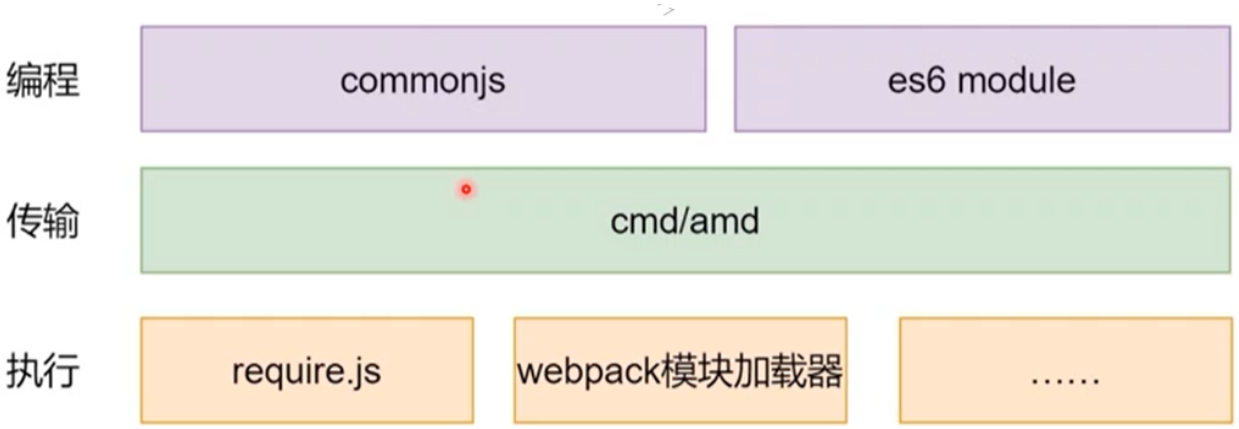
课程目标

CMD/AMD

整体结构



编程模型:我们的程序在哪里写，在commonjs和es6 module(可以这样说 commonjs一般在node ES6 一般在浏览器)，但是在浏览器和node环境下，这些model执行不了。这时候牵扯到传输层面的东西。传输层只有cmd和amd，早期cmd和amd不是作为一个中间层的，也是一个小框架，异步加载模块用的，我们程序写的是commonjs和es6 module如何编程cmd和amd，这个就需要一些的babal这些转义的工具。比如webpack打包，它就会打包成cmd或者amd这样的模型。但是真正去执行的就需要require.js和webpack模块加载器。。。等等。

AMD(Asychronous Moudle Definition 异步模块定义)

```
// 模块创建
// a.js
define(['functions/foo'], (foo) => {
   console.log( foo() )
})
```

模块创建的时候提供一个define函数，这个函数的第一个参数是一个数组，这数组里面叫做依赖 。上图有个依赖叫foo，这个依赖对应关系对应的就是funtions文件下面的foo，这个时候对应的一个依赖解析的过程。

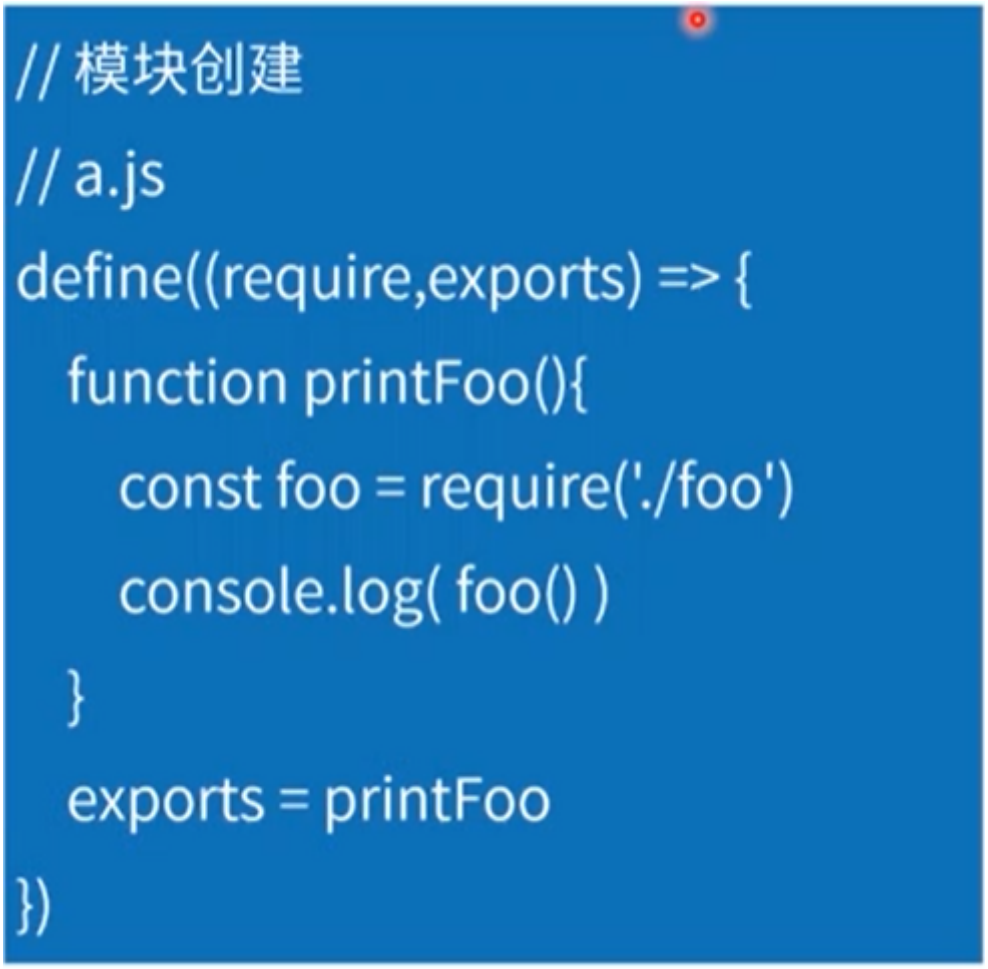
模块的使用

```
// 模块使用
// a.js
require(['jquery', 'a.js'], ($, printFoo) => {
   $("#foo").click(() =>{
      printFoo()
   })
})
```

require()第一个单数，都是一些依赖。等这些依赖解析完毕后，通过远程传输也好，本地解析也好。如 jquery依赖加载完毕后，可以使用$，printFOO来接收依赖。这种技巧我们可以好好学习一下。

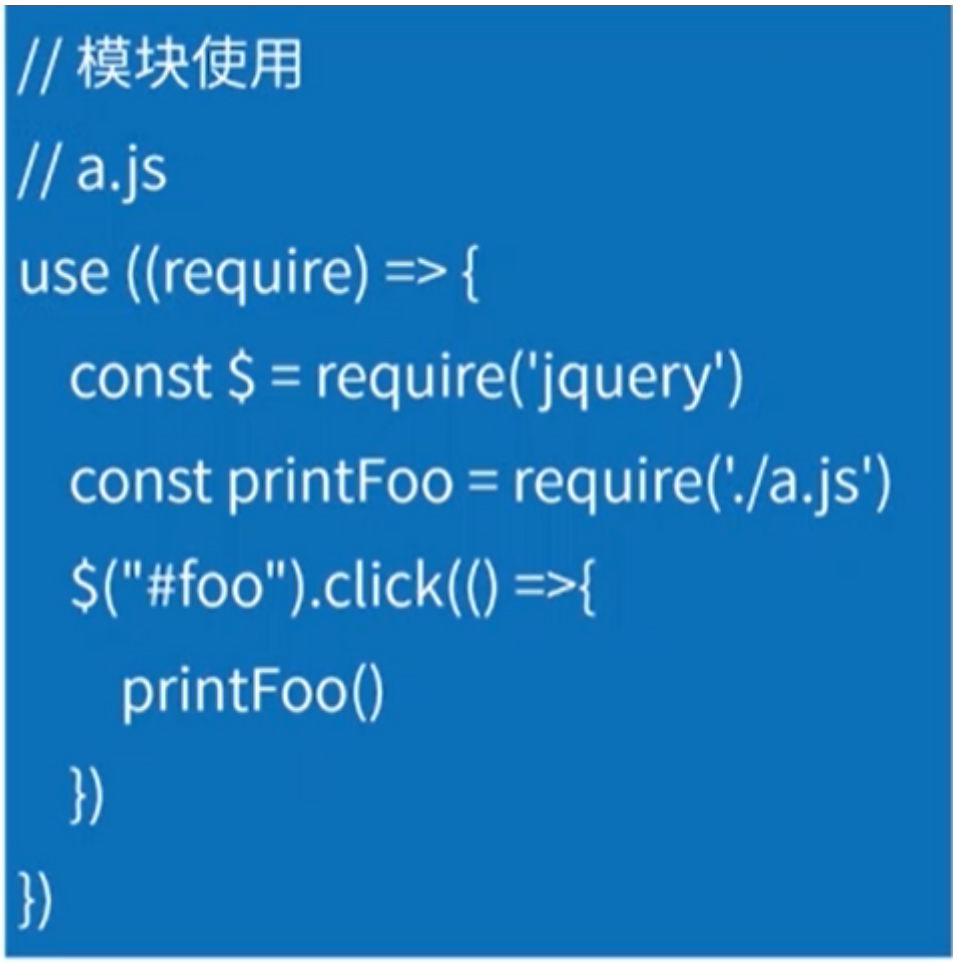
CMD(Common Moudle Definition)

模块创建



define()注入两个大家都会用函数一个require，一个是exports　一个导入函数，一个导出函数，CMD整个依赖都是后置的，是需要什么依赖用才导入什么依赖，然后再导出。

模块使用



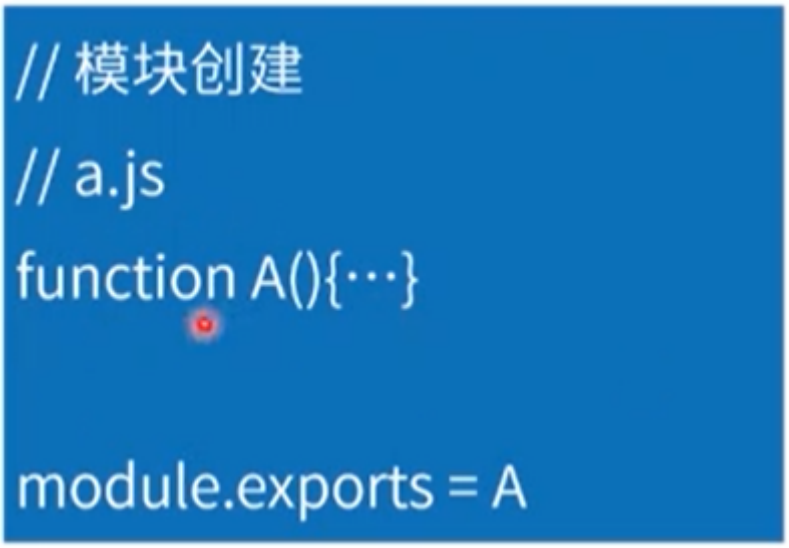
cmd更加先进了，依赖后置，按需引入，按需加载更加灵活人性化。但是引用还是比较麻烦，普及起来有点困难。

ES6/CommonJS

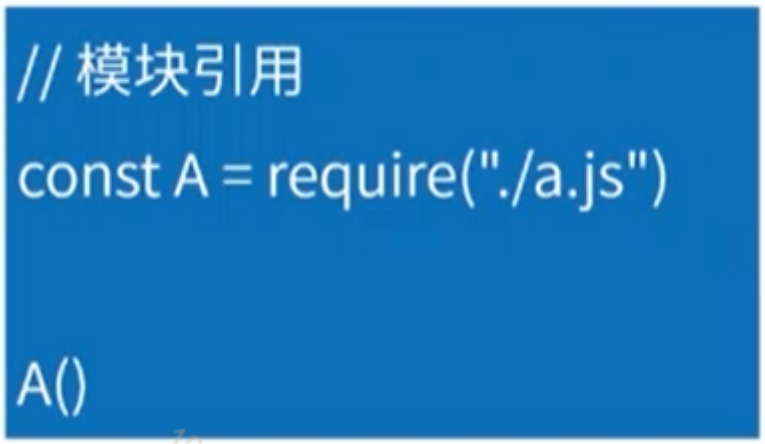
amd和cmd模型都有很大的一个缺陷，整体的语法太麻烦，普及起来有点困难。下面我们来看commonjs和ES6.

Commonjs源于nodejs的规范

模块创建：



模块使用：



模块的创建和模块引入的方式在node端的话，都没什么问题。 但是浏览器没有module也没有require。如果需要使用得需要下载相关依赖，进行转义支持。

ES6

模块创建

使用模块

加载器示例

实现一个AMD模式的require.js

服务端index.js

```
const express = require('express')
const app = require('express')()
const path = require('path')

app.use(express.static(path.resolve(__dirname,'libs')))


app.get('/',(req,res)=>{
 const html = `
<html lang="en">
```

```
<body>
mmp
</body>
<script src="require.js"></script>
<script >
 require.path = '/'
 require(['add','mult'],(add,mult)=>{
 console.log(add(3,5));
 console.log('======');
 console.log(mult(3,5));
 })
</script>
</html>
 `
 res.send(html)
})

//静态文件映射
app.get('/require.js',(req,res)=>{
 res.sendFile(path.resolve(__dirname,'require.js'))
})

app.listen(8099,()=>{

})
```

require.js

```
//定义函数
let modules = {}
//浏览器没有define函数，得手动定义
function define(name,func){
 console.log(12211)
 modules[name] = func
}
//定义路径
function lookup(name) {
  return require.path +name +".js"
 }
 //简单版本的require AMD模式
 function require(deps,callback){ //依赖 回调
```

```
function loadModule(name) {
return new Promise((resolve)=>{
const script = document.createElement('script')
script.src = lookup(name) //这里其实就是deps的加载[add,mult]的加载
script.addEventListener('load',()=>{
//script加载完毕后,会调用define函数，把依赖存期起来，并返回相关得依赖
console.log(modules[name]);
resolve(modules[name])
})
document.body.appendChild(script)
})
}
const promises = deps.map(loadModule)//执行所有得依赖
Promise.all(promises).then(res=>{
//执行完所有依赖后，在回调中用相关参数去接收依赖
callback(...res)
})

}


```

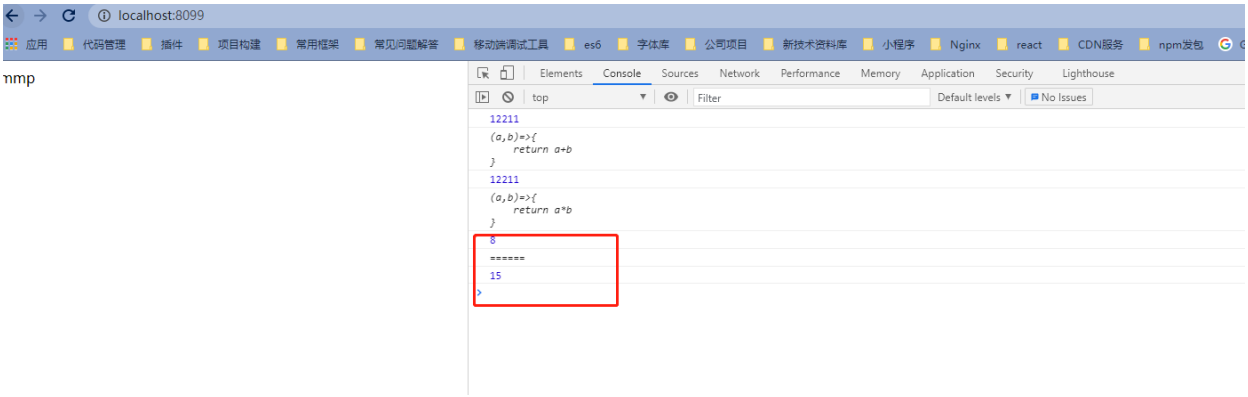
add模块

```
define('add',(a,b)=>{
 return a+b
})
```

mult模块

```
define('mult',(a,b)=>{
 return a*b
})
```

加载器相关示例结果



这只是一个demo，可以用webpack进行打包

课程小结

- CMD/AMD： 底层建设
- CommonJS/ES6 Module： 书写规范

在浏览器中，我们会使用CMD和AMD做底层建设，浏览器是不兼容这两种的commonJs/ES6 Module书写规范，但是我们写程序的时候就会用这两种，我可以用babel转义成CMD和AMD。总结，浏览器的只兼容 CMD和AMD的底层建设，commonJs/ES6 Module最终还是转义成CMD和AMD.