**正课:**

**1. ES6**

**1. ES6**

**for of : 简化版的for循环，专门用于遍历数组**

**遍历数组: 3种:**

**1. for(var i=0;i<arr.length;i++)**

**var elem=arr[i]**

**最灵活:**

**2. arr.forEach((elem,i,arr)=>{})**

**限制: 无法控制遍历的顺序和步伐**

**3. for(var elem of arr)**

**of会依次取出arr中的元素值，保存在变量elem中**

**限制: 无法获得当前位置i**

**vs for in: 专门遍历自定义下标名称的关联数组和对象**

**for of: 专门遍历数字下标的数组和类数组对象**

**class:**

**封装:**

**1. 创建一个对象:**

**var obj={**

**属性名:变量名 如果属性名和变量名一样，可只写一个**

**方法名(){**

**... ...**

**}**

**}**

**2. class:**

**什么是: 封装一个类型的构造函数及其原型对象方法的程序解构。**

**为什么: 旧式的类型创建语法，不符合封装的要求**

**何时: 只要创建一种类型，都要用class封装构造函数和原型对象方法**

**如何:**

**1. 创建class:**

**1. 用class{} 包裹构造函数和原型对象方法**

**2. 构造函数名提升为class名，构造函数去掉function,并更名为固定的constructor**

**3. 原型对象方法可省略.prototype前缀和function**

**直接定义在class下的方法，默认保存在原型对象中**

**2. 使用class: 用法和构造函数完全一致，也用new**

**访问器属性:**

**前提: 依然需要一个隐藏的数据属性被保护，实际存储数据。**

**定义访问器属性:**

**class xxx{**

**构造函数(){ ... }**

**get 属性名(){ return ...}**

**set 属性名(value){ ... }**

**}**

**相同属性名的一对儿get/set方法自动组成一个访问器属性，并保存在原型对象中。**

**静态方法:**

**class 类型名{**

**构造函数(){ ... }**

**static 方法名(){ ... }**

**}**

**调用时: 类型名.方法名()**

**继承:**

**2步:**

**1. 让子类型继承父类型:**

**class 子 extends 父{**

**...**

**}//extends代替了Object.setPrototypeOf**

**2. 子类型构造函数中借用父类型构造函数**

**constructor(...){**

**super(...) //代替了.call(this)**

**//super指代父类型，super()相当于调用父类型的构造函数**

**//super()不需要传this**

**}**

**\*\*\*\*\*Promise:**

**问题: 回调地狱 callback hell**

**为了保证多个异步任务必须顺序执行，而形成了很深的嵌套调用结构**

**解决: Promise**

**何时: 只要多个异步任务，希望顺序执行时**

**如何:**

**1. 定义Promise对象封装要执行的异步任务**

**new Promise(function(open){**

**//异步任务...**

**//在异步任务最后一步之后，打开开关: open()**

**})**

**强调:**

**1. new Promise在创建对象同时，就自动执行内部的异步任务**

**2. 如果不open(),则后续的then不执行！**

**2. 将多个Promise对象，用.then连起来：**

**Promise1**

**.then(function(){**

**return Promise2**

**})**

**...**

**.then(function(){**

**最后一项任务**

**})**

**错误处理:**

**1. 在new Promise中，如果出错，则调用err开关:**

**new Promise(function(open,err){**

**//异步任务**

**//如果正常执行完，调用open()**

**//如果出错，调用err("错误消息")**

**})**

**强调: 调用err()后，后续then不执行**

**2. 在顺序调用的结尾，只添加一个.catch()**

**.catch(function(err){ ...err变量接住"错误消息" })**

**前后两个异步调用间传参**

**前一个异步调用，通过open(参数)方式传参**

**后一个异步调用，通过.then(function(参数){ ... })方式接受**

**简化: ES7**

**ES6的Promise的问题: .then()依然使用嵌套方式：**

**.then(function(){**

**return 异步调用的函数**

**})**

**解决: await:**

**前提: await修饰的函数必须也是return new Promise()的**

**如何:**

**(async function(){ //async说明大任务整体是异步的**

**try{ //代替.catch(function(err){}) 做错误处理**

**var result=await 异步调用1**

**//其中: await是等待当前函数执行完的意思**

**// 阻塞程序执行**

**// 如果异步调用中通过open(参数)返回了数据，可用var result=来接住异步任务返回的数据，在后续步骤中继续使用。**

**var result=await 异步调用2**

**...**

**}catch(err){**

**...**

**}**

**})()**

**等待多个异步任务执行后才执行:**

**Promise.all([**

**异步调用1,**

**异步调用2,**

**... ...**

**])**

**.then(function(){**

**...**

**})**

**.catch(function(err){ ... })**