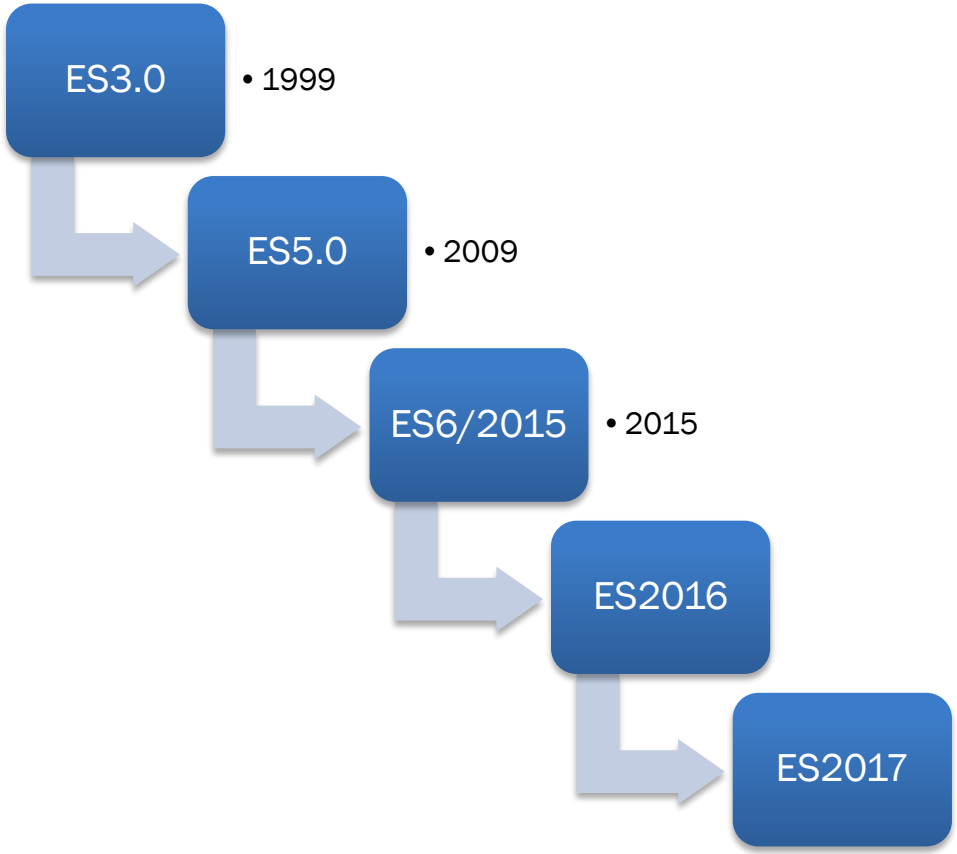
JS的核心动力

从ES5到ES2017，推动前端进程的那些核心API

乔梁

2018-06


ES3.0
• 1999
ES5.0
• 2009
ES6/2015
• 2015
ES2016
ES2017

苏宁易购 suning.com

Contents

一、面向对象

二、异步

	Java	ES3	ES6
类	Class	Function	class
对象	new Class	new Function	new class
继承	extends、implements	property	extends
多态	private、protected、public	this	this\proxy\Reflect

原型链是ES中实现面向对象的基础，后续的所有特性包括class，都是它的语法糖

▶ prototype和__proto__的区别

* prototype是函数才有的属性

* __proto__是每个对象都有的属性

* 但__proto__不是一个规范属性，只是部分浏览器实现了此属性，对应的标准属性是[[Prototype]]

注：大多数情况下，__proto__可以理解为“**构造器的原型**”，即：

__proto__ === constructor.prototype

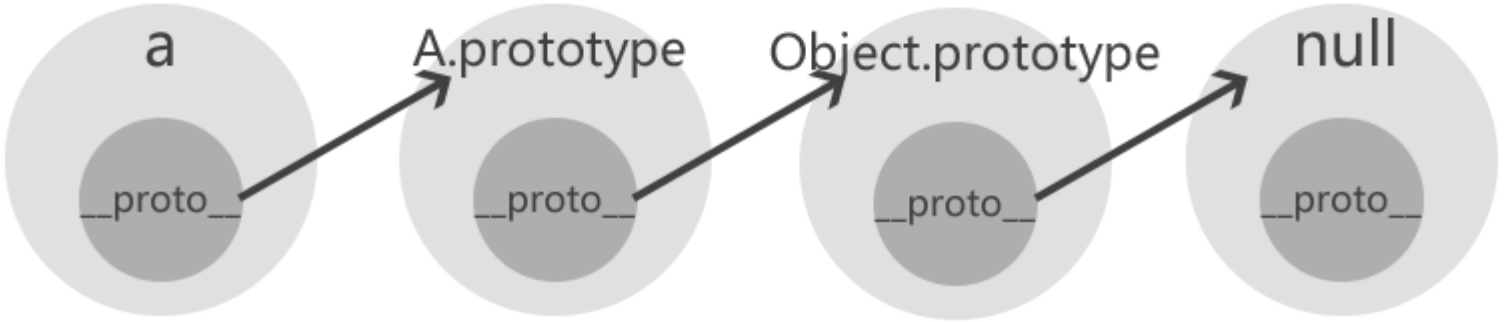
（通过Object.create()创建的对象不适用此等式，图2有说明）

▶ 什么是原型链？

由于__proto__是任何对象都有的属性，而js里万物皆对象，所以会形成一条__proto__连起来的链条，递归访问__proto__必须最终到头，并且值是 null。

当js引擎查找对象的属性时，先查找对象本身是否存在该属性，如果不存在，会在原型链上查找，但不会查找自身的prototype

```
var A = function(){};
var a = new A();
```



《JavaScript高级程序设计》

- 继承
  - 借用构造函数
  - 组合继承
  - 原型式继承
  - 寄生式继承
  - 寄生组合式继承

angularjs $scope

```
16107        if (!this.$$ChildScope) {
16108          this.$$ChildScope = createChildScopeClass(this);
16109        }
16110        child = new this.$$ChildScope();
16111      }
```

```
15944  function createChildScopeClass(parent) {
15945    function ChildScope() {
15946      this.$$watchers = this.$$nextSibling =
15947          this.$$childHead = this.$$childTail = null;
15948      this.$$listeners = {};
15949      this.$$listenerCount = {};
15950      this.$$watchersCount = 0;
15951      this.$id = nextUid();
15952      this.$$ChildScope = null;
15953    }
15954    ChildScope.prototype = parent;
15955    return ChildScope;
15956  }
```

```
15203         return function(subclass, superclass, overrides) {
15204
15205             if (Ext.isObject(superclass)) {
15206                 overrides = superclass;
15207                 superclass = subclass;
15208                 subclass = overrides.constructor !== objectConstructor ? overrides.constructor : function() {
15209                     superclass.apply(this, arguments);
15210                 };
15211             }
15212
15213             if (!superclass) {
15214                 Ext.Error.raise({
15215                     sourceClass: 'Ext',
15216                     sourceMethod: 'extend',
15217                     msg: 'Attempting to extend from a class which has not been loaded on the page.'
15218                 });
15219             }
15220
15221
15222             var F = function() {},
15223                 subclassProto, superclassProto = superclass.prototype;
15224
15225             F.prototype = superclassProto;
15226             subclassProto = subclass.prototype = new F();
15227             subclassProto.constructor = subclass;
15228             subclass.superclass = superclassProto;
15229
15230             if (superclassProto.constructor === objectConstructor) {
15231                 superclassProto.constructor = superclass;
15232             }
15233
15234             subclass.override = function(overrides) {
15235                 Ext.override(subclass, overrides);
15236             };
15237
15238             subclassProto.override = inlineOverrides;
15239             subclassProto.proto = subclassProto;
15240
15241             subclass.override(overrides);
15242             subclass.extend = function(o) {
15243                 return Ext.extend(subclass, o);
15244             };
15245
15246             return subclass;
15247         };
15248     }()),
```

面向对象：ES5：Object扩展

ES5为Object对象提供了一系列的静态方法：

- create
- defineProperty
- freeze

`Object.create()`方法创建一个新对象，使用现有的对象来提供新创建的对象的__proto__。（请查看浏览器控制台以获取视觉证据。）

JavaScript Demo: Object.create()

```
1  const person = {
2    isHuman: false,
3    printIntroduction: function () {
4      console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`)
5    }
6  };
7
8  const me = Object.create(person);
9
10 me.name = "Matthew"; // "name" is a property set on "me", but not on "p
11 me.isHuman = true; // inherited properties can be overwritten
12
13
```

面向对象：ES5：Object扩展

SUNING 苏宁

```
Object.defineProperties(obj, props)
```

参数

obj
在其上定义或修改属性的对象。

props
要定义其可枚举属性或修改的属性描述符的对象。对象中存在的属性描述符主要有两种：数据描述符和访问器描述符（更多详情，请参阅`Object.defineProperty()`）。描述符具有以下键：

configurable
`true` 当且仅当该属性描述符的类型可以被改变并且该属性可以从对应对象中删除。
默认为 **false**

enumerable
`true` 当且仅当在枚举相应对象上的属性时该属性显现。
默认为 **false**

value
与属性关联的值。可以是任何有效的JavaScript值（数字，对象，函数等）。
默认为 **undefined**.

writable
`true`当且仅当与该属性相关联的值可以用assignment operator改变时。
默认为 **false**

面向对象：ES5：Object扩展

`Object.freeze()` 方法可以冻结一个对象，冻结指的是不能向这个对象添加新的属性，不能修改其已有属性的值，不能删除已有属性，以及不能修改该对象已有属性的可枚举性、可配置性、可写性。也就是说，这个对象永远是不可变的。该方法返回被冻结的对象。

JavaScript Demo: Object.freeze()

```
1  const object1 = {
2    property1: 42
3  };
4
5  const object2 = Object.freeze(object1);
6
7  object2.property1 = 33;
8  // Throws an error in strict mode
9
10 console.log(object2.property1);
11 // expected output: 42
12
```

面向对象：ES5：Object扩展

Vue 利用Object.defineProperty() 观察data属性变化，进行数据绑定

```
4282    initState(vm);
```

```
3121    if (opts.data) {
3122      initData(vm);
3123    } else {
3124      observe(vm._data = {}, true /* asRootData */);
3125    }
```

```
927   function observe (value, asRootData) {
941     ob = new Observer(value);
```

```
861   var Observer = function Observer (value) {
873     this.walk(value);
884   for (var i = 0; i < keys.length; i++) {
885     defineReactive$$1(obj, keys[i], obj[keys[i]]);
886   }
```

```
970   var childOb = !shallow && observe(val);
971   Object.defineProperty(obj, key, {
972     enumerable: true,
973     configurable: true,
974     get: function reactiveGetter () {
975       var value = getter ? getter.call(obj) : val;
976       if (Dep.target) {
977         dep.depend();
978         if (childOb) {
979           childOb.dep.depend();
980           if (Array.isArray(value)) {
981             dependArray(value);
982           }
983         }
984       }
985       return value
986     },
987     set: function reactiveSetter (newVal) {
988       var value = getter ? getter.call(obj) : val;
989       /* eslint-disable no-self-compare */
990       if (newVal === value || (newVal !== newVal && value !== value)) {
991         return
992       }
993       /* eslint-enable no-self-compare */
994       if ("development" !== 'production' && customSetter) {
995         customSetter();
996       }
997       if (setter) {
998         setter.call(obj, newVal);
999       } else {
1000        val = newVal;
1001      }
1002      childOb = !shallow && observe(newVal);
1003      dep.notify();
1004    }
1005  });
1006 }
```

面向对象：ES6：Class

class可以看作是一个语法糖，让对象原型的写法更加清晰。

```
//定义类
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  toString() {
    return '(' + this.x + ', ' + this.y + ')';
  }
}
```

```
class ColorPoint extends Point {
  constructor(x, y, color) {
    super(x, y); // 调用父类的constructor(x, y)
    this.color = color;
  }

  toString() {
    return this.color + ' ' + super.toString(); // 调用父类的toString()
  }
}
```

ajax(XMLHttpRequest)

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    …
}

xhr.open(‘ get’ , url, true);
xhr.send(data);
```

ajax(XMLHttpRequest)

```
var xhr = new XMLHttpRequest();
xhr.onload = function() {
    …
}

xhr.onprogress = function() {
    …
}

xhr.open(‘ get’ , url, true);
xhr.send(data);
```

```
var data;

$.ajax(url, {
    success: function(result) {
        data = result.data;
        … // 获取数据，进行后续处理。
    }
});
```

```
var data1, data2;

setTimeout(function() {
    $.ajax(url, {
        success: function(result) {
            data1 = result.data;
        }
    });
}, 2000);

setTimeout(function() {
    $.ajax(url, {
        success: function(result) {
            data2 = result.data;
        }
    });
}, 2000);
```

异步：事件循环

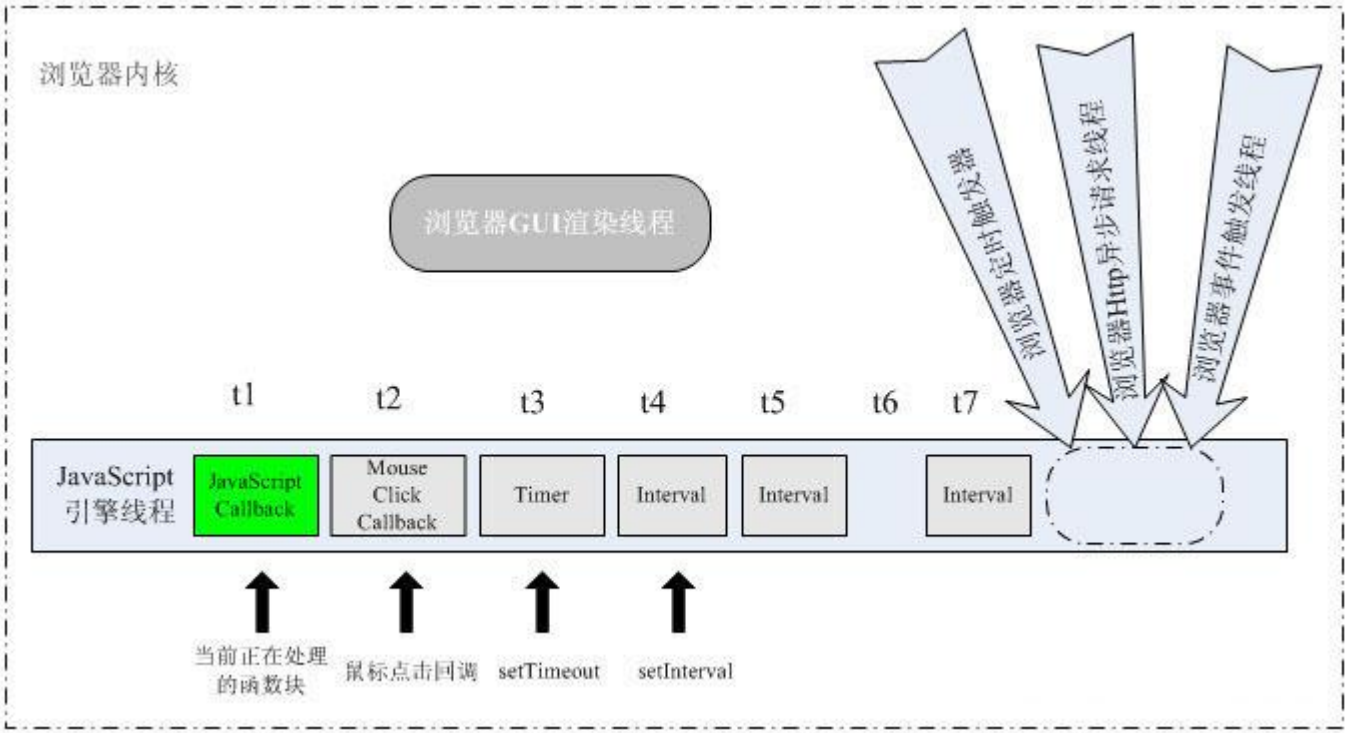
浏览器的内核是多线程的，在内核控制下各线程相互配合以保持同步，一个浏览器通常由一下线程组成：

- GUI 渲染线程
- JavaScript引擎线程
- 事件触发线程
- 异步http请求线程
- EventLoop轮询的处理线程

这些线程的作用：

- UI线程用于渲染页面
- js线程用于执行js任务
- 浏览器事件触发线程用于控制交互，响应用户
- http线程用于处理请求，ajax是委托给浏览器新开一个http线程
- EventLoop处理线程用于轮询消息队列

异步：事件循环




```
setTimeout(()=>{console.log("我才是第一");},0);
console.log("我是第一");
```

异步：callback


```
// 第一步
$.ajax(url, {
    success: function() {
        // 第二步
        $.ajax(url, {
            success: function() {
                // 第三步
                ....
            }
        })
    }
})
```

```

// 第一步
$.ajax(url, {
    success: function() {...}
});

// 第二步
$.ajax(url, {
    success: function() {...}
});

// 第三步
$.ajax(url, {
    success: function() {...}
});
```

异步：Promise

SUNING 苏宁

```
const promise = new Promise((resolve, reject) => {
    // 异步操作处理
    ...

    // 处理结果
    if ( success ) {
        resolve(success);
    } else {
        reject(reason);
    }
});

promise.then(
    // 正确处理
    success => {
    },
    // 异常处理
    reason => {

    }
);
```

异步：Promise


```
// 第一步
$.ajax(url, {
    success: function() {...}
});

// 第二步
$.ajax(url, {
    success: function() {...}
});

// 第三步
$.ajax(url, {
    success: function() {...}
});
```

```
promiseAjax(url).then(
    data => {

    },
    reason => {

    },
);
```

```
function promiseAjax(url) {
    return new Promise((resolve, reject) => {
        $.ajax(url, {
            success: function(data) { resolve(data) },
            fail: function(reason) { reject(reason) },
        });
    });
}
```

异步：promise

SUNING 苏宁

```
> $.ajax('/', { success: function(data) {console.info(data)} })
◂ ▶{readyState: 1, getResponseHeader: ƒ, getAllResponseHeaders: ƒ, setRequestHeader: ƒ, overrideMimeType: ƒ, …}
  <!DOCTYPE html>
  <html class=""><!--STATUS OK--><head><meta name="referrer" content="always" /><meta charset='utf-8' /><meta name=
  width,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no"/><meta http-equiv="x-dns-prefetch-control" content="c
  href="//m.baidu.com"/><link rel="shortcut icon" href="https://gss0.bdstatic.com/5bd1bjqh_Q23odCf/static/wiseindex
  <link rel="apple-touch-icon-precomposed" href="https://gss0.bdstatic.com/5bd1bjqh_Q23odCf/static/wiseindex/img/sc
  detection" content="telephone=no"/><noscript><style type="text/css">#page{display:none;}</style><meta http-equiv=
```

```
> function pAjax(url) {
      return new Promise((resolve, reject) => {
          $.ajax(url, {
              success: function(data) { resolve(data) },
              fail: function(reason) { reject(reason) },
          });
      });
  }
◂ undefined
> pAjax('/').then(data => console.info(data))
◂ ▶Promise {<pending>}
  <!DOCTYPE html>
  <html class=""><!--STATUS OK--><head><meta name="referrer" content="always" /><meta charset='utf-8' /><m
  width,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no"/><meta http-equiv="x-dns-prefetch-control" c
  href="//m.baidu.com"/><link rel="shortcut icon" href="https://gss0.bdstatic.com/5bd1bjqh_Q23odCf/static/
  <link rel="apple-touch-icon-precomposed" href="https://gss0.bdstatic.com/5bd1bjqh_Q23odCf/static/wiseind
```

异步：promise.then

then方法返回的是一个新的Promise实例（注意，不是原来那个Promise实例）。因此可以采用链式写法，即then方法后面再调用另一个then方法。

```
// 第一步
promiseAjax(url).then(data => {
    // 对data做处理
    return promiseAjax(url2)
})
// 第二步
.then(data2 => {
    // 对data2做处理
    return promiseAjax(url3)
})
// 第三步
.then(data3 => {
    // 对data3做处理
    return;
});
```

```

// 第一步
$.ajax(url, {
    success: function() {...}
});

// 第二步
$.ajax(url, {
    success: function() {...}
});

// 第三步
$.ajax(url, {
    success: function() {...}
});
```

异步：promise

SUNING 苏宁

```
> promiseAjax('/')
  .then(data => { console.info('first' + data); return promiseAjax('/') } )
  .then(data => { console.info('second' + data); return promiseAjax('/') })
  .then(data => console.info('last' + data));
< ▶ Promise {<pending>}
first<!DOCTYPE html>
<html class=""><!--STATUS OK--><head><meta name="referrer" content="always" /><
width,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no"/><meta http-equiv="
second<!DOCTYPE html>
<html class=""><!--STATUS OK--><head><meta name="referrer" content="always" /><meta chars
width,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no"/><meta http-equiv="x-dns-pre
href="//m.baidu.com"/><link rel="shortcut icon" href="https://gss0.bdstatic.com/5bd1bjqh_
<link rel="apple-touch-icon-precomposed" href="https://gss0.bdstatic.com/5bd1bjqh_Q23odC
detection" content="telephone=no"/><noscript><style type="text/css">#page{display:none;}
last<!DOCTYPE html>
<html class=""><!--STATUS OK--><head><meta name="referrer" content="always" /><meta charset=
width,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no"/><meta http-equiv="x-dns-prefetc
href="//m.baidu.com"/><link rel="shortcut icon" href="https://gss0.bdstatic.com/5bd1bjqh_Q23
<link rel="apple-touch-icon-precomposed" href="https://gss0.bdstatic.com/5bd1bjqh_Q23odCf/st
detection" content="telephone=no"/><noscript><style type="text/css">#page{display:none;}</st
URL=http://m.baidu.com/?cip=180.102.207.106&amp;amp;baiduid=B7493220E2B6EC622EFA999F7A8BC8EA
from=844b&amp;amp;vit=fps&amp;amp;index=&amp;amp;ssid=0&amp;amp;bd_page_type=1&amp;amp;logid
/></noscript><title>百度一下</title><script>window. performanceTimings=[['firstLine',+new Dat
```

异步：generator

SUNING 苏宁

```
> function* gen(x) {
    var y = yield x+1;
    return y;
  }
< undefined
> const g = gen(1);
< undefined
> g.next();
< ▶ {value: 2, done: false}
> g.next();
< ▶ {value: undefined, done: true}
> g.next();
< ▶ {value: undefined, done: true}
>
```

```
> function* gen() {
      var result = yield promiseAjax('/');
      console.info('first' + result);
      var result2 = yield promiseAjax('/');
      console.info('second' + result);
      var result2 = yield promiseAjax('/');
      console.info('last' + result);
  }
```

```
> gg = gen();
< ▶ gen {<suspended>}
> gg.next();
< ▶ {value: Promise, done: false}
> gg.next();
  firstundefined
< ▶ {value: Promise, done: false}
> gg.next();
  secondundefined
< ▶ {value: Promise, done: false}
> gg.next();
  lastundefined
< ▶ {value: undefined, done: true}
>
```

异步：generator 执行器

SUNING 苏宁

```
> function* gen() {
      var result = yield promiseAjax('/');
      console.info('first' + result);
      var result2 = yield promiseAjax('/');
      console.info('second' + result);
      var result2 = yield promiseAjax('/');
      console.info('last' + result);
  }
```

```
> var run = gen();
  run.next().value
  .then(data => run.next(data).value)
  .then(data => run.next(data).value)
  .then(data => run.next(data).value)
```

```
> var run = gen();
  run.next().value
  .then(data => run.next(data).value)
  .then(data => run.next(data).value)
  .then(data => run.next(data).value)
< ▶ Promise {<pending>}
  first<!doctype html>
  second<!doctype html>
  <!--[if IE 7 ]>      <html class="no-js ie ie7 lte7 lte8 lte9" lang="en-US"> <![endif]--
  <!--[if IE 8 ]>      <html class="no-js ie ie8 lte8 lte9" lang="en-US"> <![endif]-->
  <!--[if IE 9 ]>      <html class="no-js ie ie9 lte9>" lang="en-US"> <![endif]-->
  <!--[if (gt IE 9)|!(IE)]><!--> <html class="no-js" lang="en-US"> <!--<![endif]-->
  last<!doctype html>
  <!--[if IE 7 ]>      <html class="no-js ie ie7 lte7 lte8 lte9" lang="en-US"> <![endif]-
  <!--[if IE 8 ]>      <html class="no-js ie ie8 lte8 lte9" lang="en-US"> <![endif]-->
  <!--[if IE 9 ]>      <html class="no-js ie ie9 lte9>" lang="en-US"> <![endif]-->
  <!--[if (gt IE 9)|!(IE)]><!--> <html class="no-js" lang="en-US"> <!--<![endif]-->
```

异步：async


```
async function asyncAjax(url) {
    return new Promise((resolve, reject) => {
       $.ajax(url, {
           success: function(data) { resolve(data) },
           fail: function(reason) { reject(reason) },
       });
    });
}
```

```
result = await asyncAjax('/');
result2 = await asyncAjax('/');
result3 = await asyncAjax('/');
```

```
14
15  // 第一步
16  $.ajax(url, {
17      success: function() {...}
18  });
19
20  // 第二步
21  $.ajax(url, {
22      success: function() {...}
23  });
24
25  // 第三步
26  $.ajax(url, {
27      success: function() {...}
28  });
```

Thanks!