当内部函数使用了外部函数的变量 就会产生闭包;

闭包的金典场景:

当内部函数使用了外部函数的变量;且内部函数调用时;外部函数已经执行完毕;

这个时候闭包才发挥了他真正的作用

当函数可以记住 并 有能力访问到自己的作用域链时就会产生闭包

(ES5) this到底指向于谁? 看函数调用位置的调用形式

普通调用 this--> window: fn()

隐式调用 this--> obj: obj.fn()

显示调用 this--> obj: fn.call(obj) fn.apply(obj)

构造调用 this--> 构造出来的实例对象 : new fn()

隐式丢失

当使用隐式绑定的形式去进行变量的赋值时(函数的传参)容易出现隐式丢失

本来我们认为this应该指向赋值时的调用者的

最终this指向了window或者其他对象

如何解决隐式丢失

使用显示调用 或 硬绑定函数bind来定死this指向

变量的查找基于作用域链;如果整条作用域链上都没有对应变量的声明 则会报错(右查询)

变量的查找基于作用域链;如果整条作用域链上都没有对应变量的声明 则在全局自动声明一份(左查询)

属性的查找基于原型链 ;如果整条原型链上都没有对应的属性 则返回undefined

变量的查找 : 左右查询

左查询: 对等号左边的变量发起查询

右查询: 对等号非左边的变量发起查询

promise是一种机制来管理回调

本质上Promise就是一个构造函数。

promise的第一个参数：执行器函数（executor）是一个同步的回调函数!

所有的Promise实例都具备三个状态: 初始化 成功 失败

pending: 初始状态，既不是成功，也不是失败状态。

fulfilled: 意味着操作成功完成。(resolved)

rejected: 意味着操作失败

new Promise(exec): 执行器exec先同步执行 promise实例才被创建

如果执行器出错了 那么promise是失败状态的

如果执行器的resolve被调用了 那么promise是成功状态

如果执行器的reject被调用了 那么promise是失败状态

如果执行器没有进入上面三个流程中任意一个 那么promise是初始化状态

then方法返回的promise是什么状态 跟then中参数(回调函数)有关

如果回调函数在执行过程中出错 那么promise是失败状态的

如果回调函数成功的返回一个值 那么promise是成功状态的

如果回调函数成功的返回一个promise 那么then方法返回的promise

跟这个回调返回的promise保持一致的状态

失败情况

抛出错误 throw new Error（“”）

返回失败的promise then promise.Reject（“”）

成功情况

1．返回一个成功的promise resolve

2.其他所有情况（返回一个非promise的值）