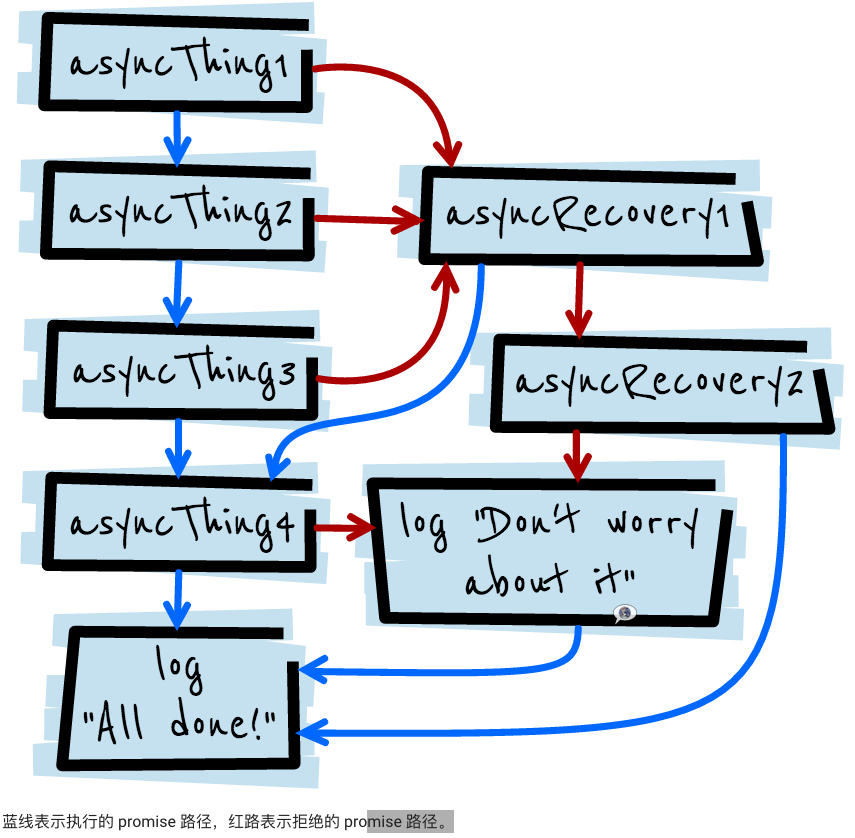
* asyncThing1().then(function() {  
    return asyncThing2();  
  }).then(function() {  
    return asyncThing3();  
  }).catch(function(err) {  
    return asyncRecovery1();  
  }).then(function() {  
    return asyncThing4();  
  }, function(err) {  
    return asyncRecovery2();  
  }).catch(function(err) {  
    console.log("Don't worry about it");  
  }).then(function() {  
    console.log("All done!");  
  })
* 
* 小结：
* 1. catch可以捕获之前的多个then处理错误
* 2. catch捕获异常之后，可以继续进行then操作
* Promise 有一段时间以库的形式出现，例如：
* [Q](https://github.com/kriskowal/q)
* [when](https://github.com/cujojs/when)
* [WinJS](https://msdn.microsoft.com/en-us/library/windows/apps/br211867.aspx)
* [RSVP.js](https://github.com/tildeio/rsvp.js)
* 以上这些与 JavaScript promise 都有一个名为 [Promise/A+](https://github.com/promises-aplus/promises-spec) 的常见标准化行为。如果您是 jQuery 用户，他们还有一个类似于名为 [Deferred](https://api.jquery.com/category/deferred-object/) 的行为。但是，Deferred 与 Promise/A+ 不兼容，这就使得它们[存在细微差异且没那么有用](https://thewayofcode.wordpress.com/tag/jquery-deferred-broken/)，因此需注意。此外，jQuery 还有 [Promise 类型](https://api.jquery.com/Types/#Promise)，但它只是 Deferred 的子集，因此仍存在相同的问题。
* 尽管 promise 实现遵照标准化行为，但其整体 API 有所不同。JavaScript promise 在 API 中类似于 RSVP.js。
* JavaScript promise 最初是在 DOM 中出现并称为“Futures”，之后重命名为“Promises”，最后又移入 JavaScript。在 JavaScript 中使用比在 DOM 中更好，因为它们将在如 Node.js 等非浏览器 JS 环境中可用（而它们是否会在核心 API 中使用 Promise 则是另外一个问题）。
* 尽管它们是 JavaScript 的一项功能，但 DOM 也能使用。实际上，采用异步成功/失败方法的所有新 DOM API 均使用 promise。[Quota Management](https://dvcs.w3.org/hg/quota/raw-file/tip/Overview.html#idl-def-StorageQuota)、[Font Load Events](http://dev.w3.org/csswg/css-font-loading/#font-face-set-ready)、[ServiceWorker](https://github.com/slightlyoff/ServiceWorker/blob/cf459d473ae09f6994e8539113d277cbd2bce939/service_worker.ts#L17)、[Web MIDI](https://webaudio.github.io/web-midi-api/#widl-Navigator-requestMIDIAccess-Promise-MIDIOptions-options)[Streams](https://github.com/whatwg/streams#basereadablestream) 等等都已经在使用 promise。
* 与其它库的兼容性
* JavaScript promise API 将任何使用 then() 方法的结构都当作 promise 一样（或按 promise 的说法为 thenable）来处理，因此，如果您使用返回 Q promise 的库也没问题，因为它能与新 JavaScript promise 很好地兼容。
* 如我之前所提到的，jQuery 的 Deferred 不那么有用。幸运的是，您可以将其转为标准 promise，这值得尽快去做：
* var jsPromise = Promise.resolve($.ajax('/whatever.json'))
* 这里，jQuery 的 $.ajax 返回了一个 Deferred。由于它使用 then() 方法，因此 Promise.resolve() 可将其转为 JavaScript promise。但是，有时 deferred 会将多个参数传递给其回调，例如：
* var jqDeferred = $.ajax('/whatever.json');  
    
  jqDeferred.then(function(response, statusText, xhrObj) {  
    // ...  
  }, function(xhrObj, textStatus, err) {  
    // ...  
  })
* 而 JS promise 会忽略除第一个之外的所有参数：
* jsPromise.then(function(response) {  
    // ...  
  }, function(xhrObj) {  
    // ...  
  })
* 幸好，通常这就是您想要的，或者至少为您提供了方法让您获得所想要的。另请注意，jQuery 不遵循将 Error 对象传递到 reject 这一惯例。
* 浏览器支持
* 现在，promise 已在各浏览器中实现。
* 在 Chrome 32、Opera 19、Firefox 29、Safari 8 和 Microsoft Edge 中，promise 默认启用。
* 如要使没有完全实现 promise 的浏览器符合规范，或向其他浏览器和 Node.js 中添加 promise，请查看 [polyfill](https://github.com/jakearchibald/ES6-Promises#readme)（gzip 压缩大小为 2k）。
* 1. 并发操作
  + Promise 还有另一个伟大的特性，它们可以同时进行多个异步操作，等他们全部完成之后再继续进行其它事件。ES2015 规范中提供了 Promise.all()，就是用来干这个事情的。

function doAsyncOp() {

  return **Promise**.**all**([

    asynchronousOperation(),

    asynchronousOperation()

  ]).then(function(vals) {

    vals.forEach(console.log);

    return vals;

  });

}

* 1. **将 Promises 转换成异步函数**
  + promise写法

function doAsyncOp () {

  return asynchronousOperation().then(function(val) {

    console.log(val);

    return val;

  });

};

* + async写法

async function doAsyncOp () {

  var val = await asynchronousOperation();

  console.log(val);

  return val;

};

* + 注意
    - 这里有相同的代码行数，但这是因为 then 和给它传递的回调函数增加了很多的额外代码。另一个让人厌烦的是两个 return 关键字。这一直有些事困扰着我，因为它很难弄清楚使用 promises 的函数确切的返回是什么。
* 2. **链式操作**
  + promise写法

function doAsyncOp() {

  return asynchronousOperation()

    .then(function(val) {

      return asynchronousOperation(val);

    })

    .then(function(val) {

      return asynchronousOperation(val);

    })

    .then(function(val) {

      return asynchronousOperation(val);

    });

}

* + async写法

async function doAsyncOp () {

  var val = await asynchronousOperation();

  val = await asynchronousOperation(val);

  val = await asynchronousOperation(val);

  return await asynchronousOperation(val);

};

* + 注意
    - 最后的 return 语句中都不需要使用 await，因为用或不用，它都返回了包含了可处理终值的 Promise。
* 3. 并发操作

async function doAsyncOp() {

  var vals = await Promise.all([

    asynchronousOperation(),

    asynchronousOperation()

  ]);

  vals.forEach(console.log.bind(console));

  return vals;

}

* 4.处理拒绝

async function doAsyncOp () {

  try {

    var val = await asynchronousOperation();

    val = await asynchronousOperation(val);

    return await asynchronousOperation(val);

  } catch (err) {

    console.err(err);

  }

};

* + 它不像其它往 async 函数的转换那样简洁，但是确实跟写同步代码一样。如果你在这里不捕捉错误，它会延着调用链一直向上抛出，直到在某处被捕捉处理。如果它一直未被捕捉，它最终会中止程序并抛出一个运行时错误。
* 5.嵌套函数的问题

async function getAllFiles(fileNames) {

  return Promise.all(

    fileNames.map(**async**function(fileName) {

      var file = **await**getFileAsync(fileName);

      return parse(file);

    })

  );

}

* + 可以通过为回调函数添加 async 关键字来解决这个问题。