模块化工具的使用感受

1. AMD（异步模块加载，代表：requirejs）、CMD（代表：seajs，国内玉伯提出的，不知道是不是规范）

**优点**：模块化（易维护，修改一个模块，则所有依赖这个模块的文件都得到更新），不污染全局变量，易测试，易清cdn缓存，易被非模块的js访问（调用全局函数require（module），callback），不阻塞页面（异步加载文件）。

**缺点**：模块多的时候需要消耗较多的http请求资源，（requirejs，模块加载后之后都会被执行，不是按需调用执行。seajs是按需执行，在一些异步操（例如某个事件发生后才会使用该模块的情况可以尽可能不执行不必要的代码），使用npm的模块时，需要手动把对应的文件放到assets/js目录下（像webpack这些在后端打包的工具可以直接requrie进来）。

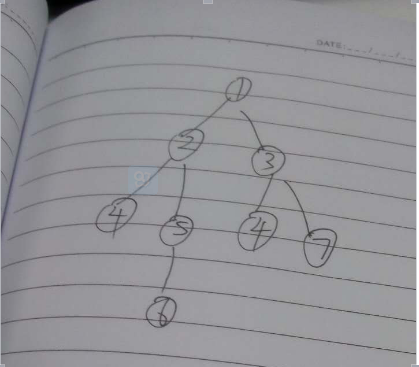
**优化**：打所有依赖打包到一个文件上，减少http请求。

**1.加载过程**：

①非打包环境：

根据define或require函数所定义的依赖数组，（1）把依赖文件下载来（因为是异步加载，每个依赖文件下载完成时顺序不定),如果下载下来的文件还有其他依赖模块，继续重复步骤（1），直到下载下来的文件没有其他依赖，就define或者require对应的回调函数。

如图：



相当于：

模块1：define/require（[2,3],callback);

模块2：define/require （[4,5],callback);

模块4：define/require（[],callback);

模块 5：define/require（[6],callback);

模块6：define/require（[],callback);

模块3：define/require([7,4],callback);

模块7：define/require（[],callback);

下面模拟一次执行过程(注意：因网络不确定，每次的执行顺序都可能不一样的，但能保证依赖顺序一样,下载完成后文件中非define和require定义的代码会被立即执行)

假设页面的入口文件是<script src=”模块1”>

1. 当模块1下载下来，发现其依赖2,3 ，则开始下载模块2和模块3（注意因为是异步加载，两个模块谁先下载完成时不确定的），
2. 2下载完成后 返现其依赖4,5，下载模块4,5。
3. 模块3下载完成，发现依赖模块4,7 ，因模块4正在下载，所以模块3只会下载模块7。
4. 模块4下载完成后，没有依赖其他模块，立即执行该模块回调。
5. 模块7下载完成，没有依赖其他模块，立即执行该模块回调。此时，模块3的依赖（4,7）执行完成，执行模块3的回调。
6. 模块5下载完成，发现依赖模块6，下载模块6。
7. 模块6下载完成，发现没有依赖其他模块，立即执行该模块回调。此时模块5的依赖加载完成，立即执行模块5的回调，此时模块2的依赖（4,5）也执行完成，立即执行模块2的回调，此时模块1的依赖执行（2，3）完成，立即执行模块1的回调，至此，程序执行完毕。

②打包环境：

把依赖合并到一个文件，相当于变成commonjs了

**2.测试：**

①非打包环境：

A.把js域名映射到本地资源服务器（可以直接映射到源码文件夹），直接修改本地代码，在本地测试好，然后构建并发布到测试机。

B.把本地js改好，构建好发布到测试机，在测试机上测试（通过sourece\_map查看源代码）

C.通过finder等工具把需要修改的js映射到本地的js

②打包环境：

需要调试的文件通过excludeShallow选项指定这些文件不需要打包

1. **清cdn缓存**

①非打包环境：

直接修改require配置文件的urlArgs v参数改为对应的版本号 ，然后把这个文件发布到测试机，然后清这个文件的cdn缓存

②打包环境：

直接清打包文件和配置文件

**4.测试代码**

git clone <https://github.com/13428282016/amd.git>

npm install

Host :

127.0.0.1 dev.com

127.0.0.1 assets.dev.com

①非打包环境：

gulp：

gulp server:web

gulp server:assets:dev

Build：

gulp build:js

Browser(请查看控制台):

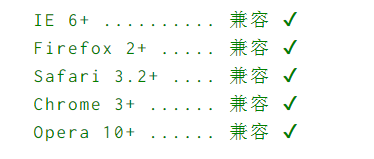
<http://dev.com:8080/amd.html>

<http://dev.com:8080/ad.html>

②打包环境:

一个样

1. **兼容**



1. **外部使用问题：**

①第三方放在我们的环境运行

A.如何导入我们的模块？

通过require（‘m’,callback）导入（此语句一定要在配置文件加载好之后导入）

B.如果第三方代码也支持AMD加载，如何加载第三方的依赖，如果对方的配置或者模块名和我们的冲突呢（关键是如何获取第三方的配置信息）？

第一种，把我们的配置信息加入到第三方（仅当配置不冲突）；

②我们的代码放在第三方的环境运行

A.如果第三方不支持模块系统？

需要根据依赖顺序手动添加script

B.如果第三方支持模块系统，还有如何加载我们的模块依赖，如果对方的配置或者模块名和我们的冲突呢，例如baseUrl冲突（关键是如何获取我们的配置信息）？

通过Require（url）导入，第一种，把第三方的配置信息加入到我们的配置中（仅当配置不冲突）；baseUrl冲突可以把配置换成绝对路径

第二种，如果配置或模块名冲突可以考虑临时切换配置（但是切换配置过程中，如果第三方的模块还在加载就会出错）

③如果第三方加载了jquery ，我们的代码又加载了一次会导致例如丢失事件的问题？

只能协商使用同一个jq了

④正确的办法是使用buildjs +命名空间 例如 kkplay.require

1. **config文件的位置**

①在requirejs加载后马上同步加载（或者合并到requirejs）

这种情况配置预先加载好，可以保证后面的require/define函数可以正确调用，但不能使用data-main属性（配置生效后data-main指向baseUrl+main.js，貌似不能改）

②把配置放在每个页面的入口文件

这种情况第三方需要使用我们的模块时，每次都要require（’配置文件‘），以确保我门的配置文件已经加载好了

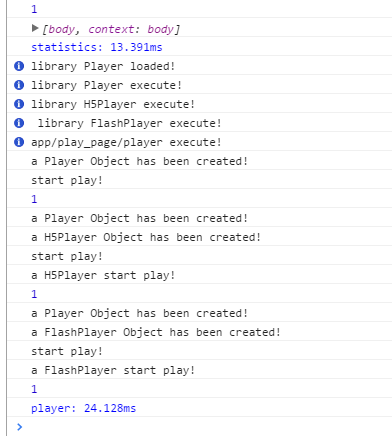
③把配置写在html里

这种情况不适合 用于静态页，因为稍微修改一下配置文件就要重新生成静态页

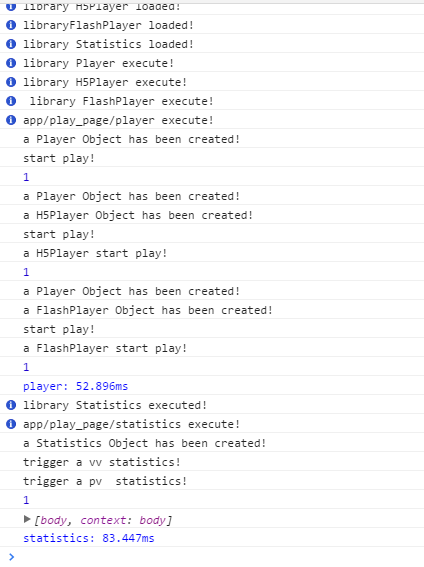
1. **优化**

**参见代码**

打包后：



**打包前：**



1. **其他问题**

**① 打包时不能使用http://形式的baseUrl导致页面不能加载某个非打包模块**

1. **直接在页面上加载requirejs，不把requirejs打包到入口文件**
2. **Requirejs的配置文件直接写到requirejs里面**
3. **入口文件依赖requirejs的配置文件，这样会导致第三方js使用我们的模块也需要依赖配置文件。**

**②如果把入口文件分开几个文件打包，暂时解决不了重复打包公共模块的问题（webpack 可以通过commonPlugin解决）**

## commonjs（通用模块加载，代表nodejs）