阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY

乘风者计划
THE WIND RIDERS PROGRAM

JS零基础入门教程(上册)

乘风者系列图书

CHENWENYANG

陈文阳

乘风者计划专家博主
高校软件技术专业教师

JS零基础入门教学

JavaScript实战案例解析

JS常用结构、工具及语法汇总详解

理论与实践的结合，零基础JS入门最全指南

关注【程序员大阳】

学习更多零基础教程

阿里云开发者“藏经阁”

海量电子手册免费下载

目录

零基础 JavaScript 入门教程(1)--走进 JavaScript 的世界

1. 前言

从今天开始，我们就进入一个全新的编程阶段：JavaScript 语言的学习。JavaScript 又简称 JS，后面我们都用 JS 来代指 JavaScript 了。

虽然 JS 是一门独立的语言，但是在初学者阶段，我们还是在浏览器中运行、调试 JS，这样比较方便。强烈建议学习 JS 之前，先学习 HTML 和 CSS。查看我的《零基础 HTML 入门教程》和《零基础 CSS 入门教程》来进行前置学习。

2. JS 的历史

JS 诞生于 1995 年，现在已经 26 岁了，正值壮年，想必比我们很多读者的年龄都大。它并不是一个新鲜事物，刚诞生的时候也不算特别出名、特别流行。

但是随着互联网的发展，尤其是 Ajax 技术的兴起，JS 语言具备了席卷互联网浏览器的力量。可以这么说吧，网页前端开发的核心就是 JS。一个 JS 语言学的好的人，绝对是不愁找到一个理想的工作的。

3. JS 的地位

JS 在网页前端开发中占据了绝对的统治地位，与后端开发语言 C/C++/C#/Java/Python 百花齐放不同，前端编程语言可以说一家独大，都是用 JS 的。

一些大家可能耳熟能详的知名框架 jQuery、BootStrap、Vue、React，都是基于 JS 进行封装的。

所以学习网页开发，学习 Web，绝对绕不开 JS，JS 在前端的地位，就是至尊王者。

4. JS 与 HTML、CSS 的区别

之前在讲 HTML 和 CSS 的时候，我们举过一个字。

HTML 表示内容，就像家里有哪些家具家电，比如床、茶几、沙发、餐桌、电视、屋门。

CSS 表示内容的样式，比如家具家电的颜色、尺寸、位置。

那么还缺少啥呢，还缺功能，比如电视机，它不光是个电视，不光有颜色、尺寸、摆放位置，它还能打开、能关闭、能播放、能换台。遥控器一按，咔嚓就打开了，这就是它的功能。

再比如屋门，它能干啥？它能打开，能关闭，能上锁，能开锁，这就是它的功能。

这些功能部分，或者说交互部分，就是 JS 负责的。

大家想必能发现，之前 HTML、CSS，基本都是表述的静态的外观，而 JS，则负责处理动态的交互。

5. JS 是真正的编程语言

之前我们学习的 HTML、CSS，其实不算正儿八经的编程语言，他们更像是一种设计语言，用一些规则来描述内容、描述样式。

而 JS 则完全不同，它是一门正儿八经的编程语言，它有变量、有数据类型、有流程控制、有对象有数组，有异常处理。它的能力千变万化，是他就是他，是他就是他，少年英雄 小哪吒！

开个玩笑，因为大家还没接触真正的编程语言，所以大家就了解下，JS 就像哪吒，拥有闹海的强大能力！

6. JS 的组成部分

JS 其实有三大块，一个是语言核心，一个是 BOM，一个是 DOM，语言核心就是 JS 作为一门编程语言基本的功能部分，BOM 就是它拥有的操作咱们浏览器的能力，DOM 就是它操作我们网页文档 HTML 的能力。

这块大家简单了解下，以后会具体一点一点的学习。

7. 小结

JS 来了，你准备好了吗。

零基础 JavaScript 入门教程(2)--在网页中使用 JS

1. 前言

上一篇，给大家聊了很多关于 JS 的事情，大家想必已经迫不及待，想知道如何在网页中使用 JS。

本篇就来介绍下。

2. 网页中使用 JS

还记得如何在网页中使用 CSS 么，可以这样：

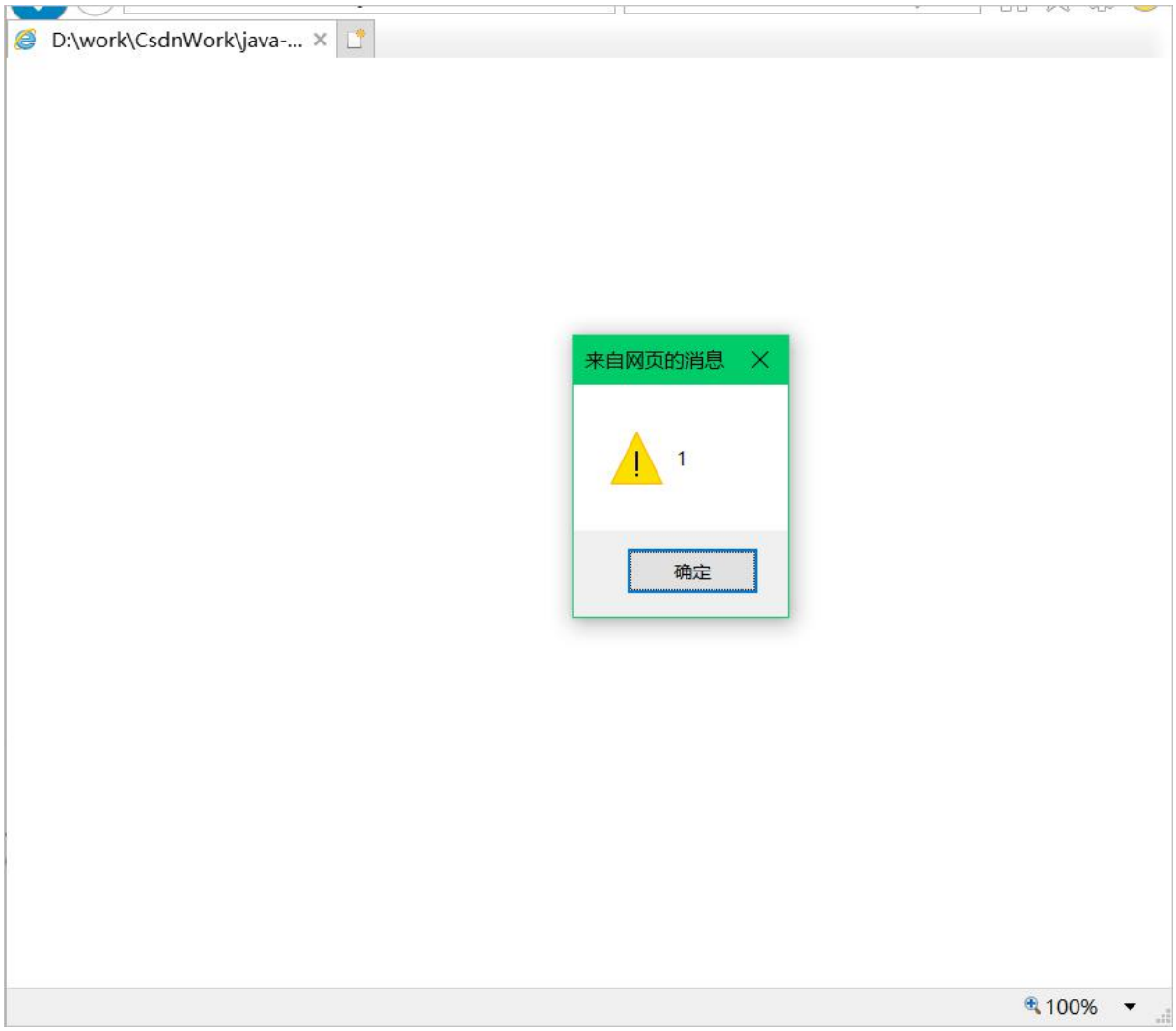
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <style>
        body {
            background-color: grey;
            color: white;
        }
    </style>
</head>
<body>
    你好
</body>
</html>
```

我们通过在头部，放置了<style>标签，然后就可以在<style>标签中编写 CSS 代码了。

差不多的是，我们可以通过在头部，放置<script>标签，来编写 JS 代码，如下：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <style>
        body {
            background-color: grey;
            color: white;
        }
    </style>
    <script>
        alert(1);
    </script>
</head>
<body>
    你好
</body>
</html>
```

<script>和</script>中间，就是我们编写的 js 代码了。alert(1);这句代码的意思是在网页上显示一个弹窗，弹窗的内容是 1。所以上面代码效果如下：



我们的 JS 已经在网页中成功运行了！

3. 小结

本篇只要掌握，可以通过 script 标签，在网页中引入 JS 代码就 OK 了，至于 JS 代码如何编写，这是我们以后需要学习的内容。

零基础 JavaScript 入门教程(3)--网页代码执行顺序与 script 标签的位置

1. 前言

上一篇讲解了在网页中，如何通过 script 标签，插入 JS 代码。

本篇来揭示下，网页上的代码，到底是如何执行的。然后根据执行的情况，我们的 script 标签，放在哪里是最合适的。

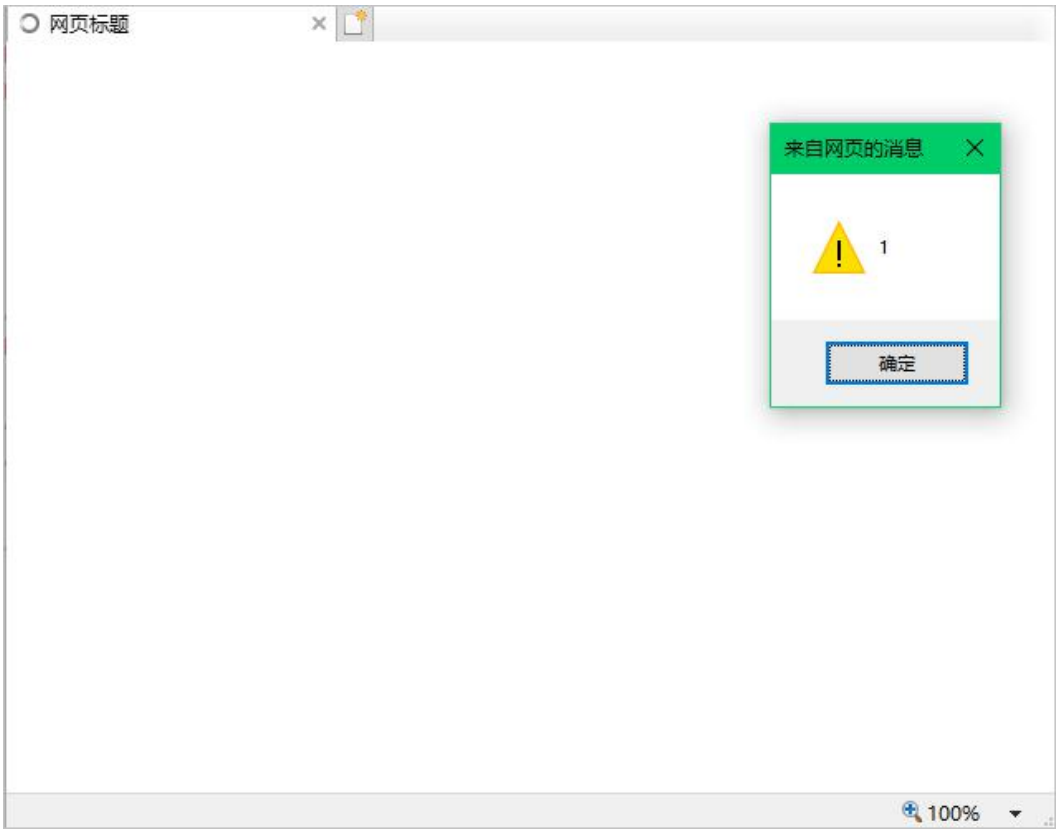
2. 代码执行顺序

首先查看下面的代码：

```
<!DOCTYPE  html>
<html>
<head>
    <meta  charset="utf-8">
    <title>网页标题</title>
    <script>
        alert(1);
    </script>
</head>
<body>
    你好
    <script>
        alert(2);
    </script>
```

```
</body>
</html>
```

我们设置了网页的标题，通过 JS 代码弹窗显示数字 1，网页内容区域是你好。我们打开该网页，发现效果如下：



从上图我们可以清晰的发现，网页已经加载了标题，执行了 alert(1);代码显示了弹窗，但是 body 区域的你好却没有显示出来。

这是为何呢？

其实非常简单，**我们的浏览器在处理网页时，是自上而下，依次加载网页代码的**，这点非常重要，所以在上面的例子中，先显示了网页，然后执行了 alert(1);，此时由于我们没有点击确定，所以 alert(1);并未执行完成，所以下面的 body 部分浏览器并未加载。所以在点击确定之前，body 区域是无法显示内容的。

3. script 标签的位置

根据上面的讨论，如果将 script 标签放到 head 区域，则会导致网页先加载 script 里面的代码，然后再加载 body 区域。

如果 script 里面的代码比较多，执行时间比较长，那么 body 就会等待一段时间才能显示出来。

此时网页会显示一片空白，用户体验无疑非常糟糕。

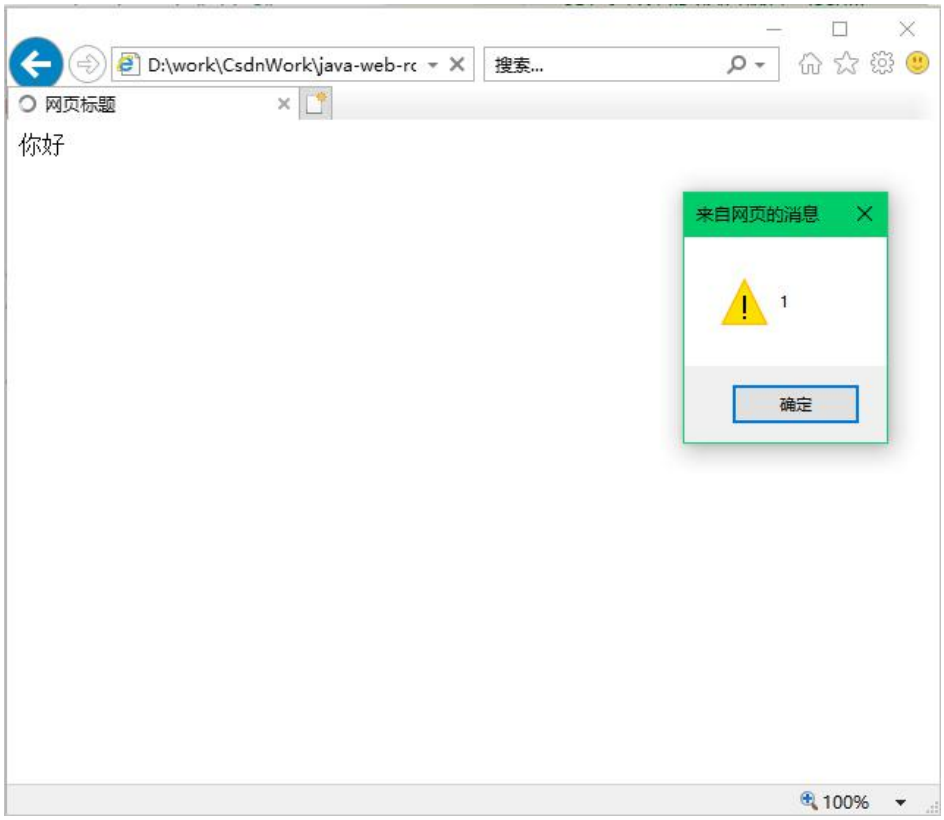
所以我们推荐的做法是：将 script 标签放到 body 内部最后边，也就是贴近</body>的前方。

如下：

```
<!DOCTYPE  html>
<html>
<head>
    <meta  charset="utf-8">
    <title>网页标题</title>
</head>
```

```
<body>
    你好
    <script>
        alert(1);
    </script>
</body>
</html>
```

我们预览该代码，可以发现在显示弹窗前，已经加载 body 的你好了。



4. 小结

网页代码都是自上而下，浏览器一次加载执行。

所以 script 放置的最佳位置，是 body 的最后头。

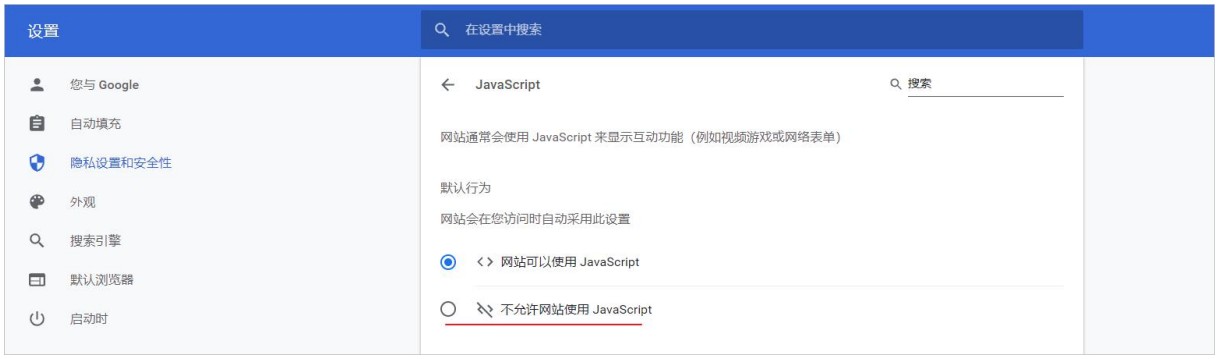
零基础 JavaScript 入门教程(4)--浏览器禁用 JS 脚本时处理方法

1. 前言

就目前的现状而言，不支持 JS 的浏览器，几乎已经找不到了。但是也不排除这种情况的存在性。

另外比较常见的情况是，浏览器禁用了 JS 功能，这个还是比较常见的。

例如在 Chrome 浏览器中，如下图，选择【不允许网站使用 JavaScript】即可禁用 JS：



那么，当浏览器不支持、或者禁用了 JS 时，咋办呢。

此时我们应该至少提示用户，您现在浏览器不支持或者禁用了 JS。

2. 使用 noscript 标签

我们可以使用 noscript 标签，来加载网页不支持 JS 脚本时提示信息，如下：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
</head>
<body>
    <noscript>
        您好，您的浏览器不支持或者禁用了 JavaScript，导致本网站某些功能无法正常使用，请知晓。
    </noscript>
</body>
</html>
```

当浏览器没禁用 JS 时，上述文本内容不会显示。当我们禁用了 JS 时，上述内容就会在网页显示出来。

大家可以使用 Chrome 浏览器尝试下。

3. 小结

由于现在几乎所有网站，都采用了 JS 技术，所以一般也没人会去禁用 JS 功能。

所以本节课的内容实用性不强，之所以讲解本节课，是让大家理解，浏览器本身默认支持 JS，但是也可以选择禁用 JS 功能。

零基础 JavaScript 入门教程(5)--理解 JS 中的语句

1. 什么是语句

本篇我们来学习 JS 中的语句，理解 JS 语句的写法。

首先在自然语言，例如我们的汉语中，句子就是一句完整的话，一般一个汉语句子使用句号、问号或者叹号结尾。

英语也差不多，英语的句子是由单词组成的，一般也是由句号、问号或叹号结尾。

那么自然语言，为什么要有句子呢。就是因为每个句子，是整个内容的一小段，我们一句话一句话的说，别人才好一句话一句话的理解。如果不分句子直接说一大堆没有停顿，谁能理解的了？

同样，JS 语言也是由语句组成的，JS 分成一个一个的语句，计算机才能一句一句的理解执行。

2. JS 语句分割

上面我们讲了，自然语言的句子，一般是使用句号、问号、或者叹号结尾，也就是说，当我们看到句号、问号或者叹号时，就知道一个句子结束了，这些符号就是自然语言语句的分割符号。

JS 语言中，使用分号;作为语句的分割。之前我们学过了 `alert(1)` 表示在网页上弹窗显示数字 1，那么完整的语句应该写作 `alert(1);`，注意最后有一个分号，表示当前语句结束了。

所以我们可以编写如下代码，让网页先弹窗显示 1，再弹窗显示 2。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
</head>
<body>
    <script>
        alert(1);alert(2);
    </script>
</body>
</html>
```

此时我们打开网页，可以看到网页会先后弹窗 1，然后弹窗 2。

那么如果我们不用分号分割，代码如下：

```
<body>
    <script>
        alert(1)alert(2)
    </script>
</body>
```

我们运行代码，会发现网页并不能正常弹窗，这是因为代码没有正常分割，我们计算机浏览器理解不了这样的代码了。

3. JS 语句分行

虽然下面的写法是正确的：

```
<body>
    <script>
        alert(1);alert(2);
    </script>
</body>
```

但是如果把很多行代码都放到一行，会显得很凌乱。

我们从代码格式化、美观易懂的角度出发，习惯上，每行我们只放一句 JS 代码如下：

```
<body>
    <script>
        alert(1);
        alert(2);
    </script>
</body>
```

大家以后在编写 JS 程序时，也应该记住，每行只写一句 JS 代码。

4. 没有分号的情况

看如下代码：

```
<body>
    <script>
        alert(1)
        alert(2)
    </script>
</body>
```

虽然没有写分号，但是两个语句之家分行了，而且我们使用浏览器查看该网页，也能正常弹窗展示数字。

这种写法是不符合规范的，不建议使用。另外这种方式虽然能正常运行，但是浏览器在分析这样的代码时，需要花费额外的时间去推测分号的位置，这会浪费浏览器的性能。

也就是说，如果有分号，浏览器会很容易确定语句如何分割。如果没有分号，浏览器得尝试根据代码的情况分割语句。

综上所述，虽然有时候没有分号，JS 也能正常运行，但是强烈建议使用分号。

5. 小结

JS 中的语句使用分号分割，建议每行代码只有一条语句，且都使用分号结束语句。

零基础 JavaScript 入门教程(6)--JS 之使用开发者工具

1. 前言

在上一篇中，我们曾经演示过一个错误实例：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
</head>

<body>
    <script>
        alert(1)alert(2)
    </script>
</body>

</html>
```

两句代码，由于没使用分号分割，导致浏览器无法识别，所以程序无法正常运行。

这段代码比较简单，我们比较容易分析它的问题。

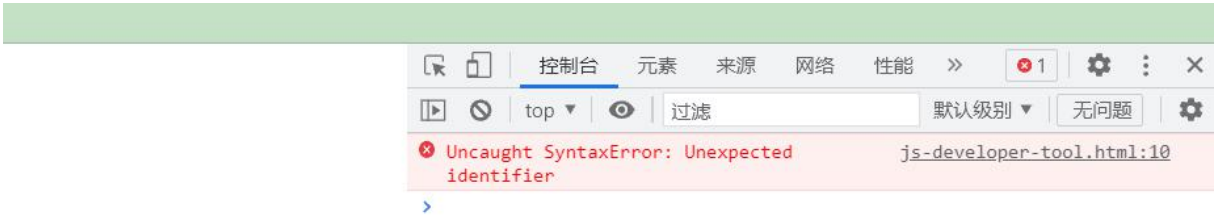
但是代码如果复杂起来，有成千上万行代码，当其中一句有问题时，我们该如何知道是哪里出了问题，出了什么问题呢？

此时就要借助浏览器自带的开发者工具了，既然浏览器要执行 JS 代码，所以浏览器肯定知道哪些语句不能正常执行。所以浏览器提供了开发者工具，以便告知开发者的问题。

当然开发者工具的功能远不止如此，我们暂时先学习下开发者工具的控制台功能，开发者工具的控制台，可以查看错误提示信息。

2. 打开开发者工具

在 Chrome 浏览器或 IE 浏览器中，都可以直接按【F12】打开开发者工具，如下图是 Chrome 浏览器的开发者工具控制台：



然后下图是 IE 浏览器的控制台：

可以看到，两个控制台都给出了错误提示信息，然后也告诉开发者，代码的问题是第 10 行。

我们 VSCode 每行代码左边有行号，我们看下，如下是我的 VSCode 编写代码的截图，第 10 行正好是有问题的代码位置。

3. 小结

平常我也是经常编写 JS 代码，习惯性上就是在查看网页 JS 运行效果时，同步打开开发者工具控制台，查看代码运行情况。

所以建议大家也养成这样的习惯，学会使用开发者工具的控制台，查看代码运行信息。

零基础 JavaScript 入门教程(7)--JS 之注释

1. 前言

在之前 HTML、CSS 注释的讲解中，我们已经充分理解到：注释的内容，计算机不再识别并运行，仅仅起到一个提示作用。

注释一般是给程序员自己看的，属于一种补充说明，没有实际执行的效果。

2. 回顾 HTML 注释

HTML 注释格式如下：

```
<body>
    《春晓》
    <!-- hr 表示一个水平分割线 -->
    <hr>
    春眠不觉晓，处处闻啼鸟。夜来风雨声，花落知多少。
</body>
```

其中<!-- hr 表示一个水平分割线 -->部分即为注释，浏览器不会识别并运行这段代码。HTML 注释就是被<!--和-->包裹的部分。

3. 回顾 CSS 注释

CSS 注释格式如下：

```
<style>
    /* 蓝色文本 */
    .text-blue {
        color: blue;
    }
</style>
```

其中/* 蓝色文本 */部分即为 CSS 注释，浏览器不会识别和运行这行代码。CSS 注释就是被/*和*/包裹的部分。

4. JS 单行注释

JS 的注释有两种：单行注释和多行注释，先看下面的例子：

```
<script>
    // alert(1);
    alert(2);
</script>
```

在上面的示例中，//表示单行注释，其后的内容是注释，浏览器不会执行。而 alert(2);前面没有//，会正常运行。

5. JS 多行注释

JS 多行注释的语法跟 CSS 相同，使用/* */来处理，例如：

```
<script>
    /*
        这是多行注释中间的部分
        所以不论我这里写多少行
        都被注释掉了
        不会执行的，放心吧，哈哈
    */
    alert(3);
</script>
```

中间的几行汉子，都被注释了，因为被多行注释包裹。

6. 小结

注释很重要，之所以先讲注释，是因为后面的例子，我们为了更好的让大家学习理解，使用了大量的注释来做说明。

所以大家要明白两种注释的意思。

零基础 JavaScript 入门教程(8)--JS 之输出语句

1. 前言

之前我们只讲过一个 JS 语句：alert，用于弹窗显示信息。一般这种程序显示信息的语句，可以称之为输出语句，意思是程序输出信息给用户看。

JS 常用的输出语句有 3 种，今天逐一介绍下。

2. 弹窗输出

可以使用 alert()语句进行弹窗输出，括号中间的内容，即为弹窗显示的信息。此处需要格外注意的是，如果输出数字，可以直接写到括号中，如果要输出英文字母或者汉字，则需要使用双引号将英文或汉字包裹起来。至于为何要使用双引号包裹，我们后续章节会详细解析，现在我们只需要记住就 OK 了。

如下示例：

```
<script>
    // 弹窗输出数字：123
    alert(123);
    // 弹窗输出汉字：你好啊，因为是汉字，所以需要双引号包裹
    alert("你好啊");
</script>
```

运行代码后，会先弹窗显示数字 123，然后弹窗显示汉字你好啊。

3. 网页内容输出

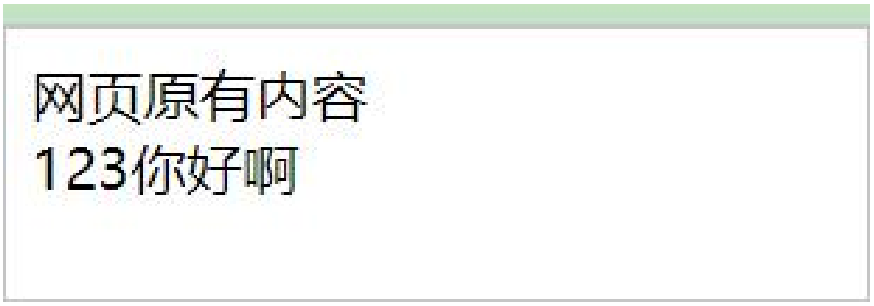
可以通过 document.write()语句，将括号中的信息输出到网页内容部分。

同样，当括号内是数字时，可以直接写数字，而当括号内是英文或者汉字时，需要用双引号包裹起来。

如下示例：

```
<body>
    网页原有内容<br>
    <script>
        // 将 123 输出到网页内容
        document.write(123);
        // 将你好啊输出到网页内容
        document.write("你好啊");
    </script>
</body>
```

body 区域已经有网页原有内容
，然后又通过 document.write 向网页内容区域输出了 123 和你好啊，所以最终运行效果如下：



4. 控制台输出

一般来说，弹窗输出用于给用户展示一些重要提示信息，例如登录用户名或者密码错误。

而使用 document.write 的方式，输出到网页内容区域，这种用法很少，因为直接在 body 里面写内容岂不是更方便，为啥还要用 document.write，多此一举。

还有一种输出方式，其实是前端开发人员最常用的，就是开发者工具的控制台输出。也就是输出的信息不在网页上显示，而是显示到控制台，这种方式一般是程序员调试使用的。

如下代码：

```
<script>
    // 将 123 输出到控制台
    console.log(123);
    // 将你好啊输出到控制台
    console.log("你好啊");
</script>
```

运行后，我们按【F12】打开控制台，如下图：

可见，console.log()会将括号中间的信息，输出到控制台显示。

5. 小结

alert 用来弹窗提示，但是浏览器提供的弹窗太丑了，所以并不常用。

document.write 用于输出到网页内容区域，这种方式不如直接在 body 中写内容方便，所以也不常用。

console.log 可以将信息打印到控制台，网页上并不会显示，程序员会通过该语句调试程序，使用频率还是很高的。

零基础 JavaScript 入门教程(9)--JS 之字面量详解

1. 前言

本篇来介绍一个非常非常重要的概念——字面量。不光在 JS 编程语言中会有字面量的概念，在 C/C++/C#/Java/Python 等语言中，同样存在字面量的概念，可以说它是编程语言的重要基石概念。

2. 何为字面量

字面一词，比较好理解，就是文字表面上的东西，没有什么内在含义。而量，一般指数量、重量的衡量。

在 JS 语言中，字面量的意思，就是所见即所得，字面量真正的含义就是所看到的东西。

我们举一个例子：

```
<script>
    //字面量 123
123
    //字面量 456
    456
</script>
```

上面的数字 123 和 456，就是表示的两个数字，就是它字面的意思，所以叫字面量。

3. 什么不是字面量

这样一说，好像这节课啥也没讲，123 本来就是 123 吗，还搞出一个字面量的概念干啥。

不要着急，再看一个例子：

```
<script>
    alert(123);
</script>
```

这个例子只有一行代码 alert(123);，其中 123 是字面量，就代表数字 123。

但是 alert 就不是字面量了，它不是一个普通的单词，而是一个功能代码，它的作用是将后面括号的内容弹窗输出。

4. 字符串字面量

那么我就想要一个普普通的单词 alert 怎么办，OK，为了区分英文单词是字面量，还是特定功能的程序代码，JS 语言的发明者想了个办法，就是直接写的单词是程序代码，而用双引号包裹起来的，是字面量：距离如下：

```
<script>
    // 程序代码
    alert(123);
    // 字面量
    "alert"
</script>
```

5. 字面量的类型

到此，我们学过两种字面量了，数字字面量，字符串字面量：

```
<script>
    // 数字字面量
    123
    // 字符串字面量
    "alert"
</script>
```

大家一定要理解，为啥字符串字面量要用引号包裹起来，是因为如果我们直接写 alert，那么 JS 语言会认为 alert 是程序代码，要执行，要弹窗。

而"alert"则表示一个普通的英语单词，一段普通的字符。

6. 小结

如果大家觉得这节课的概念特别难懂，也没事，马上我们就会学习到变量，通过变量和字面量的对比，我们就能加深理解了。

零基础 JavaScript 入门教程(10)--JS 之变量的概念、定义、使用

1. 什么是变量

变量其实是数学中的概念，例如在函数 f(x)=x+1 中，x 就是一个变量，因为它没有固定的值，可以表示一个改变的数字。

那么，在 JS 语言中，同样可以设定变量，变量可以存储变化的值，从而方便我们的程序进行各种运算。

2. 字面量和变量

字面量，例如 123，它的值就是字面的值，是固定的，不能变化，所以字面量不是变量。

而变量，则完全不同，可以存储变化的值。

3. 如何表示变量

同样是 x，有时候我们希望 x 就是一个英文字母，也就是说就是一个普通的字面量，那么可以用"x"来表示。双引号包裹的 x 就是一个普通的字符串字面量，它的含义就是普通的字符串"x"。

变量 x，可以使用关键字 var 指定。注意关键字是 JS 语言特殊关键的词汇，表达一种特殊的含义。此处的 var 是我们接触到的第一个关键字，用来定义变量。

如下：

```
<script>
        "x" //这是一个普通的字面量，其含义就是一个字母 x
        var x; //此处 x 是一个变量，可以存储各种值
        "name" //这是一个普通的字面量，其含义就是一个单词 name
        var name; //此处 x 是一个变量，可以存储各种值
    </script>
```

4. 再论字符串使用引号包裹问题

学到这里，我们应该终于要明白了，为什么字符串字面量，非得用引号包裹起来。

因为直接写 x 也好，直接写 a 也好，它代表的是一个变量。而用引号包裹起来，它表示的是字面量，这两种是完全不同的含义。所以从形式上一定要区分开，程序才能识别不同的含义。

5. 变量的定义和使用

使用 var 定义变量后，可以通过变量名直接使用变量，如下：

```
    <script>
        var a; //定义了一个变量 a
        a //使用 a
    </script>
```

我们定义了一个变量 a，a 就是变量的名字，所以后续我们想使用 a 时，可以直接写 a。

6. 变量必须先定义后使用

注意，变量必须先定义后使用，就像人必须先出生，才能做事。事物必须先存在，才能发展。如果还没定义，就直接使用变量会报错！

如下代码：

```
<script>
    var a; //定义了一个变量 a
    a //使用 a
    b //使用未定义的变量 b
</script>
```

运行上面的程序，控制台报错如下：

看错误提示：b is not defined，很明显错误提示为 b 没定义。也就是咱们的程序运行的时候，不知道 b 是个啥，所以就报错了。

而 a，我们通过 var a 已经告诉程序了，a 是一个变量，所以就可以叫 a 出来溜溜，出来使用了！

7. 小结

世界因变化而丰富，

程序因变量而多彩，

古之人诚不欺我也！

零基础 JavaScript 入门教程(11)--变量的赋值

1. 前言

上一篇我们介绍了变量的概念、定义和基本使用方法。

其实变量，本质上是内存中一块存储空间的名字，这块存储空间中存储的具体内容，就是变量的值。

那么变量的值为什么是可变的呢，这个就好理解了，我们把存储空的内容换掉，变量的值不就变化了嘛。

2. 变量的赋值操作

将值存入变量对应的存储空间，这样的操作叫做赋值操作，JS 语言中，赋值操作使用=符号。

此处务必注意，这个=符号跟咱们之前学习的数学中的=符号，含义完全不同。数学中的=表示左右两边相等，而 JS 中的=表示将右边的值赋给左边的变量。

我们举个例子：

```
<script>
    var x = 1; // 将 1 这个值，赋给 x，执行这行代码后 x 存储的值为 1
</script>
```

3. 变量定义与变量赋值的区别

变量的定义，是告诉计算机，我要设定一个变量。

变量的赋值，是告诉计算机，我要给我之前设定的某个变量，给他一个确定的值。

注意，可以先定义，后赋值。也可以同时定义，并赋值。

如下：

```
<script>
    var x; //定义了一个变量，此时没有值
    x = 1; //给变量赋值 1

    var x = 1; //将 1 赋值给变量 x，该行代码同时完成了定义、赋值
</script>
```

4. 未定义直接赋值的情况

JS 里面，如果一个变量，没有通过 var 定义，直接给它赋值，也是可以执行的：

```
<script>
    x = 1; //未使用 var 定义过，直接给它赋值
</script>
```

这种方法强烈建议不要使用，打个比方，我们如果要开公司，应该先注册公司有个营业执照，然后再去运作。

这种未经定义，直接就使用的行为，不符合代码规范，也会引起一些不必要的麻烦。所以 JS 语言虽然支持这种写法，但是我们不要使用这种写法。当然这种写法会在大型项目中，带来一些意想不到的麻烦，这个我们初学阶段就不必了解太深了。

5. 小结

在 JS 代码 `var x=1;`中，var 用来定义变量，x 是变量的名字，1 是变量的值，`=`是赋值符号。

这行代码是将 1 这个值，赋给左边 x 这个变量。

执行该语句后，x 变量的值变为 1。

零基础 JavaScript 入门教程(12)--JS 语句与变量的机械意义

1. 机械

什么是机械，简单理解，就是人类制造的一些有用的设备装置。

2. 编程相关的电脑机械

电脑上的机械装置有很多，比如鼠标、键盘、屏幕、USB 接口、网线接口、显卡、内存条、CPU。

大家都知道，CPU 是核心，CPU 越先进越贵，电脑运算速度越快。

内存也很重要，内存越大，电脑能同时跑的程序越多，越不会卡顿。

大家需要格外注意的是，咱们的 JS 语言，相关的电脑机械，主要就是 CPU 和内存条。

CPU 是干啥的，是干活的，是负责执行程序的。

内存条是干啥的，是存储的，是否则存储程序过程中的数据的。

一个公司要运作，需要工人干活，需要车间办公、仓库存货。

同理，一个程序要运行，需要 CPU 干活（执行语句），需要内存办公存货（内存是一个空间）。

3. JS 语句的机械意义

看下面的 JS 代码：

```
var a = 1;
a = 2;
a = 3
```

当程序运行的时候，CPU 负责执行这些语句，先执行第 1 条语句，执行后 a 的值为 1，然后执行第 2 条语句，执行后 a 变为 2，最后执行第 3 条语句，a 的值变为 3。

所以 CPU 性能越高，这三条语句就执行的越快。

这就是 JS 语句执行的机械意义——CPU 来执行。

> 从本篇开始，我们代码示例直接写了 JS 部分。如果实际运行的话，还是需要将 JS 代码放入 `<script>`标签中运行的。

4. JS 变量的机械意义

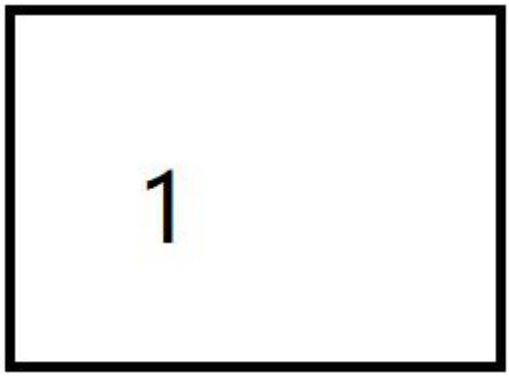
变量可以用来存储可变的值，那么这些值存到哪儿去了呢？就是存在内存中的。

看下面的代码：

```
var b = 1;
b = 2;
```

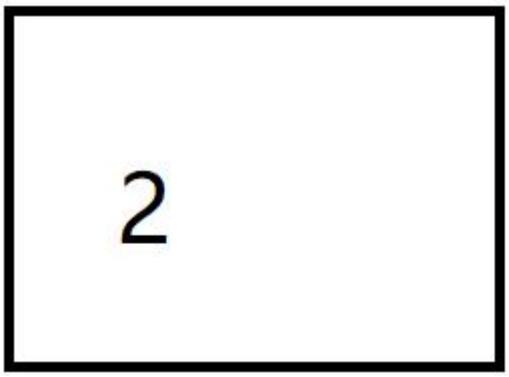
我们逐行分析下，变量变化，及对应的内存空间变化情况。

当执行完 `var b=1;`时，b 变量对应的内存空间中，存储的值是 1，如下图：



上图中的黑框，表示 b 对应的内存空间，然后执行 `var b=1` 后，空间中存的值为 1。

当执行完 `b=2;`后，b 变量对应的存储空间里面的值是 2 了，所以变为：



这就是 JS 变量的机械意义——变量名对应内存上的一块空间，变量值是该内存空间存储的内容，变量赋值操作就是改变变量对应空间的值。

5. 题外话

JS 变量赋值时内存变化这块，我用词比较谨慎，因为存在的情况可能不止一种。这个留待以后有机会探究，作为初学者大家简单理解就行。

零基础 JavaScript 入门教程(13)--JS 数据类型之数字类型

1. 数据类型

我们的世界是需要分类的，比如颜色分为红色、蓝色、黄色、绿色；语言分为汉语、英语、法语、俄语；性别分为男性、女性。

分类，能够让我们更好的理解事物，管理事物。

JS 语言，作为一种程序设计语言，在运行过程中，就是不断的处理各种数据。之前我们学习的变量，变量存储的内容就是数据。

为了方便处理、管理、应用数据，JS 语言对数据进行了分类。今天我们介绍下最简单的数据类型——数字类型。

> 其他常用的编程语言，例如 C/C++/Java/Python 等，也都对数据进行了分类。而且这些语言分类的方式差不多，所以学习了 JS 语言的数据类型之后，再学其他语言的数据类型，会轻松很多，会有一种理所当然的感觉。

2. 数字类型

顾名思义，数据类型指的就是数字，整数是数字、小数也是数字。

在下面的例子中，变量 a/b/c 存储的都是数字类型的数据。

```
    var a = 1;
    var b = 2.3;
    var c = 1000;
```

> 再次提醒，我们代码示例只写了 JS 部分。如果实际运行的话，还是需要将 JS 代码放入<script>标签中运行的。

3. 数字类型的功能

既然数字单独成为一种类型，肯定得有它的特点和功能，否则没必要单独给它一个分类了。

数字类型的基本特点，就是可以用来计数。例如：

```
    var appleNum = 100; //苹果的数量是 100
    var studentScore = 98; // 学生的分数是 98
    var age = 24; //年龄是 24 岁
```

大家注意，我们变量的命名，尽量的不要使用 a/b/c 这种，而是要让它的名字具备一定的含义。就像电脑这个名字，使用电的电脑，很形象；就像电动车，使用电驱动的车子，含义很清楚。

所以我们的变量命名，也得很明确，另外还得注意，尽量使用英文命名变量，这样咱们的程序才是世界通用的。

数字类型最常用的功能，应该是进行加减乘除等数学运算了。如下：

```
<script>
    var sum = 1 + 2;
    console.log(sum); //输出为 3
</script>
```

当执行 `var sum=1+2;`时，首先计算两个数字 1 和 2 的和，计算结果为 3，然后赋值给变量 sum。

然后我们通过 `console.log(sum);`，将 sum 的值输出到控制台。

所以运行网页，打开控制台，效果如下：

> 注意 console.log()的作用，是将括号中的内容输出到控制台上。当括号中间是字面量时，会直接将字面量输出到控制台。当括号中间是变量时，会将变量的值输出到控制台。

4. 小结

我们的程序，很多时候就是处理数据的，所以数字类型会经常使用到。

零基础 JavaScript 入门教程(14)--JS 数据类型之字符串类型

1. 前言

上一篇我们介绍了 JS 中的数字类型，实际上数字类型还不是 JS 语言中最常用的数据类型。最常用的数据类型应该是字符串类型。

2. 何为字符串

那么啥是字符串呢，比如我们网页上显示的人的姓名：张三，这个张三很明显不是数字，而是一段文本；再比如网页上显示一个英文单词 hello，这个 hello 也不是数字，它也是一段文本；再比如某人的 QQ 密码是 pass1234，这个密码中虽然有数字，但是也有字母，这个密码无法参与数学运算，其实它也是一段文本。

像这样的一段文本，数字类型无法存储，我们可以使用字符串类型来存储它们。代码如下：

```
var name = "张三";
var word = "hello";
var password = "pass1234";
```

字符串使用双引号包裹，执行上面的代码后，三个变量存储的数据类型就都是字符串类型。

3. 字符串的格式

其实，JS 支持使用双引号，或者单引号表示字符串，所以下面的代码的意思，跟上面的代码一模一样：

```
var name = '张三';
var word = 'hello';
var password = 'pass1234';
```

在 JS 语言中，字符串使用双引号或者单引号表达，都是可以的，这两种方式没有优劣之分。

我个人是比较喜欢双引号的，因为其他语言大多数是采用双引号来表示字符串，所以我也习惯了使用双引号。

4. 为啥要使用引号

我们再次讨论这个问题，就是为啥要使用引号来表示字符串，咱们不使用引号行么？

答案是不行，我们看下面的例子：

```
var a = 123;
var b = a;
var b = "a";
```

我们定义了变量 a，它的值是数字 123。

然后我们把 a 的值赋给了 b，所以 b 的值也变为 123。

然后我们把字符串"a"的值赋给 b，所以 b 的值变为字符串"a"。

从上面的例子我们可以看出，如果不使用引号，我们是无法区分 a 到底是变量，还是字符串的。

所以一定要使用引号，至于为啥选择使用引号这种符合，这是人家 JS 语言创造者设定的，咱们既然使用人家的语言，就得遵循人家的规则。

5. 小结

字符串类型可以存储一段文本，不管是汉字还是英文还是数字，都可以使用字符串存储。

但是务必注意，以下两个变量的值是不同的：

```
var a = "123";
var b = 123;
```

a 是字符串类型，b 是数字类型。

b 能够进行数学运算，而 a 不能进行数学运算，因为它不是数字。

零基础 JavaScript 入门教程(15)--JS 数据类型之布尔类型详解

1. 前言

之前两篇文章，我们先后讲解了数字类型和字符串类型，数字类型用于表示数值，字符串类型可以表示一段文本。

本节我们来讲解下布尔类型，它用来表达真假。相较于数字类型、字符串类型，布尔类型更加抽象，我给大家讲一些来龙去脉的东西，让大家好理解一些。

2. 布尔类型是干啥的

布尔类型只有两个值：true 表示真，false 表示假，用来存储判断的结果。

举个例子，数字 1 大于数字 2 吗？结果是假的，所以这个结果用布尔类型表示就是 false。

再举个例子，2000 年是闰年吗？结果是真的，所以这个结果用布尔类型表示就是 true。

3. 布尔类型是必须的吗？

首先，据我所学习过的语言，包括 C/C++/OC/C#/Java/Python/JavaScript，并不是所有语言都有布尔类型，但是大多数语言有布尔类型。

可见，布尔类型不是必须的。

布尔类型只有两个值，所以我们完全可以用数字 0 表示假的，用数字 1 表示真的。

或者我们用字符串"真"表示真的结果，字符串"假"表示假的结果。

这些都是可以的，并且有些语言真是这么做的。

4. 那为什么还需要布尔类型

虽然不用布尔类型，也能表达真假，但是容易出问题。

例如我们使用变量 sex 表达是否是男性，true 表示判断结果为真的，是男性；false 表示判断结果为假的，是女性。代码如下：

```
// 用布尔类型表示判断男性的结果
var sex = true; //判断结果为真的，是男性
sex = false; // 判断结果是假的，是女性
```

因为布尔类型，只有两个值，所以要么是男性，要么不是男性是女性，所以我们的程序表达的意义很明确。

那么如果我们不使用布尔类型，而是约定使用数字 1 表示男性，0 表示女性，如下：

```
// 用数字类型表示性别
var sex = 1; // 1 表示男性
sex = 0; // 0 表示女性
```

上面的代码是没有问题的，也能通过 1 或 0 区分性别。

但是我们也可以不小心写成了：

```
sex = 2
sex = 3;
```

那 sex 值为 2 和 3，很明显这个是不正确的，因为我们程序设计的意图就是用 sex 表示性别，出现这样的代码很不幸，没人能理解。大家不要觉得可笑，因为凡事可能必会发生，这是墨菲定律！

做一个总结吧，使用布尔类型，不是真就是假，非常明确。而使用其他类型，我们还有很多控制不了的结果。所以使用布尔类型表达判断结果，是最合适的。

因为我们的程序需要不断的判断结果，例如用户名密码是否正确，例如是否确定清空购物车，例如银行卡余额是否足够支付当前商品。所以非常有必要创建一种数据类型，来存储判断的结果。这个大家以后会慢慢体会到布尔类型的好处的。

5. 小结

布尔类型只有两个值 true，fase，数字类型和字符串类型的值是无限的。

布尔类型比较抽象不好理解，大家如果暂时不能理解也不要着急，以后在学习使用中我们会无数次的接触它，从而揭开它神秘的面纱

零基础 JavaScript 入门教程(16)--JS 数据类型之 undefined

1. 前言

之前我们讲了 3 种常见的数据类型：数字、字符串、布尔，如下：

```
var a = 1; //数字类型
var b = "1"; //字符串类型
var c = true; //布尔类型
```

那么，如果尚未给变量赋值，变量是什么类型呢，如下：

```
var d; //未赋值变量
```

2. undefined 类型

由于未给变量赋值，所以此时变量也不知道自己当前是啥类型，JS 给未赋值的变量单独设计了一个类型，即 undefined 类型。

看如下代码：

```
var a = 1; //数字类型
var b = "1"; //字符串类型
var c = true; //布尔类型
var d; //未赋值变量
console.log(a);
console.log(b);
```

```
    console.log(c);
    console.log(d);
```

运行网页后，控制台输出如下。可见：未赋值变量的值为 undefined。

3. 变量未声明的情况

大家需要区分的是，变量未声明，和未赋值还不是一个事。

变量未声明，就是根本没有这个变量，所以使用变量会报错。

而变量未赋值，已经有这个变量了，但是还没有值而已，所以可以使用变量，而它的值是指定的 undefined。

如下：

```
    var d; //未赋值变量
    console.log(d);
    console.log(e);
```

此时查看控制台，d 是可以输出的，而 e 根本没使用 var 声明过，也就是不存在这个变量，所以会报错。

4. 小结

未赋值的变量，JS 语言给他设计了一个特殊的值 undefined，代表该变量的值没有设置过。

零基础 JavaScript 入门教程(17)--算术操作符

1. 前言

在数学中，我们有加减乘除等计算符号，用来进行数学计算。

在程序的世界里，也少不了数学计算，也需要计算符号。计算机程序语言里面的计算符号，一般可以称为操作符。

本篇来学习基本加减乘除相关的计算符号，可以称之为算术操作符。

2. 加减乘除操作符

加减乘除操作符，可以进行加减乘除运算，例如：

```
var a = 1 + 2; //加法
var b = 1 - 2; //减法
var c = 1 * 2; //乘法
var d = 1 / 2; //除法
```

注意，乘法操作符是*，而除法操作符是/，不是我们习惯上的×和÷，这是我为啥呢？哈哈，简单一点理解，我们键盘上没有×和÷，所以选择了比较像的*和/代替了。

操作符除了可以对数字类型的字面量进行操作，还可以对变量进行操作，如下:

```
var a = 1;
var b = 2;
var c = a + b; //c 的值为 3
```

在上面的代码中，首先给 a 赋值 1，然后给 b 赋值 2，然后计算 a+b 的结果为 3，然后将 3 赋值给 c，所以最后 c 的值为 3。

3. 算术操作符的运算结果

需要注意的事，算术操作符操作的对象是数字类型的字面量或变量，而运算的结果是数字类型的值。

如果是形如 `var c=a+b;`这样的语句，是要先计算=右侧的运算结果，然后将结果赋值给左边的变量的。

4. 递增、递减操作符

在编程中，让变量的值加 1，减 1，是非常常见的操作：

```
var a = 1;
a = a + 1; // a 的值加 1
var b = 10;
b = b - 1; // b 的值减 1
```

由于变量值加 1，减 1 操作太频繁了，JS 提供了递增、递减操作符来快速实现加 1、减 1 效果。

```
var a = 1;
a++ // a 的值加 1
console.log(a); // 输出 2
var b = 10;
b--; // b 的值减 1
console.log(b); // 输出 9
```

可以看出，`a++`的作用与 `a=a+1` 是相同的，`b--`的作用与 `b=b-1` 是相同的，但是写起来明显更加简洁。所以能提高程序员编写代码的效率。

5. 小结

算术操作符是非常简单，容易理解的，大家练习一下应该就能够掌握。

零基础 JavaScript 入门教程(18)--关系操作符

1. 什么是关系

关系有联系的意思，表达了事务之间的相关作用、相互影响的状态。

在 JS 程序中，也需要进行关系判断，比如判断两个数值的大小。

本篇就来讲解，如何通过关系操作符来进行关系判断。

2. 关系操作符

常用的关系操作有 6 种，包括大于、小于、大于等于、小于等于、等于、不等于，对应的 JS 操作符如下：

```
1>2;//大于
1<2;//小于
1>=2;//大于等于
1<=2;//小于等于
1==2;//等于
1!=2;//不等于
```

注意，跟数学符号还是有所不同的，尤其是 JS 中=表示赋值，而==表示判断左右两边的值是否相等。

3. 关系操作符的运算结果

之前讲过算术操作符，其运算结果是数值。

那么关系操作符，运算结果是什么呢，例如 1>2，其运算结果是什么？

其实关系运算，就是一种判断，1>2 的判断结果，要么是正确、要么是错误。

之前我们正好学过一种数据类型，表达真假两个方面含义，就是布尔类型，布尔类型只有两个值 true、false，正好可以对应关系运算的结果。

在 JS 语言中，关系运算的结果是布尔类型，当判断成立时，关系运算结果为 true；当判断不成立时，关系运算的结果为 false。

所以我们看下上面运算的结果：

```
console.log(1 > 2); //false
console.log(1 < 2); //true
console.log(1 >= 2); //false
console.log(1 <= 2); //true
console.log(1 == 2); //false
console.log(1 != 2); //true
```

我们打开网页验证结果：

4. 使用变量存储关系运算结果

既然关系运算的结果是布尔类型的 true 或 false 值，那么该值自然可以用变量存储。如下：

```
var result = 1 > 2;
console.log(result); //输出 false
```

这行代码先执行右侧，1>2 不成立，所以运算结果是 false。

然后将 false 赋值给 result，所以 result 存储的就是布尔类型的 false。

5. 小结

有一些语言，关系运算的结构是用数字表示的，例如 C 语言。

但是更多的语言，选择使用布尔类型存储关系运算结果，这样更合理。因为使用数字可能会有歧义，你看到数字不一定能马上明确这是一个数学数字还是一个判断结果。而布尔类型是单独定义的类型，有单独的值 true、false，所以它肯定不会跟数字混淆，一看就知道是判断结果。

所以采用布尔类型，很合理！

零基础 JavaScript 入门教程(19)--布尔操作符

1. 前言

我们在使用用户名、密码登录某网站时，首先网站得判断用户名和密码都不能为空，如果有一个为空网站就会提示我们用户名和密码都不能为空。

之前我们已经学习过了，表示判断可以使用关系操作符，如果使用变量 username 表示用户名，使用变量 password 表示密码，则可以使用关系操作符判断用户名不为空、密码不为空：

```
username!="";//判断用户名不是空字符串
password!="";//判断密码不是空字符串
```

但是，如果判断两个同时都不为空呢？

2. 布尔运算(逻辑运算)

上面的两个判断结果都是布尔类型，判断两个结果同时成立，其实就是对布尔类型进行运算，这种运算被称为布尔运算，也就是高中时候学习过的逻辑运算。

看到这里不要慌，看似高大上的概念，其实就是不是、并且、或者的简单逻辑而已。布尔运算也就分为非、与、或三种，我们来详细解释下。

2.1 非运算

非，表示相反的意思，JS 语言中对应的布尔操作符是!，我们直接看例子，这样比较好理解：

```
var a = 2 > 1; //判断为真，所以 a 的值为 true
var b = !a; //非真，也就是假，所以 b 的值为 false
```

```
var c = !(2 > 1); // 2>1 为真，所以 c 的值为非真，所以 c 的值为 false
```

看完上面的例子，应该能体会，布尔运算是对布尔值的运算，而布尔值只有 true、false 两个，所以布尔运算就是对这两个值进行运算。

非运算会把 true 变为 false，把 false 变为 true，就这么简单。

2.2 与运算

与，表示一起、并列的意思，JS 语言中对应的布尔操作符是&&。既然表示一起，那么只有当两个结果都是真时，与运算结果才是真的，只要有一个结果是假的，那么与运算结果就是假的了。看例子：

```
var username = "张三"; //用户名
var password = "123456"; //密码
var result = username != "" && password != ""; // 判断用户名和密码是否都不为空
```

第 1 行、第 2 行代码很好理解，就是给两个变量赋值。

第 3 行比较复杂，首先我们先计算赋值右边的部分，里面有布尔运算符&&。然后布尔运算两边是username!=""和 password!=""，因为用户名、密码都不是空字符串，所以这两个运算结果都是 true。

然后对两个 true 进行与运算&&，与运算中两者均是真，则与运算结果为真。此时两个结果都是真，所以与运算结果是真的，即为 true。

所以最后，result 的值为 true。

2.3 或运算

或，表示或者的意思，JS 语言中对应的布尔操作符是||。既然是或者，那么两个结果只要有一个是真，那么或运算的结果就是真的；两个结果都是假的，或运算结果才是假。

看例子：

```
var username = "张三"; //用户名
var password = "123456"; //密码
var result = username == "" || password == ""; // 判断用户名或密码是否有为空的
```

上面的例子中，第 1 行、第 2 行对变量进行赋值，第 3 行将或运算的结果赋值给 result。

因为或运算两边是判断字符串空，两个字符串都不是空，所以或运算的对象是 false、false，当两个结果都是假的时，或运算结果为假，所以最终 result 的值为 false。

3. 小结

非运算!，true 变 false，flase 变 true。
与运算&&，a&&b，a 和 b 都为 true，结果为 true；有一个是 false，则结果为 false。
或运算||，a||b，a 和 b 中至少有一个 true，结果为 true；都是 false，结果 false。

其实就是咱们说的不是、并且、或者的意思。

零基础 JavaScript 入门教程(20)--顺序结构

1. 前言

JS 语言程序的基本结构可以分为三类：顺序结构、选择结构、循环结构。

今天我们来学习下最简单的一种：顺序结构。

所谓顺序结构，就是 JS 代码会从上直下，按顺序依次执行。

2. 实例

看例子：

```
var a = 1;
var b = 2;
var c = a + b;
console.log(c);
```

这段代码，一共有 4 条 JS 语句，会按自上而下的顺序依次执行。

所以，先把 1 赋值给 a，然后将 2 赋值给 b，然后将 a+b 的计算结果 3 赋值给 c，然后在控制台打印 c，所以控制台会显示 3，效果如下：

3. 小结

顺序结构非常简单，但却是整个程序世界的基石，也是基本的运行规则。

我们的文字书写顺序是自左而右，自上而下；我们 JS 程序顺序结构的执行顺序是自上而下，但是一些语句并不是自左而右执行的，例如赋值语句，是先计算右侧的结果，再赋值给左侧的变量，是自右而左的。

零基础 JavaScript 入门教程(21)--选择结构基础知识

1. 背景

上一篇我们讲解了 JS 语言最基本的结构——顺序结构，JS 代码在一般情况下，会自上而下，依次执行每一行代码。

但是，正如生活肯定不会总是一帆风顺，总是充满着机遇和选择。我们的程序也不会一直就平淡的执行，在很多时候需要根据不同的情况做出不同的选择。为了模拟现实社会中的不同情形下的选择，JS 语言提供了选择结构。

当然，选择是如此重要，所以不仅 JS 语言，其他高级语言如 C/C++/Java/Python/C#等，也都具备选择结构，而且不约而同使用了同一个关键字：if。

2. 条件判断

JS 语言的选择结构依据 if 条件判断语句实现，格式如下：

```
if(条件)
    语句 1;
else
    语句 2;
```

是不是很容易看懂？当条件成立时，会运行语句 1，否则运行程序 2。

那么如何算是条件成立，其实就是看条件的值，如果是布尔值 true，则为成立，如果是布尔值 false，则不成立。

我们来看一个例子，判断根据年龄判断一个人是否成年：

```
var age = 30;
if (age >= 18)
    alert("已成年");
else
    alert("未成年");
```

运行结果如下：

分析这段程序，首先给 age 赋值 30，然后判断 age>=30，因为判断为真，所以运算结果为 true，由于判断结果 true，所以执行 if 后面的代码即 alert("已成年");，所以网页弹窗显示已成年。

3. 条件判断代码的缩进问题

首先注意，下面两种写法都是正确的：

```
//写法 1
var age = 30;
if (age >= 18)
alert("已成年");
else
alert("未成年");
```

```
// 写法 2
var age = 30;
if (age >= 18)
    alert("已成年");
else
    alert("未成年");
```

但是放眼望去，想必也能发现写法 2 的代码更加条理，也更容易理解，这是因为写法 2 合理利用了缩进。

习惯上，我们可以将 if、else 后面代码缩进一个 tab 键（一般是 4 个空格）的距离，这样看起来代码更加美观。如果此时使用的是 VSCode 编辑器的话，可以直接按【Alt+Shift+F】，编辑器会自动将代码格式化为标准的缩进距离。

4. 小结

生活中处处存在选择，小到选择衣服、选择食品，大到选择学校、选择城市、选择工作；同样，程序中也是处处需要选择，需要通过条件判断语句 if 实现选择！

零基础 JavaScript 入门教程(22)--选择结构与代码块

1. 背景

选择结构的 if、else 语句，其影响力只限于之后的一行代码，例如：

```
var age = 30;
if (age >= 18)
    alert("已成年");//会执行该行代码
else
    alert("未成年");
    alert("需要保护");//会执行该行代码
```

我们的本意是，当用户 age>=18，则显示已成年；否则显示未成年、然后再显示需要保护。

上述代码运行后，实际会显示已成年，然后显示需要保护，这是因为 if、else 的影响力只能限于它俩之后一行代码，而 alert("需要保护");是不归它俩管理的，所以不论条件判断是否成立，都会执行 alert("需要保护");。

所以上述代码，实际应该写作下面的格式，我们一眼就能看出最后一行代码是不归 else 管理的，是跟 else 没关系的。

```
var age = 30;
if (age >= 18)
    alert("已成年"); //条件判断成立时执行
else
    alert("未成年"); //条件判断不成立时执行
```

```
        alert("需要保护"); //不归 if、else 管理，不论条件判断是否成立，都会执行
```

2. 代码块

此时，一种之前我没单独讲过的技术——代码块，就可以提上日程，给大家论道论道了。

JS 提供了一种技术，通过大括号包裹起来的代码，可以视为一个整块，例如：

```
{
    var a=1;
    var b=2;
    var c=a+b;
}
```

这一块代码，执行了一个加法运算，在正常的顺序结构中，其实代码块没啥具体的意义，因为本来就是自上而下执行。就算把代码放到代码块中，也不会影响代码的执行顺序。

3. 选择结构遇到代码块

但是当我们学习到选择结构时，代码块就大有用途了。因为代码块可以将若干行代码合为一个块，表达一个集体的概念。

所以我们可以在 if、else 后面写代码块，这样 if、else 就可以控制后面若干行代码了。如下：

```
        var age = 30;
        if (age >= 18) {
            alert("已成年");
        } else {
            alert("未成年");
```

```
        alert("需要保护");
    }
```

由于 if、else 后面都使用了代码块，所以它俩的影响范围会覆盖其后代码块中的全部代码。

当上面代码执行时，由于 `age>=18` 成立，所以执行 if 后面代码块的内容，而 else 后面代码块中的两行代码属于 else 的控制范围，所以不会执行。

4. 尊重业界普遍规则

在程序员界，大家普遍认为，通过使用代码块，可以让代码更清晰，功能更完善。所以哪怕我们的 if、else 只是要控制一句代码，也强烈建议使用代码块！

5. 小结

代码块，甚好！

零基础 JavaScript 入门教程(23)--选择结构的 3 种形式

1. 背景

选择其实不止 if、else 一种形式，根据具体需求的不同，还有其他用法，本篇就来具体说明下 JS 选择结构中的 3 种具体的使用形式。

2. 只有 if

假如我们只需要根据年龄判断未成年人，只有未成年人时弹窗显示信息，成年人时不做处理，那么可以只写 if 语句，如下：

```
var age = 17;
if (age < 18) {
    alert("未成年人");
}
```

这种情况下，只有 if 条件成立时，执行代码块里面的语句。

3. 使用 if 与 else

如果年龄属于未成年人时弹窗提示未成年人信息，年龄不属于未成年人时弹窗提示已成年信息，则需要同时使用 if 与 else，如下：

```
var age = 17;
if (age < 18) {
    alert("未成年人");
} else {
    alert("已成年");
}
```

在上面的代码中，如果 if 后面的条件成立，则执行 if 后紧跟的代码块，如果不成立，则执行 else 后面的代码块。

4. 使用 if、else if 与 else

还有更复杂的情况，例如根据根据分数判断成绩的情况，小于 60 是不及格 60-79 是及格，80-89 是良好，90 以上是优秀，此时可以写作：

```
var score = 85;
if (score < 60) {
    alert("不及格");
} else if (score < 80) {
    alert("及格");
} else if (score < 90) {
    alert("良好")
} else {
    alert("优秀")
}
```

程序会从上往下运行，先判断 score<60，不成立才进行第二个判断 score<80，还不成立才判断 score<90，此时成立了，所以执行后面紧跟的代码块{alert("良好");}。

所以上面的程序会弹窗显示良好。

5. 小结

JS 语言中选择结构的三种形式，使用频率都挺高，需要熟练联系、掌握。

零基础 JavaScript 入门教程(24)--为什么程序需要循环

1. 人与计算机

计算机比人聪明吗？并不是的，计算机执行任务都是依靠程序来实现的，而程序则是人编写的，所以计算机本质上只是在执行人类安排的任务。

那么为什么计算机现在这么火热，各行各业，各个领域，都渗透着计算机的身影。

那是因为计算机处理问题，不知疲倦，而且还不会马虎犯错。

所以计算机在处理全新的工作上，跟人比是没有优势的，因为计算机不会处理全新的工作，需要人类通过编写程序指导计算机如何去处理。但是计算机在处理重复性的工作上，跟人相比优势巨大，因为这个工作是重复性的，所以人只需要交给计算机一次（编写一次程序），计算机就能不知疲倦且不会马虎的一直按指定规则处理。

这就是当前阶段，计算机存在的意义。

2. 为什么程序需要循环

程序是人类编写的，用来解决的也是工作生活中的种种问题。

计算机的优势就是不知疲倦的重复，那么程序如何来实现重复这件事呢，就是通过循环。

循环的意思，就是做完一遍，再拉一遍，again and again，循环往复。

所以程序语言，一定要有循环结构，不然这门语言的能力就太有限了~

JS 语言，不外乎如是，也支持循环结构。

3. 什么是循环

那么什么是循环呢，举几个例子：

走一步是一件事情，那么走 100 步，就是把走一步这件事循环执行 100 次。

上一节课是一件事情，那么上 20 节课，就是把上一节课这件事情循环 20 次。

循环就是把一件事情重复执行若干次！

具体到 JS 程序里面，循环就是把一些语句，重复执行若干次。

既然是把指定的一些语句执行若干次，那么这些语句很自然的，可以用一个代码块包裹起来，表示一个整体。然后我们可以指定这个整体的执行次数，就能实现循环了。

4. 小结

循环在生活中无处不在，我们每天都要做起床、吃饭、睡觉这样的事情，日复一日的循环。

每个人都要生老病死，生生世世的循环。

程序，也是不断处理各种各样的循环。

零基础 JavaScript 入门教程(25)--循环语句之 while

1. 背景

上一篇我们讲了循环语句存在的必要性，本篇我们来介绍通过 while 实现循环。

我们已知，循环就是把一件事情，循环重复做若干次。

一件事情，这个比较好表达，就是一段程序语句，我们可以通过一个包含若干语句的代码块表达。

那么如何控制这件事情做若干次呢，我们可以设置一个变量，初始值为 1，事情每做一次，变量值就加 1，那么当变量值达到指定次数时，我们就不再做这件事情了。

这种思想就是 while 语句设计思想。

2. while 语句格式

while 语句格式如下：

```
while(条件){
    语句;
}
```

当条件满足时，会执行语句。每次执行完语句，会再次检查条件。直到条件不符合时，循环执行结束。执行过程为：条件判断成立--执行语句--条件判断成立--执行语句.....条件判断不成立，不再执行语句，结束循环。

3. while 语句实例

我们通过一个例子来具体说明下，假设我们想在控制台输出 10 次 hello 字符串。代码如下：

```
var i = 1;
while (i <= 10) {
    console.log("hello");
    i++;
}
```

上述代码执行后，会在控制台输出 10 次 hello。

具体执行过程，上来 i 值为 1，判断小于 10 成立，所以执行代码块；因为代码块中有 i++，所以每次执行完代码块 i 的值加 1，此时变为 2；2 依然小于 10 成立，继续执行代码块，i 值变为 3。

就这样 i 一直在加 1，直到 i 的值为 11，此时判断 i<=10 不成立，循环结束。

4. 其他实例

循环语句非常重要，所以此处多举几个例子，大家要反复练习掌握。

```
// 输出 1-100 以内的偶数
var i = 1;
while (i <= 100) {
    if (i % 2 == 0) { //判断是否为偶数，只有是偶数的才输出
        console.log(i);
    }
    i++;
}
```

```
//计算 1-1000 的和
var i = 1;
var sum = 0; // 存储和
while (i <= 1000) {
    sum = sum + i;
    i++;
}
```

```
//计算 1-10000 以内，能被 3 整除的数字的个数
var i = 0;
var count = 0; //保存个数
while (i < 10000) {
    if (i % 3 == 0) { //能被 3 整除，则个数加 1
        count++;
    }
    i++;
}
console.log(count);
```

5. 小结

从循环开始，程序结构变得复杂，一定要多加练习。

零基础 JavaScript 入门教程(26)--循环语句之 for

1. 背景

上一篇我们介绍了 JS 语言中，while 循环语句的具体实现方式。我们仔细观察下面的例子：

```
// 输出 1-100 以内的偶数
var i = 1;
while (i <= 100) {
    if (i % 2 == 0) { //判断是否为偶数，只有是偶数的才输出
        console.log(i);
    }
    i++;
}
```

我们会发现，变量 i 对于循环的控制及其重要。i 的初值为 1，每次循环执行前判断 i 的值是否还能满足条件，每次循环结束后让 i 的值发生变化，最终 i 的值不满足条件是跳出循环。

我们再观察一个例子：

```
//计算 1-1000 的和
var i = 1;
var sum = 0; // 存储和
while (i <= 1000) {
    sum = sum + i;
    i++;
}
```

```
    console.log(sum);
```

也是变量 i，有一个初值 1，然后循环条件判断是 i<=1000，然后每次循环后 i 变化(i++)。

所以，我们其实已经可以发现，JS 语言中的循环，是有一个基本套路的。

2. for 循环

for 循环将循环变量的赋初值、条件判断、循环后变量的变化放到一起，这样我们对循环的控制就一目了然。

例如，如果想输出 1-100 内的偶数，使用 for 循环后，代码如下：

```
for (var i = 1; i <= 100; i++) {
    if (i % 2 == 0) { //判断是否为偶数，只有是偶数的才输出
        console.log(i);
    }
}
```

我们来解释下代码的执行过程：

1. 进入 for 循环语句后，通过 `var i=1` 定义变量 i 并赋初值 1。
2. 然后判断 `i<=100`，判断为真，则执行循环体，即执行大括号中间的代码。
3. 每次执行完循环体后，执行 `i++`，所以 i 的值加 1，然后再次判断 `i <= 100` 是否成立，成立继续循环，不成立跳出循环。

所以上述代码，和本文开头使用 while 的代码意义完全相同。

3. for 循环的意义

for 循环让我们的思路更清晰，我们在思考一件事情的时候，应该是先整体再局部，先考虑好整个事情的起因、经过、结束等大问题，再去考虑事情的细节。

同样，我们使用 for 来循环处理一件事，先考虑循环整体从哪里开始，执行多少次，什么时候退出。然后再确定每次循环做什么，这是一种更好的整体化思想。

另外从代码形式上看，for 循环将跟 i 相关的逻辑放到了一行，即 `for (var i = 1; i <= 100; i++)` ，我们可以非常直观的看到循环的整体情况。如果使用 while 循环，我们不得不去大括号外部的前方寻找赋初值的语句，去大括号内部的后方寻找 i 变化的语句，这会消耗我们的精力。

4. 总结

鉴于 for 循环的优势，实际开发过程中，for 循环的使用频率，也是远远高于 while 循环的，我这里也强烈推荐大家使用 for 循环。

零基础 JavaScript 入门教程(27)--使用 break 结束循环

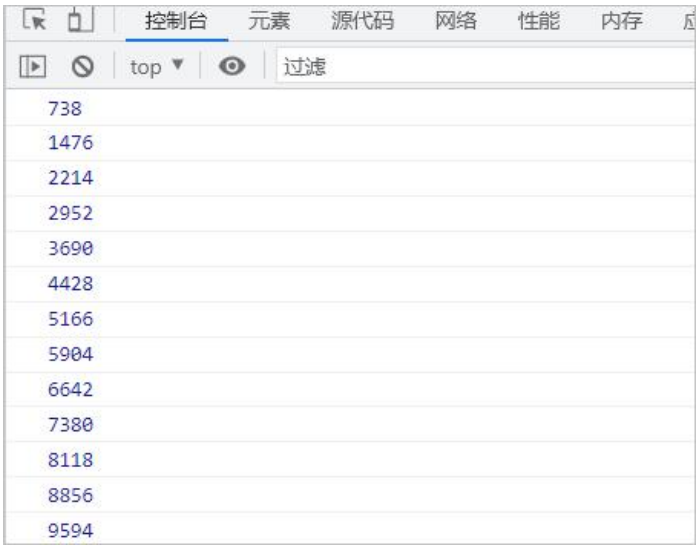
1. 背景

考虑这样的使用场景，我们需要编写程序，计算 1 至 10000 以内，第一个既能被 123，又能被 18 整除的数字。

按照我们之前学习的技术，可以使用 for 循环处理，代码如下：

```
for (var i = 1; i <= 10000; i++) { //从 1 到 10000 遍历
    if (i % 123 == 0 && i % 18 == 0) { //同时被 123 和 18 整除
        console.log(i);
    }
}
```

此时我们打开浏览器控制台，发现输出如下。我们是想找第一个符合要求的数字，也就是找到 738 就可以了，但是实际上程序还在继续寻找，直到 i 大于 10000 才结束运行。



也就是说，i 从 739 开始，到 10000，这些工作都是白干，没啥意思，浪费了计算机的计算能力。

所以我们需要找到 738 后，就提前结束循环。

2. 使用 break 跳出循环

break 的作用就是，当程序执行 break 这句代码时，会跳出 break 所在的循环语句，即直接跳出 for /while 循环。

我们改造代码如下：

```
for (var i = 1; i <= 10000; i++) { //从 1 到 10000 遍历
    if (i % 123 == 0 && i % 18 == 0) { //同时被 123 和 18 整除
        console.log(i);
        break; //跳出循环
    }
}
// xxx
```

当 i 的值达到 738 后，满足了 if 条件判断，所以执行 `console.log(i);`输出 i 的值，然后执行 `break;`跳出 for 循环，所以就会跳到`//xxx` 处执行循环外面的后续语句。

3. 在 while 中使用 break

上面的示例中，我们演示了在 for 循环中使用 break，while 循环同样可以使用 break，代码如下：

```
var i = 1;
while (i <= 10000) {
    if (i % 123 == 0 && i % 18 == 0) { //同时被 123 和 18 整除
        console.log(i);
        break; //跳出循环
    }
    i++;
}
```

4. 小结

使用 break，可以提前结束我们的工作，不必固执的执行到底。例如我们需要招聘一个 JavaScript 软件开发工程师，我们找到 1 个后就可以结束该招聘工作了，而不是永无止境的招聘下去。

零基础 JavaScript 入门教程(28)-- 使用 continue 跳过本次循环

1. 背景

上一篇学习了 break 语句，可以直接结束本次循环。

JS 语言中还提供了 continue 语句，用来跳过本次循环，在一些特殊的场景也能用到，本篇我们就来学习下。

2. 代码示例

如果我们想输出 1-100 以内，能够被 13 整除的数字，可以如下处理。

```
// 输出 1-100 之内能被 13 整除的数字
for (var i = 1; i <= 100; i++) { //从 1 到 100 遍历
    if (i % 13 == 0) {
        console.log(i);
    }
}
```

如果我们想输出不能被 13 整除的数字，可以借用 continue 如下处理。

```
// 输出 1-100 之内不能被 13 整除的数字
for (var i = 1; i <= 100; i++) { //从 1 到 100 遍历
    if (i % 13 == 0) {
        continue; //结束本次循环
    }
```

```
        console.log(i);
    }
```

解释下上面的代码，当发现 i%13==0 成立时，也就是 i 能被 13 整除时，执行 continue 语句跳过本次循环，也就是不在执行后面的 console.log(i);，转而执行进入下一次循环，也就是执行 i++，然后再次进入循环体。

也就是说，当执行 continue 语句后，循环体大括号内部 continue 后面的代码不再执行，转而执行进入下一次循环。

3. 小结

break 是直接结束整个循环，continue 是跳过本次循环后直接进入下一次循环，意义不同。

零基础 JavaScript 入门教程(29)--函数:经验的复用体

1. 程序世界的经验

我一直认为，开发程序就是为了解决现实社会中的问题。所以程序中的很多概念，就是现实社会中事物的投影。

我们可以用下面一段代码，在网页上输出学习单车的经验。

```
// 骑单车的经验
document.write("1.调整合理的座椅高度");
document.write("2.坐到自行车座位上");
document.write("3.尝试用脚蹬踏板，练习车感");
document.write("4.感觉自行要倒的时候，将车把向倒的方向慢慢弄转动，尝试找回平衡");
```

我们可以从网页上学习这样的经验：

2. 代码块的应用

既然是一个事情的经验，那么整理到一个地方会更好一些。我们可以使用代码块，将完成一件事的相关代码包裹起来。

```
        // 骑单车的经验
        {
            document.write("1.调整合理的座椅高度");
            document.write("2.坐到自行车座位上");
            document.write("3.尝试用脚蹬踏板，练习车感");
            document.write("4.感觉自行要倒的时候，将车把向倒的方向慢慢弄转动，尝试找回平
衡");
        }
```

通过代码块，同种事情或者说同一个经验，相关的代码聚集到一起，更加条理。

3. 使用代码块存储经验的问题

使用代码块后，代码显得很整洁。但是考虑这么一个场景吧，3 个小伙伴都想学习骑单车，我们需要写这样的代码：

```
        document.write("张三学习骑单车:");
        {
            document.write("1.调整合理的座椅高度");
            document.write("2.坐到自行车座位上");
            document.write("3.尝试用脚蹬踏板，练习车感");
            document.write("4.感觉自行要倒的时候，将车把向倒的方向慢慢弄转动，尝试找回平
衡");
        }
        document.write("李四学习骑单车:");
        {
            document.write("1.调整合理的座椅高度");
            document.write("2.坐到自行车座位上");
            document.write("3.尝试用脚蹬踏板，练习车感");
            document.write("4.感觉自行要倒的时候，将车把向倒的方向慢慢弄转动，尝试找回平
衡");
```

```
        }
        document.write("赵五学习骑单车:");
        {
            document.write("1.调整合理的座椅高度");
            document.write("2.坐到自行车座位上");
            document.write("3.尝试用脚蹬踏板，练习车感");
            document.write("4.感觉自行要倒的时候，将车把向倒的方向慢慢弄转动，尝试找回平
衡");
        }
```

这就好比，每次有人学开车，我们都要把这些经验完整的复述一遍，其实很麻烦。其实我们完全可以将经验保存为一本单车秘籍，每当有人学车的时候，我们就拿出秘籍交给他。

秘籍其实就是记录了经验，方便重复使用。在程序世界中，同样可以使用函数作为代码的秘籍。

4. 函数:经验的复用体

在编程语言中使用函数的目的，就是方便复用代码中的经验。

生活中，我们的经验可以存储在大脑里，也可以写到纸上，也可以录音或者保存为视频。这些经验，其实就是一些操作步骤的整合。

编程中，我们的经验需要靠函数存储，函数包含了若干行代码，通过代码块包裹为一个整体。

函数除了包括一个代码块，其实还需要一个名字。例如骑单车函数，保存的是骑单车的经验。

5. 对于函数写法的推断

经过上面的分析，函数有一个名称，还包括一个代码块，所以函数大致的模样猜测如下:

```
这是一个函数:函数名
{
    函数代码
    函数代码
    函数代码
}
```

例如骑单车函数，用 JS 的来写，差不多就是下面的样子：

```
这是一个函数: 骑单车
{
    document.write("1.调整合理的座椅高度");
    document.write("2.坐到自行车座位上");
    document.write("3.尝试用脚蹬踏板，练习车感");
    document.write("4.感觉自行要倒的时候，将车把向倒的方向慢慢弄转动，尝试找回平衡");
}
```

此时，直接使用骑单车，就可以运行代码块中所有代码。这种方式是不是更像现实中，我们使用经验的方式——骑单车这事我会啊，无非先调整坐骑然后…

6. 小结

现实社会中经验很重要，程序中就设计了函数，来对应经验，让生活、工作更加高效！

零基础 JavaScript 入门教程(30)--揭开 JS 函数的面纱

1. 背景

上一篇，我们主要给大家剖析了函数的作用——经验复用体。

然后我们分析了，函数有两个要素，一个是函数名，一个是函数体。函数名用来区分经验，比如是开车的经验还是网购的经验；函数体用来保存经验的具体操作步骤。

根据上面的分析，我们推测出，函数大概应该是这样的：

```
这是一个函数：开车
{
    console.log("1.打开车门");
    console.log("2.系好安全带");
    console.log("3.启动车辆");
    console.log("4.踩油门，开始驾驶");
}
```

当然啦，咱们并不是 JS 语言的创造者，所以咱们是没法决定 JS 函数到底怎么写的。更何况 JS 语言是外国人发明的，更不可能用这是一个函数这样的中文字样。

2. 揭开 JS 函数的面纱

我们来看看 JS 函数的真是面目，就以开车函数为例：

```
function driveCar() {
    console.log("1.打开车门");
    console.log("2.系好安全带");
    console.log("3.启动车辆");
    console.log("4.踩油门，开始驾驶");
}
```

详细的解释下：

- function 是函数的意思，这是 JS 语言专门用来定义函数的关键词，表示后面的内容就是一个函数。
- driveCar 是函数的名字，一个程序会有很多很多函数，例如开车函数、网购函数、加法函数等等等等，所以必须给函数起名来区分不同的函数。
- driveCar()后面带了一个小括号，这个是函数的特色，也就是函数名字后面要带一个小括号。后续大家会学到，这个括号其实还有更多有趣的功能。
- 最后，大括号内部可以写很多行代码，大括号形成一个代码块，这个代码块属于函数，所以可以称为函数体。

这其中最重要的，就是函数名和函数体了，函数名代表经验的类型，函数体代表经验具体的内容。

3. 函数的使用

我们人类的经验不断的积累，积累经验的意义在于下次遇到同类事务的时候，可以使用之前积累的经验。

同样，函数的意义，在于使用，专业一点，可以说调用函数，就是使用函数的意思。

我们先看下没有函数时，我们要教 3 个人开车，代码需要这么写：

```
// 教 1 个人开车
{
    console.log("1.打开车门");
    console.log("2.系好安全带");
    console.log("3.启动车辆");
    console.log("4.踩油门，开始驾驶");
}
// 教 1 个人开车
{
    console.log("1.打开车门");
    console.log("2.系好安全带");
    console.log("3.启动车辆");
    console.log("4.踩油门，开始驾驶");
}
// 教 1 个人开车
{
    console.log("1.打开车门");
    console.log("2.系好安全带");
    console.log("3.启动车辆");
    console.log("4.踩油门，开始驾驶");
}
```

使用函数后，我们可以先定义函数，在需要教人开车时，直接调用函数即可：

```
// 定义函数
function driveCar() {
    console.log("1.打开车门");
    console.log("2.系好安全带");
    console.log("3.启动车辆");
    console.log("4.踩油门，开始驾驶");
```

```
}
// 教 1 个人开车
driveCar();
// 教 1 个人开车
driveCar();
// 教 1 个人开车
driveCar();
```

也就是说，我们通过运行 driveCar();代码，即可让函数执行一次。

注意定义函数时，函数内部的代码时不执行的，只是一种经验规则的保存。只有在调用函数时，函数体才会执行。

4. 小结

经过本节的学习，其实可以发现，调用函数的方式，大局观更强。我们程序员不用事必躬亲，教给每个人怎么开车，只需要把开车这件事情写成一个函数，每次需要教开车时调用函数就行了。

零基础 JavaScript 入门教程(31)--函数的参数

1. 场景

上一篇，我们已经了解了 JS 函数的真实面目。现在我们考虑一个新的场景，还是以开车为例，之前我们开车函数如下：

```
// 定义函数
function driveCar() {
    console.log("1.打开车门");
    console.log("2.系好安全带");
    console.log("3.启动车辆");
    console.log("4.踩油门，开始驾驶");
}
```

现在我们要把开车这件事细化，因为启动车辆后，要先挂档，才能踩油门驾驶。而汽车分为自动档和手动档，自动档我们需要挂前进档，手动档我们需要挂一档。

现在的问题是，我们 driveCar()函数，只知道要开车，但是不知道是要开自动档还是手动档。

落实到生活中，我们在做一件的事情的时候，有时候是需要提供一些附加信息的。例如开车，我们需要知道开什么样的车。例如网购，我们需要知道购买什么物品。就算是进行加法运算，我们也需要知道要对哪两个数进行加法运算。

所以函数的执行，有时候还需要一些附加信息，这些附加信息，JS 语言中是通过函数参数实现的。

2. 函数参数

函数 driveCar()，小括号就是用来填写参数。我们以开车为例：

```
function driveCar(type) {
    console.log("1.打开车门");
    console.log("2.系好安全带");
    console.log("3.启动车辆");
    if (type == "自动档") {
        console.log("4.挂前进档");
    } else if (type == "手动档") {
        console.log("4.挂 1 档");
    }
    console.log("5.踩油门，开始驾驶");
}
```

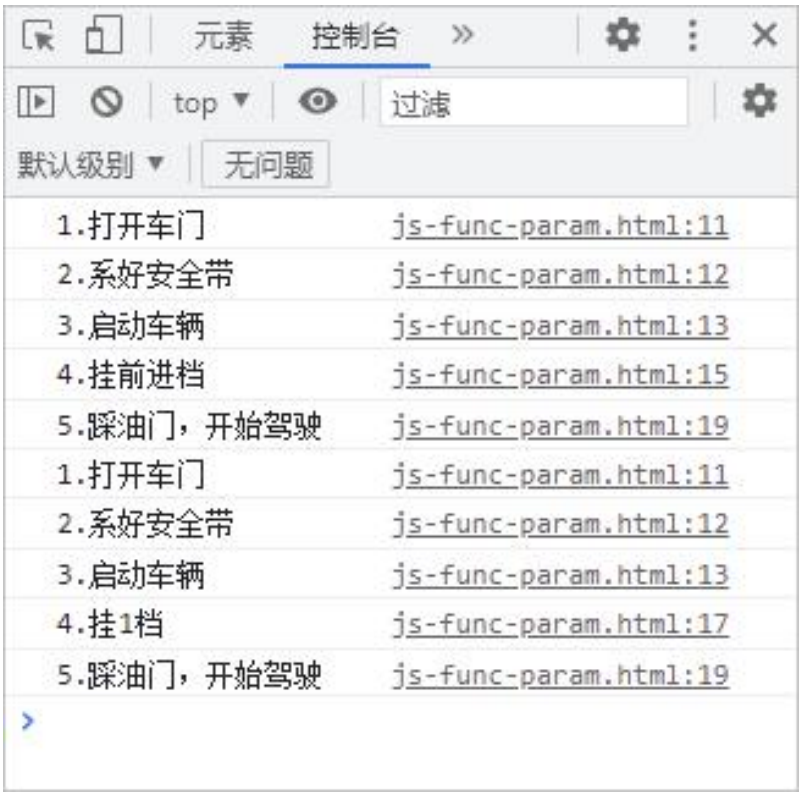
小括号中的 type，就代表函数需要的参数，其实就是一个变量。我们通过 type 变量中的值，来判断开车时，是开自动档还是手动档。

那么在调用函数时，我们告诉函数，我们是想开自动档还是手动档就 OK 了。

```
// 开自动档
driveCar("自动档");
// 开手动档
driveCar("手动档");
```

3. 运行结果

上述代码，先后运行了 2 次 driveCar 函数，第一次参数为"自动挡"，第二次参数为"手动挡",所以代码运行结果如下：



可见通过参数提供的信息，我们可以执行不同的行为了。

4. 小结

做事需要具体情况，函数需要参数信息，如此而已。

零基础 JavaScript 入门教程(32)--函数执行过程详解

1. 背景

学到这里，可能部分同学有点迷糊了，感觉函数有点太复杂了。

确实函数的概念是比较抽象的，如果是第一次接触，有些难以理解。

那么我再从头自习捋捋，分析下代码是如何运行的，让大家加深理解。

2. 执行过程分析

先看全部代码，注意左侧的行号，我们按行号对代码运行进行解释。

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta charset="utf-8">
6  </head>
7
8  <body>
9      <script>
10         function driveCar(type) {
11             console.log("1.打开车门");
12             console.log("2.系好安全带");
13             console.log("3.启动车辆");
14             if (type == "自动档") {
15                 console.log("4.挂前进档");
16             } else if (type == "手动档") {
17                 console.log("4.挂1档");
18             }
19             console.log("5.踩油门，开始驾驶");
20         }
21
22         // 开自动档
23         driveCar("自动档");
24         // 开手动档
25         driveCar("手动档");
26     </script>
27
28 </body>
29
30 </html>
```

首先明确一点，我们遵循一个最基本的原则，就是代码运行是自上而下的，在此基础上，我们分析如下：

- 第 1 行，告诉浏览器，当前是一个 HTML5 文档，请按 HTML5 标准来解析当前网页。
- 第 2 行，告诉浏览器，网页开始了。
- 第 4-6 行，是网页的头部，此处 meta 标签告诉浏览器，请用 utf-8 中文编码来接下网页的内容。
- 第 8 行，body 开始了，标志着开始浏览器开始处理内容区域。
- 第 9 行，运行到 script 标签了，浏览器开始自上而下处理 JS 代码。
- 第 10 行，运行到 function，浏览器知道这是要定义一个 JS 函数
- 第 10-20 行，浏览器理解 JS 函数的功能，此处务必注意，执行 10-20 行时，浏览器理解了函数定义的具体操作步骤，也就是代码块中的代码。但是函数体的代码并不会真正执行，因为此处只是一个定义。
- 第 23 行，执行 driveCar 函数，此时会将"自动档"传递给 type，然后真正执行 driveCar 函数体。所以此时会输出打开车门直到开始驾驶的信息。
- 第 25 行，执行 driveCar 函数，此时会将"手动档"传递给 type，然后执行函数体，由于两次执行函数体，type 接收到的值不同，所以执行结果是不同的。

3. 小结

函数定义时，函数体内的代码并不会执行，但是浏览器会记住函数执行的规则。

函数调用时，才会真正执行函数体，同时小括号内的参数会传递给函数参数。

零基础 JavaScript 入门教程(33)--函数的返回值

1. 背景

之前，我们了解了，函数其实就是经验的复用体，函数包括的是一些操作执行的步骤。

再说的透彻一点，函数其实就是去做一件事，可以把事情封装为函数。

那么就有一个问题，做事情是不是要有结果？比如学车，我们需要知道学会了吗？比如网购，我们需要知道买了多少东西、花了多少钱。

更简单的例子，比如我们要进行 2 个数的加法运算，我们希望得到的结果是两个数的和。

OK，所以说，做事要有结果；同理，执行函数也要有结果，在 JS 函数里面，结果称之为返回值。JS 语言里面，函数通过返回值来表达结果的概念。

2. JS 返回值实现

我们以计算两个数字的和为例，来演示 JS 函数返回值的实现。

2.1 定义函数

我们编写一个函数，计算两个数字的和。

```
// 计算 2 个数字的和
function add(a,b){


}
```

我们解释下上面的代码：

- // 计算 2 个数字的和是注释，用来说明函数的用途。
- function 用来定义函数，说明后面 JS 代码是函数的定义部分。
- add 是函数名称，也就是说，这个函数要做的事情是加法。
- (a,b)是函数的参数，也就是我们要计算其和的两个数字，JS 函数通过小括号来容纳参数，小括号内部的就是参数。
- {}大括号里面的部分就是函数体，也就是具体函数的操作步骤了。需要注意的是，大括号里面可以使用参数，因为参数就是为函数体提供辅助信息的。

2.2 编写函数体

我们现在，在函数体内编写计算 a,b 之和的过程，当然这个计算和的过程非常简单。代码如下：

```
// 计算 2 个数字的和
function add(a, b) {
    var sum = a + b;
}
```

上面的代码中，我们使用变量 sum 来保存 a,b 的和，完成了 a+b 的运算。

2.3 返回值

当我们的运算结束后，我们这个函数就有结果，也就是加法的结果。此时我们可以通过 return 来返回函数的运算结果。如下：

```
// 计算 2 个数字的和
function add(a, b) {
    var sum = a + b;
    return sum;
}
```

当执行 return sum;语句时，表示函数执行结束，且函数整个执行的结果为 return 后面的内容也就是 sum，所以该函数的运行结果就是 sum 的值。

2.4 函数调用

我们函数编写完成后，我要使用函数。看下面的代码：

```
add(1, 2);//计算 1+2 的和
```

上面的代码运行后，会计算 1+2 的和，然后函数的返回值 sum 值为 3，但是虽然有返回值，但是返回值并未被使用。

那么如何使用呢？我们可以将函数的返回值，赋值给一个变量，如下：

```
var result=add(1, 2);//计算 1+2 的和
```

在上面的代码中，我们知道赋值运算符=，是要先执行右边的部分，然后把右边的计算结果赋值给左边的部分。

那么在执行右边的部分，也就是 add(1,2)时，add 是一个函数，所以会执行函数体，同时把 1，2 作为参数传递给 a,b，所以函数计算结果 sum 为 1+2=3。然后函数的返回值就是 3，所以 add(1,2)这个函数的运算结果就是 3 。

最后将 3 赋值给 result，所以 result 保存的值就是函数的返回值，也就是函数的运算结果。

3. 函数无返回值的情况

还有一种特殊情况，就是函数没有返回值，也就是没有 return 语句。例如：

```
function add(a, b) {
    var sum = a + b;
}
var result = add(1, 2);
console.log(result);
```

在上面的代码中，虽然计算了 a+b，但是没有使用 return 返回值。

所以说 add 函数没有返回值！这种情况下，一个没有返回值的函数，运行结果赋值给了 result，那么 result 会得到什么东西呢？

我们直接看结果：

这个结果非常合理，因为我们没有明确的通过 return 指定返回值，所以返回值是未定义，在 JS 里面表达未定义这个含义的数据类型就是 undefined。

4. 小结

干事要有结果，函数要有返回值！

干事也可以光安排，不要结果；那么函数也可以没有返回值。

所以说程序世界，和现实是相通。

零基础 JavaScript 入门教程(34)--函数的作用域

1. 背景

首先说一下【域】的概念，我们都知道，域代表地域、领域的意思。

我们在生活中，往往把一些东西，分为私人的领域，还有公共的领域。

比如，教室里面，每个学生的书桌，是学生的【私人领域】，里面的东西属于学生个人的，未经允许其他人不能查看、使用。

还一种，就是教室里面的扫帚、拖把、过道，并不是哪个学生的私人领域，而是【公共领域】，也就是说所有人都可以使用。

总结下，私人领域，学生自己可以查看。公共领域，所有学生都能查看。

2. 函数的作用域

函数，也有它的“私人领域”，专业的名词是函数的作用域，既然是函数自己的，那么函数自己是可以使用的，但是函数外面，就不能使用该作用域里面的东西。

举一个例子：

```
// 定义一个函数
function myDesk() {
    var book = "Java 入门"; //函数作用域内部的变量
}
```

```
    console.log(book);
```

运行结果如下，这是因为 book 是函数作用域里面的东西，属于函数自己。在函数之外是无法使用的。

我们从专业角度分析，函数的作用域，其实是函数体大括号中间的部分，在函数体内定义的变量，属于函数作用域，在函数之外是无法访问的。这就是函数的作用域！

3. 全局作用域

那么不在函数体内的，也就是大括号之外，是什么呢？

这个地方也有专业名字，叫做全局作用域。也很好理解，全局都能使用。

我们看例子：

```
    // 定义了一个全局变量
    var book = "HTML 入门";
    console.log(book); // 外部可以直接访问

    function readBook() {
        console.log(book); //在函数内调用全局作用域的变量
    }
    readBook(); //调用函数时，函数内可以正常访问
```

运行结果如下：

分析下，我们先定义了一个全局变量 book，因为它不属于任何函数，所以全局都能使用它，就相当于教室里面公用的扫帚。

然后我们可以直接使用它，在函数内也可以使用它，因为它虽然不属于个人，但是是公用的。

总结：全局作用域的变量，函数内可以使用，函数外部同样可以使用。

4. 小结

正如物品、房屋等空间有私人、公共之分。

我们的程序内的作用域空间，也有全局、函数之分。全局的，到处都能访问。函数的，只有函数内部能够访问。

作用域问题其实比较复杂，此处给大家简单介绍一下，入门阶段有个大体的了解即可。

零基础 JavaScript 入门教程(35)--函数的应用实例

1. 函数的使用环境

在演示实例，我们先讲解下，在不同的使用环境下，函数的表现。对于两个数字相加来说。

首先数学函数，可以定义为:

```
f(a,b)=a+b
```

使用的话即为:

```
f(1,2)=1+2=3
```

从此可以看出，数学的表达方式是极简的。

再看 JS 的，定义为:

```
function  f(a,b){
      return  a+b;
}
```

使用的话:

```
var  sum=f(1,2);
```

最后再看下 Java 中，定义:

```
public int f(int a,int b){
    return a+b;
}
```

使用的话:

```
int sum=f(1,2);
```

这三种方式，其实意义都是计算两个数的和，但是因为环境不同，所以格式不同而已。

另外，还有一些不同，Java 的加法，只能处理 int 整数;数学的加法，可以针对任意的数字;JS 的加法，不仅能处理数字，其实还可以处理字符串等其他类型。

从这个角度讲，JS 还是有其灵活性和先进性。非常动态的语言。

2. 一些应用实例

函数的应用非常的广泛，当然在初学阶段，主要是用于一些数学运算、输入输出。接下来我们举几个例子加深理解。

2.1 判断数字是否为偶数

因为是判断，函数的运行结果最好是布尔类型，用 true 表示是偶数，用 false 表示表示偶数。判断类型的函数，其实函数返回值一般都是用布尔类型。

```
        // 判断偶数
        function judgeEven(num) {
            if (num % 2 == 0) {
                return true; //是偶数
            } else {
```

```
            return false; //不是偶数
        }
    }
    // 使用函数
    console.log(judgeEven(1)); //输出 false
    console.log(judgeEven(2)); //输出 true
```

2.2 通过生日计算年龄

输入参数为出生年份，函数返回值为对应的年龄。

```
    // 计算年龄
    function getAge(year) {
        var age = 2022 - year;
        return age;
    }
    // 使用函数
    console.log(getAge(1990)); //输出 32
```

2.3 判断素数

素数是只能被 1 和自己整除的数，注意 1 不是素数。

比如 2 是素数，因为只能被 1、2 整除。
4 不是素数，因为出除了 1、4,4 还能被 2 整除。

```
    // 判断素数
    function judgePrime(num) {
        if (num == 1) { //1 既不是素数、也不是合数
            return false;
```

```
        }

        for (var i = 2; i <= num - 1; i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
    // 使用函数输出 1-10 内的素数
    for (var i = 1; i <= 10; i++) {
        if (judgePrime(i) == true) {
            console.log(i);
        }
    }
```

运行结果如下：

输出结果是正确的，所以我们的函数编写没有问题。

3. 小结

函数的应用可以说无处不在，因为函数其实就是代表了一种规则、经验的保存。

我们通过预定义函数，下次再遇到同样的事情时，可以直接调用函数来处理，方便又快捷。