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）。

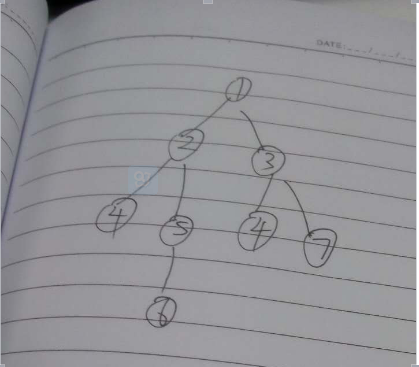
**优化**：打所有依赖打包到一个文件上，减少http请求。

**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：

**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：

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

②打包环境：

**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：

## commonjs（通用模块加载，代表nodejs）