<button onclick="bgChange()">Press me</button>

在Web上注册事件处理程序的最早方法是类似于上面所示的**事件处理程序HTML属性**(也称为内联事件处理程序)—属性值实际上是当事件发生时要运行的JavaScript代码。

btn.addEventListener('click', bgChange);

您也可以给同一个监听器注册多个处理器

可以使用removeEventListener高效地清除不用的事件处理器

有时候在事件处理函数内部，您可能会看到一个固定指定名称的参数，例如event，evt或简单的e。这被称为**事件对象**，它被自动传递给事件处理函数

当你填写详细信息并按提交按钮时，自然行为是将数据提交到服务器上的指定页面进行处理，并将浏览器重定向到某种“成功消息”页面（或 相同的页面，如果另一个没有指定。）

form.onsubmit = function(e) {

if (fname.value === '' || lname.value === '') {

e.preventDefault();

para.textContent = 'You need to fill in both names!';

}

}

video是<div>的一个子元素 - 所以点击video实际上是同时也运行<div>上的事件处理程序。

**对事件冒泡和捕捉的解释**

当一个事件发生在具有父元素的元素上(例如，在我们的例子中是<video>元素)时，现代浏览器运行两个不同的阶段 - 捕获阶段和冒泡阶段。 在捕获阶段：

* 浏览器检查元素的最外层祖先<html>，是否在捕获阶段中注册了一个onclick事件处理程序，如果是，则运行它。
* 然后，它移动到<html>中的下一个元素，并执行相同的操作，然后是下一个元素，依此类推，直到到达实际点击的元素。

在冒泡阶段，恰恰相反:

* 浏览器检查实际点击的元素是否在冒泡阶段中注册了一个onclick事件处理程序，如果是，则运行它
* 然后它移动到下一个直接的祖先元素，并做同样的事情，然后是下一个，等等，直到它到达<html>元素。

在现代浏览器中，默认情况下，所有事件处理程序都在冒泡阶段进行注册。

video.onclick = function(e) {

e.stopPropagation();

video.play();

};

#### 事件委托

冒泡还允许我们利用事件委托—— 如果你想要在大量子元素中单击任何一个都可以运行一段代码，您可以将事件监听器设置在其父节点上

document.getElementById("parent-list").addEventListener("click", function(e) {

// e.target is the clicked element!

// If it was a list item

if(e.target && e.target.nodeName == "LI") {

// List item found! Output the ID!

console.log("List item ", e.target.id.replace("post-", ""), " was clicked!");

}

});

document.getElementById("myDiv").addEventListener("click",function(e) {

// e.target was the clicked element

if (e.target && e.target.matches("a.classA")) {

console.log("Anchor element clicked!");

}

});

理解对象的原型与构造函数的prototype属性之间的区别是很重要的。前者是每个实例上都有的属性，后者是构造函数的属性。也就是说，Object.getPrototypeOf(new Foobar())和Foobar.prototype指向着同一个对象。

在javascript中，函数可以有属性。 每个函数都有一个特殊的属性叫作原型（prototype）

JavaScript中的所有对象都来自Object；所有对象从[Object.prototype](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/prototype" \o "Object.prototype 属性表示 Object 的原型对象。)继承方法和属性

[Object.create()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/create)

var person2 = Object.create(person1);

以 person1 为原型对象创建了 person2 对象。在控制台输入：

person2.\_\_proto\_\_

结果返回对象person1。

function Teacher(first, last, age, gender, interests, subject) {

Person.call(this, first, last, age, gender, interests);

this.subject = subject;

}

Teacher.prototype = Object.create(Person.prototype);

Teacher.prototype.constructor = Teacher;

Teacher.prototype.greeting = function() {

}

ECMAScript 2015 中引入的 JavaScript 类实质上是 JavaScript 现有的基于原型的继承的语法糖类语法有两个组成部分：[类表达式](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/class)和[类声明](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/class)。

### 类声明

定义一个类的一种方法是使用一个**类声明**。要声明一个类，你可以使用带有class关键字的类名（这里是“Rectangle”）。

class Rectangle {

constructor(height, width) {

this.height = height;

this.width = width;

}

}

**函数声明**和**类声明**之间的一个重要区别是函数声明会[提升](https://developer.mozilla.org/en-US/docs/Glossary/Hoisting)，类声明不会。你首先需要声明你的类，然后访问它，否则像下面的代码会抛出一个[ReferenceError](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError)：

let p = new Rectangle();

// ReferenceError

class Rectangle {}

### 类表达式

一个**类表达式**是定义一个类的另一种方式。类表达式可以是被命名的或匿名的。赋予一个命名类表达式的名称是类的主体的本地名称

/\* 匿名类 \*/

let Rectangle = class {

constructor(height, width) {

this.height = height;

this.width = width;

}

};

/\* 命名的类 \*/

let Rectangle = class Rectangle {

constructor(height, width) {

this.height = height;

this.width = width;

}

};

**注意:** 类**表达式**也同样受到类**声明**中提到的提升问题的困扰。

[constructor](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/constructor)方法是一个特殊的方法，其用于创建和初始化使用class创建的一个对象。一个类只能拥有一个名为 “constructor”的特殊方法

一个构造函数可以使用 super关键字来调用一个父类的构造函数。

[static](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Classes/static) 关键字用来定义一个类的一个静态方法。调用静态方法不需要[实例化](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript" \l "The_object_(class_instance))该类

**注:** 考虑到JavaScript的工作方式，由于原型链等特性的存在，在不同对象之间功能的共享通常被叫做 **委托** - 特殊的对象将功能委托给通用的对象类型完成。这也许比将其称之为继承更为贴切，因为“被继承”了的功能并没有被拷贝到正在“进行继承”的对象中，相反它仍存在于通用的对象中。

JavaScript对象表示法（JSON）是用于将结构化数据表示为JavaScript对象的标准格式

然它是基于 JavaScript 语法，但它独立于JavaScript，这也是为什么许多程序环境能够读取（解读）和生成 JSON。

JSON可以作为一个对象或者字符串存在，前者用于解读 JSON 中的数据，后者用于通过网络传输 JSON 数据。

 JSON 可以将任何标准合法的 JSON 数据格式化保存，不只是数组和对象。比如，一个单一的字符串或者数字可以是合法的 JSON 对象。虽然不是特别有用处……

 不像 JavaScript 标识符可以用作属性，在 JSON 中，只有字符串才能用作属性。

## 对象和文本间的转换

request.onload = function() {

var superHeroes = request.response;

populateHeader(superHeroes);

}

function populateHeader(jsonObj) {

var myH1 = document.createElement('h1');

myH1.textContent = jsonObj['squadName'];

header.appendChild(myH1);

}

上述示例就访问 JSON 而言是简单的，因为我们设置了 XHR 来访问 JSON 格式数据：

request.responseType = 'json';

但是有时候我们没有那么幸运，我们接收到一些 字符串作为 JSON 数据，然后我们想要将它转换为对象。当我们想要发送 JSON 数据作为信息，我们将需要转换它为字符串，我们经常需要正确的转换数据，

var myJSON = { "name" : "Chris", "age" : "38" };

myJSON

var myString = JSON.stringify(myJSON);

myString

这儿我们创建了一个JavaScript 对象，然后检查了它包含了什么，然后用stringify() 将它转换成JSON字符串，最后保存返回值作为变量。然后再一次检查。

### JSON 结构

{

"squadName" : "Super hero squad",

"homeTown" : "Metro City",

}

### JSON 数组

[

{

"name" : "Molecule Man",

"age" : 29,

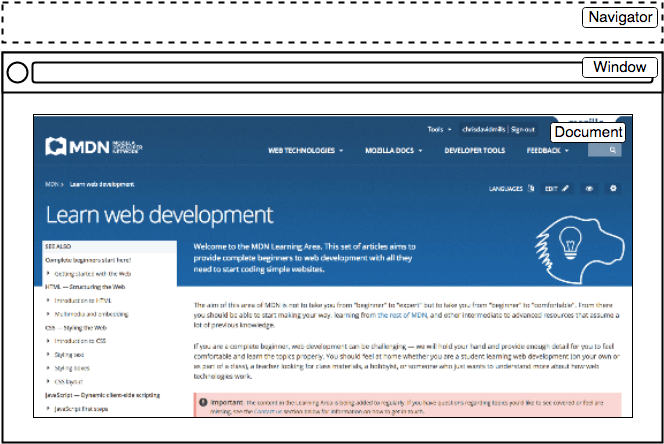
}

]

客户端JavaScript中有很多可用的API — 他们本身并不是JavaScript语言的一部分，却建立在JavaScript语言核心的顶部，为使用JavaScript代码提供额外的超强能力。他们通常分为两类：

**浏览器API**内置于Web浏览器中，能从浏览器和电脑周边环境中提取数据，并用来做有用的复杂的事情 。

**第三方API**缺省情况下不会内置于浏览器中，通常必须在Web中的某个地方获取代码和信息。例如[Twitter API](https://dev.twitter.com/overview/documentation) 使您能做一些显示最新推文这样的事情，它提供一系列特殊的结构，可以用来请求Twitter服务并返回特殊的信息。



* window是载入浏览器的标签，在JavaScript中用[Window](https://developer.mozilla.org/zh-CN/docs/Web/API/Window" \o "The window object represents a window containing a DOM document; the document property points to the DOM document loaded in that window.)对象来表示，使用这个对象的可用方法，你可以返回窗口的大小（参见[Window.innerWidth](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/innerWidth" \o "浏览器视口（viewport）宽度（单位：像素），如果存在垂直滚动条则包括它。)和[Window.innerHeight](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/innerHeight" \o "浏览器窗口的视口（viewport）高度（以像素为单位），如果存在水平滚动条，则包括它。)），操作载入窗口的文档，存储客户端上文档的特殊数据（例如使用本地数据库或其他存储设备），为当前窗口绑定[event handler](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events" \l "A_series_of_fortunate_events)，等等。
* navigator表示浏览器存在于web上的状态和标识（即用户代理）。在JavaScript中，用[Navigator](https://developer.mozilla.org/zh-CN/docs/Web/API/Navigator" \o "Navigator 接口表示用户代理的状态和标识。 它允许脚本查询它和注册自己进行一些活动。)来表示。你可以用这个对象获取一些信息，比如来自用户摄像头的地理信息、用户偏爱的语言、多媒体流等等。
* document（在浏览器中用DOM表示）是载入窗口的实际页面，在JavaScript中用[Document](https://developer.mozilla.org/zh-CN/docs/Web/API/Document) 对象表示，你可以用这个对象来返回和操作文档中HTML和CSS上的信息。例如获取DOM中一个元素的引用，修改其文本内容，并应用新的样式，创建新的元素并添加为当前元素的子元素，甚至把他们一起删除。

## 基本的DOM 操作

var link = document.querySelector('a');

还有一些更旧的方法，如：

[Document.getElementById()](https://developer.mozilla.org/zh-CN/docs/Web/API/Document/getElementById)，

[Document.getElementsByTagName()](https://developer.mozilla.org/zh-CN/docs/Web/API/Document/getElementsByTagName)，

### 创建并放置新的节点

var para = document.createElement('p');

para.textContent = 'We hope you enjoyed the ride.';

sect.appendChild(para);

在内部链接的段落中添加文本节点

var text = document.createTextNode(' — the premier source for web development knowledge.');

### 移动和删除元素

sect.removeChild(linkPara);

要删除一个仅基于自身引用的节点可能稍微有点复杂，这也是很常见的。没有方法会告诉节点删除自己，所以你必须像下面这样操作

linkPara.parentNode.removeChild(linkPara);

para.style.color = 'white';

para.setAttribute('class', 'highlight');

## 从Window对象中获取有用的信息

var WIDTH = window.innerWidth;

var HEIGHT = window.innerHeight;

然后获取视窗（显示文档的内部窗口）的宽度和高度

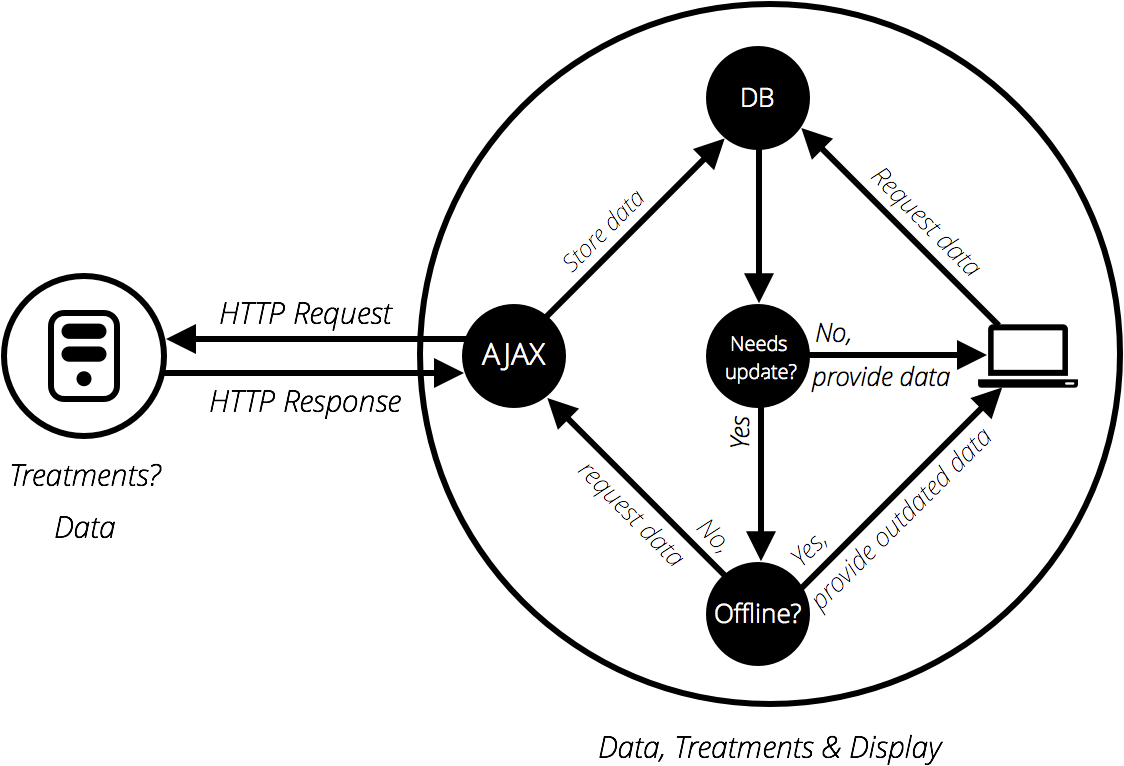
window.onresize = function() {

}

# 从服务器获取数据

**在早期，这种通用技术被称为**Asynchronous JavaScript and XML**（Ajax），** 因为它倾向于使用[XMLHttpRequest](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest) 来请求XML数据。 但通常不是这种情况 (你更有可能使用 XMLHttpRequest 或 Fetch 来请求JSON), 但结果仍然是一样的，术语“Ajax”仍然常用于描述这种技术。

为了进一步提高速度，有些网站还会在首次请求时将资产和数据存储在用户的计算机上，这意味着在后续访问中，他们将使用本地版本，而不是在首次加载页面时下载新副本。 内容仅在更新后从服务器重新加载



Fetch API基本上是XHR的一个现代替代品

var request = new XMLHttpRequest();

request.open('GET', url);

request.responseType = 'text';

request.onload = function() {

poemDisplay.textContent = request.response;

};

request.send();

等效于

fetch(url).then(function(response) {

response.text().then(function(text) {

poemDisplay.textContent = text;

});

});

当fetch() promise 解析时，这个函数会自动将响应从服务器传递给参数。在函数内部，我们获取响应并运行其text()方法。这基本上将响应作为原始文本返回，这相当于在XHR版本中的responseType = 'text'。

## 客户端存储

经常客户端和服务端存储是结合在一起使用的。例如，你可以从数据库中下载一个由网络游戏或音乐播放器应用程序使用的音乐文件，将它们存储在客户端数据库中，并按需要播放它们。用户只需下载音乐文件一次——在随后的访问中，它们将从数据库中检索

### 新流派：Web Storage 和 IndexedDB

### 未来：Cache API

一些现代浏览器支持新的 [Cache](https://developer.mozilla.org/zh-CN/docs/Web/API/Cache) API。这个API是为存储特定HTTP请求的响应文件而设计的，它对于像存储离线网站文件这样的事情非常有用，这样网站就可以在没有网络连接的情况下使用

[sessionStorage](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/sessionStorage) 和 [localStorage](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/localStorage)。 第一种方法，只要浏览器开着，数据就会一直保存 (关闭浏览器时数据会丢失) ，而第二种会一直保存数据，甚至到浏览器关闭又开启后也是这样。

### 为每个域名分离储存

每个域都有一个单独的数据存储

 <script src="index.js" defer></script> 一行里， defer 属性指明在页面加载完成之前，[<script>](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/script" \o "HTML <script> 元素用于嵌入或引用可执行脚本。)元素的内容不会执行。

你还可以使用  [localStorage.length](https://developer.mozilla.org/zh-CN/docs/Web/API/Storage/length) 来测试存储对象是否为空。

## 通过 StorageEvent 响应存储的变化

在同一个页面内发生的改变不会起作用——在相同域名下的其他页面（如一个新标签或 iframe）发生的改变才会起作用

window.addEventListener('storage', function(e) {

document.querySelector('.my-key').textContent = e.key;

document.querySelector('.my-old').textContent = e.oldValue;

document.querySelector('.my-new').textContent = e.newValue;

document.querySelector('.my-url').textContent = e.url;

document.querySelector('.my-storage').textContent = e.storageArea;

});

undefined 值在布尔类型环境中会被当作 false。

数值类型环境中 undefined 值会被转换为 NaN。

当你对一个 [null](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/null) 变量求值时，空值 null 在数值类型环境中会被当作0来对待，而布尔类型环境中会被当作 false。例如：

var n = null;

typeof(n);

// "object"

// The Null type has exactly one value, called null.

console.log(n \* 32); // 0

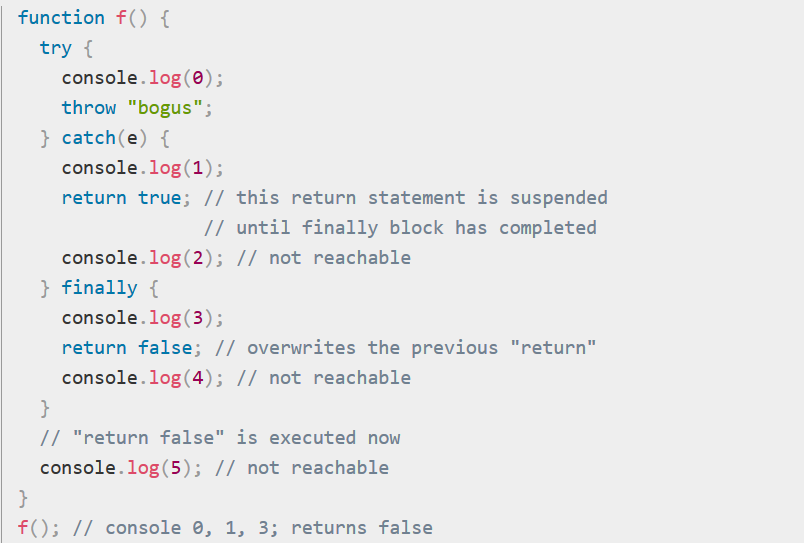
#### 单目加法运算符

将字符串转换为数字的另一种方法是使用**单目加法运算符**。

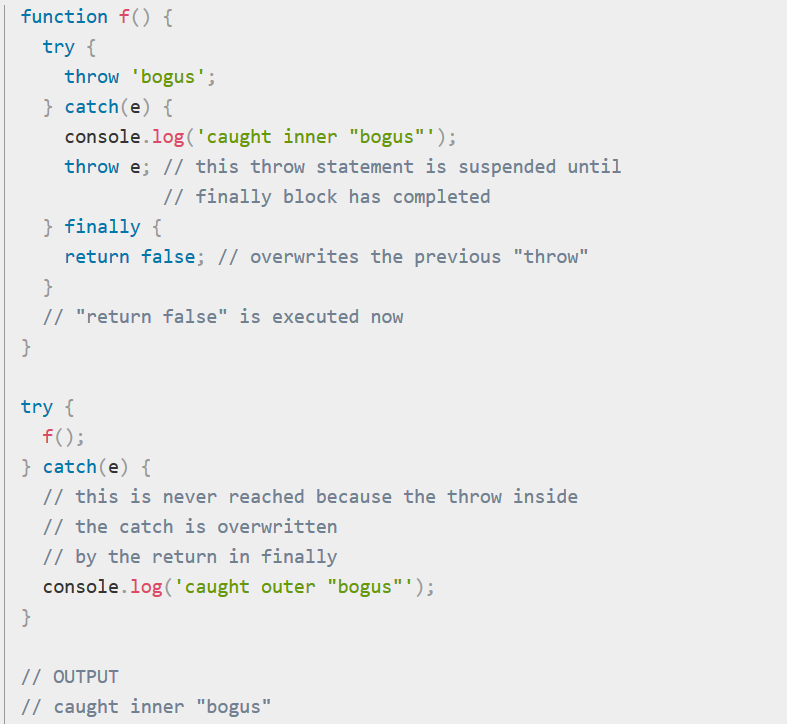
"1.1" + "1.1" = "1.11.1"

(+"1.1") + (+"1.1") = 2.2 // 注：加入括号为清楚起见，不是必需的。

如果finally块返回一个值，该值会是整个try-catch-finally流程的返回值，不管在try和catch块中语句返回了什么：



用finally块覆盖返回值也适用于在catch块内抛出或重新抛出的异常：



在 JavaScript 中，可以根据条件来定义一个函数。比如下面的代码，当num 等于 0 的时候才会定义 myFunc ：

var myFunc;

if (num == 0){

myFunc = function(theObject) {

theObject.make = "Toyota"

}

}

除了上述的定义函数方法外，你也可以在运行时用 [Function](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Function) 构造器由一个字符串来创建一个函数 ，很像 [eval()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/eval()) 函数。

函数一定要处于调用它们的域中，但是函数的声明可以被提升(出现在调用语句之后)，如下例：

函数域是指函数声明时的所在的地方，或者函数在顶层被声明时指整个程序。

调用自身的函数我们称之为递归函数

var foo = function bar() {

// statements go here

};

在这个函数体内，以下的语句是等价的：

1. bar()
2. arguments.callee()
3. foo()

[箭头函数表达式](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Functions/Arrow_functions)（也称胖箭头函数）相比函数表达式具有较短的语法并以词法的方式绑定 this

function Person() {

// The Person() constructor defines `this` as itself.

this.age = 0;

setInterval(function growUp() {

// In nonstrict mode, the growUp() function defines `this`

// as the global object, which is different from the `this`

// defined by the Person() constructor.

this.age++;

}, 1000);

}

var p = new Person();

一个 [Promise](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise) 就是一个代表了异步操作最终完成或者失败的对象。大多数人都在使用由其他函数创建并返回的 Promise，因此，本教程将首先探讨返回 Promise 的使用情况。

Promise 本质上是一个绑定了回调的对象，而不是将回调传进函数内部。

const promise2 = doSomething().then(successCallback, failureCallback);

第二个对象(promise2)不仅代表doSomething()函数的完成，也代表了你传入的 successCallback 或者failureCallback 的完成，这也可能是其他异步函数返回的 Promise。这样的话，任何被添加给 promise2 的回调函数都会被排在 successCallback 或 failureCallback 返回的 Promise 后面。

基本上，一个 Promise 链式遇到异常就会停止，查看链式的底端，寻找catch处理程序来代替当前执行。在同步的代码执行之后，这是非常模型化的。