖前置：

```
define(["a","b","c"],function(a,b,c){ //我们把所有依赖提前写好了
    a.doSomething (); //这里直接使用
    b.doSometing();
})
```

CMD推崇的是依赖就近：

```
define(function(require, exports, module) {
    var a = require('./a');
    a.doSomething();
    .....
    var b = require('./b') // 依赖可以就近书写，用到的时候再加载
    b.doSomething()
})
```

当然requireJS也支持CMD的写法，不过作者本人是不推荐这么写的