# ES5

第1天课堂笔记

[ES5 1](#_Toc23603)

[复习 4](#_Toc4400)

[Less 4](#_Toc25313)

[Sass复习 7](#_Toc25350)

[Stylus复习 9](#_Toc27998)

[ES5 12](#_Toc11948)

[JSON对象 12](#_Toc1966)

[Parse 12](#_Toc11284)

[Stringify方法 13](#_Toc24078)

[数组拓展 13](#_Toc8786)

[判断数组 13](#_Toc24543)

[判断成员位置 14](#_Toc32086)

[填充数组 14](#_Toc777)

[下午复习 16](#_Toc30296)

[forEach方法 17](#_Toc29475)

[Map用来遍历数组的 17](#_Toc3276)

[Some 判断数组中成员 18](#_Toc4898)

[Every判断数组成员 18](#_Toc15325)

[Filter 过滤方法 19](#_Toc11627)

[累积方法 19](#_Toc30899)

[严格模式 20](#_Toc19826)

[变量 21](#_Toc10270)

[局部严格模式 21](#_Toc22350)

[函数参数 21](#_Toc28246)

[对象属性 21](#_Toc21734)

[关键字，保留字，特殊变量 22](#_Toc4588)

[Delete关键字 22](#_Toc28075)

[8进制与特殊字符 22](#_Toc13095)

[Eval 23](#_Toc25850)

[Arguments.callee 23](#_Toc25438)

[With关键字 23](#_Toc7312)

[Date对象 24](#_Toc17422)

[Bind方法 24](#_Toc17347)

[对象拓展 24](#_Toc21443)

[作业 24](#_Toc24922)

# 复习

## Less

使用三种方式

浏览器端

引入less文件，通过script标签

通过link标签引入less文件，type是text/less

命令行中使用

安装less，会生成lessc指令

Lessc demo.less 输出在控制台中

Lessc demo.less demo.css 输出在demo.css文件中

Lessc demo.less demo.css --clean-css 压缩css输出文件

工程化中使用

安装fis-parser-less

配置fis-conf入口文件

获取所有的less文件

引入插件parser ： ‘less’

更改后缀名 rExt: ‘css’

定义变量

语法 @name: value;

可以先使用在定义

定义混合

混合是用来复用样式的，因此混合是一个选择器

Id选择器

类选择器（常用的）

使用混合

混合选择器名称;

方法

跟混合一样都是用来复用样式的

定义方法跟定义混合很像，只不过多了一个()参数集合

混合在编译时候保留，方法在编译时候删除

使用方式

方法名称();

传递参数

定义方式跟变量定义方式是一样的 ，多个参数之间用,隔开

可以通过:为参数赋予默认值，有默认值时候可以不用传递参数

所有参数用 @arguments

可以通过！Important来提高方法的属性权重

& 可以在选择器内部访问父选择器

作用域(less, sass, stylus)

{}或者换行符（stylus）可以定义一个模块，一个模块就创建了一个作用域

1 变量 less，sass stylus

定义在全局作用域下的变量在任何作用域内都可以访问

定义在作用域内部的变量，只能在当前作用域使用，其他作用域使用不了

变量要定义在全局作用域下

2 混合 less sass stylus

定义全局作用域下的混合，在任何作用域下都可以使用

Less中定义在作用域内部的混合，外部使用必须通过命名空间使用

Sass中定义在作用域内部的混合，外部不能使用

Stylus中定义在作用域内部的混合，外部不能使用

3 方法 less stylus

定义全局作用域下的方法，在任何作用域下都可以使用

Less 定义在作用域内部的混合，外部使用必须通过命名空间使用

stylus定义在作用域内部的混合，外部不能使用

4 继承 sass stylus

定义全局作用域下的继承选择器，在任何作用域下都可以使用

Sass 定义在作用域内部的继承，外部可以直接使用

Stylus定义在作用域内部的继承，内部和外部都要使用命名空间

内置方法

数字方法

Ceil向上取整

Floor 向下取整

Round 四舍五入

建议带单位

Percentage 转化百分数

不能使用单位，返回值 参数\*100%

Max 获取最大值

Min 获取最小值

Less中可以传递任意个参数，stylus中只能传递两个参数

如果单位不统一会做单位的转化

色彩方法

创建方法

Rgb rgb模式创建方法

Rgba rgba模式创建的方法

Hsl hsl模式创建的方法

通道方法

Red 红色通道

Green 绿色通道

Blue 蓝色通道

Alpha 透明度通道

Hue 色相通道

Saturation 饱和度通道

Lightness 亮度通道

操作方法

Fadein 提高透明度

Fadeout 降低透明度

Fade 调节透明度到某个值

Saturate 调高饱和度

Desaturate 降低饱和度

Lighten 调高亮度

Darken 降低亮度

Greyscale 灰色

Mix 混合两种颜色

混合方法

Softlight 柔光

Hardlight 强光

字符串方法

Escape url编码

Replace 替换子字符串

原始字符串

匹配字符串（正则）

替换的字符串

E 避免编译

文件的导入

@import url(文件相对地址)

可以使用文件导入的变量混合方法（尽量不要再导入的文件中定义混合）

Stylus中@import 文件相对地址，如果有url表示css的导入

使用js表达式

创建一个js环境 ``

在js环境中我们可以使用js表达式

插值

语法 @{name}

处理属性名称

处理属性值可以用字符串的拼接+

条件语句

可以通过when关键字定义条件语句（用在方法里的）

一个when定义了一个条件

When （） and （） 定义多个且条件

When not () 定义多个非条件

可以使用js中的比较运算符

> 表示大于

< 表示小于

= 表示等于

## Sass复习

使用两种方式

命令行

安装sass：先安装ruby，再通过gem包管理工具安装sass

使用方式三种

Sass demo.scss 向控制台中输出结果

Sass demo.scss demo.css 向demo.css文件中输出结果

Sass dmeo.scss demo.css -C 删除缓存

工程化中

安装fis-parser-node-sass插件

配置fis-conf

获取所有scss文件

引入插件 parser: ‘node-sass’

更改后缀名称 rExt: ‘.css’

注释

单行注释 // 编译时候会被删除

多行注释 /\*\*/ 编译时候会保留

变量

$name: value;

模块内部的变量在外部无法使用

& 符号表示在选择器内部访问父选择器

变量使用就近原则

混合

定义 @mixin 混合名称() {}

使用混合 @include 混合名称();

如果没有参数可以省略括号

()表示参数集合

定义参数跟定义变量一样 $name:value,$name2:value,$name3

可以通过:定义默认参数

获取剩余参数

语法arg...

可以获取arg变量及其后面所有的变量

前面定义的变量可以正常使用

继承

Sass支持

Id选择器继承

类选择器继承

元素名称选择器继承

自定义元素名称选择器继承

继承在编译时候保留，混合在编译时候删除

使用继承 @extend 继承选择器名称

尽量不要嵌套继承

如果不想让继承选择器在编译的时候保留加上%

运算

数字运算

加法减法 会做单位转化

乘法 只能出现一个单位

除法只能被除数有单位（除数被除数都有单位，单位会抵消）

必须出现变量或者方法的返回值

必须是表达式一部分

必须使用()

色彩运算

对每个通道分别处理

如果值小于0 就是0

如果大于ff 就是ff

运算是透明度必须统一

字符串运算

字符串的拼接 +

插值

语法 #{$name}

处理属性名称的

语句

If条件语句

@if 条件 {}

@else if 条件 {}

@else {}

循环语句

For循环

@for $i from start to end

@for $i from start through end

To 不包含end

Through 包含end

While循环

定义循环变量初始值

Whlie 条件 {

// 更高循环条件

}

枚举循环

@each $i in 枚举数据

数据之间通过逗号隔开

三元表达式

If(condition, truevalue, falsevalue)

Condition表示条件

## Stylus复习

拓展名 .styl

通常在工程化中使用

插件fis-parser-stylus

配置fis-conf

获取stylus文件

引入插件

更改后缀名

模块通过换行符定义一个模块

变量

语法 变量名称 = 值

混合

定义 混合名称()

两类

兼容样式 通常是用来覆盖内置样式，而定义同一组样式

复用样式 通常不要覆盖内置样式

参数跟定义变量是一样的

参数名称 = 默认值

可以通过=定义默认值

可以通过arg...获取剩余参数

剩余参数前面的变量可以正常使用

方法

用途：处理属性值，提供的是一个功能，用来处理数据

定义跟混合很像，只不过返回值是数据

如果返回到的数据存在多个值

()包含多个值

Return 包含多个值

此时可以想访问数组一样方式这些值

继承

继承一组选择器样式

Id继承

类继承

元素名称继承

自定义元素名称继承

使用继承通过 @extend 继承选择器

如果在模块内部或者模块外部使用继承选择器，必须使用命名空间

如果不想让继承选择器在编译的时候保留加上$

内置方法

Max，min只能两个参数

Rgb方法参数可以通过red，green，blue自定义顺序

Image-size 获取图片的大小，返回值类似数组，

通过[0]访问图片宽度

通过[1]访问图片高度

插值

语法 {key}

处理属性名称

If语句

If ()

Else

枚举循环

For $key, $index in 枚举集合

枚举集合通过空格隔开

@import导入文件，

注意不要写url，如果写了将是一个css导入

@css 避免编译的，直接保留css原代码

Keyframes定义动画

在stylus中会自动加上浏览器前缀

内部的样式不会处理，想处理只能写混合

# ES5

Javascript最早由网景公司开发的，主要应用在浏览器中，这种语言很好用，被很多网站开发者使用，后来ie出现了，ie直接嵌入到window系统中，这样ie市场占有率大了，ie就开发了jscript语言来实现脚本。

早期javascript为让语言更流行，与sun公司合作（主要提供java支持），把语言名称改成javascript，雷锋与雷峰塔的关系。网景公司做了一件好事，将语言标准提交给ECMA组织，所以javascript核心语言标准都是ECMA标准。每年ECMA标准更新，各个浏览器都争先恐后的实现这些标准。实现最好的就是FF，webkit类的浏览器（chrome，Safari等）先实现一些新的功能，提交给ECMA组织，ECMA认为这些功能比较好就是作为标准，IE浏览器就比较滞后了

当初ECMA定义的规范IE往往不去实现，后来IE一些工程师不干了，认为ie6实现的功能太搓了，不去维护了（抛弃xp），开发新的浏览器IE9+（开发的完，实现的功能只是标准一部分）。更可气的是国内的一些互联网公司，例如360等公司维护xp系统，所以ie6在国内还是有很大的市场占有率，所以很多同学毕业招工作还是要兼容ie6。

好消息：移动端不要兼容IE6，直接使用高级样式以及方法

国内一些浏览器套的webkit内核，非要改成自己的内核

ES5

ES5绝大部分内容已经被高级浏览器支持了，所以在工作都是直接使用

# JSON对象

## Parse

解析json字符串转化成一个js对象

第一个参数表示字符串

第二个参数是一个回调函数，

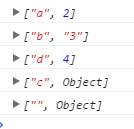
在回调函数中我们可以处理每一个数据

第一个参数表示key（属性名称）

第二个参数表示value（属性值）

返回值就是处理结果

解析原理，是从叶子节点到根节点解析，（由外向内解析的）



|  |
| --- |
| 1. var obj = JSON.parse(str, function (key, value) { 2. // console.log(arguments) 3. // console.log(key, value) 4. // 如果值是字符串类型，我们转化成数字 5. if (typeof value === 'string') { 6. return +value; 7. } 8. // return 111 9. // 返回value 10. return value 11. }); |

JSON对象

## Stringify方法

将js对象转化成一个json字符串

第一个参数表示js对象

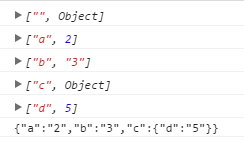
第二个参数表示回调函数

第一个参数表示key（属性名称）

第二个参数表示value（属性值）

返回值就是处理的结果

解析原理：从根节点遍历到叶子节点，从内向外处理的



|  |
| --- |
| 1. var result = JSON.stringify(obj, function (key, value) { 2. console.log(arguments) 3. // 将数字转换成字符串 4. if (typeof value === 'number') { 5. return value + ''; 6. } 7. return value; 8. }) |

# 数组拓展

## 判断数组

数组在js中被看作对象

工作中数组类型是很重要的

ES5为数组拓展一个方法判断数组

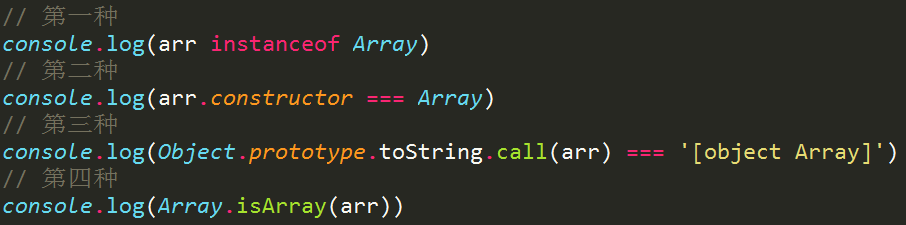
isArray，弥补数组本身不能判断数组的

Instanceof 判断是数组的实例化对象

Constructor 判断构造函数的

Object.prototype.toString.call 将对象的转化成字符串的

这种方法都可以判断是否是数组，并且可以区分类数组



|  |
| --- |
| 1. function demo () { 2. console.log(arguments instanceof Array) 3. // 第二种 4. console.log(arguments.constructor === Array) 5. // 第三种 6. console.log(Object.prototype.toString.call(arguments) === '[object Array]') 7. // 第四种 8. console.log(Array.isArray(arguments)) 9. } 10. demo() |

## 判断成员位置

indexOf和lastIndexOf用来判断数组的成员索引值的

如果没有匹配的返回-1

如果有匹配就返回索引值（从0开始计数）

是一个非贪婪搜索，匹配到就不再去匹配了



|  |
| --- |
| 1. /\*\* 2. \* 我们实现indexOf方法 3. \* @arr 原数组 4. \* @value 遍历的成员 5. \* return 索引值 6. \*\*/ 7. function icktIndexOf(arr, value) { 8. // 遍历arr 9. for (var i = 0; i < arr.length; i++) { 10. // 判断成员值是否与当前值全等 11. if (arr[i] === value) { 12. // 返回索引值 13. return i 14. } 15. } 16. // 如果没有获取到就返回-1 17. return -1; 18. } |

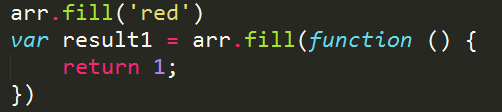
## 填充数组

Fill 向数组每个位置填充一个数据（fill参数）

参数不会执行，原封不动的添加到该位置上

参数就是传递数据

返回值就是数组



|  |
| --- |
| 1. function fill (arr, value) { 2. // 遍历数组 3. for (var i = 0; i < arr.length; i++) { 4. // 替换原位置的数据 5. arr[i] = value 6. } 7. // 返回原数组 8. return arr; 9. } 10. Array.prototype.icktFill = function (value) { 11. // 访问数组。可以通过this 12. // console.log(this.length) 13. // 遍历数组 14. for (var i = 0; i < this.length; i++) { 15. // 给当前对象填充数据 16. this[i] = value 17. } 18. // 我们要返回原数组 19. return this; 20. } |

## 下午复习

ES5

Json对象

Parse 解析json字符串到js对象

第一个参数表示原始字符串

第二个参数表示回调函数

Key属性名称

Value属性值

返回值是处理后的value值

有外到内，有叶子节点到根节点

Stringify 将对象转化成json字符串

第一个参数表示对象

第二个参数表示回调函数

Key 对象属性名称

Value 属性值

返回值是处理后的value值

有内到外，由根节点到叶子节点

判断数组

isArray ES5新增判断数组方法

Instanceof 判断是否是数组实例化对象

Constructor 判断构造函数

Object.prototype.toString.call 判断对象的类型

获取数组中成员位置

indexOf

lastIndexOf

参数都表示成员

返回值表示索引值，

如果成员不存在返回-1

Fill方法

向数组中每个位置添加成员

参数就是添加的数组

返回值就是数组

## forEach方法

数组迭代方法，解决for循环中作用域的问题的

forEach是Es5新增的方法

参数是一个函数

第一个参数表示value值

第二个参数表示索引值

第三个参数表示原数组

返回值对forEach方法执行的结果无影响

作用域是全局作用域

返回值undefined

|  |
| --- |
| 1. /\*\* 2. \* 定义Each方法遍历数组 3. \* @arr 表示原数组 4. \* @fn 回调函数 5. \*\*/ 6. function each(arr, fn) { 7. // 遍历数组 8. for (var i = 0; i < arr.length; i++) { 9. // 用回调函数处理每一个成员 10. // fn(arr[i], i, arr) 11. // 如果想让fn执行时作用域是arr，只需要更改作用域就可以了 12. fn.call(arr, arr[i], i, arr) 13. } 14. } |

## Map用来遍历数组的

方法的返回值是一个新数组，每个成员有回调函数的返回值决定

参数就是回调函数

要定义返回值

第一个参数表示value

第二个参数表示index

第三个参数表示原数组

作用域是全局作用域

|  |
| --- |
| 1. /\*\* 2. \* 实现map方法 3. \* @arr 原数组 4. \* @fn 回调函数 5. \* return 新数组 6. \*\*/ 7. function map (arr, fn) { 8. // 将每一次执行的结果保存，并返回 9. var result = []; // 返回结果的容器 10. // 遍历数组并执行每一次循环 11. for(var i = 0; i < arr.length; i++) { 12. // 参数：value， index， arr 13. result.push(fn.call(arr, arr[i], i, arr)) 14. // result[i] = fn(arr[i], i, arr) 15. } 16. // 循环执行之后，我们将结果返回 17. return result; 18. } |

断言方法

## Some 判断数组中成员

至少有一个满足条件返回true

一个都不满足条件返回false

参数就是回调函数

第一个参数表示value

第二个参数表示index

第三个参数表示原数组

作用域全局作用域

返回值就是判断条件

Some方法返回值就是一个布尔值，非贪婪判断

|  |
| --- |
| 1. /\*\* 2. \* 实现some方法 3. \* @arr 原数组 4. \* @fn 回调函数 5. \* return 布尔值 6. \*\*\*/ 7. function some (arr, fn) { 8. // 遍历数组 9. for (var i = 0; i < arr.length; i++) { 10. // 根据回调函数执行结果来判断 11. if (fn(arr[i], i, arr)) { 12. return true 13. } 14. } 15. // 如果所有成员都不满足 16. return false; 17. } |

## Every判断数组成员

如果都满足条件返回true

有一个不满足返回false

参数就是回调函数

第一个参数表示value

第二个参数表示index

第三个参数表示原数组

作用域全局作用域

返回值就是判断条件

every方法返回值就是一个布尔值，非贪婪判断

|  |
| --- |
| 1. function every (arr, fn) { 2. // 对每一个成员执行一次回调函数，如果结果全部是真返回true 3. // 如果有一个不满足条件就返回false 4. for (var i = 0; i < arr.length; i++) { 5. // 如果回到函数执行结果是false，就直接返回false 6. if (!fn(arr[i], i, arr)) { 7. return false 8. } 9. } 10. // 如果全部满足条件， 返回true 11. return true; 12. } |

## Filter 过滤方法

将数组中符合条件的成员保存在一个数组中并返回

参数也是一个回调函数

第一个参数表示value

第二个参数表示index

第三个参数表示原数组

作用域全局作用域

返回值就是判断条件

Filter方法返回式是一个新数组

|  |
| --- |
| 1. /\*\* 2. \* 实现filter方法 3. \* @arr 原数组 4. \* @fn 回调函数 5. \* return 返回值 6. \*\*/ 7. function filter(arr, fn) { 8. // 循环数组，寻找符号条件的成员，并保存在一个新数组中 9. // 定义一个数组容器 10. var result = []; 11. for (var i = 0; i < arr.length; i++) { 12. // 执行函数寻找成员 13. if (fn(arr[i], i, arr)) { 14. // 将值保存在result中 15. result.push(arr[i]) 16. } 17. } 18. // 循环结束，将结果返回 19. return result; 20. } |

## 累积方法

Reduce与reduceRight

对数组执行累积结果

参数是一个回调函数

第一个参数表示上一次执行时累积的结果

第二个参数表示成员值value

第三个参数表示成员索引值index

第四个参数表示原数组

作用域全局作用域

遍历是从第二次开始

方法的返回值是累积的结果

Reduce是从前向后累积的

reduceRight是从后向前累积

|  |
| --- |
| 1. /\*\* 2. \* 数组累积方法 3. \* @arr 原数组 4. \* @fn 回调函数 5. \* return 累积的结果 6. \*\*\*/ 7. function reduce(arr, fn) { 8. // 创建结果，执行循环，每一将循环的结果传递到下一次循环中，一次次累积，最终将结果返回 9. // 第一次执行时累积的结果是第一个成员 10. var result = arr[0]; // 定义返回结果 11. // 循环是从第二个成员开始 12. for (var i = 1; i < arr.length; i++) { 13. // 执行fn得到结果,一次一次累积 14. result = fn(result, arr[i], i, arr) 15. } 16. // 最终将结果返回 17. return result; 18. } |

# 严格模式

是ES5新增的一个模式，

在以前的模式中，我们书写代码有一些问题，但是js无法帮我们捕获这些问题

A=12 结果创建了一个全局变量

执行eval方法污染全局作用域

例如使用with或者arguments.callee无法进行代码优化

原有的模式有太多太多的问题，而js无法捕获，

所以ES5新增严格模式，这样可以避免这些问题的产生，优化我们代码，是代码执行效率提高

ECMA鼓励我们使用严格模式

没有副作用，只需要在页面中插入一个行字符串‘use strict’

支持严格模式的浏览器捕获这个字符串自动切换到严格模式

## 变量

严格模式

对变量定义的约束

在严格模式下，定义变量如果没有使用var会抛出错误

Uncaught ReferenceError: color is not defined

|  |
| --- |
| 1. color = 'green' 2. console.log(color) |

## 局部严格模式

Es5允许我们使用局部的严格模式

在函数体内使用严格模式是局部的严格模式，此时函数体内将转化一个严格模式，函数体外部就是正常模式

严格模式对作用域的改变

正常模式下函数内部作用域是全局作用域

严格模式下函数内部的作用域是undefined

|  |
| --- |
| 1. function demo () { 2. 'use strict' 3. // 函数体内是正常模式 4. // color = 'red'; 5. // 严格模式下作用域是undefined 6. console.log(this) 7. this.color = 'blue' 8. } 9. demo() |

## 函数参数

严格模式对函数参数的要求

正常模式下，在函数参数中我们可以定义相同名称参数的

在严格模式下，定义相同名称的参数会报错

Uncaught SyntaxError: Duplicate parameter name not allowed in this context

|  |
| --- |
| 1. function demo(param1, param1) { 2. console.log(arguments) 3. console.log(param1) 4. } 5. // 调用方法 6. demo(2, 3) |

## 对象属性

严格模式下

通过对象字面量方式定义对象，是不允许定义相同属性的（会有覆盖问题）

严格模式下不允许定义相同属性名称

|  |
| --- |
| 1. var obj = { 2. color: 'red', 3. // 相同名称的属性，后面会覆盖前面的 4. color: 'green' 5. } |

## 关键字，保留字，特殊变量

正常模式下可以使用这些定义变量

严格模式下

在严格模式下不允许使用关键字，保留字，特殊变量定义变量的

保留字 Uncaught SyntaxError: Unexpected strict mode reserved word

特殊变量 Uncaught SyntaxError: Unexpected eval or arguments in strict mode

|  |
| --- |
| 1. var public = 'red' 2. console.log(public) 3. function demo() { 4. // var arguments = 123 5. console.log(arguments) 6. } 7. // 调用demo 8. demo(4, 5, 6) 9. var eval = 'hello' 10. console.log(eval) |

## Delete关键字

删除对象属性

正常模式下用delete删除变量，方法是失效的，但是无法检测

严格模式下删除变量，方法会抛出错误

Uncaught SyntaxError: Delete of an unqualified identifier in strict mode.

|  |
| --- |
| 1. var a = 123; 2. // delete a; 3. console.log(a) 4. // 定义函数 5. function demo () {} 6. // delete demo; 7. console.log(demo) 8. // 删除属性 9. var obj = { 10. color: 'red' 11. } 12. delete obj.color; 13. console.log(obj) |

## 8进制与特殊字符

正常模式下可以使用8进制，可以使用特殊字符

严格模式下不允许使用8进制，以及特殊字符（可以是使用16进制）

Uncaught SyntaxError: Octal literals are not allowed in strict mode.

|  |
| --- |
| 1. var num = 012; 2. console.log(num) 3. var str = '\012' 4. console.log('1' + str + '2') |

## Eval

正常模式下，可以使用，并且会在全局作用域下创建变量

严格模式下使用eval不会污染作用域

在严格模式下还想使用正常模式下的eval，我们可以将eval保存一个变量中

所以严格模式下对eval约束是检测这个名称而不是屏蔽了这个功能

|  |
| --- |
| 1. var myEval = eval; 2. myEval('var b = "green"') 3. console.log(b) |

## Arguments.callee

正常模式下为了解决函数功能与函数名称耦合的问题，我们可以通过arguments.callee解决

但是在严格模式下不允许使用，会抛出错误

Uncaught TypeError: 'caller', 'callee', and 'arguments' properties may not be accessed on strict mode functions or the arguments objects for calls to them

|  |
| --- |
| 1. function addNum(num) { 2. // 如果num是1直接返回，不要再迭代了 3. if (num <= 1) { 4. return num; 5. } 6. // 调用下一个num相加 7. // 这里功能与方法的名称耦合了，所以方法名称永远不能修改 8. // return num + addNum(num - 1) 9. // 为了解决这个问题，提供了arguments.callee方法 10. return num + arguments.callee(num - 1) 11. } |

## With关键字

更改作用域的

正常模式下我们可以通过with来更改作用域，方便调用某个对象上的方法

严格模式下不允许

Uncaught SyntaxError: Strict mode code may not include a with statement

|  |
| --- |
| 1. var a = 4; 2. var b = 5; 3. var obj = {c: 6} 4. with(obj) { 5. a = a + b 6. } |

# Date对象

新增一个toJSON方法，转化一个可读性强的日期格式

|  |
| --- |
|  |

# Bind方法

ES5对Function对象拓展的一个方法，用来在函数执行时更改作用域

Call和apply函数定义的时候，就立即执行并更改作用域

|  |
| --- |
| 1. /\*\* 2. \* 实现函数绑定 3. \* @fn 表示执行的函数 4. \* @obj 表示绑定的作用域 5. \* return 绑定作用域后的方法 6. \*\*/ 7. function bind(fn, obj) { 8. // 我们调用bind时候可以传递其他参数。如1,2，我们要将他们获取 9. // 通过arguments获取 10. var args = [].slice.call(arguments, 2) 11. // 返回一个方法 12. return function () { 13. // 获取参数，并和args放在一起 14. var params = args.concat([].slice.call(arguments, 0)) 15. // 当执行返回的方法的时候，我们要执行fn 16. // 将参数传递进来 17. fn.apply(obj, params) 18. } 19. } |

# 对象拓展

Create方法实现继承（继承参数对象）实现原理是寄生式继承，（寄生工厂方法）

|  |
| --- |
| 1. /\*\* 2. \* 继承工厂方法 3. \* @obj 被继承的对象 4. \*\*/ 5. function create(obj) { 6. // 创建寄生类 7. var F = function () {}; 8. // 原型继承 9. F.prototype = obj; 10. // 实例化寄生类 11. return new F() 12. } |

# 作业

回去安装fis3-parser-typescript插件

|  |
| --- |
|  |