前端必备基础概念与实战

- JavaScript 技术体系
  - 发展历史与标准化
    - 无标准阶段
      - 1993 年 NCSA 发布了 NCSA Mosaic
      - 1994 年 发布 Mosaic Netscape 0.9
        - 公司 Netscape Communications
        - Netscape Navigator
      - 1995 年 Mocha 语言发布
        - 网景决定发明一种与 Java 搭配使用的辅助脚本语言并且语法上有些类似
        - 9月 改名 LiveScript
        - 12 月 改名 JavaScript
          - Netscape Navigator 2.0 Beta 3
        - 当时网景公司与昇阳计算机公司组成的开发联盟为了让这门语言搭上 Java 这个编程语言"热词"，因此将其临时改名为 JavaScript，日后这成为大众对这门语言有诸多误解的原因之一。
          - zh.wikipedia.org/wiki/JavaScript
      - 微软
        - Internet Explorer 3
        - JScript：一种 JavaScript 实现
        - Internet Explorer 4
      - "用 Netscape 可达到最佳效果" 或 "用 IE 可达到最佳效果"
    - 标准化阶段
      - 1996 年 11 月
        - 网景向 ECMA（欧洲计算机制造商协会）提交标准
      - 1997 年 6 月
        - ECMA 以 JavaScript 语言为基础制定了ECMAScript 标准规范 ECMA-262。

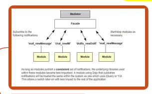
版本	发布日期	基于	Netscape Navigator	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.0	1996年3月		2.0		3.0			
1.1	1996年8月		3.0					
1.2	1997年6月		4.0 – 4.05			3		
1.3	1998年10月	ECMA-262 1st + 2nd edition	4.06 – 4.7x		4.0	5[21]		
1.4			Netscape Server			6		
1.5	2000年11月	ECMA-262 3rd edition	6.0	1.0	5.5（JScript 5.5） 6（JScript 5.6） 7（JScript 5.7） 8（JScript 5.8）	7.0	3.0–5	1.0 – 10.0.666
1.6	2005年11月	1.5 + Array extras + Array and string generics + E4X		1.5				
1.7	2006年10月	1.6 + Pythonic generators + Iterators + Let		2.0				28.0.1500.95
1.8	2008年6月	1.7 + Generator expressions + Expression closures		3.0		11.50		
1.8.1		1.8 + native JSON support + Minor updates		3.5				
1.8.2	2009年6月22日	1.8.1 + Minor updates		3.6				
1.8.5	2010年7月27日	1.8.2 + New features for ECMA-262 5th edition compliance		4.0				

        - ActionScript 和 JScript 也都是 ECMAScript 规范的实现语言
      - ECMAScript 标准化

版本	发表日期	与前版本的差异
1	1997年6月	首版
2	1998年6月	格式修正，以使得其形式与ISO/IEC16262国际标准一致
3	1999年12月	强大的正则表达式，更好的词法作用域链处理，新的控制指令，异常处理，错误定义更加明确，数据输出的格式化及其它改变
4	放弃	由于关于语言的复杂性出现分歧，第4版本被放弃，其中的部分成为了第5版本及Harmony的基础
5	2009年12月	新增"严格模式（strict mode）"，一个子集用作提供更彻底的错误检查,以避免结构出错。澄清了许多第3版本的模糊规范，并适应了与规范不一致的真实世界实现的行为。增加了部分新功能，如getters及setters，支持JSON以及在对象属性上更完整的反射[4][5][6][7][8]
6	2015年6月	ECMAScript 2015（ES2015），第 6 版，最早被称作是 ECMAScript 6（ES6），添加了类和模块的语法，其他特性包括迭代器，Python风格的生成器和生成器表达式，箭头函数，二进制数据，静态类型数组，集合（maps，sets 和 weak maps），promise，reflection 和 proxies。作为最早的 ECMAScript Harmony 版本，也被叫做ES6 Harmony。
7	2016年6月	ECMAScript 2016（ES2016），第 7 版，多个新的概念和语言特性[9]
8	2017年6月	ECMAScript 2017（ES2017），第 8 版，多个新的概念和语言特性[10]
9	2018年6月	ECMAScript 2018 （ES2018），第 9 版，包含了异步循环，生成器，新的正则表达式特性和 rest/spread 语法。

脚本引擎	参考应用程序	兼容性[11]			
		ES5[12]	ES6[13]	ES7[14]	较新（2016+）[14][15]
Chakra	Microsoft Edge	100%	96%	100%	54%
SpiderMonkey	Firefox	100%	98%	100%	77%
Chrome V8	Google Chrome、Opera	100%	98%	100%	93%
JavaScriptCore（Nitro）	Safari	97%	99%	100%	83%

  - 模块化



    - huangxuan.me/js-module-7day/
  - 模块化
    - CommonJS
      - 主要实现者：Node.js
        - exports
        - require

```
// ./blog.js
//导出模块
exports.blog={
    say:function(name){
        return 'hello, '+name;
    }
}
```

```
// ./app.js
//加载模块
var blog=require('./blog').blog;
console.log(blog.say('zhaiqianfeng'));
```

执行输出

```
$ node app.js
hello, zhaiqianfeng
```

    - CommonJS 模块的加载不适合网络加载，依赖的模块未加载会出现错误。
    - AMD (Asynchronous Module Definition)
      - 接口定义 define(id?, dependencies?, factory);

如定义AMD模块

```
// ./base.js
define(function(){
    return {
        say:function(name){
            return 'hello,'+name;
        }
    };
});
```

AMD调用模块也是使用关键字require，但不同于CommonJS，而是和AMD定义相似

```
require([module], callback);
```

因此调用base模块的代码如

```
// ./app.js
require(['base'],function(base){
    var words=base.say('zhaiqianfeng');
        document.write(words);
});
```

先引用AMD的实现RequireJS，html代码如下

```
<script src="require.js"></script>
<script src="app.js"></script>
```

      - 主要实现者 RequireJS
    - CMD (Common Module Definition)
      - AMD 的优化版
        - AMD 是依赖前置，提前加载依赖；而 CMD 依赖后置，使用时才加载。
        - RequireJS 已提供了延迟加载功能
      - 实现：SeaJS
    - UMD (Universal Module Definition)
      - CommonJS 侧重服务器，而 AMD 侧重于浏览器，两者的模块不能共享。UMD 的思想很简单：判断是 AMD 则使用 AMD 方式，是 CommonJS 则使用 CommonJS 方式，都不是则将模块公开给全局 (window 或 global)。

```
(function (root, factory) {
    if (typeof define === 'function' && define.amd) {
        // AMD方式
        define(['b'], function (b) {
            return (root.returnExportsGlobal = factory(b));
        });
    } else if (typeof module === 'object' && module.exports) {
        // Node/CommonJS方式
        module.exports = factory(require('b'));
    } else {
        // 公开暴露给全局
        root.returnExportsGlobal = factory(root.b);
    }
}(this, function (b) {
    return {};
}));
```

    - ES6
      - ES6 自带的模块化
    - SystemJS
      - SystemJS 是一个通用的模块加载器，它能在浏览器或者 NodeJS 上动态加载模块，并且支持 CommonJS、AMD、全局模块对象和 ES6 模块。通过使用插件，它不仅可以加载 JavaScript，还可以加载 CoffeeScript 和 TypeScript。SystemJS 的另一个优点是，它建立在 ES6 模块加载器之上，所以它的语法和 API 在将来很可能是语言的一部分，这会让我们的代码更不会过时。

要异步输入一个模块，可以用如下语法：

JavaScript 代码：

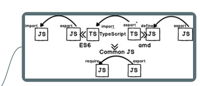
```
System.import('module-name');
```

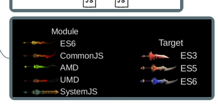
然后我们可以用配置 API 来配置 SystemJS 的行为：

JavaScript 代码：

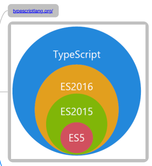
```
System.config({
  transplier: 'babel',
  baseURL: '/app'
});
```

    - 经典图片





- Babel 的使用
  - babeljs.io/
  - babeljs.io/docs/en/babel-preset-env
- TypeScript
  - typescriptlang.org/



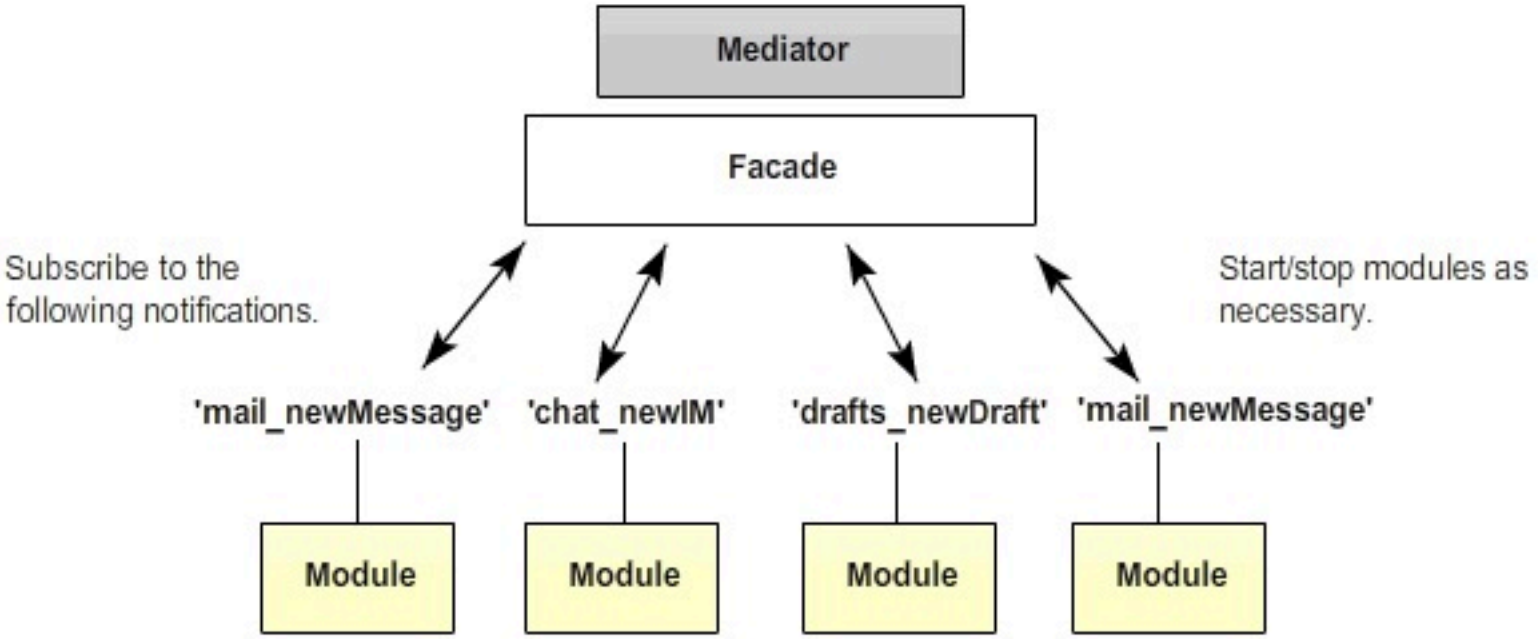
  - TypeScript 环境搭建与代码实战
    - typescriptlang.org/docs/handbook/typescript-in-5-minutes.html
- Webpack 的作用与实战
  - webpack.github.io/
  - typescriptlang.org/docs/handbook/react-&-webpack.html
- 再看可以开发 Web / 移动端 / 桌面端 / 服务器端的 JavaScript
  - nodejs.org/en/
  - npmjs.com/
- 前端三剑客 Angular / React / Vue.js 框架对比
  - angular.io/
  - reactjs.org/
  - cn.vuejs.org/
- 移动端混合开发 Ionic / React Native / Weex 框架对比
  - ionicframework.com/
  - facebook.github.io/react-native/
  - weex.apache.org/cn/guide/
- 桌面端 Electron 框架
  - electronjs.org/

/前端必备基础概念与实战 / *JavaScript* 技术体系 / 发展历史与标准化 / 标准化阶段 / *1997* 年 *6* 月 / *ECMA* 以 *JavaScript* 语言为基础制定了*ECMAS*

版本	发布日期	基于	Netscape Navigator	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.0	1996年3月		2.0		3.0			
1.1	1996年8月		3.0					
1.2	1997年6月		4.0 – 4.05			3		
1.3	1998年10月	ECMA-262 1st + 2nd edition	4.06 – 4.7x		4.0	5[21]		
1.4			Netscape Server			6		
1.5	2000年11月	ECMA-262 3rd edition	6.0	1.0	5.5（JScript 5.5） 6（JScript 5.6） 7（JScript 5.7） 8（JScript 5.8）	7.0	3.0–5	1.0 – 10.0.666
1.6	2005年11月	1.5 + Array extras + Array and string generics + E4X		1.5				
1.7	2006年10月	1.6 + Pythonic generators + Iterators + Let		2.0				28.0.1500.95
1.8	2008年6月	1.7 + Generator expressions + Expression closures		3.0		11.50		
1.8.1		1.8 + native JSON support + Minor updates		3.5				
1.8.2	2009年6月22日	1.8.1 + Minor updates		3.6				
1.8.5	2010年7月27日	1.8.2 + New features for ECMA-262 5th edition compliance		4.0				

版本	发表日期	与前版本的差异
1	1997年6月	首版
2	1998年6月	格式修正，以使得其形式与ISO/IEC16262国际标准一致
3	1999年12月	强大的正则表达式，更好的词法作用域链处理，新的控制指令，异常处理，错误定义更加明确，数据输出的格式化及其它改变
4	放弃	由于关于语言的复杂性出现分歧，第4版本被放弃，其中的部分成为了第5版本及Harmony的基础
5	2009年12月	新增“严格模式（strict mode）”，一个子集用作提供更彻底的错误检查,以避免结构出错。澄清了许多第3版本的模糊规范，并适应了与规范不一致的真实世界实现的行为。增加了部分新功能，如getters及setters，支持JSON以及在对象属性上更完整的反射[4][5][6][7][8]
6	2015年6月	ECMAScript 2015（ES2015），第 6 版，最早被称作是 ECMAScript 6（ES6），添加了类和模块的语法，其他特性包括迭代器，Python风格的生成器和生成器表达式，箭头函数，二进制数据，静态类型数组，集合（maps，sets 和 weak maps），promise，reflection 和 proxies。作为最早的 ECMAScript Harmony 版本，也被叫做ES6 Harmony。
7	2016年6月	ECMAScript 2016（ES2016），第 7 版，多个新的概念和语言特性[9]
8	2017年6月	ECMAScript 2017（ES2017），第 8 版，多个新的概念和语言特性[10]
9	2018年6月	ECMAScript 2018 （ES2018），第 9 版，包含了异步循环，生成器，新的正则表达式特性和 rest/spread 语法。

脚本引擎	参考应用程序	兼容性[11]			
		ES5[12]	ES6[13]	ES7[14]	较新（2016+）[14][15]
Chakra	Microsoft Edge	100%	96%	100%	54%
SpiderMonkey	Firefox	100%	98%	100%	77%
Chrome V8	Google Chrome、Opera	100%	98%	100%	93%
JavaScriptCore（Nitro）	Safari	97%	99%	100%	83%



As long as modules publish a **consistent** set of notifications, the underlying libraries used within these modules become less important. A module using Dojo that publishes notifications will be treated the same within the system as one which uses jQuery or YUI. This allows a switch later-on with less impact to the rest of the application.

```
// ./blog.js
//导出模块
exports.blog={
    say:function(name){
        return 'hello, '+name;
    }
}
```

```
// ./app.js
//加载模块
var blog=require('./blog').blog;
console.log(blog.say('zhaiqianfeng'));
```

执行输出

```
$ node app.js
hello, zhaiqianfeng
```

如定义AMD模块

```
// ./base.js
define(function(){
    return {
        say:function(name){
            return 'hello,'+name;
        }
    };
});
```

AMD调用模块也是使用关键字require，但不同于CommonJS，而是和AMD定义相似

```
require([module], callback);
```

因此调用base模块的代码如

```
// ./app.js
require(['base'],function(base){
    var words=base.say('zhaiqianfeng');
        document.write(words);
});
```

先引用AMD的实现RequireJS，html代码如下

```
<script src="require.js"></script>
<script src="app.js"></script>
```

/ 前端必备基础概念与实战 / *JavaScript* 技术体系 / 模块化 / *UMD (Universal Module Definition) / CommonJS* 侧重服务器，而 *AMD* 侧重于浏览器，

```
(function (root, factory) {
    if (typeof define === 'function' && define.amd) {
        // AMD方式
        define(['b'], function (b) {
            return (root.returnExportsGlobal = factory(b));
        });
    } else if (typeof module === 'object' && module.exports) {
        // Node/CommonJS方式
        module.exports = factory(require('b'));
    } else {
        // 公开暴露给全局
        root.returnExportsGlobal = factory(root.b);
    }
}(this, function (b) {
    return {};
}));
```

要异步输入一个模块，可以用如下语法：

JavaScript 代码：

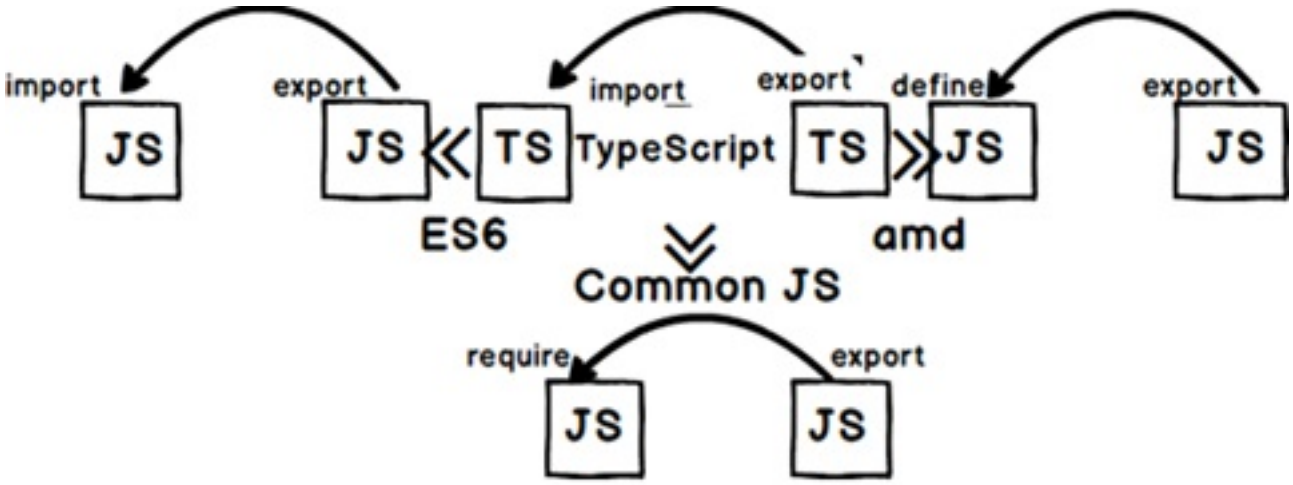
```
System.import('module-name');
```

然后我们可以用配置 API 来配置 SystemJS 的行为：

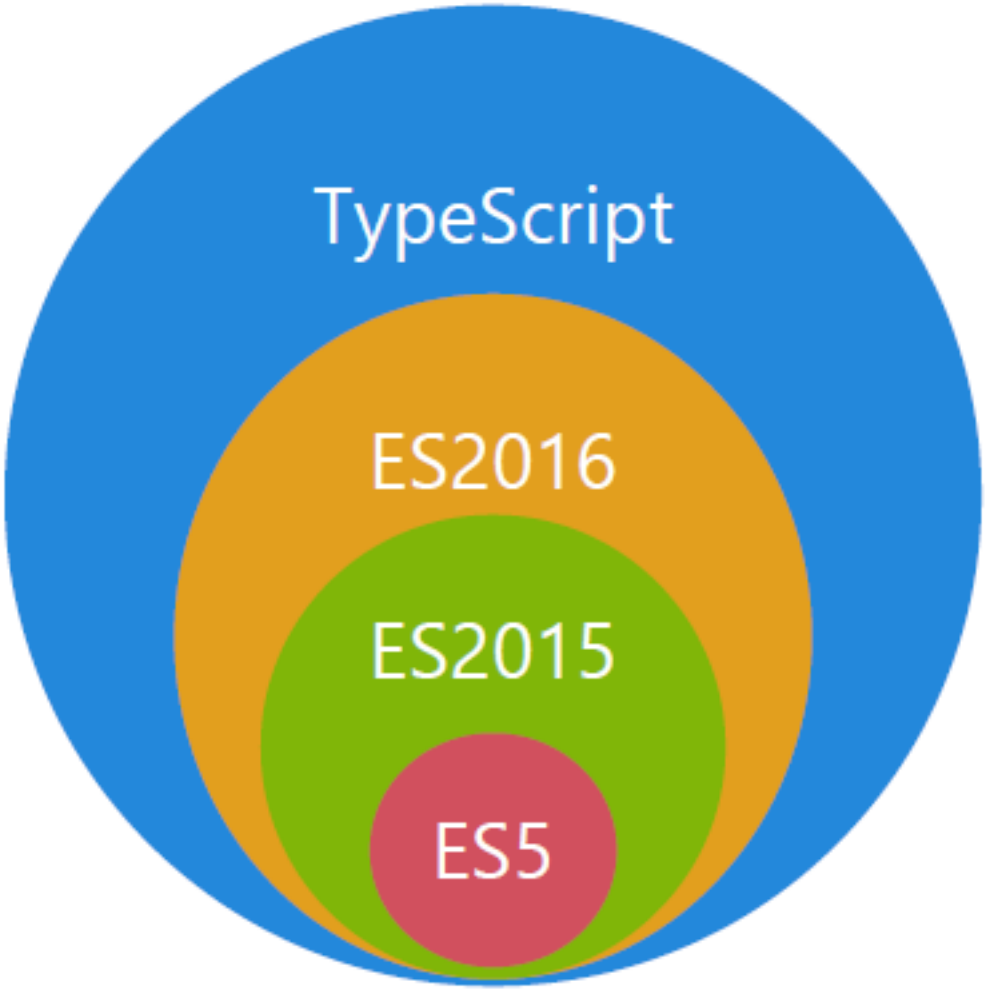
JavaScript 代码：

```
System.config({
  transplier: 'babel',
  baseURL: '/app'
});
```

/ 前端必备基础概念与实战 / *JavaScript* 技术体系 / 模块化 / 经典图片




Module
ES6
CommonJS
AMD
UMD
SystemJS
Target
ES3
ES5
ES6


TypeScript
ES2016
ES2015
ES5