1. **ES6的兼容性**

<http://kangax.github.io/compat-table/es5/>

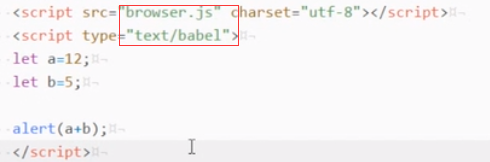
1、ES6(ES2015)--IE10+、Chrome、FireFox、移动端、NodeJS。

2、编译、转换

A、在线转换

B、提前编译

Babel = browser.js



1. **变量let和const**

var

1. 可以重复声明。
2. 无法限制修改。 如pi
3. 函数作用域，没有块级作用域 {} 。闭包模拟
4. 变量提升。undefined

let 不可以重复声明，变量-可以修改，有块级作用域{}，没有变量提升有暂时性死区报错

const 不可以重复声明，常量-不可以修改，有块级作用域{}，没有变量提升有暂时性死区报错

1. **函数-箭头函数**
2. 如果只有一个参数，()可以省略
3. 如果只有一条return语句，{}可以省略
4. 修正this。相对正常运行
5. 与普通函数的区别
6. 箭头函数是匿名函数，不能作为构造函数，不能使用new
7. 箭头函数不能绑定arguments，用rest参数展开运算符解决
8. 箭头函数不绑定this，会捕获其所在的上下文的this值作为自己的this;
9. 箭头函数通过call()，bind()，apply()方法调用一个函数时，只传入了一个参数，对this没影响
10. 箭头函数没有原型属性prototype

ii、仿造python

1. **函数-参数**
2. 参数的展开/扩展(rest paramters)。
3. 默认参数

参数扩展

1. 收集剩余参数

function show(a, b, ...args){}

\*Rest parameter 必须是最后一个，args自定义的

1. 展开一个数组/对象

\*展开后的效果就相当于写在这儿。

默认参数

function show(a, b=10, c= 34){}

1. **解构赋值**
2. 左右两边必须结构一样
3. 右边必须是个东西
4. 声明和赋值必须一起不能分开（必须在一句话里面完成）

例子





1. **数组**
2. map 映射
3. reduce 汇总（一堆出来一个）

let result = arr.reduce(temp, item, index) {}

1. filter 过滤器
2. forEach 循环迭代

\*这些方法配合箭头函数书写非常优雅

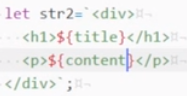
1. 数组的其他方法



1. **字符串**
2. 多了三个新方法

str.startsWith(‘’, index)， str.endsWith(‘’, index)， str.includes(‘’, index)。Index代表搜索位置，对应的正则表达式^ 和 $。可以看polyfill源码

1. 字符串模板



1. 可以通过${变量}形式在字符串里面加数据
2. 可以折行
3. **面向对象-基础**
4. 有了class关键字，构造器和类分开了。

class XXX{

constructor(a, b){this.a = a, this.b = b}

fun1(){}

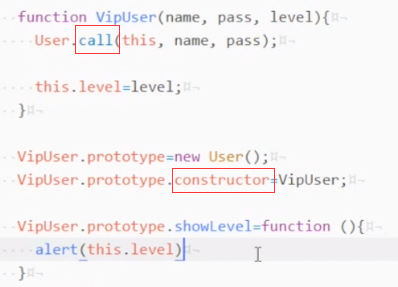
fun2(){}

...

}

1. class里面直接加方法。以前方法是外挂在prototype里面

继承：



call继承属性， prototype继承方法



super--超类==父类 继承属性， constructor继承方法

继承的最大意义是一切都可以不从零开始

1. **面向对象-实例**

React

1. 组件化---class
2. jsx。js的扩展版
3. **JSON**
4. JSON对象

JSON.stringify()

JSON.parse()

1. JSON的简写

A、Key和value一样 留一个就行

B、方法 :funtion一块删

show: function(){...}

show(){...}

JSON的标准写法：

a、必须用双引号

b、所有的名字都必须用双引号包起来

1. **Promise--承诺**

异步：操作之间没啥关系，同时进行多个操作，并发处理。

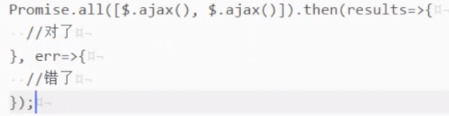
同步：同时只能做一件事。

异步：代码书写更复杂。 Ajax回调嵌套地域

同步：代码书写简单。

Promise--消除异步操作

\*用同步的方法来书写异步 promise.all() 高版本的jquery中ajax是一个promise对象





promise.all() 所有异步都成功了 替换回调嵌套

\*promise.race()--竞速

谁先成功就用谁。

1. **Generator--生成器函数**

Generator-生成器

----------------------------------------------------------------

普通函数--一路到底

generator函数--中间能停

-------------------------------------------------------------

funtion \*show(){

alert(1);

yield;//放弃的意思

alert(2);  
}

var generObj = show();

generObj.next(); //a

generObj.next(); //b

------------------------------------------------------------

踹一脚走一步

----------------------------------------------------------------

本质上generator生成了很多小函数

1. **Generator-yield是个啥**

yield 传参



yield返回



1. **Generator-实例：runner**

异步操作：

1. 回调

$.ajax({

url:‘/xx/xx’,

dataType:’json’,

success(data1){

$.ajax({

url:‘/xx/xx’,

dataType:’json’,

success: funtion(data1){

...

},

error: function(){

}

})

},

error(){

}

})

1. Promise--一次读一堆

Promise.all([

$.ajax(...),

$.ajax(...),

$.ajax(...),

]).then(resultArr=>{},err=>{})

1. Generator--带逻辑的读取，本质也是对promise的封装

runner(function \*(){

let s = yield $.ajax(...);

If(s){...}else{

let f =yield $.ajax(...)

}

})

4、async await

1. **Generator-实例：KOA**

\*nodejs

\*最新版本已经弃用了generator函数

\*用得最多的异步操作就是数据库操作

1. **ES7&ES8预览**
2. 数组 includes
3. 数组 keys/values/enteries与for...of配合使用

数组 JSON

for...of 值value x

for...in 下标key key

Keys->把所有的key拿出来

Values->把所有的value拿出来

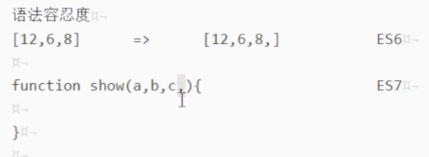
Enteries->所有的key-value拿出来

\*entry 实体

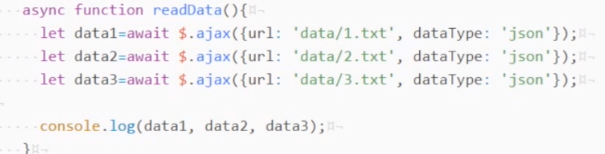
1. 幂

Math.pow(3,8) == 3\*\*8

1. 字符串padStart(num, world)前补位
2. 字符串padEnd(num, world)后补位
3. 语法容忍度



1. async await 代替 generator yield



1. 不再依赖与外部的runner
2. 可以用箭头函数

**1、深拷贝和浅拷贝**

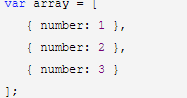
深拷贝作用：**在改变新的数组（对象）的时候，不改变原数组（对象）；**

**①**区别：一个是引用赋值；一个是创建一个新的数组或对象（复制）；

**②**深拷贝程度要求（拷贝几层）。

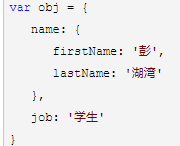
**③**方法

1. 直接遍历 数组第一层深拷贝
2. slice() 数组第一层深拷贝
3. **concat() 数组**第一层深拷贝
4. **ES6的展开运算符 数组**第一层深拷贝

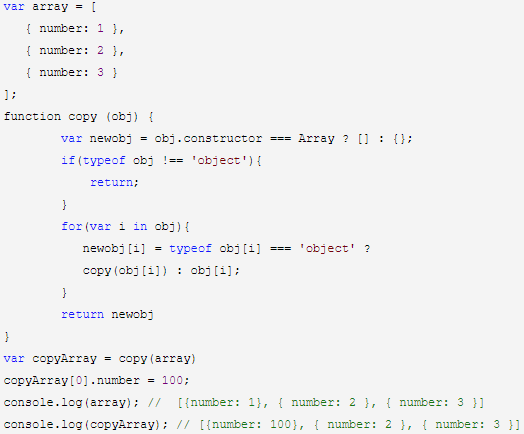
**以上三种方法对于第一层数组元素是引用类型（对象和数组）的都将失效**

1. **直接遍历 对象**第一级深拷贝
2. **ES6的object.assign() 对象**第一级深拷贝
3. **ES6的展开运算符 对象**第一级深拷贝

**以上三种方法对于多层嵌套对象的都将失效**



1. JSON.parse(JSON.stringify(xxxx)) 拷贝所有层级。数组，对象通吃
2. 手动写递归 拷贝所有层级



**④**大量深拷贝存在一个性能瓶颈。

引入immutable生成不可变的immutable对象，set,get方法而不改变原来的值。

在大量深拷贝中显著的减少性能消耗。