根据promise/deferred模式

var fs = require('fs');

var Promise = function(){

this.isPromise = true;

//检查then中的参数(方法)返回的是不是Promise对象

this.queue = [];

//用于存放then的参数，实现队列功能

};

Promise.prototype = {

then:function (fulfilledHandler,errHandler) {

var handler = {};

if(typeof fulfilledHandler === 'function'){

handler.fulfilled = fulfilledHandler;

}

if(typeof errdHandler === 'function'){

handler.error = errHandler;

}

this.queue.push(handler);

//then的作用就是将他的参数push进 //Promise的queue数组中

return this;

//返回Promise对象以实现链式调用then

}

};

var Deferred = function () {

this.promise = new Promise();

};

Deferred.prototype = {

resolve:function (obj) {

//resolve作用就是将Promise的queue数组中的方法 //按序取出并执行

var promise = this.promise;

var handler;

while((handler = promise.queue.shift())){

//先进先出

if(handler && handler.fulfilled){

var ret = handler.fulfilled(obj);

if(ret && ret.isPromise){

//这里的判断很重要，若then中的方法返回的是Promise对象就像Promise对象更新为then返回的Promise，然后return，只要then中的参数返回Promise对象，则resolve的while就循环一次，这样then参数中的方法调用了resolve后才能继续向下执行，这样才能保证按顺序传递参数，相当于下一步的执行一来上一步的完成。

ret.queque = promise.queue;

this.promise = ret;

//这里将新返回的promise对象引用指向最初的promise，//因此程序执行期间只有一个promise对象

return;

}

}

}

},

reject:function (obj) {

var promise = this.promise;

var handler;

while((handler = promise.queue.shift())){

if(handler && handler.error){

var ret = handler.error(obj);

if(ret && ret.isPromise){

ret.queque = promise.queue;

this.promise = ret;

return;

}

}

}

},

};

var deferred = new Deferred();

function f1(){

fs.readFile('zy1.txt','utf-8',function(err,file) {

if (err) {

deferred.reject(err);

}

if(file){

deferred.resolve(file);

//只有这一步执行后才会执行第一个then中的方法，

//因为第一个then中的参数返回Promise对象，所以then中的 //while只执行一次就停止

}

});

return deferred.promise;

}

f1().then(function(file1){

//then 的参数不执行只是push进f1返回的Promise对象 //的queue数组中

console.log(file1);

fs.readFile(file1,'utf-8',function(err,file){

console.log(file);

if (err) {

deferred.reject(err);

}

if(file){

deferred.resolve(file);

//只有这一步执行后才会执行第二个then中的方法。

}

});

return deferred.promise;

}).then(function (file2) {

console.log(file2);

});

原理分析：

1用队列存放then中的参数，当resolve时顺序执行

2一旦检测到then中的参数返回Promise对象就立即停止执行，然后将当前deferred中的promise引用指向新的then中的参数返回Promise。

new Promise(executor);创建一个新的promise，初始化为等待状态，并提供解决函数的引用，用于改变其状态。executo函数会马上被调用，它带有两个参数的函数对象executor(resolve, reject) 构造函数直到执行完成才会返回。解决函数可以在任何时间被调用，在执行过程完成前后都可以，其目标是控制promise的最终状态。如果执行过程抛出异常，异常值会被传递到reject解决函数中。

**then()**

一旦promise完成或被拒绝，就立即执行一个回调函数，并返回一个新的promise对象，新promise的状态取决于promise和传入的回调函数。回调函数始终会在then返回值后被调用，即使promise早已完成或已被拒绝。也可对多次调用同一个promise的then方法。所有的回调函数会按照其注册的顺序被调用。

Promise.then()方法接受两个参数，这两个参数的类型必须是函数，在第一个参数函数内，你只可以做下面三件事：

1、return另一个Promise

2、return一个同步值（包括undefined）

3、throw一个异常

如果then回调函数里缺少了重要的返回值，那么JS会默认返回undefined。而如果返回undefined这个同步值的话，这个then所代表的Promise会立即决议为undefined，并开始下一个then的运行。

then只接受function类型的参数，若传入的并非function类型，所以为会认为传入了一个null，其结果和.then(null)是一样的。这样会导致前一个promise会穿透到下面。