/\*\*

\* 对于没有代码分割的，webpack会打包生成main.js一个大的自执行函数

\* 函数参数是一个对象，键值分别是路径和模块的函数

\* 函数内部定义了一些方法，包括\_\_webpack\_require\_\_

\* 函数内部执行逻辑会从一个入口开始进行webpackrequire按内部依赖的逻辑来执行函数

\*

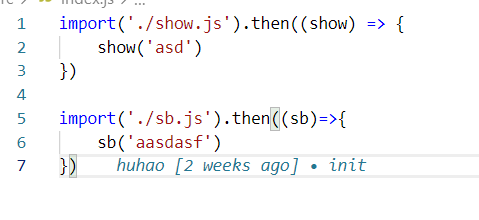
\*/

/\*

对于有代码分割的内容，webpack除了main.js还会生成0.js,1.js...等

原始代码:

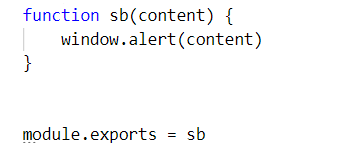
index.js:



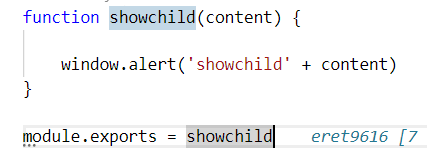
show.js:



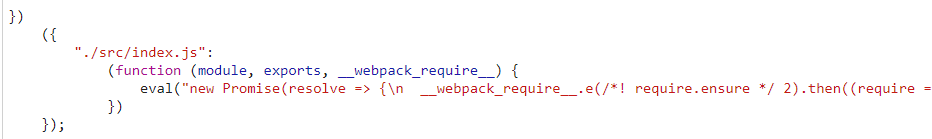
sb.js:



show-child.js:

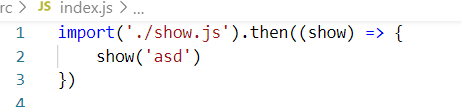


打包生成的main.js里的大自执行函数的函数参数是这样的:

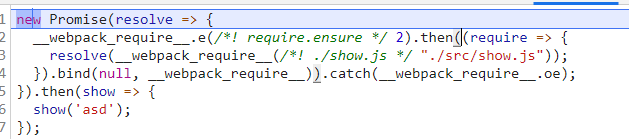


（./src/index.js是入口）

**下面的自执行函数是这样的:**



被转换成了



先执行\_\_webpack\_require\_\_.e然后在then里面resolve(\_\_webpack\_require\_\_(‘./src/show.js))

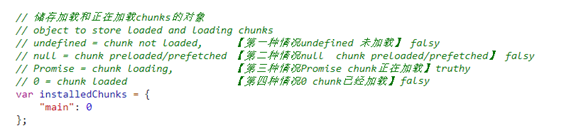
\_\_webpack\_require\_\_.e是干什么的:



webpack\_require\_\_.e内部声明了一个空数组promises

功能:

1.该函数会修改在大自执行函数中定义的installedChunks （显示安装chunk的状态）



2.创建了一个installedChunkData对象 是installedChunks[chunkId]

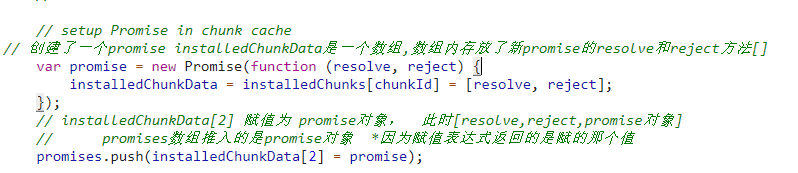
对象的结构是[resolve,reject,promise]

3.如果正在加载

会执行

promises.push(installedChunkData[2]); <- - -这里的[2]就是promise对象

否则 创建一个新promise对象



创建script标签:

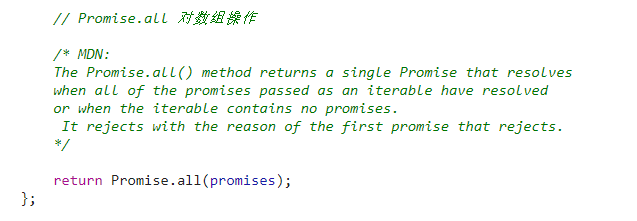




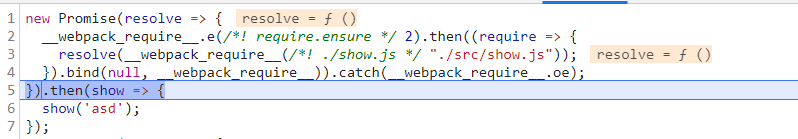
\_\_webpack\_require\_\_.e函数创建了一个script标签，引入0.js..等 插入到head标签的后面

返回一个promise，promise.then里面可以拿到一个数组，[]数组中内容是

\_\_webpack\_require\_\_.e函数最终返回的是一个promise



总结: \_\_webpack\_require\_\_.e是创建一个script标签，然后异步加载， 函数最后返回一个promise ，当promise被resolve或reject的时候可以被后面的.tn接收到



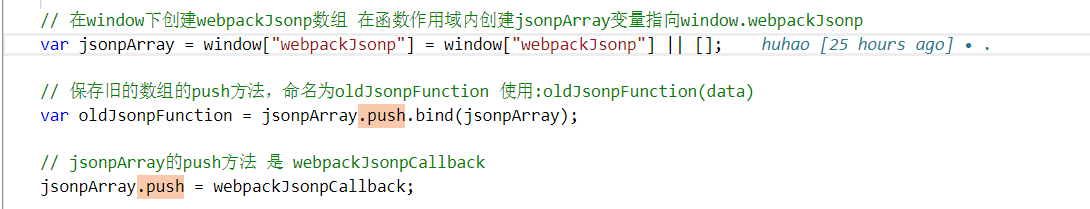
2.js内容:

创建的这个script标签（异步加载）的内容是什么呢：



window.webpackJsonp数组 中使用push方法 : [[2],{‘….’:{ }}]

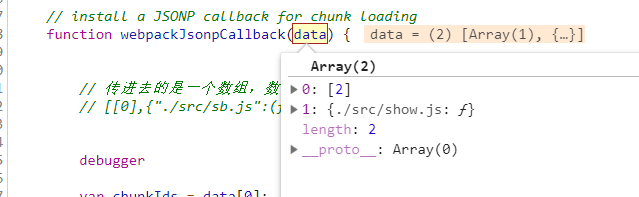
此时的push方法已经被改写了:

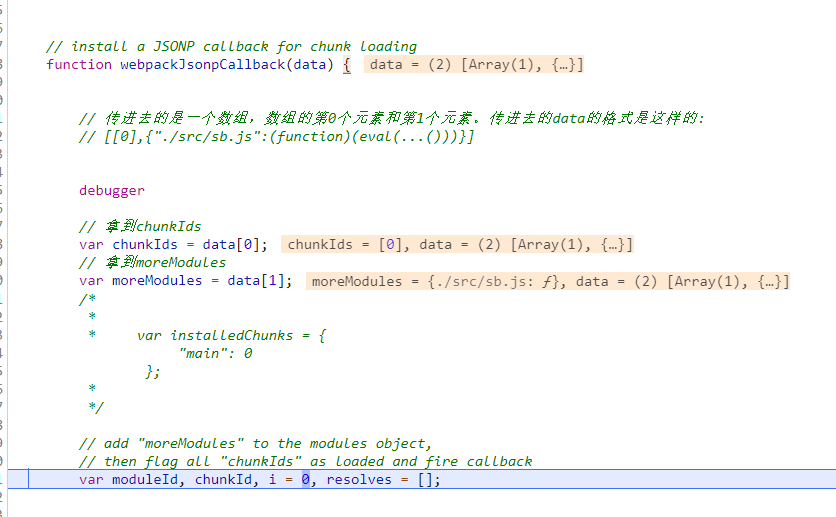


window下jsonpArray的push方法是webpackJsonpCallback

于是调用 webpackJsonpCallback([[2],{‘…’{}}]

执行:







该方法会将chunk们的加载状态记录管理，将moreModules装载到modules中

流程：创建script标签加载chunk

chunk 指一个要异步加载的大数据块

然后chunk里面一般是这样的 然后调用被改写过的window.webpackJsonp.push方法进行加载，将chunk们的加载状态记录管理，将moreModules装载到modules中

