课程总结：

ES6

Let和const

解构赋值

正则扩展

数值扩展 移动到numer的方法

数组扩展 from fill

函数扩展 剪头函数

对象扩展

Obj增加的方法

Symbol

避免不同的key起冲突

Proxy

保护的对象通过代理访问

**Reflect**

Obj的方法移植上去

类

多重继承

**class** Lottery **extends** *mix*(Base,Timer,Calculate,Interface)

工具

Gulp 和webpack

功能

切换玩法

利用tab修改同一个dom的文本 文本来自map 使用es6模板字符串拼接

方法

切换样式是并且修改当前对象的玩法

服务端通信的接口写在同一个模块（interface。js）

第一章

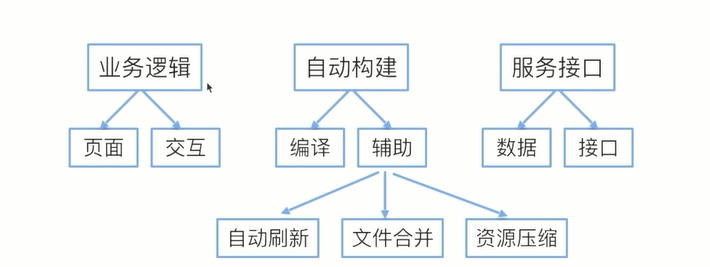
模块化概念

工程构建

第二章

架构基础

IE8 IE6 IE5 识别ES3



任务自动化（gulp）

减少人工操作，自动监听变化

Sass 转成浏览器可以识别的css

学习

Gulp中文文档

Gulp插件

实战

编译工具（babel （将es6编译成浏览器可以识别的es5）webpack 处理依赖关系）

Babel

js的编译器

webpack-stream

webpack对babel的支持

代码实现

目录结构

自动构建

服务器搭建

代码实现

App前端

Server

Tasks

构建工具目录

Server

模板文件为什么用ejs（可以当做html使用）

Nodejs使用的魔板引擎是ejs

ES6文件夹下面

Express -e .

在当前目录下使用ejs模板引擎

Tasks

Util常见脚本

入口文件

根目录

Package.Json

项目需要安装依赖包（？）的过程中需要

可以通过npm install安装需要的依赖包

Touch .bablerc

设置bable编译工具

Touch liunux创建一个新的文件

gulpfile.bable.js

因为我们需要用到es6 所以中间有一个bable

Script.js 处理js

先import确定要用哪些包在安装

在packgejson创建安装包的依赖

--save-dev

//处理js构建脚本

//处理模板构建脚本

//服务器相关的构建脚本

//清空指定目录文件的脚本

把写好的任务自动完成

Gulp 没有指定任务会默认去找default.Js

支持到es几需要在babellrc里面进行配置

# ES6语法

Es6强制开启严格模式

变量未声明不能引用

Let和const

ES5

全局作用域

函数作用域

Let块作用域

Let变量不能重复声明

Const 声明的时候必须赋值

对象类型不能修改的是指针

字符串

如果超出unicode编码的字符怎么打印

超过了0XFFF 码值大于两个字节 超出的部分作为单独的字符

Console.log（‘s’,’\u{20BB7}’）🡪打印出了一个字符 但是length长度是2

**字符串的遍历器接口**

Let code of str 字符串遍历器

能够处理大于0XFFF的unicode字符

**API**

Includes

StartsWith

endsWith

repeat 字符串重复出现几次

padStart 填补到达指定的长度

raw导致斜杠不生效

**模板字符串**

**let *name*** = **"aaa"**;  
**let *info*** = **"bbb"**;  
**let *m*** = **`i am** ${***name***} ${***info***}**`**;  
**console**.log(***m***);

**标签模板**

**let *user*** ={  
 **name**:**"list"**,  
 **info**:**"hello world"**}  
**console**.log(*abc***`iam** ${***user***.**name**}**,**${***user***.**info**}**`**);  
**function** *abc*(s,v1,v2) {  
 **console**.log(s,v1,v2);  
 **return** s+v1+v2;  
}

**console 的是函数return的结果**

**怎么用**

**使用场景**

防止XSS工具

多语言转换

# PROMISE

等到所有图片加载完在显示页面

**function** *loadImg*(src) {  
 **return new** Promise((resolve,reject)=>{  
 **let** img = **document**.createElement(**"img"**);  
 img.**src** = src;  
 img.onload = **function**(){  
 resolve(img);  
 }  
 img.onerror = **function** (err) {  
 reject(err);  
 }  
 })  
}  
**function** *showImg*(imgs){  
 img.forEach(**function**(img){  
 **document**.**body**.appendChild(img);  
 })  
}  
*//多个PROMISE实例当做一个promise实例 三个状态都改变了就会显示图片*Promise.all([*loadImg*(url1),*loadImg*(url2),*loadImg*(url3)]).then(*showImg*)  
 *//哪个图片先加载完就显示* Promise.race([*loadImg*(url1),*loadImg*(url2),*loadImg*(url3)]).then(*showImg*)

# 类

基本方法

继承

静态方法和属性

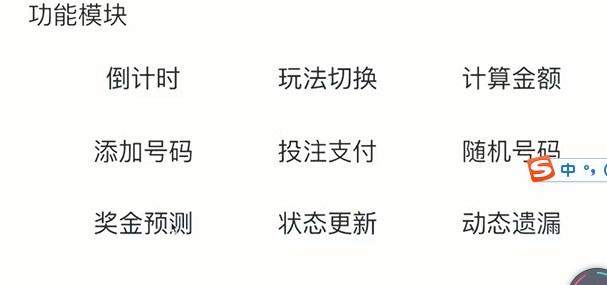
Get setter

# PROXY和reflect

概念

适用场景

# 实战



4-3

没有强制要求constructor，因为没有需要在一开始添加一些属性

方法的声明发生了改变

Class的方法之间不需要用，分割

getState(issue){  
 **let** self = **this**;  
 **return new** Promise((resolve,reject)=>{  
 $.ajax({  
 **url**:**'/get/state'**,  
 **data**:{  
 **issue**:issue  
 },  
 **dataType**:**'json'**,  
 success:**function** (res) {  
 resolve.**call**(self,res);  
 },  
 error:**function** (err) {  
 reject.**call**(err);  
 }  
 })  
 })

用一个self保存this指向的原因是 箭头函数的this是定义的时候的this指向

Interface.Js

1返回一个promise的实例支持链式调用

2调用实例的方法设置实例的属性 其他额方法也可以共享这个数据 所以不需要在回调

4-6

切换玩法

Entries

for (let index of ['a', 'b'].keys()) {

console.log(index);

}

// 0

// 1

for (let elem of ['a', 'b'].values()) {

console.log(elem);

}

// 'a'

// 'b'

for (let [index, elem] of ['a', 'b'].entries()) {

console.log(index, elem);

}

// 0 "a"

// 1 "b"