目录

[模块化开发 1](#_Toc25733)

[复习： 3](#_Toc11610)

[一、 异步加载模块 5](#_Toc7011)

[二、 配置 6](#_Toc3418)

[2.1 alias 7](#_Toc558)

[2.2 paths 7](#_Toc1724)

[2.3 map 8](#_Toc18017)

[2.4 vars 8](#_Toc31797)

[2.5 base 9](#_Toc9827)

[2.6加载css插件 9](#_Toc1802)

[2.7预加载文件 10](#_Toc16629)

[下午复习： 11](#_Toc21312)

[三、 Require.js 13](#_Toc8490)

[3.1 体验requirejs 13](#_Toc6715)

[3.2 定义模块 14](#_Toc29447)

[3.4 加载具有id的模块 16](#_Toc5976)

[3.5 模块的覆盖关系 16](#_Toc28218)

[3.6 接口定义 17](#_Toc28226)

[3.7 模块信息对象 19](#_Toc12178)

[四、 配置 19](#_Toc30999)

[4.1 paths 19](#_Toc16152)

[4.2 shim 20](#_Toc25755)

[4.3 map 20](#_Toc12273)

[4.4 baseUrl 21](#_Toc29442)

[五、 加载css插件 21](#_Toc28800)

# 复习：

seajs：

想要使用seajs必须现在页面中引入该文件，引入之后会向全局暴露两个变量：seajs、define

在模块的外部想要引入其它模块文件要通过seajs.use方法

该方法接收两个参数

第一个参数是一个数组，数组中的每一项都是一个模块文件地址，如果引入的之后一项文件可以省略数组

第二个参数是一个回调函数

函数中的参数就是前面模块向外暴露的功能

定义模块：

可以传递一个参数：

传递值类型的时候

这些数据会作为接口直接暴露

传递的是引用类型的时候

这些数据也会作为接口暴露

当传递一个函数的时候（90%以上都是使用这种方式）

此时函数中有三个参数

require: 引入其它模块文件的

exports: 向外暴露功能的对象

module: 模块信息对象

可以传递两个参数：

第一个参数可以是字符串表示模块的id

第一个参数还可以是数组，表示模块的依赖集合

第二个参数是回调函数（同第三种定义模块方式中的函数一致）

可以传递三个参数：

第一个参数是字符串表示模块的id

第二个参数是一个数组，表示模块的依赖集合

第三个参数是回调函数（同第三种定义模块方式中的函数一致）

requrie模块：

在一个模块中想要引入其它模块要分为两步走

第一步通过require引入模块文件

第二步通过模块文件读取模块

1 require 不能被简写

2 require 不能被修改

1 require不能被赋值

2 require不能赋值给其它变量

3 reuqire不能作为子函数中的参数使用

4 在子函数中不能被修改

3 require只能是字符串不能拼接

接口定义：

exports.接口

module.exports.接口

module.exports = 值类型

module.exports = 对象

module.exports = 函数

return 值类型

return 对象

return 函数

define(值类型)

define({})

模块信息对象：

每一个模块都有一个对象用来保存模块的信息

id: 表示模块的id

exports: 向外暴露功能的对象

uri: 是模块的文件路径，默认情况下（没有显示id）与模块id是一致的

# 异步加载模块

在浏览器端使用一个文件的时候，要到服务器端去加载， 这个过程是异步的，seajs为了实现这样的功能，在模块的内部提供了可以异步加载模块文件的方法: seajs.async方法

该方法可以实现在模块内部异步的加载模块文件

接收两个参数：

第一个参数是模块的依赖集合

如果是一个字符串，表示一个模块文件

如果是一个数组， 表示多个模块文件

第二个参数是模块函数

函数中的参数就是前面模块向外暴露的功能并且是一一对应的

require.aync加载的模块只能在模块内部使用，再模块外部不能够被访问

require.async与seajs.use方法是一致的，只不过seajs建议我们:

在模块的外部要使用seajs.use方法

在模块的内部要使用require.async方法

requrie.async方法和seajs.use方法一样不能够加载具有id的模块

但是我们可以利用依赖集合成功的使require.async方法加载具有id的模块

|  |
| --- |
| // 定义模块  define(function(require, exports, module) {  // define(["module/dom"], function(require, exports, module) {  // 通过requrie方法引入其他模块文件  // var dom = require("module/dom");  // console.log(dom);  // 异步加载  // var dom = require.async(["module/dom", "module/color"], function(dom, color) {  // var dom = require.async("myId", function(dom) {  var dom = require.async("module/dom", function(dom) {  // console.log(this, arguments);  console.log(dom);  });  // console.log(dom);  // console.log(require);  }) |

在条件语句中执行require.async以及requrie方法

对于require

在if执行块中

加载并执行模块文件

在if不执行块中

加载了模块文件，但是没有执行模块

对于async

在if执行块中

加载并执行模块文件

在if不执行块中

没有加载模块文件

|  |
| --- |
| 1. // 条件语句 2. if (true) { 3. // 这里会执行 4. // var dom = require("module/dom"); 5. // console.log(dom); 6. // 异步加载 7. require.async("module/dom", function(dom) { 8. console.log(dom); 9. }) 10. } else { 11. // 这里不能执行 12. // var color = require("module/color"); 13. // console.log(color); 14. // 异步加载 15. require.async("module/color", function(color) { 16. console.log(color); 17. }) 18. } |

在异步语句中执行require.async以及require方法

对于require

立即加载模块文件，但是会在异步语句执行之后执行require加载

对于async

没有立即加载模块文件，会在异步语句执行之后加载并执行该模块

|  |
| --- |
| 1. // 异步语句 2. setTimeout(function() { 3. // require加载 4. // var dom = require("module/dom"); 5. // console.log(dom); 6. // require.async加载 7. require.async("module/color", function(color) { 8. console.log(color); 9. }) 10. }, 2000) |

# 配置

seajs提供了config方法用来设置配置信息的

配置：本身这个框架具有的功能，只是隐藏了，需要开启要进行配置

插件：本身这个框架没有的功能，只是别人实现了，想要使用必须引入

## 2.1 alias

作用： 简化文件的引用

例如： 文件名称过长，我们可以简化它

值是一个对象

key: 新的文件名

value: 原始文件名

举例：

|  |
| --- |
| // 使用config方法进行配置  seajs.config({  // 配置alias  alias: {  "jquery": "module/js/jquery-1.7.2"  }  })  // 引入jquery  seajs.use("jquery", function(jq) {  console.log(jq);  }) |

结果：

|  |
| --- |
|  |

## 2.2 paths

作用: 简化文件路径

例如: 文件路径过长，我们可以简化它

值也是一个对象

key: 新的文件路径

value: 原始路径

举例：

|  |
| --- |
| // 配置  seajs.config({  // 配置paths  paths: {  "module": "module/js"  }  })  // 引入jquery  seajs.use("module/jquery-1.7.2", function(jq) {  console.log(jq);  }) |

结果：

|  |
| --- |
|  |

## 2.3 map

用于：批量处理文件的

值是一个二维数组

数组中的每一项都是一个数组

数组中的第一项表示要匹配到的文件

数组中的第二项表示要修改的文件

比如压缩之后的文件都是带有.min.js后缀，我们可以通过map进行配置， 配置哪些文件可以带有.min.js， 哪些文件可以不需要带.min.js

举例：

|  |
| --- |
| // 使用config方法进行配置  seajs.config({  // 配置map  map: [  ["jquery.js", "jquery.min.js"]  // [".js", ".min.js"]  ]  })  // 引入main.js入口文件  seajs.use("module/js/main"); |

main.js:

|  |
| --- |
| 1. // 这里是入口文件 2. define(function(require, exports, module) { 3. // 引入jquery.min.js 4. var jq = require("module/js/jquery"); 5. console.log(jq); 6. }) |

## 2.4 vars

作用： 解决字符串不能拼接的问题

值是一个对象

key: 模板变量名称

value: 模板变量名称对应的值

举例：

|  |
| --- |
| // 使用config方法进行配置  seajs.config({  // 配置vars  vars: {  "q": "query"  }  })  // 引入main.js入口文件  seajs.use("module/js/main"); |

main.js:

|  |
| --- |
| 1. // 这里是入口文件 2. define(function(require, exports, module) { 3. // 引入jquery-1.7.2.js文件 4. // var jq = require("module/js/j"+ "query" +"-1.7.2"); 5. var jq = require("module/js/j{q}-1.7.2"); 6. console.log(jq); 7. }) |

结果：

|  |
| --- |
|  |

## 2.5 base

作用：用于改变根目录

|  |
| --- |
| // 进行配置  seajs.config({  // 配置base  base: "module/js"  })  // 引入main.js入口文件  seajs.use("main", function(main) {  console.log(main);  }); |

## 2.6加载css插件

加载css插件使用起来非常简单

第一步要在seajs.js文件之后引入seajs-css.js文件

第二步直接在模块中加载css文件即可

seajs对js文件敏感，但是对css文件不敏感，所以要加上.css拓展名

举例：

|  |
| --- |
| define(function(require, exports, module) {  // 引入index.css文件  require("module/css/index.css");  }) |

## 2.7预加载文件

使用预加载文件也非常简单

第一步要在seajs文件之后引入seajs-proload.js文件

第二步要进行配置，配置proload

值是一个数组， 数组中的每一项都是要预加载的文件

我们就可以使用这些预加载的文件

预加载文件的加载时间点

是seajs.use方法之后

模块加载之前

通常，我们要将预加载的文件尽量放在最前面进行配置

|  |
| --- |
| <script type="text/javascript" src="sea.js"></script>  <!-- 第一步要在sea.js文件之后引入seajs=preload.js -->  <script type="text/javascript" src="seajs-preload.js"></script>  <script type="text/javascript">  // 进行preload配置  seajs.config({  preload: ["jquery-1.7.2"]  })  //引入mian.js  seajs.use("main", function(main) {  console.log(main);  console.log($);  })  </script> |

## 下午复习：

异步加载模块：

seajs中提供了一个可以在模块内部异步加载模块的方法: require.async

该方法接收两个参数

第一个参数是模块的依赖集合

如果是一个字符串表示一个模块文件

如果是一个数组表示多个模块文件

第二个参数是模块函数

函数中的参数就是前面模块向外暴露的功能

require.async与seajs,use方法一致

只不过seajs中建议我们

在模块的外部要使用seajs,use方法

在模块的内部要使用requrie.async

配置：

本身这个框架就有的功能，只是隐藏了，需要开启要进行配置

插件：

本身这个矿建没有的功能，只是别人实现了，想要使用必须引入

alias:

作用：简化文件的引用

值是一个对象  
 key: 新的文件名

value: 原始文件名称

paths:

作用: 简化文件的路径

值是一个对象

key: 新的文件路径

value: 原始路径

map:

作用：批量处理文件

值是一个二维数组

数组中的每一项都是一个数组

数组中的第一项表示要匹配的文件

数组中的第二项表示要修改的文件

vars:

解决字符串不能拼接的问题

值是一个对象

key: 模板变量名称

value: 模板变量名称对应的值

base:

改变根目录的

加载css插件：

使用css插件非常简单

第一步要在seajs文件之后引入seajs-css.js

第二步直接在模块中引入css文件即可

需要注意的是引入的css文件不能省略.css拓展名

预加载文件：

使用预加载文件也非常简单

第一步在seajs文件之后引入seajs-preload.js文件

第二步要进行preload配置

值是一个数组

数组中的每一项都是一个模块文件

加载预加载文件的时间点

在seajs.use方法执行之后

在模块加载之前

# Require.js

Requirejs是第一个模块化开发框架，提出了Module transports规范，基于前端模型的一个规范（加载时候都是异步的），又借鉴Nodejs,也支持commonjs规范，requirejs推荐使用commonjs规范

Requirejs是基于AMD规范

Requirejs的官网：

requirejs.org

requirejs.cn

## 3.1 体验requirejs

想要使用requirejs与seajs是一样的要在页面中引入文件

根目录：

如果有data-main属性，此时是以data-main的属性值为根目录

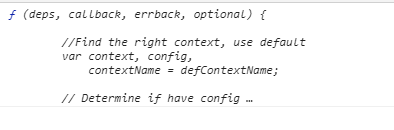
如果没有data-main属性，则是以html文件所在的目录为根目录

在seajs中，是以seajs所在的文件目录决定的

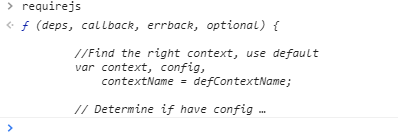
在requirejs中，是以引入的html文件目录所决定的

当引入requirejs之后会向全局暴露三个变量：

require:

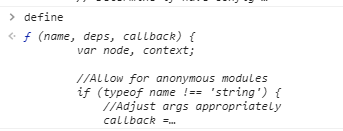


requirejs:



require和requirejs是一样的

define:



在seajs中是通过seajs.use方法引入项目的入口文件

在requirejs中的script标签上有一个data-main的属性来引入项目入口文件

|  |
| --- |
| 1. <script type="text/javascript" data-main="js/main.js" src="require.js"></script> |

除了以上方式可以引入项目入口文件，还有一种方式可以引入入口文件

requirejs中的require(requirejs)方法相当于seajs中的seajs对象

因此，require方法也可以引入项目入口文件

该方法接收两个参数

第一个参数是模块依赖集合，是一个数组，数组中的每一项都是一个模块文件

必须是数组

第二个参数是回调函数

函数中的参数就是与前面模块向外暴露的功能 （一一对应的）

作用域是全局作用域

requirejs对js文件比较敏感，因此可以省略.js拓展名

|  |
| --- |
| 1. // 通过require方法来引入项目入口文件 2. require(["js/main.js", "js/dom.js"], function() { 3. console.log(this, arguments); 4. }) |

## 3.2 定义模块

requirejs中也提供了一个define方法，用来定义模块：

可以传递一个参数

如果是字符串，这种方式不支持

对象也是支持的，但是不支持数组

当传递一个函数的时候，此时函数遵循的是commonjs规范，requirejs提供了参数注入的技术，如果想要使用哪个模块，就必须注入哪个模块

当没有注入exports和module的时候

作用域是全局作用域

当注入了exports和module的时候

作用域是向外暴露功能的对象

|  |
| --- |
| 1. // 传递函数 2. define(function(requrie, exports, module) { 3. console.log(this, arguments); 4. }) |

结果：

|  |
| --- |
|  |

可以传递两个参数：

第一个参数是一个字符串，表示模块的id

第一个参数还可以是数组，表示模块的依赖集合

第二个参数回调函数

如果第一个参数是字符串，此时函数遵循的是commonjs规范(同上面传递一个函数中的表现形式一致)  
 如果第一个参数是数组，此时函数遵循的是module transports规范，想要使用哪个模块，就必须在依赖集合中去注入，包括内置模块（require，exports, module）

如果没有注入exports和module

作用域是全局作用域

如果注入了exports和module

作用域是向外暴露功能的对象

|  |
| --- |
| 1. // 传递两个参数 第一个参数是字符串 2. // define("abc", function() { 3. define("dom", function(require, exports, module) { 4. console.log(this, arguments); 5. }) 6. // 传递两个参数 第一个参数是一个数组 7. define(["color", "require", "exports", "module"], function(color, require, exports, module) { 8. // console.log(this, arguments); 9. console.log(color); 10. console.log(require); 11. console.log(exports); 12. console.log(module); 13. }) |

可以传递三个参数：

第一个参数字符串，表示模块的id

第二个参数是数组，表示模块的依赖集合

第三个参数回调函数，此时函数遵循module transports规范（同上面的函数表现形式一致）

在module transports规范中定义的模块文件名称不要与内置模块(require, exports, module)是相同的

在工作中，最后一种方式是最常用的（id一定要与模块路径统一）

|  |
| --- |
| 1. // 传递三个参数 2. // define("abc", ["color"], function() { 3. define("dom", ["color", "require", "exports", "module"], function(color, require, exports, module) { 4. console.log(this, arguments); 5. }) |

## 3.4 加载具有id的模块

在一个模块中如果存在显示的id，将无法使用该模块

在一个模块中没有存在显示的id，是可以直接引用

想要加载具有id的模块要分为两步走

第一步要在依赖集合中去加载模块文件

第二步通过require方法指定模块的id(这里的require是异步方法)

commonjs规范是无法引入具有id的模块的

|  |
| --- |
| 1. // 加载具有id的模块 2. define(["color", "require"], function(color, require) { 3. // console.log(this, arguments); 4. // 通过require指定模块的id 5. var color = require(["myId"], function(color) { 6. console.log(color); 7. }) 8. // console.log(color); 9. // console.log(require); 10. }) |

## 3.5 模块的覆盖关系

如果一个模块文件中存在两个没有id的模块， 上面的模块覆盖下面的模块

如果一个模块文件中存在两个具有id的模块， 上面的模块覆盖下面的模块

如果一个模块文件中存在两个不同id的模块，可以是同时存在的，以require指定的id为准

|  |
| --- |
| 1. // 定义模块 2. define(['color', 'require'], function(color, require) { 3. // console.log(color); 4. // console.log(require); 5. require(['myId1', 'myId2'], function(color1, color2) { 6. console.log(color1, color2); 7. }) 8. }) |

color.js:

|  |
| --- |
| 1. // 定义两个没有id的模块 2. define({ 3. b: 2 4. }) 5. define({ 6. a: 1 7. }) 8. // 定义两个具有id的模块 9. define('myId', { 10. b: 2 11. }) 12. define('myId', { 13. a: 1 14. }) 15. // 定义两个不同id的模块 16. define('myId1', { 17. a: 1 18. }) 19. define('myId2', { 20. b: 2 21. }) |

## 3.6 接口定义

在seajs中定义接口的方式：

exports.接口

module.exports.接口

module.exports = 值类型

module.exports = 对象

module.exports = 函数

以上几种方式是基于commonjs规范的，在requirejs中同样适用

return 值类型

return 对象

return 函数

这几种方式是module transports规范， 因此它们不依赖于require, exports,module 开发更灵活

define（值类型） 字符串是不支持的

define(对象) 这种方式也支持

除了以上几种方式之外，当注入exports和module的时候，作用域是向外暴露功能的对象

|  |
| --- |
| 1. // 定义模块 2. // commonjs规范 3. define(function(require, exports, module) { 4. 第一种定义接口的方式 5. exports.title = "nihao"; 6. 以下方式是绝对不允许使用 7. exports = { 9. } 10. 第二种定义接口的方式 11. module.exports.title = "nihao"; 12. 3 13. module.exports = 123; 14. 4 15. module.exports = { 16. num: 100 17. } 18. 5 19. 这种方式向外暴露的内容，外部要作为方法使用 20. module.exports = function() { 21. console.log("success"); 22. } 23. }) 24. // module transports规范 25. define([], function() { 26. 6 27. return 123; 28. 7 29. return { 30. a: 1, 31. b: 2 32. } 33. 8 34. return function() { 35. console.log("success"); 36. } 37. }) 38. // 9 39. define(123); 40. // define("abc"); // 这种方式不允许 41. define(true); 42. // 10 43. define({ 44. a: 1, 45. b: 2 46. }) 47. // 当注入exports和module的时候作用域是向外暴露功能的对象 48. define(function(require, exports, module) { 49. // console.log(this, arguments); 50. this.color = "red"; 51. }) |

## 3.7 模块信息对象

id: 模块的id，默认与uri不一致

uri: 表示模块的文件路径，相对于html文件的

如果没有data-main属性，会在路径的前面多个./

config：用于定义配置信息

exports: 向外暴露功能的对象

查看：

|  |
| --- |
|  |

# 配置

配置: 本身这个框架具有的功能， 只是隐藏了， 需要开启的时候要进行配置

requirejs提供了config方法进行配置信息

## paths

作用：简化文件的路径

值是一个对象

key: 新的文件名称

value：原始文件名

举例：

|  |
| --- |
| 1. // 进行配置 2. requirejs.config({ 3. // 配置paths 4. paths: { 5. "module": "module/header" 6. } 7. }) 8. // 这里是入口文件 9. define(["module/header"], function(header) { 10. console.log(header); 11. }) |

结果：

|  |
| --- |
|  |

## shim

用于将文件转为模块的

key: 模块的文件路径

vlaue是一个对象

exports: 配置向外暴露的对象

deps: 配置依赖集合

举例：

|  |
| --- |
| 1. // 配置 2. require.config({ 3. // 配置shim 4. shim: { 5. "lib/jquery-1.7.2": { 6. exports: "$", 7. deps: [] 8. } 9. } 10. }) 11. // 这里是入口文件 12. define(["module/header/header"]); |

## map

作用：配置模块文件的

key: 模块的文件路径

value是一个对象

key: 引用的模块文件

value: 匹配的模块文件

举例：

|  |
| --- |
| 1. // 配置 2. require.config({ 3. // 配置shim 4. shim: { 5. "lib/jquery-1.7.2": { 6. // 配置接口 7. exports: "$", 8. // 配置依赖集合 9. deps: [] 10. }, 11. "lib/jquery-100": { 12. // 配置接口 13. exports: "$", 14. // 配置依赖集合 15. deps: [] 16. } 17. }, 18. // 配置map 19. map: { 20. "module/header": { 21. "lib/jquery": "lib/jquery-1.7.2" 22. }, 23. "module/footer": { 24. "lib/jquery": "lib/jquery-100" 25. } 26. } 27. }) 28. // 这里是入口文件 29. define(["module/header/header", "module/footer/footer"]); |

## baseUrl

改变根目录

baseUrl > data-main > require

# 加载css插件

在requirejs中不能直接引入css文件

要做为模块来引入，所以要进行配置

require.config({

map: {

“\*”: {

“css”: 文件目录

}

}

})

在引入css文件的模块中要加上css!前缀

|  |
| --- |
| 1. // 定义模块 2. define(["css!module/header/index.css"]); |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |