正课: ES6

1. \*\*\*let

2. 箭头函数

3. 参数增强

4. 模板字符串

5. 解构

6. OOP

7. for of

8. \*\*\*\*\*Promise

1. \*\*\*let

什么是: 专门用于代替var声明变量

何时: 今后，所有的变量几乎都用let声明

作用: 2个:

1. 防止声明提前(hoist)

原理: let检查不允许在let之前使用未声明的变量。

2. 新增块级作用域

块级作用域: if...else for while do...while switch

什么是: 块内的变量，不再提前到块外，影响外部！而是，只能在块内使用!——同Java

为什么: 防止内外变量互相干扰！

原理: 当前块上自动添加匿名函数自调，为let划分临时作用域!

2. 箭头函数:

什么是: 对所有回调函数的简化

何时: 几乎所有的回调函数，都可用箭头函数简化

如何:

1. 去function，改为=> (参数列表)=>{函数体}

2. 如果函数体只有一句话，可省略{}

如果仅有的一句话还是return，则return也可省略

3. 如果只有一个参数，可省略()

特点: 改为箭头函数，则回调函数内外共用同一个this

好处: 代替bind，让函数内外this，通用!

坏处: 如果刚好希望内外this不通用时，就不能用=>

3. 参数增强:

1. 参数默认值Default:

function fun(参数1,参数2,参数3=默认值)

强调: 带默认值的参数必须放在参数列表结尾

ES5: fromi=fromi||0;

2. 剩余参数REST:

function fun(参数1,参数2,...数组名)

在调用时, 会在函数内创建同名数组，数组中接收除前两个参数外的所有剩余参数!

为什么: arguments的缺点:

1. 类数组对象，无法使用数组API

2. 只能选择全部，无法有选择的获取部分参数.

3. 书写复杂!

何时: 今后，都用...数组,代替arguments

优点: 1. 纯正的数组! 可用数组的所有API

2. 可有选择的获取部分参数，而不是全部!

3. 可自定义数组名

模拟: ES5中

将类数组对象转化为纯正的数组:

var subArr=arr.slice(starti,endi+1)

不修改原数组，仅复制出指定位置的元素组成新数组返回

slice省略第二个参数: 表示到结尾

再省略第一个参数: 表示从头到尾全部复制

问题1: arguments.slice() 报错! arguments不是数组

解决: 抢call

var arr=Array.prototype.slice.call(arguments,starti);

arguments.slice(starti)

问题2: 仅选取结尾的部分参数:

解决: 在.call(arguments,starti)

3. 散播: Spread

什么是: 打散数组类型参数为单个元素

何时: 取代.apply()

如何: fun(...数组)

将数组打散为单个元素，再分别作为参数传入fun中

ES5: fun.apply(obj,数组)

4. 模板字符串:

什么是: 支持动态执行代码的字符串拼接

何时: 只要复杂的字符串拼接，都用模板字符串代替

如何: 2步:

1. 用模板字符串定义格式: 格式模板用反引号`包裹

2. 在格式中，用js表达式动态生成内容

每项动态生成的内容必须放在${js表达式}内

5. 解构:

什么是: 将一个对象或数组整体，分解到多个变量中

简化: 变量的批量赋值

如何: 2种:

1. 数组解构: 按下标对应关系结构

var arr=[89,59,91];

var [math,chs,eng]=arr;

2. 对象解构: 按属性名对应关系结构

var obj={

math:89,

chs:59,

eng:91

};

var {math:math,chs:chs,eng:eng}=obj;

//ES6

var {math,chs,eng}=obj

其实,结构也可发生在参数传递时

6. OOP

为了简化OOP:

1. 创建类型:

用class结构包裹构造函数和原型对象

将构造函数名提升为类型名放在class后

将"function 类型名"改为"constructor"

直接定义在class内的方法，自动保存在原型对象中

class 类型名{

constructor(属性参数列表){

this.属性名=属性参数;

... = ... ;

}

方法1(){...},

方法2(){...},

}

2. 继承:

1. 让子类型继承父类型

class Sub extends Father

2. 在子类型构造函数中，借用父类型构造函数

super(属性参数列表)

//super自动指向父类型构造函数

//自动替换其中this为当前对象

3. 访问器属性:

2步:

1. 在构造函数中为当前对象this添加一个隐藏的数据属性

2. 在constructor外部平级，添加访问器方法:

get 访问器属性名(){return this.受保护的隐藏数据属性}

set 访问器属性名(val){验证，赋值或报错}

4. 静态方法:

什么是: 保存在构造函数对象上，不需要创建子对象，就可用构造函数直接调用的方法。比如: Array.isArray() Object.seal()

何时: 只要一个方法，不确定将来执行时的对象类型

如何: 在构造函数的外部平级定义:

static 静态方法名()

调用:

错误: 子对象.静态方法()

正确:类型名.静态方法名()

5. 重写:

在子类型的重写方法中，要借用父类型的方法

super.父类型方法()