正课:

1. ES6

1. ES6

1. Promise:

什么是: 专门保证多个异步函数必须顺序执行，而又不会形成回调地狱的技术

为什么: 解决回调地狱问题

何时: 多个异步函数必须顺序执行

如何:

1. 将原异步任务函数的语句，原封不动的封装在一个new Promise对象中，并返回到函数外部:

function liang(){

return new Promise(

//赠door

// ↓自动传入

function(door){

console.log("亮起跑!")

setTimeout(function(){//异步回调

console.log("亮到达终点!");

console.log("===亮开门！===")

door();//在自己的任务最后一句话执行后手动开门

//一开门，就自动触发提前用.then串联的下一个任务函数

},10000)

}

)

}

强调: 函数内部创建的new Promise如果不返回，就不能用.then()和下一个异步任务串联

2. 多个返回new Promise()对象的函数，可用.then()串联起来:

liang().then(ran).then(dong).then(()=>console.log("比赛结束"));

强调: 1. 后边串联的函数都不要加()，因为不是立刻执行。只有前一个函数开门后，自动调用then中的函数

2. 只有返回new Promise()的函数后才能接下一个then。

3. then其实即能接new Promise()对象，又可以接普通的函数。只不过，接普通函数时，不能再用then()继续接下一步任务。

Promise的状态机制:

其实，liang().then(ran).then(dong)

做了两件事：

1. 调动liang()

2. 将所有后续.then中收集的函数，放入一个队列中保存起来。

在liang()的Promise对象内部，假设有一个死循环。死循环根据状态判断是继续等待，还是可以向后执行队列中下一个任务函数。

Promise中有三种状态:

1. pending: 当调用任意一个任务函数时，暂时处于pending状态，是不会去执行下一个任务的。

2. resolve: 当前任务执行完，且执行成功，任务函数内部会手动调用resolve()函数。Promise对象会将任务的状态改为resolve状态。死循环发现状态变为resolve，则自动i++，找到集合中排在当前位移的下一个位置的任务函数执行。并在此将状态改为pending。直到当前任务执行完，才判断是否可以继续。

3. reject: 如果当前任务执行过程中出错了，无法继续向后执行，则任务函数中会先手动调用reject()函数。Promise对象将任务的状态转为reject。死循环发现任务的状态变为reject，则不再继续循环，而是退出循环，执行最后一个catch的内容。

如何处理Promise多个异步任务的状态:

1. 定义函数时：如果内部执行出错，就调用reject("错误提示")； 如果执行正确，就调用resolve()。

function liang(){

return new Promise(

function(resolve,reject){

console.log("亮起跑!")

setTimeout(function(){//异步回调

if(Math.random()<0.3){ //模拟如果出错

reject("啊！亮摔倒了!");

}else{//否则，如果正确执行

console.log("亮到达终点!");

console.log("===亮开门！===")

resolve();//在自己的任务最后一句话执行后手动开门

//一开门，就自动触发提前用.then串联的下一个任务函数

}

},10000)

}

)

}

2. 调用函数时: 在最后加.catch(e=>{错误处理代码})

liang()/\*return new Promise()\*/.then(ran)/\*return new Promise()\*/.then(dong)/\*return new Promise()\*/.then(()=>console.log("比赛结束")).catch(e=>console.log("不好啦!"+e));

结果: 任何一步执行通过都调用resolve，然后触发下一个then()

任何一步执行错误都调用reject，然后不再执行.then()，而是执行.catch()。

前后两个任务间传值:

上一步: resolve(一个变量)

下一步函数定义时： function next(形参)

上一步().then(next)

结果: next的形参自动接到上一步传来的一个变量值。

强调: 通过resolve()传参传值，只能传一个变量。

如果就要传递多个值，可放在一个对象或数组中统一传递。

需要等待多个任务执行完才能继续执行:

1. 每个任务只要让别人等待，就必须是return new Promise()

2. Prmise.all([

任务1(), //return new Promise()

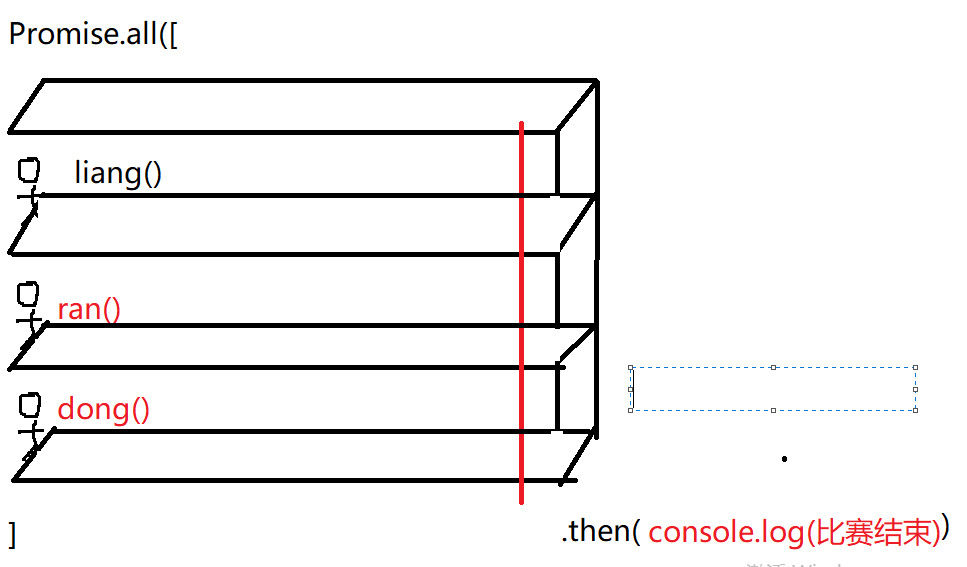
任务2(), //return new Promise()

任务3() //return new Promise()

]).then(()=>{

比赛结束

}).catch(e=>{...})



如果多个任务都有返回值，都是放在一个数组中返回的:

任务1() resolve(101)

任务2() resolve(102)

任务3() resolve(103)

.then(arr=>{

... arr:[101, 102, 103]

强调: arr中返回值的顺序，不是以函数调用结束的顺序来组织。而是以调用的顺序组织的！

})

ES7中对promise的简化!

定义时，没有简化。

调用时，提供了更直观的调用方式:

liang().then(ran).then(dong).then(()=>console.log(比赛结束))

//ES7 异步 函数自调

(async function(){

try{

var bang=await liang();

bang=await ran(bang);

bang=await dong(bang);

console.log("比赛结束");

}catch(e){

... ...

}

})()

其中: async 是整个串联的异步任务还是异步任务，不会变成同步的回主程序执行，所以必须写async

await 是让后续程序，必须等待当前函数执行完，才能继续执行。

每个异步函数resolve(返回值)，可用"var 变量="的方式接住。

try{ ... }catch(e){ ... }结构同样可以接住Promise函数中reject抛出的错误！

单词列表:

1. pending 挂起 当前函数正在执行，还未执行完，好没有获得结果！

2. fulfilled 已解决

3. resolve 解决

4. reject 拒绝

5. asynchronous; 异步， 简写为async

6. await 等待