正课:

1. 解构

2. for of

3. class

4. \*\*\*Promise

1. 解构:

什么是: 将一个对象或数组中的成员，分别赋值给多个单独的变量——简化批量赋值

何时: 只要将一个对象或数组中的成员，分别赋值给多个单独变量时

如何: 2种:

1. 数组解构:

var arr=[1,2,3];

var [a,b,c]=arr;

console.log(a,b,c);//1 2 3

强调: 数组解构靠下标对应

2. 对象解构:

var date={y:2017,m:9,d:21};

var {y:year,m:month,d:date}=date;

console.log(year,month,date);

强调: 对象解构靠属性名对应

函数调用时结构:

定义函数时: function fun({y:y,m:m,d:d}){...}

调用时: var date={y:2017,m:9,d:21};

fun(date)

2. for of: 简化版的for循环

如何: for(var elem of arr)

elem可直接获得arr中每个元素值

何时: 代替for循环，遍历下标为数字的数组或类数组对象

强调: 不能遍历关联数组和对象，因为下标不是数字

缺点:

1. 无法获得下标——要用下标时，不能用for of

2. 只能遍历所有，不能有选择的遍历

3. 只能从前向后

3. class类型:

什么是: 封装一类对象统一结构和API的 程序结构

——简化js中的面向对象

为什么: 传统的js类型定义方式不符合封装的特点

如何:

简化类型定义——封装:

1. 用class 类型名{}包括之前的构造函数和原型对象方法

2. 构造函数名升级为class名, 构造函数更名为constructor关键词

3. 原型对象方法不用写类型.prototype和function

直接写在class中的函数，自动添加prototype中

简化继承:

1. class 子类型 extends 父类型

类似于:

Object.setPrototypeOf(

子类型.prototype, 父类型.prototype

);

2. 借用父类型构造函数: super(参数...)

不用call，不用this！

其中: super特指父类型构造函数,且自动用当前正确this替换父类型构造函数中的this

super.父类型原型方法()

静态方法:

什么是: 不需要实例化对象，就可直接访问的成员

何时: 只要一个方法，不需要实例化也能使用时

原理: 相当于直接定义在构造函数对象上的方法

如何: static 方法(){...}

访问器属性:

get 属性名(){ return this.\_属性名;}

set 属性名(val){

if(判断)

this.\_属性名=val;

否则

报错

}}

4. \*\*\*Promise

什么是: 回调函数的一种更直观的写法

——规范回调函数的使用

何时: 如果回调函数，必须等待主函数执行完，才能执行时

为什么: callback hell

如何: 2步:

0. 前一个函数定义不要在参数上写callback

1. 前一个函数定义中: 用一个巨大的Promise对象包裹函数的内容，并将Promise对象返回

Promise的参数是一个巨大的回调函数:

1. 参数: callback: 用于接收将来承诺要做的后续函数

2. 函数体: 包括原正常执行的逻辑，并在逻辑中决定何时调用callback

2. 调用前一个函数时，可用.then传入下一项任务的函数对象作为回调函数

错误处理: 2步

1. 其实: promise的函数参数，还有第二个回调函数参数，用于在出错时调用 then catch

return new Promise(function(resolve,reject){

正常逻辑中:

如果正确执行, 调用resolve()

如果出错,调用reject("错误信息"/错误对象)

})

强调: 如果前一个函数调用了reject相当于抛出一个错误，不处理，程序就闪退

2. 错误处理:

在调用时，最后一个then的结尾追加

.catch(function(err){//err接住reject的参数

错误处理代码

})

等待多个任务完成才执行:

如何: Promise.all([ 函数1(), 函数2(), .......]).then(end)

1. 数组中的多个函数，并行！

2. 数组中最晚的任务执行后，自动调用end