## switch 语句

### 初识 switch 语句

在前文中，我们已经学习并尝试了**分支结构**。我们将在几个示例中进一步了解它的使用。

设想一下，我们有一个有奖猜数活动，参与者可以随机输入一个 1 和 3 之间的整数，我们告诉参与者是否猜中，或与答案相差多少。利用 if 语句，我们可以这样写：

let number = +prompt("请输入 1 和 3 之间的整数");  
  
if (number === 1) {  
 alert("太小了！");  
} else if (number === 2) {  
 alert("刚刚好！");  
} else if (number === 3) {  
 alert("太大了！");  
} else {  
 alert("哦，数字超出范围！");  
}

我们共需依次比较三种情况，根据每种情况作出相应回应。而不在我们考虑范围内的“其他情况”则放入 else 子句中处理。在需考虑的情况较少时，使用 if 语句以此判断尚可应付需求，但当我们需要考虑到情况变得多，使用 if 语句就会显得力不从心。此外，如果我们要用相同的方式分别处理不同的情况，使用 if 语句会相当麻烦。

为此，JavaScript 中提供了另一种实现分支结构的语句：*switch 语句*。

switch 语句的一般格式如下：

switch (表达式) {  
 case 情况1: 处理语句1; break;  
 case 情况2: 处理语句2; break;  
 case 情况3: 处理语句3; break;  
 //...  
 default: 默认处理语句;  
}

switch 语句遵循这样的执行顺序：

* 每个 case 关键字都用于标明一个情况。
* JavaScript 依次查看每种情况，然后对表达式求值，查看表达式的值是否能够与这种情况匹配。
* 如果匹配（严格相等），就执行冒号后的处理语句。如果以 break; 结尾，那么终止 switch 语句。
* 否则，继续查看下一个情况，以此类推。
* 当所有列举的情况都查看完之后，如果有一个 default 标志，就执行它的处理语句，作为默认情况。

前面的例子可以使用 switch 语句改写如下：

let number = +prompt("请输入 1 和 3 之间的整数");  
  
switch (number) {  
 case 1:  
 alert("太小了！");  
 break;  
 case 2:  
 alert("刚刚好！");  
 break;  
 case 3:  
 alert("太大了！");  
 break;  
 default: alert("哦，数字超出范围！");  
}

练习 4.2.1

1. 找到一个可以使用 switch 进行处理的生活中的例子，并编写程序实现它。

### 使用 break

让我们回忆一下 if 语句的机制：

对每个条件进行查看，如果成立，就执行相应处理语句，然后结束 if 语句。

当我们使用 switch 语句的时候，我们并不一定希望在找到匹配情况后，仍然继续匹配其余情况。但是，switch 语句有个特点：**当它匹配到一种情况之后，会继续执行之后其他情况的处理语句，**甚至包括 default 。

为了模仿 if 语句“干完事就走人”，不拖泥带水，我们需要在每一个 case 的处理语句后添加一行 break; 。

let fruit = prompt("请输入一种水果的名字：");  
  
switch (fruit) {  
 case "橙子":  
 alert("橙子卖 0.59 美元。");  
 break;  
 case "苹果":  
 alert("苹果卖 0.32 美元。");  
 break;  
 case "香蕉":  
 alert("香蕉卖 0.48 美元。");  
 break;  
 case "车厘子":  
 alert("车厘子卖 3 美元");  
 break;  
 case "芒果":  
 case "桑葚":  
 alert("芒果和桑葚卖 2.79 美元。");  
 break;  
 default:  
 alert("抱歉，本店没有水果" + fruit + "。");  
}

switch 语句每当遇到匹配的情况，执行相应处理语句后，就会终止。

如果 break; 后面还有处理语句，那么它不会被执行，因为 break; 已经起到了终止 switch 语句的作用。

我们使用 switch 语句时一般都会添加 break; ，这是一种良好习惯。没有添加 break; 以至于所有 case 都会被查看一遍的 switch 语句被称为 *switch 穿越*，可能会引发一些问题，我们将在下文中看到 switch 穿越所展现出的效果。。

### 关联操作

这个例子阐述了利用 switch 语句进行的关联操作。如前文所述，当我们输入一个数字并匹配之后，它会执行其后，一直第一个到 break; 之前的所有处理语句。

let value = 1;  
let output = "输出： "  
switch (value) {  
 case 10:  
 output += "所以";  
 case 1:  
 output += "你的";  
 output += "名字";  
 case 2:  
 output += "是";  
 case 3:  
 output += "什么";  
 case 4:  
 output += "？";  
 alert(output)  
 break;  
 case 5:  
 output += "！";  
 alert(output);  
 break;  
 default:  
 alert("请选择一个 1 到 6 之间的数字！");  
}

尝试改变 value 的值，相信你会对 switch 的机制有更深的体会。如果你不是特意为了制造出这种效果，请记得：务必在每种情况的相应处理语句末尾添加 break;。

练习 4.2.3

1. 使用 switch 语句的穿越特性，写一个程序，模拟一个会根据指令来问好的机器人。

### 分组

生活中有什么事情会像在 1 ~ 3 之间猜一个数字这样简单而乏味呢——让我们将目光投向更“实际”的问题。

现在我们摇身一变成了动物保护专家，向好奇的小朋友普及动物保护的知识，告诉他们哪些动物已经灭绝而湮没在历史中，哪些动物在悬崖边上苦苦挣扎，哪些动物暂时毫无危险。

let animal = prompt("请输入一种动物的名字：");  
switch (animal) {  
 case "猫":  
 case "金鱼":  
 case "鸵鸟":  
 case "企鹅":  
 case "火鸡":  
 case "马":  
 alert(animal + "没有危险！");  
 break;  
 case "大象":  
 case "熊猫":  
 case "江豚":  
 alert(animal + "处于危险之中，我们要一起保护它们。");  
 break;  
 case "渡渡鸟":  
 case "恐龙":  
 case "象鸟":  
 alert(animal + "已经灭绝。");  
 break;  
 default:  
 alert("我没听过这种动物的名字。");  
}

我们对小朋友提出的问题进行了分类，只用一行处理代码，统一回应相同性质的提问。

练习 4.2.4

1. 思考生活中有哪些实际问题可以使用 switch 进行分组处理。

### 使用动态条件

我们用一个小示例来结束本节。

let i = Math.floor(Math.random() \* 7)  
switch (i) {  
 case ((i >= 0 && i <= 5) ? i : -1): // 如果 0 ≤ i ≤ 5，那么待匹配的值为 i ，否则为 -1。  
 alert("0 ~ 5");   
 break;  
 case 6:  
 alert("6");  
 break;  
}

如上所示，你可以在 switch 语句的 case 中进行一些运算，以此呈现出“动态”的匹配效果。