

让状态来影响行为

添加属性fuel后,就可实现一些有趣的行为了。例如,如果没油,车就开不动!下面首先来调整方法drive,在其中检查油量以确定油箱不是空的,并在每次开动时都将fuel减1。实现这些行为的代码如下:

```
var fiat = {
  // 其他属性和方法
  drive: function() {
    if (this.started) {
      if (this.fuel > 0) {
        alert(this.make + " " +
              this.model + " goes zoom zoom!");
        this.fuel = this.fuel - 1;
      } else {
        alert("Uh oh, out of fuel.");
        this.stop();
      }
    } else {
      alert("You need to start the engine first.");
    }
  },
  addFuel: function(amount) {
    this.fuel = this.fuel + amount;
  }
};
```

现在可以在开动汽车前进行检查,确认油箱不是空的。另外,如果能够开动汽车,就应在每次开动时都减少余下的油量。

如果没油了,就现实一条消息并熄灭发动机。要再次开动汽车,你必须加油并启动发动机。

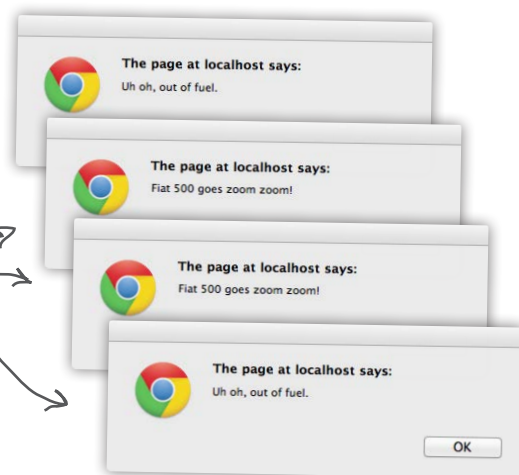
加点油以便试驾



请修改你的代码,并尝试运行它们!这是我们使用下面的测试代码得到的结果:

```
fiat.start();
fiat.drive();
fiat.addFuel(2);
fiat.start();
fiat.drive();
fiat.drive();
fiat.drive();
fiat.stop();
```

首先,我们尝试在没油的情况下开动汽车;然后,我们加了点油,再不断地开,直到油耗尽!请尝试添加自己的测试代码,并确保它们像你预期的那样工作。





要在汽车对象中完整地实现属性fuel，还有一些工作要做。例如，没油时能够启动发动机吗？请看方法start的代码：

```
start: function() {  
    this.started = true;  
}
```

显然是可以的。

请完善这个方法，在其中检查油位，并在没油时使用简单的提示（如The car is on empty, fill up before starting!）告知驾驶员。在下面重写方法start，将其加入你的代码中，并进行测试。继续往下阅读前，请查看本章末尾的答案。

在这里编写代码。



请查看对象fiat的代码，你还能在其他地方根据属性fuel来调整汽车的行为或创建修改属性fuel的行为吗？将你的想法记录在下面。



祝贺你熟悉了对象

你完成了有关对象的第一章，可以继续前行了。还记得你刚接触JavaScript时的情形吗？你眼里的编程领域就是数字、字符串、语句、条件、for循环等。现在大不相同了，你从更高的层次思考问题，眼里看到的是对象和方法。看看下面的代码就明白了：

```
fiat.addFuel(2);  
fiat.start();  
fiat.drive();  
fiat.stop();
```

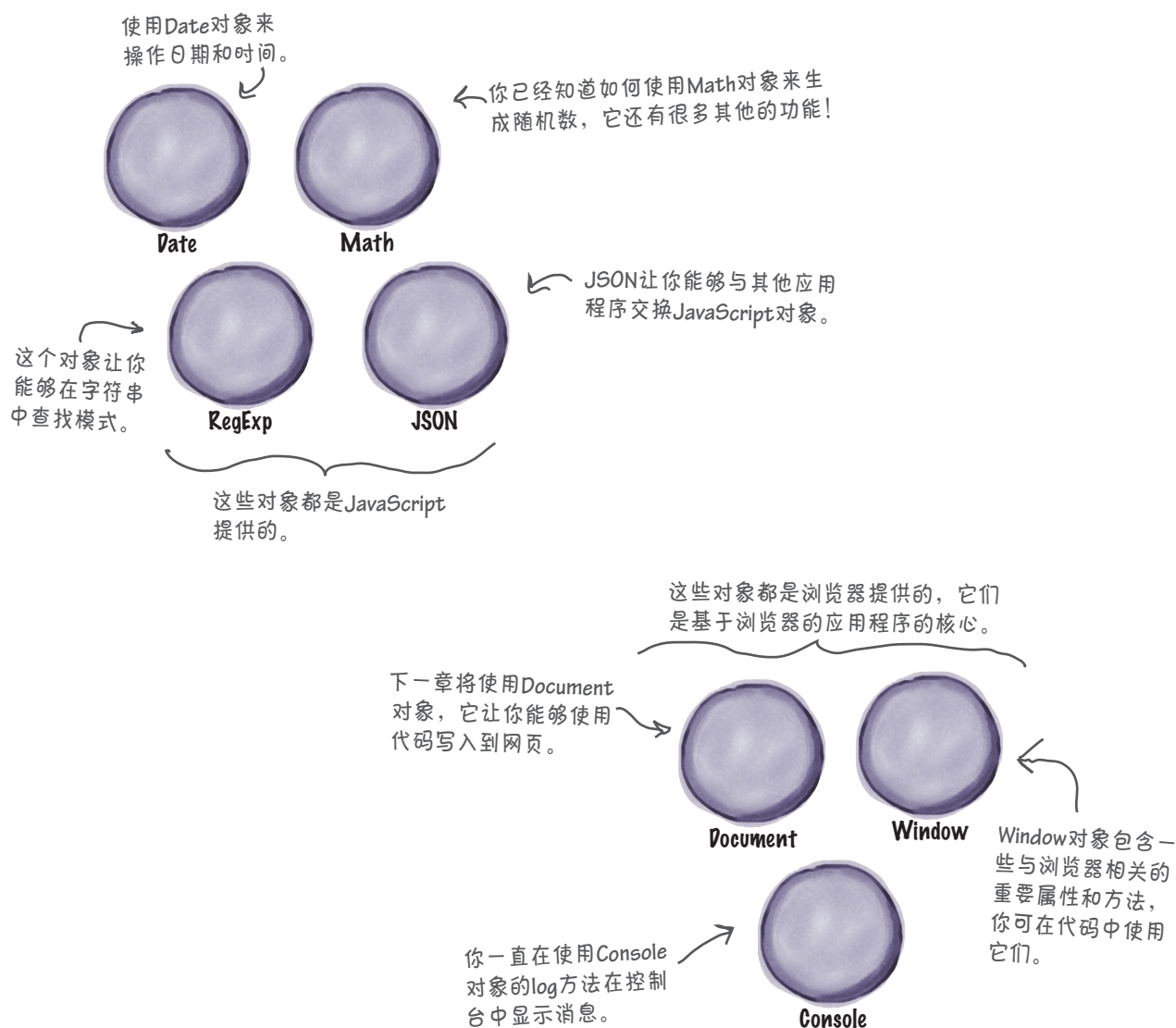
这些代码很容易理解，因为它使用包含状态和行为的对象来描绘世界。

这仅仅是开始，你可以走得更远，也确实会走得更远。熟悉对象后，接下来我们将继续提高你的技能，让你能够使用其他的JavaScript特性编写出真正的面向对象代码，并遵循众多至关重要的最佳实践。

结束本章前，还有一点你必须知道。

对象无处不在 (它们让你的生活更美好)

对对象略知一二后，通向一个全新世界的大门就向你打开了，因为JavaScript提供了很多对象（用于执行数学计算的对象、用于操作字符串的对象、用于创建日期和时间的对象，等等），你可在代码中使用它们。JavaScript还提供了一些编写浏览器代码时需要的重要对象，下一章将介绍其中的一个。现在花点时间来熟悉一些这样的对象，本书后面将经常用到它们。





起底对象

本周访谈：
且看对象自己怎么说

Head First: 欢迎对象，这一章非常精彩。从对象的角度考虑代码，真是让人醍醐灌顶。

对象: 哦，是吗？一切才刚刚开始。

Head First: 何出此言？

对象: 对象是一组属性，是吧？有些属性用于存储对象的状态，有些实际上是函数（准确地说是方法），赋予对象以行为。

Head First: 这些我都明白。我实际上没有将方法也视为属性，但如果能够将函数也视为值的话，方法也是由名称和值组成的。

对象: 可以将函数视为值！相信我，你可以。事实上，你可能没意识到，这种想法很有见地。先讲到这儿吧，我猜你还有很多问题要问。

Head First: 但是你刚才说到……

对象: 好吧，你见过很多包含属性的对象，还创建了很多对象，如各种汽车。

Head First: 没错。

对象: 但这些对象都太随意。真正的威力在于，你能够创建模板，用于生成统一的对象。

Head First: 哦，你是说生成的对象都是相同的类型。

对象: 大致是这个意思。类型是一个有趣的JavaScript概念。你将发现，当你能够编写处理相同类型对象的代码时，便能充分发挥对象的威力。例如，你可以编写处理车辆的代码，而无需关心它们是自行车、小汽车还是公交车。这就是对象的威力所在。

Head First: 听起来很有意思。为此，我们还需要知道什么？

对象: 你必须更深入地了解对象，还需要一种创建相同类型对象的方法。

Head First: 我们已经这样做了，不是吗？所有汽车对象不都是相同类型的吗？

对象: 它们确实差不多是相同类型的，但这是因为用来创建这些汽车的代码类似。换句话说，它们包含相同的属性和方法。

Head First: 没错，事实上，我们提到过这些对象的很多代码都相同。从维护的角度说，这未必是件好事。

对象: 接下来，你需要学习如何使用相同的代码创建完全相同的对象，而且这些代码只出现在一个地方。这将涉及如何设计面向对象的代码。你已经为此作好了充分准备，因为你已经掌握了基本的对象知识。

Head First: 听到你这么说，读者一定很高兴。

对象: 但还有一些有关对象的情况你必须知道。

Head First: 是吗？

对象: 你可在代码中直接使用很多现成的对象。

Head First: 真的吗？这我倒是没有注意到，在哪里呢？

对象: 注意到了`console.log`吗？你认为`console`是什么？

Head First: 考虑到这里讨论的是对象，我猜它是一个对象。

对象: 你猜对了。`log`呢？

Head First: 属性……哦不，方法？

对象: 你猜对了。`alert`呢？

Head First: 不知道。

对象: 它与对象有关，咱们以后再说。

Head First: 关于对象，你真是让我们大长见识，期望你再次接受采访。

对象: 一定。

Head First: 太好了！咱们下次再见。



密码破解

在试图统治地球的过程中，Evel博士不小心将包含操作密码的内部网页放到了网上。如果能获得这个密码，我们就能占据上风。当然，发现这个网页被放在网上后，Evel博士马上就把它删除了。好在我们的特工人员将这个网页记录了下来，但问题是他们不懂HTML和JavaScript。你能根据下面的代码帮忙找出密码吗？别忘了，这关乎地球的存亡。

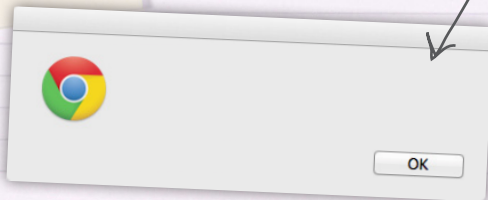
```
var access =  
    document.getElementById("code9");  
var code = access.innerHTML;  
code = code + " midnight";  
alert(code);
```

这是JavaScript代码。

这些代码好像使用了
document对象。

提示框显示的密码会是
什么？

请将答案写在下面
的提示框中。



这是HTML代码。

```
<!doctype html>  
<html lang="en">  
  <head>  
    <meta charset="utf-8">  
    <title>Dr. Evel's Secret Code Page</title>  
  </head>  
  <body>  
    <p id="code1">The eagle is in the</p>  
    <p id="code2">The fox is in the</p>  
    <p id="code3">snuck into the garden last night.</p>  
    <p id="code4">They said it would rain</p>  
    <p id="code5">Does the red robin crow at</p>  
    <p id="code6">Where can I find Mr.</p>  
    <p id="code7">I told the boys to bring tea and</p>  
    <p id="code8">Where's my dough? The cake won't</p>  
    <p id="code9">My watch stopped at</p>  
    <p id="code10">barking, can't fly without umbrella.</p>  
    <p id="code11">The green canary flies at</p>  
    <p id="code12">The oyster owns a fine</p>  
    <script src="code.js"></script>  
  </body>  
</html>
```

在这里链接到了上面的
JavaScript代码。

绝密



如果你跳过了前一页，请回过头去破解密码。这对第6章来说至关重要。



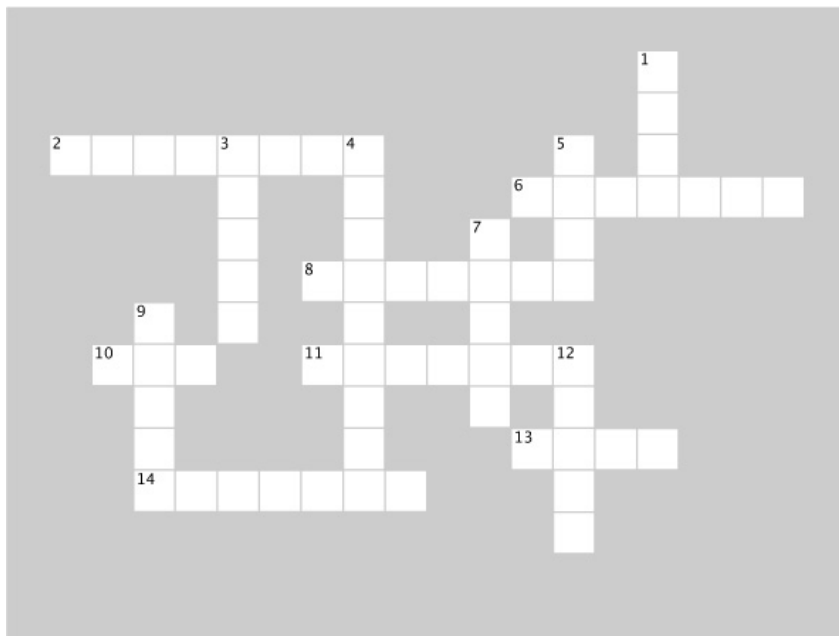
要点

- 对象是一系列属性。
- 要访问属性，可使用句点表示法：包含对象的变量名称、句点和属性名。
- 可随时给对象添加属性，为此只需给要添加的属性赋值。
- 你还可删除对象的属性，为此可使用 delete 运算符。
- 基本类型变量存储了字符串、数字或布尔值等实际值；对象变量不同，它存储的并不是对象本身，而是指向对象的引用。因此对象变量也被称为引用变量。
- 向函数传递对象时，函数获得的是指向该对象的引用的副本，而不是对象本身的副本。因此，如果在函数中修改属性的值，修改的将是原始对象的值。
- 对象的属性可以是函数。对象中的函数被称为方法。
- 使用句点表示法来调用方法：对象名、句点、方法的属性名和圆括号。
- 方法与函数没什么两样，只是位于对象中。
- 就像可以向常规函数传递实参一样，也可向方法传递实参。
- 调用对象的方法时，关键字 `this` 指向其方法被调用的对象。
- 要在对象的方法中访问对象的属性，必须使用句点表示法，但使用关键字 `this` 而不是对象名。
- 在面向对象编程中，我们从对象而不是过程的角度思考问题。
- 对象包含状态和行为。状态可能影响行为，而行为也可能影响状态。
- 对象封装（隐藏）了其状态和行为的复杂性。
- 设计良好的对象包含负责完成相关任务的方法，让你无需操心与完成这些任务相关的细节。
- 除了自己创建的对象外，JavaScript 也提供了很多内置对象供你使用。本书后面将用到很多这样的内置对象。



JavaScript填字游戏

填字游戏对象是什么样的呢？它有很多线索属性，可帮助你明白要达成的目标。



横向

2. 方法赋予对象什么？
6. 方法log是哪个对象的一个属性？
8. this是一个_____，而不是常规变量。
10. 访问对象的属性时使用的表示法。
11. 它们像常规函数一样，可以有局部变量和形参。
13. 用于表示汽车对象品牌的属性。
14. 给汽车对象加油，进而影响其状态的方法。

纵向

1. 访问属性started时没有使用哪个关键字，导致菲亚特无法开动？
3. 对象引用是以什么方式传递给函数的，就像基本类型变量一样？
4. 将对象赋给变量时，变量存储的是指向对象的什么？
5. 通常将属性名指定为一个什么？
7. 在对象中，将属性的名称和值用什么分隔？
9. 在每个属性值（最后一个除外）的后面，都需要加上什么？
12. 汽车和小狗对象都可以有行为和什么？



我们要制作一个表格，在其中列出汽车属性的名称和值。你能帮助我们完成这项任务吗？答案如下。

在这里列出属
性的名称。

在这里列出属性的值。

```
{
  make      : "Chevy",
  model     : "Bel Air",
  year      : 1957,
  color     : "red",
  passengers : 2,
  convertible : false,
  mileage   : 1021,
  accessories : "Fuzzy Dice",
  whitewalls : true
};
```

我们在合适的地方使用了
字符串、布尔值和数字。

将答案写在这里。你
可随意拓展这个列表，
在其中添加你想到的
其他属性。



请利用新学到的有关this的知识补全下面的代码。答案如下。

```
var eightBall = { index: 0,
  advice: ["yes", "no", "maybe", "not a chance"],
  shake: function() {
    this.index = this.index + 1;
    if (this.index >= this.advice.length) {
      this.index = 0;
    }
  },
  look: function() {
    return this.advice[this.index];
  }
};

eightBall.shake();
console.log(eightBall.look());
```

重复执行这些代码多次，对
你补全的代码进行测试。

JavaScript控制台

```
no
maybe
not a chance
```



并非只能使用一个对象。稍后你将看到，对象的真正威力在于，你可以创建大量的对象，并编写代码来操作提供给它的对象。来练练手，从头开始创建另一个汽车对象，并将其代码写在下面。答案如下。

```
var cadi = {  
  make: "GM",  
  model: "Cadillac",  
  year: 1955,  
  color: "tan",  
  passengers: 5,  
  convertible: false,  
  mileage: 12892  
};
```

← 在这里定义对象的属性。

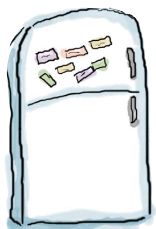
它不是敞篷车，最多可乘坐5人（后排是一个超大的凹背座椅）。

这是一辆通用1955年
产的凯迪拉克。

车身为棕褐色。

里程为12 892英里。





对象冰箱贴答案

请利用你的对象创建和句点表示法技能，使用冰箱贴将下面的代码补全。请注意，其中有些冰箱贴可能是多余的！答案如下。



多出来的冰箱贴。



小狗对象

```
var dog = {
  name: "Fido",
  weight : 20.2,
  age: 4,
  breed : "mixed",
  activity: "fetch balls"
};
```

```
var bark;
```

```
if (dog.weight > 20) {
  bark = "WOOF WOOF";
} else {
```

```
  bark = "woof woof";
}
```

```
var speak = dog.name + " says " + dog.bark + " when he wants to " + dog.activity;
console.log(speak);
```



Fido渴望你将
其属性归位。



```
var cadi = {
  make: "GM",
  model: "Cadillac",
  year: 1955,
  color: "tan",
  passengers: 5,
  convertible: false,
  mileage: 12892
};

prequal(cadi);
```

false

↗
在这里填写
函数prequal
返回的值。

轮到你了。这里还有三个汽车对象；将它们分别传递给函数prequal时，结果如何呢？请先通过心算来回答，再编写代码来检查你的答案是否正确。答案如下。



```
var fiat = {
  make: "Fiat",
  model: "500",
  year: 1957,
  color: "Medium Blue",
  passengers: 2,
  convertible: false,
  mileage: 88000
};

prequal(fiat);
```

false



```
var chevy = {
  make: "Chevy",
  model: "Bel Air",
  year: 1957,
  color: "red",
  passengers: 2,
  convertible: false,
  mileage: 1021
};

prequal(chevy);
```

true



有人给你提供了一个绝密文件，还有两个让你能够获取和设置该文件内容的函数，但要使用这两个函数，你必须知道正确的密码。第一个函数是`getSecret`，它在密码正确时返回文件的内容，并将每次访问尝试都记录在案。第二个函数是`setSecret`，它修改文件的内容，并将访问次数重置为零。你的任务是补全下面的JavaScript代码，并对完成后的函数进行测试。



```
function getSecret(file, secretPassword) {
  file.opened = file.opened + 1;
  if (secretPassword == file.password) {
    return file.contents;
  }
  else {
    return "Invalid password! No secret for you.";
  }
}
```

向函数`getSecret`传递了对象`superSecretFile`，该对象被赋给形参`file`。因此，使用句点表示法访问该对象的属性（如`opened`和`password`）时，必须将对象名指定为`file`。

```
function setSecret(file, secretPassword, secret) {
  if (secretPassword == file.password) {
    file.opened = 0;
    file.contents = secret;
  }
}
```

道理与前面一样。

```
var superSecretFile = {
  level: "classified",
  opened: 0,
  password: 2,
  contents: "Dr. Evel's next meeting is in Detroit."
};
```

```
var secret = getSecret(superSecretFile, 2);
console.log(secret);
```

可将对象`superSecretFile`传递给函数`getSecret`和`setSecret`。

```
setSecret(superSecretFile, 2, "Dr. Evel's next meeting is in Philadelphia.");
secret = getSecret(superSecretFile, 2);
console.log(secret);
```

变身浏览器答案

下面的JavaScript代码存在一些错误，你的任务是变身浏览器，将这些错误找出来。答案如下。



```
var song = {
  name: "Walk This Way",
  artist: "Run-D.M.C.",
  minutes: 4,
  seconds: 3,
  genre: "80s",
  playing: false,

  play: function() {
    if (!this.playing) {
      this.playing = true;
      console.log("Playing "
        + this.name + " by " + this.artist);
    }
  },

  pause: function() {
    if (this.playing) {
      this.playing = false;
    }
  }
};

this song.play();
this song.pause();
```

这里少了关键字this。

这里少了属性名playing。

访问这两个属性时，都需要使用关键字this。

访问属性playing时，也需要使用关键字this。

在方法外面引用对象时，不使用关键字this，而使用其变量名。



该把整个车队拉出来“遛遛”了。给每个汽车对象都添加方法drive，然后添加对它们进行发动、驾驶和熄火的代码。答案如下。



```
var cadi = {
  make: "GM",
  model: "Cadillac",
  year: 1955,
  color: "tan",
  passengers: 5,
  convertible: false,
  mileage: 12892,
  started: false,
  start: function() {
    this.started = true;
  },
  stop: function() {
    this.started = false;
  },
  drive: function() {
    if (this.started) {
      alert(this.make + " " +
        this.model + " goes zoom zoom!");
    } else {
      alert("You need to start the engine first.");
    }
  }
};
```



```
var chevy = {
  make: "Chevy",
  model: "Bel Air",
  year: 1957,
  color: "red",
  passengers: 2,
  convertible: false,
  mileage: 1021,
  started: false,
  start: function() {
    this.started = true;
  },
  stop: function() {
    this.started = false;
  },
  drive: function() {
    if (this.started) {
      alert(this.make + " " +
        this.model + " goes zoom zoom!");
    } else {
      alert("You need to start the engine first.");
    }
  }
};
```



```
var taxi = {
  make: "Webville Motors",
  model: "Taxi",
  year: 1955,
  color: "yellow",
  passengers: 4,
  convertible: false,
  mileage: 281341,
  started: false,
  start: function() {
    this.started = true;
  },
  stop: function() {
    this.started = false;
  },
  drive: function() {
    if (this.started) {
      alert(this.make + " " +
        this.model + " goes zoom zoom!");
    } else {
      alert("You need to start the engine first.");
    }
  }
};
```

务必在添加的
每个属性后面
加上逗号。

```
cadi.start();
cadi.drive();
cadi.stop();

chevy.start();
chevy.drive();
chevy.stop();

taxi.start();
taxi.drive();
taxi.stop();
```

我们将代码复制并粘贴
到每个对象中，让每个
汽车对象都有相同的属
性和方法。

现在可以使用相同的
方法名对每个汽车对
象进行发动、驾驶和
熄火了。

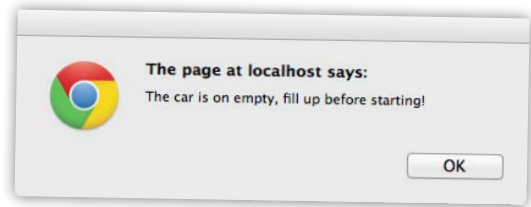


要在汽车对象中完整地实现属性fuel，还有一些工作要做。例如，没油时能够启动发动机吗？请完善方法start，在其中检查油位，并在没油时使用简单的提示（如The car is on empty, fill up before starting!）告知驾驶员。在下面重写方法start，将其加入你的代码中，并进行测试。继续往下阅读前，请查看本章末尾的答案。答案如下。

```
var fiat = {
  make: "Fiat",
  model: "500",
  year: 1957,
  color: "Medium Blue",
  passengers: 2,
  convertible: false,
  mileage: 88000,
  fuel: 0,
  started: false,

  start: function() {
    if (this.fuel == 0) {
      alert("The car is on empty, fill up before starting!");
    } else {
      this.started = true;
    }
  },

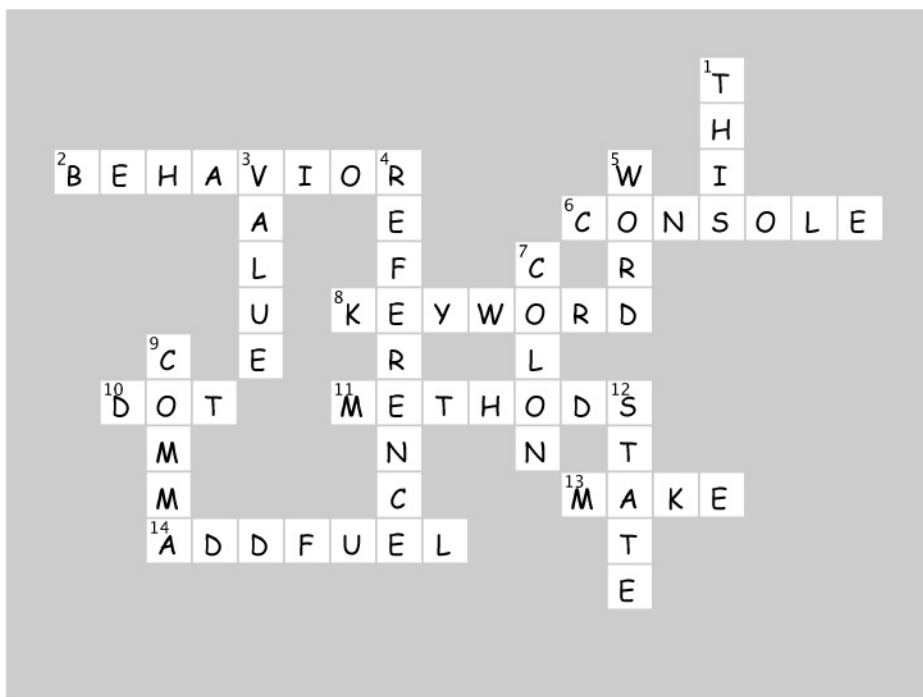
  stop: function() {
    this.started = false;
  },
  drive: function() {
    if (this.started) {
      if (this.fuel > 0) {
        alert(this.make + " " +
              this.model + " goes zoom zoom!");
        this.fuel = this.fuel - 1;
      } else {
        alert("Uh oh, out of fuel.");
        this.stop();
      }
    } else {
      alert("You need to start the engine first.");
    }
  },
  addFuel: function(amount) {
    this.fuel = this.fuel + amount;
  }
};
```





JavaScript填字游戏答案

填字游戏对象是什么样的呢？它有很多线索属性，可帮助你明白要达成的目标。



了解DOM



你的JavaScript水平有了很大的提高。事实上，你从门外汉成了脚本编写人员，又成了程序员，但还有一些东西没学。要充分利用你的JavaScript技能，就必须知道如何与代码所属的网页交互。只有这样，你才能编写出动态网页：能够对用户操作作出响应的网页，能够在加载后自动更新的网页。那么，如何与网页交互呢？使用DOM，即文档对象模型（document object model）。本章将详细介绍DOM，看看如何使用它和JavaScript赋予网页新功能。

上一章的密码破解难题

在上一章的密码破解难题中，提供了如下HTML和位于外部文件中的JavaScript代码，这些代码是从Evel博士的网站抓取的：



```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Dr. Evel's Secret Code Page</title>
  </head>
  <body>
    <p id="code1">The eagle is in the</p>
    <p id="code2">The fox is in the</p>
    <p id="code3">snuck into the garden last night.</p>
    <p id="code4">They said it would rain</p>
    <p id="code5">Does the red robin crow at</p>
    <p id="code6">Where can I find Mr.</p>
    <p id="code7">I told the boys to bring tea and</p>
    <p id="code8">Where's my dough? The cake won't</p>
    <p id="code9">My watch stopped at</p>
    <p id="code10">barking, can't fly without umbrella.</p>
    <p id="code11">The green canary flies at</p>
    <p id="code12">The oyster owns a fine</p>
    <script src="code.js"></script>
  </body>
</html>
```

这是HTML。

注意到给每个段落都指定了一个id。

这是JavaScript代码。

document是一个全局对象。

getElementById是一个方法。

方法名getElementById的大小写必须正确无误，否则代码将不能正确运行。

```
var access =
  document.getElementById("code9");
var code = access.innerHTML;
code = code + " midnight";
alert(code);
```

这里使用了句点表示法，因此access可能是一个包含属性innerHTML的对象。

你需要利用你的推理能力，根据这些代码破解出Evel博士的密码。

这些代码是做什么的呢

下面来详细解读这些代码，搞清楚Evel博士是如何生成密码的。对每个步骤进行解读后，你将逐渐明白其中的工作原理。

本章将全面介绍document和element对象。

- ① 上述JavaScript代码首先调用对象document的方法getElementById，并将"code9"传递给它，再将结果赋给变量access。这个方法返回的是一个element对象。

```
var access =
  document.getElementById("code9");
var code = access.innerHTML;
code = code + " midnight";
alert(code);
```

获取id为“code9”的元素，即下面这样的元素。

<p id="code9">My watch stopped at</p>

- ② 接下来，我们使用属性innerHTML获取这个元素（即id为"code9"的元素）的内容，并将其赋给变量code。

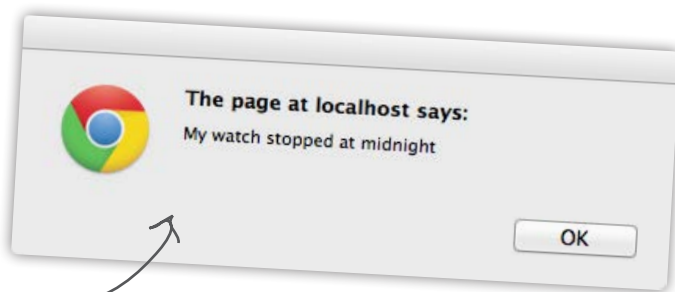
```
var access =
  document.getElementById("code9");
var code = access.innerHTML;
code = code + " midnight";
alert(code);
```

id为“code9”的元素是一个段落元素，其内容（即属性innerHTML）为文本My watch stopped at。

- ③ 在code包含的字符串（即"My watch stopped at"）末尾添加字符串"midnight"。然后创建一个提示框，在其中显示包含在变量code中的密码。

```
var access =
  document.getElementById("code9");
var code = access.innerHTML;
code = code + " midnight";
alert(code);
```

在“My watch stopped at”末尾加上“midnight”，得到密码“My watch stopped at midnight”，再弹出一个提示框来显示这个密码。



简单总结一下

刚才我们都做了什么呢？我们使用了一些JavaScript代码，它们深入页面（也叫文档），抓取一个元素（id为"code9"的元素），获取其内容（"My watch stopped at"），在末尾加上"midnight"，再将结果作为密码显示出来。

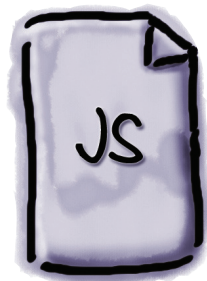
- ① Evel博士的网页包含各种候选密码，每个都放在一个用HTML id标识的段落中。



浏览器

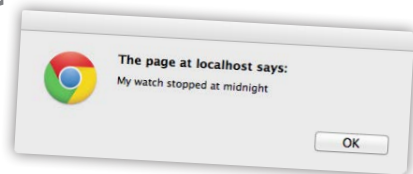
- ② 在幕后，JavaScript代码抓取id为"code9"的元素。

`<p id="code9">My watch stopped at</p>`



`"My watch stopped at" + " midnight"`

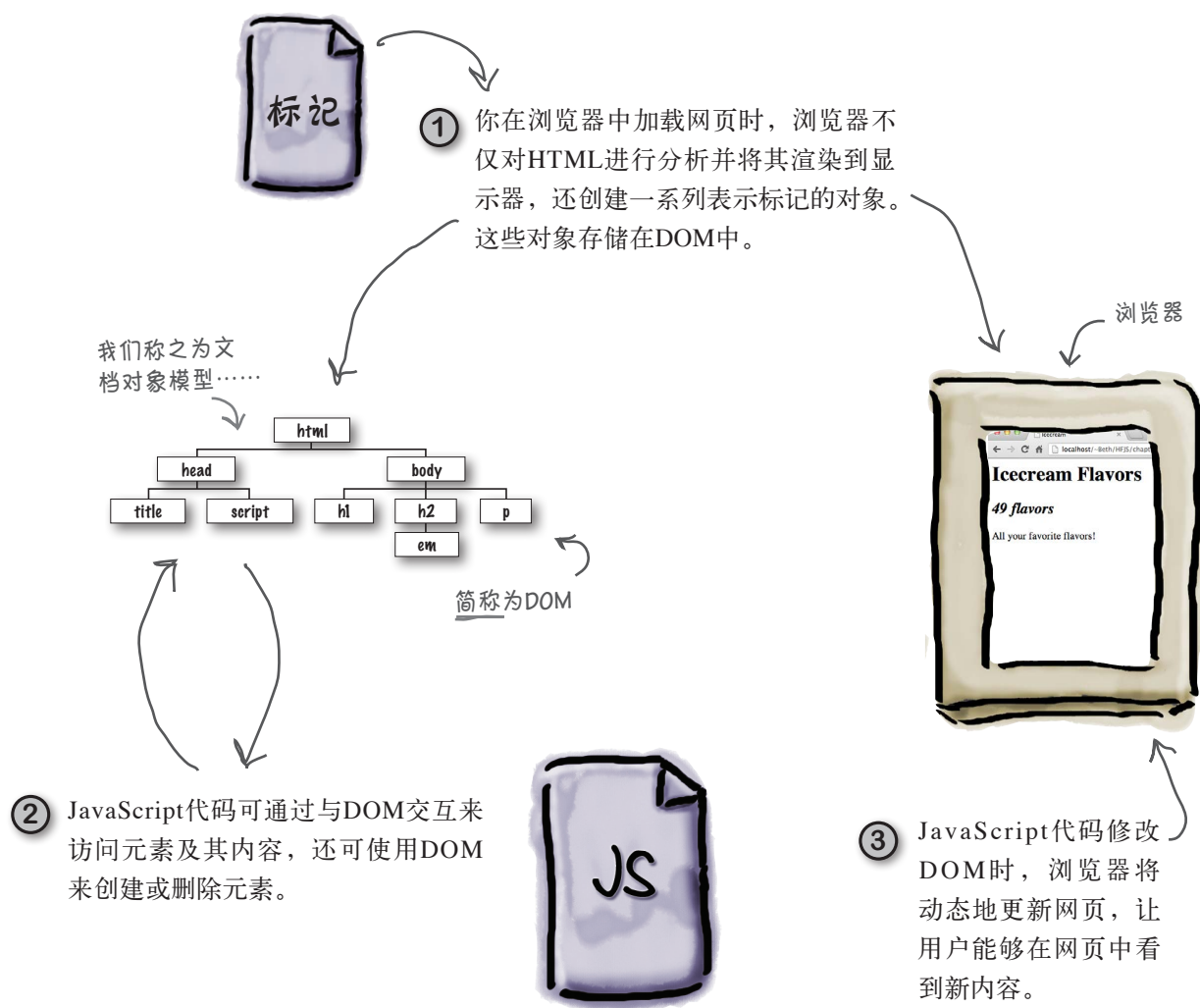
- ③ 获取这个元素的内容，在末尾加上"midnight"，并在一个提示框中显示结果。



现在，我们的JavaScript技能超过了Evel博士，但愿他的安全措施天衣无缝。这里要说的重点是，网页是一个动态的数据结构，你可通过JavaScript与之交互：你可以访问并读取网页中元素的内容，还可以修改网页的内容或结构。为此，我们必须暂时回过头去更深入地了解JavaScript和HTML是如何协同工作的。

JavaScript如何与网页交互

JavaScript和HTML是两样不同的东西：HTML是标记，而JavaScript是代码。它们如何交互呢？这是通过网页的表示，即文档对象模型（DOM）实现的。DOM是怎么来的呢？它是浏览器在加载网页时创建的。创建过程如下。



如何自己动手制作DOM

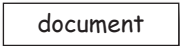
咱们来使用一些标记来制作DOM。下面是一个制作DOM的简单食谱。

原料

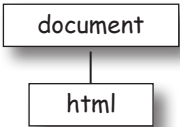
- 一个设计良好的HTML5网页
- 一个准备就绪的现代Web浏览器

做法

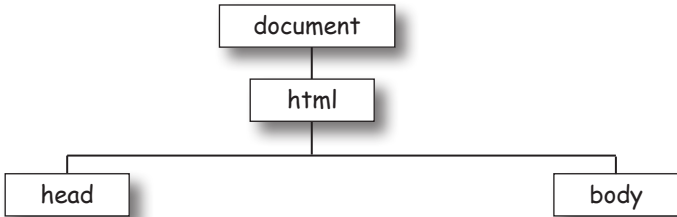
1. 首先，创建一个document节点。



2. 接下来，提取HTML网页的顶级元素（这里是<html>元素），将其视为当前元素，并作为document的子节点加入DOM中。

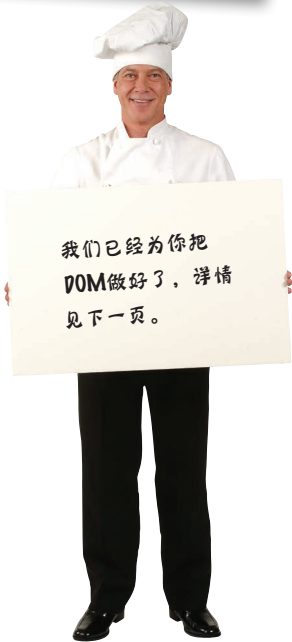


3. 对于嵌套在当前元素的每个元素，都将其作为当前元素的子节点加入到DOM中。



4. 对于刚添加的每个元素，都重复第3步，直到所有元素都加入到了DOM中。

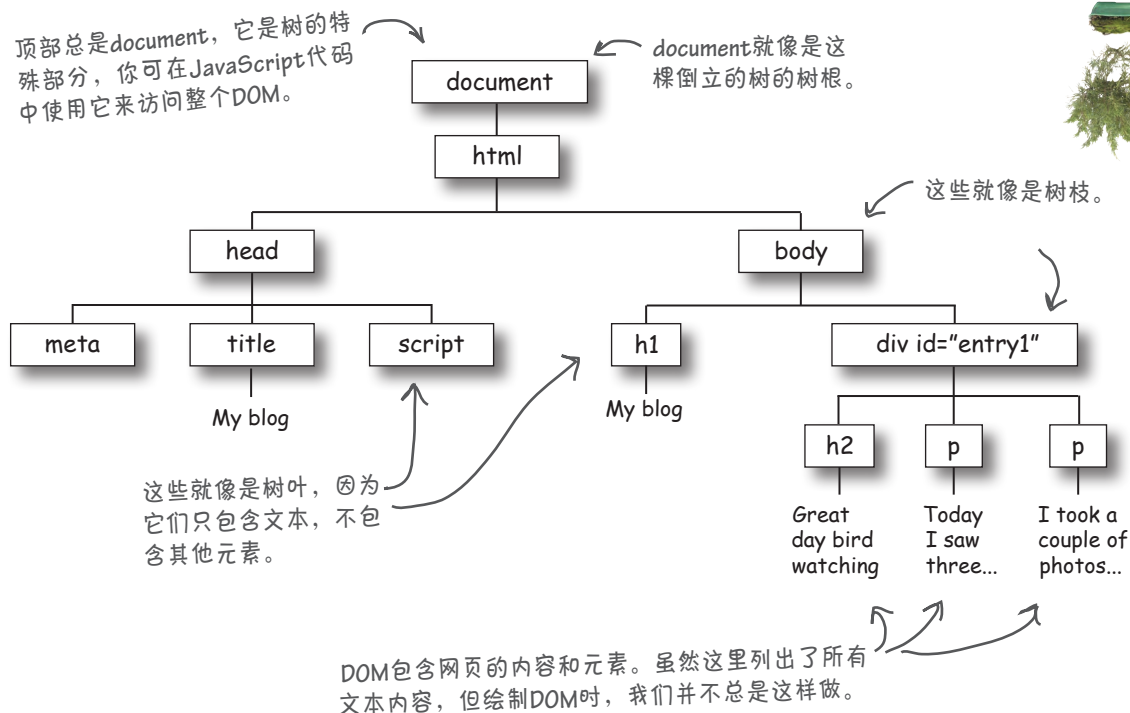
```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>My blog</title>
  <script src="blog.js"></script>
</head>
<body>
  <h1>My blog</h1>
  <div id="entry1">
    <h2>Great day bird watching</h2>
    <p>
      Today I saw three ducks!
      I named them
      Huey, Louie, and Dewey.
    </p>
    <p>
      I took a couple of photos...
    </p>
  </div>
</body>
</html>
```



初尝DOM

如果你按前面的DOM食谱做，最终将得到一个类似于下面的结构。每个DOM顶部都是一个document对象，它下面是一棵树，其中包含由HTML标记中各个元素构成的树枝和树叶。下面来仔细研究一下。

我们将这种结构比作一棵树，不仅因为树是计算机科学领域的一种数据结构，还因为它看起来就像一棵倒立的树：最上面是树根，最下面是树叶。



有了DOM后，想怎么查看和修改它都可以。

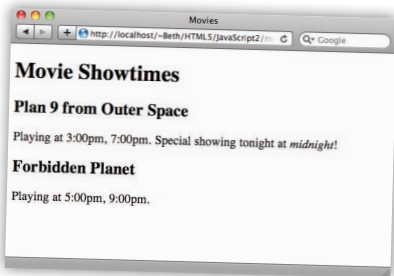




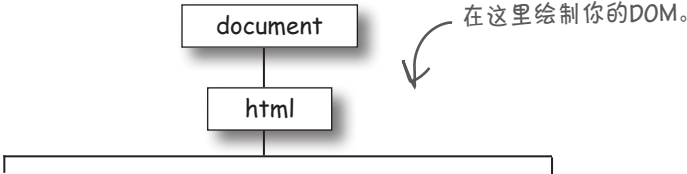
变身浏览器

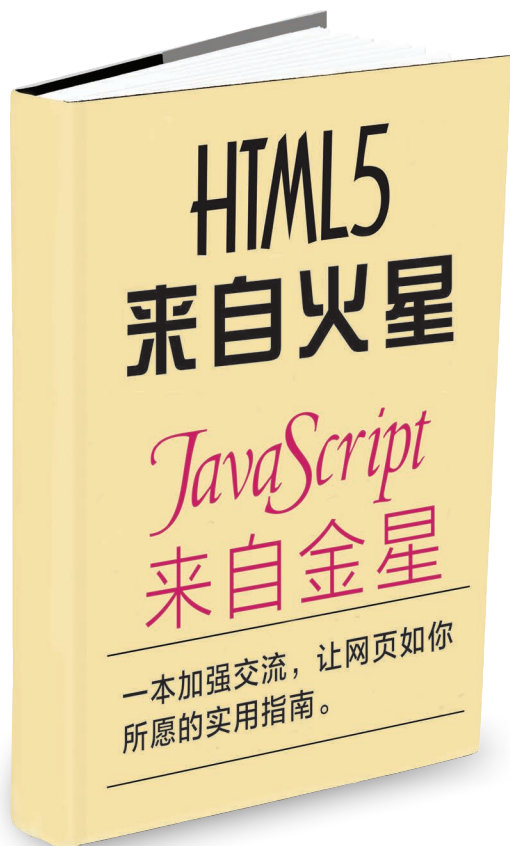
你的任务是变身浏览器，亲自对HTML进行分析并制作DOM。现在就对右边的HTML进行分析，并在下面绘制DOM吧。开头部分已经为你绘制好了。

请将你绘制的DOM与本章末尾的答案进行比较，再接着往下阅读。



```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Movies</title>
  </head>
  <body>
    <h1>Movie Showtimes</h1>
    <h2 id="movie1">Plan 9 from Outer Space</h2>
    <p>Playing at 3:00pm, 7:00pm.
      <span>
        Special showing tonight at <em>midnight</em>!
      </span>
    </p>
    <h2 id="movie2">Forbidden Planet</h2>
    <p>Playing at 5:00pm, 9:00pm.</p>
  </body>
</html>
```



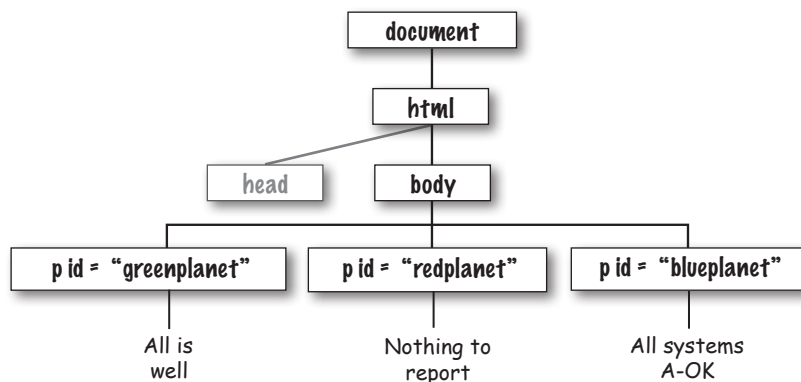


两种截然不同的技术怎么能走到一起呢？

HTML和JavaScript肯定来自不同的星球。何出此言？HTML的DNA由声明式标记组成，让你能够描述一系列组成网页的嵌套元素；而JavaScript的DNA完全是由算法遗传物质组成的，用于对计算进行描述。

它们根本不是一路人，是不是就无法交流呢？当然不是，因为它们之间有座桥：DOM。通过DOM，JavaScript能够与网页交流，反之亦然。交流的途径有多种，这里只介绍其中一种：它就像一个小小的虫洞，让JavaScript能够访问网页中的任何元素。这个虫洞就是 `getElementById`。

先来看一个DOM。这个DOM很简单，其中包含一些HTML段落，每个段落都由id标识为绿色星球、红色星球或蓝色星球。另外，每个段落都包含一些文本。当然，还有一个<head>元素，但出于简化考虑，我们暂时不管这些细节。



下面使用JavaScript来增加一些趣味。假设我们要将元素greenplanet的文本All is well改为Red Alert: hit by phaser fire!。将来你可能要根据用户的操作乃至来自Web服务的数据执行类似的操作，这些都将在本书后面介绍，但这里直接修改元素greenplanet的文本。为此，需要获取id为greenplanet的元素，如下面的代码所示：

document表示浏览器显示的整个网页，它包含完整的DOM，因此可以请它来完成查找id为指定值的元素等任务。

在这里，我们请document提供一个元素：查找id为指定值的元素。

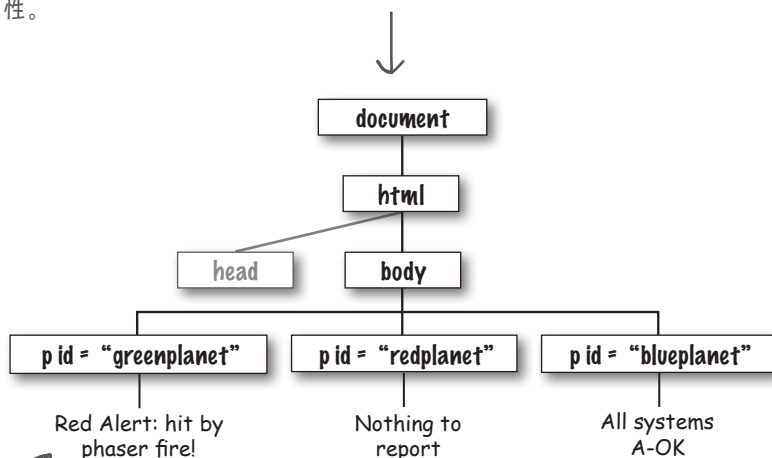
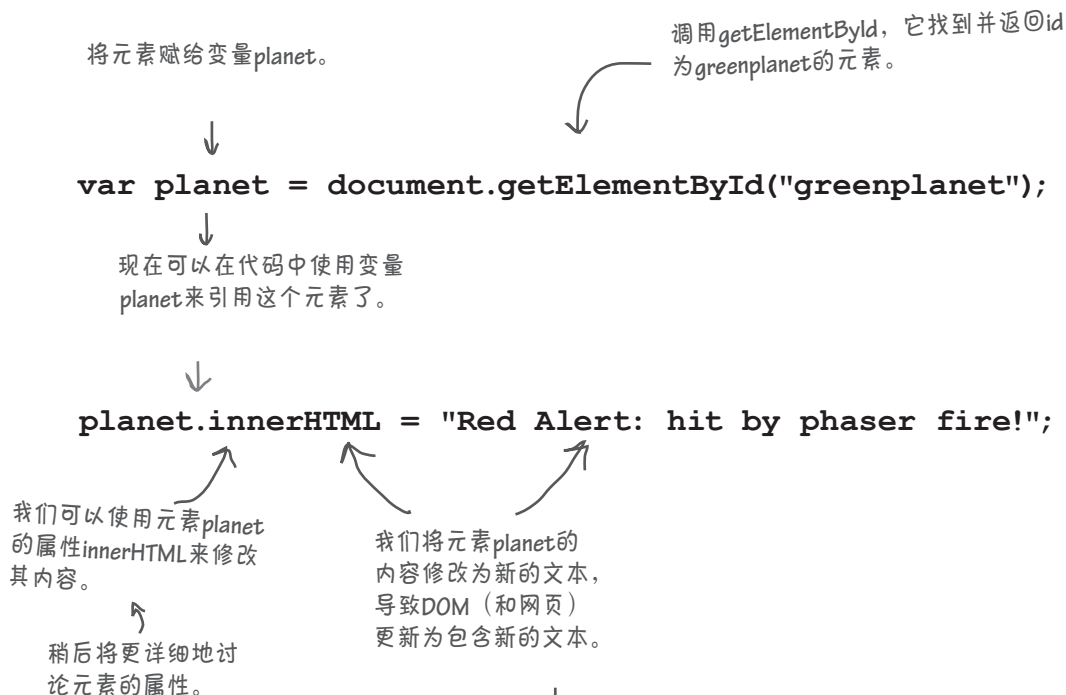
```
document.getElementById("greenplanet");
```

getElementById("greenplanet")返回id为“greenplanet”的段落元素。



然后，JavaScript代码就可对它执行各种有趣的操作了。

`getElementById`将元素提供给你后，你就可以对其执行操作了，如将其文本改为Red Alert: hit by phaser fire!。为此，通常将这个元素赋给一个变量，以便能够在任何地方引用它。下面就来这样做，并对元素的文本进行修改：

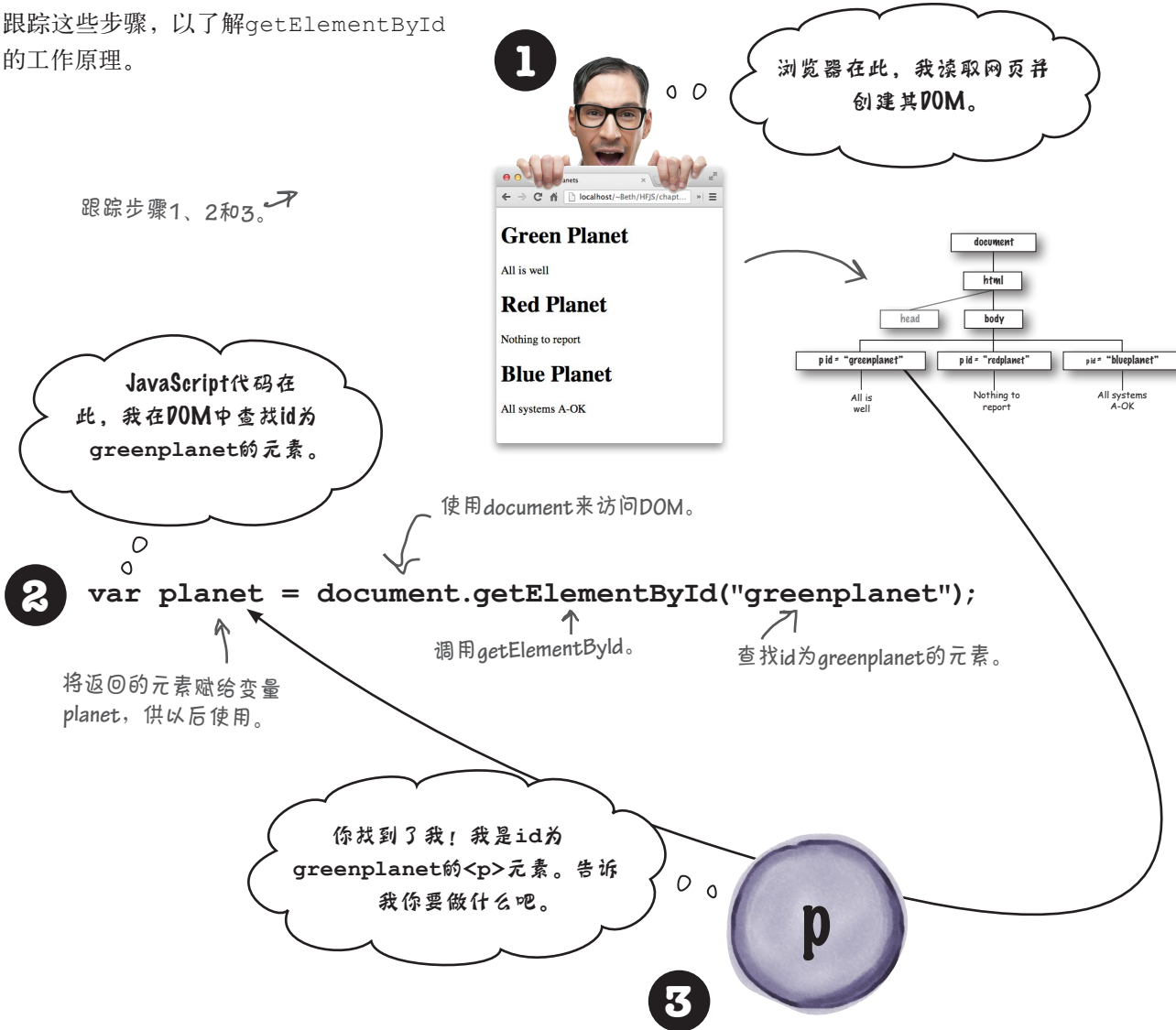


对DOM所做的任何修改都将在浏览器渲染得到的网页中反映出来，因此这个段落将发生变化，其中包含的是新内容！

使用getElementById获取元素

刚才我们都做什么了？下面更详细地说明。我们在代码中使用document对象来访问DOM，这是一个内置对象，有许多属性和方法，其中包括我们用来从DOM中获取一个元素的getElementById。方法getElementById接受一个id，并返回该id指定的元素。在CSS中，你可能使用过id来选择元素并设置其样式，但这里使用id从DOM中获取一个元素——id为greenplanet的<p>元素。

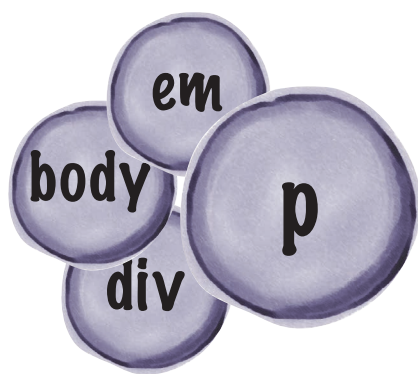
获得正确的元素后，就可对其进行修改了。这些将稍后介绍，现在的重点是跟踪这些步骤，以了解getElementById的工作原理。



从DOM获取的到底是什么

使用`getElementById`从DOM获取元素时，得到的是一个元素对象，你可使用它来读取、修改或替换元素的内容和特性。神奇之处在于，当你修改元素时，所做的修改也将反映到网页中。

重要的事先办，咱们再来看看刚才从DOM获取的元素对象。我们知道，这个元素对象表示的是网页中id为`greenplanet`的`<p>`元素，该元素的文本内容为All is well。与其他JavaScript对象一样，元素对象也有属性和方法。就元素对象而言，我们可以使用其属性和方法来读取和修改元素。下面是你可对元素对象执行的一些操作。



获取内容（文本或HTML）。

修改内容。

读取特性。

添加特性。

修改特性。

删除特性。

你可对元素对象执行的一些操作。

对于这个`<p>`元素（别忘了，它是id为`greenplanet`的`<p>`元素），我们要做的是，将其内容All is well改为Red Alert: hit by phaser fire!。我们已经将这个元素存储到了变量`planet`中，下面使用这个变量来修改该元素的属性之一——`innerHTML`：

变量`planet`包含一个元素对象——id为
`greenplanet`的`<p>`元素。

```
var planet = document.getElementById("greenplanet");
```

```
planet.innerHTML = "Red Alert: hit by phaser fire!";
```

可使用元素对象的属性`innerHTML`来修改元素的内容。

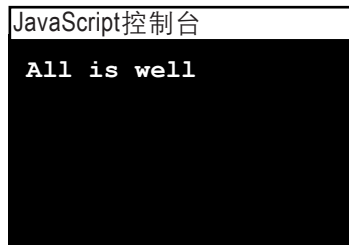
查找内部HTML

innerHTML是个重要的属性，使用它可以读取或替换元素的内容。如果你查看innerHTML的值，看到的将是包含在元素中的内容（元素的HTML标签除外），“内部”HTML因此而得名。咱们来做个小实验，通过显示属性innerHTML，将元素对象planet的内容显示到控制台，如下所示：

```
var planet = document.getElementById("greenplanet");
console.log(planet.innerHTML);
```

将属性planet.innerHTML传递给console.log，
从而将其值显示到控制台。

属性planet.innerHTML的值是一个字符串，在控制台中的显示方式与其他字符串没什么两样。



下面来尝试修改属性innerHTML的值。当我们这样做时，将修改网页中id为greenplanet的<p>元素的内容，因此你将看到网页也发生了变化。

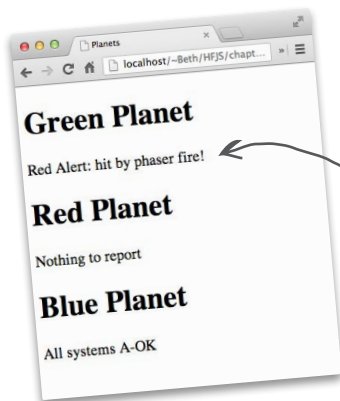
```
var planet = document.getElementById("greenplanet");
planet.innerHTML = "Red Alert: hit by phaser fire!";
console.log(planet.innerHTML);
```

修改元素的内容：将其属性innerHTML设置为字符串“Red Alert: hit by phaser fire!”。

再次将属性innerHTML的值写入控制台时，我们看到的是新设置的值。



网页也发生了变化。





快速复习

请坐下来休息一会儿。你可能自言自语地说：等等，我对id和类还有点印象，但细节记不太清楚了，它们不是CSS中的东西吗？别担心，我们来简单复习一下，提供一点点背景知识，再重回正题。

在HTML中，id让我们能够唯一地标识元素；给元素指定独一无二的id后，就可在CSS中使用id来选择元素以设置其样式。另外，正如你已经看到的，在JavaScript中，可使用id来获取元素。

来看一个示例：

```
<div id="menu">
...
</div>
```

我们给这个<div>元素指定了一个独一无二的id——menu。在整个网页中，必须确保只有这个元素的id为menu。

指定id后，就可在CSS中使用它来选择元素以设置其样式了，如下所示：

```
div#menu {
    background-color: #aaa;
}
```

div#menu 是一个id选择器。

div#menu选择id为menu的<div>元素，这让我们能够给这个元素（也只有这个元素）设置样式。

在JavaScript中，也可通过id来访问这个元素：

```
var myMenu = document.getElementById("menu");
```

别忘了，还有另一种标识元素的方式：使用类（class）。类能够让我们标识一组元素，如下所示：

```
<h3 class="drink">Strawberry Blast</h3>
<h3 class="drink">Lemon Ice</h3>
```

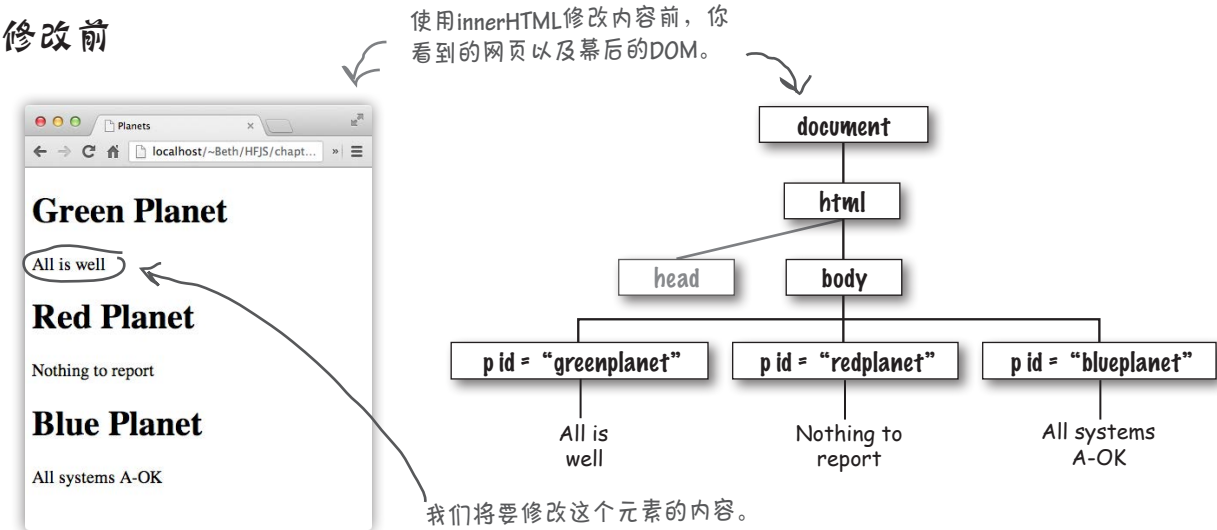
这两个<h3>元素都属于drink类。类像编组，你可以将多个元素加入到同一个编组中。

在CSS和JavaScript中，都可根据类来选择元素，稍后将介绍如何在JavaScript中使用类。顺便说一句，如果你觉得这里复习的内容不够，可参阅《Head First HTML与CSS》的第7章，也可参阅你喜欢的其他HTML和CSS指南。

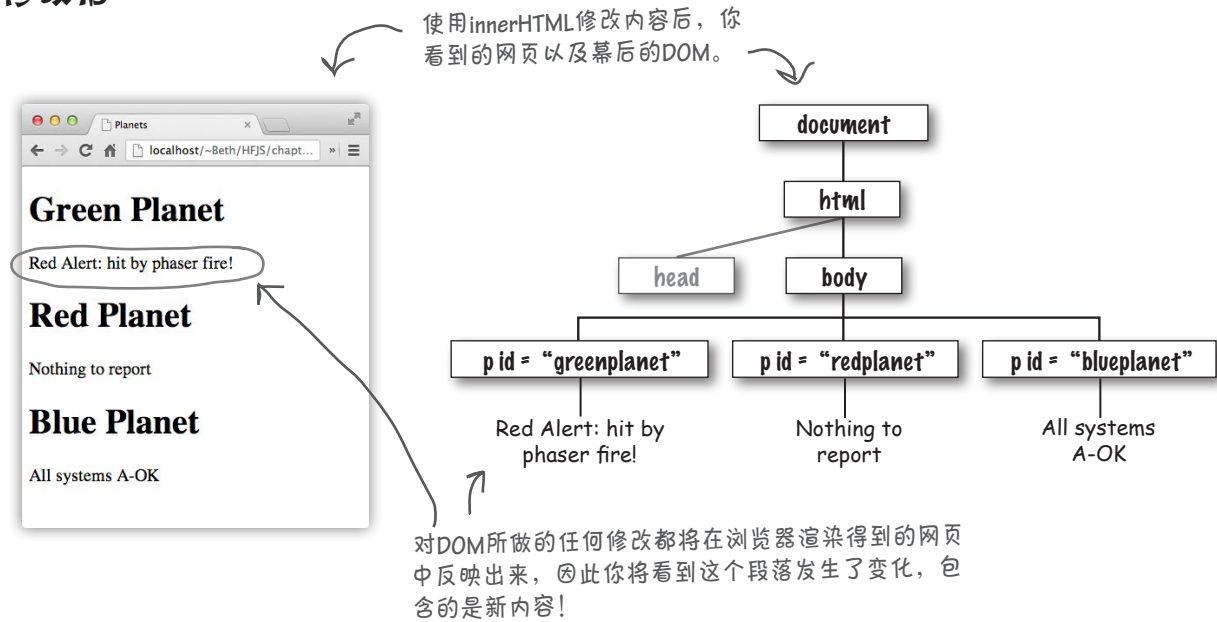
修改DOM的影响

使用innerHTML修改元素的内容时，到底会有什么样的影响呢？你实际上是在即时地修改网页的内容。因此，当你修改DOM的内容时，网页也将随之发生变化。

修改前



修改后



世上没有 愚蠢的问题

问：调用`document.getElementById`时，如果传入的id不存在，结果将如何？

答：根据id从DOM获取元素时，如果指定的id不存在，`getElementById`将返回`null`。调用`getElementById`时，检查返回的是否是`null`是个不错的主意，这样可确保只在返回了一个元素时才访问其属性。`null`将在下一章更详细地介绍。

问：也可使用`document.getElementById`根据类来获取元素吗？例如，如果有很多属于`planets`类的元素，是否可以这样做？

答：不能，但你的思路是对的。调用`getElementById`时，只能传入id。但有一个名为`getElementsByClassName`的DOM方法，可用来根据类名获取元素。调用这个方法时，返回的是一个元素集合，其中包括属于指定类的所有元素（因为可能有多个元素属于同一个类）。还有一个返回元素集合的方法——`getElementsByTagName`，它返回与指定标签名匹配的所有元素。`getElementsByTagName`将在稍后介绍，届时你将明白如何处理它返回的元素集合。

问：元素对象到底是什么呢？

答：问得好。在浏览器内部，使用元素对象来表示你在HTML文件中输入的内容，如`<p>some text</p>`。加载并分析HTML文件时，浏览器为网页中的每个元素创建一个元素对象，并将它们加入到DOM中。因此，DOM其实就是一棵由元素对象组成的大树。另外，别忘了，与其他对象一样，元素对象也有属性（如`innerHTML`）和方法，本书后面将介绍元素对象的其他一些属性和方法。

问：我觉得元素对象应包含一个名为`content`或`html`的属性，为何将该属性命名为`innerHTML`呢？

答：我也是这么认为的，这个属性名有点怪。属性`innerHTML`表示元素包含的所有内容，包括嵌套的元素（例如，段落元素除文本外，还可能包含`<em>`和`<img>`元素）。换句话说，它表示元素内部的HTML。是不是还有属性`outerHTML`呢？有！这个属性表示元素内部的HTML以及元素本身。实际上，`outerHTML`用得不多，但`innerHTML`常被用来修改元素的内容。

问：也就是说，通过给`innerHTML`赋值，可替换任何元素的内容。是否也可使用`innerHTML`来修改`<body>`元素的内容呢？

答：你说的没错，`innerHTML`能够让你方便地替换元素的内容。你也可以使用它来替换`<body>`元素的内容，这将导致整个网页被替换为新内容。

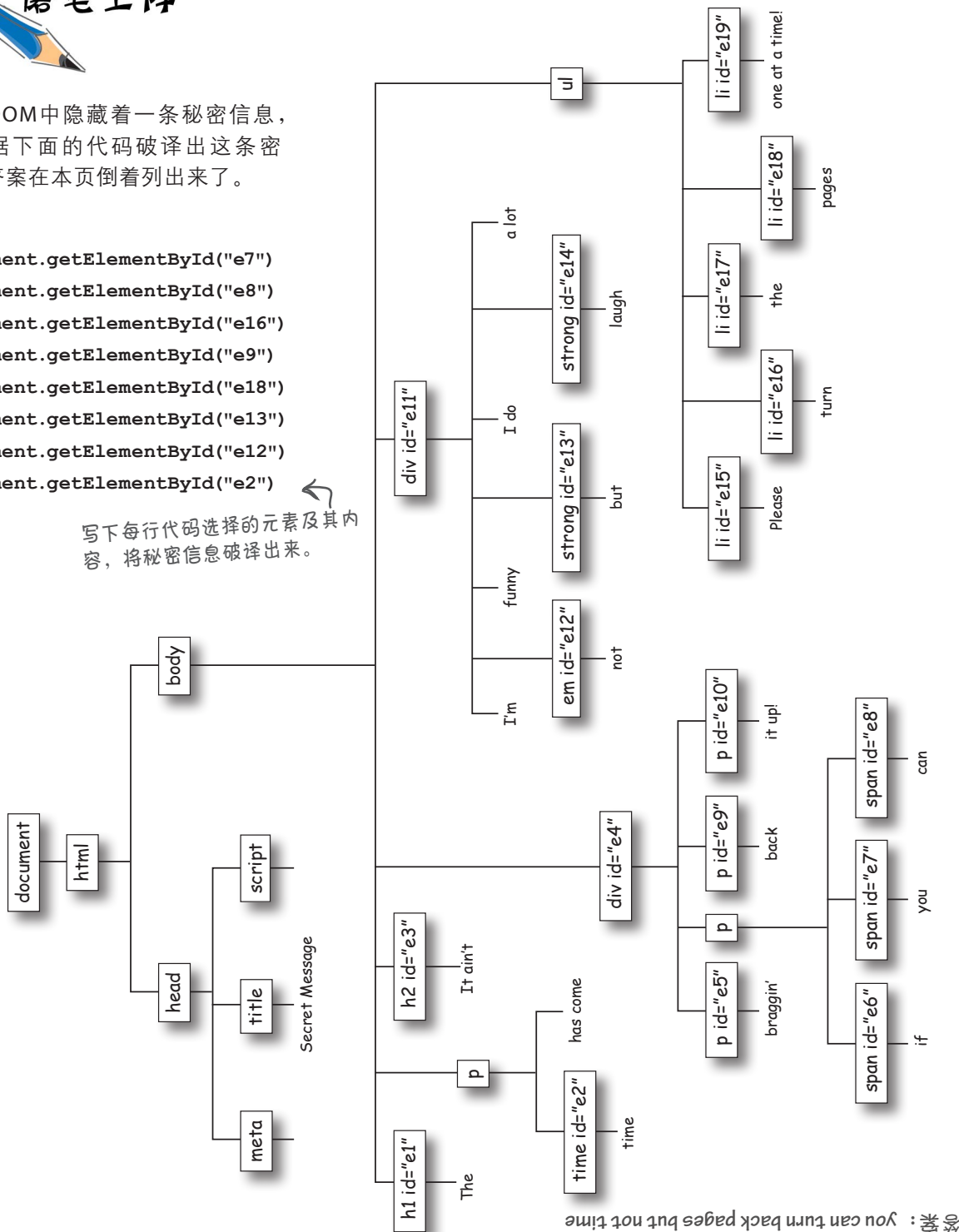


磨笔上阵

这个DOM中隐藏着一条秘密信息，请根据下面的代码破译出这条密电，答案在本页倒着列出来了。

```
document.getElementById("e7")
document.getElementById("e8")
document.getElementById("e16")
document.getElementById("e9")
document.getElementById("e18")
document.getElementById("e13")
document.getElementById("e12")
document.getElementById("e2")
```

写下每行代码选择的元素及其内容，将秘密信息破译出来。



围绕星球进行试驾

你知道如何使用`document.getElementById`来访问元素，还知道如何使用`innerHTML`来修改元素的内容，来真刀真枪地尝试一下吧。

下面是显示星球的HTML。除了三个表示绿色星球、红色星球和蓝色星球的段落外，还在`<head>`元素中嵌入了一个用于放置JavaScript代码的`<script>`元素。请输入这些HTML以修改DOM的JavaScript代码——如果你还没有这样做的话：



```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Planets</title>
  <script>
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
  </script>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
</body>
</html>
```

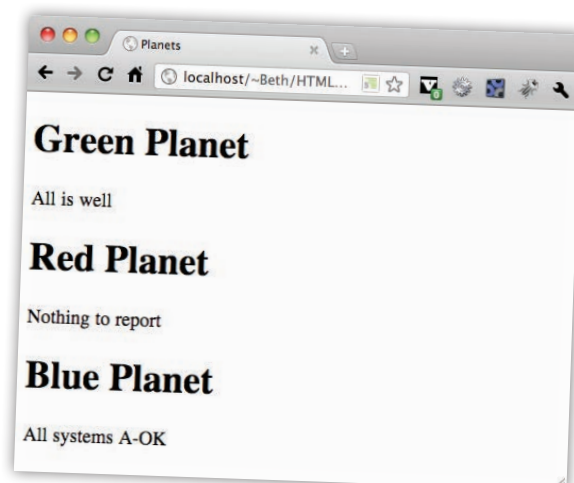
包含代码的
`<script>`元素。

像前面介绍的一样，获取
id为`greenplanet`的`<p>`元
素，并修改其内容。

你要使用JavaScript
代码修改的`<p>`元素。

输入这些代码后，在浏览器中加载这个网页，看看绿色星球上发生的DOM奇迹。

哎呀！出问题了，绿色星球显示的还是All is well。到底是怎么回事呢？





标记和代码我都检查了三遍，
就是不行，我没有看到网页有
任何变化。

哦，对不起，有一点我们忘记说了。

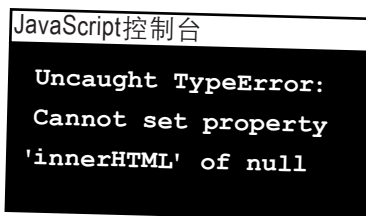
处理DOM时，确保代码在网页完全加载后再执行至关重要；否则，代码执行时，DOM很可能还未创建。

来看看到底出了什么问题：我们将代码放在了网页的<head>元素中，因此它执行时浏览器尚未读取网页的其他部分。这是个大问题，因为此时还没有id为greenplanet的段落元素。

这将导致什么样的结果呢？调用getElementById时，将返回null而不是我们希望的元素，进而导致错误。但浏览器沉得住气，对此置若罔闻，依然对网页进行渲染，只是没有修改绿色星球的内容。

如何修复这种问题呢？可将代码移到<body>元素的末尾，但有一种更简单的办法可确保代码在正确的时候运行，它让浏览器在加载网页并创建DOM后再执行代码。下面就来看看如何做。

加载该网页时，如果查看控制台，你在大多数浏览器中都将看到错误消息。控制台是个很不错的调试工具。



网页加载完毕前别想运行代码

怎么做呢？除了将代码移到<body>元素末尾外，还有一种办法，那就是通过代码（有些人认为这种办法更为干净利落）。

具体做法如下。首先，创建一个函数，在其中包含要在网页加载完毕后执行的代码。然后，将这个函数赋给对象window的属性onload。

window是一个JavaScript内置对象，表示浏览器窗口。

这有何作用呢？对象window将调用你赋给其属性onload的函数，但仅在网页加载完毕后才这样做。这要感谢对象window的设计人员，是他们给你提供了一个途径，让你能够指定在网页加载完毕后将要执行的代码。请看下面的示例：

```
<script>
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
}

window.onload = init;
</script>
```

首先，创建一个名为init的函数，并将原来的代码加入到这个函数中。

可随便给这个函数命名，但根据约定通常将其命名为init。

这是原来的代码，现在放到了函数init中。

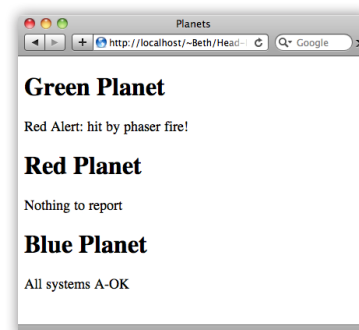
将函数init赋给属性window.onload。千万不要在函数名后添加括号，因为这里不是调用函数，而只是将函数作为一个值赋给属性window.onload。

再来试一试



重新加载改用了函数init和属性onload的网页。这次浏览器将加载网页、创建完整的DOM，然后才调用函数init。

这就对了，现在绿色星球显示了Red Alert，与我们希望的一样。



你说“事件处理程序”，我说“回调函数”

下面来更深入地研究一下onload的工作原理，因为它使用了一个很常见的JavaScript编码模式。

假设有重大的事件（如网页加载完毕）即将发生，而你必须在其发生后第一时间获悉。对于这种情形，一种常见的处理方式是使用回调函数 [callback，也叫事件处理程序（event handler）] 。

回调函数的工作原理如下：给了解事件的对象提供一个函数；事件发生后，这个对象将通过调用这个函数来通知你。在JavaScript中，对很多事件都采用了这种处理模式。

我是回调函数。如果你愿意，也可称我为事件处理程序。

浏览器，我等你加载完网页，好让我能够做些事情。

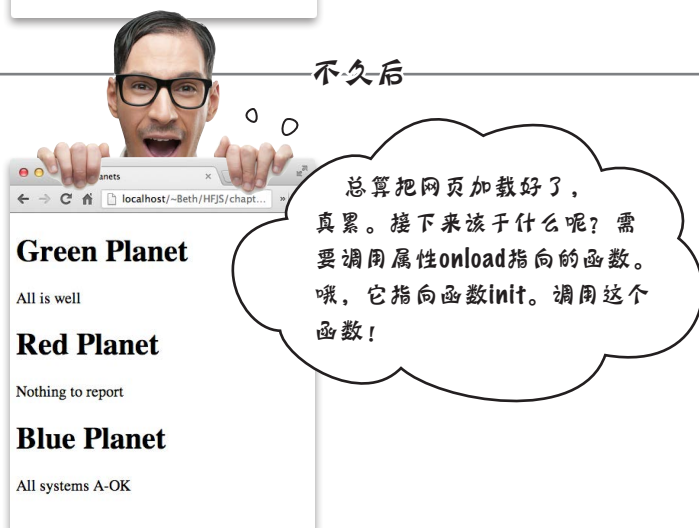
我是浏览器，更准确地说是对象window。

别坐等呀，给我提供一个回调函数好了，我完事就调用它。

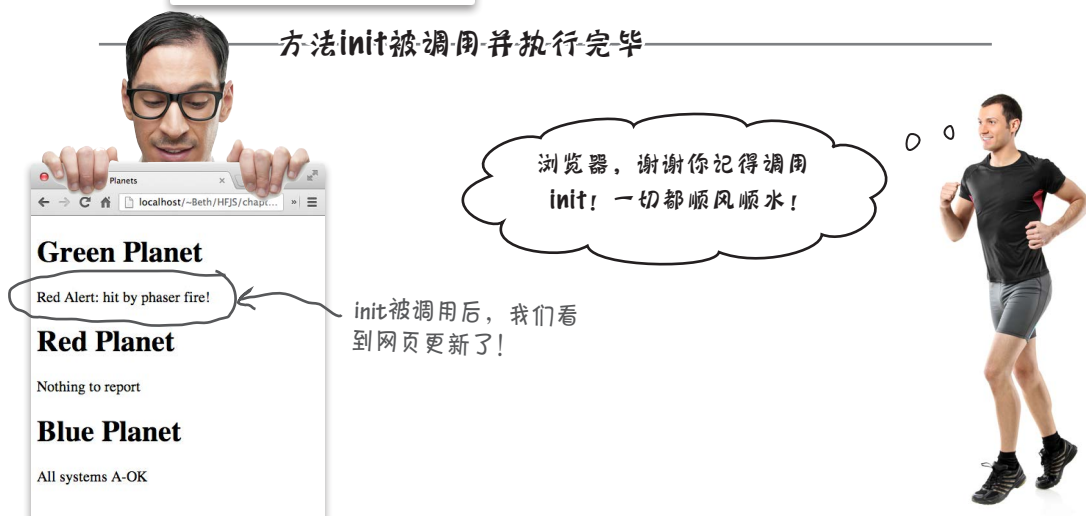
没问题，这就给，它名为init。



不久后



方法init被调用并执行完毕

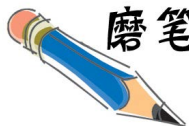




有意思。我可以将事件发生后要执行的代码放在函数中。还可以像这样使用函数来处理哪些其他的事件呢？

你说的没错。如果你愿意，还能使用这种方式处理很多事件。有些事件是浏览器生成的，如加载事件，有些事件是用户与网页交互时生成的，还有些事件是JavaScript代码生成的。

刚才的示例演示了“网页加载完毕”事件，我们通过设置对象window的属性onload来处理它。你还可编写这样的事件处理程序：每隔5秒钟调用一次；在有来自Web服务的数据需要处理时调用；在用户单击按钮后，需要从表单获取数据时调用；等等。创建更像应用程序而不是静态文档的网页（谁都希望自己的网页如此）时，将大量地使用上述事件。刚才简要地介绍了一下事件处理程序，鉴于它们在JavaScript编程中扮演着至关重要的角色，本书后面将花大量的篇幅讨论它们。



磨笔上阵

下面的HTML显示一个歌曲播放列表，但该列表是空的。请补全下述在该列表中添加歌曲的JavaScript代码，然后查看本章末尾的答案，紧接着往下阅读。

```
<!doctype html>
```

这是网页的HTML。

```
<html lang="en">
```

```
<head>
```

```
<title>My Playlist</title>
```

```
<meta charset="utf-8">
```

脚本。这些代码在元素定义的列表中添加歌曲。

```
<script>
```

```
_____ addSongs() {
```

```
  var song1 = document._____("_____");
```

```
  var _____ = _____("_____");
```

```
  var _____ = _____.getElementById("_____");
```

请补全这些代码，在播放列表中添加歌曲。

```
  _____.innerHTML = "Blue Suede Strings, by Elvis Pagely";
```

```
  _____ = "Great Objects on Fire, by Jerry JSON Lewis";
```

```
  song3._____ = "I Code the Line, by Johnny JavaScript";
```

```
}
```

```
window._____ = _____;
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<h1>My awesome playlist</h1>
```

```
<ul id="playlist">
```

```
<li id="song1"></li>
```

```
<li id="song2"></li>
```

```
<li id="song3"></li>
```

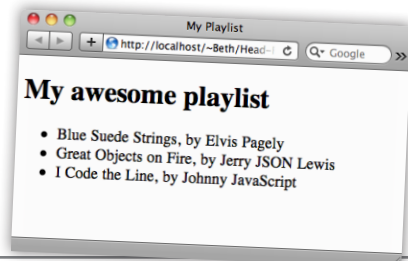
```
</ul>
```

```
</body>
```

```
</html>
```

空的歌曲列表。前述JavaScript代码将给播放列表中的每个元素添加内容。

正确地补全JavaScript代码并加载该网页后，它应该类似于这样。



别停下，继续前进

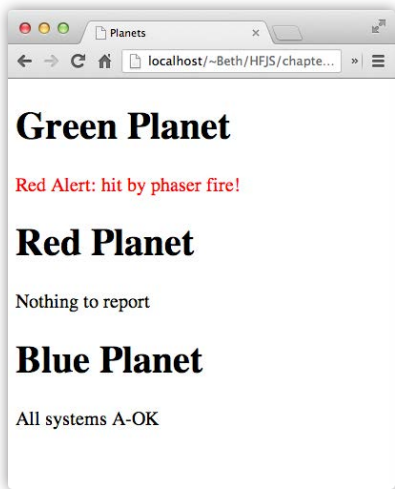
来看看你刚才做的：你在一个静态网页中添加代码，从而动态地修改了一个段落的内容。这看似简单，却是创建真正交互式网页的第一步。

这正是我们的目标，第8章将全面实现这个目标。

下面来迈出第二步：知道如何获取DOM中的元素后，咱们来使用代码设置元素的特性。

这很有趣，为什么呢？继续前面简单的星球示例，在将段落的内容设置为Red Alert的同时，还可将其颜色设置为红色，这无疑将更清晰地传递信息。

具体做法如下。



- ① 给redtext类定义一条CSS规则，将段落文本设置为红色。这样，被加入到这个类的元素的文本都将为红色。
- ② 接下来，添加这样的代码：获取id为greenplanet的段落元素，并将其加入redtext类。

就这么简单。现在，我们只需学习如何设置元素的特性，就能够编写这样的代码了。



将大脑的另一部分也开动起来如何？请给redtext类定义CSS样式，将星球段落的文本设置为红色。如果你很久没有编写CSS，也不用担心，你完全可以试一试；如果你闭着眼睛都能编写出来，那就太好了。答案在本章末尾。

如何使用setAttribute设置特性

元素对象有一个setAttribute方法，你可调用它来设置HTML元素的特性。要调用这个方法，可像下面这样做：

获取的元素对象

```
planet.setAttribute("class", "redtext");
```

使用方法setAttribute添加新特性或修改既有特性。

这个方法接受两个实参：要添加或修改的特性的名称……

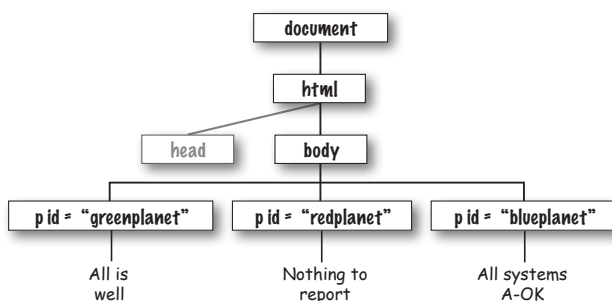
要设置的特性值。

请注意，如果指定的特性不存在，将在元素中创建它。

可对任何元素调用setAttribute来修改特性的值；如果指定的特性不存在，将在元素中添加它。例如，来看看执行上述代码将给DOM带来的影响：

执行前

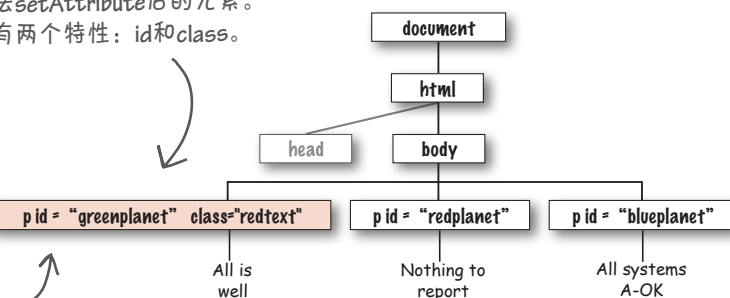
调用方法setAttribute前的元素，注意这个元素已经有一个特性id。



执行后

调用方法setAttribute后的元素。现在它有两个特性：id和class。

别忘了，调用方法setAttribute时，将修改DOM中的元素对象。这种修改将立即在浏览器显示的网页中反映出来。



获取特性

需要获取元素的特性值？没问题，有一个方法叫`getAttribute`，你可调用它来获取HTML元素的特性的值。

使用`getElementById`获取一个指向元素的引用，再使用方法`getAttribute`获取特性的值。

传入要获取其值的特性的名称。

```
var scoop = document.getElementById("raspberry");
var altText = scoop.getAttribute("alt");
console.log("I can't see the image in the console,");
console.log(" but I'm told it looks like: " + altText);
```

如果指定的特性不存在，结果将如何

还记得调用`getElementById`时指定的id在DOM中不存在的情形吗？将返回`null`。`getAttribute`亦如此。如果指定的特性不存在，将返回`null`。下面演示了如何检查这种情况：

```
var scoop = document.getElementById("raspberry");
var altText = scoop.getAttribute("alt");
```

检查是否返回了一个特性值。

```
if (altText == null) {
    console.log("Oh, I guess there isn't an alt attribute.");
} else {
    console.log("I can't see the image in the console,");
    console.log(" but I'm told it looks like " + altText);
}
```

如果没有返回特性值，就这样做。

如果返回了特性值，就可在控制台中显示其文本内容。

别忘了`getElementById`也可能返回`null`

每当你获取值时，都一定要确认获得了期望的值……

调用`getElementById`时，如果指定的元素id在DOM中不存在，它将返回`null`。为遵循最佳实践，试图获取元素时，也务必检查返回的是否是`null`。本书原本也应遵守这种规则，但如果这样做，篇幅将长得多。

回到星球示例

现在将这些代码整合到星球示例中，并最后一次试驾。

```

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Planets</title>
  <style>
    .redtext { color: red; }
  </style>
  <script>
    function init() {
      var planet = document.getElementById("greenplanet");
      planet.innerHTML = "Red Alert: hit by phaser fire!";
      planet.setAttribute("class", "redtext");
    }
    window.onload = init;
  </script>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
</body>
</html>

```

星球示例的完整HTML、CSS和JavaScript代码。

我们定义了redtext类，这样在代码中将元素的特性class设置为redtext时，其文本将变成红色。

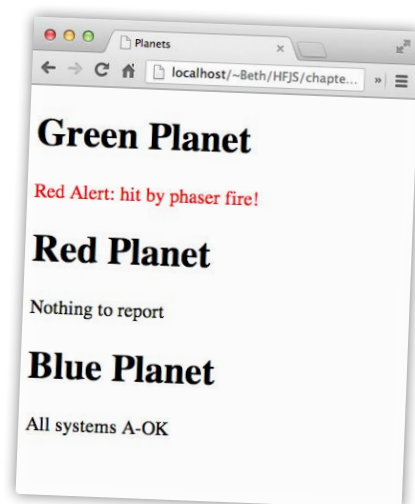
网页加载完毕后再调用函数init。

再来复习一下：获取id为greenplanet的元素，并将其存储到变量planet中。然后，修改这个元素的内容。最后，给这个元素添加一个class特性，将这个元素的文本改成红色。

最后一次试驾

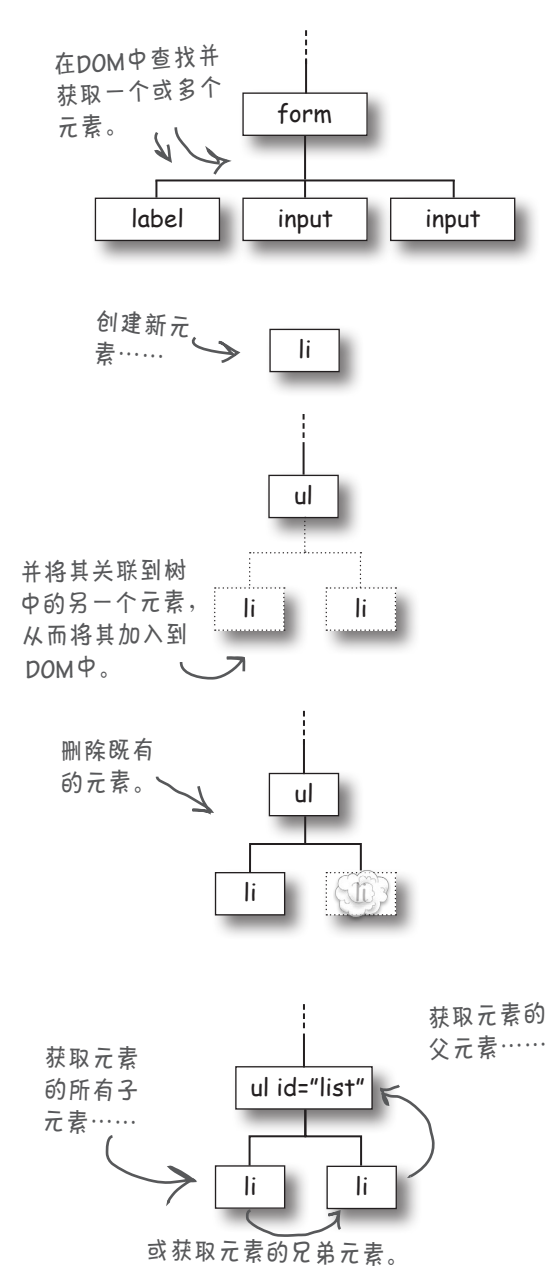


在浏览器中加载这个网页，你将看到绿色星球被移相武器（phaser）击中了，且消息为亮红色，相信你一定会注意到！



DOM还有哪些功能

除了前面介绍的以外，DOM还有很多其他功能，本书后面将介绍其中的一些。下面简单地说说，让它们深深地刻在你脑海中。



从DOM获取元素

这一点你早就知道了，因为我们一直在使用 `document.getElementById`，但还有其他获取元素的方式。事实上，你可以使用标签名、类名和特性来获取一系列元素（如所有归属于 `on_sale` 类的元素）。你还可获取用户输入的表单值，如 `input` 元素的文本。

创建元素并将其加入到DOM中

你可以创建新元素，并将它们加入到DOM中。当然，你对DOM所做的任何修改，都将在浏览器渲染DOM时立即反映出来。（这是件好事！）

从DOM删除元素

你还可以从DOM中删除元素，这将同时删除其所有子元素。同样，将元素从DOM删除后，它也将立即从浏览器窗口中消失。

遍历DOM中的元素

有了指向元素的句柄后，就可找到其所有子元素，可获取其兄弟元素（位于同一层级的所有元素），还可获取其父元素。DOM的结构类似于家谱！



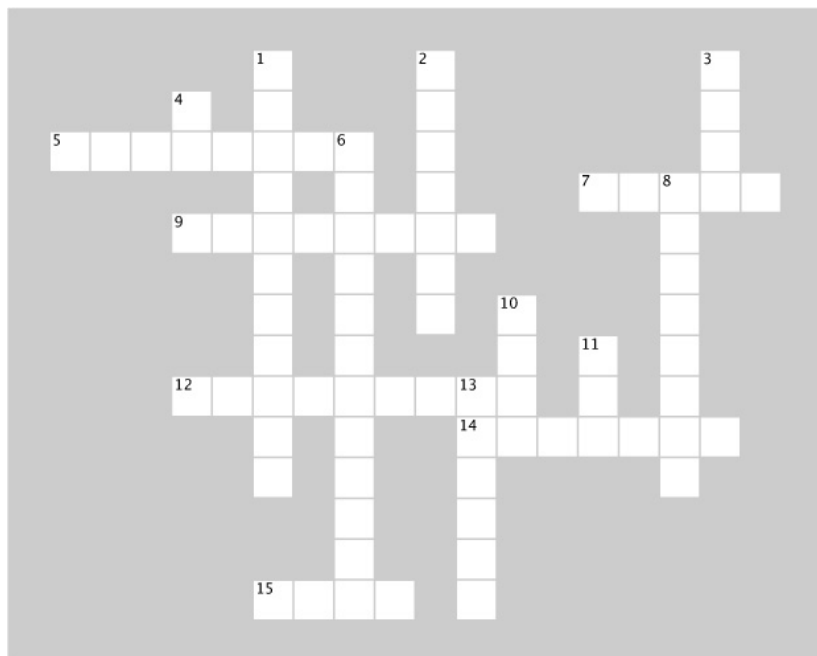
要点

- 在浏览器内部，使用文档对象模型（DOM）来表示网页。
- 浏览器在加载并分析HTML时创建网页的DOM。
- 在JavaScript代码中，使用document对象来访问DOM。
- document对象包含你可用来访问和修改DOM的属性和方法。
- 方法document.getElementById根据id从DOM获取一个元素。
- 方法document.getElementById返回一个表示网页中元素的**元素对象**。
- 元素对象包含一些属性和方法，你可使用它们来读取和修改元素的内容。
- 属性innerHTML包含元素的文本内容和全部嵌套的HTML内容。
- 要修改元素的内容，可修改其属性innerHTML的值。
- 通过修改属性innerHTML来修改元素时，所做的修改将立即在网页中反映出来。
- 要获取元素的特性值，可使用方法getAttribute。
- 要设置元素的特性值，可使用方法setAttribute。
- 如果将包含JavaScript代码的<script>元素放在网页的<head>元素中，就必须确保在网页加载完毕后再修改DOM。
- 可使用window对象的onload属性给加载事件指定**事件处理程序**（回调函数）。
- 网页加载完毕后，将立即调用window对象的onload属性指向的事件处理程序。
- 在JavaScript中，可使用事件处理程序来处理很多不同类型的事件。



JavaScript填字游戏

请完成这个填字游戏，将DOM知识深深地刻在你的脑海中。



横向

5. 处理事件的函数被称为事件_____（复数形式）。
7. 为破解Evel博士的密码提供了线索的元素的id。
9. 为处理加载事件，可将一个什么赋给属性window.onload?
12. 要修改元素内部的HTML，可使用元素对象的哪个属性?
14. 方法setAttribute所属的对象。
15. DOM的形状像什么?

纵向

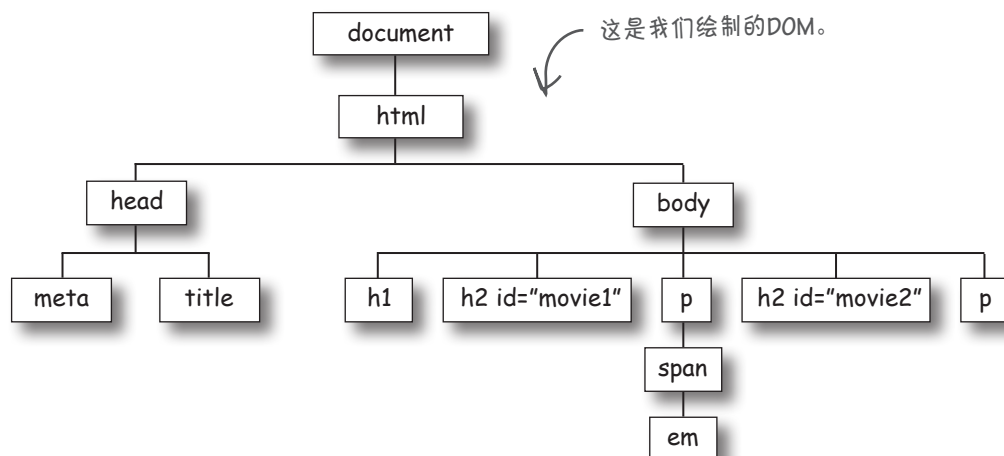
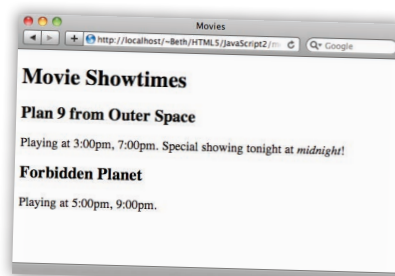
1. 被移相武器击中的星球。
2. 用于查看代码是否有错误的工具。
3. 使用代码来获取或修改网页的元素前，确保什么已加载完毕至关重要?
4. 方法getElementById根据什么来获取一个元素?
6. 要修改元素的属性class，可使用哪个方法?
8. 总是位于DOM树顶部的对象。
10. 使用getElementById时，务必检查它返回的是否是什么?
11. 在浏览器中加载网页时，浏览器创建一个什么来表示网页的所有元素和内容?
13. getElementById是document对象的一个什么?



变身浏览器 答案

你的任务是变身浏览器，亲自对HTML进行分析并制作DOM。现在就对右边的HTML进行分析，并在下面绘制DOM吧。开头部分已经为你绘制好了。

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Movies</title>
  </head>
  <body>
    <h1>Movie Showtimes</h1>
    <h2 id="movie1">Plan 9 from Outer Space</h2>
    <p>Playing at 3:00pm, 7:00pm.
      <span>
        Special showing tonight at <em>midnight</em>!
      </span>
    </p>
    <h2 id="movie2">Forbidden Planet</h2>
    <p>Playing at 5:00pm, 9:00pm.</p>
  </body>
</html>
```





下面的HTML显示一个歌曲播放列表，但该列表是空的。请补全下述在该列表中添加歌曲的JavaScript代码。答案如下。

`<!doctype html>` ← 这是网页的HTML。

`<html lang="en">`

`<head>`

`<meta charset="utf-8">`

`<title>My Playlist</title>`

脚本。这些代码在`<ul>`元素定义的列表中添加歌曲。

`<script>`

`function addSongs() {`

`var song1 = document.getElementById("song1");`

请补全这些代码，在播放列表中添加歌曲。

`var song2 = document.getElementById("song2");`

`var song3 = document.getElementById("song3");`

`song1.innerHTML = "Blue Suede Strings, by Elvis Pagely";`

`song2.innerHTML = "Great Objects on Fire, by Jerry JSON Lewis";`

`song3.innerHTML = "I Code the Line, by Johnny JavaScript";`

`}`

`window.onload = addSongs;`

`</script>`

`</head>`

`<body>`

`<h1>My awesome playlist</h1>`

`<ul id="playlist">`

空的歌曲列表。前述JavaScript代码将给播放列表中的每个`<li>`元素添加内容。

`<li id="song1"></li>`

`<li id="song2"></li>`

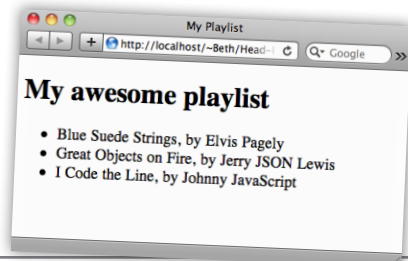
`<li id="song3"></li>`

`</ul>`

`</body>`

`</html>`

加载该网页后，它类似于这样。





磨笔上阵 答案

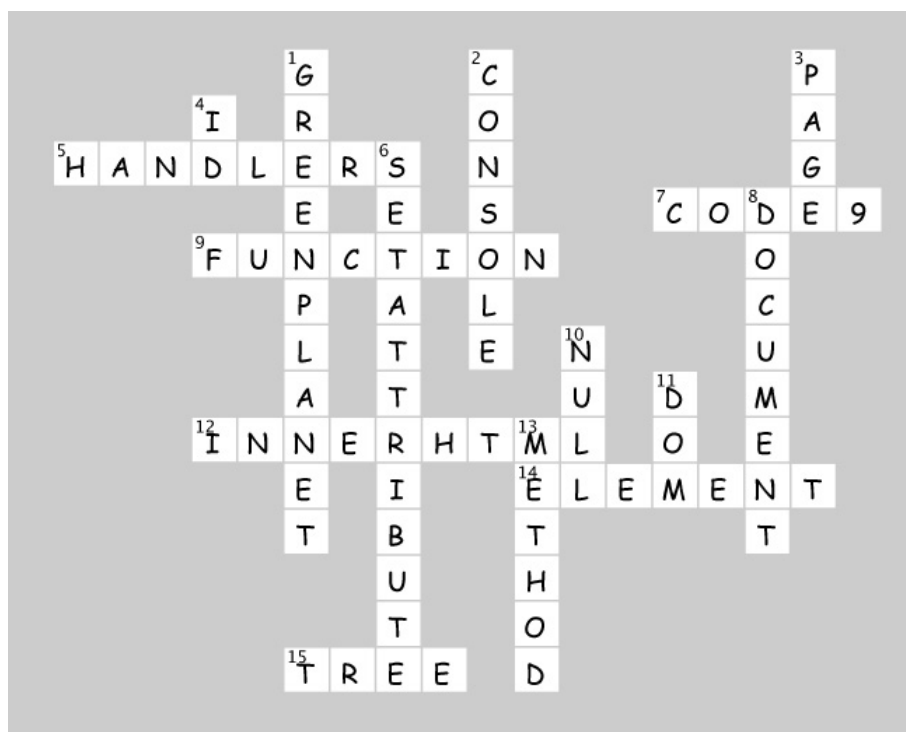
将大脑的另一部分也开动起来如何？请给redtext类定义CSS样式，将星球段落的文本设置为红色。如果你很久没有编写CSS，也不用担心，你完全可以试一试；如果你闭着眼睛都能编写出来，那就太好了。

```
.redtext { color: red; }
```



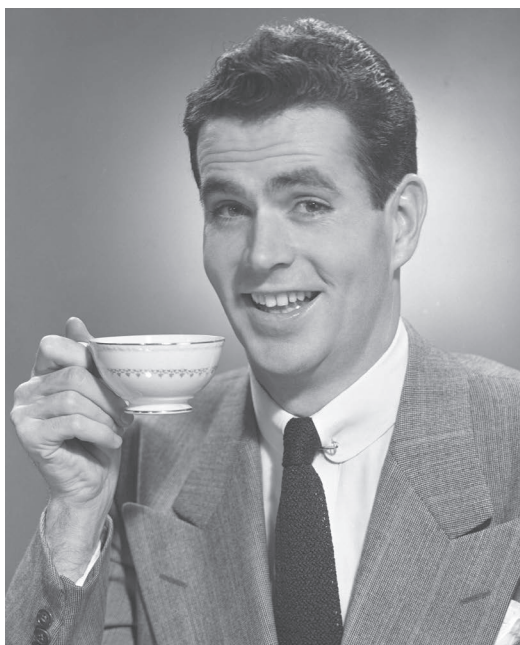
JavaScript填字游戏答案

请完成这个填字游戏，将DOM知识深深地刻在你的脑海中。



7 类型、相等、转换等

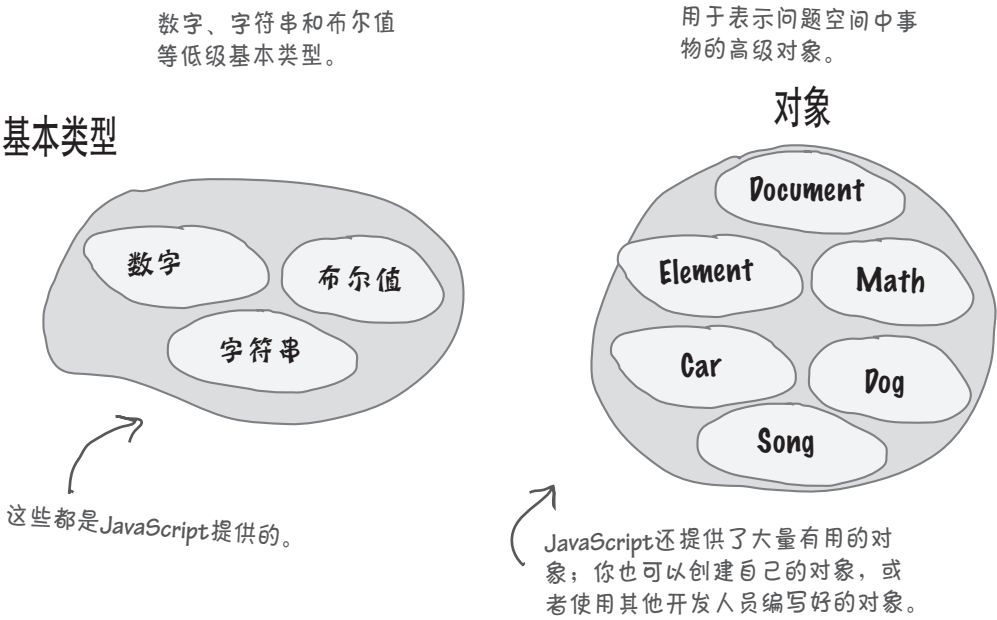
系统地讨论类型



该系统地讨论类型了。JavaScript的优点之一是，你无需知道这种语言的很多细节，就可以使用它来做很多事情。但要真正掌握这门语言、提升水平，做到人生中梦寐以求的事情，你必须对类型了如指掌。还记得本书前面是怎么说JavaScript的吗？是不是说它命不好，不是出生在高贵的学术殿堂，也未经同行审阅？确实如此，但非科班出身不妨碍史蒂夫·乔布斯和比尔·盖茨取得成功，对JavaScript来说亦如此。这确实意味着JavaScript的类型系统并不特别缜密，我们会发现它有一些怪癖。不过不用担心，本章将把这些怪癖解释得一清二楚，你很快就能避免各种与类型相关的难堪错误。

真相就摆在那里

你已经有丰富的JavaScript类型使用经验，你知道有数字、字符串和布尔值等基本类型，还知道有各种对象，其中有些是JavaScript提供的（如Math对象），有些是浏览器提供的（如document对象），还有些是你自己编写的。你不正沐浴在JavaScript简单、强大、一致的类型系统的光辉下吗？



有这样的Web镇官方语言，你还有什么不满足的呢？事实上，如果你只是脚本编写人员，可能考虑坐下来，喝杯Web镇生产的马提尼，放松一下早已疲惫不堪的身心……

但你不仅仅是脚本编写人员，总觉得还缺点什么。你隐隐地觉得，在Web镇尖桩篱栅的后面，有什么离奇的东西在起作用。你听人说字符串就像对象一样，你阅读过探讨null类型的博客，还有传言说JavaScript解释器近来一直在做一些怪异的类型转换。所有这一切都意味着什么呢？我们不知道，但真相就摆在那里，本章将把它揭示出来，这可能会完全颠覆你当前的看法。



猜猜我是谁



一大拨JavaScript值乔装打扮，正在玩一个“猜猜我是谁”的晚会游戏，其中还混入了一些不速之客。你需要根据它们对自己的描述猜测它们的身份——假设它们说的都是真话。请用箭头将每条描述连接到相应的参与者，我们已经猜出一个了。请将你的答案与本章末尾的答案对比，再继续往下阅读。

如果你发现这个练习很难，看看答案也没关系。

参与游戏的值：

我是没有`return`语句的函数返回的值。

我是未赋值的变量的值。

我是稀疏数组中不存在的数组元素的值。

我是不存在的属性的值。

我是已删除的属性的值。

我是创建对象时不能赋给属性的值。

零

空对象

`null`

`undefined`

`NaN`

`infinity`

`area 51`

`...---...`

`{ }`

`[]`

小心，你可能意外遭遇undefined

正如你看到的，在任何不确定的情况下——使用未初始化的变量、访问不存在（或已删除）的属性、使用不存在的数组元素时——你都将遭遇undefined。

undefined到底是什么？它其实并不复杂。你可以这么认为：对于任何还没有值（即还未初始化）的东西，都会将undefined赋给它。

这样做有什么好处呢？undefined让你能够判断是否给变量（属性或数组元素）赋值了。来看两个示例，先从未赋值的变量着手：



```
var x;
```

```
if (x == undefined) {  
    // x未定义！处理这种情况！  
}
```

可检查变量（如x）是否是未定义的。为此，只需将其与undefined进行比较。

请注意，这里使用的是值undefined，请不要将其与字符串“undefined”混为一谈。

对象的属性呢？

```
var customer = {  
    name: "Jenny"  
};  
if (customer.phoneNumber == undefined) {  
    // 获取客户的电话号码  
}
```

可检查属性是否是未定义的。为此，也可将其与undefined进行比较。

世上没有愚蠢的问题

问： 在什么情况下，需要检查变量（属性或数组元素）是否是未定义的？

答： 这取决于你的代码设计。在你编写的代码中，如果代码块执行时，属性或变量可能没有值，就需要检查它是否是undefined，避免在计算中使用未定义的值。

问： 既然undefined是一个值，那它有类型吗？

答： 有。undefined的类型是undefined。为什么？我们认为原因如下：它不是对象，不是数字、字符串或布尔值，也不是任何明确的东西。既然如此，为何不将这种类型也视为未定义的呢？这是JavaScript怪异的灰色地带之一，你不得不接受。

实验

在实验室，我们喜欢将东西拆开，看看里面的情况，接上诊断工具以搞清楚到底出了什么问题。今天，我们要研究的是JavaScript的类型系统，而我们已经找到了一个小小的诊断工具`typeof`，可用于检查变量。请你穿上实验服，戴上护目镜，进入实验室，与我们一道来做实验吧。

`typeof`是一个内置的JavaScript运算符，可用于探测其操作数（要对其进行操作的东西）的类型，如下例所示：



```
var subject = "Just a string";
var probe = typeof subject;
console.log(probe);
```

运算符`typeof`接受一个操作数，并返回该操作数的类型。

这里的类型是`string`。注意到`typeof`使用字符串来表示类型，如“`string`”“`boolean`”“`number`”“`object`”“`undefined`”等。



该你了。请收集下述实验的结果。

```
var test1 = "abcdef";
var test2 = 123;
var test3 = true;
var test4 = {};
var test5 = [];
var test6;
var test7 = {"abcdef": 123};
var test8 = ["abcdef", 123];
function test9(){return "abcdef"};

console.log(typeof test1);
console.log(typeof test2);
console.log(typeof test3);
console.log(typeof test4);
console.log(typeof test5);
console.log(typeof test6);
console.log(typeof test7);
console.log(typeof test8);
console.log(typeof test9);
```

这些是实验数据，而这些是实验。





我记得前一章说过，指定的id不存在时，`getElementById`返回null，而不是undefined。null到底是什么？`getElementById`为何不返回undefined呢？

这确实让人感到迷惑。在很多语言中，都有一个表示“无对象”的概念，这挺好。就拿方法`document.getElementById`来说吧，是不是要求它返回一个对象？如果它无法返回一个对象呢？在这种情况下，我们希望返回一个含义如下的值：要是对象，我就会是一个对象，可当前没有。这正是null的含义。

你还可直接将变量设置为null：

```
var killerObjectSomeday = null;
```

将变量设置为null是什么意思呢？意思是说我原本要将一个对象赋给这个变量，但现在没有这样的对象。

你可能挠着头说，为何不使用undefined来表达这种意思呢？很多人都这么想。原因是JavaScript刚面世时是这么想的：用一个值表示变量还未初始化，用另一个值表示没有对象可赋给变量。这并不完美，而且显得有些多余，但现实情况就是如此。你只需牢记undefined和null各自的用途，并知道下面一点就行了：在应该提供一个对象，但无法创建或找到时，将提供null；在变量未初始化、对象没有指定属性或数组没有指定元素时，将返回undefined。

回到实验室

哎呀，刚才的实验数据遗漏了null。下面是遗漏的实验：

```
var test10 = null;  
  
console.log(typeof test10);
```

将结果写在这里。

JavaScript控制台



如何使用null

很多函数和方法都返回对象，你需要确认获得的是货真价实的对象而不是null，以防函数无法找到或创建要返回的对象。在上一章，你见过需要进行这种检查的示例：

```
var header = document.getElementById("header");
if (header == null) {
    // 如果没有header元素，肯定出了大问题
}
```

咱们来查找非常重要的header元素。

哎呀！它不存在，赶快弃船！

别忘了，返回null未必意味着出了问题。这可能只是意味着有什么东西不存在，需要创建它或可以忽略它。假设你的网站有一个天气小部件，用户可以打开它，也可以关闭它。如果用户打开了它，就会有一个id为weatherDiv的<div>元素；否则，就没有这样的元素。突然之间，null便可派上大用场：

```
var weather = document.getElementById("weatherDiv");
if (weather != null) {
    // 为div元素weather创建内容
}
```

看看是否有id为weatherDiv的元素。

如果getElementById返回的不是null，就说明网页中有这样的元素，那么创建一个漂亮的天气小部件（假定在其中显示当地的天气情况）。

我们可以使用null来检查对象是否存在。

别忘了，null用于表示对象不存在。

密苏里州布莱恩市
温度始终保持在
19°C，降雨概率为
40%。



信不信由你！！

不是数字的数字



可以很容易地编写JavaScript语句来生成含义不明确的数值。

下面就是几条这样的语句：

```
var a = 0/0;
```

↑ 在数学中，这没有明确的答案，因此不能要求JavaScript知道答案。

```
var b = "food" * 1000;
```

↑ 我们不知道结果是什么，但肯定不是数字！

```
var c = Math.sqrt(-9);
```

↑ 如果你的高中数学不是体育老师教的，肯定知道负数的平方根是虚数，在JavaScript中无法表示。

信不信由你，有些数字值在JavaScript中无法表示！有鉴于此，JavaScript提供了一个替代值：

NaN

JavaScript使用NaN [通常被称为非数字 (Not a Number)] 来表示它无法表示的数值结果。就拿0/0来说吧，在计算机中无法表示其结果，因此JavaScript使用NaN来表示它。



NaN可能是世上最怪异的值。它不仅用于表示所有无法表示的数值，还是JavaScript中唯一一个与自己不相等的值！

你没听错。如果你将NaN和NaN进行比较，结果是不相等！

NaN != NaN

处理NaN

你可能认为，需要处理NaN的情形很少，但只要你编写任何使用数字的代码，都将惊讶地发现它现身的频率非常频繁。你经常需要检查NaN。基于学到的JavaScript知识，你可能认为如何检查显而易见：

```
if (myNum == NaN) {
  myNum = 0;
}
```

你可能认为这样做可行，但实际上行不通。

错误的做法！

敏锐的读者会认为，这就是检查变量是否为NaN的方式，但实际上行不通。为什么呢？因为NaN与任何东西（包括它自己）都不相等，因此不能以任何方式检查变量与NaN是否相等。相反，需要使用特殊函数isNaN，如下所示：

```
if (isNaN(myNum)) {
  myNum = 0;
}
```

使用函数isNaN，它在传入的值不是数字时返回true。

正确的做法！

比想象的还要怪异

来更深入地考虑一下。既然NaN指的是“不是数字”，那它是什么呢？如果指出它是什么，而非它不是什么，是不是更容易理解呢？那你认为它是什么呢？为获得一点线索，可检查其类型：

```
var test11 = 0 / 0;
console.log(typeof test11);
```

这是我们得到的结果。

JavaScript控制台
number

到底是怎么回事？NaN的类型是数字？一个不是数字的东西，其类型怎么可能是数字呢？别急，沉住气，你可以这样想：NaN是一个糟糕的名称；与其称之为“不是数字”，还不如称之为“无法表示的数字”（必须承认，这样首字母缩写将不那么出色）。如果你这样想，就可认为NaN是一个数字，只是无法表示而已（至少对计算机来说如此）。

请将这一点加入到JavaScript灰色地带中。

如果你一点也不吃惊，那么本书对你来说唯一的用途也许就是拿来点火了。

世上没有愚蠢的问题

问：向`isNaN`传递一个不能转换为数字的字符串时，它会返回`true`吗？

答：那是肯定的，与你期望的完全一致。如果变量存储的是NaN或其他任何不是数字的值，将其传递给`isNaN`时，它都将返回`true`（否则，将返回`false`）。但有几点需要注意，这将在后面讨论类型转换时介绍。

问：为何NaN与它自己不相等？

答：如果你对这一点很感兴趣，可参阅IEEE浮点数规范。对于这个问题，简单的答案是，因为NaN指的是无法表示的数字，但并非所有无法表示的数字都相同。例如，`sqrt(-1)`和`sqrt(-2)`的结果都是NaN，但它们显然不相同。

问：`0 / 0`的结果为NaN，而`10 / 0`的结果为Infinity。Infinity与NaN不是一回事吗？

答：好眼力。在JavaScript中，Infinity指的是任何超过浮点数上限1.7976931348623157E+10308（对-Infinity而

言，是指超过浮点数下限-1.7976931348623157E+10308）的值。Infinity的类型为数字，怀疑某个值太大时，可检查它是否为Infinity：

```
if (tamale == Infinity) {
    alert("That's a big tamale!");
}
```

问：得知NaN的类型是数字后，我确实很吃惊。还有其他令人吃惊的细节吗？

答：这个问题真有趣。将Infinity与它自己相减时，结果为NaN；这令你吃惊吗？如果连这一点都能明白，说明你的数学非常出色。

问：既然说到细节，null的类型是什么呢？

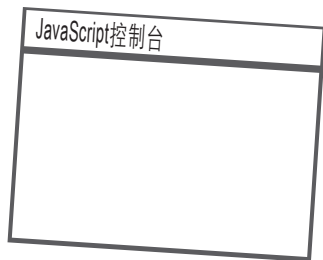
答：要迅速获得答案，可对null使用运算符`typeof`。如果你这样做，结果将为"object"。这合情合理，因为null用于表示不存在的对象。不过这存在很大的争议，在最新的规范中，null的类型为null。你可能发现，浏览器的JavaScript实现没有遵守这种规范，但很少需要在代码中确定null的类型。



本章一直在讨论有趣的值，下面来看看一些有趣的行为。请将下面的代码添加到一个基本网页的<script>元素中，再加载该网页，看看控制台显示的是什么。为什么会出现这种情况呢？你能猜出其中的原因吗？

```
if (99 == "99") {
    console.log("A number equals a string!");
} else {
    console.log("No way a number equals a string");
}
```

将你看到的输出写在这里。



有一点我们必须告诉你

对于JavaScript，有一点我们一直故意瞒着没有说。这一点原本早就可以说，但现在说最合适。

这并不意味着我们一直在欺骗你，而只是意味着还有些事情需要补充而已。是什么事情呢？咱们来看看：

设置一个变量，这里将其设置为数字99。



```
var testMe = 99;
```

然后，在条件测试中将其与一个数字进行比较。



```
if (testMe == 99) {  
    // 要执行的操作  
}
```

简单易懂吧？确实如此，还有比这更简单的吗？然而，本书前面至少做了一次类似于下面的事情，只是你可能没有注意到：

设置一个变量，这里将其设置为字符串“99”。



```
var testMe = "99";
```

我们说过这次使用的
是一个字符串吗？

然后，在条件测试中将其与一个数字进行比较。

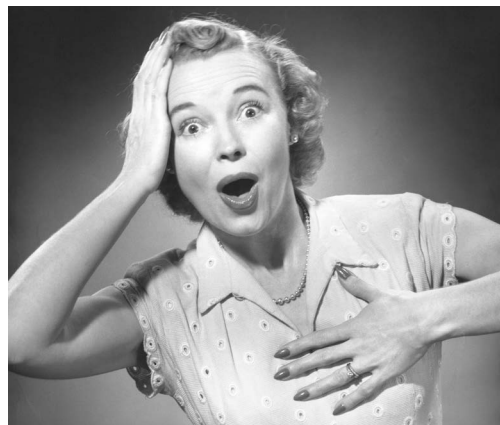


```
if (testMe == 99) {  
    // 要执行的操作  
}
```

将一个字符串与一个数字进行比较。

将数字与字符串进行比较时，结果将如何呢？太乱了？计算机崩溃？犹如大街上的骚乱？

非也，JavaScript很聪明，能够判断出99和“99”实际上是相等的。但它在幕后到底是如何作出这种判断的呢？咱们来看看。



要点

下面简单地复习一下赋值和相等的差别。

- **var x = 99;**
=是赋值运算符，用于给变量赋值。
- **x == 99**
==是比较运算符，用于对两个值进行比较，看看它们是否相等。

理解相等运算符

你可能认为，相等运算符很容易理解，不就是`1 == 1`、`"guacamole" == "guacamole"`、`true == true`吗？然而，对于`"99" == 99`，显然需要做更多的工作。相等运算符到底是如何进行比较的呢？

实际上，进行比较时，运算符`==`会考虑其操作数（要比较的两个值）的类型。这存在下面两种情况。

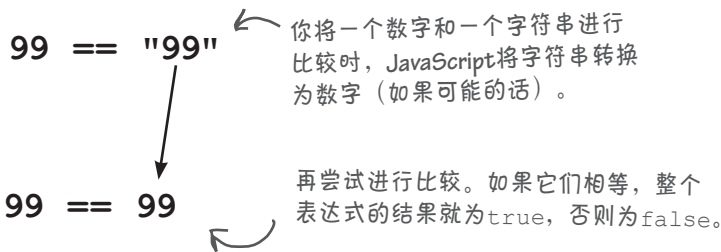
如果两个值的类型相同，就直接进行比较

如果要比较的两个值的类型相同，如比较两个数字或两个字符串，将按你期望的方式进行比较：将这两个值进行比较，如果它们相同，结果为`true`。非常简单。

如果两个值的类型不同，则尝试将它们转换为相同的类型，再进行比较

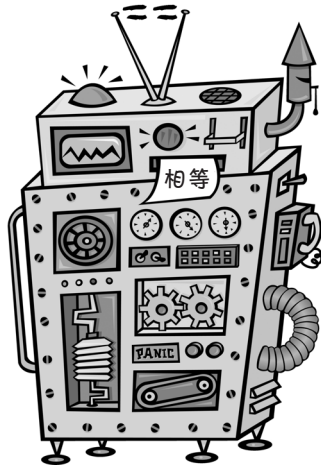
这种情况更有趣。假设你要对两个类型不同的值进行比较，如一个数字和一个字符串，JavaScript将把字符串转换为数字，再进行比较，如下所示：

← 请注意，这种转换是临时性的，旨在方便比较。



这让你有一定的直观感受，但其中的规则是什么呢？如果将布尔值和数字、`null`和`undefined`或其他值进行比较呢？我怎么知道将进行什么样的转换呢？为比较数字和字符串是否相等，为何不将数字转换为字符串，或采用其他转换方式呢？JavaScript规范包含一套非常简单的规则，指定了对两个不同类型的值进行比较时，将如何进行转换。你必须牢记这些规则，这样才能掌握JavaScript比较的工作原理。

← 这也将让你胜人一筹，在下次面试中脱颖而出。



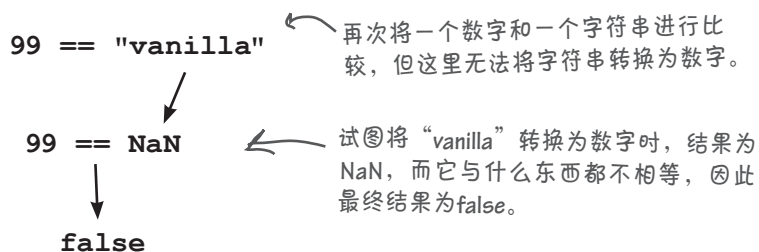
相等运算符如何转换操作数 (听起来比实际上危险)

我们知道，将两个类型不同的值进行比较时，JavaScript会把一个值转换为另一个值的类型，再进行比较。你可能觉得这很奇怪，因为在其他语言中，这通常不是自动进行的，而需要编写代码显式地进行转换。不过，不用担心，JavaScript自动类型转换通常是件好事，只要你明白转换是在何时以及如何进行的。下面就来了解这一点：转换是何时以及如何进行的。

来看4种简单的情形。

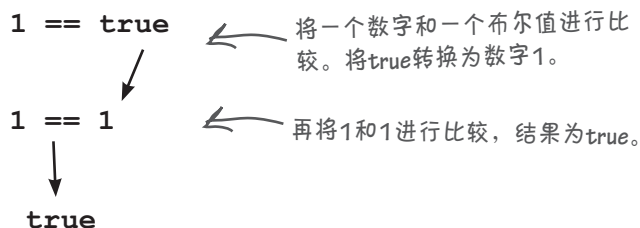
情形1：比较数字和字符串

比较字符串和数字时，都将把字符串转换为数字，再对两个数字进行比较。这并非总是可行，因为并非所有的字符串都可转换为数字。字符串不能转换为数字时，结果将如何呢？下面来看看：



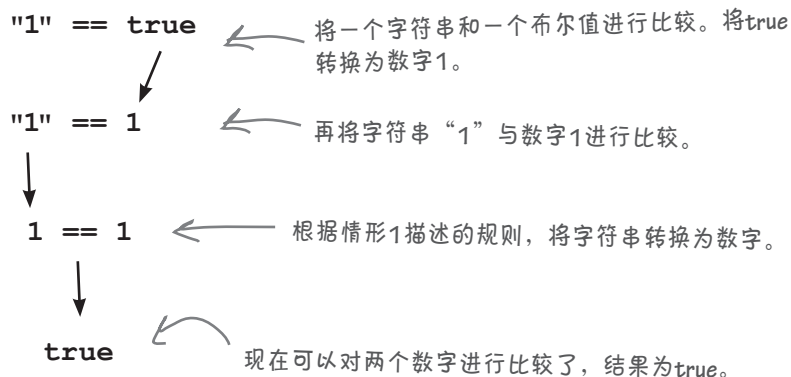
情形2：比较布尔值和其他类型

在这种情况下，将把布尔值转换为数字，再进行比较。这看起来有点怪，但只要记住true将被转换为1，而false将被转换为0，就容易理解了。另外，你必须明白，在有些情况下，需要做多次类型转换。下面是一个示例：



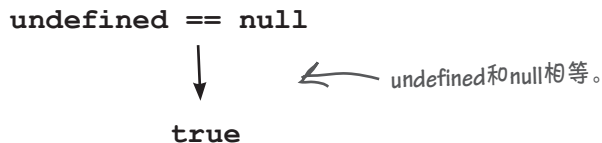
比较值

再看一个示例，这次将一个布尔值与一个字符串进行比较。注意这需要执行额外的步骤：



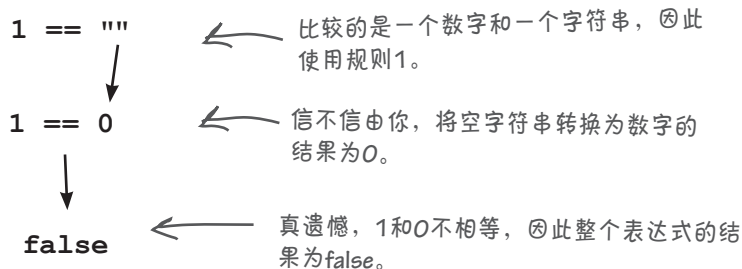
情形3：比较null和undefined

这两个值的比较结果为true。这好像有点怪，但规则就是这样的。你可以这么理解，它们其实表示的都是“没有值”（没有值的变量和没有值的对象），因此认为它们相等。



情形4：其实只有三种情形。

这就是所有的情形，根据这些规则，可确定任何相等表达式的值。然而，还存在一些边界情况和需要注意的地方。一个需要注意的地方是对象的比较，这将在本书后面介绍。另一个需要注意的地方是，有些转换可能让你猝不及防，例如：



如果对两个值进行比较时，无需操心类型转换的问题就好了；如果能够判断两个值是否相等且类型相同就好了；如果不用操心所有这些规则以及可能导致的错误就好了。我知道，这些都是白日做梦。



严格相等

我对比较的要求更严格些。

还有一点需要告诉你：相等运算符有两个，而不是只有一个。我们已经介绍了==（相等），另一个是===（严格相等）。

没错，就是三个等号。在可以使用==的任何地方，都可使用===，但在这样做之前，一定要明白它们之间有何不同。

你知道，使用==对两个类型不同的操作数进行比较时，涉及很多有关如何对操作数进行转换的规则。使用===时，涉及的规则更复杂。

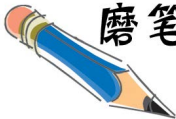
刚才才是跟你开玩笑的，===涉及的规则实际上只有一条。

当且仅当两个值的类型和值都相同时，它们才是严格相等的

请再读一遍。这意味着，如果两个值的类型相同，就对其进行比较；否则，不管它们是什么，结果都为false——不进行转换，也不考虑各种复杂的规则。你只需记住，仅当两个值的类型和值都相同时，===才认为它们相等。



↑ 编辑提醒：请确保获得了Doug Crockford允许使用这张照片的书面授权。



磨笔上阵

对于下面的比较表达式，在运算符==和===下方分别写出它们的结果：

	==	===
"42" == 42	<u>true</u>	<u>"42" === 42</u>
"0" == 0	<u></u>	<u>"0" === 0</u>
"0" == false	<u></u>	<u>"0" === false</u>
"true" == true	<u></u>	<u>"true" === true</u>
true == (1 == "1")	<u></u>	<u>true === (1 === "1")</u>

难度极高！

难度极高！

世上没有愚蠢的问题

问：如果将数字（如99）同无法转换为数字的字符串（如"ninety-nine"）进行比较，结果将如何？

答：JavaScript尝试将"ninety-nine"转换为数字，但以失败告终，结果为NaN。因此，这两个值不相等，结果为false。

问：JavaScript如何将字符串转换为数字？

答：它使用一种算法对字符串中的每个字符进行分析，并尝试将其转换为一个数字。因此，如果字符串为"34"，JavaScript将其中的字符"3"和"4"转换为3和4。你还可将类似于"1.2"这样的字符串转换为浮点数，因为JavaScript很聪明，知道这样的字符串可转换为数字。

问：如果我编写类似于"true" == true这样的代码，结果将如何呢？

答：这是对字符串和数字进行比较。根据相关的规则，JavaScript将首先把true转换为1，再对"true"和1进行比较。接下来，它将尝试将"true"转换为数字，但以失败告终，因此结果为false。

问：既然有运算符==和===，是否意味着也有运算符<=和<==以及>=和>==呢？

答：答案是否定的，没有运算符<==和>==，只有<=和>=。这些运算符只知道比较字符串和数字（true <= false没有意义），因此如果你使用它们来比较除数字和字符串外的其他值，JavaScript将尝试根据前面讨论的规则进行类型转换。

问：如果我编写代码99 <= "100"，结果将如何呢？

答：根据相关的规则，将把"100"转换为数字，再同99进行比较。因为99小于100，所以结果为true。

问：有运算符!==吗？

答：有。===比==严格，同样!==也比!=严格。有关===的规则也适用于!==，只是它检查是否不等，而非是否相等。

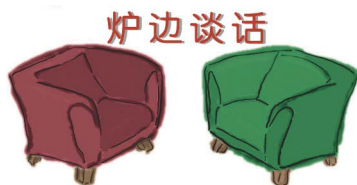
问：使用<和>比较布尔值和数字（如0 < true）时，也会使用同样的规则吗？

答：没错。就这个例子而言，将把true转换为1；因为0小于1，所以结果为true。

问：两个字符串相等好理解，但如何确定两个字符串谁大谁小呢？

答：问得好。说"banana"<"mango"是什么意思呢？比较两个字符串时，根据字母排列顺序来确定谁大谁小。由于"banana"以b打头，"mango"以m打头，而b在字母表中排在m前面，因此"banana"比"mango"小。"mango"比"melon"小，因为虽然它们的第一个字母相同，但比较第二个字母时，发现a排在e的前面。

然而，这种按字母排列顺序进行比较的方式可能让你犯错。例如，"Mango" < "mango"的结果为true，但你可能以为M比m大，因为前者是大写的。字符串的排列顺序取决于计算机中用于表示各个字符的Unicode值的排列顺序（Unicode是一种以数字方式表示字符的标准），而这种排列顺序并非总是与你的预期一致。想了解有关Unicode的详细信息，请使用关键字Unicode进行搜索。但在大多数情况下，你只需知道基本的字母排列顺序，就能判断两个字符串谁大谁小。



今晚主题：相等运算符和严格相等运算符争夺头把交椅。

==

跟我争，也不看看我是谁。

在现有的全部JavaScript代码中，我的使用量处于领先地位。你落后了，落后得还不是一星半点。

我不这么认为。我提供的服务非常重要，谁不需要时不时地将字符串格式的用户输入同数字进行比较呢？

上小学时，你每天上学和放学都必须爬雪坡吗？你做什么事情都避易就难？

现实情况是，你能做的比较我不仅都能做，还会进行类型转换。

难道不管三七二十一，都判断结果为false了事？

===

你别忘了，多位JavaScript大师都说了，开发人员应使用我，且只使用我。他们认为应将你从JavaScript中踢出去。

你说的也许没错，但开发人员逐渐醒悟过来了，这些统计数字正在发生变化。

要使用==，就必须牢记各种规则。使用===吧，这样生活和代码都更简单；如果要用户输入转换为数字，有提供这种方法。

真搞笑。行事严谨，确保比较语义清晰没有错；不牢记各种规则就可能出现意外才糟糕呢。

每次看到你那些规则，我都如鲠在喉。我是说要布尔值同任何值比较，都得将其转换为数字吗？在我看来，这太不合理了。

==

到目前为止，这确实管用。看看既有的JavaScript代码吧，很多都是脚本编写人员编写的。

你是说谈话后洗个澡什么的？

这是要让我出局的节奏？我乐意躺在沙滩上，喝着玛格丽塔悠闲地消磨时光。

争论==和===谁才是老大是在浪费时间。我是说人生中还有更有趣的事情要做。

现实情况是，大家绝不会不再使用==，有时候它确实很方便。大家可以合理地使用它，在合适的情况下充分发挥其优势。就拿处理用户输入来说吧，为何不使用==呢？

我现在的看法是，如果大家愿意用你，那很好，而我随时待命；顺便说一句，不管大家怎么选择，都不影响我每月领薪水！还有很多使用==的遗留代码——我永远都不会失业。

===

当然不是，我是说你那些复杂的规则可能让人不堪重负。

没错，但网页越来越复杂、越来越精致，该遵守一些最佳实践了。

比如坚持只使用===什么的。这让代码更清晰，还避免了在比较中出现古怪的边界情况。

我看不像呀，还以为你要占着头号等号运算符的宝座不放，直到永远呢。出什么事了？

我都不知道说什么好了。

我说过，世事难料。



好像还不够令人迷惑似的，
竟然有两个相等运算符。我
该使用哪个呢？

沉住气。这方面存在很大的争论，专家也是见仁见智。我们的观点是，传统上，由于对这两个运算符及其差别认识不足，程序员通常使用`==`（相等运算符）。现在，大家的认识更深刻了，在大多数情况下，使用`===`（严格相等运算符）都可行；从某种程度上说，这也更安全，因为你知道结果会是什么。当然，使用`==`时，你也知道结果会是什么，但考虑到它可能进行各种转换，有时候很难考虑到所有的可能性。

在有些情况下，如比较数字和字符串时，`==`提供了很大的便利。在这些情况下，应大胆地使用`==`，特别是考虑到你与很多JavaScript程序员都不同，对`==`的作用已了如指掌，因此更应如此。讨论`===`后，本书改弦易辙，在大多数情况下都使用`===`，但我们会灵活处理，在`==`能够简化工作且不会引入问题的情况下，也会使用它。

有些开发人员将`===`
（严格相等）称为等
同运算符。



下面是对一些运算符的描述，你能将它们与相应的运算符连起来吗？
请注意，每个运算符可能对应于0、1或多项描述。我们已经将一项描述与相应的运算符连起来了。

=

将两个值进行比较，看它们是否相等。它被称为相等运算符，必要时会对操作数进行类型转换，再进行比较。

==

将两个值进行比较，看它们是否相当；如果两个值的类型不同，将直接返回false。

===

给变量赋值。

====

比较两个对象引用；如果它们相同，就返回true，否则返回false。

深入探讨类型转换

并非只有条件语句会导致类型转换，其他几个运算符也可能导致类型转换。这些转换旨在为程序员提供便利，通常也确实如此，但你必须准确地知道类型转换可能在什么情况下和什么地方发生。下面就来看一看。

再谈拼接和加法运算符

你可能知道了，用于数字时，运算符+表示加法运算，而用于字符串时，表示拼接（concatenation）。然而，如果+的操作数的类型不同，结果将如何呢？下面就来看看。

如果你试图将数字和字符串相加，JavaScript将把数字转换为字符串，再进行拼接。这与==运算符的情况大致相反。

```
var addi = 3 + "4";
```

← 将数字和字符串相加时，执行的
是拼接，而不是加法运算。

```
var result = 3 + "4";
```

← 变量result被设置为“34”，
而不是7。

```
var plusi = "4" + 3;
```

← 同理，结果为“43”。

将数字和字符串相加时，即便将字符串放在前面，结果也相同：将数字转换为字符串，再进行拼接。

其他算术运算符呢

对于其他算术运算符，如乘法、除法和减法运算符，JavaScript将认为你要执行的是算术运算，而不是字符串运算。

```
var multi = 3 * "4";
```

← JavaScript将字符串“4”转换为数字4，再将其乘以3，结果为12。

```
var divi = 80 / "10";
```

← 将字符串“10”转换为数字10，再将80除以数字10，结果为8。

```
var mini = "10" - 5;
```

← 将字符串“10”转换为数字10，再将其减去5，结果为5。

世上没有愚蠢的问题

问：只要有一个操作数为字符串，就会将+视为字符串拼接吗？

答：是的。然而，由于+的结合性是从左到右的，如果你编写了下面的代码：

```
var order = 1 + 2 + " pizzas";
```

结果将为"3 pizzas"，而不是"12 pizzas"。这是因为从左往右计算时，先将1和2相加（因为它们都是数字），结果为3；接下来，将3与字符串"pizzas"相加，因此将3转换为字符串，再拼接这两个字符串。为确保得到想要的结果，可使用括号来指定先执行的运算。例如，下面的语句确保变量order存储的是"3 pizzas"：

```
var order = (1 + 2) + " pizzas";
```

而下面的语句确保变量order存储的是"12 pizzas"：

```
var order = 1 + (2 + " pizzas");
```

问：就这些吗？换句话说，还有其他可能发生类型转换的情形吗？

答：在其他一些情形下，也会导致类型转换。例如，运算符-对数字取负，将其用于true时，结果将为-1。另外，将布尔值与字符串相加时，结果为字符串。例如，true + "love"的结果为true love。这些情形很少出现，笔者在编写代码时从来不这样做，这里提及它们只是想让你知道而已。

问：要让JavaScript将字符串转换为数字，并与另一个数字相加，该如何办呢？

答：有一个将字符串转换为数字的函数，名为Number（没错，其中的N就是大写的）。可像下面这样使用它：

```
var num = 3 + Number("4");
```

这条语句导致变量num的值为7。函数Number接受一个实参，并将其转换为数字。如果指定的实参无法转换为数字，Number将返回NaN。



磨笔上阵

来检查一下你的类型转换知识吧。对于下面的表达式，在旁边的空白处写出其结果。我们已经完成了一个。请查看本章末尾的答案，再接着往下阅读。

Infinity - "1" _____

"42" + 42 "4242" _____

2 + "1 1" _____

99 + 101 _____

"1" - "1" _____

console.log("Result: " + 10/2) _____

3 + " bananas " + 2 + " apples" _____



还有一点没有讨论，那就是对象的相等性。例如，两个对象相等意味着什么呢？

很高兴你想到了这一点。说到对象相等性，可从简单的角度考虑，也可从复杂的角度考虑。从简单的角度考虑时，它指的是这个对象与那个对象是否相等，即如果有两个指向对象的变量，它们指向的是同一个对象吗？这将在下一页讨论。从复杂的角度考虑时，将涉及对象的类型，即如何判断两个对象的类型相同。你知道，可创建看起来是相同类型的对象，如两个汽车对象，但如何判断它们的类型确实相同呢？这个问题很重要，将在本书后面讨论。

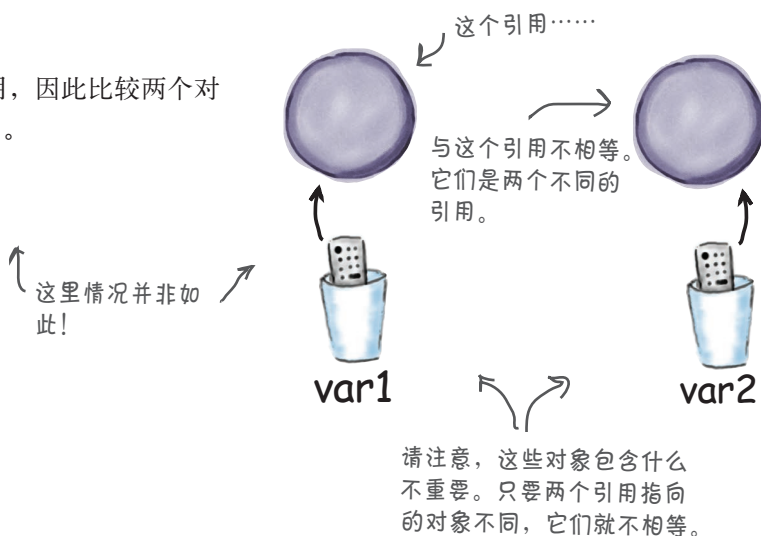
如何判断两个对象是否相等

你可能会问：该使用==还是===呢？好消息是，比较两个对象时，使用哪个运算符都没有关系。换句话说，如果两个操作数都是对象，可使用==，也可使用===，因为在这种情况下，它们的工作原理完全相同。检查两个对象是否相等时，情况如下：

检查两个对象变量是否相等时，比较的是指向对象的引用

别忘了，对象变量存储的是指向对象的引用，因此比较两个对象变量时，实际上比较的是指向对象的引用。

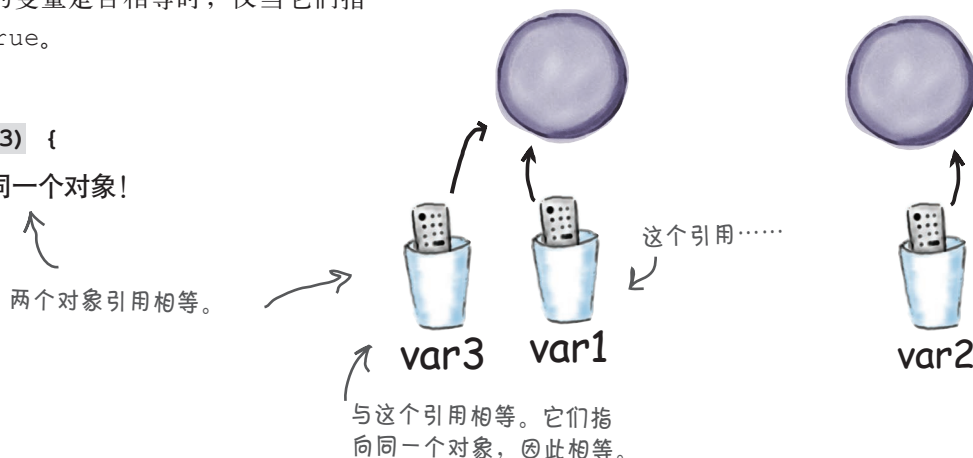
```
if (var1 === var2) {
    // 哇，它们是同一个对象！
}
```

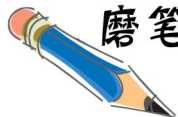


仅当两个引用指向的是同一个对象时，它们才相等

检查两个包含对象引用的变量是否相等时，仅当它们指向同一个对象才会返回true。

```
if (var1 === var3) {
    // 哇，它们是同一个对象！
}
```





磨笔上阵

下面的代码用于在Earl停车场寻找汽车，请据此写出loc1到loc4的值。

```
function findCarInLot(car) {
    for (var i = 0; i < lot.length; i++) {
        if (car === lot[i]) {
            return i;
        }
    }
    return -1;
}

var chevy = {
    make: "Chevy",
    model: "Bel Air"
};

var taxi = {
    make: "Webville Motors",
    model: "Taxi"
};

var fiat1 = {
    make: "Fiat",
    model: "500"
};

var fiat2 = {
    make: "Fiat",
    model: "500"
};
```

```
var lot = [chevy, taxi, fiat1, fiat2];
```

```
var loc1 = findCarInLot(fiat2);
var loc2 = findCarInLot(taxi);
var loc3 = findCarInLot(chevy);
var loc4 = findCarInLot(fiat1);
```

在这里写出你的答案。

Earl停车场的
工作人员



真值就摆在那里

没错，说的就是真值（truthy），而不是真相。我们还将谈到假值（falsey或falsy）。我们到底要说什么呢？在有些语言中，true和false的定义非常严格，但JavaScript没那么严格。事实上，JavaScript对true和false的定义比较宽松。怎么个宽松法呢？在JavaScript中，有些值并非true或false，但用于条件表达式中时，被视为true或false。我们将这些值称为真值和假值，因为严格地说它们并非true或false，但用于条件表达式时，它们的行为像true或false。



理解真值和假值的秘诀如下：知道哪些值是假值后，就可将其他值就视为真值。来看一些在条件表达式中使用假值的示例：

```
var testThis;
if (testThis) {
    // 执行某种操作
}
```

← 这个条件测试有点怪，其中的变量为undefined。这可行吗？这是有效的JavaScript条件表达式吗？（答案是肯定的）。

```
var element = document.getElementById("elementThatDoesntExist");
if (element) {
    // 执行某种操作
}
```

← element的值为null。这是要做什么呀？

```
if (0) {
    // 执行另一种操作
}
```

← 检查0是否为true？

```
if ("") {
    // 这些代码会被执行吗？你猜猜。
}
```

← 将一个空字符串用于条件测试。有人愿意猜猜结果吗？

```
if (NaN) {
    // 在布尔测试中使用NaN的结果会是什么呢？
}
```

← 现在要在布尔条件中使用NaN？那么结果会是什么呢？

JavaScript将哪些值视为假值

再次重申，要知道哪些值是真值，哪些值是假值，只需知道哪些值是假值，余下的其他所有值就都是真值。

在JavaScript中，假值有5个：

`undefined`

`null`

`0`

空字符串

`NaN`

**要记住哪些值是真值，
哪些值是假值，只需记住5个假值：`undefined`、`null`、`0`、“”和`NaN`，
其他的值都是真值。**

因此，前一页的每个条件测试都为`false`。前面说过，除假值外的其他值都是真值（当然，不包括`false`）。下面是一些真值示例：

```
if ([]) {  
    // 将执行这些代码  
}
```

↓ 这是一个数组。它不是`undefined`、`null`、`0`、“”或`NaN`，因此结果必然为`true`。

```
var element = document.getElementById("elementThatDoesExist");  
if (element) {  
    // 将执行这些代码  
}
```

↑ 这是一个`element`对象，它不是假值，因此必然为真值。

```
if (1) {  
    // 将执行这些代码  
}
```

← 只有数字`0`是假值，其他数字都是真值。

```
var string = "mercy me";  
if (string) {  
    // 还将执行这些代码  
}
```

↙ 只有空字符串为假值，其他字符串都是真值。



磨笔上阵

该来用用测谎仪了。判断哪些值是真值，哪些值是假值，找出犯罪嫌疑人撒了多少次谎，进而判断指控的罪名是否成立。然后，查看本章末尾的答案，紧接着往下阅读。当然，你也可以在浏览器中尝试这些测试。

```
function lieDetectorTest() {
    var lies = 0;

    var stolenDiamond = { };
    if (stolenDiamond) {
        console.log("You stole the diamond");
        lies++;
    }
    var car = {
        keysInPocket: null
    };
    if (car.keysInPocket) {
        console.log("Uh oh, guess you stole the car!");
        lies++;
    }
    if (car.emptyGasTank) {
        console.log("You drove the car after you stole it!");
        lies++;
    }
    var foundYouAtTheCrimeScene = [ ];
    if (foundYouAtTheCrimeScene) {
        console.log("A sure sign of guilt");
        lies++;
    }
    if (foundYouAtTheCrimeScene[0]) {
        console.log("Caught with a stolen item!");
        lies++;
    }
    var yourName = " ";  一个只包含一个空格的字符串。
    if (yourName) {
        console.log("Guess you lied about your name");
        lies++;
    }
    return lies;
}

var numberOfLies = lieDetectorTest();
console.log("You told " + numberOfLies + " lies!");
if (numberOfLies >= 3) {
    console.log("Guilty as charged");
}
```





你认为这些代码是做什么的？根据你的基本类型知识，你觉得这些代码有什么怪异的地方吗？

```
var text = "YOU SHOULD NEVER SHOUT WHEN TYPING";
var presentableText = text.toLowerCase();
if (presentableText.length > 0) {
    alert(presentableText);
}
```

字符串的神秘生活

每种类型都归属于两个阵营之一：要么是基本类型，要么是对象。基本类型的生活非常简单，而对象记录了状态，还拥有行为（换句话说，它们有属性和方法）。

这些说法都没错，但并不是故事的全部。事实上，字符串比较神秘，请看下面的代码：



```
var emot = "XOxxOO";
var hugs = 0;
var kisses = 0;

emot = emot.trim();
emot = emot.toUpperCase();

for(var i = 0; i < emot.length ; i++) {
    if (emot.charAt(i) === "X") {
        hugs++;
    } else if (emot.charAt(i) == "O") {
        kisses++;
    }
}
```

这好像是一个正常的字符串。

等等，对字符串调用方法？

字符串还有属性？

还有其他方法？

为什么字符串既像基本类型 又像对象

字符串是如何表现得既像基本类型又像对象的呢？因为JavaScript提供了这样的支持。换句话说，在JavaScript中，可创建作为基本类型的字符串，也可创建作为对象的字符串（支持大量的字符串操作方法）。本书前面从未谈及如何创建作为对象的字符串；在大多数情况下，你无需显式地这样做，因为JavaScript解释器会在需要时替你创建字符串对象。

JavaScript解释器会在什么情况下这样做呢？它为什么这样做呢？来看看字符串的生活：

```
var name = "Jenny";
var phone = "867-5309";
var fact = "This is a prime number";

var songName = phone + "/" + name;

var index = phone.indexOf("-");
if (fact.substring(10, 15) === "prime") {
    alert(fact);
}
```

创建三个基本类型字符串，
并将它们赋给变量。

将一些基本类型字符串拼
接起来，生成另一个基本
类型字符串。

调用一个方法。此时，JavaScript在
幕后将phone暂时转换为字符串对象。

同样，字符串fact被暂时
转换为对象，以支持方法
substring。

再次使用字符串
fact，但这次不需要
对象，因此它恢复为
基本类型字符串。

基本类型字符串

功能强大的对象



这看起来非常令人迷惑。字符串不断地在基本类型和对象之间转换？该如何跟踪字符串是基本类型还是对象呢？

你不需要这样做。一般而言，将字符串视为对象就好，其中包含大量帮助操作其文本的方法。JavaScript会负责处理所有的细节。因此，你可以这样想：你对JavaScript幕后的情况有更深入的认识，但在日常编码工作中，完全依赖JavaScript来做正确的事情（它不会辜负你的厚望）。

世上没有愚蠢的问题

问：我想确认一下，需要跟踪字符串何时是基本类型、何时是对象吗？

答：大多数情况下不需要，JavaScript解释器会为你处理所有的转换。你只管编写代码，假定字符串支持属性和方法，一切都将按期望的进行。

问：JavaScript为何支持字符串既可作为基本类型，又可作为对象呢？

答：你这样想想吧：只执行基本的字符串操作时，如比较、拼接、写入DOM等，可获得将字符串作为基本类型的效率；但需要做更复杂的字符串处理时，也有现成的字符串对象可用。

问：给定任意一个字符串，如何判断它是对象还是基本类型？

答：除非以特殊的方式（使用本书后面将讨论的对象构造函数）创建，否则字符串都是基本类型。可随时使用运算符typeof来判断一个变量是字符串还是对象。

问：其他基本类型也可以像对象一样吗？

答：是的，数字和布尔值也可以像对象一样，但它们不像字符串那样有那么多属性，你也不会像字符串那样经常将它们用作对象。别忘了，转换都是在幕后自动完成的，因此你真的不用考虑那么多；只管在需要时使用属性，让JavaScript替你处理临时转换就是了。

问：如何了解字符串对象的所有方法和属性？

答：这是优秀文档的用武之地。有很多很有用的在线文档，如果你更喜欢看书，可参阅《JavaScript权威指南》，其中介绍了JavaScript字符串的每个属性和方法。你还可以在网上进行搜索。

字符串方法和属性简明教程

考虑到我们正在讨论字符串，而你刚得知字符串也支持方法，下面就暂停对类型的讨论，介绍一些你可能用到的常见字符串方法。有几个字符串方法经常要用到，值得花时间来了解它们，现在就开始吧。

几句打气的話：我们原本可以专辟一章，详细介绍字符串支持的每个方法和属性。如果这样做，本书将长达2000页，重得像板砖；但当前这对你来说并非必不可少——你已掌握了方法和对象的基本知识，如果你真的想深入了解字符串处理的细节，只需找一部优秀的参考指南即可。

属性length

属性length指出了字符串包含多少个字符，为迭代字符串中的字符提供了极大的便利。

```
var input = "jenny@wickedlysmart.com";
for(var i = 0; i < input.length; i++) {
  if (input.charAt(i) === "@") {
    console.log("There's an @ sign at index " + i);
  }
}
```

方法charAt获取字符串中指定索引处的字符。

使用属性length来迭代字符串中的每个字符。

JavaScript控制台

There's an @ sign at index 5

方法charAt

方法charAt将一个整数作为参数（这个参数必须在0和字符串长度减1之间），并返回一个字符串，其中包含指定位置处的字符。可以认为字符串有点像数组，其中每个字符都有对应的索引，而索引从0开始（与数组一样）。如果指定的索引大于或等于字符串的长度，这个方法将返回一个空字符串。

请注意，JavaScript没有字符类型，因此通过返回一个字符串来返回字符，其中只包含指定的字符。

方法indexOf

这个方法将一个字符串作为参数，并在字符串中该参数首次出现的位置返回该参数中第一个字符的索引。

要对其调用indexOf的字符串。

```
var phrase = "the cat in the hat";
```

我们的目标是找到这条短语中的第一个“cat”。

```
var index = phrase.indexOf("cat");
```

```
console.log("there's a cat sitting at index " + index);
```

返回第一个“cat”中字符c的索引。

JavaScript控制台

There's a cat sitting at index 4

你还可指定第二个参数。它是一个索引，指定从什么位置开始查找。

```
index = phrase.indexOf("the", 5);
```

```
console.log("there's a the sitting at index " + index);
```

由于指定了从索引5开始查找，因此忽略了第一个“the”，找到的是第二个“the”。它位于索引11处。

JavaScript控制台

There's a the sitting at index 11

```
index = phrase.indexOf("dog");
```

```
console.log("there's a dog sitting at index " + index);
```

请注意，如果没有找到指定的字符串，将返回索引-1。

JavaScript控制台

There's a dog sitting at index -1

方法substring

方法substring将两个索引作为参数，提取并返回这两个索引之间的子串。

```
var data = "name|phone|address";
var val = data.substring(5, 10);
console.log("Substring is " + val);
```

要对其调用substring的字符串。

← 我们想要返回从索引5到索引10（不包括）的子串。

↑ 我们获得了一个新的字符串，其中包括从索引5到索引10的字符。

JavaScript控制台
Substring is phone

你可以省略第二个参数；在这种情况下，substring将提取从指定索引到字符串末尾的子串。

```
val = data.substring(5);
console.log("Substring is now " + val);
```

JavaScript控制台
Substring is now phone|address

方法split

方法split将一个用作分隔符的字符作为参数，并根据这个分隔符将字符串分成多个部分。

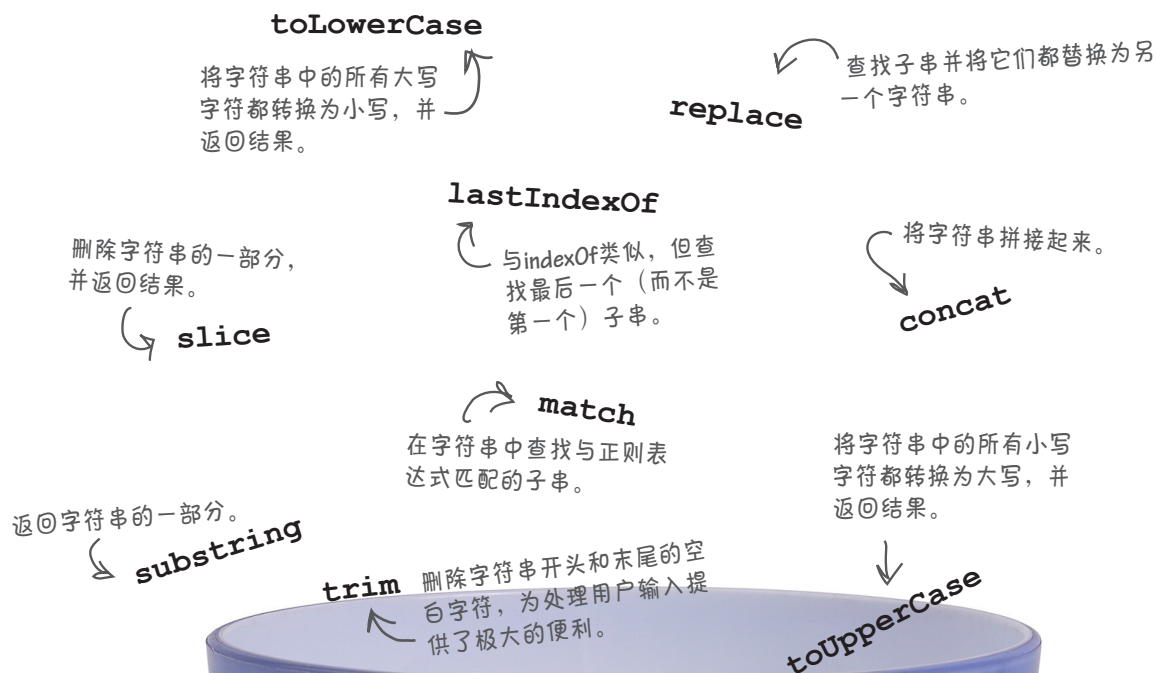
```
var data = "name|phone|address";
var vals = data.split("|");
console.log("Split array is " + vals);
```

← split根据分隔符将字符串分成多个部分，并返回一个包含这些部分的数组。

注意到这里向console.log传递了两个用逗号分隔的实参。这样在控制台显示数组vals前，不会将其转换为字符串。

JavaScript控制台
Split array is ["name", "phone", "address"]

字符串方法 群英谱



可对字符串执行的操作非常多，怎么学也学不完。这里列出了其他一些方法，你现在只需大致了解一下，等真的需要时再去了解详情。

座椅争夺战

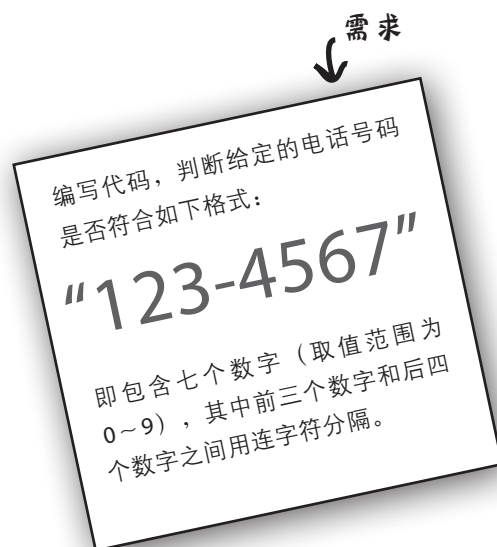
(熟悉类型可能改变你的生活)

有一家软件公司要求两位程序员编写实现指定需求的代码。为让他们尽快交付代码，讨厌的项目经理承诺，谁先交付代码谁就将得到硅谷人手一把的Aeron座椅。公司的台柱子Brad是个脚本编写高手，而Larry刚大学毕业，他们都认为这是小菜一碟。

Larry坐在小隔间里想：代码必须做些什么事情呢？需要确认字符串足够长，需要确认中间是一个连字符，还需确认其他所有字符都是数字。为此，可使用属性length来获取字符串的长度，并使用方法charAt来访问每个字符。

与此同时，Brad悠闲地喝着咖啡并盘算着：代码必须做些什么事情呢？他首先想到的是，字符串是一个对象，有很多方法可用来帮助验证电话号码。我得研究这些方法，迅速将程序编写出来。毕竟，对象就是对象。

请接着往下看，看看Brad和Larry是如何编写程序的，并找到问题的答案：谁得到了Aeron座椅？



在Larry的小隔间里

Larry开始使用字符串方法编写代码，很快他就写出了下面的代码：

```
function validate(phoneNumber) {
  if (phoneNumber.length !== 8) {
    return false;
  }
  for (var i = 0; i < phoneNumber.length; i++) {
    if (i === 3) {
      if (phoneNumber.charAt(i) !== '-') {
        return false;
      }
    } else if (isNaN(phoneNumber.charAt(i))) {
      return false;
    }
  }
  return true;
}
```

Larry使用属性length来确定字符串包含多少个字符。
 他使用方法charAt来检查字符串中的每个字符。
 首先，他确认索引3处的字符为连字符。
 然后，他确认索引0~2和4~6处的字符都是数字。

在Brad的小隔间里

Brad编写了如下代码，这些代码检查第一部分和第二部分是否是数字，还检查中间是否是连字符：

```
function validate(phoneNumber) {
  if (phoneNumber.length !== 8) {
    return false;
  }
  var first = phoneNumber.substring(0,3);
  var second = phoneNumber.substring(4);
  if (phoneNumber.charAt(3) !== "-" || isNaN(first) || isNaN(second)) {
    return false;
  }
  return true;
}
```

Brad开头的做法与Larry类似。

但利用了自己掌握的字符串方法的知识。

他使用方法substring创建一个字符串，其中包含索引0~3处的三个字符。

他如法炮制，再创建一个字符串，其中包含从索引4开始到末尾的字符。

接下来，他在一个条件表达式中检查正确的电话号码必须满足的所有条件。

有趣的是，他还有意或无意地利用自动类型转换将字符串转换为数字，再使用isNaN判断得到的是否是数字。非常巧妙！

等等，需求变了

项目经理说：“Larry，严格地说先完成的是你，因为Brad花了一些时间研究各种方法的使用，但我们需要轻微地修改一下需求。对你们两个程序员高手来说，这根本不是问题。”

Larry知道这不过是漂亮的说辞，他心里想：“如果每次听到这句话就能得到一毛钱，我早就是富翁了。可Brad看起来非常平静，到底是怎么回事呢。”Larry依然深信，Brad不过是故作镇定罢了，下一轮自己也将先编写好代码，再次获胜。



等等，你认为Brad这样使用isNaN是否可能引入bug？

编写代码，判断给定的电话号码是否符合如下格式：

"123-4567"

即包含七个数字（取值范围为0~9），在前三个数字和后四个数字之间，可以用连字符分隔，也可以不用。

新的需求

回到Larry的小隔间

Larry认为，原来的大部分代码都可以接着用，只需再处理包不包含连字符的边界情况：电话号码要么只包含7位数字，要么在前三位和后四位之间还有一个连字符。很快，Larry就添加了额外的逻辑（并进行简单测试以确保正确无误）：

```
function validate(phoneNumber) {
  if (phoneNumber.length > 8 ||
      phoneNumber.length < 7) {
    return false;
  }
  for (var i = 0; i < phoneNumber.length; i++) {
    if (i === 3) {
      if (phoneNumber.length === 8 &&
          phoneNumber.charAt(i) !== '-') {
        return false;
      } else if (phoneNumber.length === 7 &&
                  isNaN(phoneNumber.charAt(i))) {
        return false;
      }
    } else if (isNaN(phoneNumber.charAt(i))) {
      return false;
    }
  }
  return true;
}
```

Larry必须添加这些额外的逻辑。代码不多，但理解起来有点难。

海滩上，Brad坐在笔记本电脑前

Brad面带微笑，喝着玛格丽塔，很快就完成了修改。为获得电话号码的第二部分，他将电话号码的长度减去4，得到第二部分的起始索引（不再假定存在一个连字符），进而以硬编码的方式指定起始索引。这差不多就改好了，只是还需重新编写检查连字符的代码，因为仅当电话号码包含8个字符时，才需要检查连字符。

```
function validate(phoneNumber) {  
    if (phoneNumber.length > 8 ||  
        phoneNumber.length < 7) {  
        return false;  
    }  
    var first = phoneNumber.substring(0,3);  
    var second = phoneNumber.substring(phoneNumber.length - 4);  
  
    if (isNaN(first) || isNaN(second)) {  
        return false;  
    }  
    if (phoneNumber.length === 8) {  
        return (phoneNumber.charAt(3) === "-");  
    }  
    return true;  
}
```

修改的数量与Larry几乎相同，但Brad的代码依然更容易理解。

为获取第二部分，Brad根据电话号码的长度来计算起始索引。

仅当电话号码包含8个字符时，才检查第4个字符是否是连字符。

返回测试条件的结果——true或false。

考考你的
脑力

我们认为Brad的代码依然存在bug，你能找出来吗？

考考你的
脑力

要转而使用方法split，该如何重写Brad的代码呢？

Larry比Brad抢先一步进入项目经理的办公室

讨厌的项目经理说：“Brad，你的代码很容易理解和维护，干得好。”Larry脸上得意的笑容消失得无影无踪。

不过，Larry也不用太担心，因为我们都知道，代码漂亮与否并非唯一的考虑因素。Brad的代码需要详尽地测试，因为不确定它们在任何情况下都管用。你怎么认为呢？你认为谁该得到Aeron座椅呢？

急死我了，到底谁得到了座椅



二楼的Amy。
(大家都不知道，项目经理将需求交给了三位程序员。)

哇，只有一行代码！有关这些代码的工作原理，请参阅附录！

```
function validate(phoneNumber) {  
    return phoneNumber.match(/^\d{3}-?\d{4}$/);  
}
```

这是Amy编写的代码。

继续实验

实验人员继续使用运算符`typeof`来研究JavaScript，发现了这门语言中另外一些有趣的地方。在实验过程中，他们发现了另一个运算符`instanceof`。有了这个运算符，他们就能跻身最前沿的行列。请穿上实验服，戴上护目镜，看看你能否帮忙破解下面的JavaScript代码及其结果。请注意，这些绝对是本书到目前为止最怪异的代码。



代码如下。请阅读、运行、修改它们，看看它们是做什么的。

```
function Duck(sound) {
  this.sound = sound;
  this.quack = function() {console.log(this.sound);}
}

var toy = new Duck("quack quack");

toy.quack();

console.log(typeof toy);
console.log(toy instanceof Duck);
```

好怪异。有点像函数，又有点像对象。

哇，new，以前没见过。估计是新建一个Duck，并将其赋给变量toy。

如果看起来像对象，走起来像对象，那它就是对象。下面来验证这一点。

这是运算符`instanceof`。

务必将你认为的输出与本章末尾的答案进行比对。这些代码到底是什么意思呢？两章后将全面介绍。你可能没有意识到，你已经在通往高级JavaScript程序员之路上走了很远。我可没开玩笑！

JavaScript控制台

将结果写在这里。有什么让你感到惊讶的地方吗？





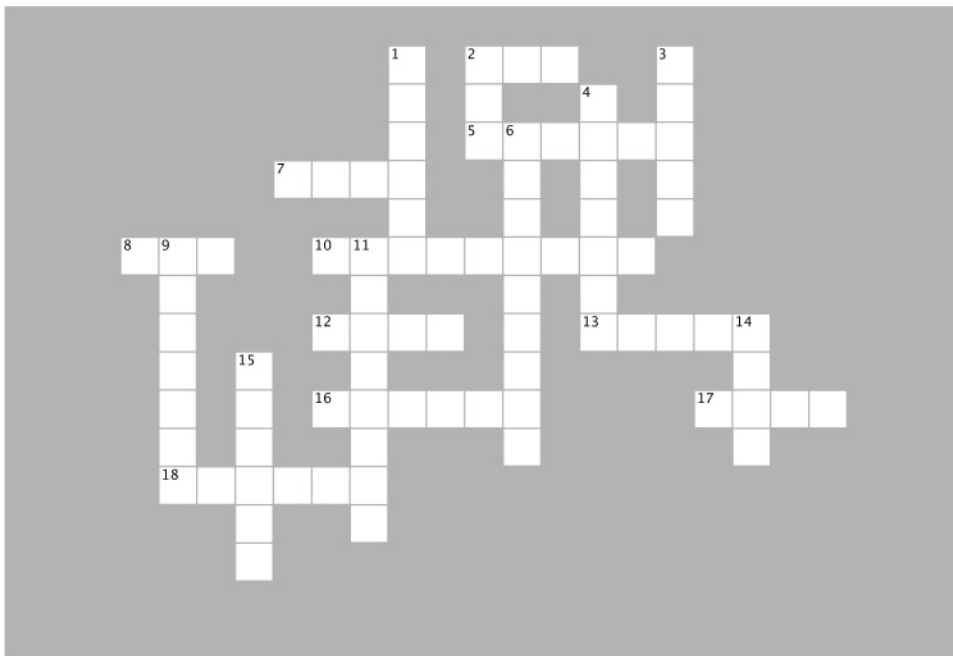
要点

- 在JavaScript中，类型分两组：**基本类型**和**对象**。不属于基本类型的值都是**对象**。
- 基本类型包括数字、字符串、布尔值、`null`和`undefined`；其他的值都是对象。
- `undefined`意味着变量（属性或数组元素）还未初始化。
- `null`表示“无对象”。
- `NaN`指的是“非数字”，但更准确地说，`NaN`指的是在JavaScript中无法表示的数字。`NaN`的类型为数字。
- `NaN`与包括它自己在内的任何值都不相等，因此要检查一个值是否是`NaN`，应使用函数`isNaN`。
- 要检查两个值是否相等，可使用`==`或`===`。
- 如果两个操作数的类型不同，相等运算符（`==`）尝试将一个操作数转换为另一个操作数的类型，再检查它们是否相等。
- 如果两个操作数的类型不同，严格相等运算符（`===`）将返回`false`。
- 要避免进行类型转换，可使用`===`，但在有些情况下，自动执行类型转换的`==`可提供极大的便利。
- 其他运算符也会自动执行类型转换，如算术运算符和字符串拼接运算符。
- 在JavaScript中总共有5个**假值**：`undefined`、`null`、`0`、“ ”（空字符串）和`false`；其他值都是**真值**。
- 在有些情况下，字符串的行为像对象。你使用基本类型字符串的属性或方法时，JavaScript将把它暂时转换为对象，使用指定的属性或方法，再将其转换为基本类型字符串。这是在幕后自动进行的，不用你操心。
- 字符串有很多执行字符串操作的方法。
- 仅当两个变量包含的对象引用指向同一个对象时，它们才相等。



JavaScript填字游戏

你在本章学到了大量的JavaScript技能，请完成下面的填字游戏，以充分理解这些知识。答案都可在本章找到。



横向

2. JavaScript中唯一一个与什么都不相等的值。
5. Infinity的类型。
7. JavaScript中有多少个假值。
8. 谁得到了Aeron座椅？
10. 仅当两个包含对象引用的变量_____同一个对象时，它们才相等。
12. 要求返回一个对象，但没有所需的对象时返回的值。
13. 返回一个数组的字符串方法。
16. 密苏里州的一个城市，温度始终保持在19°C。
17. JavaScript规范中null的类型。
18. 一个相等运算符，仅当两个操作数的类型和值都相同时才返回true。

纵向

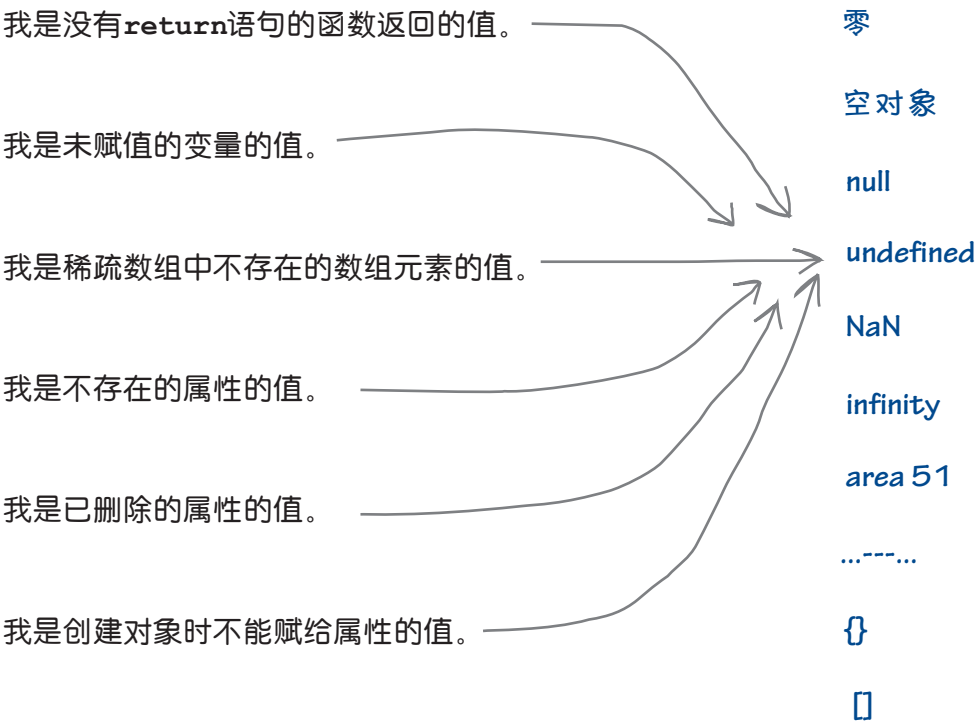
1. 一个用来确定值的类型的运算符。
2. 世上最怪异的值。
3. 你的菲亚特停在哪个停车场？
4. 字符串有时伪装成什么？
6. 不存在的属性的值。
9. 字符串有很多可用于操作字符串的什么？
11. 将两个运算符转换为相同的类型，再比较它们是否相等的运算符。
14. null == undefined的结果。
15. 用来获取字符串中特定索引处字符的方法。



一大拨JavaScript值乔装打扮，正在玩一个“猜猜我是谁”的晚会游戏，其中还混入了一些不速之客。你需要根据它们对自己的描述猜测它们的身份——假设它们说的都是真话。请用箭头将每条描述连接到相应的参与者，我们已经猜出一个了。

答案如下。

参与游戏的值：



实验

答案

在实验室，我们喜欢将东西拆开，看看里面的情况，接上诊断工具以搞清楚到底出了什么问题。今天，我们要研究的是JavaScript的类型系统，而我们已经找到了一个小小的诊断工具`typeof`，可用于检查变量。请你穿上实验服，戴上护目镜，进入实验室，与我们一道来做实验吧。

`typeof`是一个内置的JavaScript运算符，可用于探测其操作数（要对其进行操作的东西）的类型，如下例所示：



```
var subject = "Just a string";
var probe = typeof subject;
console.log(probe);
```

运算符`typeof`接受一个操作数，并返回该操作数的类型。

这里的类型是`string`。注意到`typeof`使用字符串来表示类型，如“`string`”“`boolean`”“`number`”“`object`”“`undefined`”等。



该你了。请收集下述实验的结果。

```
var test1 = "abcdef";
var test2 = 123;
var test3 = true;
var test4 = {};
var test5 = [];
var test6;
var test7 = {"abcdef": 123};
var test8 = ["abcdef", 123];
function test9(){return "abcdef"};

console.log(typeof test1);
console.log(typeof test2);
console.log(typeof test3);
console.log(typeof test4);
console.log(typeof test5);
console.log(typeof test6);
console.log(typeof test7);
console.log(typeof test8);
console.log(typeof test9);
```

这些是实验数据，而这些是实验。



回到实验室 答案



哎呀，刚才的实验数据遗漏了null。下面是遗漏的实验：

```
var test10 = null;
```

```
console.log(typeof test10);
```

这是我们得到的结果。

JavaScript控制台

object

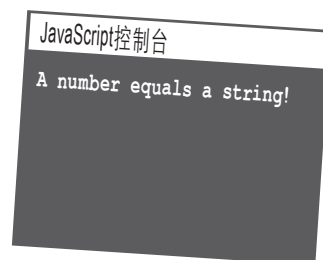


练习答案

本章一直在讨论有趣的值，下面来看看一些有趣的行为。请将下面的代码添加到一个基本网页的<script>元素中，再加载该网页，看看控制台显示的是什么。为什么会出现这种情况呢？你能猜出其中的原因吗？

```
if (99 == "99") {  
    console.log("A number equals a string!");  
} else {  
    console.log("No way a number equals a string");  
}
```

这是我们得到的输出。





对于下面的比较表达式，在运算符==和===下方分别写出它们的结果。

	==	===	
"42" == 42	<u>true</u>	<u>false</u>	"42" === 42
"0" == 0	<u>true</u>	<u>false</u>	"0" === 0
"0" == false	<u>true</u>	<u>false</u>	"0" === false
"true" == true	<u>false</u>	<u>false</u>	"true" === true
true == (1 == "1")	<u>true</u>	<u>false</u>	true === (1 === "1")

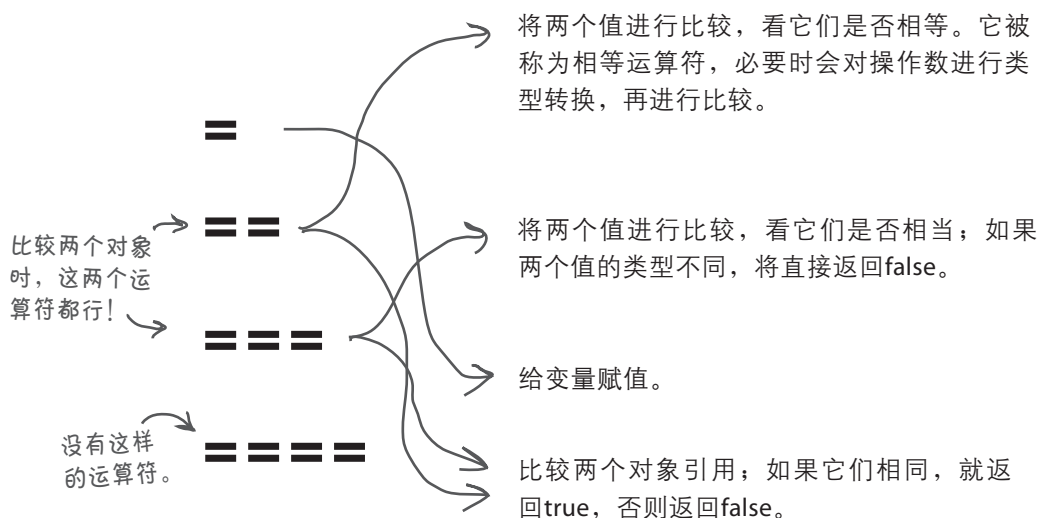
难度极高!

将两个==都替换为===后，结果为false。

连连看

答案

下面是对一些运算符的描述，你能将它们与相应的运算符连起来吗？请注意，每个运算符可能对应于0、1或多项描述。答案如下。





对于下面的表达式，在旁边的空白处写出其结果。我们已经完成了一个。答案如下。

`Infinity - "1"` Infinity

“1”被转换为1，而
Infinity - 1的结果为
Infinity。

`"42" + 42` "4242"

`2 + "1 1"` "21 1"

`99 + 101` 200

`"1" - "1"` 0

两个字符串都被转
换为1，而1 - 1的
结果为0。

`console.log("Result: " + 10/2)` "Result: 5"

先计算10/2，再将结
果转换为字符串，并
与字符串“Result:”
拼接。

`3 + " bananas " + 2 + " apples"` "3 bananas 2 apples"

每个+都被视为字符串拼接运算符，
因为至少有一个操作数为字符串。



该来用用测谎仪了。判断哪些值是真值，哪些值是假值，找出犯罪嫌疑人撒了多少次谎，进而判断指控的罪名是否成立。答案如下。你在浏览器中尝试了这些测试吗？

```
function lieDetectorTest() {
```

```
    var lies = 0;
```

```
    var stolenDiamond = { };
```

```
    if (stolenDiamond) {
```

```
        console.log("You stole the diamond");
```

```
        lies++;
```

```
    }
```

```
    var car = {
```

```
        keysInPocket: null
```

```
    };
```

```
    if (car.keysInPocket) {
```

```
        console.log("Uh oh, guess you stole the car!");
```

```
        lies++;
```

```
    }
```

```
    if (car.emptyGasTank) {
```

```
        console.log("You drove the car after you stole it!");
```

```
        lies++;
```

```
    }
```

```
    var foundYouAtTheCrimeScene = [ ];
```

```
    if (foundYouAtTheCrimeScene) {
```

```
        console.log("A sure sign of guilt");
```

```
        lies++;
```

```
    }
```

```
    if (foundYouAtTheCrimeScene[0]) {
```

```
        console.log("Caught with a stolen item!");
```

```
        lies++;
```

```
    }
```

```
    var yourName = " ";
```

```
    if (yourName) {
```

```
        console.log("Guess you lied about your name");
```

```
        lies++;
```

```
    }
```

```
    return lies;
```

```
}
```

```
var numberOfLies = lieDetectorTest();
```

```
console.log("You told " + numberOfLies + " lies!");
```

```
if (numberOfLies >= 3) {
```

```
    console.log("Guilty as charged");
```

```
}
```

对象是真值，哪怕它是空对象。

犯罪嫌疑人没偷车，因为属性 keysInPocket 的值为假值 null。

犯罪嫌疑人也没有将车开走，因为属性 emptyGasTank 的值为假值 undefined。

[] (空数组) 为真值，因此犯罪嫌疑人被当场抓住。

这个数组不包含任何元素，因此索引 0 处的数组元素为假值 undefined。犯罪嫌疑人肯定将赃物藏起来了。

一个只包含一个空格的字符串。

任何非空字符串都是真值，哪怕它只包含一个空格！

犯罪嫌疑人撒了3次谎，因此我们认为他有罪。



JavaScript控制台

You stole the diamond

A sure sign of guilt

Guess you lied about your name

You told 3 lies!

Guilty as charged



下面的代码用于在Earl停车场寻找汽车，请据此写出loc1到loc4的值。

```
function findCarInLot(car) {
    for (var i = 0; i < lot.length; i++) {
        if (car === lot[i]) {
            return i;
        }
    }
    return -1;
}

var chevy = {
    make: "Chevy",
    model: "Bel Air"
};

var taxi = {
    make: "Webville Motors",
    model: "Taxi"
};

var fiat1 = {
    make: "Fiat",
    model: "500"
};

var fiat2 = {
    make: "Fiat",
    model: "500"
};
```

```
var lot = [chevy, taxi, fiat1, fiat2];
```

这是我们的答案。

```
var loc1 = findCarInLot(fiat2);
var loc2 = findCarInLot(taxi);
var loc3 = findCarInLot(chevy);
var loc4 = findCarInLot(fiat1);
```

3
1
0
2



继续实验答案

实验人员继续使用运算符`typeof`来研究JavaScript，发现了这门语言中另外一些有趣的地方。在实验过程中，他们发现了另一个运算符`instanceof`。有了这个运算符，他们就能跻身最前沿的行列。请穿上实验服，戴上护目镜，看看你能否帮忙破解下面的JavaScript代码及其结果。请注意，这些绝对是本书到目前为止最怪异的代码。



代码如下。请阅读、运行、修改它们，看看它们是做什么的。

```
function Duck(sound) {
  this.sound = sound;
  this.quack = function() {console.log(this.sound);}
}
```

好怪异。有点像函数，又有点像对象。

```
var toy = new Duck("quack quack");
```

哇，new，以前没见过。估计是新建一个Duck，并将其赋给变量toy。

```
toy.quack();
```

如果看起来像对象，走起来像对象，那它就是对象。下面来验证这一点。

```
console.log(typeof toy);
console.log(toy instanceof Duck);
```

这是运算符instanceof。

这些代码到底是什么意思呢？两章后将全面介绍。你已经可能没有意识到，你在通往高级JavaScript程序员之路上走了很远。我可没开玩笑！

JavaScript控制台

quack quack ← toy就像是对象……我们可以调用它的方法。

object ← toy的类型为对象。

true ← 总之，toy是一个Duck实例。

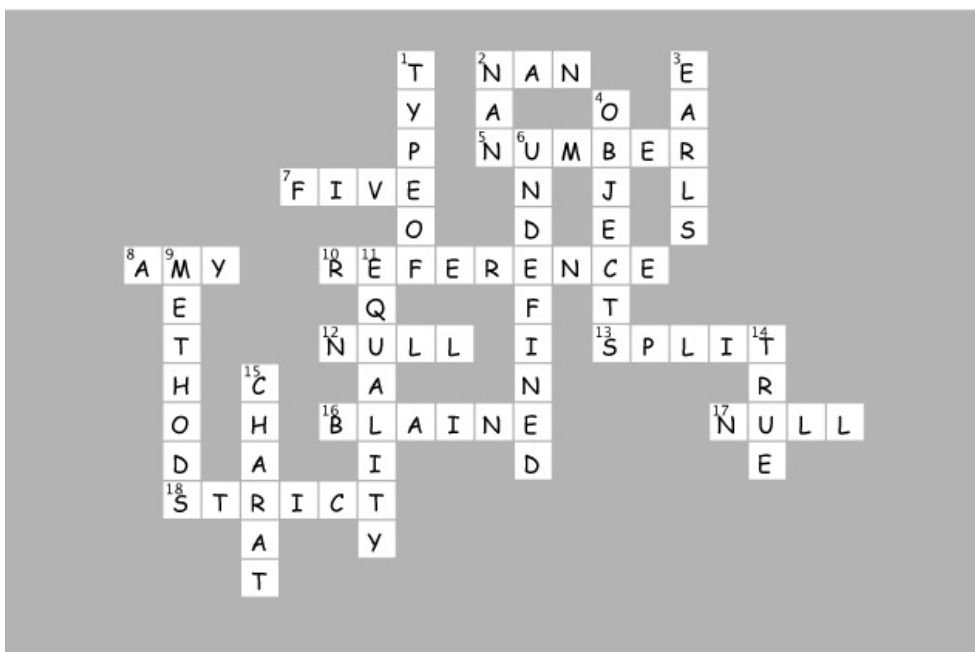
↑ 这是我们得到的结果。





JavaScript填字游戏答案

你在本章学到了大量的JavaScript技能，请完成下面的填字游戏，以充分理解这些知识。答案都可在本章找到。答案如下。



8 综合应用

编写一个应用程序

你真厉害，将各种食材
搅拌在一起，就做出了
美味的食物。



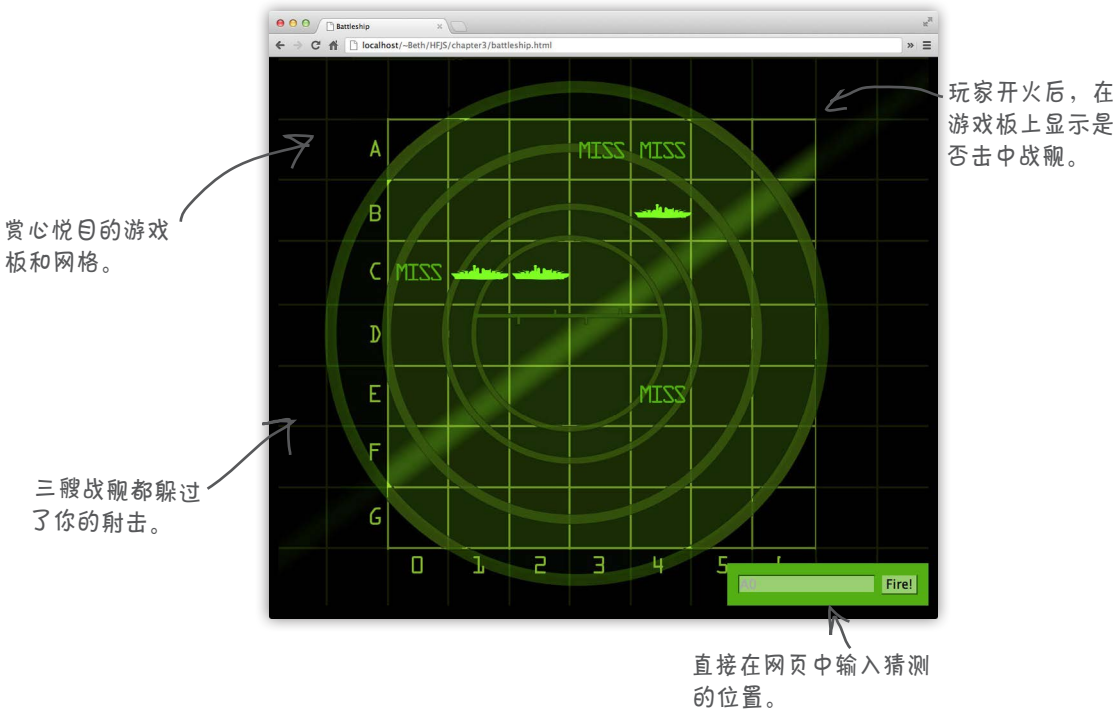
嘿！睁开眼吧！奶
油蛋糕来了。

系上工具腰带，就是那个囊括你新学的所有编码技能、DOM知识甚至HTML和CSS知识的工具腰带。本章将结合使用这些知识打造第一个货真价实的Web应用。不再是只有一艘战舰和一行藏身之地的小儿科游戏，本章将提供全面的体验：又大又漂亮的游戏板、多艘战舰、获取用户输入……这些都包含在一个网页中。我们将使用HTML定义游戏网页的结构，使用CSS设置游戏的视觉样式，并使用JavaScript代码定义游戏的行为。坐好了，我们将全力以赴，将油门踩到底，编写一些正式的代码。

编写一个货真价实的战舰游戏

在第2章，你从零开始，编写了一款不错的战舰游戏。你完全可以因此感觉良好，但必须承认，这款游戏有点小儿科：它能够正常运行，玩起来也凑合，但根本无法给朋友留下深刻的影响；你也无法凭它来融得第一轮风险投资。要给人留下深刻影响，需要有看得见的游戏板、华丽的战舰，并让玩家能够直接在游戏中输入（而不是使用全能的浏览器对话框）。你还需改进前一个版本，使其支持三艘战舰。

换句话说，你希望游戏界面类似于下面这样。



请暂时忘记JavaScript，看看上面的战舰游戏模型。你将如何使用HTML和CSS创建这个网页的结构和视觉效果？

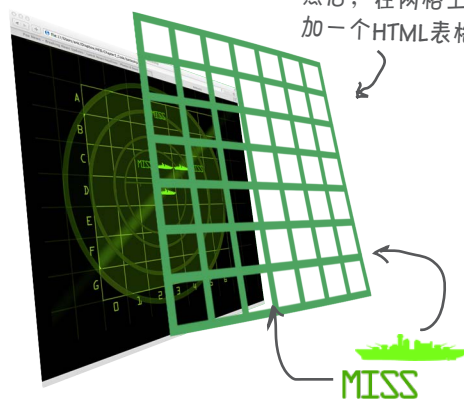
回顾HTML和CSS

要创建时髦的交互式网页（应用），需要使用三种技术：HTML、CSS和JavaScript。你知道，HTML用于定义结构，CSS用于设置样式，而JavaScript用于定义行为。不再纸上谈兵，本章要玩真的了。咱们先从HTML和CSS着手。

我们的第一个目标是创建前一页所示的游戏板。不仅如此，我们还需正确地设计其结构，以便能够在JavaScript中获取玩家输入以及显示击中、未击中等信息。

为此，我们将把网页的背景设置为一幅图像，以显示网格和雷达扫描图案。然后在背景图像上放置一个HTML表格，以便显示战舰等内容。我们还将使用一个HTML表单来获取玩家输入。

我们将网页背景设置为一幅描绘游戏网格的图像。



然后，在网格上面添加一个HTML表格。

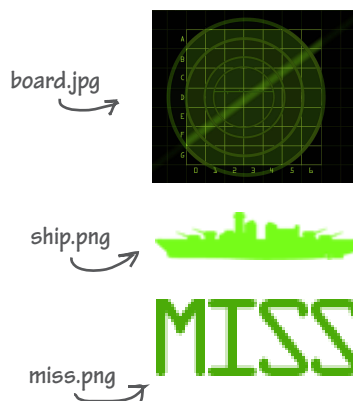
这样就可以在需要时将战舰或MISS图像放到单元格中。

下面就来开发这个游戏。我们将先退一步，花几页的篇幅编写重要的HTML和CSS；完成这些工作后，就可以开始编写JavaScript代码了。

获取战舰游戏工具包

下面是让你能够开始编写新版本战舰游戏的工具包。

工具包括



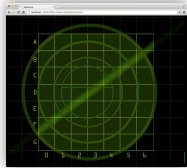
这个工具包中有三幅图像：board.jpg、ship.png和miss.png。board.jpg是主背景图像，用于显示带网格的游戏板。ship.png是一幅小战舰图像，将被放置到游戏板上；注意到它是一幅部分透明的PNG图像，这样可直接将其放在背景图像上面。最后，图像miss.png也将被直接放在游戏板上。在这个游戏中，玩家击中战舰时，我们将在相应的位置显示战舰图像，而未击中时，就显示MISS图像。

可从<http://wickedlysmart.com/hfjs>下载这个游戏需要的所有素材。

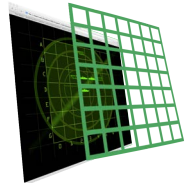
创建HTML页面：大致轮廓

下面是创建HTML页面的进攻计划。

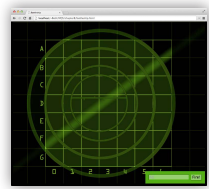
- 1 首先，我们将专注于游戏的背景，包括将背景设置为黑色，并在网页中添加雷达网格图像。
- 2 接下来，我们创建一个HTML表格，并将其放在背景图像上面。这个表格中的每个单元格都对应于游戏板的一个格子。
- 3 然后，我们将添加一个HTML表元素，让玩家能够输入其猜测，如A4。我们还将添加一个消息区域，用于显示诸如You sank my battleship!等消息。
- 4 最后，我们将决定如何使用表格在游戏板上显示战舰图像（击中时）和MISS图像（未击中时）。



我们在背景中放置一幅图像，让游戏界面拥有正在进行雷达扫描的效果。



背景上面是一个HTML表格，它创建的游戏板指定了战舰所处的范围。



供玩家输入的HTML表单。



我们将在需要时将
这些图像放置到表格中。

第1步：基本HTML

咱们开始吧！首先需要有一个HTML页面。我们将创建一个简单的HTML5页面，并使用样式指定背景图像。在这个网页中，我们将添加一个<body>元素，其中只包含一个<div>元素。这个<div>元素将用于放置游戏网格。

请看下一页，其中的网页包含基本的HTML和CSS。



觉得知识有点不够用？

如果你觉得自己的HTML和CSS知识有点不够用，请参阅与本书配套的《Head First HTML与CSS》。

`<!doctype html>` 一个普通的HTML页面。

`<html lang="en">`

`<head>`

`<meta charset="utf-8">`

`<title>Battleship</title>`

`<style>`

`body {`

`background-color: black;`

`}`

`div#board {`

`position: relative;`

`width: 1024px;`

`height: 863px;`

`margin: auto;`

`background: url("board.jpg") no-repeat;`

`}`

`</style>`

`</head>`

`<body>`

`<div id="board">`

`</div>`

`<script src="battleship.js"></script>`

`</body>`

`</html>`

我们要将网页的背景设置为黑色。

我们要让游戏板位于网页中央，因此将width（游戏板的宽度）设置为1024像素，并将margin（外边距）设置为auto。

将<div>元素“board”的背景设置为图像board.jpg，从而将该图像添加到网页中。我们对这个<div>元素进行相对定位，以便相对于这个<div>对下一步添加的表格进行定位。

我们将在这个<div>元素中添加表示游戏板的网格以及用于获取玩家输入的表单。

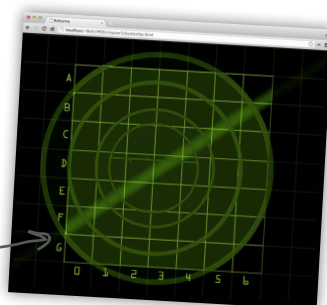
我们将把代码放在文件battleship.js中，因此要先为此创建一个空白文件。



试驾

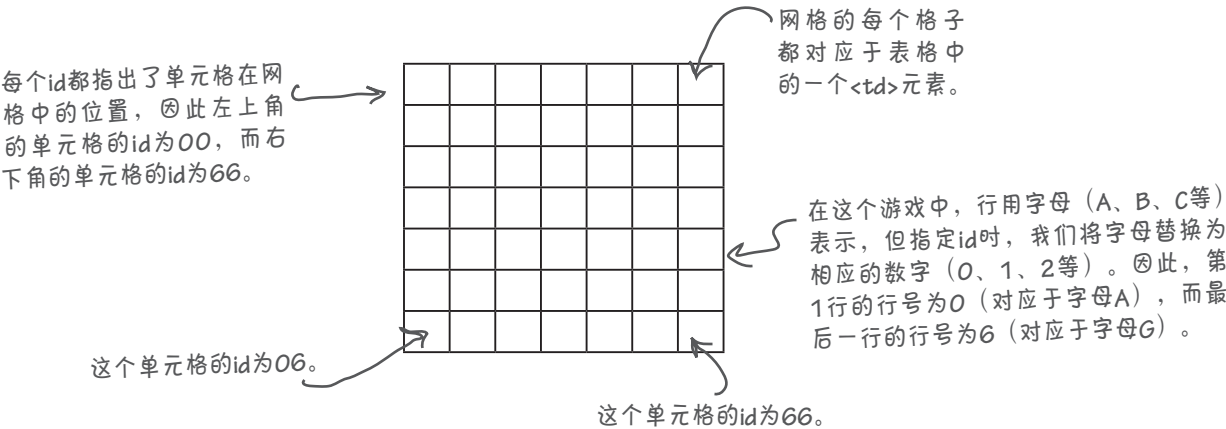
在文件中输入上述代码，并将其保存为battleship.html（也可从本书配套网站<http://wickedlysmart.com/hfjs>下载所有的代码），再在浏览器中加载这个网页。下面是我们的测试情况。

当前，这个网页类似于这样。



第2步：创建表格

接下来需要创建表格。这个表格覆盖背景图像中的网格，并提供了显示击中和未击中图像的区域。每个单元格（即每个<td>元素）都正好在背景图像中的一个格子上面。这里的诀窍是，给每个单元格都指定一个id，以便能够在CSS和JavaScript中操作它。来看看如何创建这些id并添加定义表格的HTML。



下面是定义这个表格的HTML，请将它们添加到<div>标签之间：

```
<div id="board">  ← 我们将表格嵌套在<div>元素board内。

  <table>
    <tr>
      <td id="00"></td><td id="01"></td><td id="02"></td><td id="03">
</td><td id="04"></td> <td id="05"></td><td id="06"></td>
    </tr>
    <tr>
      <td id="10"></td><td id="11"></td><td id="12"></td><td id="13"></td>
<td id="14"></td> <td id="15"></td><td id="16"></td>
    </tr>
    ...
    <tr>
      <td id="60"></td><td id="61"></td><td id="62"></td><td id="63"></td>
<td id="64"></td><td id="65"></td><td id="66"></td>
    </tr>
  </table>
</div>
```

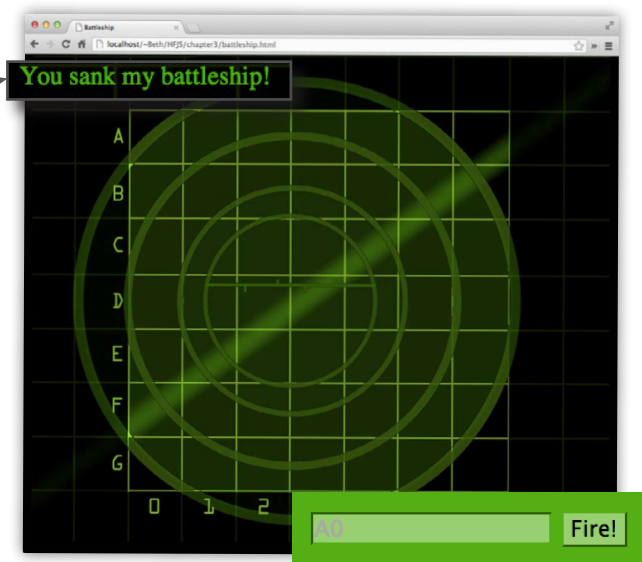
确保每个<td>元素的id都是正确的，对应于在网格中的行和列。

为节省篇幅，省略了几行，但我们深信你能补全它们。

第3步：与玩家交互

现在需要一个让玩家能够输入猜测位置（如A0或E4）的HTML元素，还需要一个用于向玩家显示消息（如You sank my battleship!）的元素。我们将使用一个<form>和一个<input>元素来让玩家能够输入猜测位置，并使用一个<div>元素来提供向玩家显示消息的区域。

玩家击沉战舰后，我们将在左上角显示一条消息，告知玩家这一点。



玩家可在这里输入其猜测的位置。

```
<div id="board">
```

```
  <div id="messageArea"></div>
```

```
  <table>
```

```
    ...
```

```
</table>
```

```
  <form>
```

```
    <input type="text" id="guessInput" placeholder="A0">
```

```
    <input type="button" id="fireButton" value="Fire!">
```

```
</form>
```

```
</div>
```

请注意，<div>元素messageArea、<table>元素和<form>元素都嵌套在<div>元素board中。这对下一页的CSS来说很重要。

在代码中，将使用<div>元素messageArea来显示消息。

这个<form>元素包含两个<input>元素，其中一个用于输入猜测的位置（类型为text），另一个是一个按钮。注意到给这些元素指定了id，后面编写代码来获取玩家的猜测时将用到它们。

再添加一些样式

如果你现在加载这个网页（去试试），大多数元素的位置和尺寸都不对。因此，我们需要编写一些CSS，让所有的元素都处于正确的位置，并确保所有元素（如单元格）的尺寸都与游戏板图像匹配。

为让每个元素都处于正确的位置，我们将使用CSS来给它们定位。我们已经以相对方式定位了<div>元素board，因此可以指定消息区域、表格和表单在<div>元素board内的具体位置，让它们出现在我们希望的地方。

首先来指定<div>元素messageArea的位置。它嵌套在<div>元素board内，我们希望它位于游戏板的左上角：

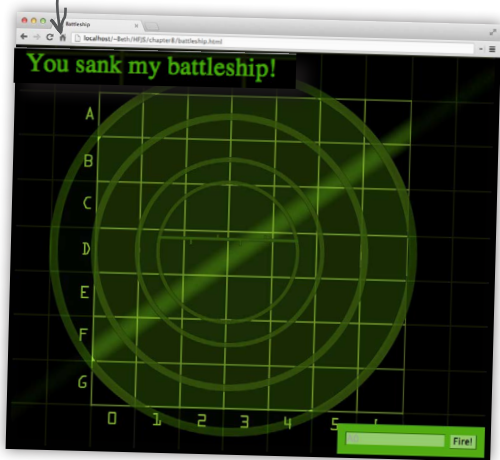
```
body {
    background-color: black;
}
div#board {
    position: relative;
    width: 1024px;
    height: 863px;
    margin: auto;
    background: url("board.jpg") no-repeat;
}
div#messageArea {
    position: absolute;
    top: 0px;
    left: 0px;
    color: rgb(83, 175, 19);
}
```

<div>元素board的定位方式是相对的，因此能以相对于该<div>元素的方式定位它嵌套的所有元素。

我们将消息区域定位到游戏板的左上角。

<div>元素messageArea嵌套在<div>元素board中，因此以相对于<div>元素board的方式指定其位置。它将位于这样的位置：离<div>元素board的上边缘和左边缘都为0px。

我们想把消息区域放在游戏板的左上角。



要点

- position: relative按正常的网页排版方式定位元素。
- position: absolute基于父元素位置定位元素。
- 可使用属性top和left来指定元素相对于默认位置偏离多少像素。

我们也可以使用绝对定位方式指定表格和表单在<div>元素board中的位置，从而将这些元素准确地放置到所需的位置。下面列出了所需的CSS：

```
body {
    background-color: black;
}
div#board {
    position: relative;
    width: 1024px;
    height: 863px;
    margin: auto;
    background: url("board.jpg") no-repeat;
}
div#messageArea {
    position: absolute;
    top: 0px;
    left: 0px;
    color: rgb(83, 175, 19);
}
table {
    position: absolute;
    left: 173px;
    top: 98px;
    border-spacing: 0px;
}
td {
    width: 94px;
    height: 94px;
}
form {
    position: absolute;
    bottom: 0px;
    right: 0px;
    padding: 15px;
    background-color: rgb(83, 175, 19);
}
form input {
    background-color: rgb(152, 207, 113);
    border-color: rgb(83, 175, 19);
    font-size: 1em;
}
```

让<table>元素离游戏板的左边缘和上边缘分别为173像素和98像素，从而确保它与背景图像中的网格对齐。

将每个<td>元素的宽度和高度都指定为特定的值，让表格单元格与网格单元格对齐。

将<form>元素放在游戏板的右下角。这将遮住右下角数字的一小部分，但玩家依然知道这些数字是什么，因此没有关系。我们还将<form>元素设置为漂亮的绿色，使其匹配背景图像。

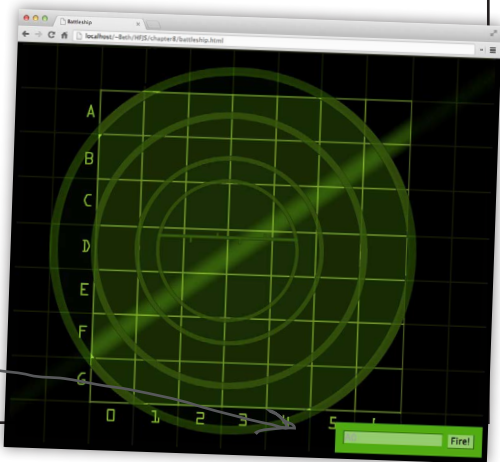
最后，稍微设置一下两个<input>元素的样式，使其与游戏的主题匹配。就这些！



再来测试一下这款游戏。将所有的HTML和CSS代码都输入到你的HTML文件中，再在浏览器中加载这个页面，你应看到如下界面。

表格位于网格的上面，但你看不到它（因为它是不可见的）。

你可以在表单中输入猜测了，但在编写JavaScript代码前，游戏不会有任何反应。



第4步：指出是否击中战舰

游戏板非常漂亮，但还需在玩家每次猜测后指出是否击中了战舰，即在游戏板的相应位置显示图像ship.png或miss.png。就目前而言，我们只考虑如何编写所需的标记或样式，后面将在代码中使用同样的技巧。

那么，如何在游戏板中显示图像ship.png或miss.png呢？一种简单的方法是，使用CSS将<td>元素的背景设置为合适的图像。为此，我们将创建hit和miss类，并在这两个类中将CSS属性background设置为相应的图像。这样，使用hit类设置元素的样式时，其背景将为图像ship.png，而使用miss类设置元素的样式时，其背景将为图像miss.png。这两个类类似于下面这样：



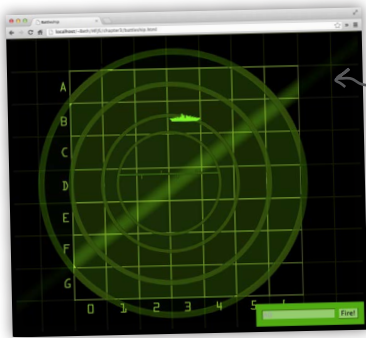
```
.hit {  
    background: url("ship.png") no-repeat center center;  
}  
.miss {  
    background: url("miss.png") no-repeat center center;  
}
```

如果元素归属于hit类，其背景将为图像ship.png；如果元素归属于miss类，其背景将为图像miss.png。

这两条CSS规则都在元素内居中放置一幅图像。

使用hit和miss类

请务必将hit和miss类的定义添加到你的CSS中。你可能纳闷，如何使用这些类呢？下面通过一个小实验来演示这一点：假设有艘战舰隐藏在单元格B3、B4和B5处，而玩家猜测的位置为B3，这将击中战舰！因此你需要在B3处显示图像ship.png。为此，你首先将B转换为数字1（因为A对应于0，B对应于1，依此类推），进而找到表格中id为13的<td>元素。接下来，将该<td>元素的class属性设置为hit，如下所示：



将<td>元素的属性class设置为hit。

```
<tr>
<td id="10"></td> <td id="11"></td> <td id="12"></td> <td id="13" class="hit"></td>
<td id="14"></td> <td id="15"></td> <td id="16"></td>
</tr>
```

务必在你的CSS中添加前一页的hit和miss类。

现在重新加载网页时，你将在游戏板的B3格子内看到战舰图像。

将id为13的<td>元素的class属性设置为hit后看到的结果。



实弹演习

在编写代码在游戏板上指出是否击中了战舰前，再来做一些练习，以明白CSS的工作原理。假设玩家作出如下猜测，请你在标记中手动添加hit或miss类。请务必与本章末尾的答案进行对比！

- 战舰 1: A6, B6, C6
- 战舰 2: C4, D4, E4
- 战舰 3: B0, B1, B2

玩家的猜测如下：

A0, D4, F5, B2, C5, C6

请查看本章末尾的答案，再接着往下阅读。

别忘了，需要将字母转换为数字（A对应于0……G对应于6）。

完成这个练习后，将你给<td>元素添加的class属性删除，确保编写代码时游戏板是空的。

世上没有愚蠢的问题

问：在我们的表格中，将id特性设置成了数字，这行吗？

答：没问题。在HTML5中，可将数字用作元素id。只要id值不包含空格就没问题。就这个战舰游戏而言，将每个id都设置为数字非常合适。这可以跟踪表格中的每个位置，让我们能够快速而轻松地访问相应的元素。

问：对于游戏板中的每个单元格，我们都使用一个

答：没错。界面由几部分组成：为吸引眼球，使用了一个显示网格的背景图像；在背景图像上面，放置了一个透明的HTML网格，并在需要时使用hit或miss类将单元格的背景设置为一幅图像。单元格背景设置都是在代码中完成的：动态地设置元素的class特性。

问：我们好像需要将字母转换为数字，例如将A6转换为06。JavaScript会自动替我们完成这项工作吗？

答：不会，我们必须自己做，但有一种简单的方式：利用你的数组知识快速进行转换。敬请期待。

问：我不确定自己完全明白了CSS定位的工作原理。

答：定位让你能够指定元素的准确位置。以相对方式定位时，将基于网页的正常排版方式来定位元素；以绝对方式定位时，元素将在最近的已定位祖先元素内处于特定的位置。有时候，要对整个页面进行绝对定位。在这种情况下，可指定它相对于Web浏览器左上角的位置。在这个战舰游戏中，我们以绝对方式定位表格和消息区域，但参考点为游戏板的左上角，因为对表格和消息区域来说，游戏板是最近的已定位祖先元素。

如果你要更深入地复习CSS定位，请参阅《Head First HTML与CSS》的第11章。

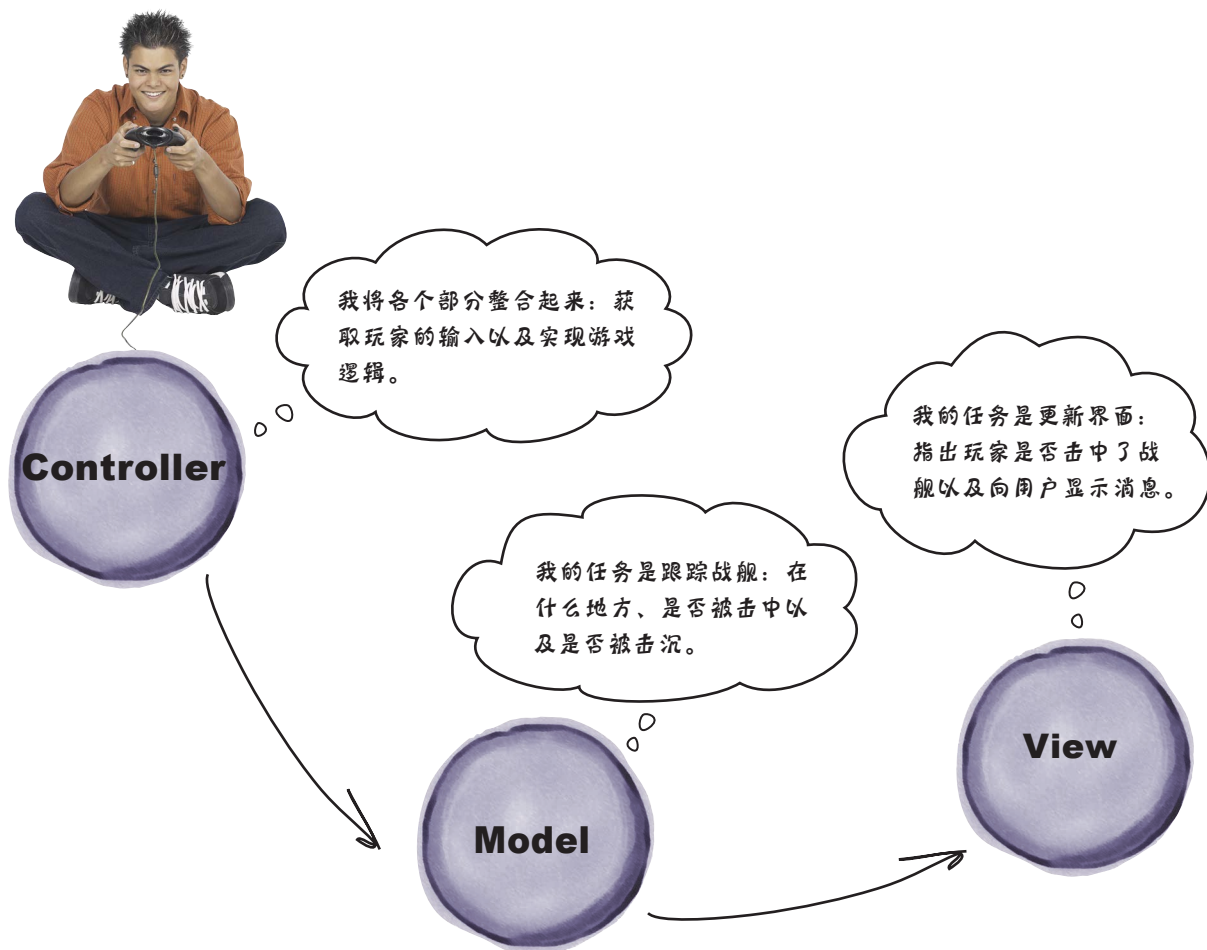
问：学习HTML元素<form>时，我得知它有一个提交表单的action特性。我们为何没有使用它？

答：我们不需要设置<form>的action特性，因为不需要将表单提交给服务器端应用程序。在这个游戏中，我们将使用代码在浏览器中完成所有处理。因此，我们不提交表单，而为表单按钮实现一个单击事件处理程序，其中的代码将处理一切，包括从表单获取用户输入。请注意，需要将表单数据提交给服务器上运行的PHP程序或其他程序时，表单按钮的类型为submit，但我们的表单按钮的类型为button。这个问题问得好，本章后面将更深入地讨论。

如何设计这款游戏

编写好HTML和CSS后，来看看如何设计这款游戏。第2章编写战舰游戏的第一个版本时，还未学习函数、对象和封装，也未学习面向对象设计，因为我们采用了过程型设计，即将游戏设计为一系列包含决策逻辑和迭代的步骤。那时你也没有学习DOM，因此第一个版本的交互性不太强。这次，我们将把这款游戏设计为一系列各司其职的对象，并利用DOM来与用户交互。你将发现，这种设计让问题理解起来容易得多。

首先来介绍一下要设计并实现的对象，总共有3个：model、view和controller。model将存储游戏的状态，如每艘战舰的位置以及哪个部位被击中；view负责更新界面；controller将各个部分整合起来，用于处理用户输入、实现游戏逻辑并判断游戏是否结束。



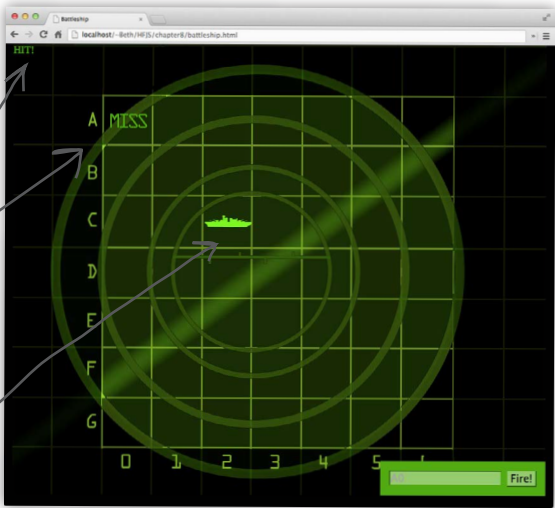


来做一些对象设计工作吧。我们将从view对象着手。前面说过，view对象负责更新视图。根据下面的视图，你能确定view对象需要实现哪些方法吗？请在下面写出这些方法的声明（这里只考虑声明，方法体的代码稍后再考虑），并使用一两条注释指出每个方法的功能。我们为你编写了一个方法的声明。继续往下阅读前，务必查看本章末尾的答案。

这是一条消息。消息为“HIT!” “You missed.” “You sank my battleship!”等。

在网格中显示MISS图像。

在网格中显示战舰图像。



`var view = {` 定义一个对象，并将其赋给变量view。

```
// 这个方法将一个字符串作为参数，并在消息区域中显示它
displayMessage: function(msg) {
    // 稍后将编写的代码
}
```

在这里声明你认为需要提供的方法！

`};`

实现view对象

如果你没有这样做，我真替你脸红。现在就去看看！

如果你查看了前一个练习的答案，就知道我们在view对象中定义了三个方法：`displayMessage`、`displayHit`和`displayMiss`。这绝非唯一正确的答案。例如，也可只定义`displayMessage`和`displayPlayerGuess`两个方法，并让`displayPlayerGuess`接受一个指出玩家是否猜对了的参数。这样的设计也无可挑剔。这里继续以我们的设计为例，看看如何实现第一个方法——`displayMessage`：

view对象

```
var view = {
  displayMessage: function(msg) {
    },
  displayHit: function(location) {
    },
  displayMiss: function(location) {
    }
};
```

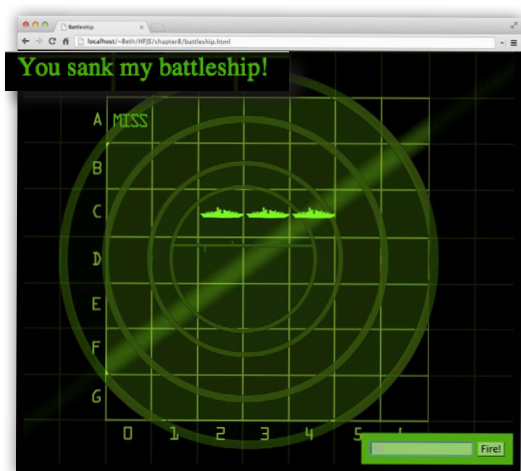
我们将从这里着手。

displayMessage的工作原理

为实现方法`displayMessage`，请再看一眼HTML。你将发现其中有一个用于显示消息的`<div>`元素，其id为`messageArea`：

```
<div id="board">
  <div id="messageArea"></div>
  ...
</div>
```

我们将使用DOM来获取这个`<div>`元素，再使用`innerHTML`来设置其文本。别忘了，每当你修改DOM时，所作的修改将立即在浏览器中反映出来。





等等，在没有获取玩家输入的情况下，怎么可能实现view对象呢？

这是对象的一大优点：可在不考虑程序的其他任何细节的情况下，确保对象履行其职责。在这里，视图只需知道如何更新消息区域以及如何在网格上放置击中和未击中标记。正确地实现了这些行为后，view对象就编写好了，可以接着处理程序的其他部分。

面向对象设计的另一个优点是，可独立地测试视图，确保它能够正确地运行。同时测试程序的多个方面时，出现问题的可能性更大，找出问题的难度也更高（因为要找出问题，必须研究更多的代码）。

要在程序的其他部分还未完成的情况下测试独立的对象，需要编写一些最后将被丢弃的测试代码，但这没有关系。

下面来实现并测试view对象，接着处理程序的其他部分！

实现displayMessage

咱们来编写displayMessage的代码。前面说过，它需要：

- 使用DOM来获取id为messageArea的元素；
- 将这个元素的innerHTML设置为传入的消息。

新建一个文件，将其命名为battleship.js，并在其中添加如下view对象。

```
var view = {
  displayMessage: function(msg) {
    var messageArea = document.getElementById("messageArea");
    messageArea.innerHTML = msg;
  },
  displayHit: function(location) {
  },
  displayMiss: function(location) {
  }
};
```

方法displayMessage接受一个参数——msg。

获取网页中的元素messageArea。

将元素messageArea的innerHTML设置为msg，以更新该元素的文本。

下面接着编写其他两个方法，再对这些代码进行测试。这些方法不太复杂，我们可同时测试整个对象。

displayHit和displayMiss的工作原理

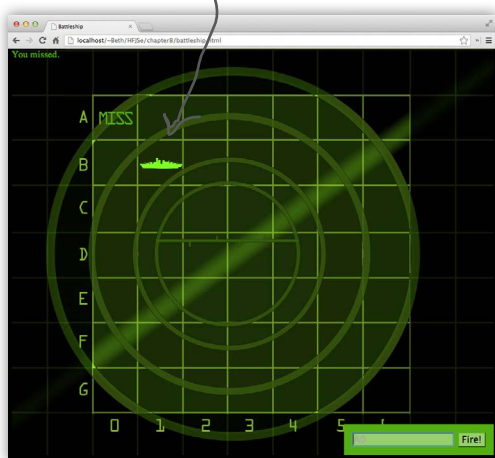
前面说过，要在游戏板上显示图像，需要获取一个<td>元素，并将其class特性设置为hit或miss。将class特性设置为hit时，将在单元格中显示图像ship.png；而设置为miss时，将在单元格中显示图像miss.png。

可将<td>元素的class特性设置为hit或miss来影响其外观，现在需要在代码中这样做。

```
<tr>
<td id="10"></td> <td class="hit" id="11"></td> <td id="12"></td> ...
</tr>
```

在代码中，我们将使用DOM来获取一个<td>元素，再使用element对象的方法setAttribute将class特性设置为hit或miss。设置class特性后，相应的图像将立即出现在浏览器中。我们需要完成的工作如下。

- 获取一个由两个数字组成的字符串id，它指出了要将哪个单元格的class特性设置为hit或miss。
- 使用DOM来获取id为指定值的元素。
- 在displayHit方法中，将该元素的class特性设置为hit；在方法displayMiss中，将该元素的id设置为miss。



实现displayHit和displayMiss

方法displayHit和displayMiss都将一个射击位置作为参数，这个位置必须是表示游戏板HTML表格的单元格（<td>元素）之一的id。因此，我们首先需要做的是使用方法getElementById获取指向这个元素的引用。在方法displayHit中尝试这样做：

```
displayHit: function(location) {
    var cell = document.getElementById(location);
},
```

别忘了，location是根据行号和列号生成的，它是一个<td>元素的id。

下一步是将单元格的class特性设置为hit，为此可像下面这样使用方法setAttribute：

```
displayHit: function(location) {
    var cell = document.getElementById(location);
    cell.setAttribute("class", "hit");
},
```

接下来，我们将这个<td>元素的class特性设置为hit。这将立即在这个元素中显示战舰图像。

现在将这些代码加入到view对象中，并编写方法displayMiss：

```
var view = {
    displayMessage: function(msg) {
        var messageArea = document.getElementById("messageArea");
        messageArea.innerHTML = msg;
    },

    displayHit: function(location) {
        var cell = document.getElementById(location);
        cell.setAttribute("class", "hit");
    },

    displayMiss: function(location) {
        var cell = document.getElementById(location);
        cell.setAttribute("class", "miss");
    }
};
```

使用根据玩家猜测生成的id来获取要更新的元素。

然后将这个元素的class特性设置为hit。

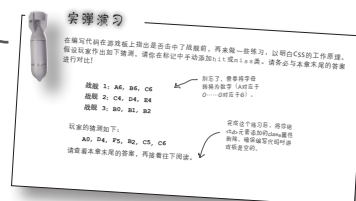
在displayMiss中如法炮制，但将class特性设置为miss，在元素中显示MISS图像。

请将方法displayHit和displayMiss的代码都添加到你的文件battleship.js中。

再次试驾

处理程序的其他部分前，咱们来测试一下这些代码：使用代码实现前面“实弹演习”中的猜测，看看结果如何。我们要实现的猜测如下：

A0, D4, F5, B2, C5, C6
 ↑ ↑ ↑ ↑ ↑ ↑
 未击中 击中 未击中 击中 未击中 击中



为了在代码中表示这些猜测，请在你的JavaScript文件 battleship.js 末尾添加如下代码：

```
view.displayMiss("00");
view.displayHit("34");
view.displayMiss("55");
view.displayHit("12");
view.displayMiss("25");
view.displayHit("26");
```

← "A0"
 ← "D4"
 ← "F5"
 ← "B2"
 ← "C5"
 ← "C6"

别忘了，displayHit和displayMiss将一个游戏板中的位置作为参数。这个参数对应于一个单元格的id，它是这样得到的：将一个字母和一个数字转换为由两个数字组成的字符串。

另外，别忘了测试一下方法displayMessage：

```
view.displayMessage("Tap tap, is this thing on?");
```

这个测试很简单，使用任何消息都可以。

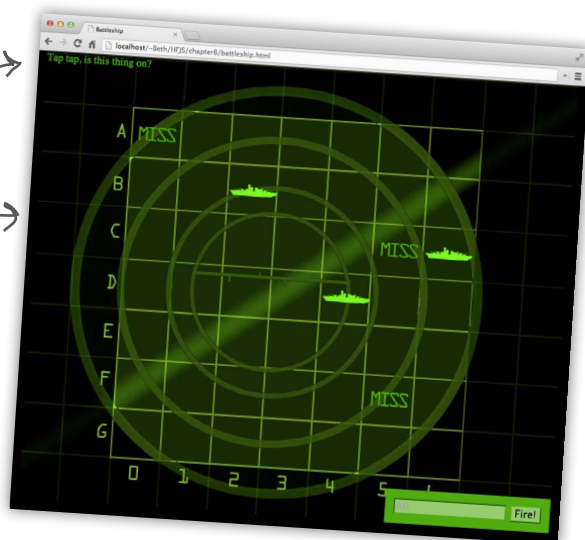
添加这些代码后，在浏览器中重新加载网页，看看视图有何变化。

将代码组织为对象并让每个对象只承担一项职责的好处之一是，可分别对每个对象进行测试，确保它能正确地完成任务。

消息“Tap tap……”显示在视图的左上角。

使用view对象显示的战舰图像和MISS图像位于游戏板中。

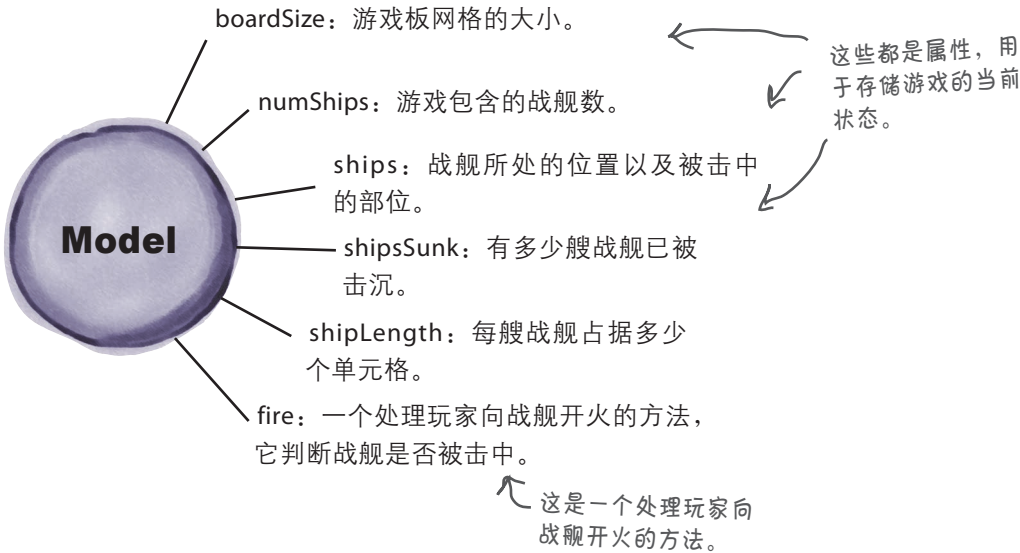
请核实每幅图像都显示在正确的单元格中。



模型

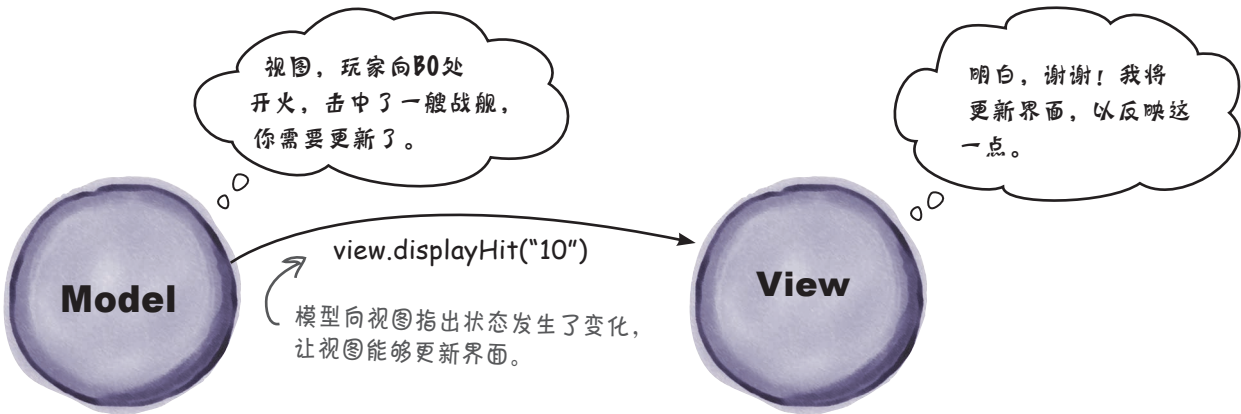
view对象编写好后，接着来处理模型。模型用于存储游戏的状态，通常还包含一些有关如何修改状态的逻辑。在这个游戏中，状态包括战舰的位置、战舰的哪些部位被击中以及有多少艘战舰已被击沉。就目前而言，需要实现的唯一逻辑是：判断玩家是否击中了战舰；如果击中了，将战舰的相应部位标记为被击中。

下面说明了model对象是什么样的。



模型如何与视图交互

游戏的状态发生变化，即玩家击中或未击中战舰时，视图需要更新界面。为此，模型必须与视图交流，好在有多个方法可供模型来完成这项任务。我们将先实现模型中的游戏逻辑，再添加更新视图的代码。



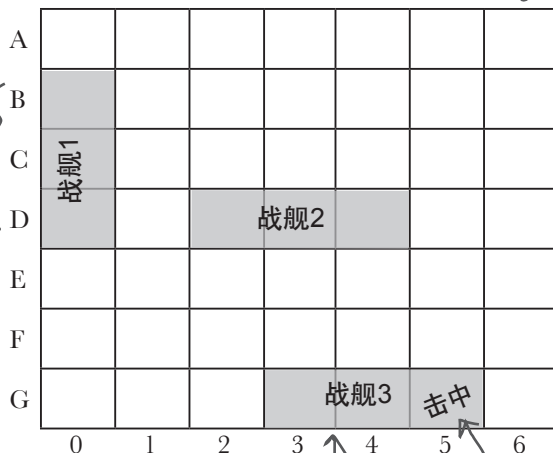
你需要更多战舰和更大的游戏板

编写模型代码前，需要考虑如何在模型中表示战舰的状态。在第2章的简单战舰游戏中只有一艘战舰，它隐藏在一个 1×7 的游戏板中。现在的情况更复杂一些：有3艘战舰，它们隐藏在一个 7×7 的游戏板中，如下所示。

每艘战舰都占据2D游戏板中的3个单元格。

这艘战舰占据单元格B0、C0和D0。

这是第二艘战舰，它占据单元格D2~D4。



这是第三艘战舰，它占据单元格G3~G5。

在现实世界，战舰不可能叠在一起；如果在游戏中出现这种情况，玩家将感到很怪异。后面讨论在游戏板中随机地放置战舰时，将介绍如何避免战舰叠在一起。

我们还需要跟踪被击中的部位。每艘战舰都占据三个单元格，因此对于每艘战舰，都需要存储三个可能被击中的部位。



磨笔上阵

根据前面对新游戏板的描述，你将如何在模型中表示战舰呢？（只考虑它们的位置，击中的部位以后再说。）请从下面选择最佳的解决方案。

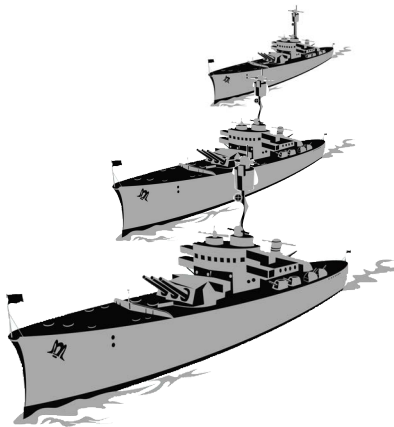
- ☐ 像第2章那样处理战舰，使用9个变量来存储战舰占据的位置。
- ☐ 使用三个数组，每个数组都包含3个元素，指出了一艘战舰占据的位置。
- ☐ 使用一个包含49个元素的数组，每个元素对应于游戏板的一个单元格，其中存储了占据了该单元格的战舰的编号。
- ☐ 使用包含三个位置属性的对象ship，并将所有的战舰都存储在数组ships中。
- ☐ _____
- ☐ _____
- ☐ _____

你也可以在这里列出你的解决方案。

如何表示战舰

前面列出了很多表示战舰的方式，你还可能想出了其他方式。不管是什么样的数据，存储的方式都有很多，你在选择时必须进行权衡：有些方式可节省存储空间，有些可优化运行时间，有些更容易理解，等等。

我们选择了一种非常简单的战舰表示方式：将每艘战舰分别用一个对象表示。这个对象存储了战舰占据的位置以及可能被击中的部位。来看看如何表示每艘战舰：



每艘战舰都是一个对象。

这个对象包含属性 `locations` 和 `hits`。

```
var ship1 = {  
  locations: ["10", "20", "30"],  
  hits: ["", "", ""]  
};
```

属性 `locations` 是一个数组，其中存储了战舰占据的游戏板单元格。

属性 `hits` 也是一个数组，指出了战舰的各个部位是否被击中。我们将该数组的每个元素都初始化为空字符串，并在战舰的某个部位被击中时将相应的元素改为“hit”。

注意到我们使用两个数字来表示战舰占据的单元格，其中0对应于A，1对应于B，依此类推。

下面演示了如何表示全部三艘战舰：

每艘战舰都用两个数组表示，分别指出了战舰占据的位置以及被击中的部位。

```
var ship1 = { locations: ["10", "20", "30"], hits: ["", "", ""] };  
var ship2 = { locations: ["32", "33", "34"], hits: ["", "", ""] };  
var ship3 = { locations: ["63", "64", "65"], hits: ["", "", "hit"] };
```

我们不使用三个变量来存储这些战舰，而是使用一个数组来存储它们，如下所示：

注意到变量名 `ships` 为复数形式。

将一个数组赋给变量 `ships`，这个数组存储了全部三艘战舰。

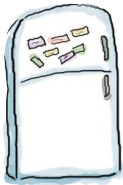
```
var ships = [  
  { locations: ["10", "20", "30"], hits: ["", "", ""] },  
  { locations: ["32", "33", "34"], hits: ["", "", ""] },  
  { locations: ["63", "64", "65"], hits: ["", "", "hit"] }  
];
```

这是第一艘战舰。

这是第二艘。

这是第三艘。

这艘战舰位于游戏板单元格65的部位被击中。



战舰冰箱贴

假设玩家向下面的位置开火。请根据表示战舰的数据结构，将战舰和MISS冰箱贴放置到游戏板的正确位置。玩家击沉了所有的战舰吗？我们指出了玩家第一次开火后的效果。

玩家向下面的位置开火：

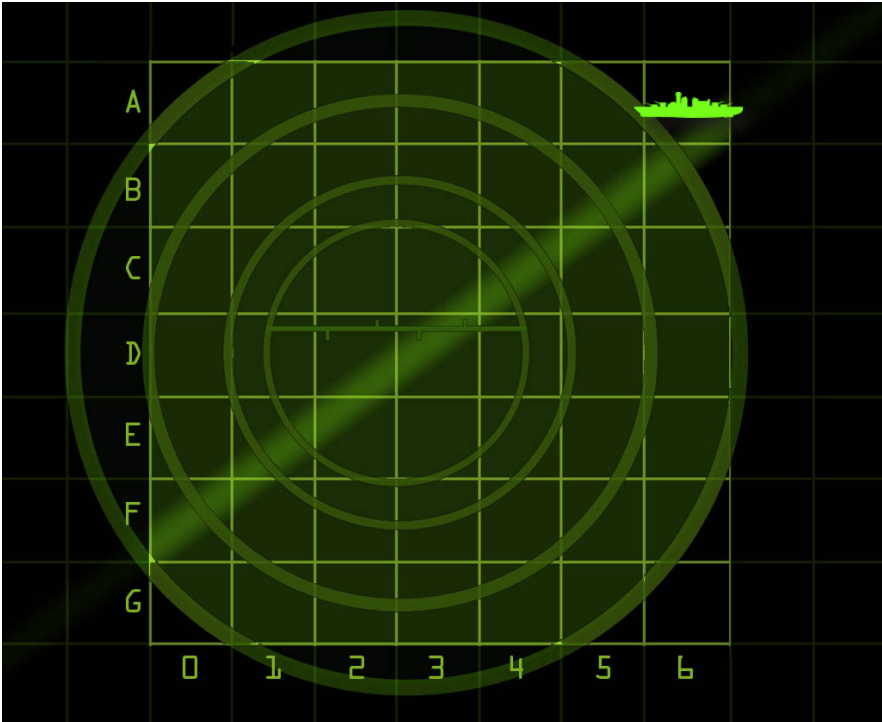
A6, B3, C4, D1, B0, D4, F0, A1, C6, B1, B2, E4, B6

在游戏板上指出玩家这样开火的效果。

```
var ships = [{ locations: ["06", "16", "26"], hits: ["hit", "", ""] },
  { locations: ["24", "34", "44"], hits: ["", "", ""] },
  { locations: ["10", "11", "12"], hits: ["", "", ""] }];
```

这是表示战舰的数据结构。请在其中指出玩家开火后，各艘战舰的哪些部位被击中。

这是游戏板和冰箱贴。



可能不会用完所有的冰箱贴。



下面练习使用数据结构ships来模拟开火的效果。假设数据结构ships如下，请据此回答下面的问题并补全代码。这是游戏工作原理的重要组成部分，请务必查看本章末尾的答案，再继续往下阅读。

```
var ships = [{ locations: ["31", "41", "51"], hits: ["", "", ""] },
              { locations: ["14", "24", "34"], hits: ["", "hit", ""] },
              { locations: ["00", "01", "02"], hits: ["hit", "", ""] }];
```

哪些战舰已被击中？____ 被击中的是哪些部位？____

假设玩家向D4处开火，会击中战舰吗？____ 如果会，击中的是哪艘？____

假设玩家向B3处开火，会击中战舰吗？____ 如果会，击中的是哪艘？____

请补全下面的代码，访问第二艘战舰的中间部位，并使用console.log打印其值：

```
var ship2 = ships[____];
var locations = ship2.locations;
console.log("Location is " + locations[____]);
```

请补全下面的代码，检查第三艘战舰的第一个部位是否被击中：

```
var ship3 = ships[____];
var hits = ship3.____;
if (____ === "hit") {
    console.log("Ouch, hit on third ship at location one");
}
```

请补全下面的代码，指出第一艘战舰的第三个部位被击中：

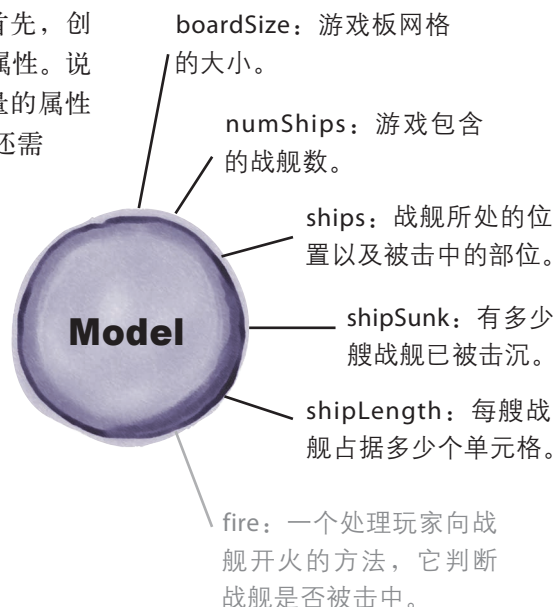
```
var _____ = ships[0];
var hits = ship1._____;
hits[____] = _____;
```

实现model对象

知道如何表示战舰及其被击中的部位后，来编写一些代码。首先，创建model对象，并将前面创建的ships数据结构作为它的一个属性。说到属性，model对象还需包含其他一些属性，如存储战舰数量的属性numShips。你可能会问，我们都知道总共有三艘战舰，为何还需要属性numShips呢？也许你以后要提高游戏的难度，在其中包含4艘或5艘战舰。通过使用一个属性来表示战舰数，而不是使用硬编码值（并在代码中始终使用这个属性，而不是硬编码值），可避免给以后需要修改战舰数时带来麻烦，因为这样只需修改一个地方。

说到硬编码，我们将暂时以硬编码的方式指定战舰的位置。知道战舰的位置后，游戏测试起来将更容易，并可将重点放在游戏的核心逻辑上。后面将编写在游戏板上随机放置战舰的代码。

下面来创建model对象：



model是一个对象。

```
var model = {
  boardSize: 7,
  numShips: 3,
  shipLength: 3,
  shipsSunk: 0,

  ships: [{ locations: ["06", "16", "26"], hits: ["", "", "" ] },
           { locations: ["24", "34", "44"], hits: ["", "", "" ] },
           { locations: ["10", "11", "12"], hits: ["", "", "" ] }
        ]
};
```

这三个属性让我们能够避免使用硬编码值：boardSize（游戏板网格的大小）、numShips（游戏包含的战舰数）和shipLength（每艘战舰占据多少个单元格）。

属性shipsSunk（游戏开始时被初始化为0）指出玩家当前击沉了多少艘战舰。

稍后将随机地生成战舰占据的单元格，现在暂时以硬编码的方式指定它们，以简化游戏测试工作。

属性ships是一个ship对象数组，其中每个元素都存储了一艘战舰的位置和被击中的部位。注意到我们将ships从变量改成了model对象的属性。

现在已经有不少状态了！

另外，对于数组locations和hits，我们以硬编码的方式指定其长度。本书后面将讨论如何动态地生成数组。

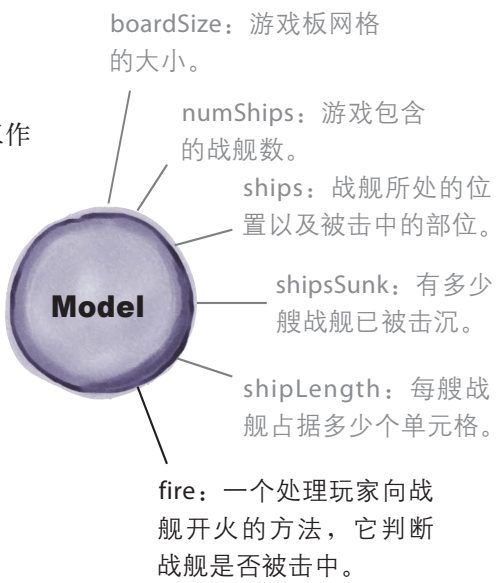
规划方法fire

方法fire判断玩家猜测的位置是否击中了战舰。你知道，显示结果的工作由view对象负责，但方法fire必须提供判断是否击中了战舰的游戏逻辑。

判断是否击中了战舰很容易：给定玩家猜测的位置，你只需如下这样做。

- 检查每艘战舰，看它是否占据了这个位置。
- 如果是，就说明击中了，因此需要设置数组hits的相应元素，并让视图知道击中了。为指出击中了战舰，还需从方法fire返回true。
- 如果没有战舰位于猜测的位置，就没有击中。我们将让视图知道这一点，并从方法fire返回false。

击中了一艘战舰时，方法fire还需判断该战舰是否被击沉。我们将在编写好其他逻辑后再处理这一点。



实现方法fire

下面来建立方法fire的基本框架。这个方法将猜测的位置作为参数，并迭代每艘战舰以确定是否击中了它。这里先不编写击中检测代码，而只是建立大致的框架：

```
var model = {
  boardSize: 7,
  numShips: 3,
  shipsSunk: 0,
  shipLength: 3,
  ships: [{ locations: ["06", "16", "26"], hits: ["", "", ""] },
           { locations: ["24", "34", "44"], hits: ["", "", ""] },
           { locations: ["10", "11", "12"], hits: ["", "", ""] }],
```

别忘了在这里加上一个逗号。

```
fire: function(guess) {
  for (var i = 0; i < this.numShips; i++) {
    var ship = this.ships[i];
  }
}
```

这个方法接受一个参数——guess。

迭代数组ships，每次检查一艘战舰。

获得一艘战舰，接下来需要检查guess是否是该战舰占据的位置之一。

判断是否击中了战舰

在每次循环中，都需要判断`guess`是否是当前战舰占据的位置之一：

```
for (var i = 0; i < this.numShips; i++) {
    var ship = this.ships[i];
    locations = ship.locations;
}
```

遍历每艘战舰。

获取战舰占据的位置，这是战舰的一个数组属性。

需要编写判断`guess`是否位于数组`locations`中的代码。

当前面临的情形如下：我们有一个字符串`guess`，需要在数组`locations`中查找它。如果在数组中找到它，就说明击中了战舰：

```
guess = "16";
locations = ["06", "16", "26"];
```

我们需要判断`guess`的值是否包含在战舰的位置数组中。

为此，可再编写一个循环来遍历数组`locations`的每个元素，并将其与`guess`进行比较；如果它们相同，就说明击中了战舰。

这里不这样做，而是使用另一种更简单的方式：

```
var index = locations.indexOf(guess);
```

方法`indexOf`在数组中查找指定的值；如果找到，就返回相应的索引，否则返回`-1`。

通过使用`indexOf`，可像下面这样编写代码来判断是否击中了战舰：

```
for (var i = 0; i < this.numShips; i++) {
    var ship = this.ships[i];
    locations = ship.locations;
    var index = locations.indexOf(guess);
    if (index >= 0) {
        // 击中了战舰!
    }
}
```

注意到数组的方法`indexOf`与字符串的方法`indexOf`类似。它将一个值作为参数，并返回这个值在数组中的索引（如果没有找到这个值，就返回`-1`）。

因此，如果返回的索引大于或等于零，就意味着玩家猜测的值包含在数组`locations`中，因此击中了战舰。

相比于使用循环，使用`indexOf`并不能提高效率，但需要编写的代码更少，而且代码的意图也更清晰：使用`indexOf`时，更容易明白要在数组中查找什么值。总之，你的编程工具腰带上又多了一个工具。

整合起来

为完成方法`fire`，还需作出一个判断：如果击中了，该怎么办呢？就目前而言，我们只需在模型中进行标记，即将数组`hits`的相应元素设置为`"hit"`。下面来将这些整合起来：

```
var model = {
  boardSize: 7,
  numShips: 3,
  shipsSunk: 0,
  shipLength: 3,
  ships: [ { locations: ["06", "16", "26"], hits: ["", "", ""] },
            { locations: ["24", "34", "44"], hits: ["", "", ""] },
            { locations: ["10", "11", "12"], hits: ["", "", ""] } ],

  fire: function(guess) {
    for (var i = 0; i < this.numShips; i++) {
      var ship = this.ships[i];
      var locations = ship.locations;
      var index = locations.indexOf(guess);
      if (index >= 0) {
        ship.hits[index] = "hit";
        return true;
      }
    }
    return false;
  }
};
```

对于每艘战舰……

如果`guess`包含在数组`locations`中，就说明击中了该战舰。

因此，将数组`hits`的相应元素设置为“hit”。

由于击中了战舰，需要返回`true`。

如果遍历所有战舰后，也没有发现被击中的战舰，就说明没有击中任何战舰，因此返回`false`。

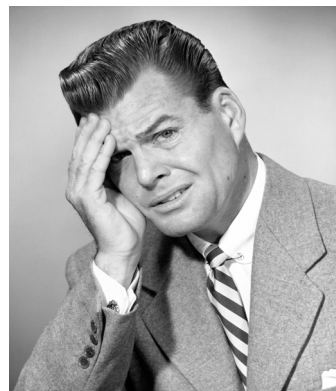
这为实现`model`对象开了一个好头，余下的工作还有两项：判断是否有战舰被击沉；让视图知道模型发生了变化，以便将最新的情况告知玩家。下面就来处理这两项任务。

别这么啰嗦好不好

不好意思，我们不得不旧话重提。前面使用对象和数组时，代码有点啰嗦，再来看一眼这些代码：

```
for (var i = 0; i < this.numShips; i++) {
    var ship = this.ships[i];
    var locations = ship.locations;
    var index = locations.indexOf(guess);
    ...
}
```

← 首先获取战舰。
← 然后，获取战舰的位置（数组locations）。
← 再获取guess在数组locations中的索引。



在有些人看来，这些代码太啰嗦了。为什么呢？因为可使用串接（chaining）来简化这些引用。串接让我们能够将对象引用连接起来，从而避免创建临时变量，如上述代码中的locations。

你可能会问，为何说locations是临时变量呢？因为我们只将locations用来临时存储ship.locations，以便对其调用方法indexOf来获取guess的索引；在这个方法的其他任何地方，都不需要locations。通过使用串接，可避免使用临时变量locations，如下所示：

```
var index = ship.locations.indexOf(guess);
```

← 将上面标出的两行代码合并成了一行。



串接的工作原理

串接只是一种快捷方式，可避免使用多个步骤来访问对象（和数组）的属性和方法。下面来详细地看看前面使用串接将两条语句合而为一的做法：

```
var ship = { locations: ["06", "16", "26"], hits: ["", "", ""] };
var locations = ship.locations;
var index = locations.indexOf(guess);
```

← 这是一个ship对象。
← 获取ship对象中的数组locations。
← 再通过这个数组来访问方法indexOf。

通过将表达式串接起来，可将最后两条语句合而为一，并避免使用变量locations：

ship.locations.indexOf(guess)

① ship对象 ② 这个对象包含数组属性locations ③ 这个数组包含方法indexOf

回到战舰

现在需要编写判断战舰是否被击沉的代码。判断准则你已经知道了：当所有部位都被击中时，战舰就被击沉。我们可添加一个小小的辅助方法来检查战舰是否被击沉：

我们将这个方法命名为`isSunk`。它接受一艘战舰作为参数，在该战舰被击沉时返回`true`，在它还浮在水面上时返回`false`。

```
isSunk: function(ship) {
  for (var i = 0; i < this.shipLength; i++) {
    if (ship.hits[i] !== "hit") {
      return false;
    }
  }
  return true;
}
```

这个方法将一艘战舰作为参数，并检查是否其每个部位都被击中。

只要有任何部位未被击中，战舰就还浮在水面上，因此返回`false`。

否则，战舰已被击沉，因此返回`true`。

请将这个方法添加到对象`model`中——紧跟在方法`fire`后面。

现在，可以在方法`fire`中调用这个方法来判断战舰是否被击沉了：

```
fire: function(guess) {
  for (var i = 0; i < this.numShips; i++) {
    var ship = this.ships[i];
    var index = ship.locations.indexOf(guess);
    if (index >= 0) {
      ship.hits[index] = "hit";
      if (this.isSunk(ship)) {
        this.shipsSunk++;
      }
      return true;
    }
  }
  return false;
},
```

确定战舰被击中后，执行这个检查。如果战舰被击沉，就将击沉的战舰数（存储在`model`对象的属性`shipsSunk`中）加1。

```
isSunk: function(ship) { ... }
```

紧跟在方法`fire`的后面添加新方法`isSunk`。务必在`model`对象的属性和方法之间加上逗号，千万别忘了！

最后一击

model对象就要实现好了。模型存储游戏状态，并包含判断猜测的位置是否击中了战舰的逻辑。现在唯一缺失的是将判断结果告诉视图的代码，下面就来编写它们：

```
var model = {
  boardSize: 7,
  numShips: 3,
  shipsSunk: 0,
  shipLength: 3,
  ships: [ { locations: ["06", "16", "26"], hits: ["", "", ""] },
            { locations: ["24", "34", "44"], hits: ["", "", ""] },
            { locations: ["10", "11", "12"], hits: ["", "", ""] } ],
  fire: function(guess) {
    for (var i = 0; i < this.numShips; i++) {
      var ship = this.ships[i];
      var index = ship.locations.indexOf(guess);
      if (index >= 0) {
        ship.hits[index] = "hit";
        view.displayHit(guess);
        view.displayMessage("HIT!");
        if (this.isSunk(ship)) {
          view.displayMessage("You sank my battleship!");
          this.shipsSunk++;
        }
        return true;
      }
      view.displayMiss(guess);
      view.displayMessage("You missed.");
      return false;
    },
    isSunk: function(ship) {
      for (var i = 0; i < this.shipLength; i++) {
        if (ship.hits[i] !== "hit") {
          return false;
        }
      }
      return true;
    }
  };
```

这里列出了完整的model对象，旨在让你在一个地方看到它。

告诉视图，玩家的猜测击中了战舰。

并让视图显示消息“HIT!”。

让玩家知道他击沉了一艘战舰！

告诉视图，玩家的猜测没有击中战舰。

让视图显示消息“You missed.”。

前面说过，view对象的这些方法将一个元素的属性class设置为“hit”或“miss”，这个元素的id由字符串guess指定的行号和列号组成。换句话说，view对象将数组hits中的“hit”转换为HTML中的“hit”。但别忘了，HTML中的“hit”只影响界面显示，而model对象中的“hit”表示的是实际状态。



试驾



要获得这里所示的结果，需要删除或注释掉前面测试视图的代码。文件battleship_tester.js演示了如何做。

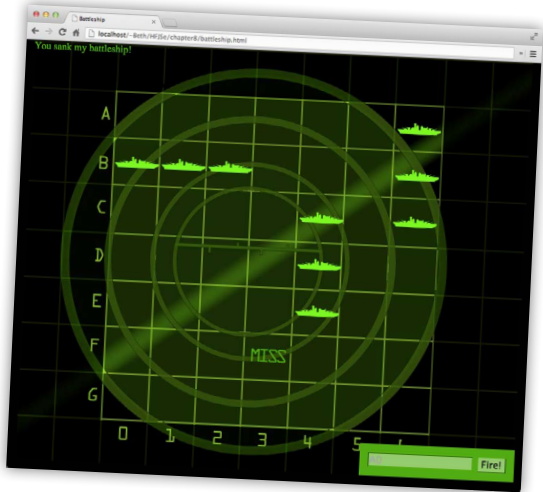
```
model.fire("53");
```

```
model.fire("06");  
model.fire("16");  
model.fire("26");
```

```
model.fire("34");  
model.fire("24");  
model.fire("44");
```

```
model.fire("12");  
model.fire("11");  
model.fire("10");
```

重新加载battleship.html，你将在游戏板上看到战舰和MISS图像。



世上没有愚蠢的问题

问： 使用串接将语句合并一定胜过让语句分开吗？

答： 不一定。串接并不能极大地提高效率（可节省一个变量），但确实让代码更简短。我们认为，串接链条不长（最多两三层）时，阅读起来比分为多行代码更容易，但这只是我们的喜好。如果你想让语句分开，也完全可以。如果你要使用串接，一定不要让串接链条太长；否则阅读和理解起来都更难。

问： 在这里，数组（ships）里面有对象（ship），而对象又包含数组（locations）。像这样嵌套对象和数组时，最多可嵌套多少层？

答： 从理论上说，想嵌套多少层都可以，但实际上，嵌套层数不会太多。如果你发现嵌套层数超过了三四层，可能说明你的数据结构太复杂，需要重新考虑组织方式。

问： 我注意到，model对象包含属性boardSize，但从未使用过它。这个属性是做什么用的？

答： 在后面编写的代码中，将使用boardSize以及model对象的其他属性。model的职责是管理游戏的状态，而boardSize绝对是一个不可或缺的状态。控制器将通过访问model对象的属性来了解游戏的状态，后面还将给model对象添加其他使用这些属性的方法。

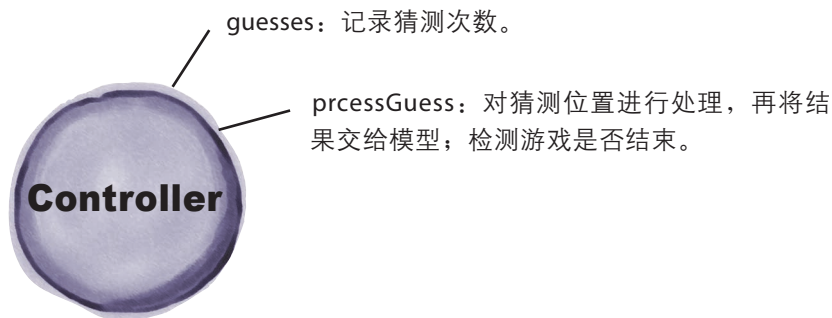
实现控制器

实现好视图和模型后，下面来实现控制器，将这个应用程序的各个部分整合起来。从较高的层次看，控制器获取和处理猜测，并将其交给模型，从而将各个部分整合起来。它还跟踪一些管理型细节，如已猜测多少次以及游戏的进度。为此，控制器依赖于模型来跟踪游戏状态，并依赖于视图来显示游戏界面。

具体地说，我们将把如下职责交给控制器。

- 获取并处理玩家的猜测（如A0或B1）。
- 记录猜测次数。
- 让模型根据当前猜测更新自己。
- 判断游戏是否结束（即是否击沉了所有的战舰）。

为实现控制器，首先在controller对象中定义一个属性guesses。然后实现一个方法processGuess，它接受一个由字母和数字组成的猜测位置，对其进行处理，并将结果交给模型。



controller对象的基本框架如下，接下来的几页将填充其代码：

```
var controller = {
  guesses: 0,
  processGuess: function(guess) {
    // 其他代码
  }
};
```

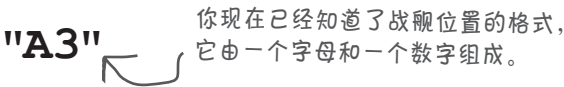
这里定义了controller对象，它包含一个属性guesses，这个属性被初始化为0。

这是方法processGuess的方法头，它将一个格式为“A0”的猜测位置作为参数。

处理玩家的猜测

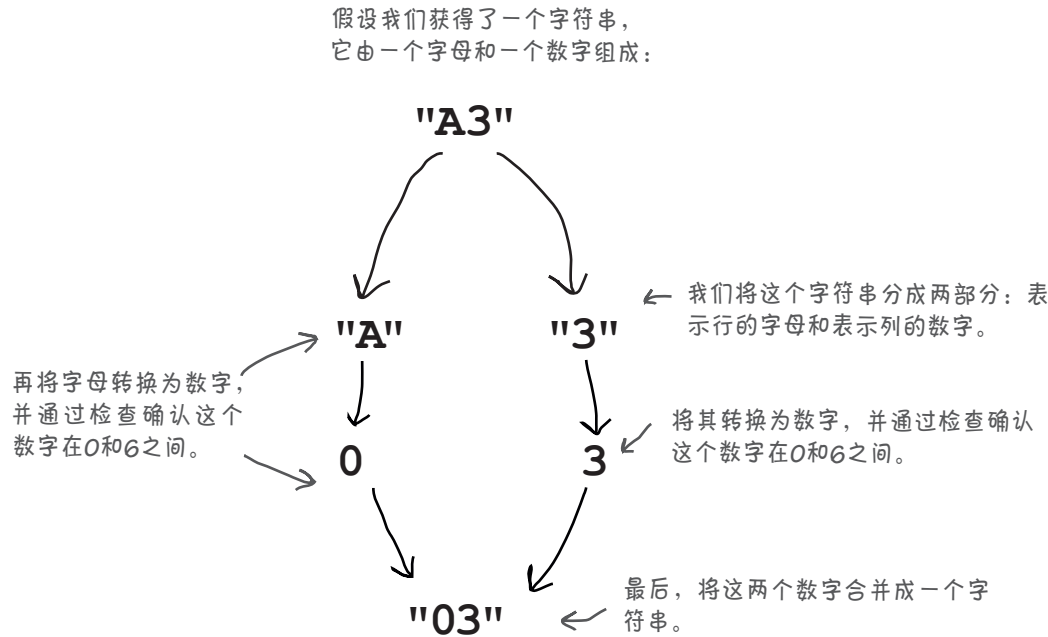
控制器的职责是获取玩家的猜测，确认猜测的位置是有效的，再将其交给model对象。但它从哪里获取玩家的猜测呢？现在不要考虑这一点，稍后将介绍。就目前而言，我们只需这样假设：在某个地方，有代码调用controller对象的方法processGuess，并给它提供一个格式如下的字符串。

这是一个很有用的编程技巧——专注于当前要编写的代码的需求。同时考虑整个问题通常更难成功。



获得这种格式的猜测位置（一个字母和一个数字组成的字符串，如"A3"）后，你需要将其转换为模型能够理解的格式（两个数字组成的字符串，如"03"）。下面简要地演示了如何将有效的输入转换为只包含数字的格式。

当然，玩家不会输入一个无效的猜测，对吧？我们最好确保输入是有效的。



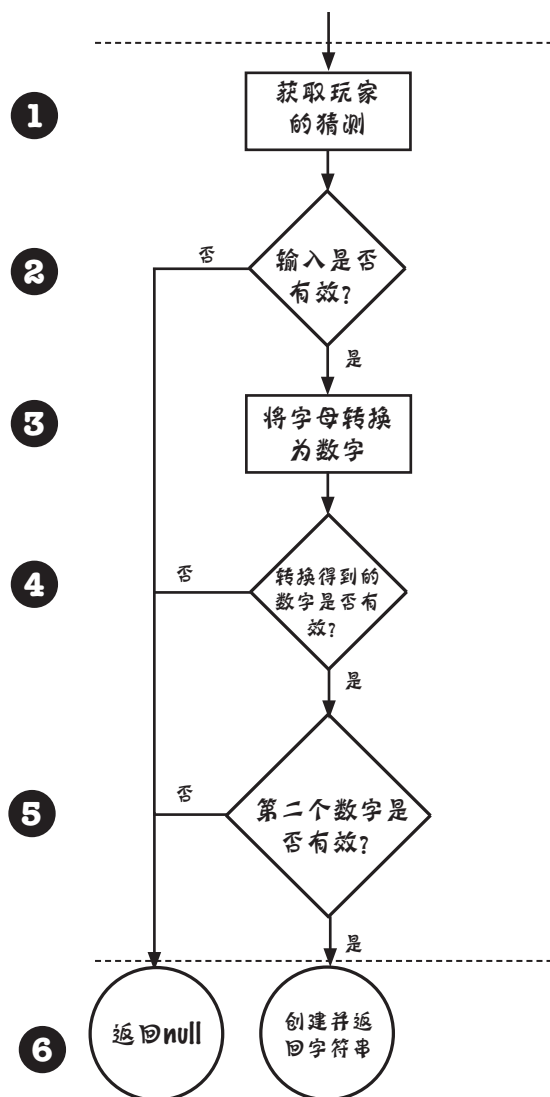
重要的事情先做。我们必须先检查输入是有效的。编写代码前，我们先来进行规划。

规划代码

我们不将处理猜测的代码都放在方法processGuess中，而是编写一个小小的辅助函数（因为可能需要重用这些代码），并将其命名为parseGuess。

编写代码前，先来看看它将如何工作。

- 1** 获取由一个字母和一个数字组成的玩家猜测。
- 2** 对输入进行检查，确认它是有效的（不为null，且不太长也不太短）。
- 3** 将其中的字母转换为数字：A转换为0，B转换为1，依此类推。
- 4** 检查第3步转换得到的数字是否有效（在0和6之间）。
- 5** 检查第二个数字是否有效（也在0和6之间）。
- 6** 如果任何检查失败，都返回null；否则，将这两个数字拼接成一个字符串并将其返回。



实现parseGuess

制订可靠的编码计划后，来动手编写代码吧。

1—2

咱们先来完成第1步和第2步。我们只需将玩家的猜测作为参数，并通过检查确认它是有效的。当前，我们只需这样定义有效性：一个不为null且包含两个字符的字符串。

将猜测的位置赋给形参guess。

↓

```
function parseGuess(guess) {
  if (guess === null || guess.length !== 2) {
    alert("Oops, please enter a letter and a number on the board.");
  }
}
```

然后检查guess不为null且长度为2。

↑

如果不是这样的，就提醒玩家。

3

接下来，我们获取其中的字母，并使用一个包含字母A~G的辅助数组将其转换为数字。为此，可使用方法indexOf获取这个字母在数组中的索引，如下所示。

一个数组，它包含可出现在有效猜测中的所有字母。

↓

```
function parseGuess(guess) {
  var alphabet = ["A", "B", "C", "D", "E", "F", "G"];

  if (guess === null || guess.length !== 2) {
    alert("Oops, please enter a letter and a number on the board.");
  } else {
    firstChar = guess.charAt(0);
    var row = alphabet.indexOf(firstChar);
  }
}
```

获取guess中的第一个字符。

↑

再使用indexOf获取一个0~6的数字，它是这个字母在数组中的位置。为明白其中的工作原理，请尝试几个这样的示例。

4

5

检查组成猜测位置的两个字符，看看它们是否都是0~6的数字，即确认它们都是有效的游戏板位置。

```
function parseGuess(guess) {
    var alphabet = ["A", "B", "C", "D", "E", "F", "G"];

    if (guess === null || guess.length !== 2) {
        alert("Oops, please enter a letter and a number on the board.");
    } else {
        firstChar = guess.charAt(0);
        var row = alphabet.indexOf(firstChar);
        var column = guess.charAt(1);

        if (isNaN(row) || isNaN(column)) {
            alert("Oops, that isn't on the board.");
        } else if (row < 0 || row >= model.boardSize ||
            column < 0 || column >= model.boardSize) {
            alert("Oops, that's off the board!");
        }
    }
}
```

获取字符串中的第二个字符，它表示列号。

使用函数isNaN检查row和column是否都是数字。

我们还确认这些数字都在0和6之间。

这里大量利用了自动类型转换！column是一个字符串，检查它的值是否在0和6之间时，我们利用了自动类型转换来将其转换为数字，以便进行比较。

实际上，这里的做法更通用。我们没有以硬编码的方式指定数字6，而是让模型告诉我们游戏板有多大，并同这个数字进行比较。



考考你的
脑力

这里没有以硬编码的方式将row和column可存储的最大值指定为6，而是使用了model对象的属性boardSize。从长远看，你认为这样做有何优点？

6

现在来编写函数parseGuess的最后一部分代码。如果前述有效输入检查失败，就返回null；否则将猜测的行号和列号合并成一个字符串，并将其返回。

```
function parseGuess(guess) {
    var alphabet = ["A", "B", "C", "D", "E", "F", "G"];

    if (guess === null || guess.length !== 2) {
        alert("Oops, please enter a letter and a number on the board.");
    } else {
        firstChar = guess.charAt(0);
        var row = alphabet.indexOf(firstChar);
        var column = guess.charAt(1);

        if (isNaN(row) || isNaN(column)) {
            alert("Oops, that isn't on the board.");
        } else if (row < 0 || row >= model.boardSize ||
            column < 0 || column >= model.boardSize) {
            alert("Oops, that's off the board!");
        } else {
            return row + column;
        }
    }
    return null;
}
```

至此，row和column都有效，因此返回它们。

注意到我们将row和column拼接成了一个字符串，并将其返回。这里再次利用了自动类型转换：row是一个数字，而column是一个字符串，因此结果是一个字符串。

如果执行到了这里，说明有检查是失败的，因此返回null。



试驾

请在battleship.js中输入这些代码，再添加类似下面的函数调用代码：

```
console.log(parseGuess("A0"));
console.log(parseGuess("B6"));
console.log(parseGuess("G3"));
console.log(parseGuess("H0"));
console.log(parseGuess("A7"));
```

重新加载battleship.html，并确保打开了控制台窗口。你将在控制台窗口中看到parseGuess的结果，还可能有一两条提示消息。

JavaScript控制台

```
00
16
63
null
null
```

回到控制器

辅助函数`parseGuess`编写好后，接着来实现控制器。首先，将函数`parseGuess`与既有的控制器代码整合起来：

```
var controller = {
  guesses: 0,

  processGuess: function(guess) {
    var location = parseGuess(guess);
    if (location) {
    }
  }
};
```

使用`parseGuess`来验证玩家猜测的有效性。

只要返回的不是`null`，就说明获得的位置是有效的。

别忘了，`null`是一个假值。

将在这里添加其他控制器代码。

这就实现了控制器的第一项职责，来看看还有哪些职责需要实现。

- 获取并处理玩家的猜测（如A0或B1）。
- 记录猜测次数。
- 让模型根据当前猜测更新自己。
- 判断游戏是否结束（即是否击沉了所有的战舰）。

} ← 我们将处理这些工作。

记录猜测并开火的次数

下一项任务很简单：为记录猜测次数，只需在玩家每次猜测时都将属性`guesses`加1。你将看到，我们决定在玩家的猜测无效时，不增加猜测次数。

接下来，我们将调用`model`对象的方法`fire`，让模型根据当前猜测更新自己。毕竟，玩家进行猜测的目的是要开火，并希望击中战舰。别忘了，方法`fire`将一个字符串作为参数，其中包含行号和列号，而要获得这个字符串，可调用`parseGuess`。非常方便。

下面将这些整合起来，以实现下一项职责：

```
var controller = {  
  guesses: 0,
```

```
  processGuess: function(guess) {  
    var location = parseGuess(guess);  
    if (location) {  
      this.guesses++;  
      var hit = model.fire(location);  
    }  
  }  
};
```

如果玩家的猜测有效，
就将`guesses`加1。

别忘了，`this.guesses++`将
属性`guesses`加1，其工作原
理与循环中的`i++`完全相同。

请注意，如果玩家输入的
游戏板位置无效，我们不
会惩罚玩家——不计入猜
测次数。

接下来，我们以字符串的方式将行号和列号传递
给`model`对象的方法`fire`。别忘了，仅当击中了战
舰时，方法`fire`才返回`true`。

判断游戏是否结束

现在余下的全部工作就是判断游戏是否结束。如何判断呢？我们知道，三艘战舰都被击沉时，游戏便结束了。因此，每次击中战舰时，我们都使用属性`model.shipsSunk`确定击沉的战舰数是否为三艘。为让代码更通用些，我们不将这个属性与数字3进行比较，而是将其与`model`对象的属性`numShips`进行比较。这样，当你以后决定修改战舰数（如改为2或4）时，就无需修改这些代码了。

```
var controller = {  
  guesses: 0,
```

```
  processGuess: function(guess) {  
    var location = parseGuess(guess);  
    if (location) {  
      this.guesses++;  
      var hit = model.fire(location);  
      if (hit && model.shipsSunk === model.numShips) {  
        view.displayMessage("You sank all my battleships, in " +  
          this.guesses + " guesses");  
      }  
    }  
  }  
};
```

如果击中了战舰，且击沉的战舰数与游
戏包含的战舰数相等，就向玩家显示一
条消息，指出他击沉了所有的战舰。

我们还向玩家指出，他经过多少次猜测
就击沉了所有的战舰。其中的`guesses`
是`this`对象（即`controller`对象）的一个
属性。



试驾

请在文件**battleship.js**中输入控制器的所有代码，再添加一些对控制器进行测试的函数调用代码。然后，重新加载网页**battleship.html**，注意到游戏板上将显示战舰和MISS图像。这些图像所处的位置正确吗？要查看我们使用的测试代码，请下载**battleship_tester.js**。

同样，要获得这里所示的结果，你必须删除或注释掉以前的测试代码。**battleship_tester.js**演示了如何这样做。



```
controller.processGuess("A0");
```

```
controller.processGuess("A6");
```

```
controller.processGuess("B6");
```

```
controller.processGuess("C6");
```

```
controller.processGuess("C4");
```

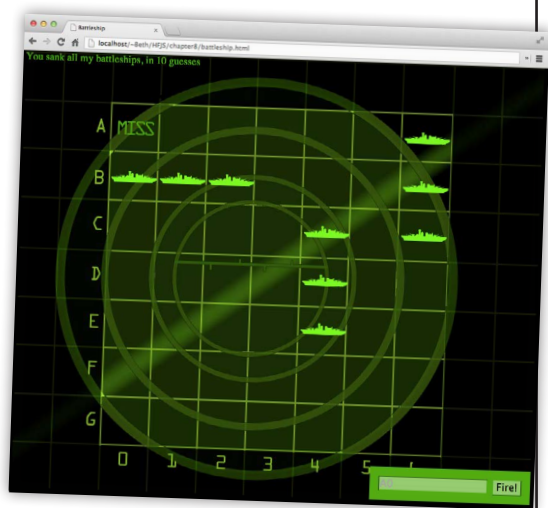
```
controller.processGuess("D4");
```

```
controller.processGuess("E4");
```

```
controller.processGuess("B0");
```

```
controller.processGuess("B1");
```

```
controller.processGuess("B2");
```



调用**controller**对象的方法**processGuess**，并传入由一个字母和一个数字组成的猜测位置。

考考你的脑力

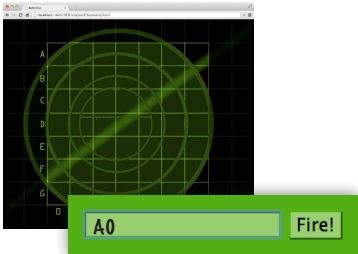
玩家击沉全部三艘战舰后，我们会通过消息区域告诉玩家游戏结束了，但玩家还可以接着猜测。该如何在玩家击沉全部战舰后禁止输入猜测，从而修复这个问题呢？

获取玩家的猜测

实现游戏的核心逻辑和显示功能后，需要让玩家输入猜测并获取玩家的猜测，这样玩家才能真正地玩这个游戏。你可能还记得，我们已经在HTML中包含了一个用于输入猜测的<form>元素，但如何将其与游戏关联起来呢？

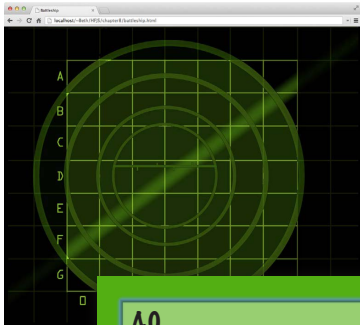
为此，需要一个事件处理程序。本书前面简要地讨论了事件处理程序，下面来使用它们让游戏能够正常运行；有关事件处理程序的详细内容将在下一章介绍。这里的目标是，让你对事件处理程序和表单元素如何协同工作有大致的认识，而不要求你现在就对事件处理程序了如指掌。

情况大致是这样的：



这是我们的HTML表单元素，可用于获取用户输入。

- ① 玩家输入猜测并单击Fire!按钮；
- ② Fire!按钮被单击时，将调用一个预先指定的事件处理程序；
- ③ 这个事件处理程序获取表单中的玩家输入，并将其交给控制器。



如何给Fire!按钮添加事件处理程序

为实现这些功能，需要给Fire!按钮添加一个事件处理程序。为此，首先需要根据这个按钮的id获得一个指向它的引用。再来看看HTML，你将发现Fire!按钮的id为fireButton。知道按钮的id后，只需调用document.getElementById就可获得指向它的引用。有了指向按钮的引用后，就可将其属性onclick设置为一个事件处理程序，如下所示：

需要将这段代码放在一个地方，因此我们创建一个名为init的函数。

首先，使用Fire!按钮的id获取一个指向它的引用。

```
function init() {
    var fireButton = document.getElementById("fireButton");
    fireButton.onclick = handleFireButton;
}
```

然后，给这个按钮添加单击事件处理程序handleFireButton。

别忘了定义函数handleFireButton。

```
function handleFireButton() {
    // 从表单中获取值的代码
}
```

这是函数handleFireButton。每当玩家单击Fire!按钮时，都将调用这个函数。

我们马上就会编写这些代码。

第6章介绍过，我们希望浏览器在网页加载完后执行函数init。

```
window.onload = init;
```

获取表单中玩家的猜测

开火是通过单击Fire!按钮发起的，但玩家的猜测包含在表单元素guessInput中。要获取这个表单元素中的值，可访问其属性value，如下所示：

首先，使用这个表单元素的id (guessInput) 来获取一个指向它的引用。

```
function handleFireButton() {
    var guessInput = document.getElementById("guessInput");
    var guess = guessInput.value;
}
```

然后，从这个表单元素中获取猜测，它存储在这个表单元素的属性value中。

获取猜测后，就可以使用它了。我们已经编写了大量使用它的代码，下面就将它交给这些代码。

将输入交给控制器

下面来将代码关联起来。我们有一个控制器，它急不可耐地想获取玩家的猜测。我们只需将玩家的猜测交给控制器即可，如下所示：

```
function handleFireButton() {  
    var guessInput = document.getElementById("guessInput");  
    var guess = guessInput.value;  
    controller.processGuess(guess);  
  
    guessInput.value = "";  
}
```

我们将玩家的猜测交给控制器，然后一切就像魔术一样发生了！

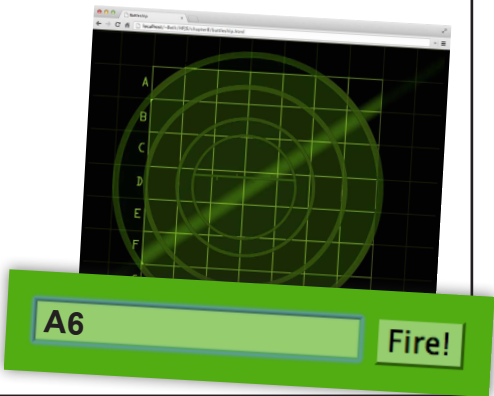
这行代码将表单输入元素的值重置为空字符串。这样，玩家再次猜测时，就无需选择并删除前一次的猜测了，否则将让玩家很恼火。



试驾

这不仅仅是试驾，终于可以好好地玩这款游戏了！请将所有的代码都添加到battleship.js中，再在浏览器中重新加载battleship.html。当前，战舰的位置是以硬编码的方式指定的，因此你很清楚如何在游戏中获胜。下面列出了获胜的招式，但务必对代码进行详尽的测试：输入不能击中战舰的猜测、无效的猜测以及显然是错误的猜测。

- A6
 - B6
 - C6
 - C4
 - D4
 - E4
 - B0
 - B1
 - B2
- 这些是获胜的招式，它们是按战舰排列的，但不必完全按这样的顺序进行猜测。请尝试输入其他的猜测：在这些猜测之间输入无效的猜测以及无法击中战舰的猜测。这对质量保证测试来说不可或缺。





编码技巧

每次猜测时，玩家都必须单击Fire!按钮，是不是很傻？单击确实管用，但这种做法既不快捷也不方便。如果玩家只需按回车键就行，是不是容易得多？下面是处理按回车键事件的代码：

```
function init() {
    var fireButton = document.getElementById("fireButton");
    fireButton.onclick = handleFireButton;
    var guessInput = document.getElementById("guessInput");
    guessInput.onkeypress = handleKeyPress;
}
```

↑ 添加一个新的处理程序，用于处理HTML输入字段的按键事件。

这是按键事件处理程序。每当用户在这个表单输入字段中按键时，都将调用这个处理程序。

浏览器向事件处理程序传递一个事件对象，其中包含有关用户按下了哪个键的信息。

```
function handleKeyPress(e) {
    var fireButton = document.getElementById("fireButton");
    if (e.keyCode === 13) {
        fireButton.click();
        return false;
    }
}
```

如果用户按下的是回车键，事件对象的属性keyCode将为13。在这种情况下，我们希望Fire!按钮就像自己被单击一样行事。为此，我们调用fireButton的方法click，让它以为自己被单击了。

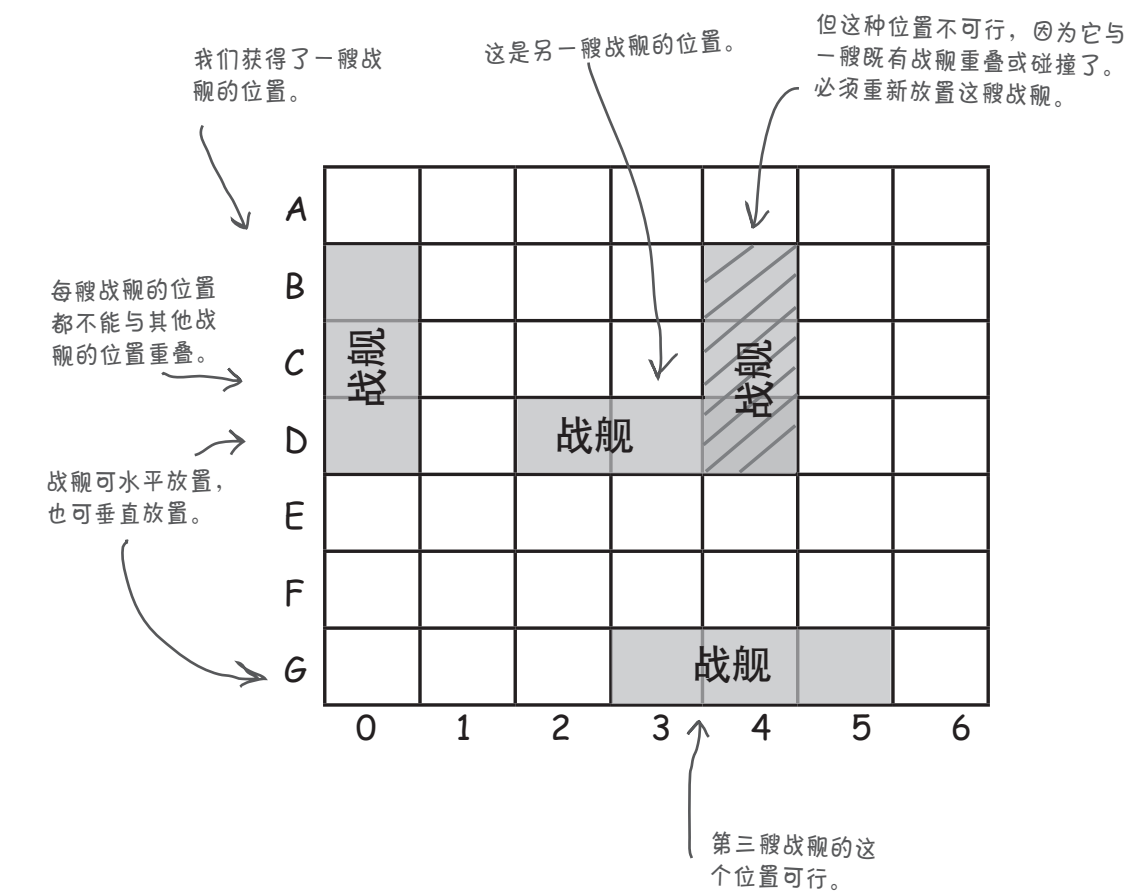
我们返回false，让表单不做其他任何事情（如提交）。

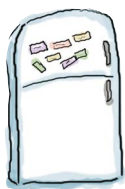
请更新你的init函数，并在代码中添加函数handleKeyPress（任何地方都可以）。重新加载网页，开始游戏吧！

还有哪些地方要处理呢？哦，讨厌，战舰位置是以硬编码方式指定的

至此，你创建了一个基于浏览器的游戏。它虽然只包含少量的HTML、几幅图像和100行左右的JavaScript代码，却非常惊艳。然而，这款游戏有一个地方不太令人满意，那就是每次玩游戏时，战舰都位于相同的位置。你还需要编写一些代码，在每次开始新游戏时随机地放置战舰，否则这款游戏就太无趣了。

这样做之前，我们想让你知道，介绍这些代码时，我们将稍微加快点步伐，因为你阅读和理解代码的能力更强了，而且这些代码也没有太多新东西。下面就开始吧。需要考虑的因素如下。





代码冰箱贴

算法不过是一种花哨的说法，其实指的就是一系列解决问题的步骤。

下面是一系列散乱的冰箱贴，请将它们按照正确的顺序放置，组成一个生成战舰位置的算法。继续往下阅读前，请查看本章末尾的答案。

为新战舰生成随机位置。

执行循环，循环次数等于要创建的战舰数。

为新战舰生成随机方向（垂直或水平）。

将新战舰加入到ships数组中。

检查新战舰的位置是否与既有战舰的位置冲突。

如何放置战舰

在游戏板上放置战舰时，需要考虑的因素有两个。首先，战舰可水平放置，也可垂直放置。其次，战舰在游戏板上不能彼此重叠。我们将编写处理这两种约束的代码。前面说过，我们不会详尽地介绍这些代码，但你具备理解这些代码所需的全部知识。只要花足够的时间，你就能理解这些代码的每个细节，因为其中设计的所有内容都在本书前面介绍过（有一项例外，我们将详细讨论）。我们开始吧。

我们将把这些代码组织为model对象的三个方法。

- **generateShipLocations:** 这是主方法，它创建model对象中的ships数组，其中包含的战舰数由model对象的属性numShips指定。
- **generateShip:** 这个方法创建一艘战舰，并指定其在游戏板中的位置。指定的位置可能与其他战舰重叠，也可能不重叠。
- **collision:** 这个方法将一艘战舰作为参数，并判断它是否与游戏板中既有的战舰重叠。

方法generateShipLocations

首先来实现方法generateShipLocations。这个方法使用循环不断地创建战舰，直到在model对象的数组ships中添加足够数量的战舰。在每次循环中，都使用方法generateShip生成一艘新战舰，并使用方法collision来确保不与其他战舰重叠。如果发生重叠，就将战舰扔掉，并接着尝试。

需要注意的一点是，在这些代码中，我们将使用一种新的迭代器——do while循环。do while循环的工作原理与while循环几乎相同，只是先执行循环体中的语句，再检查循环条件。对于有些逻辑条件，使用do while循环比使用while循环更合适，但是这样的情况很少见。

我们将在model对象中添加这个方法。

```
generateShipLocations: function() {
    var locations;
    for (var i = 0; i < this.numShips; i++) {
        do {
            locations = this.generateShip();
        } while (this.collision(locations));
        this.ships[i].locations = locations;
    }
},
```

这里使用了do while循环！

循环次数与要为其生成位置的战舰数相同。

生成战舰占据的一系列位置

并检查这些位置与游戏板中既有战舰的位置是否重叠。如果重叠，就需要再次尝试，不断地生成新位置，直到不重叠为止。

生成可行的位置后，将其赋给数组model.ships中相应战舰的属性locations。

编写方法generateShip

方法generateShip创建一个数组，其中包含一艘战舰的随机位置，而不关心战舰是否与游戏板中的其他战舰重叠。我们将用分两步来实现这个方法。第一步为战舰随机地选择方向：水平还是垂直的？我们将用一个随机数来决定这一点：如果这个随机数为1，战舰将是水平的；如果为0，战舰将是垂直的。我们将使用老朋友Math.random和Math.floor像以前一样来完成这项任务。

生成随机数0或1
有点像抛硬币。

使用Math.random来生成一个0~1的随机数，再将结果乘以2，得到一个0~2（但不包括2）的随机数。然后，使用Math.floor将这个随机数转换为0或1。

这个方法也将添加到model对象中。

```
generateShip: function() {
```

```
    var direction = Math.floor(Math.random() * 2);
```

```
    var row, col;
```

```
    if (direction === 1) {
```

```
        // 生成水平战舰的起始位置
```

```
    } else {
```

```
        // 生成垂直战舰的起始位置
```

```
    }
```

我们将首先生成新战舰的起始位置，如row=0和column=3。我们需要根据战舰的方向使用不同的规则来生成起始位置（其中的原因将在稍后介绍）。

为创建新战舰的locations属性，我们首先创建一个空数组，再在其中逐一添加位置。

```
    var newShipLocations = [];
```

```
    for (var i = 0; i < this.shipLength; i++) {
```

```
        if (direction === 1) {
```

```
            // 在水平战舰的位置数组中添加位置
```

```
        } else {
```

```
            // 在垂直战舰的位置数组中添加位置
```

```
        }
```

```
    }
```

```
    return newShipLocations;
```

```
},
```

生成所有的位置后，
返回这个数组。

我们使用一个循环，其循环次数为战舰占据的单元格数。

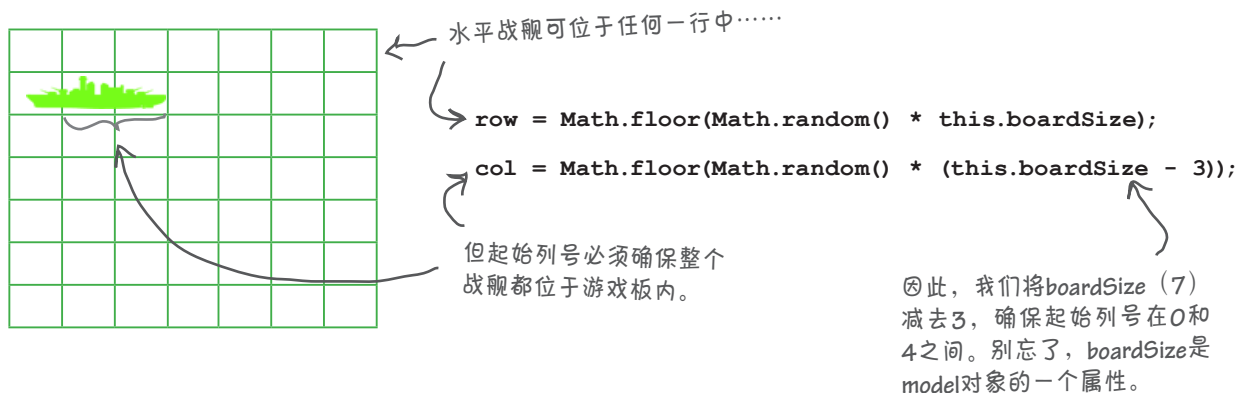
在每次循环中，都在数组newShipLocations中添加一个单元格。同样，需要根据战舰的方向使用稍微不同的代码来生成单元格。

我们将从下一页开始补全余下的代码。

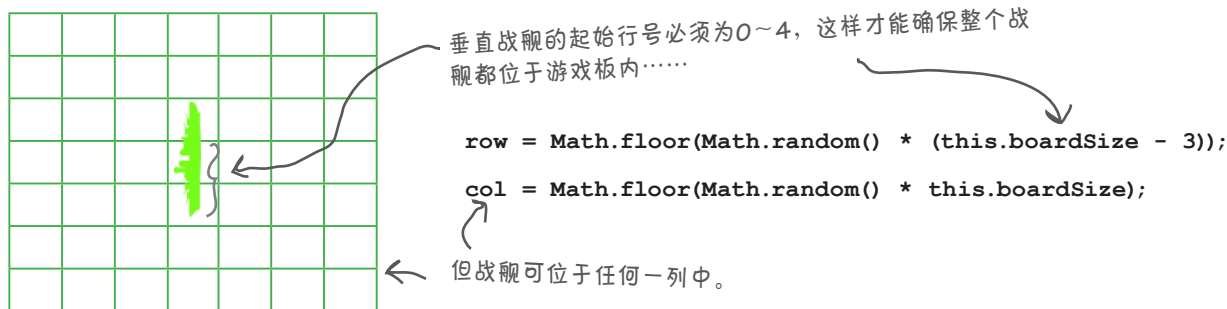
生成新战舰的起始位置

知道战舰的方向后，就可以为其生成位置了。我们将首先生成起始位置（战舰占据的第一个单元格）：如果战舰是水平的，余下的位置将为下两列相应的单元格；如果战舰是垂直的，余下的位置将为下两行相应的单元格。

为此，需要生成两个表示战舰起始位置的随机数：一个为行号，另一个为列号。这两个随机数都必须在0和6之间，这样才能确保战舰位于游戏板内。但别忘了，如果战舰是水平放置的，那么起始列号必须在0和4之间，这样才能确保整个战舰都在游戏板内。



同理，如果战舰是垂直放置的，起始行号就必须在0和4之间，这样才能确保整个战舰都在游戏板内。



完成方法generateShip的编写工作

插入前一页所示的代码后，余下的唯一工作就是将起始位置和接下来的两个单元格添加到数组newShipLocations中。

```
generateShip: function() {
    var direction = Math.floor(Math.random() * 2);
    var row, col;

    if (direction === 1) {
        row = Math.floor(Math.random() * this.boardSize);
        col = Math.floor(Math.random() * (this.boardSize - this.shipLength));
    } else {
        row = Math.floor(Math.random() * (this.boardSize - this.shipLength));
        col = Math.floor(Math.random() * this.boardSize);
    }

    var newShipLocations = [];
    for (var i = 0; i < this.shipLength; i++) {
        if (direction === 1) {
            newShipLocations.push(row + "" + (col + i));
        } else {
            newShipLocations.push((row + i) + "" + col);
        }
    }
    return newShipLocations;
},
```

这些代码生成战舰在游戏板中的起始位置。

将前一页的数字3替换成了this.shipLength，让代码更通用，支持任何战舰长度。

这是生成水平战舰位置的代码。下面来分别阐述其各个部分……

这里使用圆括号确保先将i和col相加，再将结果转换为字符串。

将一个新位置压入数组newShipLocations。

这个位置是一个字符串，它由行号（刚才计算得到的起始行号）……

……和列号（起始列号+i）组成。第一次循环时，i为0，因此列号为起始列号；第二次循环时，为下一列；而第三次循环时，为再下一列。因此，这个数组将存储类似于“01”“02”和“03”的数据。

执行的操作与前面类似，只是针对的是垂直战舰。

因此，每次循环时，都将行号（而不是列号）加i。

对于垂直战舰，这个数组将存储类似于“31”“41”和“51”的数据。

别忘了，将字符串和数字相加时，+表示拼接而不是加法运算，因此结果为一个字符串。

使用战舰的位置填充这个数组后，将其返回给调用方法generateShipLocations。

避免碰撞

方法collision将战舰的位置作为参数，并检查这些位置是否与游戏板中已有的战舰重叠（或碰撞）。
← 要知道我们在什么地方调用了方法collision，请回过头去看第364页。

为实现这种功能，我们使用了两个嵌套循环。外部循环迭代模型（属性model.ships）中所有的战舰；而内部循环迭代新战舰的所有位置（存储在数组locations中），并检查这些位置是否被游戏板中已有的战舰占据。

locations是一个数组，存储了我们要放置到游戏板中的新战舰的位置。

```
collision: function(locations) {  
  for (var i = 0; i < this.numShips; i++) {  
    var ship = model.ships[i];  
    for (var j = 0; j < locations.length; j++) {  
      if (ship.locations.indexOf(locations[j]) >= 0) {  
        return true;  
      }  
    }  
  }  
  return false;  
}
```

迭代游戏板中已有的每艘战舰……

检查新战舰的locations数组中的位置是否包含在既有战舰的locations数组中。

为检查位置是否被既有战舰占据，我们使用了indexOf。如果返回的索引大于或等于0，就说明这个位置已被既有战舰占据，因此返回true（这意味着发生了碰撞）。

从内部循环返回，这将立即终止两个循环，退出函数并返回true。

如果执行到了这里，就说明我们检查的所有位置都未被占据，因此返回false（这意味着没有发生碰撞）。



上述代码包含两个循环：外部循环和内部循环。前者迭代模型中的所有战舰，而后者迭代要检查冲突的所有位置。在外部循环中，我们使用了循环变量i，而在内部循环中，我们使用了循环变量j。为何要使用两个不同的循环变量呢？

最后两项修改

为战舰生成随机位置的代码都编写好了，余下的唯一工作就是整合它们。完成最后两项代码修改后，就可对这款新战舰游戏进行试驾了！

```
var model = {
  boardSize: 7,
  numShips: 3,
  shipLength: 3,
  shipsSunk: 0,
  ships: [ { locations: ["06", "16", "26"], hits: ["", "", ""] },
            { locations: ["24", "34", "44"], hits: ["", "", ""] },
            { locations: ["10", "11", "12"], hits: ["", "", ""] } ],
  ships: [ { locations: [0, 0, 0], hits: ["", "", ""] },
            { locations: [0, 0, 0], hits: ["", "", ""] },
            { locations: [0, 0, 0], hits: ["", "", ""] } ],
  fire: function(guess) { ... },
  isSunk: function(ship) { ... },
  generateShipLocations: function() { ... },
  generateShip: function() { ... },
  collision: function(locations) { ... }
};

function init() {
  var fireButton = document.getElementById("fireButton");
  fireButton.onclick = handleFireButton;
  var guessInput = document.getElementById("guessInput");
  guessInput.onkeypress = handleKeyPress;

  model.generateShipLocations();
}
```

删除以硬编码方式指定战舰位置的代码。

将位置数组的元素都初始化为0。

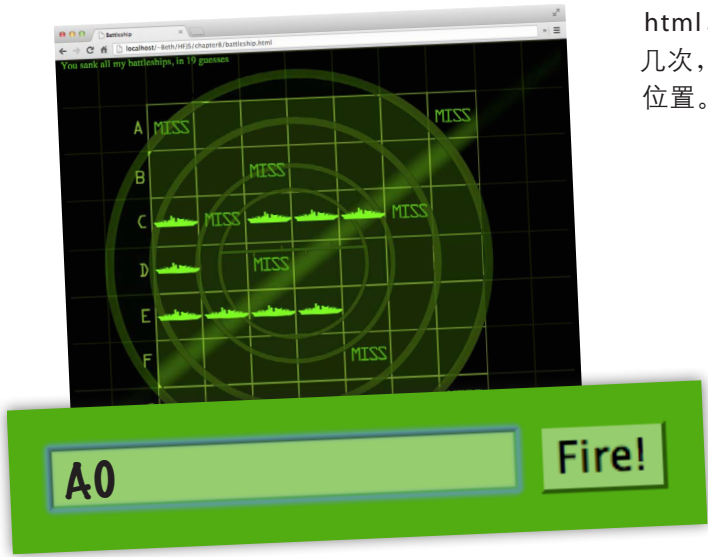
当然，需要调用生成战舰位置的函数。这个函数将修改模型中的空数组。

我们在函数init中调用model.generateShipLocations，以便在网页加载完毕后修改这些数组。这样，当你开始玩游戏时，所有战舰的位置已准备就绪。

别忘了，你可前往<http://wickedlysmart.com/hfjs>下载这款战舰游戏的完整代码。



在最后一次试驾中，测试的是真正的游戏，其中的战舰位置是随机的。将所有代码都添加到 battleship.js 中，在浏览器中重新加载 battleship.html，并开始游戏！请进行详尽的测试：多玩几次，每次都重新加载网页，以生成不同的战舰位置。



如何作弊

要作弊，可打开开发控制台，输入 `model.ships` 并按回车。你将看到三个战舰对象，它们都包含数组 `locations` 和 `hits`。这样你就有了内幕消息，知道战舰都位于游戏板的什么地方。不过，这种内部消息可不是我们透露出来的！

```
JavaScript控制台
> model.ships
[ Object,      Object,      Object ]
  hits: Array[3], hits: Array[3], hits: Array[3]
  locations: Array[3], locations: Array[3], locations: Array[3]
    0: "63"      0: "20"      0: "60"
    1: "64"      1: "21"      1: "61"
    2: "65"      2: "22"      2: "62"
```

每次都能打败计算机。



祝贺你，该创业了

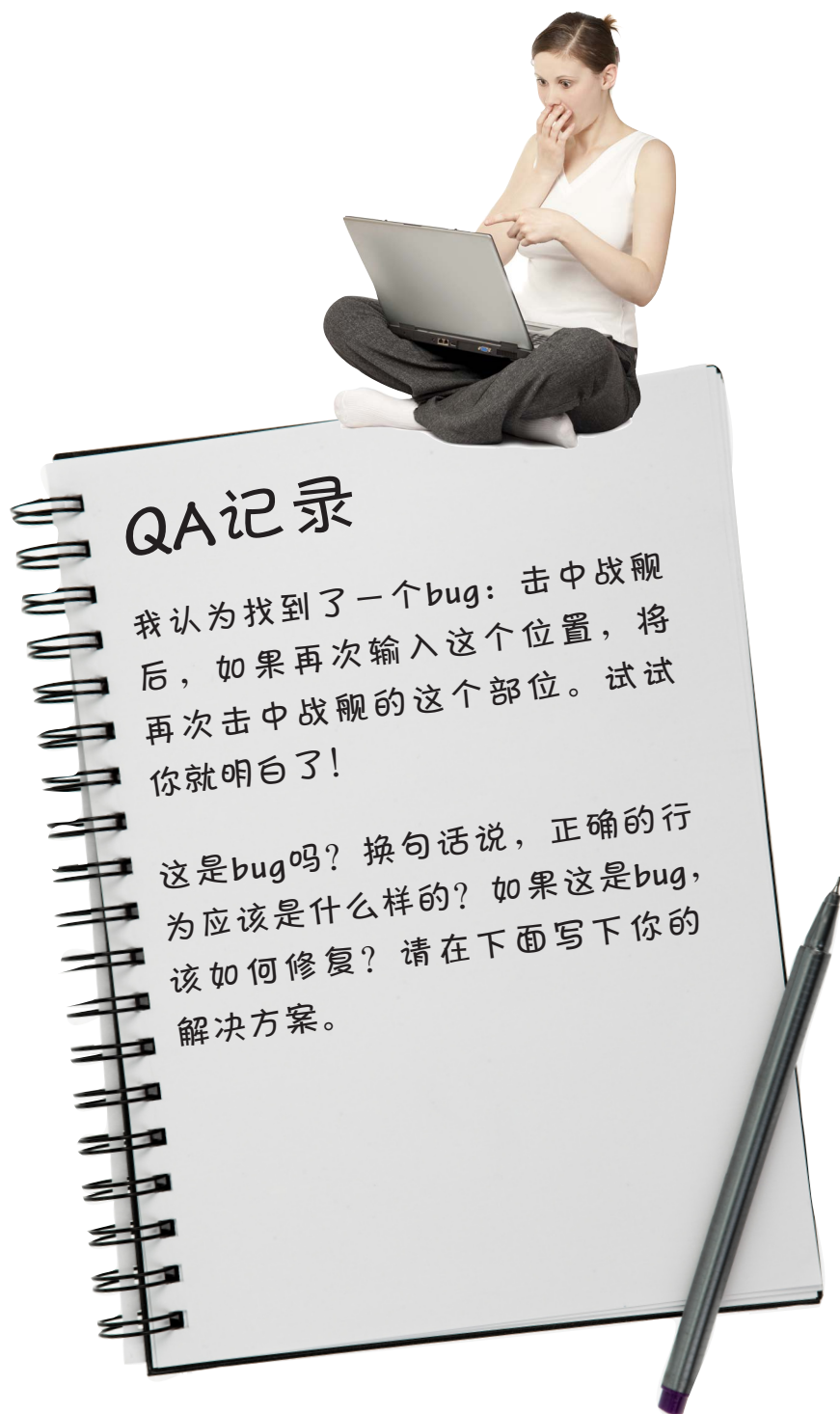
你创建了一个出色的Web应用程序，它包含大约150行代码，还有一些HTML和CSS。我们说过，这些代码都是你的；现在只要制作一份漂亮的商业计划书，你就能拿到风险投资了。有这样的好事，谁愿意错过呢？

经过一番艰苦努力，你该放松一下，玩几盘战舰游戏了。很是引人入胜，不是吗？

不过，这才刚刚开始，还有一些强大的JavaScript功能需要学习。它们能让你编写的应用程序与使用其他语言编写的应用程序媲美。

在本章中，你编写了大量的代码，现在美美地吃一顿，好好地休息一下，让这些知识都沉入脑海中。不过在此之前，还有一些要点需要复习，还有一个填字游戏需要完成。千万不要跳过它们，重复是充分理解知识的良方！





QA记录

我认为找到了一个bug：击中战舰后，如果再次输入这个位置，将再次击中战舰的这个部位。试试你就明白了！

这是bug吗？换句话说，正确的行为应该是什么样的？如果这是bug，该如何修复？请在下面写下你的解决方案。



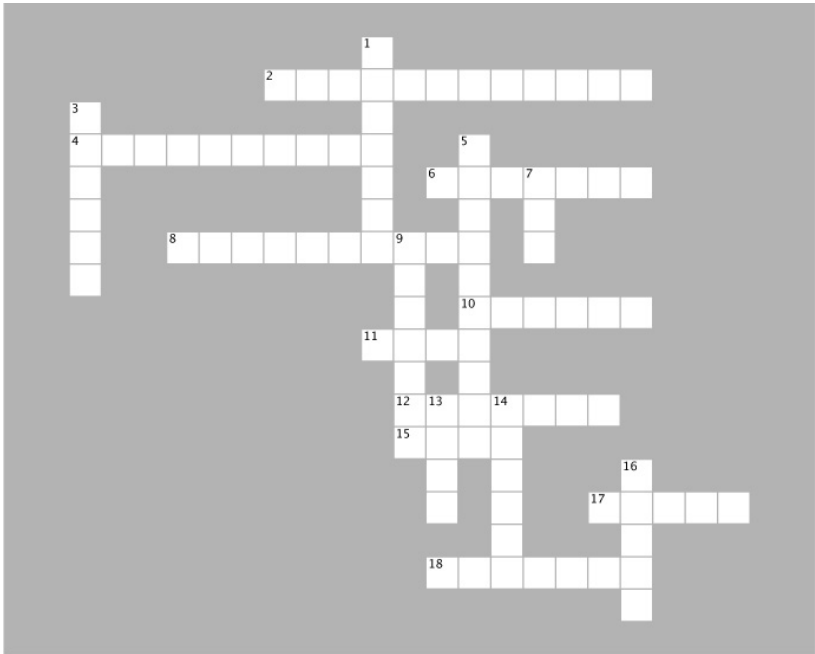
要点

- 我们使用HTML来定义战舰游戏的结构，使用CSS来设置其样式，并使用JavaScript定义其行为。
- 表格中每个<td>元素的id都被用来更新该元素的背景图像，以指出是否击中了战舰。
- 表单包含一个类型为button的输入(input)元素。我们给这个按钮指定了一个**事件处理程序**，以便玩家输入猜测后，我们能够知道这一点。
- 要获取表单中输入文本元素中的值，可使用其value属性。
- 使用CSS定位可精确地指定元素在网页中的位置。
- 我们使用了三个对象来组织代码：model、view和controller。
- 游戏中的每个对象都有一项**主要职责**。
- model的职责是存储游戏的状态以及实现状态修改逻辑。
- view的职责是在model存储的状态发生变化时更新界面。
- controller的职责是将游戏的各个部分整合起来，将玩家的猜测交给model以更新状态以及判断游戏是否结束了。
- 通过将游戏设计为**各司其职**的对象，可独立地创建和测试游戏的每个部分。
- 为简化创建和测试model的工作，我们一开始以硬编码的方式指定战舰的位置。确定model没有问题后，我们将这些硬编码位置替换为代码生成的随机位置。
- 在model中，我们定义了numShips和shipLength等属性，避免了在方法中使用以后可能需要修改的硬编码值。
- 数组有一个indexOf方法，它类似于字符串的indexOf方法。数组的indexOf方法将一个值作为参数，并返回这个值在数组中的索引；如果这个值不包含在数组中，就返回-1。
- 使用**串接**，可利用句点运算符将对象引用连接起来，从而将多条语句合而为一，并避免使用临时变量。
- do while循环类似于while循环，只是先执行一次循环体中的语句，再检查条件。
- **质量保证(QA)**是代码开发过程的重要组成部分。它不仅要求测试输入有效时的情形，还要求测试输入无效时的情形。



JavaScript填字游戏

本章的编码难题让你的大脑备受煎熬。请完成下面的填字游戏，将这种煎熬推向高潮。



横向

- 2. 我们使用哪个方法来设置元素的class特性？
- 4. 为在游戏板上显示战舰或MISS图像，我们将这幅图像放在<td>元素的什么地方？
- 6. 哪种循环的循环体中的语句至少会执行一次？
- 8. 时髦的交互式网页使用HTML、CSS和什么？
- 10. 我们使用什么来表示游戏中的每艘战舰？
- 11. <td>元素的id对应于游戏板的一个什么？
- 12. 函数collision的职责是确保战舰不会彼此_____。
- 15. 为让模型根据玩家猜测来更新游戏的状态，我们调用哪个方法？
- 17. 谁负责存储状态？
- 18. 你可使用什么来作弊，以获取战舰的位置？

纵向

- 1. 为获取表单输入元素中的猜测位置，我们给这个元素添加了一个单击事件_____。
- 3. 串接用于指向什么的引用？
- 5. 谁擅长将程序的各个部分整合起来？
- 7. 为在游戏板中显示战舰图像，我们将相应<td>元素的class特性设置为什么？
- 9. 本章的游戏包含三个对象：model、controller和什么？
- 14. 13是哪个键的键码？
- 16. 谁在其状态发生变化时通知视图？



实弹演习答案

稍后你将学习如何使用JavaScript在游戏板中添加MISS和战舰图像，但这样做之前，你需要在HTML模拟器中进行练习。我们创建了两个供你练习的CSS类，请将这些规则添加到CSS中，再假设战舰隐藏在下面的位置：

战舰 1: A6, B6, C6

战舰 2: C4, D4, E4

战舰 3: B0, B1, B2

玩家依次输入了下面的猜测：

A0, D4, F5, B2, C5, C6

你需要让表格中相应的<td>元素归属于下面两个类之一，从而在正确的网格单元格中显示MISS或战舰图像：

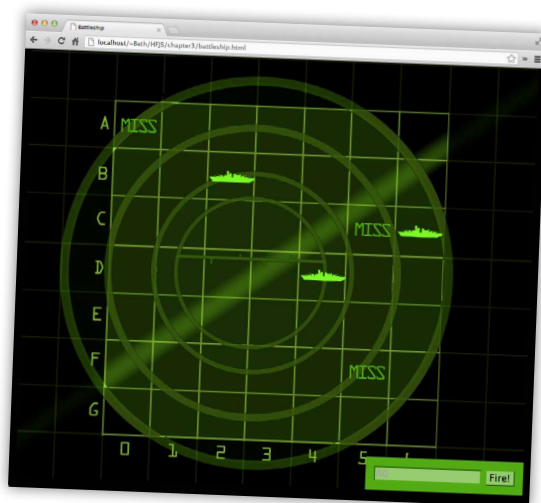
```
.hit {
    background: transparent url("ship.png") no-repeat center center;
}
.miss {
    background: transparent url("miss.png") no-repeat center center;
}
```

答案如下：你应让id为00、25和55的<td>元素归属于miss类，让id为12、26和34的<td>元素归属于hit类。要指定元素归属的类，可像下面这样设置class特性：

```
<td class="miss" id="55">
```

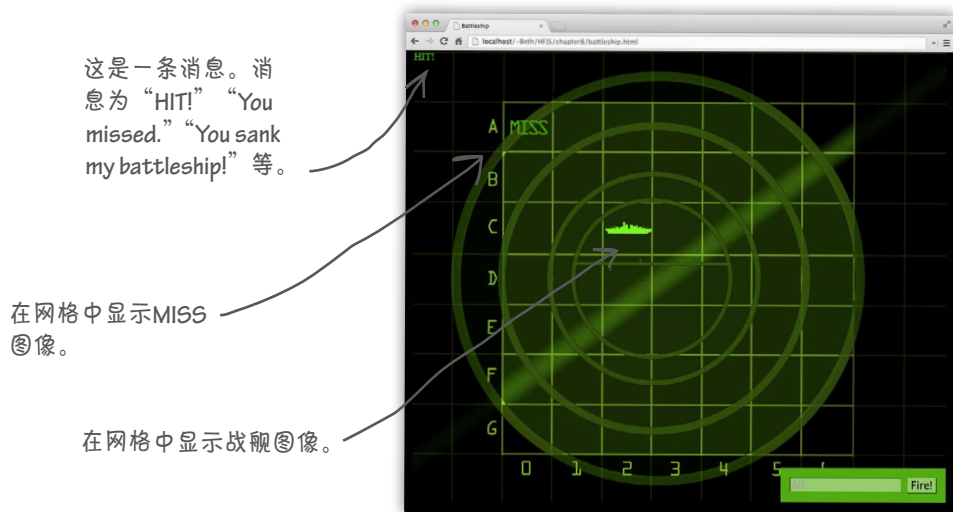
给正确的元素指定其所属的类后，游戏板应类似于这样。

务必下载这个练习所需的一切，包括下面两幅图像。





来做一些对象设计工作吧。我们将从view对象着手。前面说过，view对象负责更新视图。根据下面的视图，你能确定view对象需要实现哪些方法吗？请在下面写出这些方法的声明（这里只考虑声明，方法体的代码稍后再考虑），并使用一两条注释指出每个方法的功能。我们为你编写了一个方法的声明。答案如下。



`var view = {` 定义一个对象，并将其赋给变量view。

// 这个方法将一个字符串作为参数，并在消息区域中显示它

`displayMessage: function(msg) {`

// 稍后将编写的代码

`},`

`displayHit: function(location) {`

// 这个方法的代码

`},`

`displayMiss: function(location) {`

// 这个方法的代码

`}`

`};`

在这里声明你认为需要提供的方法！



根据前面对新游戏板的描述，你将如何在模型中表示战舰呢？（只考虑它们的位置，击中的部位以后再说。）请从下面选择最佳的解决方案。

- ☐ 像第2章那样处理战舰，使用9个变量来存储战舰占据的位置。
- ☐ 使用三个数组，每个数组都包含3个元素，指出了一艘战舰占据的位置。
- ☒ 使用包含三个位置属性的对象ship，并将所有的战舰都存储在数组ships中。
- ☐ _____
- ☐ _____
- ☐ _____

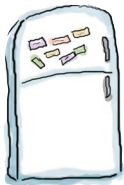
你也可以在这里列出你的解决方案。

这些解决方案都可行！（事实上，在寻找最佳解决方案的过程中，我们尝试过所有这些解决方案。）但本章使用的是这个解决方案。



JavaScript 填字游戏 答案





战舰冰箱贴答案

假设玩家向下面的位置开火。请根据表示战舰的数据结构，将战舰和MISS冰箱贴放置到游戏板的正确位置。玩家击沉了所有的战舰吗？我们指出了玩家第一次开火后的效果。

玩家向下面的位置开火：

A6, B3, C4, D1, B0, D4, F0, A1, C6, B1, B2, E4, B6

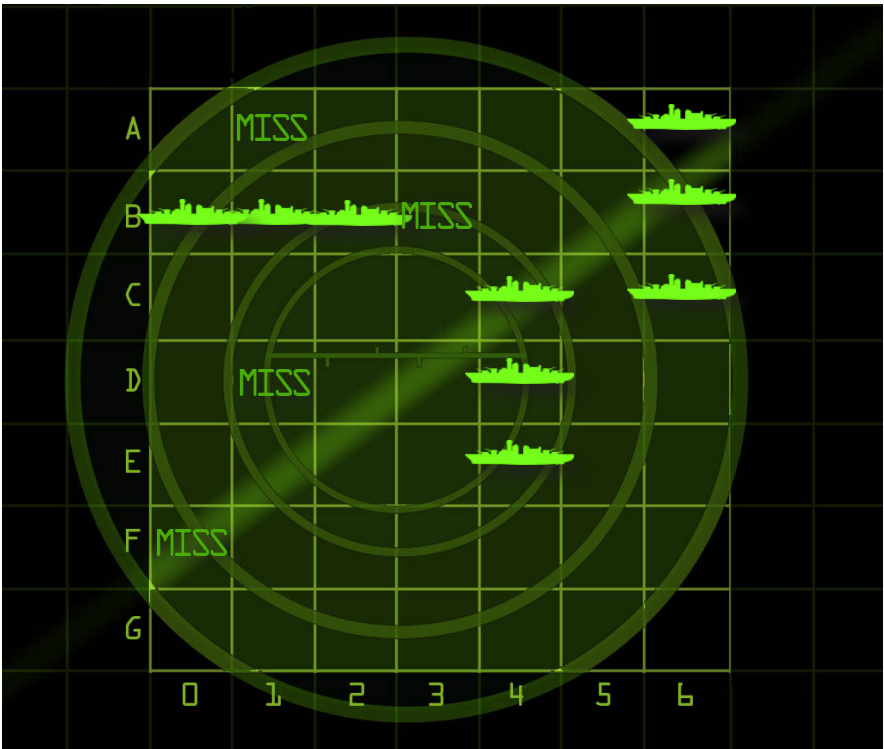
在游戏板上指出玩家这样开火的效果。

答案如下：

```
var ships = [{ locations: ["06", "16", "26"], hits: ["hit", "hit", "hit"] },
  { locations: ["24", "34", "44"], hits: ["hit", "hit", "hit"] },
  { locations: ["10", "11", "12"], hits: ["hit", "hit", "hit"] }];
```

三艘战舰都被击沉！

这是游戏板和冰箱贴。



余下的冰箱贴。



磨笔上阵 答案

下面练习使用数据结构ships来模拟开火的效果。假设数据结构ships如下，请据此回答下面的问题并补全代码。这是游戏工作原理的重要组成部分，请务必查看本章末尾的答案，再继续往下阅读。

```
var ships = [{ locations: ["31", "41", "51"], hits: ["", "", ""] },
              { locations: ["14", "24", "34"], hits: ["", "hit", ""] },
              { locations: ["00", "01", "02"], hits: ["hit", "", ""] }];
```

哪些战舰已被击中？战舰2和战舰3 被击中的是哪些部位？C4、A0

假设玩家向D4处开火，会击中战舰吗？会 如果会，击中的是哪艘？战舰2

假设玩家向B3处开火，会击中战舰吗？不会 如果会，击中的是哪艘？

请补全下面的代码，访问第二艘战舰的中间部位，并使用console.log打印其值：

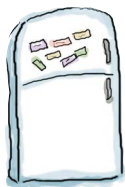
```
var ship2 = ships[_ 1 _];
var locations = ship2.locations;
console.log("Location is " + locations[_ 1 _]);
```

请补全下面的代码，检查第三艘战舰的第一个部位是否被击中：

```
var ship3 = ships[_ 2 _];
var hits = ship3._ hits _;
if (hits[_ 0 _] === "hit") {
    console.log("Ouch, hit on third ship at location one");
}
```

请补全下面的代码，指出第一艘战舰的第三个部位被击中：

```
var ship1 = ships[0];
var hits = ship1._ hits _;
hits[_ 2 _] = "hit";
```



代码冰箱贴答案

下面是一系列散乱的冰箱贴，请将它们按照正确的顺序放置，组成一个生成战舰位置的算法。答案如下。

执行循环，循环次数
等于要创建的战舰数。

为新战舰生成随机方向
(垂直或水平)。

为新战舰生成随机位置。

检查新战舰的位置是否
与既有战舰的位置冲突。

将新战舰加入到ships数组中。

9 异步编码

处理事件^{✦✦}



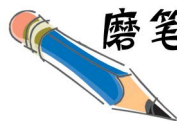
阅读完本章，你的编程方式再不似从前^①。到目前为止，你编写的代码通常都是按从上到下的顺序执行的。虽然你的代码要复杂些，还使用了一些函数、对象和方法，但从某种程度上说，它们都是按部就班地执行的。然而，JavaScript代码通常不是这样编写的，大多数JavaScript代码都是事件响应式的。很抱歉到现在才告诉你这一点。那么都是什么样的事件呢？用户单击网页、通过网络收到数据、定时器到期、DOM发生变化等。事实上，在浏览器中，幕后始终有各种事件在不断发生。本章将反思我们的编码方式，学习如何编写响应事件的代码以及为何要这样做。

^① 原文为 “you aren’t in kansas anymore”，出自1939年电影《绿野仙踪》。——编者注

考考你的脑力

你知道浏览器是做什么的，不是吗？它获取网页及其所有内容，再渲染网页。然而，浏览器所做的远不止这些。它还做什么呢？你认为浏览器还在幕后执行下面哪些任务呢？如果你没有把握，就尽力猜一猜吧。

- | | |
|---|---|
| ☐ 知道网页已加载并显示完毕。 | ☐ 监视所有的鼠标移动。 |
| ☐ 跟踪用户在网页上执行的所有单击操作，无论单击的是按钮、链接还是其他地方。 | ☐ 监视时钟并管理定时器和定时事件。 |
| ☐ 知道用户提交了表单。 | ☐ 获取网页所需的额外数据。 |
| ☐ 知道用户按下了键盘按键。 | ☐ 跟踪用户缩放或滚动网页的操作。 |
| ☐ 知道元素获得了用户界面焦点。 | ☐ 知道Cookie已创建完毕。 |



磨笔上阵

挑选上面列出的两个事件。如果浏览器能够在这些事件发生时发出通知，你能编写哪些有趣的代码来响应这些事件？

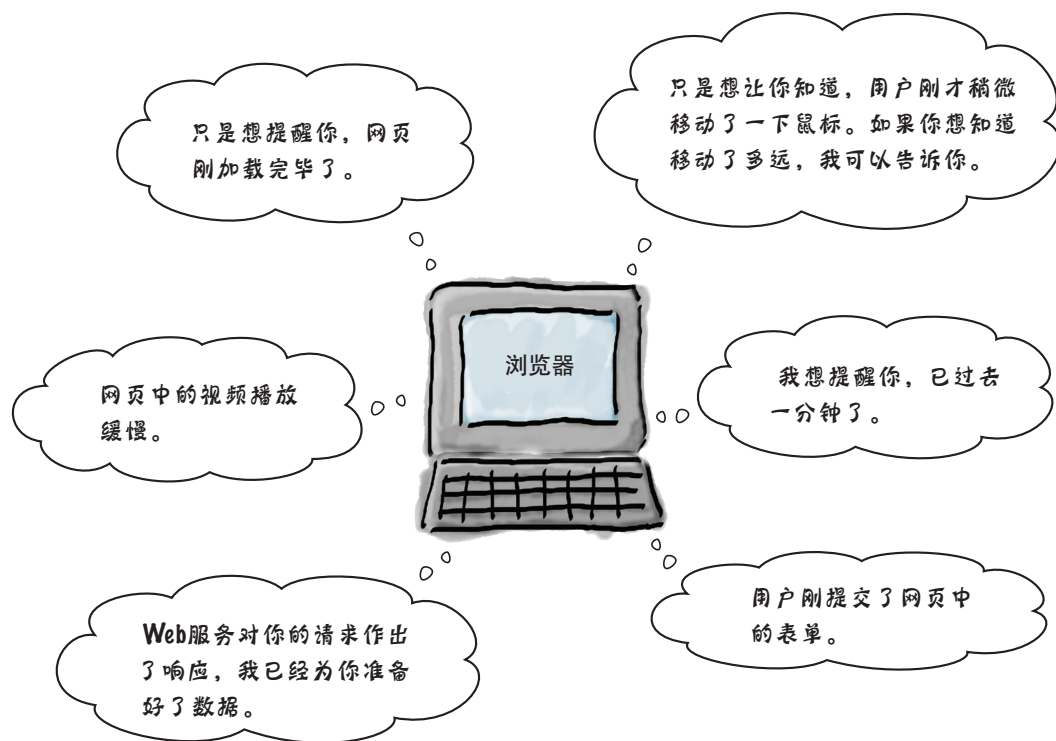
当然，你不能选择与Cookie相关的事件。

事件是什么

你现在肯定知道，浏览器不会在获取并显示页面后就甩手不管了。在幕后，有很多事情在不断发生：用户单击按钮、鼠标位置发生变化、通过网络收到数据、窗口大小发生变化、定时器到期、浏览器位置发生变化等。这些都会触发**事件**。

每当有事件发生时，你都可以在代码中**处理**它，即提供将在事件发生时调用的代码。你不必处理所有的事件，而只需处理自己关心的事件。例如，发生按钮单击事件时，你可能想在播放列表中添加一首歌曲；收到新数据时，你可能想对其进行处理并显示到网页中；定时器到期时，你可能想告诉用户，他手上的音乐会前排门票就要过期了；等等。

想了解有关浏览器定位和众多其他的高级事件，请参阅《Head First HTML5 Programming 中文版》。本书只介绍你必须知道的重要事件。



**每当有事件发生时，你都
可以在代码中处理它。**

事件处理程序是什么

我们编写处理程序来处理事件。处理程序通常是一小段代码，知道事件发生时该如何做。从代码的角度说，处理程序就是一个函数。事件发生时，其处理程序函数将被调用。

你也许会听到开发人员称其为回调函数或监听器，而非处理程序。

为让处理程序在事件发生时得以调用，你首先需要注册它。你将看到，注册的方式有多种，具体采用哪种取决于事件的类型。后面将深入探讨所有这些注册方式，现在先来看一个简单的示例。这个示例你以前见过，它就是网页加载完毕后触发的事件。



如何创建第一个事件处理程序

要理解事件，最佳方式是编写一个处理程序，并将其关联到一个事件。别忘了，你见过多个事件处理示例，其中包括网页加载事件，但我们还未全面地阐述事件处理的工作原理。网页加载事件的触发时间点是：浏览器加载完网页、显示网页的所有内容并生成了表示网页的DOM。

下面来详细介绍如何编写这种处理程序并确保在网页加载事件发生时调用它。

- 1 首先，需要编写一个处理网页加载事件的函数。在这里，这个函数将在网页加载完毕时显示“I'm alive!”。

处理程序其实就是一个函数。

```
function pageLoadedHandler() {  
    alert("I'm alive!");  
}
```

别忘了，我们通常称之为处理程序或回调函数。

这是我们的函数。我们将其命名为pageLoadedHandler，但你可以根据自己的喜好随意命名。

这个事件处理程序的功能有限，它只是显示一个提示框。

- 2 处理程序编写好后，需要建立关联，让浏览器知道有这么一个函数，应在加载事件发生时调用它。为此，我们这样使用window对象的属性onload：

```
window.onload = pageLoadedHandler;
```

为将这个处理程序关联到加载事件，我们将window对象的属性onload设置为它。

这样，网页加载事件发生时，将调用函数pageLoadedHandler。

你将看到，指定处理程序的方式随事件的类型而异。

- 3 就这些！编写这些代码后，我们就知道网页加载完毕后，浏览器将调用赋给属性window.onload的函数。

测试事件处理程序

请新建一个文件，将其命名为event.html，并添加对加载事件处理程序进行测试的代码。在浏览器中加载这个网页，并确定你看到了提示框。

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title> I'm alive! </title>
  <script>
    window.onload = pageLoadedHandler;
    function pageLoadedHandler() {
      alert("I'm alive!");
    }
  </script>
</head>
<body>
</body>
</html>
```

浏览器首先加载你的网页，开始分析其中的HTML并创建DOM。

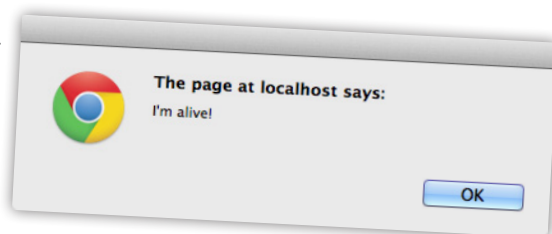
遇到脚本后，浏览器开始执行其中的代码。

当前，脚本只定义了一个函数并将其赋给属性window.onload。别忘了，这个函数将在网页加载完毕后被调用。

接下来，浏览器继续分析HTML。

分析完HTML并创建好DOM后，浏览器调用网页加载处理程序。

在这里，这个处理程序创建一个提示框，在其中显示消息“I'm alive!”。



考考你的
脑力

在不编写函数的情况下，能否指定事件处理程序？

但凡你想成为真正的JavaScript开发人员，就必须学习如何处理事件。



前面说过，到目前为止，你编写代码的方式都是线性的（linear）：你根据要解决的问题，如找出最佳的泡泡配方或生成歌曲*99 Bottles of Beer*的歌词，按从上到下的顺序逐步编写代码。

还记得战舰游戏吗？其代码的执行方式不完全是线性的：你确实编写了一些设置游戏、初始化模型等的代码，但这个游戏的主要部分并非是以线性方式执行的。每次要射击战舰时，玩家都在表单的输入元素中输入猜测，再单击Fire!按钮。这将引发一系列操作，导致游戏接着往下进行。换句话说，你编写的代码对用户输入作出响应。

以响应事件的方式组织代码是另一种代码编写方式。要以这种方式编写代码，需要考虑可能发生的事件以及代码应如何响应这些事件。在计算机科学中，通常说这种代码是异步（asynchronous）的，因为我们编写的代码仅在相应的事件发生时才会被调用。这种编码方式也改变了我们看待问题的角度，不再逐步地编写代码来实现算法，而是将处理各种事件的众多处理程序整合起来，构成一个应用程序。

通过创建一个游戏来理解事件

理解事件的最佳方式是实践，下面通过编写一个简单的游戏来获取更多的实践经验。这个游戏是这样的：你加载一个网页，它显示一幅图像。这幅图像不是普通的图像，而是一幅非常模糊的图像。你的任务是猜出图像是什么。要核实你是否猜对了，可单击图像，让它不再模糊不清。

过程如下所示。



下面先来编写HTML标记。我们将使用两幅JPG图像，其中一幅是模糊的，另一幅是清晰的。我们将它们分别命名为zeroblur.jpg和zero.jpg。HTML标记如下：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title> Image Guess </title>
  <style> body { margin: 20px; } </style>
  <script> </script>
</head>
<body>
  
</body>
</html>
```

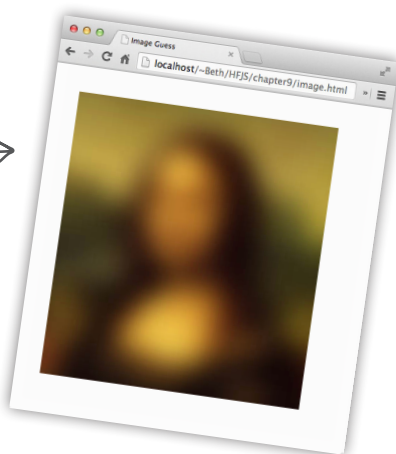
这些HTML代码非常简单，其中有一个准备就绪的<script>元素，在其中添加JavaScript代码。出于简化考虑，我们不打算使用独立的JavaScript文件，而是直接在这里添加脚本。你将看到，实现这个游戏需要的代码很少。

这是网页默认显示的模糊图像。我们把它id指定为zero，稍后你将看到如何使用这个id。

实现游戏

在浏览器中加载这些标记，你将看到模糊的图像。为实现这个游戏，需要在用户单击这幅图像时作出响应，将图像的清晰版本显示出来。

好在每当网页中的HTML元素被单击（在移动设备上触摸）时，都将触发一个事件。你只需为这个事件创建一个处理程序，并在其中编写显示图像清晰版本的代码，如下所示。



- ❶ 访问DOM中的这个图像对象，并将其属性onclick设置为一个处理程序。
- ❷ 在这个处理程序中编写代码，将图像的src属性从模糊图像改为清晰图像。

下面就来按步骤编写代码。

第1步：访问DOM中的图像

访问图像对你来说已经不新鲜了，只需使用老朋友方法 `getElementById` 就能获得指向它的引用。

```
var image = document.getElementById("zero");
```

← 获取指向图像元素的引用，并将其赋给变量 `image`。

哦，还需确保这些代码在网页的DOM创建好之后再运行；为此，可使用 `window` 对象的属性 `onload`。我们将这些代码放在函数 `init` 中，并将属性 `onload` 设置为这个函数。

← 别忘了，只有网页加载完毕后，我们才能从DOM中获取这幅图像。

```
window.onload = init;
function init() {
    var image = document.getElementById("zero");
}
```

← 我们创建函数 `init`，并将其赋给属性 `onload`，从而确保这些代码在网页加载完毕后才执行。

← 在函数 `init` 中，我们获取一个引用，它指向id为 `zero` 的图像元素。

← 别忘了，在JavaScript中，定义函数的顺序无关紧要。因此可先将 `init` 赋给属性 `onload`，再定义它。

第2步：添加更新图像的处理程序

要添加一个响应图像单击事件的处理程序，只需将一个函数赋给图像的属性`onclick`。下面将这个函数命名为`showAnswer`并定义它。

```

window.onload = init;
function init() {
    var image = document.getElementById("zero");
    image.onclick = showAnswer; ← 将一个处理程序赋给从DOM中获取的图像对象的onclick属性。
}

```

接下来需要编写函数`showAnswer`，它将图像元素的`src`属性改为清晰的图像，让图像变清晰：

```

function showAnswer() {
    var image = document.getElementById("zero");
    image.src = "zero.jpg";
}

```

首先，必须再次从DOM获取该图像元素。

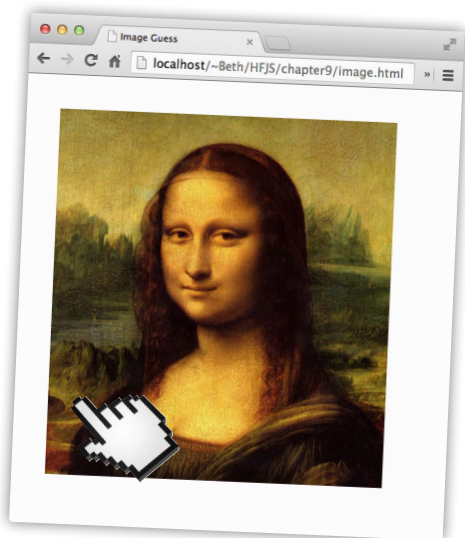
别忘了，模糊版名为`zeroblur.jpg`，而清晰版名为`zero.jpg`。

获得图像元素后，就可修改它了——将其`src`属性设置为清晰的图像。

试驾

来测试一下这个简单的游戏。将所有的HTML、CSS和JavaScript都输入一个文件中，并命名为`image.html`。再从<http://wickedlysmart.com/hfjs>下载所需的图像，并将它们存储到`image.html`所在的文件夹。然后，在浏览器中加载这个HTML文件。

单击图像的任何地方都将导致处理程序`showAnswer`被调用，它修改这个图像元素的`src`，揭晓答案。





等等，为何需要在showAnswer中再次调用getElementById呢？我不确定这里的执行流程是什么样的。

啊哈，代码包含大量事件处理程序时，有时很难搞清楚其执行流程。别忘了，函数init在网页加载完毕后被调用，但函数showAnswer要等到用户单击图像时才被调用。因此，这两个事件处理程序被调用的时间不同。

另外，别忘了作用域规则。在函数init中，我们将getElementById返回的对象存储在局部变量image中。这意味着函数结束时，这个变量不在作用域内，进而被销毁。因此，等函数showAnswer被调用时，我们必须再次从DOM获取这个图像对象。诚然，我们可以将这个对象存储在一个全局变量中，但滥用全局变量可能导致代码难以理解，还容易出现bug。这正是我们要避免的。

世上没有愚蠢的问题

问： 设置图像的属性src与使用setAttribute设置特性src相同吗？

答： 就这里而言，效果是相同的。使用getElementById从DOM获取HTML元素时，获得的是包含多个方法和属性的元素对象。所有的元素对象都包含属性id，其值为相应HTML元素的id（如果在HTML中设置了）。表示图像的元素对象还有一个src属性，其值为相应元素的src特性指定的图像文件。

然而，并非所有HTML特性都有对应的对象属性，因此对于这些特性，必须使用setAttribute和getAttribute来设置和获取。就src和id而言，要设置和获取，可使用元素对象的相应属性，也可使用getAttribute和setAttribute，而且这两种方法的效果相同。

问： 前面在一个处理程序中调用了另一个处理程序？

答： 其实并非如此。网页加载完毕后，将调用加载事件处理程序。在这个处理程序中，我们将一个处理程序赋给了图像的属性onclick，但要等到用户单击图像时才会调用它。用户单击图像时（这可能是在网页加载完毕的很久之后），才会调用单击事件处理程序showAnswer。因此，这两个处理程序是在不同的时间调用的。

变身浏览器



下面是猜图游戏的代码，你的任务是变身浏览器，搞清楚每个事件发生后该如何做。完成这个练习后，请查看本章末尾的答案，看看你是否都明白了。我们已为你完成了第一部分。

```
window.onload = init;
function init() {
    var image = document.getElementById("zero");
    image.onclick = showAnswer;
}

function showAnswer() {
    var image = document.getElementById("zero");
    image.src = "zero.jpg";
}
```

↑ 这是你要执行的代码……

网页加载时……	定义函数init和showAnswer
网页加载事件发生时……	
图像单击事件发生时……	

↑ 在这里列出你的答案。

考考你的 脑力

如果网页包含多幅图像，要求单击每幅图像时都显示清晰版，该如何用代码来处理这种情形呢？实现这种目标的傻办法是什么？能否对既有代码进行少量修改来实现这个目标呢？



Judy：大家好！到目前为止，猜图游戏运行得很好，但我们应对其进行扩展，在网页中包含更多图像。

Jim：确实如此，我也是这样想的。

Joe：我准备了很多图像，就差编写代码了。前面将两幅图像命名为zero.jpg和zeroblur.jpg。我给新增的图像命名时遵循了这种约定，将它们命名成了one.jpg和oneblur.jpg等。

Jim：需要给每幅图像都编写一个新的单击事件处理程序吗？如果这样做，将有大量重复的代码。毕竟，每个事件处理程序所做的事情都完全相同：将模糊版替换为清晰版，不是吗？

Joe：确实如此，但我不知道如何将同一个事件处理程序用于多幅图像。这可能吗？

Judy：我们可将同一个处理程序（就是同一个函数）赋给游戏中每幅图像的属性onclick。

Joe：这样做的话，用户单击任何图像时，都将调用同一个函数？

Judy：没错。我们将把showAnswer用作每幅图像的单击事件处理程序。

Jim：但我们如何知道要让哪幅图像变清晰呢？

Joe：你的意思是说单击事件处理程序不知道这一点？

Jim：它怎么知道？当前，函数showAnswer假定用户单击的是id为零的图像。如果对于每幅图像的单击事件都调用showAnswer，就必须让其代码适用于任何图像。

Joe：哦，没错。那么我们如何确定用户单击的是哪幅图像呢？

Judy：实际上，我一直在仔细研究事件，我想单击事件处理程序有办法确定用户单击的是哪个元素。现在暂时不管这一点，先在游戏中添加一些图像，看看如何为它们指定相同的事件处理程序，再考虑如何确定用户单击的是哪幅图像。

Joe和Jim：有道理！

再添加一些图像

我们有一系列新图像，先来将它们添加到网页中。我们再添加5幅图像，这样网页将总共包含6幅图像。我们还将修改CSS，让图像之间有一定的间距。

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title> Image Guess </title>
  <style>
    body { margin: 20px; }
    img { margin: 20px; }
  </style>
  <script>
    window.onload = init;
    function init() {
      var image = document.getElementById("zero");
      image.onclick = showAnswer;
    }
    function showAnswer() {
      var image = document.getElementById("zero");
      image.src = "zero.jpg";
    }
  </script>
</head>
<body>
  
  
  
  
  
  
</body>
</html>
```

使用这个CSS属性将图像的外边距指定为20像素。

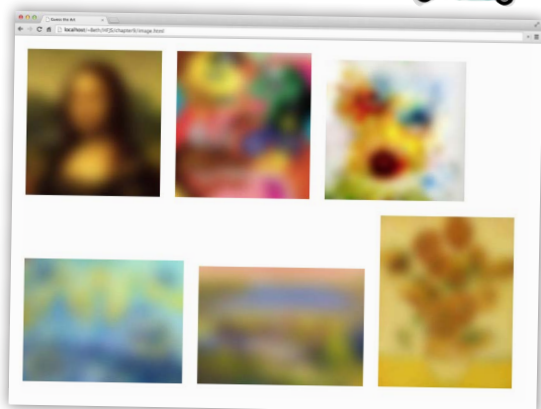
在这里又添加了5幅图像。注意到对于每幅图像，都使用了相同的id、src和文件命名方案。稍后你将看到这样做的好处。



获取图像

从<http://wickedlysmart.com/hfjs>下载压缩文件，可在文件夹chapter9中找到所有这些图像。

如果你对这个网页进行测试，结果将类似于这样。



将同一个事件处理程序赋给每幅图像的 属性onclick

在网页中包含更多图像后，需要做的工作也更多。当前，你可单击第一幅图像（即《蒙娜丽莎》）以显示其清晰版，但其他图像呢？

我们可以为每幅图像编写一个处理程序，但从前面的讨论可知，这样做既麻烦又浪费，如下所示：

```

window.onload = init;
function init() {
    var image0 = document.getElementById("zero");
    image0.onclick = showImageZero;
    var image1 = document.getElementById("one");
    image1.onclick = showImageOne;
    ...
}
function showImageZero() {
    var image = document.getElementById("zero");
    image.src = "zero.jpg";
}
function showImageOne() {
    var image = document.getElementById("one");
    image.src = "one.jpg";
}
...

```

← 可以获取网页中的每幅图像，并给它们指定不同的单击处理程序。如果这样做，将需要重复6次，但这里只显示了其中的两次。

← 其他四次。

← 我们还需要编写6个单击处理程序，每幅图像一个。

← 需要的其他四个处理程序。



为每幅图像编写不同的处理程序存在哪些缺点？请勾选所有符合的选项。

- | | |
|---|--|
| ☐ 每个处理程序都包含大量重复的代码。 | ☐ 如果需要修改代码，可能需要修改所有的处理程序。 |
| ☐ 需要的代码很多。 | ☐ 难以跟踪所有的图像和处理程序。 |
| ☐ 难以以通用的方式处理任意数量的图像。 | ☐ 代码更难维护。 |

如何将同一个处理程序用于所有图像

显然，为每幅图像编写不同的处理程序不是解决这个问题的好办法。那么如何将既有处理程序showAnswer用于处理所有图像的单击事件呢？为此，显然需要对showAnswer稍作修改。要将showAnswer用于所有图像，需要做如下两项工作。

- ❶ 将函数showAnswer指定为网页中每幅图像的单击事件处理程序。
- ❷ 修改函数showAnswer，使其能够显示任何图像的清晰版，而不仅仅是zero.jpg。

我们将以通用的方式完成这两项任务，使得在网页中再添加其他图像时，函数showAnswer依然适用。换句话说，如果代码编写正确，在网页中添加或删除图像后，也无需对代码作任何修改。下面就来这样做。

将这个单击处理程序赋给网页中的每幅图像

我们要克服的第一个障碍如下：在当前的代码中，我们使用方法getElementById来获取指向图像zero的引用，并将函数showAnswer赋给它的属性onclick。我们将采用一种更简单的方式，而不是使用getElementById来分别获取指向每幅图像的引用：一次性获取所有图像，再通过迭代来设置每幅图像的单击事件处理程序。为此，我们将使用一个以前没有介绍过的DOM方法：document.getElementsByTagName。这个方法将一个标签名（如img、p或div）作为参数，并返回一个列表，其中包含所有匹配的元素。下面就来使用它：

```
function init() {  
    var image = document.getElementById("zero");  
    image.onclick = showAnswer;
```

删除获取图像zero并设置其单击事件处理程序的代码。

```
    var images = document.getElementsByTagName("img");  
    for (var i = 0; i < images.length; i++) {  
        images[i].onclick = showAnswer;  
    }  
};
```

使用标签名img来获取网页中的元素。这将查找并返回网页中的所有图像。我们将返回的图像存储在变量images中。

然后，迭代images，依次将每幅图像的单击处理程序设置为showAnswer。这样就将每幅图像的属性onclick都设置成了处理程序showAnswer。



聚焦document.getElementsByTagName

方法document.getElementsByTagName的工作原理与document.getElementById很像，但不是根据id来获取元素，而是根据标签名（这里是img）来获取元素。当然，HTML中可能包含很多元素，所以这个方法可能返回很多或一个元素，也可能不返回任何元素；具体返回多少元素取决于网页包含多少幅图像。在这个猜图游戏中，网页包含6个元素，因此返回的列表中将包含6个图像对象。

我们获取的是一个element对象列表，其中的对象都与指定标签名匹配。

```
var images = document.getElementsByTagName("img");
```

返回的是类似于数组的对象列表。它类似于数组，但与数组又不完全相同。

注意到这里有个s，这意味着可能获取很多元素。

用引号将标签名括起，但不包含<和>!

世上没有愚蠢的问题

问：你说过，getElementsByTagName返回一个列表。这指的是数组吗？

答：实际上返回的是一个NodeList对象，但你可像处理数组一样处理它。NodeList是一个Node集合，而Node指的其实就是DOM树中的element对象。你可以像数组一样迭代这种集合：使用length属性来获取其长度，再通过用方括号括起的索引来访问NodeList的每个项目。然而，数组和NodeList的相似之处仅此而已，因此处理NodeList对象时必须小心。除非需要在DOM中添加或删除元素，否则不必对NodeList有更深入的了解。

问：可以给任何元素指定单击处理程序吗？

答：差不多是这样的。对于网页中的任何元素，只需获取它，再将其onclick属性设置为一个函数即可。正如你看到的，处理程序可能是某个元素专用的，也可将其用于很多元素。当然，不会在网页中显示出来的元素（如<script>和<head>）不支持单击等事件。

问：会给处理程序传递参数吗？

答：这个问题问得好，也问得正是时候。会给处理程序传递参数，我们马上将讨论传递给一些处理程序的事件对象。

问：除单击事件外，元素还支持其他事件吗？

答：还有很多其他的事件。实际上，你在战舰游戏中见过其中的一个：按键事件。在这个游戏中，每当用户在表单的输入元素中按回车键时，都将调用一个事件处理程序。本章后面将介绍其他几种事件。



Judy, 我们现在有一个事件处理程序 `showAnswer`, 它处理所有图像的单击事件。你是不是说过, `showAnswer` 被调用时, 你知道如何判断用户单击的是哪幅图像吗?

Judy: 我说过。每次调用单击事件处理程序时, 都将向它传递一个事件对象, 你可使用这个对象来获取有关事件的细节。

Joe: 比如用户单击的是哪幅图像?

Judy: 更准确地说是触发事件的元素, 这被称为目标 (target)

Joe: 什么是目标呢?

Judy: 我说过, 它是触发事件的元素。例如, 用户单击特定的图像时, 目标就是这幅图像。

Joe: 因此, 如果用户单击id为zero的图像, 目标将被设置为这幅图像?

Judy: 更准确地说, 是表示这幅图像的元素对象。

Joe: 又绕回来了?

Judy: 可将这个元素对象视为使用值zero调用`document.getElementById`得到的东西。它表示DOM中id为zero的图像。

Joe: 明白, 那么如何获取这个目标呢? 好像有了它就能确定用户单击的是哪幅图像。

Judy: `target`就是事件对象的一个属性。

Joe: 太好了, 这正是`showAnswer`梦寐以求的, 马上就能达成目标了。等等, 向`showAnswer`传递了一个事件对象?

Judy: 没错。

Joe: 当前, 函数`showAnswer`的代码是什么样的呢? 虽然给它传递了一个事件对象, 可它没有接受这个对象的形参!

Judy: 别忘了, JavaScript允许你省略形参。

Joe: 哦, 是这样的。

Judy: Joe, 你别忘了, 现在需要搞清楚的是, 如何将图像的src设置为清晰版图像。当前, 我们假定清晰版图像名为zero.jpg, 但这种假设不再成立。

Joe: 也许可以使用图像的id特性来确定清晰版图像的名称, 因为所有图像元素的id都是其清晰版图像的名称。

Judy: 听起来好像行得通!

事件对象的工作原理

单击事件处理程序被调用时，将向它传递一个事件对象。事实上，大多数文档对象模型（DOM）事件发生时，都会向相应的处理程序传递一个事件对象。事件对象包含一些有关事件的常规信息，如事件是哪个元素触发的以及事件是何时发生的。另外，你还将获得有关事件的特有信息。例如，用户单击鼠标时，你将获得单击位置的坐标。

除DOM事件外，还有其他的事件，你将在本章后面看到这样的示例。

下面来说明事件对象的工作原理。



那么，事件对象都包含哪些信息呢？有关事件的常规信息和具体信息。具体信息取决于事件的类型，将在稍后介绍。常规信息包括属性`target`，它存储了一个引用，指向触发事件的元素。因此，如果用户单击网页中的元素（如图像），该元素就将是`target`。我们可以像下面这样来访问它：

```
function showAnswer(eventObj) {
  var image = eventObj.target;
}
```

`target`指出了触发事件的是哪个元素。



如果你运行的是IE8或更低版本的浏览器，请参阅附录。

在IE旧版本中，需要以稍微不同的方式获取事件对象。



前面说过，表示DOM事件的事件对象包含一些属性，它们提供了有关事件的额外信息。下面是事件对象可能包含的其他一些属性，请将每个属性及其用途连接起来。

target

要确定用户单击的位置离浏览器窗口的上边缘有多远？使用我吧。

type

我存储着触发事件的对象。我可以是各种不同的对象，但通常是元素对象。

timeStamp

在触摸设备上，可使用我来确定用户使用了多少根手指来触摸屏幕。

keyCode

我是一个字符串，如"click"或"load"，指出了发生的是哪种事件。

clientX

想知道事件是何时发生的？我可向你提供这种信息的属性。

clientY

要确定用户单击的位置离浏览器窗口的左边缘有多远？使用我吧。

touches

我能告诉你用户刚按下了哪个键。

使用事件对象

对事件有了更深入的了解（具体地说是向单击处理程序传递了事件对象）后，咱们来看看如何利用事件对象中的信息让网页上相应的图像变清晰。为此，我们再来看看HTML标记：

```
<!doctype html>
...
<body>
  
  
  
  
  
  
</body>
</html>
```

这里再次列出了HTML标记。

每个图像元素都有一个id，它是相应清晰版图像的名称。因此id为zero的图像元素的清晰版图像为zero.jpg，而id为one的图像元素的清晰版图像为one.jpg，依此类推。

注意到每个图像元素的id都对应清晰版图像的名称（不包括扩展名.jpg）。如果能够获取这个id，则加上扩展名.jpg后，就能得到相应清晰版图像的名称，从而将图像元素的src属性改为清晰版图像的名称了，如下所示：

```
function showAnswer(eventObj) {
  var image = eventObj.target;

  var name = image.id;
  name = name + ".jpg";
  image.src = name;
}
```

别忘了，每当用户单击图像时，都将传入一个事件对象。

该事件对象的属性target是一个引用，指向被单击的图像元素。

因此，我们可以使用该对象的id属性来获取清晰版图像的名称。

最后，将图像元素的src属性设置为该名称。

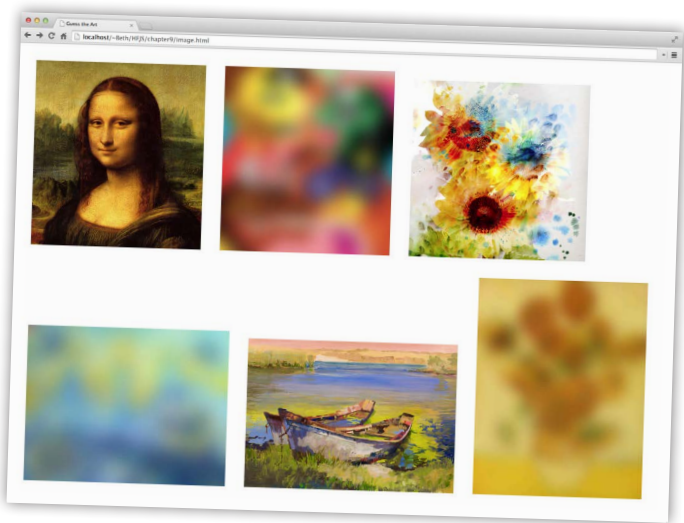
修改图像元素的src属性后，浏览器将立即获取指定的新图像，并在模糊版图像的位置显示它。

测试事件对象和target属性



请更新文件image.html的代码，再对其进行测试。猜测图像、单击它并查看显示的清晰版。想想这个程序吧，它并没有被设计成从头到尾运行的，而是完全由一系列操作组成，这些操作是用户单击图像时触发的事件引起的。另外，你使用了相同的代码来处理所有图像的单击事件。这些代码很聪明，知道用户单击的是哪幅图像。请尝试运行这个游戏。如果你单击两次，结果如何呢？是否有事情发生？

现在，单击任何图像时，都将看到其清晰版。真棒！



世上没有愚蠢的问题

问：也会给加载事件处理程序传递一个事件对象吗？

答：是的。这个事件对象包含目标（window对象）、事件发生的时间、事件类型（"load"）等信息。在加载事件处理程序中，通常很少使用事件对象，因为对这种事件来说，它没有提供什么有用的信息。你将发现，事件对象在有些情况下很有用，在其他情况下又毫无用处。这完全取决于事件类型。如果你不确定事件对象包含哪些有关事件的具体信息，请参阅JavaScript的参考手册。

考考你的脑力

如果你要在答案揭晓后，让图像几秒钟后又变模糊，该如何实现呢？



起底事件

本周访谈：浏览器说事件

Head First: 浏览器，你好，很高兴你能接受采访。我们知道你有多忙。

浏览器: 很荣幸接受你的采访。你说的没错，有各种事件需要我去管理，真是忙得不可开交。

Head First: 你是如何管理这些事件的呢？让我们也知道知道这种奇迹背后的情况。

浏览器: 你知道，事件几乎是在不间断地发生：用户移动鼠标或在移动设备上执行手势，通过网络收到数据，定时器到期……我就像纽约中心车站，需要管理的东西实在太多了。

Head First: 我还以为除了调用为事件指定的处理程序外，你需要做的其他事情不多呢。

浏览器: 即便是没有指定处理程序的事件，我也需要处理。我需要捕获事件，对其进行解读，并检查是否有处理程序等待它发生。如果有，我就必须调用处理程序。

Head First: 事件这么多，你如何跟踪它们呢？如果有很多事件同时发生，该怎么办呢？毕竟负责处理事件的人只有你一个。

浏览器: 确实，可能在很短的时间内发生很多事件，有时我忙不过来，无法实时地处理它们。有鉴于此，我把事件都放入队列，再遍历这个队列，并在必要时调用相应的处理程序。

Head First: 这像极了我做快餐店厨师的日子。

浏览器: 确实如此，如果快餐店每隔一秒左右就有一个订单的话。

Head First: 你必须按顺序一个一个地去处理？

浏览器: 是的，明白这一点对理解JavaScript至关重要：只有一个队列和一个控制线程。这意味着只有我独自去逐个处理事件。

Head First: 对学习JavaScript的读者来说，这意味着什么呢？

浏览器: 假设你编写了一个处理程序，而它需要执行大量的计算，即需要占用大量的计算时间。只要该处理程序在执行计算，我就得等待，直到它执行完毕，才能接着处理队列后面的事件。

Head First: 哇，等待代码长时间执行的情况多吗？

浏览器: 这样的情况时有发生，但如果网页或应用程序因处理程序执行时间过长而响应缓慢，Web开发人员很快就会发现。因此，只要Web开发人员知道事件队列的工作原理，这种问题就不会太常见。

Head First: 我们的读者现在都知道事件队列的工作原理了！回到事件，有很多不同类型的事件吗？

浏览器: 很多。有基于网络的事件、定时器事件、与网页相关的DOM事件等。有些事件（如DOM事件）会生成事件对象，其中包含很多有关事件的细节。例如，鼠标单击事件包含有关单击位置的信息，按键事件包含有关按下的是哪个键的信息，等等。

Head First: 你的很多时间都花在处理事件上，这是你利用时间的最佳方式吗？毕竟，你还需负责检索、分析和渲染网页等工作。

浏览器: 花时间来处理事件非常重要。这年头，你必须编写代码来让网页与用户互动并引人入胜，为此必须使用事件。

Head First: 确实如此，简单网页的时代一去不复返了。

浏览器: 太对了。噢，天啊，事件队列就要溢出了，我得赶紧走！

Head First: 好的，下次再会！

事件和队列

你知道，浏览器维护着一个事件队列。在幕后，浏览器不断地从这个队列中取出事件，并调用相应的事件处理程序来处理它们——如果有的话。



浏览器逐个处理这些事件，明白这一点很重要。因此，应尽可能让处理程序简短而高效，否则事件队列可能包含大量等待处理的事件，导致浏览器无法及时地处理它们。这会带来什么糟糕的后果呢？界面的响应速度可能非常缓慢。

如果情况变得极其糟糕，将出现一个对话框，指出脚本运行缓慢。这表明浏览器承认它已无能为力！



海盗船长得到了一张藏宝图，需要你帮忙找出宝藏的坐标。为此，你将编写一些代码，使在地图上移动鼠标时显示其坐标。下一页提供了部分代码，但需要你帮忙补全。

这是地图，其中X是宝藏所处的位置！

等你补全代码后，将鼠标指向X时，将看到宝藏所处位置的坐标。



补全后的代码将在地图下方显示坐标。

COORDINATES: 10, 20

备注：强烈建议你完成这个练习，因为如果你不让海盗知道宝藏的坐标，他们会很生气，后果很严重。哦，为补全这些代码，你需要具备如下知识。



mousemove事件

当鼠标在特定元素上移动时，mousemove事件通知相应的处理程序；要指定处理程序，可使用元素的属性onmousemove。这样会给这种事件处理程序传递一个event对象，其中包含如下属性。

clientX和clientY: 鼠标相对于浏览器窗口左边缘和上边缘的距离，单位为像素。

screenX和screenY: 鼠标相对于设备屏幕左边缘和上边缘的距离，单位为像素。

pageX和pageY: 鼠标相对于网页左边缘和上边缘的距离，单位为像素。



下面就是所需的代码。它们在网页中包含了地图，并创建了一个用于显示坐标的段落元素。你需要补全处理事件的代码。祝你好运，我们可不想看到你葬身海底！



```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Pirates Bounty</title>
  <script>
    window.onload = init;
    function init() {
      var map = document.getElementById("map");
      _____
    }

    function showCoords(eventObj) {
      var map = document.getElementById("coords");
      _____
      _____
      map.innerHTML = "Map coordinates: "
                    + x + ", " + y;
    }
  </script>
</head>
<body>
  
  <p id="coords">Move mouse to find coordinates...</p>
</body>
</html>
```

↑ 在这里指定处理程序。

↖ 在这里获取坐标。

补全代码后，加载这个网页，
并将宝藏的坐标记录在这里。



其他事件

到目前为止，我们介绍了三种事件：加载事件（浏览器加载网页完毕时发生）、单击事件（用户单击网页元素时发生）和鼠标移动事件（用户在元素上移动鼠标时发生）。你可能会遇到很多其他的事件，如与通过网络收到数据相关的事件、浏览器地理定位事件、基于时间的事件等。

对于前面介绍过的所有事件，为将它们关联到处理程序，你总是将处理程序赋给某个属性，如`onload`、`onmousemove`或`onclick`。但这种做法并非适用于所有事件。例如，对于基于时间的事件，不是将处理程序赋给属性，而是调用函数`setTimeout`并向它传递处理程序。

来看一个示例：假设你要在5秒后执行某种操作，可像下面这样使用`setTimeout`和处理程序。



首先，编写一个事件处理程序。这个处理程序将在时间事件发生时被调用。

```
function timerHandler() {
    alert("Hey what are you doing just sitting there staring at a blank screen?");
}
```

这个事件处理程序只是显示一个提示框。

```
setTimeout(timerHandler, 5000);
```

在这里，我们调用`setTimeout`。它接受两个参数：事件处理程序和时间间隔（单位为毫秒）。

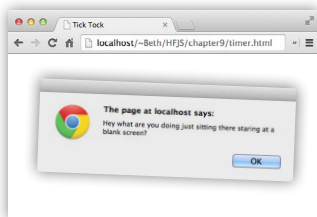
在这里，我们让定时器等待5000毫秒（5秒）。然后调用处理程序`timerHandler`。

`setTimeout`的用法与设置秒表类似。

测试定时器

别袖手旁观，该动手测试这些代码了！将这些代码放在一个简单的HTML网页中，再加载这个网页。一开始你什么都看不到，但5秒后你将看到提示框。

请耐心等待，5秒后你将看到这里所示的提示框。如果几分钟后还没有看到提示框，你也许该踢计算机一脚。开个玩笑而已，正确的做法是检查你的代码。



考考你的
脑力

如何让提示框没完没了地每隔5秒显示一次呢？

setTimeout的工作原理

下面详细研究刚才发生的情况。

1 在网页加载过程中，我们做了两件事情：定义了一个名为timerHandler的处理程序；调用setTimeout创建了一个将在5000毫秒后发生的时间事件。这个事件发生时，将执行处理程序timerHandler。

2 在定时器以毫秒为单位进行倒计时的同时，浏览器继续完成其常规工作。

3 倒计时到零后，浏览器调用处理程序。

4 处理程序被调用，它创建一个提示框并在浏览器中显示出来。

```
function timerHandler() {  
    alert("Hey what are you doing just sitting there staring at a blank screen?");  
}
```

浏览器执行事件处理程序，你看到了这个提示框。

好了，该开始了。我有一个定时器，它将在5000毫秒后到期，届时我必须调用一个处理程序。

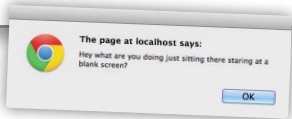
定时器由浏览器管理。

浏览器跟踪所有的定时器（可以同时有多个定时器）及其需要调用的处理程序。

5000, 4999, 4998...
..., 6, 5, 4, 3, 2, 1, 0.

5000毫秒过去，定时器已到期，该调用处理程序了。

已调用处理程序，对定时器的处理到此结束。





使用setTimeout时，向一个函数传递了另一个函数，我这样理解对吗？

好眼力！你还记得本章前面说过再不似从前的话吗？在电影《绿野仙踪》中，这句台词（you aren't in Kansas anymore）过后，画面就从黑白变成了彩色的。回到你的问题，我们定义了一个函数，再将它传递给setTimeout（它实际上是个方法）。

```
setTimeout(timerHandler, 5000);
```

在这里，将一个指向函数的引用传递给了setTimeout（另一个函数）。

我们为何这样做呢？这样做意味着什么呢？从本质上说，函数setTimeout创建一个倒计时的定时器，并将其关联到一个处理程序。当定时器倒计时到零后，将调用这个处理程序。为告诉setTimeout应调用哪个定时器，需要向它传递一个指向处理程序函数的引用。setTimeout将这个引用存储起来，供定时器到期后使用。

如果你说“这合乎情理”，那就太好了。但是，你也可能说：“将一个函数传递给另一个函数？你脑子没进水吧？”如果你这样说，你很可能以前使用过C或Java等语言。在这些语言中，像这样将一个函数传递给另一个函数根本行不通；但在JavaScript中，这行得通。事实上，能够传递函数提供了一种强大的功能，在编写响应事件的代码时尤其如此。

此时你很可能说：“我好像有点明白了，但不太确定。”如果是这样，也不用担心。就目前而言，这样想就可以了：向setTimeout传递了一个指向处理程序的引用，定时器到期后，将调用该处理程序。下一章将更详细地讨论函数以及如何使用它们（如将它们传递给其他函数），因此现在只需这样理解就可以了。



这是代码。

```
var tick = true;
function ticker() {
    if (tick) {
        console.log("Tick");
        tick = false;
    } else {
        console.log("Tock");
        tick = true;
    }
}
setInterval(ticker, 1000);
```

请在这里记录你的分析。

请看下面的代码，你能搞清楚setInterval是做什么的吗？它类似于setTimeout，但存在细微的差别。答案见本章末尾。



这是输出。

世上没有愚蠢的问题

问： 有办法停止setInterval吗？

答： 有。setInterval返回一个timer对象。要停止该定时器，可将其传递给另一个函数clearInterval。

问： 你说过，setTimeout是一个方法，但它看起来像是一个函数。它是哪个对象的方法呢？

答： 好眼力。严格地说，它应写作window.setTimeout，但由于window是全局对象，可省略对象名，直接写作setTimeout。实际编写代码时经常这样做。

问： 引用属性window.onload时，也可省略window吗？

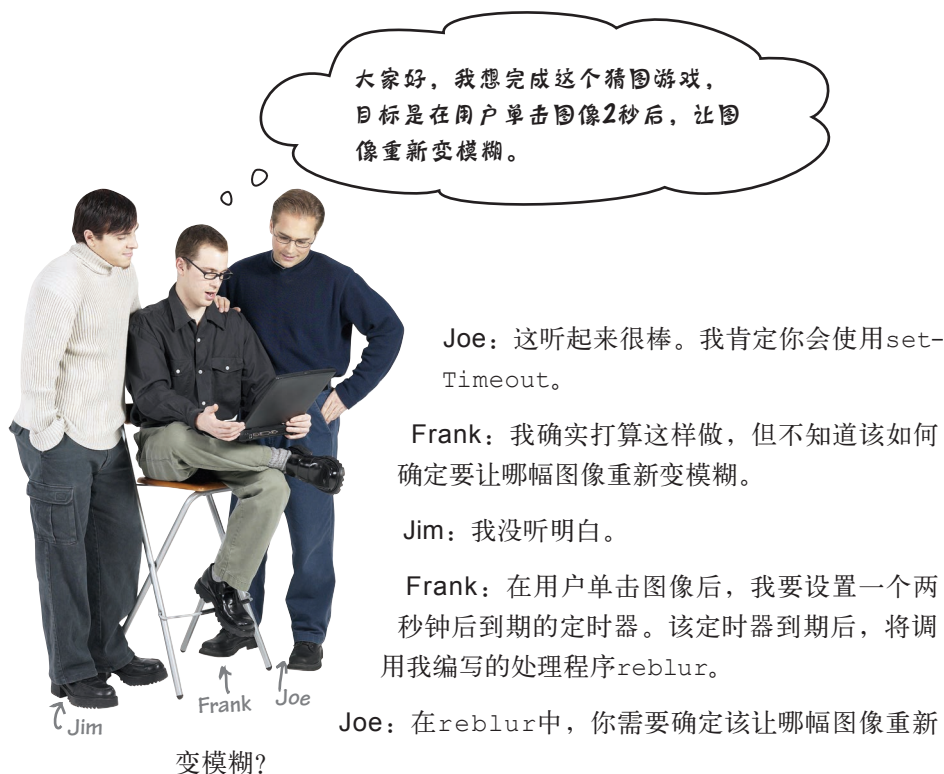
答： 可以，但大多数人都不会这样做，因为onload是一个常见的属性名（其他元素也可能有属性onload）。如果省略window，将让人不知道指的是哪个对象的onload属性。

问： 通过给onload赋值，我们将一个处理程序关联到了相应的事件。但通过使用setTimeout，好像可以给任意数量的定时器指定处理程序，是这样的吗？

答： 没错。调用setTimeout时，实际上是创建了一个定时器，并指定了与之相关联的处理程序。你可以创建任意数量的定时器，浏览器将跟踪每个定时器及其相关联的处理程序。

问： 还有其他向函数传递函数的示例吗？

答： 这样的例子很多。实际上，你将发现，在JavaScript中传递函数的情形非常普遍。不仅很多内置函数（如setTimeout和setInterval）都将函数作为参数，你编写的很多函数也会将函数作为参数。但这不是故事的全部，下一章将深入探讨这个主题。届时你将发现，在JavaScript中，可以以各种有趣的方式使用函数。



Frank: 没错。我没有向这个处理程序传递任何参数，它只是在定时器到期后被浏览器调用，因此我没法告诉处理程序，该让哪幅图像重新变模糊。我好像卡壳了。

Jim: 你查看了 `setTimeout` API 吗？

Frank: 没有，我只知道 Judy 告诉我的：`setTimeout` 将一个函数和一个时间间隔（单位为毫秒）作为参数。

Jim: 调用 `setTimeout` 时，你还可以传入第三个参数；触发时间事件时，这个参数将被传递给处理程序。

Frank: 哦，那太好了。我可以传入一个引用，它指向要重新变模糊的图像。那么调用处理程序时，将把这个引用传递给它？

Jim: 没错

Frank: Joe，你听明白了吗？

Joe: 听明白了，咱们来试一试。

完成猜图游戏

来完成猜图游戏的最后一道工序。我们要实现的目标如下：在图像的清晰版显示几秒之后，让它自动重新变模糊。刚才说过，调用setTimeout时，可指定一个要传递给事件处理程序的参数，如下所示：

```
window.onload = function() {  
    var images = document.getElementsByTagName("img");  
    for (var i = 0; i < images.length; i++) {  
        images[i].onclick = showAnswer;  
    }  
};
```

这些代码与前面相同，没有任何变化。

```
function showAnswer(eventObj) {  
    var image = eventObj.target;  
    var name = image.id;  
    name = name + ".jpg";  
    image.src = name;
```

但现在向用户显示清晰版图像时，同时调用setTimeout来创建一个将在2秒后触发的事件。

```
    setTimeout(reblur, 2000, image);  
}
```

我们将处理程序指定为reblur，将时间间隔设置为2000毫秒（2秒），并将第三个参数指定为要重新变模糊的图像。

```
function reblur(image) {  
    var name = image.id;  
    name = name + "blur.jpg";  
    image.src = name;  
}
```

这样，这个处理程序被调用时，将把要重新变模糊的图像传递给它。

这个处理程序获取图像的id，据此得到模糊版图像的名称。将属性src设置为该名称后，将用模糊版图像替换清晰版图像。



小心!

在IE8和更早版本的浏览器中，setTimeout不支持额外的参数。

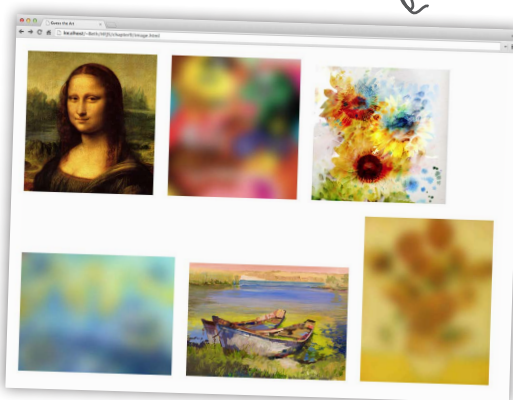
没错，这些代码不适用于IE8和更早的版本，但阅读本书时，你使用的应该不是IE8。本书后面将介绍另一种处理方式，让代码适用于IE8和更早版本的浏览器。

测试定时器

添加的代码虽然不多，却让这个猜图游戏大不相同。现在，用户单击图像时，浏览器将通过定时器事件在幕后跟踪何时需要调用处理程序`reblur`，让图像重新变模糊。这给人以异步的感觉：何时单击图像由用户决定，但在幕后，将根据单击事件和定时器事件异步地调用代码。这里并没有任何高深的算法控制该在何时调用哪些代码，有的只是一系列设置、创建和响应事件的代码。

现在当你单击图像时，将显示其清晰版，并在2秒后重新变模糊。

请快速单击大量的图像，对这个游戏进行详尽的测试。效果始终如你期望的那样吗？请参阅前面的代码，想想浏览器是如何跟踪所有需要重新变模糊的图像的。



世上没有愚蠢的问题

问：调用`setTimeout`时，只能向指定的处理程序传递一个参数吗？

答：不是这样的。实际上，你可以向处理程序传递任意数量的参数：0个、1个或更多。

问：`setTimeout`为何不向事件处理程序传递一个事件对象？

答：事件对象主要用于DOM事件处理程序。`setTimeout`不向处理程序传递事件对象，因为时间事件并非由特定的元素触发。

问：`showAnswer`是一个处理程序，并且在其代码中创建了另一个处理程序`reblur`。是这样的吗？

答：没错。实际上，在JavaScript中，这样的做法很常见：在处理程序中，创建其他事件处理程序再正常不过了。这就是本章开头所说的编程风格——异步编程。为创建猜图游戏，我们并没有编写按从头到尾的顺序执行的算法，而是创建了一些在事件发生时执行的事件处理程序。它们跟踪被单击的图像，显示其清晰版，再让图像重新变模糊。

问：有基于DOM的事件、定时器事件等。是不是有很多不同类型的事件？

答：在JavaScript中，你处理的很多事件都是DOM事件（如单击元素触发的事件）或定时器事件（使用`setTimeout`或`setInterval`创建的事件）。还有与API相关的事件，如Geolocation、LocalStorage、Web Worker等JavaScript API触发的事件。有关这些API的详细信息，请参阅《Head First HTML5 Programming中文版》。最后，还有一系列与I/O相关的事件，如使用`XmlHttpRequest`向Web服务请求数据时引发的事件以及使用Web套接字引发的事件。同样，有关`XmlHttpRequest`的详细信息，请参阅《Head First HTML5 Programming中文版》。



〇〇

大家好，台式机用户希望将鼠标指向图像时不单击就能显示其清晰版。我们能够实现这种功能吗？

Judy：为实现这种功能，需要使用鼠标移动事件。要给元素的这种事件指定处理程序，可使用属性onmouseover：

```
myElement.onmouseover = myHandler;
```

Judy：另外，鼠标移出元素时，mouseout事件将指出这一点。要为这种事件指定处理程序，可使用属性onmouseout。



练习

请重新编写前面的代码，使得在用户将鼠标指向图像元素和将鼠标移出图像元素时，分别显示清晰版图像以及让图像重新变模糊。务必对你编写的代码进行测试，并查看本章末尾的答案。

在这里编写JavaScript代码。



练习

编写好图像猜测游戏后，Judy又编写了一些代码，在每周团队例会中要求大家进行审核。实际上，她发起了一个小小的竞赛：谁能第一个正确地指出这些代码的功能，就奖励谁一顿午餐。谁将获胜呢？Jim、Joe、Frank还是你？

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Don't resize me, I'm ticklish!</title>
<script>
    function resize() {
        var element = document.getElementById("display");
        element.innerHTML = element.innerHTML + " that tickles!";
    }
</script>
</head>
<body>
<p id="display">
    Whatever you do, don't resize this window! I'm warning you!
</p>
<script>
    window.onresize = resize;
</script>
</body>
</html>
```



请在这里描述这些代码的功能。代码中涉及哪些事件？是如何指定处理程序的？这些事件在何时发生？别只描述这些代码的功能，也请在浏览器中运行一下它们。

代码实验室

我们发现了一些极度可疑的代码，需要你帮忙测试。我们对这些代码进行了基本分析，它们看起来是完全标准的JavaScript代码，但有些地方有点怪。下面是两段代码，你需要找出每段中怪异的地方，通过测试确认它们管用，再尝试分析它们到底是做什么的，并将这些记录在本页中。要查看我们的分析，请参阅下一页。

代码片段 #1

```
var addOne = function(x) {  
    return x + 1;  
};  
var six = addOne(5);
```

代码片段 #2

```
window.onload = function() {  
    alert("The page is loaded!");  
}
```



代码实验室：分析

代码片段 #1

```
var addOne = function(x) {  
    return x + 1;  
};  
var six = addOne(5);
```

乍一看，这些代码好像只是定义了一个函数。这个函数将传入的参数加1，再返回结果。

但仔细观察后发现，这不是普通的函数定义，而是声明了一个变量，并将一个没有名称的函数赋给它。

接下来调用了这个函数。调用是通过变量名，而不是函数定义中指定的名称进行的。

确实很奇怪，虽然这让我们想起在对象中定义方法的方式。

代码片段 #2

```
window.onload = function() {  
    alert("The page is loaded!");  
}
```

这里的情况看起来与前面类似。不是定义函数后，再将其名称赋给属性`window.onload`，而是直接将函数赋给这个属性。同样，这里也没有指定函数的名称。

我们将这些代码添加到HTML页面中，并进行测试。它们的作用看起来与预期的相同。对于代码片段#1，调用赋给了变量`addOne`的函数时，得到的结果比传入的数字大1，好像完全正确。对于代码片段#2，网页加载后显示了一个提示框，其中包含消息“The page is loaded!”。

这些测试表明，定义函数时好像可以不指定名称，而这样的函数定义可放在任何需要表达式的地方。

所有这些都意味着什么呢？请继续往下阅读，下一章将全面阐述这些怪异的函数。



要点

- 大多数JavaScript代码都是用来响应事件的。
- 可编写代码来响应的事件类型众多。
- 要响应事件，可以编写并注册一个**事件处理程序**。例如，要指定单击事件处理程序，可将事件处理程序赋给元素的onclick属性。
- 并非必须响应所有的事件，可以只处理你感兴趣的事件。
- 我们将**函数**用作事件处理程序，因为它们让我们能够编写在事件发生时再执行的代码。
- 编写用于处理事件的代码不同于从头到尾执行的代码。事件处理程序的运行时间和运行顺序都是不确定的，它们是**异步**的。
- 发生在DOM元素上的事件（DOM事件）导致一个event被传递给事件处理程序。
- event对象包含一些属性，这些属性提供了有关事件的额外信息，其中包括事件的类型（"click"或"load"）和目标（触发事件的对象）。
- 在较旧的IE版本（IE8和更早的版本）中，使用的事件模型不同于其他浏览器。有关这方面的详细信息，请参阅附录。
- 可能在短时间内发生很多事件。发生的事件太多，浏览器无法实时地处理时，这些事件将按发生的顺序存储到**事件队列**中，让浏览器能够依次调用每个事件的处理程序。
- 如果事件处理程序执行的计算非常复杂，将导致队列中的其他事件无法得到及时处理，因为浏览器每次只能执行一个事件处理程序。
- 函数setTimeout和setInterval都用于创建在指定时间过后发生的事件。
- 方法getElementByTagName返回一个NodeList，其中包含0个、1个或更多的element对象。NodeList类似于数组，你可以对其进行迭代。

事件群英谱

click

在网页中单击（或轻按）时将触发这个事件。

load

浏览器加载网页完毕后触发的事件。

mousemove

在元素上移动鼠标时，将触发这个事件。

keypress

用户按下任何键都将触发这个事件。

unload

用户关闭浏览器窗口或切换到其他网页时，将触发这个事件。

mouseover

用户将鼠标指向元素时，将触发这个事件。

mouseout

用户将鼠标从元素上移开时将触发这个事件。

resize

每当用户调整浏览器窗口的大小时，都将触发这个事件。

dragstart

用户拖曳网页中的元素时，将触发这个事件。

touchstart

在触摸设备上，用户触摸并按住元素时，将触发这个事件。

play

用户单击网页中<video>元素的播放按钮时，将触发这个事件。

drop

用户放下拖曳的元素时，将触发这个事件。

pause

用户单击<video>元素的暂停按钮时，将触发这个事件。

touchend

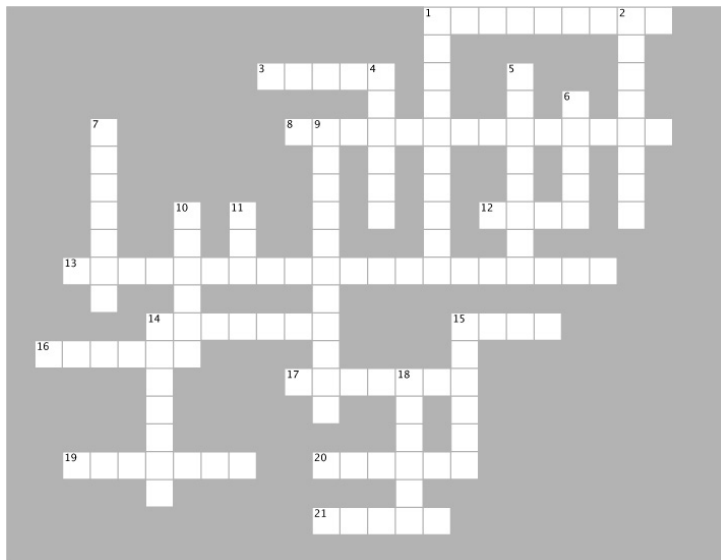
用户停止触摸时，将触发这个事件。

我们介绍了load、click、mousemove、mouseover、mouseout、resize事件以及定时器事件，这些只是全部事件的冰山一角。这里的事件群英谱列出了你肯定会遇到且在学习Web编程时必须探索的事件。



JavaScript填字游戏

请利用你掌握的事件知识，完成这个填字游戏。



横向

1. 使用事件对象的这个属性，可获取事件发生的时间。
3. 用户单击鼠标时触发的事件。
8. 事件是以什么方式处理的？
12. 5000毫秒相当于多少秒？
13. 根据标签名从DOM中获取多个元素的方法。
14. 设计用于响应事件的函数被称为事件_____。
15. 方法setTimeout用于创建什么事件？
16. 浏览器只有一个控制_____。
17. 浏览器每次只执行一个事件_____。
19. 在传递给mouseover事件处理程序的事件对象中，包含的表示鼠标X坐标的属性。
20. 对于DOM事件处理程序，将向它传递一个事件_____。
21. 要向时间事件处理程序传递参数，可通过setTimeout的第几个参数进行传递？

纵向

1. 用户在触摸屏设备上触摸时，将触发的事件。
2. zero.jpg是谁的画像？
4. 电影《绿野仙踪》中的台词“you're not in _____ anymore”，用来说明开始事件编程后的情形非常贴切。
5. 在JavaScript中，可以向函数传递_____。
6. 如果短时间内发生的事件太多，浏览器将把这些事件存储在事件_____中。
7. 很多操作都会触发事件，但创建_____时不会。
9. 要让时间事件反复地发生，可使用哪个函数？
10. 要指定网页加载事件处理程序，可使用window对象的哪个属性？
11. 本章一个参与讨论如何编写代码的人物。
14. 要给时间事件指定事件处理程序，可将_____作为第一个参数传递给setTimeout。
15. 事件对象的一个属性，让你能够确定图像猜测游戏用户单击的是哪幅图像。
18. 一种程序执行方式，包含事件处理代码的程序不是以这种方式执行的。



磨笔上阵 答案

挑选上面列出的两个事件。如果浏览器能够在这些事件发生时发出通知，你能编写哪些有趣的代码来响应这些事件？

以用户提交表单时触发的事件为例。如果发生这种事件时会通知你，你可获取用户在表单中填写的所有数据，并通过检查确认它们都是有效的。（例如，在电话号码文本框中输入的东西像电话号码，或者所有必须填写的文本框都不为空。）执行这种检查后，就可以将表单提交给服务器了。

鼠标移动事件呢？如果用户移动鼠标时你会得到通知，你就可以创建直接在浏览器中运行的绘图应用程序。

如果用户滚动网页时你会得到通知，你就可以做一些有趣的事件，如滚动到图像所处的位置时显示它们。

变身浏览器答案



下面是猜图游戏的代码，你的任务是变身浏览器，搞清楚每个事件发生后该如何做。完成这个练习后，请查看本章末尾的答案，看看你是否都明白了。答案如下。

```
window.onload = init;
function init() {
    var image = document.getElementById("zero");
    image.onclick = showAnswer;
}

function showAnswer() {
    var image = document.getElementById("zero");
    image.src = "zero.jpg";
}
```

网页加载时……	定义函数init和showAnswer。
	将init指定为加载事件处理程序。
网页加载事件发生时……	调用加载事件处理程序init。
	获取id为零的图像元素。
	将该图像元素的单击事件处理程序设置为showAnswer。
图像单击事件发生时……	
	调用showAnswer。
	获取id为零的图像元素。
	将其src属性设置为“zero.jpg”。

连连看

答案

前面说过，表示DOM事件的事件对象包含一些属性，它们提供了有关事件的额外信息。下面是事件对象可能包含的其他一些属性，请将每个属性及其用途连接起来。

target

要确定用户单击的位置离浏览器窗口的上边缘有多远？使用我吧。

type

我存储着触发事件的对象。我可以是各种不同的对象，但通常是元素对象。

timeStamp

在触摸设备上，可使用我来确定用户使用了多少根手指来触摸屏幕。

keyCode

我是一个字符串，如"click"或"load"，指出了发生的是哪种事件。

clientX

想知道事件是何时发生的？我是可向你提供这种信息的属性。

请注意，较旧的IE版本不支持属性timeStamp。

clientY

要确定用户单击的位置离浏览器窗口的左边缘有多远？使用我吧。

touches

我能告诉你用户刚按下了哪个键。



海盗船长得到了一张藏宝图，需要你帮忙找出宝藏的坐标。为此，你将编写一些代码，使在地图上移动鼠标时显示其坐标。

下面就是所需的代码。它们在网页中包含了地图，并创建了一个用于显示坐标的段落元素。你需要补全处理事件的代码。祝你好运，我们可不想看到你葬身海底！答案如下。



练习
答案



```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Pirates Booty</title>
  <script>
    window.onload = init;
    function init() {
      var map = document.getElementById("map");
      map.onmousemove = showCoords;
    }

    function showCoords(eventObj) {
      var map = document.getElementById("coords");
      var x = eventObj.clientX;
      var y = eventObj.clientY;
      map.innerHTML = "Map coordinates: "
                    + x + ", " + y;
    }
  </script>
</head>
<body>
  
  <p id="coords">Move mouse to find coordinates...</p>
</body>
</html>
```

我们将鼠标指向X时，显示的坐标为： 200, 190



请看下面的代码，你能搞清楚setInterval是做什么的吗？它类似于setTimeout，但存在细微的差别。

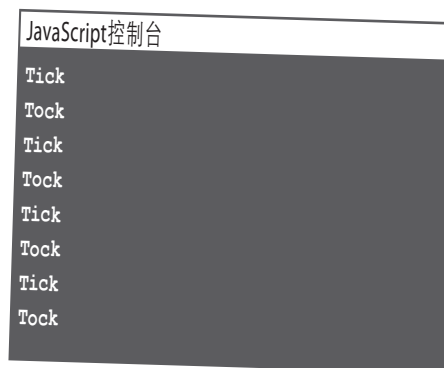
答案如下。

这是代码。

```
var tick = true;

function ticker() {
  if (tick) {
    console.log("Tick");
    tick = false;
  } else {
    console.log("Tock");
    tick = true;
  }
}

setInterval(ticker, 1000);
```



这是输出。

请在这里记录你的分析。

与setTimeout一样，setInterval的第一个参数也是事件处理程序，第二个参数也是时间间隔。

但不同于setTimeout，setInterval执行事件处理程序多次；实际上是不断地执行，没完没了。（实际上，可以让它停止执行事件处理程序，如下所示。）在前面的示例中，setInterval每隔1000毫秒（1秒）调用处理程序ticker一次，而处理程序ticker根据变量tick的值决定在控制台中显示Tick还是Tock。

也就是说，setInterval在定时器到期时触发事件，并重启定时器。

```
var t = setInterval(ticker, 1000);

clearInterval(t);
```

← 可将setInterval返回的结果存储在一个变量中……

← 在要停止间隔定时器时调用clearInterval，并将这个变量传递给它。



请重新编写前面的代码，使得在用户将鼠标指向图像元素和将鼠标移出图像元素时，分别显示清晰版图像以及让图像重新变模糊。务必对你编写的代码进行测试。答案如下：

```
window.onload = function() {  
    var images = document.getElementsByTagName("img");  
    for (var i = 0; i < images.length; i++) {  
        images[i].onclick = showAnswer;  
        images[i].onmouseover = showAnswer;  
        images[i].onmouseout = reblur;  
    }  
};  
  
function showAnswer(eventObj) {  
    var image = eventObj.target;  
    var name = image.id;  
    name = name + ".jpg";  
    image.src = name;  
  
    setTimeout(reblur, 2000, image);  
}  
  
function reblur(eventObj) {  
    var image = eventObj.target;  
    var name = image.id;  
    name = name + "blur.jpg";  
    image.src = name;  
}
```

首先，删除将事件处理程序showAnswer赋给属性onclick的语句。

然后，将这个事件处理程序赋给图像元素的属性onmouseover。

接下来，需要将reblur用作mouseout事件处理程序（而不是定时器事件处理程序）。为此，我们将reblur赋给图像元素的属性onmouseout。

我们不再利用定时器来让图像重新变模糊，而是在用户将鼠标移出图像元素时让图像重新变模糊。

鉴于reblur现在被用作mouseout事件处理程序，为确定应让哪幅图像重新变模糊，必须使用事件对象。与showAnswer中一样，我们使用属性target来获取要重新变模糊的图像。获得该图像元素后，使图像重新变模糊的代码与以前相同。



练习答案

编写好猜图游戏后，Judy又编写了一些代码，在每周团队例会中要求大家进行审核。实际上，她发起了一个小小的竞赛：谁能第一个正确地指出这些代码的功能，就奖励谁一顿午餐。谁将获胜呢？Jim、Joe、Frank还是你？



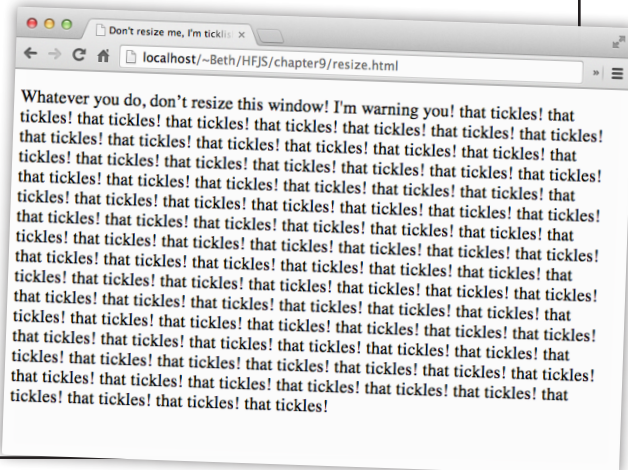
```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Don't resize me, I'm ticklish!</title>
<script>
    function resize() {
        var element = document.getElementById("display");
        element.innerHTML = element.innerHTML + " that tickles!";
    }
</script>
</head>
<body>
<p id="display">
    Whatever you do, don't resize this window! I'm warning you!
</p>
<script>
    window.onresize = resize;
</script>
</body>
</html>
```

事件处理程序名为`resize`，它只是在`id`为`display`的段落元素中添加一些文本。

这里要响应的是`resize`事件，因此我们创建一个名为`resize`的处理程序，并将其赋给`window`对象的属性`onresize`。

我们在网页末尾的`<script>`元素中指定`resize`事件处理程序。别忘了，我们希望仅当网页加载完毕后才运行这个脚本，因此不能过早地指定`resize`事件处理程序。

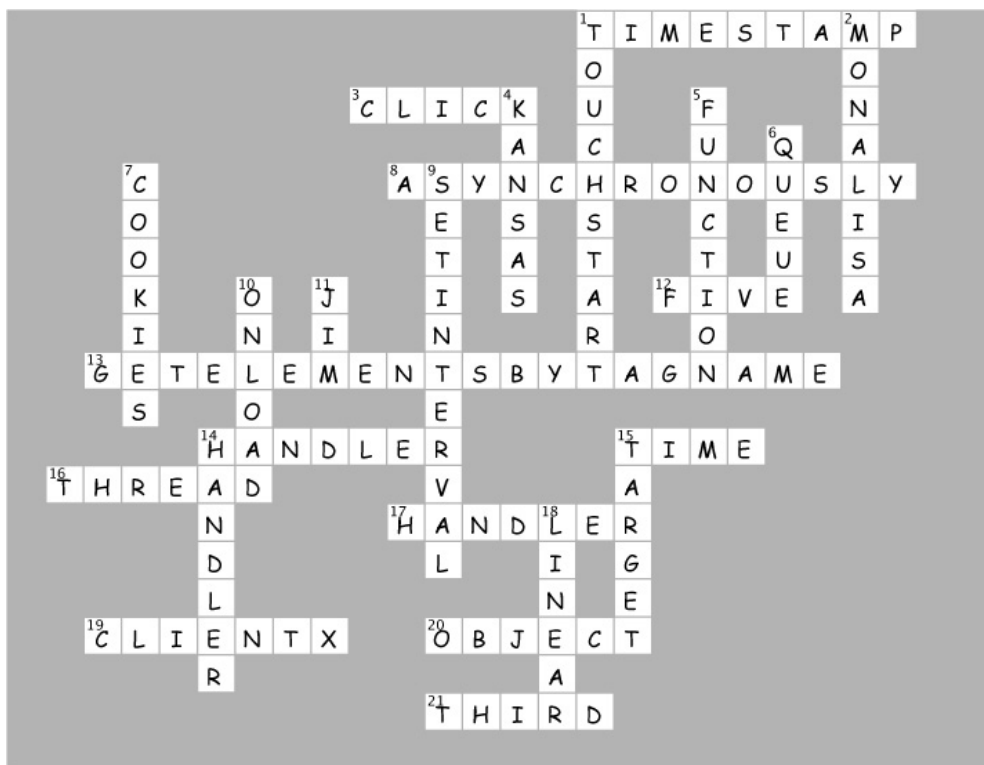
当你调整浏览器窗口的大小时，将调用事件处理程序`resize`。它这样更新网页：在`id`为`display`的段落元素中添加新文本内容`that tickles`。





JavaScript填字游戏答案

请利用你掌握的事件知识，完成这个填字游戏。



10 函数是一等公民

自由的函数



熟知函数你就能成为明星。无论在什么行业和学科，都存在让大师有别于芸芸众生的重要分水岭；就JavaScript而言，这个分水岭就是**熟知函数**。在JavaScript中，函数不可或缺，很多**设计和组织代码**的技巧都要求你精通并熟练地使用函数。通往精通函数的学习道路很有趣，但常常会让人迷惑不解，因此你一定要有心理准备。本章将更深入地介绍JavaScript函数，你将像刘姥姥进了大观园，见到一些奇异、古怪而神奇的东西。

关键字function神秘的双面人生

到目前为止，我们都像下面这样声明函数：

```
function quack(num) {  
  for (var i = 0; i < num; i++) {  
    console.log("Quack!");  
  }  
}
```

一个标准的函数声明，由关键字function、函数名、形参和代码块组成。

我们可以这样调用这个函数：在函数名后加上用圆括号括起的实参。

```
quack(3);
```



这里没有什么令人惊讶的地方，下面来详细阐述相关的术语：上述第一条语句是一个函数声明，它创建一个函数。这个函数的名称为quack，可用于引用和调用这个函数。

然而，正如你在上一章末尾看到的，关键字function还有另一种用法，这让情况变得扑朔迷离：

```
var fly = function(num) {  
  for (var i = 0; i < num; i++) {  
    console.log("Flying!");  
  }  
};
```

看起来不那么标准：这个函数没有名称，而且位于一条赋值语句右边，被赋给一个变量。

我们也可以调用这个函数，但必须使用变量fly来调用。

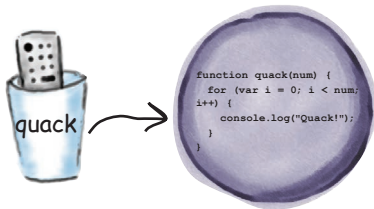
```
fly(3);
```

在语句（如赋值语句）中以这样的方式使用关键字function时，创建的是函数表达式。请注意，不同于函数声明，上述函数没有名称。另外，该表达式的结果是一个值，这个值被赋给变量fly。这是一个什么样的值呢？我们将它赋给变量fly，然后调用了它，因此它肯定是一个指向函数的引用。



编码技巧

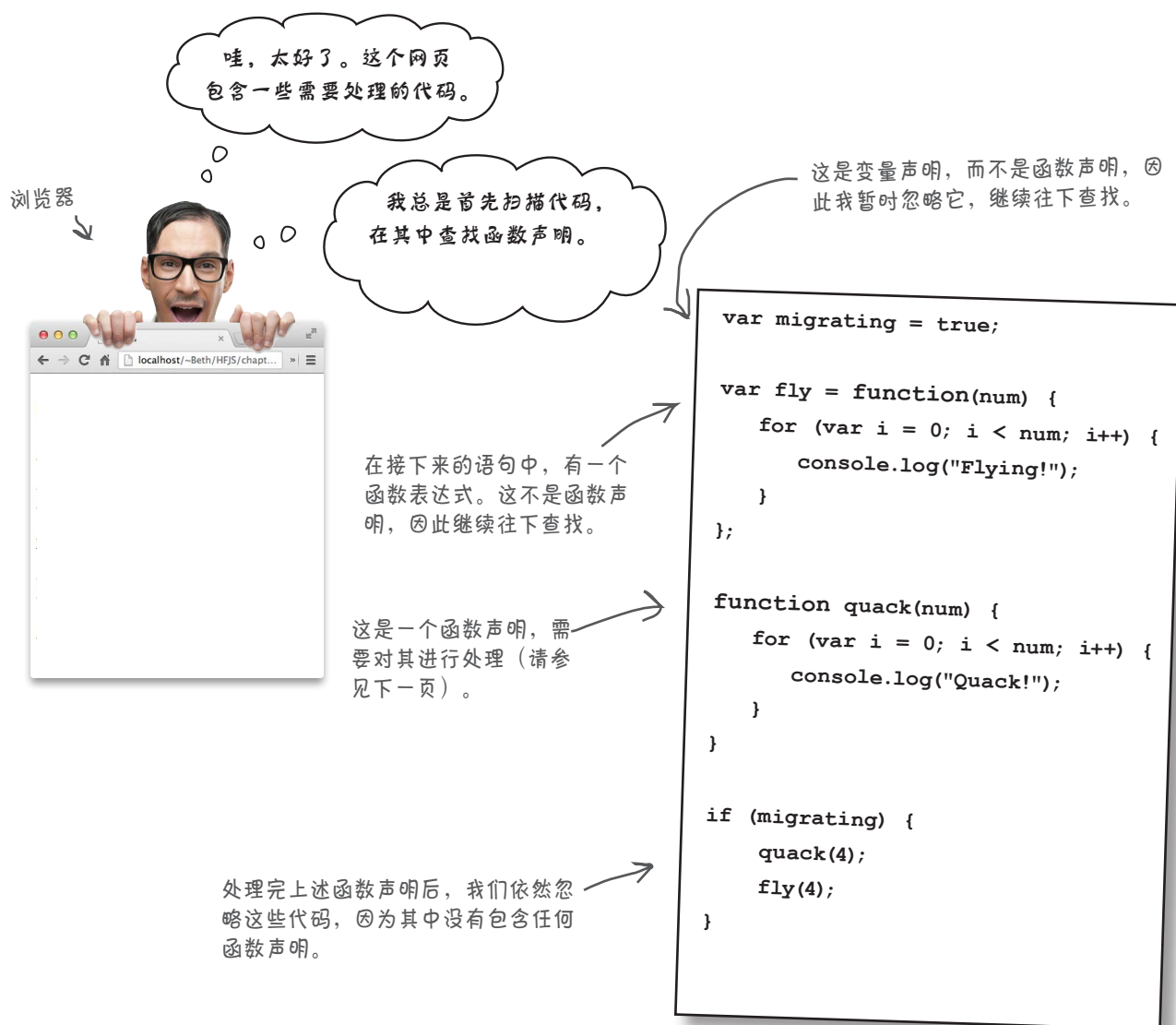
顾名思义，函数引用是指向函数的引用。你可以使用函数引用来调用函数，还可以将它们赋给变量、存储在对象中、传递给函数或从函数返回它们（就像对象引用一样）。



函数声明和函数表达式

无论你使用函数声明还是函数表达式，得到的都是函数。那么它们的区别何在呢？仅仅是函数声明更方便，还是函数表达式有些独特之处使其很有用？抑或它们只是两种做相同事情的不同方式而已？

乍一看，函数声明和函数表达式差别不大，但实际上它们存在根本性差别。要明白这种差别，首先需要研究一下浏览器在运行阶段如何处理代码。下面就来看看浏览器如何分析并执行网页中的代码。



分析函数声明

在分析网页期间（执行任何代码之前），浏览器查找函数声明。找到函数声明时，浏览器创建相应的函数，并将得到的函数引用赋给与函数同名的变量，如下所示。

我们遇到了一个函数声明，必须先对其进行处理，再做其他的事情。

这是代码中的函数声明，来看看浏览器如何处理它。

```
var migrating = true;

var fly = function(num) {
  for (var i = 0; i < num; i++) {
    console.log("Flying!");
  }
};

function quack(num) {
  for (var i = 0; i < num; i++) {
    console.log("Quack!");
  }
}

if (migrating) {
  quack(4);
  fly(4);
}
```

我将这个函数存储起来，以便能够在它被调用时获取它。

```
function quack(num) {
  for (var i = 0; i < num; i++) {
    console.log("Quack!");
  }
}
```

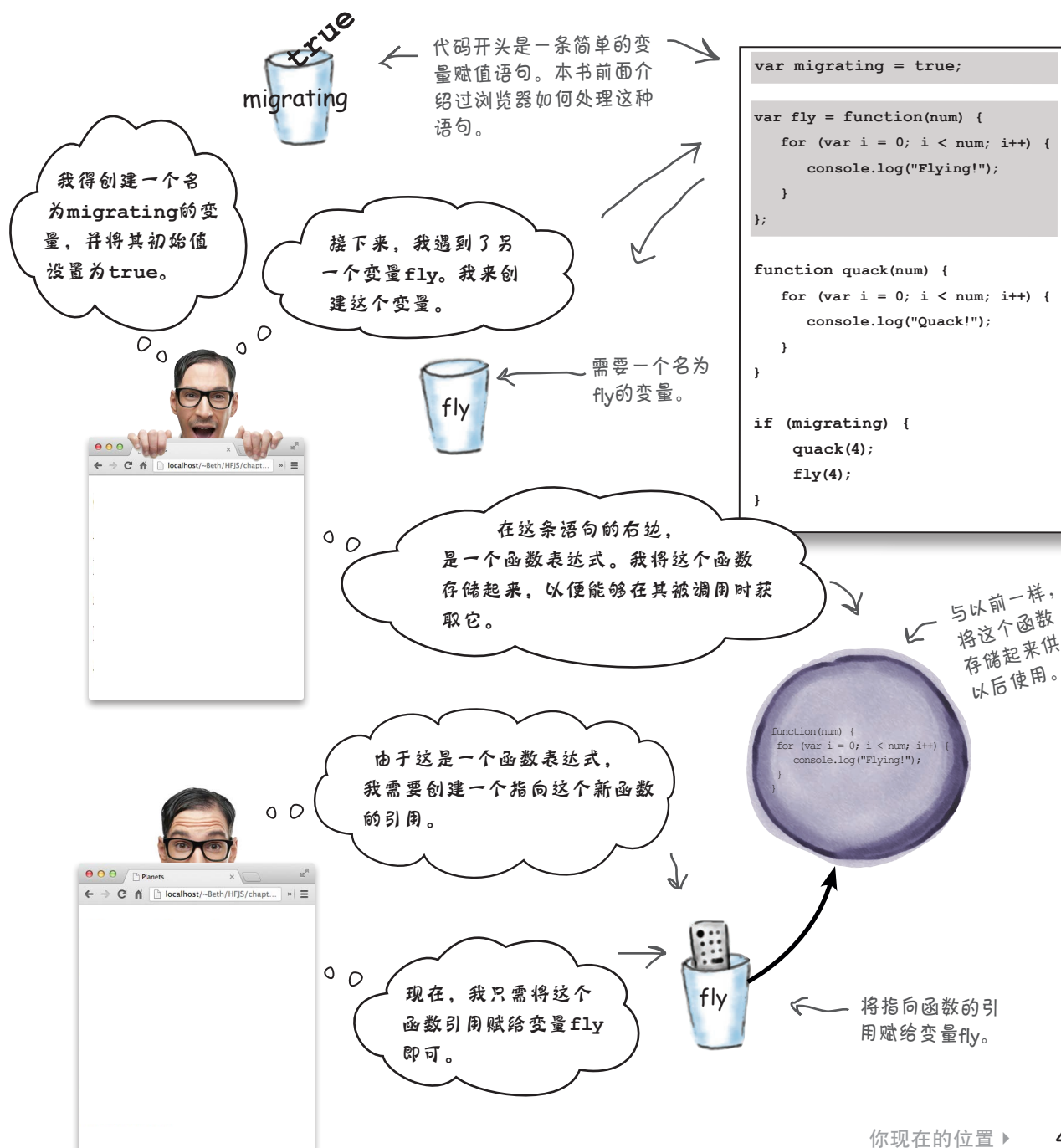
这个函数名为quack，因此创建一个名为quack的变量来存储指向这个函数的引用。



将函数存储在这里，以便函数被调用时能够使用它。

接下来呢？浏览器执行代码

处理所有的函数声明后，浏览器回到代码开头，开始按从头到尾的顺序执行代码。我们来看看浏览器是如何做的。



继续往下执行

处理完变量fly后，浏览器继续往下执行。接下来的语句是函数quack的声明，因为已经处理过了，所以浏览器跳过声明，接着执行后面的条件语句。我们来看看浏览器是如何做的。

处理完函数表达式后，继续往下执行。

```
var migrating = true;

var fly = function(num) {
  for (i = 0; i < num; i++) {
    console.log("Flying!");
  }
};

function quack(num) {
  for (i = 0; i < num; i++) {
    console.log("Quack!");
  }
}

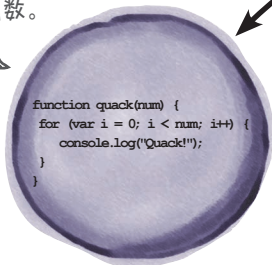
if (migrating) {
  quack(4);
  fly(4);
}
```

变量migrating为true，因此需要执行if语句的代码块。在if语句的语句块中，调用了函数quack。我怎么知道这是函数调用呢？因为这里使用了函数名quack，且它后面跟着圆括号。

quack(4);

别忘了，quack变量是一个引用，指向我之前存储的函数。

这是根据函数quack的函数声明创建的函数。



在这个函数调用中，指定了一个实参，因此我将这个实参传递给函数……

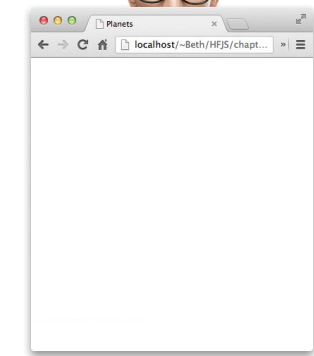
为调用函数，我们将实参的副本传递给形参……

4

```
function quack(num) {
  for (var i = 0; i < num; i++) {
    console.log("Quack!");
  }
}
```

再执行函数体。

再执行函数体中的代码，这些代码在控制台中打印Quack!四次。



结束

余下的全部工作是调用根据函数表达式创建的函数fly。我们来看看浏览器是如何处理这一点的。

看，另一个函数调用。我怎么知道这是一个函数调用呢？因为这里使用了变量名fly，而且后面跟着圆括号。

fly(4);

```
var migrating = true;

var fly = function(num) {
  for (var i = 0; i < num; i++) {
    console.log("Flying!");
  }
};

function quack(num) {
  for (var i = 0; i < num; i++) {
    console.log("Quack!");
  }
}

if (migrating) {
  quack(4);
  fly(4);
}
```

别忘了，变量fly是一个引用，指向我之前存储的函数。

这是变量fly
指向的函数。

```
function(num) {
  for (var i = 0; i < num; i++) {
    console.log("Flying!");
  }
}
```



在这个函数调用中，指定了一个实参，因此我将这个实参传递给函数……

4

为调用函数，我们将实参的副本传递给形参……

```
function(num) {
  for (var i = 0; i < num; i++) {
    console.log("Flying!");
  }
}
```

再执行函数体。

再执行函数体中的代码，这些代码在控制台中打印Flying!四次。



磨笔上阵

基于前述浏览器处理函数quack和fly的代码的方式，你能得出哪些有关函数声明和函数表达式的结论呢？请选择你认为正确的所有说法。继续往下阅读前，务必查看本章末尾的答案。

- ☐ 在处理其他代码之前，先处理函数声明。
- ☐ 函数表达式随其他代码一起被处理。
- ☐ 函数声明不返回指向函数的引用；而是创建一个与函数同名的变量，并将指向函数的引用赋给它。
- ☐ 函数表达式返回一个引用，该引用指向函数表达式创建的函数。
- ☐ 可在变量中存储指向函数的引用。
- ☐ 函数声明是完整的语句，而函数表达式只是语句的一部分。
- ☐ 对于函数声明和函数表达式创建的函数，用相同的方式处理调用它们的语句。
- ☐ 函数声明是一种屡试不爽的函数创建方式。
- ☐ 应尽可能使用函数声明，因为它们首先被处理。

世上没有愚蠢的问题

问：我们见过像`3+4`和`Math.random() * 6`等这样的表达式，但函数怎么可能是表达式呢？

答：结果为一个值的任何东西都是表达式。`3+4`的结果为7，`Math.random() * 6`的结果是一个随机数，而函数表达式的结果是一个函数引用。

问：函数声明为何不是表达式呢？

答：函数声明是一条语句。可以认为它包含一条隐藏的赋值语句，这条语句将函数引用赋给一个变量。函数表达式不自动将函数引用赋给任何变量，你必须显式地这样做。

问：让变量指向一个函数能带来什么好处？

答：一个好处是，你可以使用它来调用相应的函数：

```
myFunctionReference();
```

另外，还可以将函数引用传递给函数或从函数返回函数引用，这将在稍后几页介绍。

问：函数表达式只能出现在赋值语句右边吗？

答：根本不是这样的。函数表达式和其他表达式一样，可出现在很多不同的地方。这个问题问得很好，稍后将详细讨论，敬请期待。

问：变量可存储指向函数的引用，这种变量指向的是什么呢？仅仅是函数体中的代码吗？

答：这是一种看待函数的不错方式，但你可以更进一步，将函数视为经过简单包装的代码，可随时调用。在本章后面你将看到，这种经过简单包装的函数中还有其他一些东西，而不仅仅是函数体的代码。



我们刚才看到了，对于函数声明和函数表达式创建的函数，调用方式完全相同。函数声明和函数表达式到底有何不同呢？我感觉自己好像遗漏了一些细微的差别。

存在细微的差别。首先，你说的没错，无论是使用函数声明还是函数表达式，最终都将得到一个函数。但它们之间存在一些重要的差别。一个差别是，使用函数声明时，函数将在**执行代码前**创建；而使用函数表达式时，函数将在**运行阶段**执行代码时创建。

另一个差别与函数命名相关：使用函数声明时，将创建一个与函数同名的变量，并让它指向函数；而使用函数表达式时，通常不给函数指定名称，因此你要么在代码中将函数赋给一个变量，要么以其他方式使用函数表达式。

本章后面将讨论这些使用函数表达式的方式。

请将这些差别牢记在心，稍后这些差别的重要性将显现出来。就目前而言，你只需记住函数声明和函数表达式的处理方式以及函数名是如何处理的即可。

变身浏览器



下面是一些JavaScript代码。你的任务是像浏览器那样执行它们，并在每个函数被创建时，在右边将其记录下来。别忘了分两遍处理这些代码：第一遍处理函数声明，第二遍处理函数表达式。

```
var midi = true;
var type = "piano";
var midiInterface;

function play(sequence) {
    // 函数体的代码
}

var pause = function() {
    stop();
}

function stop() {
    // 函数体的代码
}

function createMidi() {
    // 函数体的代码
}

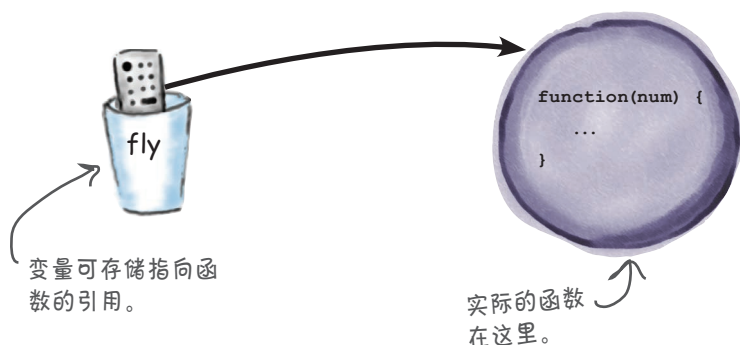
if (midi) {
    midiInterface = function(type) {
        // 函数体的代码
    };
}
```

请按顺序列出所创建函数的名称。对于根据函数表达式创建的函数，列出它被赋给的变量的名称。我们已经写出了创建的第一个函数。

play

函数怎么也是值呢

确实，我们都将函数视为调用的东西，但也可将函数视为值。这个值实际上是指向函数的引用。正如你看到的，无论是使用函数声明还是函数表达式来定义函数，得到的都是指向这个函数的引用。



对于函数，可执行的最简单的操作之一是将其赋给变量，如下所示：

```
function quack(num) {
  for (var i = 0; i < num; i++) {
    console.log("Quack!");
  }
}

var fly = function(num) {
  for (var i = 0; i < num; i++) {
    console.log("Flying!");
  }
}

var superFly = fly;
superFly(2);

var superQuack = quack;
superQuack(3);
```

这里再次列出了前面的两个函数。别忘了，`quack`是使用函数声明定义的，而`fly`是使用函数表达式定义的。这些做法得到的都是函数引用，而这些引用分别存储在变量`quack`和`fly`中。

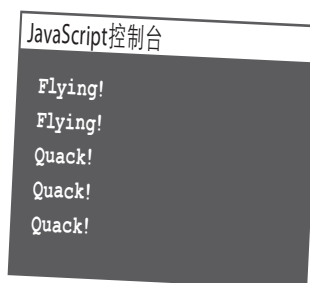
函数声明自动将引用赋给与函数同名的变量，这里是`quack`。

使用函数表达式时，你必须显式地将得到的引用赋给一个变量。这里将引用存储到了变量`fly`中。

将`fly`赋给变量`superFly`后，`superFly`存储了相应的函数引用，因此可在这个变量后面加上圆括号和实参来调用相应的函数！

虽然`quack`是由函数声明创建的，但它存储的也是一个函数引用，因此可以将它赋给变量`superQuack`，进而通过这个变量来调用相应的函数。

换句话说，引用就是引用，不管是以什么方式创建的（使用函数声明或者函数表达式创建）！





磨笔上阵

为牢牢地记住函数是值的概念，咱们来玩一个碰运气取胜的游戏。请尝试下面这个猜豆子游戏，你会赢还是输呢？请试一试，并找出结果。

```
var winner = function() { alert("WINNER!"); };
var loser = function() { alert("LOSER!"); };
// 咱们来热热身
winner();
// 将函数引用赋给其他变量
var a = winner;
var b = loser;
var c = loser;
a();
b();
// 来玩一个猜豆子游戏，看看你的运气怎样
c = a;
a = b;
b = c;
c = a;
a = c;
a = b;
b = c;
a();
```

← 别忘了，这些变量分别存储了指
← 向函数winner和loser的引用。我
们可以像其他任何值一样，将这
些引用赋给其他变量。

别忘了，随时都可以
调用指向函数的引用。

← 手动执行这些代码，
看看你会赢还是输。



请将函数视为值，就像数字、字符串、布尔值和对象一样。相比于上述其他值，函数值的不同之处在于我们可以调用它。



起底函数

本周访谈：理解函数

Head First: 函数，很高兴你又来到了我们的节目。你非常神秘，我们的读者渴望更深入地了解你。

函数: 确实，我晦涩难懂。

Head First: 咱们先来说说这个概念吧：你既可以使用声明来创建，也可以使用表达式来创建。为何要提供两种定义方式呢？难道一种方式不够吗？

函数: 别忘了，这两种函数定义方式的功能存在细微的差别。

Head First: 但结果都一样，就是一个函数，不是吗？

函数: 你说的没错，但函数声明在幕后为你做了一些工作：它在创建函数的同时，还创建了一个用于创建函数引用的变量。函数表达式也创建函数，结果为一个引用，但如何使用它由你决定。

Head First: 无论在何种情况下，我们不是都将函数表达式创建的引用存储到变量中吗？

函数: 绝对不是。事实上，我们通常不这样做。别忘了，函数引用是一个值。想想吧，对于其他类型的值，如对象引用，你都能做哪些事情呢？对于函数引用，也可以做这些事情。

Head First: 这怎么可能呢？我声明函数、调用函数，这些差不多就是语言允许我对函数做的所有事情，不是吗？

函数: 大错特错。你需要将函数视为值，就像对象和基本类型值一样。获得函数后，你就可以对它做各种事情了。不过，函数和其他类型的值之间存在一个重要差别：可以调用函数来执行函数体内的代码。正是这一点让函数与众不同。

Head First: 这听起来了不得，但除了定义和调用外，我不知道还能对函数做什么。

函数: 这就是年收入达六位数的程序员和脚本编写人员的区别。能够像其他值一样看待函数后，便能使用各种有趣的编程结构。

Head First: 你能举例说明吗？一个就可以。

函数: 没问题。假设你要编写一个排序函数。为此，可让它接受两个参数：一个需要排序的集合；一个知道如何对集合中的任意两个条目进行比较的函数。使用JavaScript，你可轻而易举地编写这样的代码。编写一个可用于对各种集合进行排序的函数，并向它传递一个知道如何进行比较的函数，让它比较集合中的条目。

Head First: 哦……

函数: 我说过，这就是年收入达六位数的程序员和脚本编写人员的区别。再说一遍，我们将一个知道如何进行比较的函数传递给另一个函数。换句话说，我们将函数视为值，从而将其作为值传递给函数。

Head First: 除了让人感到迷惑外，这能带来什么好处？

函数: 好处包括工作更轻松、代码更少、更可靠、更灵活、更易于维护，还有更高的薪水。

Head First: 听起来很不错，但我不知道怎么得到这些好处。

函数: 要得到这些好处，你需要做些工作。这绝对是需要你扩展思路的地方。

Head First: 函数，我的脑袋已经很乱，都快要爆炸了。我得去躺一会儿。

函数: 请便。谢谢你的采访！

我们提到过函数在JavaScript中是一等公民吗

如果你学习JavaScript前使用的是传统的编程语言，你可能认为函数就是函数，可以声明和调用它们，仅此而已。

你现在知道，在JavaScript中，函数是值，是可赋给变量的值。你还知道，对于其他类型的值，如数字、布尔值、字符串和对象，可对其做各种事情。例如将它们传递给函数，从函数返回它们，甚至将它们存储在对象或数组中。

实际上，对于这些类型的值，计算机科学家使用了一个术语——**一等值**。对于一等值，你可以做如下事情。

- ❑ 将其赋给变量或存储在数组和对象等数据结构中。
- ❑ 将其传递给函数。
- ❑ 从函数中返回它们。

你猜怎么着？对于函数，也可以做上述所有事情。事实上，在JavaScript中，对其他值可以做的事情，也都可以对函数做。因此，在JavaScript中，函数与你知道的各种值（数字、字符串、布尔值和对象）一样，也是一等值。

下面以更正规的方式定义了一等值。

一等值：在编程语言中，可像对待其他任何值一样对待的值，包括赋给变量、作为实参传递给函数以及从函数中返回。

你将看到，JavaScript函数完全具备一等值所需的资质。接下来咱们花点时间看看在上述各种情况下，函数作为一等值到底意味着什么。在此之前，先提一个小小的建议：不要再认为函数是特殊的，有别于JavaScript中的其他值。将函数视为值可带来很大的好处，本章余下的篇幅将说明其中的原因。

我们始终认为，称之为“可进入任何地方的VIP”更合适，但人家不接受这个名字，所以我们还是采用一等的说法吧。



乘坐头等舱

下次面试时若被问及“是什么让JavaScript函数成为一等公民”，你将凯旋而归。但在庆祝获得新工作前，别忘了到目前为止，你对函数是一等公民的认识都只是纸上谈兵。诚然，你知道对于作为一等公民的函数可以做哪些事情。

- ✓ 可以将函数赋给变量。
- 可以将函数传递给函数。
- 可以从函数返回函数。

← 这一点你已经看到过。

← 这一点即将介绍。

← 这点稍后介绍。

如果你凭借前面的答案获得了好工作，可别忘了我们！巧克力、比萨、比特币什么的，这些礼物我们都很喜欢。



但你知道如何以及何时在代码中使用这些技巧吗？不用担心，下面以向函数传递函数为例来说明这一点。我们将从简单的地方着手。事实上，我们将从一个表示航班乘客的简单数据结构着手：

表示乘客的数据结构：

所有乘客都存储在一个数组中。

← 总共有4名乘客（你可在这个列表中添加朋友和家人）。

```
var passengers = [
  { name: "Jane Doloop", paid: true },
  { name: "Dr. Evel", paid: true },
  { name: "Sue Property", paid: false },
  { name: "John Funcall", paid: true } ];
```

← 每位乘客都由一个包含属性name和paid的对象表示。

↑ 属性name为简单的文本字符串。

↑ 属性paid是一个布尔值，表示乘客是否已买票。

我们的目标是，编写一些代码来检查这个乘客列表，确保满足特定条件后才允许航班起飞。例如，确保没有乘客在禁飞名单上；确保所有乘客都已买票。我们还想将航班的乘客名单打印出来。

考考你的脑力

请想想如何编写代码来完成这三项任务：确保没有乘客在禁飞名单上；确保乘客都已买票；打印乘客名单。

编写处理和检查乘客的代码

对于上述每项任务，通常做法是各编写一个相应的函数：一个检查禁飞名单的函数，一个检查乘客都已买票的函数以及一个打印乘客名单的函数。但如果这样做，你将发现这些函数都大致相同，如下所示。

```
function checkPaid(passengers) {
  for (var i = 0; i < passengers.length; i++) {
    if (!passengers[i].paid) {
      return false;
    }
  }
  return true;
}
```

```
function checkNoFly(passengers) {
  for (var i = 0; i < passengers.length; i++) {
    if (onNoFlyList(passengers[i].name)) {
      return false;
    }
  }
  return true;
}
```

```
function printPassengers(passengers) {
  for (var i = 0; i < passengers.length; i++) {
    console.log(passengers[i].name);
    return false;
  }
  return true;
}
```

这两个函数唯一的差别在于，一个检查乘客是否已买票，而另一个检查乘客是否在禁飞名单上。

print也遍历所有的乘客。相比于其他两个函数，它唯一的不同之处在于，不检查乘客，而是将其传递给console.log；另外，我们不关心这个函数的返回值。

有很多代码是重复的：这些函数都遍历所有的乘客，并对每位乘客进行处理。如果以后需要做其他的检查，结果将如何呢？如果要检查乘客是否关闭了笔记本电脑、是否要从经济舱调到头等舱、是否身体有恙等呢？这将编写大量重复的代码。更糟糕的是，如果将存储乘客的数据结构从简单的对象数组改成了其他东西呢？在这种情况下，可能必须重新编写每个函数。这不太好。

鉴于函数是一等公民，这个小问题很容易解决。解决方法如下：编写一个知道如何遍历乘客的函数，并向它传递一个知道如何执行检查的函数（即检查乘客是否在禁飞名单上、检查乘客是否已买票等）。



下面来做一些准备工作：编写两个函数。第一个函数将一名乘客作为参数，检查他是否在禁飞名单上；如果在，就返回true，否则返回false。第二个函数也将一名乘客作为参数，检查他是否已买票；如果已买票，就返回true，否则返回false。我们编写好了这些函数的函数头，你只需编写它们的代码即可。答案见下一页，可别偷看哦！

```
function checkNoFlyList(passenger) {

}
```

```
function checkNotPaid(passenger) {

}
```

提示：假设禁飞名单上只有一个人，他就是Evel博士。



磨笔上阵

下面来热热身，尝试将一个函数传递给另一个函数。请在心中执行下面的代码，看看它们的结果是什么。继续往下阅读前，务必查看本章末尾的答案。

```
function sayIt(translator) {
    var phrase = translator("Hello");
    alert(phrase);
}

function hawaiianTranslator(word) {
    if (word === "Hello") return "Aloha";
    if (word === "Goodbye") return "Aloha";
}

sayIt(hawaiianTranslator);
```

迭代所有的乘客

我们需要编写一个函数，它将所有乘客作为参数，还将另一个知道如何判断乘客是否满足特定条件（如是否在禁飞名单上）的函数作为参数，如下所示：

函数processPassengers有两个形参，其中第一个为乘客数组。

第二个形参是一个知道如何判断乘客是否满足特定条件的函数。

```
function processPassengers(passengers, testFunction) {  
  for (var i = 0; i < passengers.length; i++) {  
    if (testFunction(passengers[i])) {  
      return false;  
    }  
  }  
  return true;  
}
```

以每次一位的方式迭代所有的乘客。

对于每位乘客，都调用传入的函数。

如果这个函数返回的结果为true，就返回false。换言之，如果乘客未通过测试（例如，还没有买票或者在禁飞名单上），就不允许航班起飞！

如果执行到了这里，就说明所有乘客都通过了测试，因此返回true。

现在，只需编写一些检查乘客的函数就大功告成了。所幸的是，你在前面的“磨笔上阵”练习中编写了这些函数，如下所示：

请注意，这是一位乘客（一个对象）而不是所有乘客（一个对象数组）。

```
function checkNoFlyList(passenger) {  
  return (passenger.name === "Dr. Evel");  
}
```

这个函数检查乘客是否在禁飞名单上。我们的禁飞名单很简单：除Evel博士外，其他人都可乘坐。如果乘客为Evel博士，就返回true，否则返回false（即乘客不在禁飞名单上）。

```
function checkNotPaid(passenger) {  
  return (!passenger.paid);  
}
```

这个函数检查乘客是否已买票。为此，只需检查乘客的属性paid。如果乘客没买票，就返回true。

向函数传递函数

至此，我们编写了一个将函数作为参数的函数（`processPassengers`），还编写了两个可作为实参传递给`processPassengers`的函数（`checkNoFlyList`和`checkNotPaid`）。

现在该将它们整合起来了。音乐响起……

向函数传递函数很容易，只需将函数名用作实参即可。

这里传入的是函数`checkNoFlyList`，因此`processPassengers`将检查每位乘客，看看他们是否在禁飞名单上。

```
var allCanFly = processPassengers(passengers, checkNoFlyList);
if (!allCanFly) {
  console.log("The plane can't take off: we have a passenger on the no-fly-list.");
}
```

只要有乘客在禁飞名单上，这个函数就将返回`false`，因此我们将在控制台中看到这条消息。

```
var allPaid = processPassengers(passengers, checkNotPaid);
if (!allPaid) {
  console.log("The plane can't take off: not everyone has paid.");
}
```

这里传入的是函数`checkNotPaid`，因此`processPassengers`将检查每位乘客，看看他们是否买了票。

只要有乘客没买票，这个函数就将返回`false`，因此我们将在控制台中看到这条消息。

试驾航



为对这些JavaScript代码进行测试，只需将它们添加到一个简单的HTML页面中，再在浏览器中加载该页面。

JavaScript控制台

```
The plane can't take off: we have a passenger
on the no-fly-list.
The plane can't take off: not everyone has paid.
```

看起来航班根本就不能起飞，因为乘客有问题！幸亏进行了检查。





又轮到你了：编写一个函数，将乘客姓名以及是否已买票的信息打印到控制台。将这个函数传递给`processPassengers`，对其进行测试。我们编写了这个函数的函数头，你只需编写其代码即可。继续往下阅读前，务必查看本章末尾的答案。

```
function printPassenger(passenger) {
```

← 在这里编写这个函数的代码。

```
}
```

```
processPassengers(passengers, printPassenger);
```

将这个函数传递给
`processPassengers`
时，它将打印乘客
名单。

世上没有愚蠢的问题

问：难道不能将这些代码都放在`processPassengers`中吗？我们可以在一次迭代中执行所有检查，从而一次性完成所有的检查并打印乘客清单。这样效率不是更高吗？

答：如果代码很简短，这样做也许可行。然而，我们追求的是灵活性。如果以后要在既有的函数中不断地添加检查（如乘客是否都关闭了笔记本电脑）或需求，结果将如何呢？在这种情况下，前面采用的设计可降低修改或增添的复杂性，减少在代码中引入bug的可能性。

问：向函数传递函数时，传递的到底是什么呢？

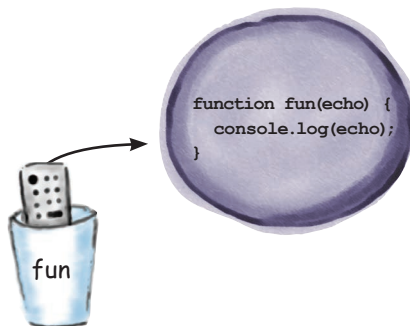
答：传递的是指向函数的引用。可将这种引用视为指针，它指向函数本身的内部表示。引用本身可存储在变量中，可赋给变量，还可作为实参传递给函数。另外，在函数引用后面加上圆括号，将调用相应的函数。



磨笔上阵

下面的代码创建了一个函数，并将其赋给变量fun。

```
function fun(echo) {
  console.log(echo);
};
```



请执行下面的代码，并将其输出写在右边。请先在心中执行这些代码；如果看不懂，再在计算机中执行它们。

```
fun("hello");
```

```
function boo(aFunction) {
  aFunction("boo");
}
```

```
boo(fun);
```

```
console.log(fun);
```

```
fun(boo);
```

```
var moreFun = fun;
```

```
moreFun("hello again");
```

选做题（这涉及
接下来要介绍的
内容）

```
function echoMaker() {
  return fun;
}
```

```
var bigFun = echoMaker();
bigFun("Is there an echo?");
```

继续往下阅读前，务必查看并理解本章末尾的答案，这非常重要！

从函数返回函数

至此，我们演示了作为一等公民的函数的两个特征：将函数赋给变量以及将函数传递给函数，但还未演示如何从函数返回函数。

- ✓ 可以将函数赋给变量。
- ✓ 可以将函数传递给函数。
- 可以从函数返回函数。

下面就来这样做。

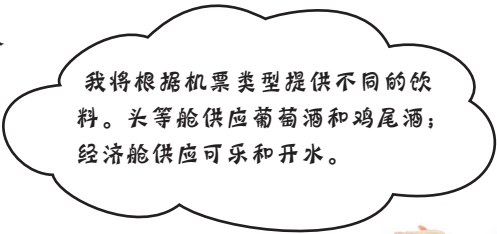


我们来扩展航班示例，探索为何以及在什么情况下要从函数返回函数。为此，我们将给每位乘客添加一个属性ticket，并根据乘客购买的机票类型，将这个属性设置为coach或firstclass。

```
var passengers = [ { name: "Jane Doloop", paid: true, ticket: "coach" },  
  { name: "Dr. Evel", paid: true, ticket: "firstclass" },  
  { name: "Sue Property", paid: false, ticket: "firstclass" },  
  { name: "John Funcall", paid: true, ticket: "coach" } ];
```

添加这个属性后，我们将编写一些代码，来处理空乘人员需要做的各种事情：

这是空乘人员需要向乘客提供的所有服务。



```
function serveCustomer(passenger) {  
  // 让乘客点饮料  
  // 让乘客点餐  
  // 清理垃圾  
}
```

我们首先从点饮料开始实现。



你可能知道，相比于经济舱，为头等舱提供的服务通常有所不同。至少在电影中是这样的……
头等舱乘客可以点鸡尾酒或葡萄酒，而经济舱可能只供应可乐和开水。

编写让乘客点饮料的代码

你编写的代码可能类似于下面这样：

```
function serveCustomer(passenger) {
```

```
  if (passenger.ticket === "firstclass") {
    alert("Would you like a cocktail or wine?");
  } else {
    alert("Your choice is cola or water.");
  }
}
```

```
  // 让乘客点餐
```

```
  // 清理垃圾
```

```
}
```

如果乘客乘坐的是头等舱，就显示一个提示框，询问他要喝葡萄酒还是鸡尾酒。

如果乘客乘坐的是经济舱，就询问他要喝可乐还是开水。

不错。对于简单的代码，这样做挺好：将乘客的属性`ticket`作为参数，并根据乘客购买的机票类型显示一个提示框。来看看这种代码可能存在的一些缺点。确实，让乘客点饮料的代码很简单，但如果问题更复杂时，函数`serveCustomer`将如何呢？例如，可能需要为三类乘客提供服务（头等舱、商务舱和经济舱，而且还可能有第4类乘客——豪华经济舱）。如果提供的饮料种类更复杂呢？如果可选择的饮料随启程或目的机场而异呢？

例如，在前往夏威夷的航班上，头等舱通常只供应麦台鸡尾酒。

如果必须处理这些复杂的因素，函数`serveCustomer`将变得很大。它除让乘客点饮料外，还需处理很多与供应的饮料类型相关的问题。然而，设计函数时有一个不错的经验规则，那就是只让每个函数做一件事，并把这件事做好。



请再次阅读前面两个段落列出的潜在问题，再想想该如何设计代码，让`serveCustomer`只专注于做一件事情，同时让我们能够轻松地扩展饮料供应功能。

编写让乘客点饮料的代码：另一种方式

前面编写的代码不赖，但正如你看到的，饮料供应方式变得更复杂时，这些代码可能出现问题。下面采用另一种方式来重新编写代码：将让乘客点饮料的逻辑放在一个独立的函数中。这样可将所有相关的逻辑放在一个地方，在需要修改这些逻辑时，只需要修改这个地方即可：

我们创建一个新函数`createDrinkOrder`，并将一位乘客传递给这个函数。

```
function createDrinkOrder(passenger) {
  if (passenger.ticket === "firstclass") {
    alert("Would you like a cocktail or wine?");
  } else {
    alert("Your choice is cola or water.");
  }
}
```

我们将在这个函数中实现让乘客点饮料的所有逻辑。

这样，函数`serveCustomer`就不再包含大量让乘客点饮料的逻辑。

现在回到函数`serveCustomer`，将所有让乘客点饮料的逻辑都删除，转而调用这个新函数。

```
function serveCustomer(passenger) {
  if (passenger.ticket === "firstclass") {
    alert("Would you like a cocktail or wine?");
  } else {
    alert("Your choice is cola or water.");
  }
  createDrinkOrder(passenger);
  // 让乘客点餐
  // 清理垃圾
}
```

在`serveCustomer`中，将让乘客点饮料的逻辑都删除……

转而调用函数`createDrinkOrder`。

将传递给`serveCustomer`的`passenger`传递给函数`createDrinkOrder`。

不内嵌让乘客点饮料的逻辑，转而调用一个函数后，代码可读性无疑要好得多。另外，通过将让乘客点饮料的逻辑都放在一个易于找到的地方，修改起来也更方便。然而，别着急对这些代码进行测试，我们又发现了一个问题。

等等，我们需要多次供应饮料

等等，我们刚刚得知，在航班上只供应一次饮料不够。事实上，航班空乘人员提供的服务更像是下面这样的：

```
function serveCustomer(passenger) {
  createDrinkOrder(passenger);
  // 让乘客点餐
  createDrinkOrder(passenger);
  createDrinkOrder(passenger);
  // 播放电影
  createDrinkOrder(passenger);
  // 清理垃圾
}
```

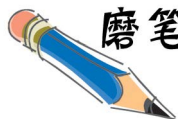
我们修改了函数 `serveCustomer` 的代码，以反映在整个飞行中将多次调用 `createDrinkOrder` 这一事实。



一方面，我们的代码设计得很好，只需多次调用 `createDrinkOrder` 即可；另一方面，每次调用 `createDrinkOrder` 让乘客点饮料时，都需要重新确定乘客的类型，这是不必要的。

你可能会说，不过是几行代码而已。确实如此，但这只是一个简单示例。在实际程序中，如果需要在移动设备中与Web服务通信，以检查机票类型，结果将如何呢？这将是耗时而昂贵的。

不过别担心，作为一等公民的函数正骑着白马来拯救我们。你知道，通过利用从函数返回函数的功能，可解决这种问题。



磨笔上阵

你认为下面的代码是做什么的？你能通过几个示例来说明如何使用这个函数吗？

```
function addN(n) {
  var adder = function(x) {
    return n + x;
  };
  return adder;
}
```

↑
请把答案写在这里。

利用作为一等公民的函数让乘客点饮料

下面来看看在这种情况下，作为一等公民的函数将如何提供帮助。具体的计划如下：对于每位乘客，不多次调用`createDrinkOrder`，而是只调用它一次，并让它返回一个知道如何让这位乘客点饮料的函数。这样，每当需要让乘客点饮料时，都只需调用这个函数即可。

下面首先来重新定义`createDrinkOrder`，使其创建一个让乘客点饮料的函数，再返回这个函数，供我们在需要时使用。

这是函数`createDrinkOrder`的新版本，它返回一个知道如何让乘客点饮料的函数。

```
function createDrinkOrder(passenger) {
  var orderFunction;

  if (passenger.ticket === "firstclass") {
    orderFunction = function() {
      alert("Would you like a cocktail or wine?");
    };
  } else {
    orderFunction = function() {
      alert("Your choice is cola or water.");
    };
  }
  return orderFunction;
}
```

首先，创建一个变量，用于存储要返回的函数。

接下来，执行检查乘客机票类型的条件代码。（这些代码只会执行一次。）

如果乘客乘坐的是头等舱，就创建一个知道如何让头等舱乘客点饮料的函数。

否则，就创建一个知道如何让经济舱乘客点饮料的函数。

返回创建的函数。

下面来重新编写函数`serveCustomer`。我们首先调用`createDrinkOrder`来获取一个知道如何给乘客点饮料的函数，再反复调用这个函数来让乘客点饮料。

```
function serveCustomer(passenger) {
  var getDrinkOrderFunction = createDrinkOrder(passenger);
  getDrinkOrderFunction();
  // 让乘客点餐
  getDrinkOrderFunction();
  getDrinkOrderFunction();
  // 播放电影
  getDrinkOrderFunction();
  // 清理垃圾
}
```

函数`createDrinkOrder`现在返回一个函数，我们将这个函数存储在变量`getDrinkOrderFunction`中。

每当需要让乘客点饮料时，我们都调用`createDrinkOrder`返回的那个函数。

试驾航



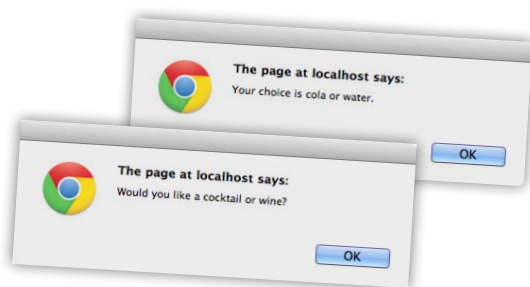
咱们来测试一下这些新代码。为此需要编写一个简单的函数，在其中迭代所有的乘客，并对每位乘客调用`serveCustomer`。将这些代码加入HTML文件后，在浏览器中加载这个文件，让乘客点饮料。

```
function servePassengers(passengers) {
  for (var i = 0; i < passengers.length; i++) {
    serveCustomer(passengers[i]);
  }
}
```

在这里，我们只需迭代数组`passengers`中的乘客，并对每位乘客调用`serveCustomer`。

```
servePassengers(passengers);
```

当然，我们还需调用`servePassengers`以执行其中的代码。这将显示大量的提示框，一定要有心理准备！



世上没有愚蠢的问题

问：调用`createDrinkOrder`时，我们获得了一个函数；要让乘客点饮料，我们必须调用这个返回的函数。我这样理解正确吗？

答：没错。我们首先调用`createDrinkOrder`来获得函数`getDrinkOrderFunction`，它知道如何让乘客点饮料。然后，每当要让乘客点饮料时，我们都调用这个函数。请注意，`getDrinkOrderFunction`比`createDrinkOrder`简单得多。它只是显示一个提示框，询问乘客要点哪种饮料。

问：`getDrinkOrderFunction`怎么知道该显示哪种提示框呢？

答：因为它是我们根据乘客购买的机票类型创建的。再来看一眼`create-`

`DrinkOrder`，它根据乘客购买的机票类型返回相应的函数：如果是头等舱，创建的`getDrinkOrderFunction`显示的提示框将询问乘客要喝头等舱供应的哪种饮料；如果是经济舱，创建的`getDrinkOrderFunction`显示的提示框将询问乘客要喝经济舱供应的哪种饮料。通过根据乘客购买的机票类型返回相应的函数，使得每次需要让乘客点饮料时，都可轻松地调用简单、快速的点饮料函数。

问：这些代码让一位乘客点饮料、为其播放电影等，可空乘人员通常会让更多乘客点饮料、为他们播放电影等，不是吗？

答：我们是在考验你！你通过了考验。你说的一点都没错，这些代码每次对一位乘客调用函数`serveCustomer`。在实际程序中，情况

完全不是这样的。这只是一个演示复杂主题（返回函数）的简单示例，并非十全十美。既然你指出了我们的错误，就请拿出一张纸，完成下面的“考考你的脑力”吧。

考考你的脑力

如果要重写这些代码，让所有的乘客都点饮料、点餐并给他们播放电影，同时避免没完没了地根据乘客购买的机票类型来确定他可以点哪些饮料和餐食，你会如何做呢？你会使用作为一等公民的函数吗？



你的任务是修改前面的代码，以支持第三类乘客：豪华经济舱乘客。对于豪华经济舱乘客，除了可乐和开水外，还可以点葡萄酒。另外，请实现支持下述菜单的`getDinnerOrderFunction`。

头等舱：鸡肉或意大利面。

豪华经济舱：快餐或芝士拼盘。

经济舱：花生饼或椒盐脆饼。

请务必查看本章末尾的答案。另外，别忘了对你编写的代码进行测试。

务必使用作为一等公民的函数来实现这些功能！

Web镇可乐公司

Web镇可乐公司需要有人帮忙管理其产品线代码。为施以援手，我们来看看其用来存储碳酸饮料产品的数据结构：



他们将产品存储在一个对象数组中，其中每个对象都表示一款产品。

```
var products = [ { name: "Grapefruit", calories: 170, color: "red", sold: 8200 },
                  { name: "Orange", calories: 160, color: "orange", sold: 12101 },
                  { name: "Cola", calories: 210, color: "caramel", sold: 25412 },
                  { name: "Diet Cola", calories: 0, color: "caramel", sold: 43922 },
                  { name: "Lemon", calories: 200, color: "clear", sold: 14983 },
                  { name: "Raspberry", calories: 180, color: "pink", sold: 9427 },
                  { name: "Root Beer", calories: 200, color: "caramel", sold: 9909 },
                  { name: "Water", calories: 0, color: "clear", sold: 62123 }
                ];
```

在每款产品中，都存储了产品名、卡路里数、颜色以及每月销售了多少瓶。

我们亟需有人对这些产品进行排序。我们需要按每种属性对这些产品进行排序：名称、卡路里数、颜色和销量。当然，我们希望以尽可能高的效率完成这些工作，同时提供足够的灵活性，以便能够以众多不同的方式进行排序。

Web镇可乐公司的分析师





Frank: 大家好, 我接到了Web镇可乐公司的电话, 让我们帮忙对他们的产品数据进行排序。他们希望能够按任意属性对产品进行排序, 如名称、销售的瓶数、饮料颜色、每瓶的卡路里数等, 同时希望排序方案足够灵活, 以便未来也能够按其他属性进行排序。

Joe: 当前他们是如何存储产品数据的呢?

Frank: 每款产品都是一个存储在数组中的对象, 包含表示名称、销量、卡路里数等信息的属性。

Joe: 明白。

Frank: 我首先想到的是实现一种简单的排序算法。Web镇可乐公司的产品不多, 因此要求排序算法简单。

Jim: 哦, 我想到了一种更简单的办法, 但这要求你具备作为一等公民的函数的知识。

Frank: 只要简单我就喜欢! 但这与作为一等公民的函数有什么关系呢? 听起来好像很复杂。

Jim: 一点都不复杂。只需编写一个知道如何对两个值进行比较的函数, 再将它传递给另一个实际进行排序的函数即可。

Joe: 这个函数的工作原理到底是什么样的呢?

Jim: 你不用管排序的问题, 只需编写一个知道如何对两个值进行比较的函数。例如, 假设要根据销量这个产品属性进行排序, 可这样编写这个函数:

```
function compareSold(product1, product2) {  
    // 执行比较的代码  
}
```

这个函数将两款产品作为参数, 并对它们进行比较。

详细的代码细节稍后再说。这里的关键是, 有了这样的函数后, 你只需将其传递给一个排序函数, 这个排序函数将为你完成其他所有的工作——它只要求你让它知道如何对产品进行比较。

Frank: 等等, 这样的排序函数在哪里呢?

Jim: 它实际上是一个可对任何数组调用的方法。你可以对数组`products`调用方法`sort`, 并将上述比较函数传递给它。这样, 当方法`sort`执行完毕后, 数组`products`便排好序了, 而排序标准就是`compareSold`使用的排序标准。

Joe: 也就是说, 如果要按销量排序, 由于销量都是数字, 函数`compareSold`只需确定哪个值更大或更小即可?

Jim: 没错。下面来更深入地了解一下数组排序的工作原理。

数组方法sort的工作原理

JavaScript数组包含一个sort方法，它根据一个知道如何对两个数组元素进行比较的函数对数组进行排序。这大致说明了sort方法的工作原理以及比较函数在其中的作用。排序算法为大家所熟知并被广泛地实现，更重要的是，排序代码可重用于任何数据集。但有一点需要注意：要对特定的数据集进行排序，排序代码必须知道如何对这些数据进行比较。就拿数字列表、名称列表和对象列表来说吧。对它们进行排序时，比较方式取决于列表项的类型：对于数字，我们使用<、>和==进行比较；对于字符串，我们按字母顺序进行比较（在JavaScript中，也可以使用<、>和==进行比较）；而对于对象，我们必须采用自定义的方式根据对象的属性进行比较。

对Web镇可乐公司的产品数组进行排序之前，咱们先来看一个简单的示例。在这个示例中，我们将使用方法sort将一个简单的数字数组按升序排列。这个数组如下：

```
var numbersArray = [60, 50, 62, 58, 54, 54];
```

为此，我们需要编写一个函数，它知道如何比较这个数组的两个值。这是一个数字数组，因此我们的函数需要比较两个数字。假设要按升序排列，sort方法要求我们的函数在第一个数字大于第二个数字时返回一个大于0的值，在它们相等时返回0，并在第一个数字小于第二个数字时返回一个小于0的值，如下所示：

上述数组包含的是数字，因此我们需要每次对两个数字进行比较。

```
function compareNumbers(num1, num2) {
  if (num1 > num2) {
    return 1;
  } else if (num1 === num2) {
    return 0;
  } else {
    return -1;
  }
}
```

我们首先检查num1是否大于num2；如果是，就返回1。

如果它们相等，就返回0。

最后，如果num1小于num2，就返回-1。



提示

JavaScript数组提供了很多方法，可用来以各种方式操作数组。要全面了解这些方法及其用法，David Flanagan所著的《JavaScript权威指南》是不错的参考。



磨笔上阵

你知道，传递给sort的比较函数需要根据比较结果返回一个大于0、等于0或小于0的数字：如果第一项大于第二项，就返回一个大于0的值；如果第一项等于第二项，就返回0；如果第一项小于第二项，就返回一个小于0的值。

利用这些知识，你能够重写对两个数字进行比较的compareNumbers，让其包含更少的代码吗？

继续往下阅读前，请务必查看本章末尾的答案。

整合起来

编写好比较函数后，只需对numbersArray调用方法sort，并将这个函数传递给它即可，如下所示：

```
var numbersArray = [60, 50, 62, 58, 54, 54];
```

```
numbersArray.sort(compareNumbers);
```

对数组调用方法sort，并将函数compareNumbers传递给它。

```
console.log(numbersArray);
```

方法sort执行完毕后，这个数组将按升序排列。为验证这一点，我们将这个数组显示到控制台。

按升序排列的数组。

请注意，方法sort是破坏性的，因为它就地修改数组，而不是返回一个排序后的新数组。

JavaScript控制台

[50, 54, 54, 58, 60, 62]



方法sort为何按升序排列数组numbersArray呢？因为我们在比较函数中返回1、0和-1时，实际上是在这样告诉sort方法：

如果返回值为1，就将第一项放在第二项后面；

如果返回值为0，就说明两项相等，可保留它们的位置不变；

如果返回值为-1，就将第一项放在第二项前面。

要修改代码，以便按降序排列，只需反转上述逻辑，让返回值1表示将第二项放在第一项后面，返回值-1表示将第二项放在第一项前面（返回值0表示保持位置不变）。请编写一个按降序排列的新比较函数：

```
function compareNumbersDesc(num1, num2) {
    if (____ > ____ ) {
        return 1;
    } else if (num1 === num2) {
        return 0;
    } else {
        return -1;
    }
}
```