中国科学院大学
University of Chinese Academy of Sciences

计算机科学导论实验

个人作品

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

个人作品实验安排

周次	日期	实验课	课程内容	作业提交内容	截止时间
14	5月31日	制作静态网页	HTML结构 HTML语法	作品名称、 personal_artifact.zip	6月28日 10:00前
15	6月7日	制作网页版计算器	修改计算器的结果 读取操作数 执行选中的运算		
16	6月14日	制作网页版时钟	添加静态时钟 让时钟动起来 Debug方法 对比JS和Go		

本课程将会以“王小二”编写一个网页博客介绍他发明了网页版时钟和计算器的伟大事迹为例，讲述相关的编程知识。

目标

- 使用创造性的表达制作**动态**的网页。
- 完成本实验需要同学们展示**自学**的能力。
- 每位同学需要完成并**向全班展示**：
  - 一个**动态**的网页，包含HTML，CSS，JavaScript代码
  - 该网页展示了你的**创造力**

评分标准

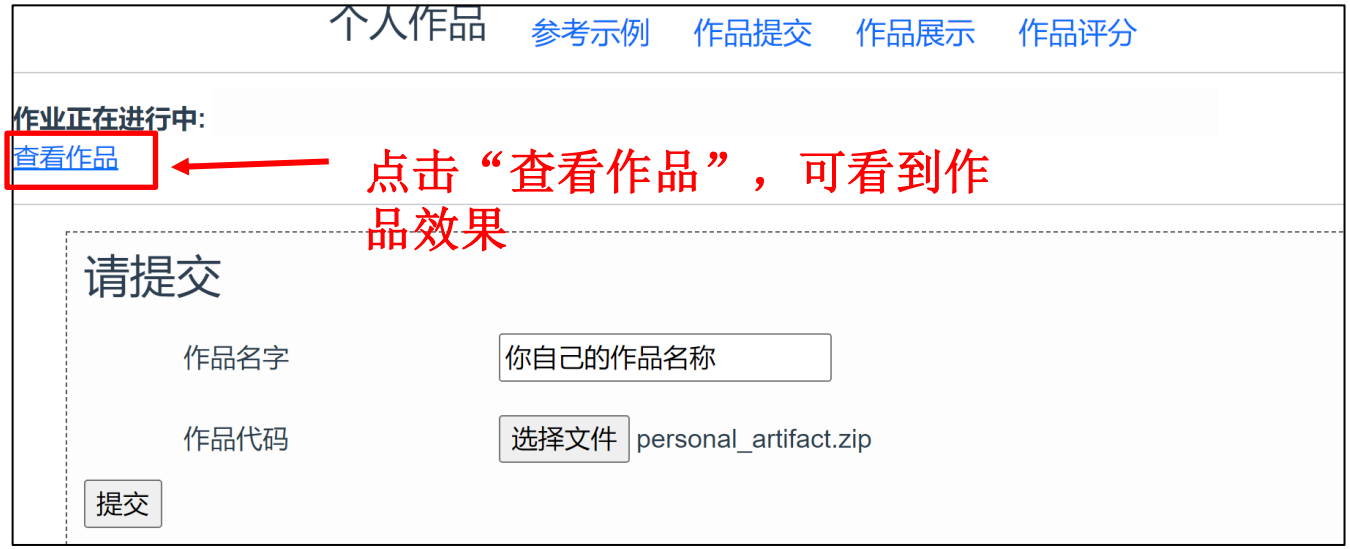
- 教师评分+同学互评，每个同学评7份作品
  - 要在7份作品中明确指明最优秀和最不优秀作品

分数	评判标准
4	独立完成且正确，少量使用他人的素材（有标注致谢），有创意
3	独立完成且大部分正确，少量使用他人的素材（有标注致谢）
2	完成且基本正确（如代码运行但结果有错），并标注致谢了使用他人的素材
1	截止时间前提交，但有明显错误（如代码不能运行）或缺失 50%标注致谢
0	未在截止时间前提交，或抄袭

- 致谢以文字的形式在页面底部显示（而不是注释）
  - 如果引用外部图片等资源，必须明确指明图片等资源链接
- 标注创造性：在<head>部分以注释的形式说明创造性

作品提交

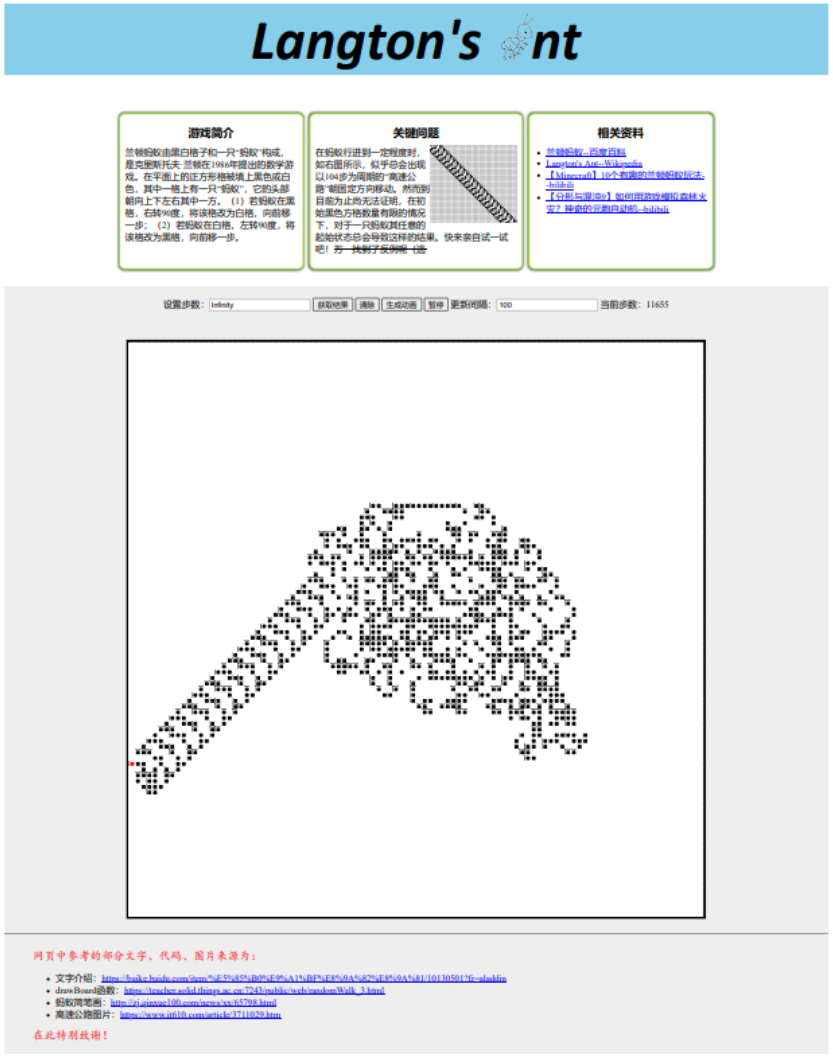
- 时间：2024 年 6 月 28 日 10:00 前
- 提交内容：作品名称 + personal_artifact.zip
  - 把包含index.html的文件夹命名为personal_artifact/，压缩为personal_artifact.zip压缩包
  - 在https://course.things.ac.cn:10088/exp/personal_artifact的作品提交界面提交personal_artifact.zip压缩包及作品名称



参考

- https://www.w3schools.com/
  - 在线教程
- https://course.things.ac.cn:10088/exp/personal_artifact
  - 实验参考示例

例子-结合你的专业

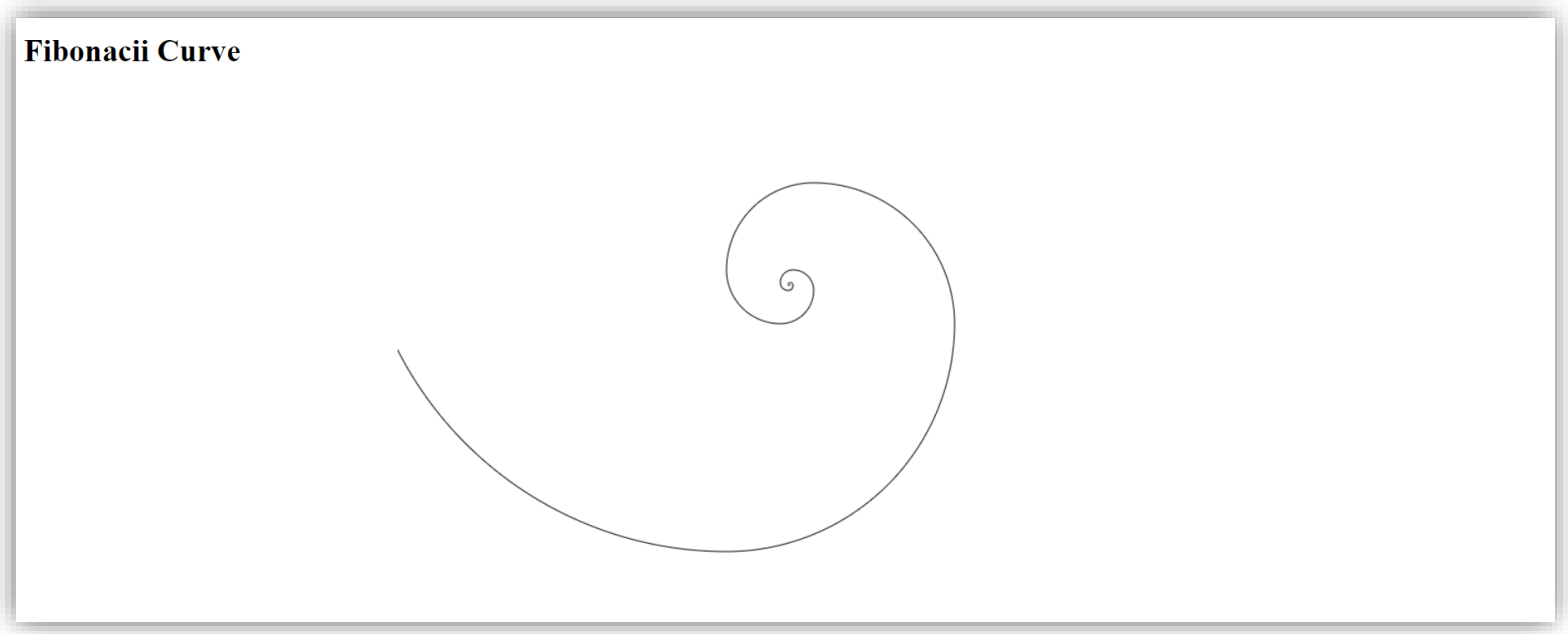


Langton's_Ant.html

“Langton’s Ant（兰顿蚂蚁）” 是一个数学游戏

例子-编程习惯好

- 此网页展示了斐波那契亚曲线
- 遵循了良好的编程习惯



Fibonacci.html

例子–编程习惯好

```
// code executed directly:
var len = 14; // the number of Fibonicaii numbers to form the Curve (Global variable)
var fibonaciiArray = new Array(len); // Fibonicaii numbers to form the Curve
calculate_fibnacii();
drawFibonacciArc();
// function definition:
function calculate_fibnacii(){
    fibonaciiArray[0] = 1; fibonaciiArray[1] = 1;
    for(var i=2;i<len;i++) fibonaciiArray[i]=fibonaciiArray[i-1]+fibonaciiArray[i-2];
}
function drawFibonacciArc(){
    // show the whole curve by draw many 1/4 circles
    var canvas = document.getElementById("myCanvas"); // get Canvas
    var ctx = canvas.getContext("2d");
    var x0 = 400, y0 = 200;
    var x = x0, y = y0, startDegree = 0, endDegree = 90;

    for (var i = 0; i < len; i++) {
        // Draw 1/4 Circle whose radius is fibonaciiNumber[i]
        var radius = fibonaciiArray[i];
        ctx.beginPath();
        ctx.strokeStyle = "black";
        ctx.arc(x,y,radius,(startDegree/180)*Math.PI,(endDegree/180)*Math.PI);
        ctx.stroke();
        // update the position of the center of next 1/4 circle
        var direction = startDegree/90;
        if(i<len-1){
            var inc_x = (direction%2-2*Math.floor(direction/3))*(fibonaciiArray[i+1]-fibonaciiArray[i]);
            var inc_y = -(1-direction+2*Math.floor(direction/3))*(fibonaciiArray[i+1]-fibonaciiArray[i]);
            x += inc_x; y += inc_y;
        }
        // update the range of degrees of next 1/4 circle
        startDegree = (startDegree+90)%360;
        endDegree = (endDegree+90)%360;
    };
}
```

- 把常数的定义放在前面
- 使用描述性的变量名
- 避免重复的代码
- 使用注释
- 避免“魔数”

例子-Tips

```
// code executed directly:
var len = 14; // the number of Fibonicaii numbers to form the Curve (Global variable)
var fibonaciiArray = new Array(len); // Fibonicaii numbers to form the Curve
calculate_fibnacii();
drawFibonacciArc();
// function definition:
function calculate_fibnacii(){
    fibonaciiArray[0] = 1; fibonaciiArray[1] = 1;
    for(var i=2;i<len;i++) fibonaciiArray[i]=fibonaciiArray[i-1]+fibonaciiArray[i-2];
}
function drawFibonacciArc(){
    // show the whole curve by draw many 1/4 circles
1   var canvas = document.getElementById("myCanvas"); // get Canvas
2   var ctx = canvas.getContext("2d");
    var x0 = 400, y0 = 200;
    var x = x0, y = y0, startDegree = 0, endDegree = 90;

    for (var i = 0; i < len; i++) {
        // Draw 1/4 Circle whose radius is fibonaciiNumber[i]
        var radius = fibonaciiArray[i];
        ctx.beginPath();
3       ctx.strokeStyle = "black";
        ctx.arc(x,y,radius,(startDegree/180)*Math.PI,(endDegree/180)*Math.PI);
4       ctx.stroke();
        // update the position of the center of next 1/4 circle
        var direction = startDegree/90;
        if(i<len-1){
            var inc_x = (direction%2-2*Math.floor(direction/3))*(fibonaciiArray[i+1]-fibonaciiArray[i]);
            var inc_y = -(1-direction+2*Math.floor(direction/3))*(fibonaciiArray[i+1]-fibonaciiArray[i]);
            x += inc_x; y += inc_y;
        }
        // update the range of degrees of next 1/4 circle
        startDegree = (startDegree+90)%360;
        endDegree = (endDegree+90)%360;
    };
}
```

- Canvas 的用法:
  1. canvas = document.getElementById("…")
  2. ctx = canvas.getContext("2d")
  3. ctx.XXX()
  4. ctx.stroke()

第一周
制作静态网页

本节课制作：

index.html

第一周 提纲

- HTML 结构
- HTML 语法

HTML：超文本标记语言
英文全称：HyperText Markup Language
这种语言用来描述网页上有什么

HTML结构

```
<html>
        <head>
        一些代码
        </head>
        <body>
        一些代码
        </body>
</html>
```

所有 HTML 代码都需要遵循这一结构

注意 “/“ 的存在!

HTML 结构–例子

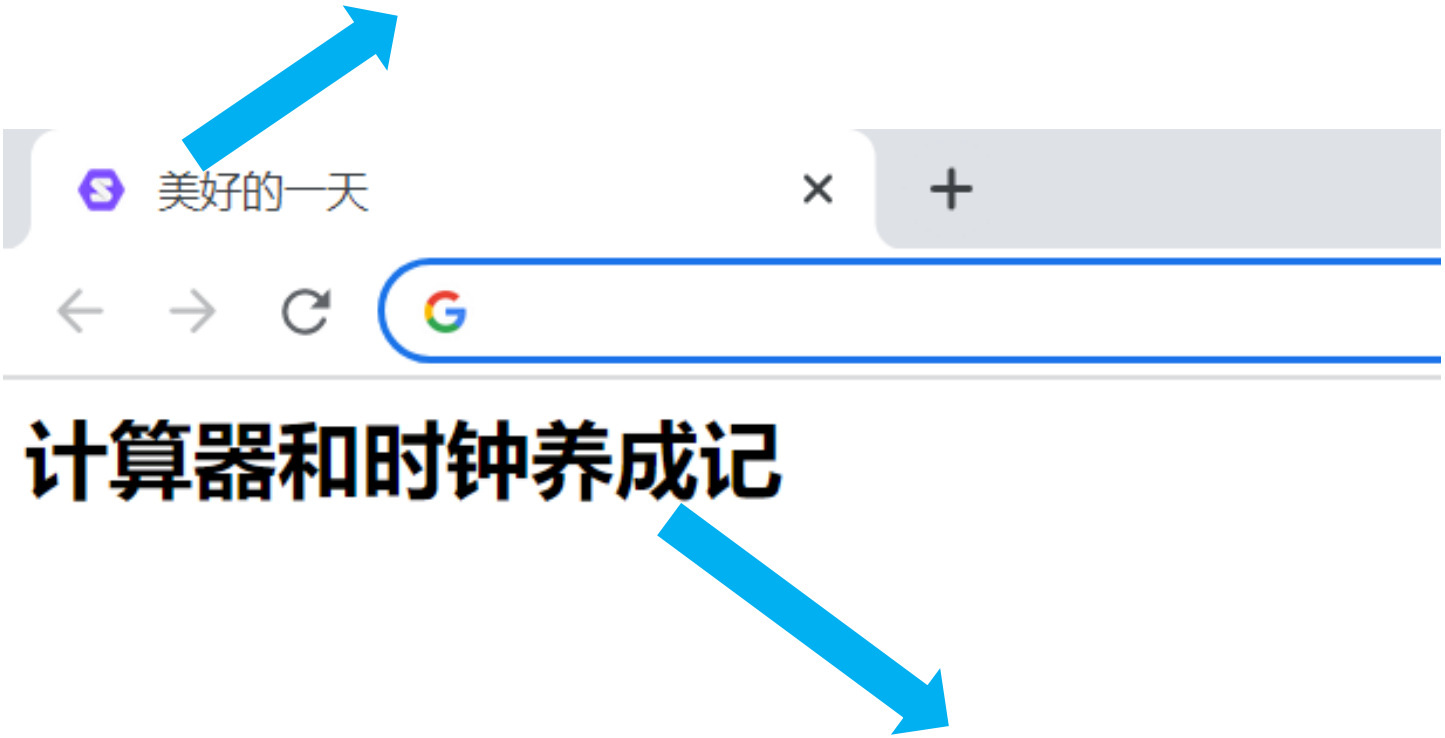
新建 index.html 文档，并写入下面的代码：

[代码意义稍后讲述]

```
<html>
    <head>
        <meta charset="UTF-8" >
        <title>美好的一天</title>
    </head>
    <body>
        <h1>计算器和时钟养成记</h1>
    </body>
</html>
```

使用浏览器(建议chrome)打开上述文档

<title>美好的一天</title>



<h1>计算器和时钟养成记</h1>

HTML 结构–例子

- 所有的html源代码的文件名均以“.html”为扩展名

第一周 提纲

- HTML 结构
- HTML 语法

HTML 语法–概述

- **标记语言：**HTML是一种标记语言，它由一个个标签组成，例如：
  - <html>、</html>
  - <head>、</head>
  - <body>、</body>
- 标签之间允许**嵌套：**即一对标签之间可以定义另一对标签，
  - 上例中，<html>…</html>之间定义了<head>…</head>和<body>…</body>
  - <head>…</head>之间定义了<meta>和<title>
  - <body>…</body>之间定义了<h1>…</h1>
- **与Go不同：**
  - Go作为一种编程语言，常常包含对数字符号的各种操作；
  - 而HTML只是用来指定一个网页上显示什么内容

HTML 语法–基本概念

- **标签**:HTML标签是一个形如"<keyword>"的单词，如<html>
- 大部分HTML标签都按照 <keyword>…</keyword> 的模式
  - 成对出现，如<h1>…</h1>
  - 前面的<h1>称为起始标签（开放标签），后面的</h1>称为结束标签（闭合标签，注意：以 / 开头）
- **属性**：HTML起始标签可以带有属性
  - 属性的值是用引号括起来的字符串，其语法如下：

<h1 *attribute_name*="…">…</h1>

- **元素**："<key_word>…</key_word>" 这一整体称为一个HTML元素

HTML 语法–<head>中的元素

- 一对<head>…</head>中间一般都需要一个<meta>标签和一对<title>标签，比如：

```
<head>
    <meta charset="UTF-8">
    <title>文本</title>
</head>
```

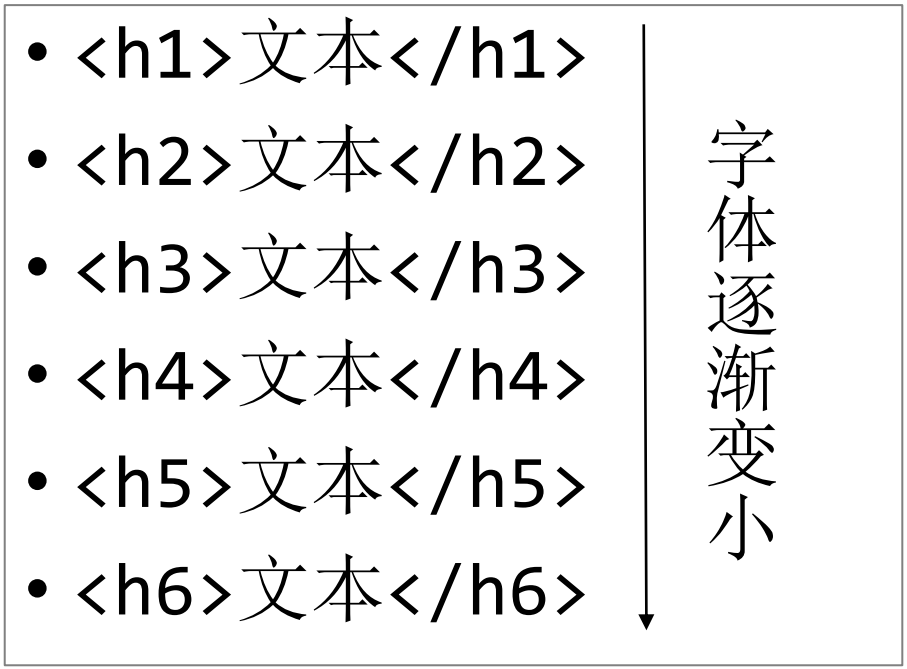
名字空间：浏览器允许的字符串（中文、英文和空格，多个空格会合并成一个）

- 它们的含义为：

标签	含义
<meta charset="UTF-8">	规定使用 utf-8 编码
<title>…</title>	规定整个网页的标题(名字)

添加小标题–标题

- 王小二觉得要介绍自己的创建过程，他的文章要分为三个部分，取三个标题（显示在网页中），分别为：“发明前”，“发明中”和“发明后”
- 使用以下知识：



代码：

```
<html>
<head>
    <meta charset="UTF-8">
    <title>美好的一天
</title>
</head>
<body>
    <h1>计算器和时钟养成记
</h1>
    <h2>发明前</h2>
    <h2>发明中</h2>
    <h2>发明后</h2>
</body>
</html>
```

效果：

添加自然段-段落、换行、水平线、注释

- 王小二在添加小标题之后，开始兴高采烈地着手写每一个小标题的下面的段落，并且希望每一个小段之间尽可能空开一些。
- 需要用到的知识如下：

段落： 文本

换行：
（或者
）

水平线：<hr/>（或者<hr>）

注释：<!--注释内容-->

代码（只展示body）：

```
<body>
    <h1>计算器和时钟养成记</h1>
    <h2>发明前</h2>
    <p>在发明前，我也不知道自己能不能做成。</p>
    <br>
    <hr>
    <h2>发明中</h2>
    <p>过程中，查了好多资料，好累但很快乐。</p>
    <br>
    <hr>
    <h2>发明后</h2>
    <p>终于做好了，虽然很粗糙，那也是亲儿子</p>
</body>
```

效果：

添加名字和时间–文本格式

- 王小二在添加段落之后，想要在自己的博客上添加名字和书写时间。名字希望斜体加粗并且上标一个小星星，时间希望是斜体。
- 需要用到的知识为文本格式化标签：
  - 文本格式化标签通常是一对标签，这对标签指定了标签中间的文本的格式，例如：

- 加粗：加粗文本
- 斜体：斜体文本
- 上标：^{上标文本}
- 下标：_{下标文本}

代码（只展示body）：

```
<body>
    <h1>计算器和时钟养成记</h1>
    作者：<b><i>王小二</i></b>
<sup>*</sup> 时间：<i>2020年02月26
日</i>
    <h2>发明前</h2>
    <p>在发明前，我也不知道自己能不
能做成。</p>
    <br>
    <hr>
    <h2>发明中</h2>
    <p>过程中，查了好多资料，好累但
很快乐。</p>
    <br>
    <hr>
    <h2>发明后</h2>
    <p>终于做好了，虽然很粗糙，那也
是亲儿子</p>
</body>
```

效果：

添加进度表–表格

- 王小二在添加完姓名和时间之后，开始回忆自己的创作过程，希望用一个表格的方式记录自己这几天干的事情。
- 需要用的知识如下：
  - 用一对 **<table>…</table>** 定义**表格**；
  - 在这对标签中间，用*n*对**<tr>…</tr>**定义*n*行；
  - 在每对<tr>…</tr>之间，用*m*对**<td>…</td>**在同一行定义*m*个单元格；
  - 一对<td>…</td>中间的文本就是显示在单元格中的内容。
  - 例如：

```
<table>
        <tr><td>(row0,col0)</td><td>(row0,col1)</td></tr>
        <tr><td>(row1,col0)</td><td>(row1,col1)</td></tr>
</table>
```

代码（“发明中”部分）：

表格框线条粗细

```
<table border="1">
    表：王小二艰难历程表
    <tr>
        <td>事件</td><td>感悟</td>
    </tr>
    <tr>
        <td>看CSS语法</td><td>一天没睡</td>
    </tr>
    <tr>
        <td>看HTML语法</td><td>还是没睡</td>
    </tr>
    <tr>
        <td>看JS语法</td><td>还是GO香</td>
    </tr>
</table>
```

效果：

发明中

过程中，查了好多资料，好累但很快乐。

表：王二小艰难历程表

事件	感悟
看CSS语法	一天没睡
看HTML语法	还是没睡
看JS语法	还是GO香

添加动机–有序列表

把<ol>换成<ul>会定义无序列表（列表项前不显示序号）

- 王小二在添加了创造过程后，突然发现“发明前”还没写，于是用一个有序列表讲了讲自己写网页的目的
  - 有序列表用一对
…
定义，
  - 列表每一项用一对
…
定义，
  - 一对
…
中间的文本是显示在列表项中的内容，
  - 有序列表在浏览器中显示时会自动出现序号

```
<ol>
    <li>文本</li>
    <li>文本</li>
</ol>
```

代码（“发明前”部分）：

```
列表：王小二的动机
<ol>
        <li>学习CSS</li>
        <li>学习HTML</li>
        <li>学习JS</li>
        <li>写计算器和时钟</li>
</ol>
```

效果：

列表：王小二的动机

1. 学习CSS
2. 学习HTML
3. 学习JS
4. 写计算器和时钟

添加计算器–表单

- 王小二在添加了写网页的目的后，想在页面上显示计算器，于是用下面关于表单的知识设计了计算器的样子：

- **表单**是由一对 **<form>…</form>** 组成的元素，在这对标签中间可以使用单一的 **<input>** 定义各种各样的**表单元素**。

- 表单通常用来获取用户的输入，表单内的表单元素通常是文本框或者选项，用户可以在文本框输入文本或者选择选项。

添加计算器–表单元素

- 各种表单元素使用 <input> 上的 **type 属性**来区分：

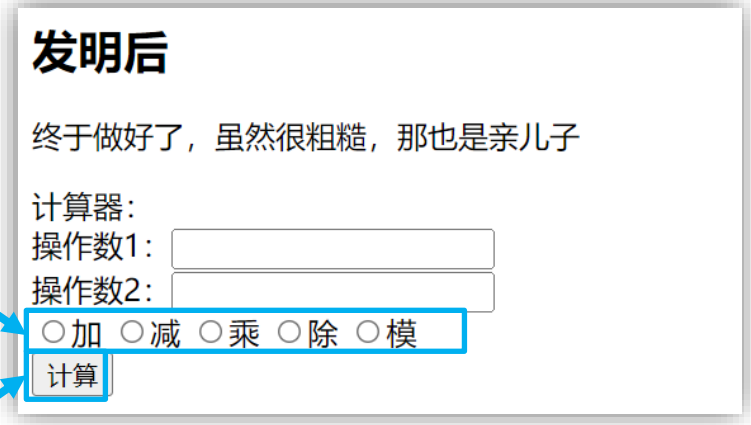
语法	描述
<input type="text">	能输入文本的框，实时显示用户输入的文本
<input type="password">	用来输入密码的框，将用户输入的字符显示为小圆点
<input type="radio" name="xx">	单选框，同一个表单多个中name属性相同的单选框只能选中一个
<input type="checkbox">	复选框
<input type="button" value="xx">	按钮（value属性的值为按钮上的文字）

代码（“发明后”部分）：

一对 <form>…</form> 中同样**name**属性的单选框是一组，只能选中其一

```
计算器:
  <form>
    操作数1:  <input type="text"><br/>
    操作数2:  <input type="text">
    <br/>
    <input type="radio" name="op">加
    <input type="radio" name="op">减
    <input type="radio" name="op">乘
    <input type="radio" name="op">除
    <input type="radio" name="op">模
    <br/>
    <input type="button" value="计算">
  </form>
```

效果：



- 注：现在只有计算器的样子，还没有定义功能。

HTML语法知识–块级元素 & 内联元素

- 王小二正苦于不知道怎么写，不经意间发现了下面的知识。
  - 在网页中，有的HTML元素可以嵌入在某一行里，比如文本格式化标签<i>组成的元素，具有这种性质的元素称为**内联元素**
  - 反之，像由<h1>元素那样以新的一行开始的元素，称为**块级元素**
- 例子：
  - ** … :**
    - 一个块级元素，无特殊含义。
    - 用于组合多个块级元素（一般网页比较复杂时可为网页分块），方便将来对多个元素进行整体操作。
  - **…:**
    - 一个内联元素，无特殊含义。
    - 用于组合多个行内元素或在某一行内嵌入元素，方便将来操作一行内的一部分。

代码分块– div 标签

- 王小二正好觉得代码有点乱，层次不够清楚，于是使用 标签把代码里发明三阶段分块，这样代码变得好看得多。

```
<body>
    ……
    <div>
        <h2>发明前</h2>
        ……
    </div>
    ……
    <div>
        <h2>发明后</h2>
        ……
    </div>
</body>
```

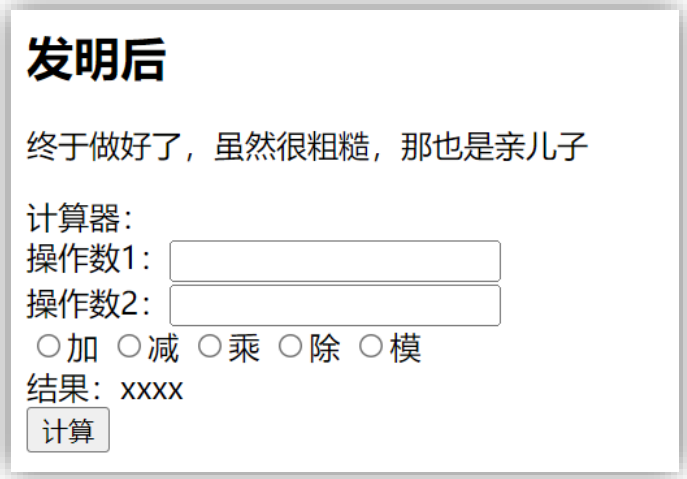
- 加上 后只是改变了代码的结构，网页并没有什么变化。

给计算器结果留位置– span 标签

- 王小二眉头一皱，发觉计算器没给计算结果留下位置，于是使用 在 <form> 里加上结果的位置。

```
计算器:
<form>
  操作数1:  <input type="text"><br/>
  操作数2:  <input type="text">
  <br/>
  <input type="radio" name="op">加
  <input type="radio" name="op">减
  <input type="radio" name="op">乘
  <input type="radio" name="op">除
  <input type="radio" name="op">模
  <br/>
  结果:  <span>xxxx</span><br/>
  <input type="button" value="计算">
</form>
```

效果：



添加元素的样式

- 王小二改完代码，开始思考怎样让自己的网页更漂亮，于是学习了下面有关CSS的知识：
  - CSS（Cascading Style Sheets，层叠样式表）是一种用来指定HTML元素的样式（如：字体颜色、背景颜色）的语言，其语句具有“名称：值；”的形式，HTML文档有两种方法来加入CSS语句：
- **方法一**：直接在起始标签上定义 **style 属性**，属性的值是若干 CSS 语句，例如：

```
<p style="background-color: gold;">一个金色背景的段落</p>
<p style="color:red;">一个红色字体的段落</p>
```

- 也可以同时规定不同方面的样式

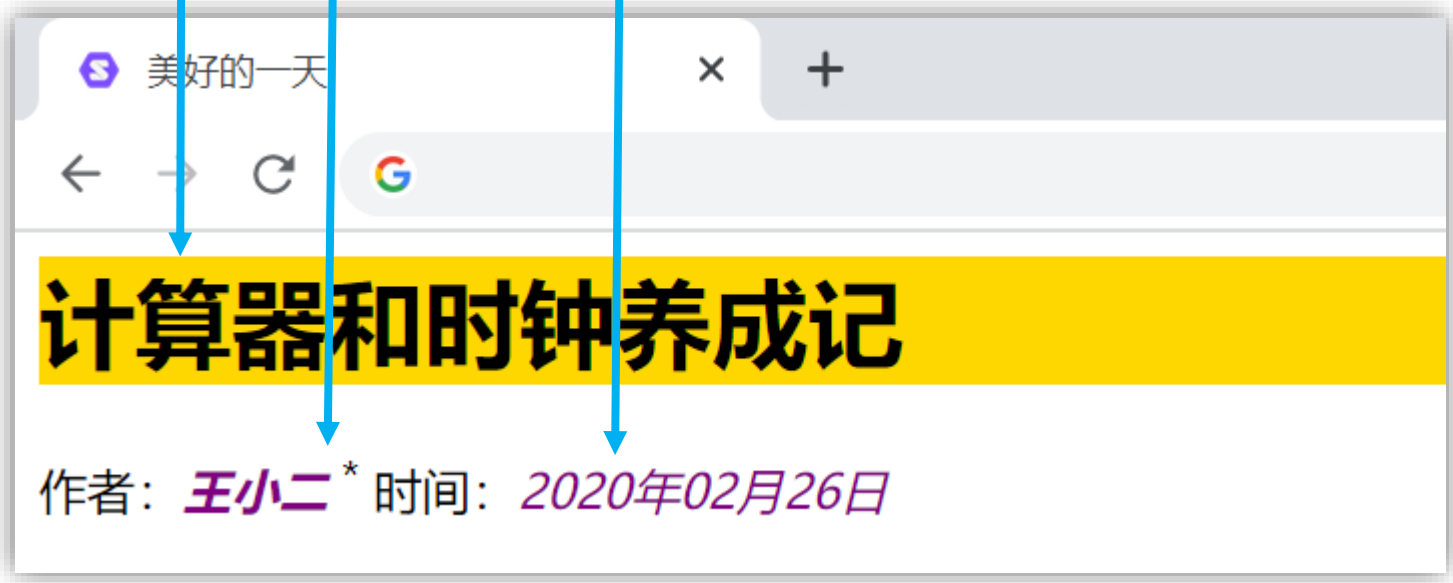
```
<h1 style="color:blue;text-align:center;">一个蓝色字体、居中的标题</h1>
```

注意：CSS 语句中名称和值不能是随便一个字符串，而是 CSS 语法里有的字符串

给元素添加颜色

```
<h1 style="background-color: gold;">计算器和时钟养成记</h1>
作者：<b><i style="color: purple;">王小二</i></b> <sup>*</sup>
时间：<i style="color: purple;">2020年02月26日</i>
```

效果：



添加元素的样式

- 王小二觉得自己的网页还不够好看，想要把发明前、中、后各自美化一下，于是使用了下面的知识：
- **方法二**：在<head>…</head>中间定义**<style>…</style>**，这对<style>内部是若干CSS语句 。
- 例子：

```
<style>
    h1 {color:red;}  /* 规定某类标签的样式：所有<h1>元素字体为红色 */
    #id属性的值 {color:green;}  /* 规定某个元素的样式：被id属性【稍后讲述】指定的元素字体颜色为绿色 */
    .class属性的值 {color:yellow;} /* 规定某“组”元素的样式：具有相同class属性【稍后讲述】的元素字体颜色为黄色 */
</style>
```

元素的通用属性

- 几乎所有的元素都可定义下面几种属性：

属性	描述
id	元素的标识，**不同元素需要不同id，可以是任意一个字符串**
name	元素的名称，可以是任意一个字符串
class	元素的类，一个元素可以有若干个类，不同元素可以有相同类，可以是任意一个字符串
style	描述了元素的样式

- 定义以上属性的语法

```
<tag_name  attribute1_name="…"  attribute2_name="…" >…</tag_name>
```

给元素添加属性

```
<body>
    ……
    <div id="before">
        ……
    </div>
    <div id="during">
        ……
    </div>
    <div id="after">
        ……
    </div>
</body>
```

- 只是加上id属性不会改变网页的显示效果
- 但是可以被<style></style>中间的语句选中，来添加网页的样式。

给元素添加颜色

和 id 中间不能有空格

```
<style>
    #before {
     background-color:skyblue;
    }
    #during {
     background-color:springgreen;
    }
    #after {
     background-color:tomato;
    }
    h2 {
     background-color:violet;
    }
</style>
```

效果：

个人作品

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

我们将实现网页版计算器：

计算器和时钟养成记

作者：*王小二*[*] 时间：*2020年02月26日*

发明前

在发明前，我也不知道自己能不能做成。

列表：王小二的动机

1. 学习CSS
2. 学习HTML
3. 学习JS
4. 写计算器和时钟

发明中

过程中，查了好多资料，好累但很快乐。下表是我这几天的艰辛历程：

表：王小二艰难历程表

事件	感悟
看CSS语法	一天没睡
看HTML语法	还是没睡
看JS语法	还是GO香

发明后

终于做好了，虽然很粗糙，那也是亲儿子

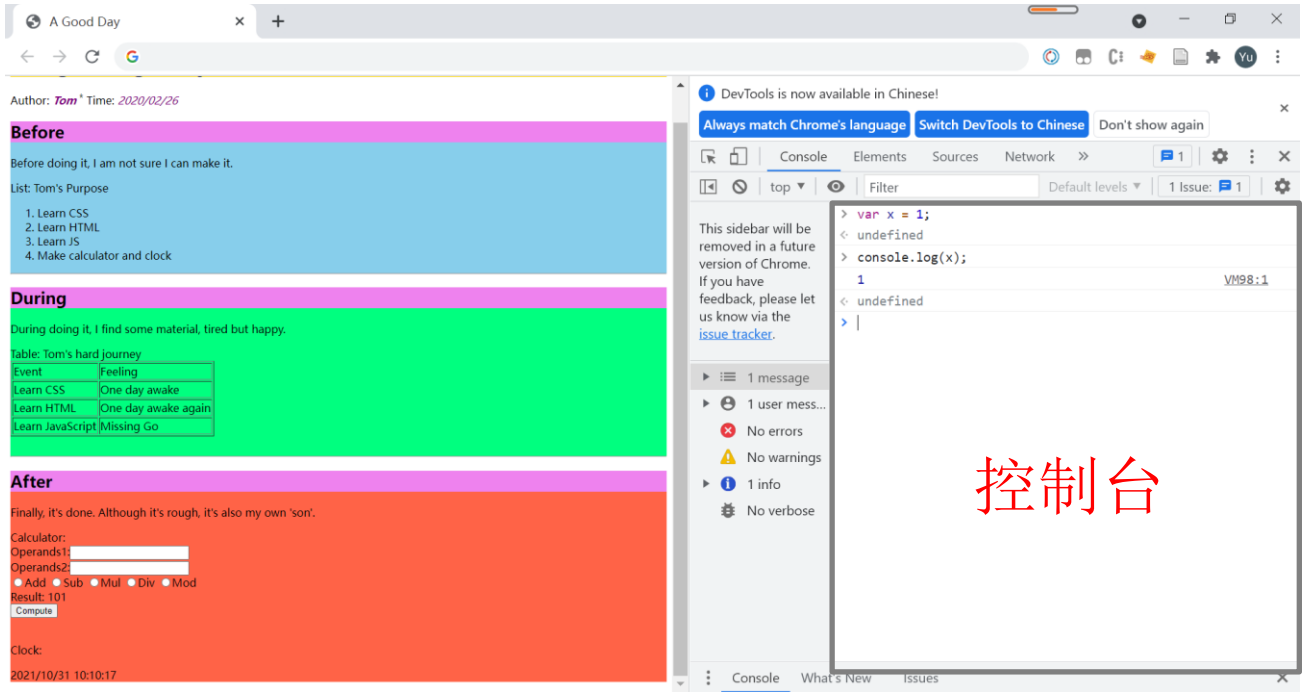
计算器：
操作数1：
操作数2：
○加 ○减 ○乘 ○除 ○模
结果: xxxx
计算

网页版计算器

index.html

控制台 Console

- 浏览器都有一个类似于终端(terminal)的**控制台(console)**
  - 控制台是 debug 时的标准输入(StdIn)、标准输出(StdOut)和标准错误输出(StdErr)
  - 我们可以直接在控制台执行 JS 代码，无需一个 html 文档
- 打开浏览器，按 **F12** 可打开控制台



控制台 Console

- 一般来说，在控制台执行代码不改变网页的行为
  - 除非在控制台直接修改网页内容
- 在控制台输入下列代码，网页内容不改变

```
> var x=1;
< undefined
> console.log(x);
  1
< undefined
> x = x + 1;
< 2
> console.log(x);
  2
< undefined
>
```

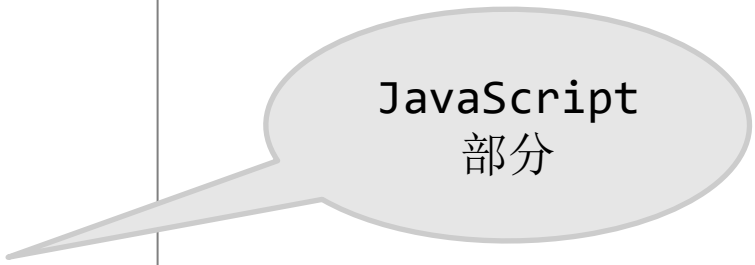
Debug: console.log 函数

- 对比 console.log(…) 函数与 fmt.Println(…) 函数

	console.log	fmt.Println
输入	字符串、整数、浮点数、对象	字符串、整数、浮点数
输出	输出到控制台	输出到终端
用途	专门用于 debug	可以用于 debug，**也用于程序和用户之间的交互**

插入 JavaScript 代码

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>美好的一天</title>
    </head>
    <body>
    ……(HTML 部分)
        <script>
        …
        </script>
    </body>
</html>
```



第二周 提纲

- 修改计算器的结果
  - 代表 HTML 元素的 JavaScript 对象
- 读取操作数
  - JavaScript "onclick" 事件
- 执行选中的运算
  - JavaScript Array 对象
  - JavaScript for-循环

JavaScript 编程知识–对象

- **对象**可以认为是一种“复合”的数据类型，它是一些**属性**和一些**方法**的集合，比如：将一个人看成一个对象，它有身高、体重、年龄、性别等**属性**，以及走路、跑步、说话等**方法**
- 使用 .（英文句点）来访问对象的属性和方法，如下：
- 对象的属性：类似于对象包含的变量
  - `obj.property`
- 对象的方法：类似于对象包含的函数
  - `obj.method();`

JavaScript编程知识–代表HTML元素的对象

- 为了实现计算器，王小二需要知道怎样修改计算器的结果，所以他学习了如何用 JS 访问 HTML 元素：
- JS 可以把已定义的HTML元素**当作对象**来访问 (2 步)：
  1. 通过 id 属性获取 HTML 元素

     ```
     var x = document.getElementById("p1");
     ```

     - document.getElementById("p1") 的返回值是代表id属性为 "p1" 的 HTML 元素的对象
     - 因此，x 是代表 id 属性为 "p1" 的 HTML 元素的对象
  2. 通过读写获取到的对象的属性来读写 HTML 元素的属性

     ```
     x.innerHTML = "newContent";
     ```

     - 设置该 HTML 元素的 innerHTML 属性为 "newContent"
     - id 属性为 “p1” 的 HTML 元素的起始、终止标签中间的内容就变成了 "newContent"
- 上面 2 步可以合成一句：

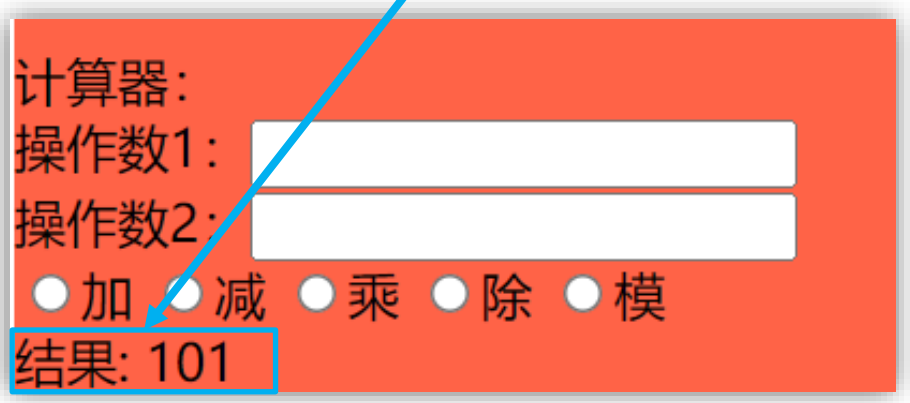
  ```
  document.getElementById("p1").innerHTML = "newContent";
  ```

修改计算器的结果

代码：

```
<script>
    var z = 101;
    var result =document.getElementById("result");
    result.innerHTML = z;
</script>
```

效果：



第二周 提纲

- 修改计算器的结果
  - 代表 HTML 元素的 JavaScript 对象
- 读取操作数
  - JavaScript "onclick" 事件
- 执行选中的运算
  - JavaScript Array 对象
  - JavaScript for-循环

JavaScript 编程知识–事件

- 王小二学会改变计算器的结果后，还需要实现：当点击按钮后，网页读取用户输入的操作数。因此，他学习了关于事件的知识：
- 3 个常见事件：
  - 用户点击按钮 (<input type=“button” onclick="*JS 代码* ">)
  - 用户的输入内容改变 (<input type=“text” oninput="*JS 代码* ">)
  - 浏览器加载完某个网页 (<script> window.onload=*一个函数定义* </script>)
- 当一个事件发生时，可以执行一段 JS 代码来处理该事件

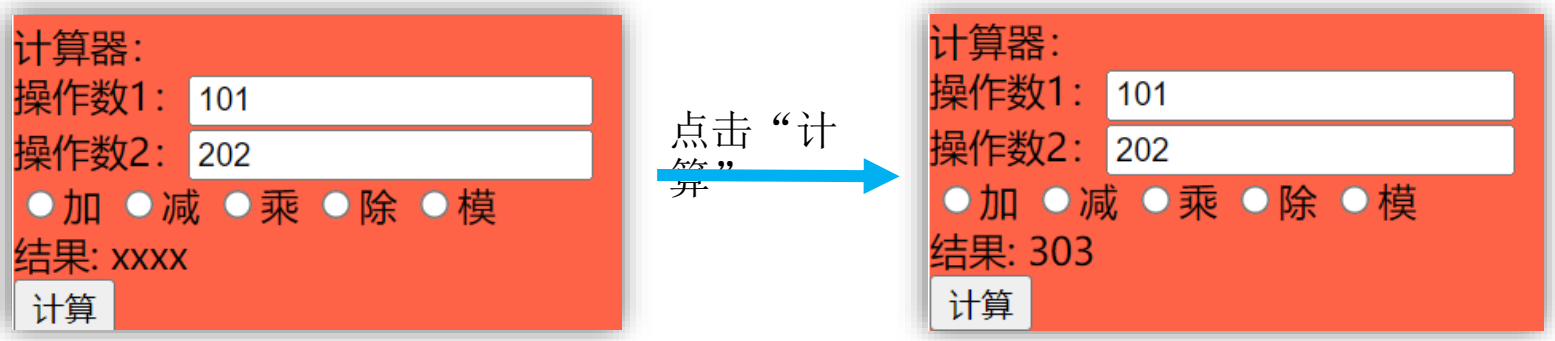
添加事件响应

- onclick=“JS 代码 ” 中的 “JS 代码” 通常是一条函数调用语句，例如：

代码：

```
…
<input type=“button” value="计算" onclick="run_calculator()">
…
<script>
  function run_calculator(){
  var operand1 = document.getElementById('operand1').value - 0;
  var operand2 = document.getElementById('operand2').value - 0;
  var z = operand1 + operand2;
  document.getElementById('result').innerHTML = z;
}
</script>
```

效果：



第二周 提纲

- 修改计算器的结果
  - 代表 HTML 元素的 JavaScript 对象
- 读取操作数
  - JavaScript "onclick" 事件
- 执行选中的运算
  - JavaScript Array 对象
  - JavaScript for-循环

JavaScript 编程知识– Array 对象

- 王小二成功读取了计算器的两个操作数后，还需要根据用户选择的运算种类进行运算，于是他将进行 5 种运算的 5 个函数放在一个数组中：
  - Array 对象（数组）的**创建**

```
var ar = new Array();  // 变量 ar 中存一个数组 (不需要指定长度，
                                          使用时直接赋值，长度自动确定)
var ar = ["Java", "Script"]; // 可以使用 "[]" 创建数组
var ar = [0, "ABC", [true,false]]; // 数组的各元素类型可以不同，
                                          下标从0开始
```

  - 数组元素的**修改**

```
ar[0]=1;
```

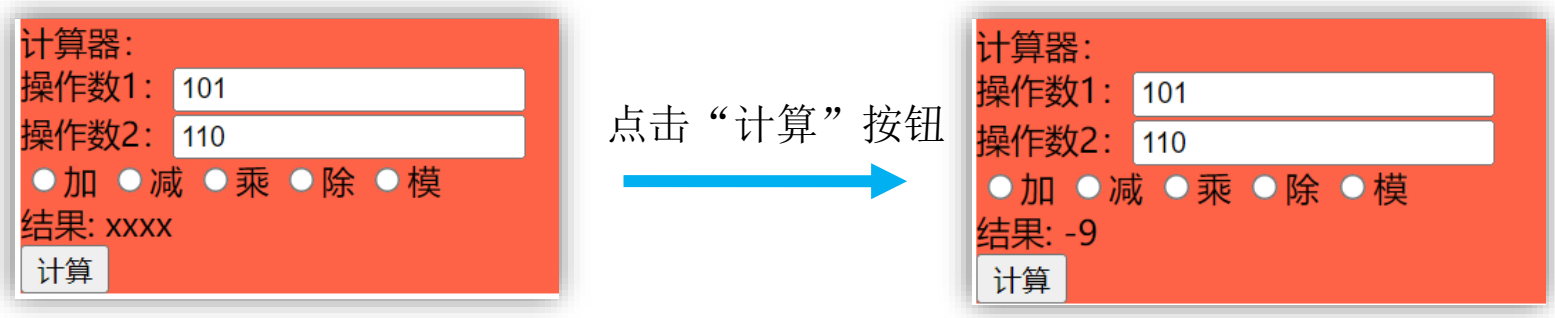
  - 数组的**属性**

```
var ar=[0,1,2];
var len = ar.length; // len=3（数组的长度）
```

使用函数数组

```
<script>
…
function add(x,y){ return x+y; } //加
function sub(x,y){ return x-y; } //减
function mul(x,y){ return x*y; } //乘
function div(x,y){ return x/y; } //除
function mod(x,y){ return x%y; } //模
function run_calculator(){
  var operand1 = document.getElementById('operand1').value - 0;
  var operand2 = document.getElementById('operand2').value - 0;
  var function_array = [add, sub, mul, div, mod];
  var z = function_array[1](operand1, operand2);
  document.getElementById("result").innerHTML=z;
}
</script>
```

效果：



第二周 提纲

- 修改计算器的结果
  - 代表 HTML 元素的 JavaScript 对象
- 读取操作数
  - JavaScript “onclick” 事件
- 执行选中的运算
  - JavaScript Array 对象
  - JavaScript for-循环

JavaScript 编程知识—for 循环

- 在王小二之前的代码里，无论用户所选运算是什么，都会执行减法。因此，他还需要遍历一遍网页中的五个选项，以确定用户选择了哪种运算，于是学习了下面关于 for 循环的知识
- for 循环：

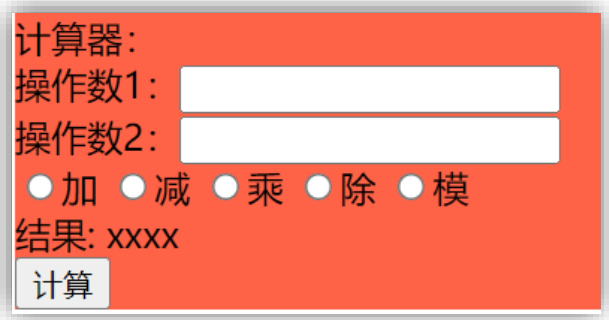
```
for(statement1;expression1;statement2){
    //some code
}
```

- 和 Go 中的 for 循环类似，区别仅仅是 JS 中的 for 循环需要在 for 后面加一对圆括号

回顾计算器的 HTML 代码：

```
计算器:
<form>
    操作数1:  <input type="text"><br/>
    操作数2:  <input type="text">
    <br/>
    <input type="radio" name="op">加
    <input type="radio" name="op">减
    <input type="radio" name="op">乘
    <input type="radio" name="op">除
    <input type="radio" name="op">模
    <br/>
    结果:  <span>xxxx</span><br/>
    <input type="button" value="计算">
</form>
```

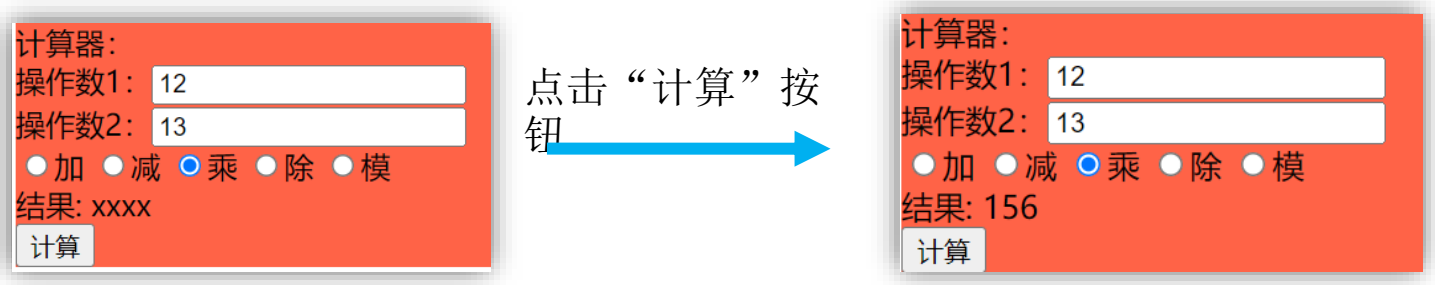
效果：



用 for 循环遍历数组

```
function run_calculator(){
  …
  var function_array = [add, sub, mul, div, mod];
  var options = document.getElementsByName("op");
  var z;
  for(var i=0; i<function_array.length; i++){
    if(options[i].checked == true){
      z = function_array[i](operand1, operand2);
      break;
    }
  }
  document.getElementById('result').innerHTML = z;
}
```

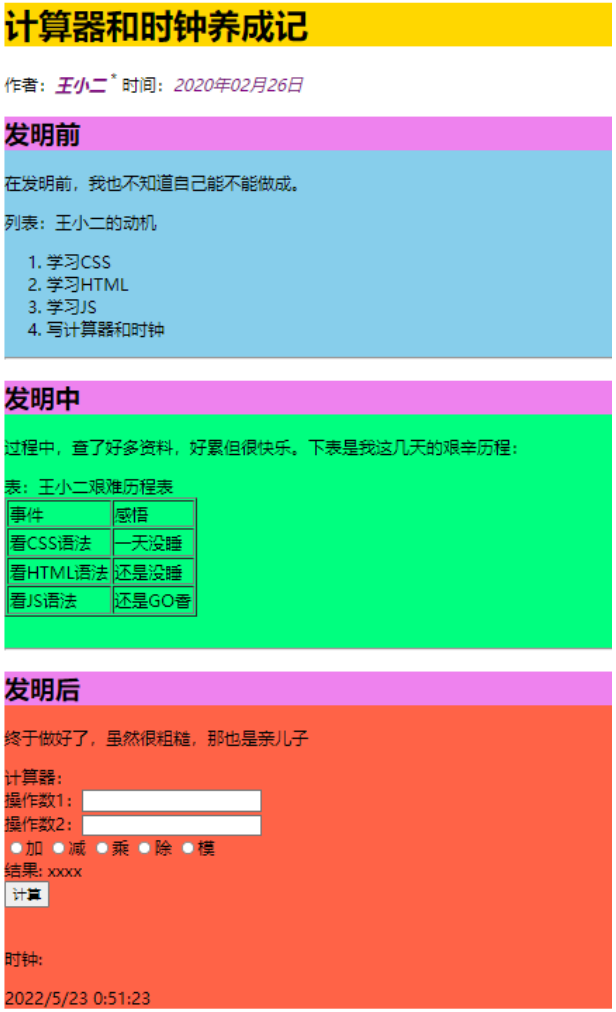
效果：



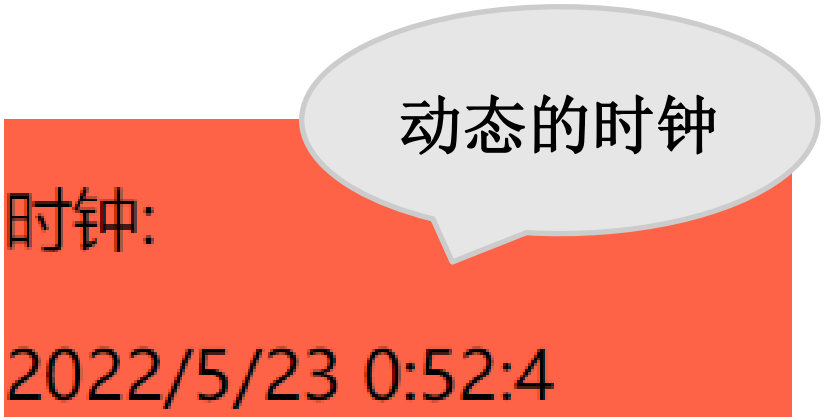
个人作品

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

我们将实现一个网页版时钟：



index.html



第三周 提纲

- 添加静态的时钟
  - JavaScript Date 对象
- 让时钟动起来
  - JavaScript 动画
- Debug 方法
  - console.log 函数
- 对比 JS 和 Go

JavaScript 编程知识—Date 对象

- 王小二写完计算器，开始写网页版时钟，并且希望把年月日、时分秒显示在时钟上，于是学习了JavaScript的Date对象：
- Date对象的**创建**

```
var date = new Date();  // 变量date中存一个Date对象
var date = new Date;    // 创建时可以不加圆括号
```

- Date对象的**方法的使用**

```
date.getDate();          // 返回该月第几天（从1开始）
date.getDay();           // 返回该周第几天（从1开始，星期日为0）
date.getFullYear();      // 返回年份
date.getMonth();         // 返回月份（从0开始）
date.getHours();         // 返回小时（在一天中）
date.getMinutes();       // 返回分钟（在一小时中）
date.getSeconds();       // 返回秒（在一分钟中）
```

添加静态时钟

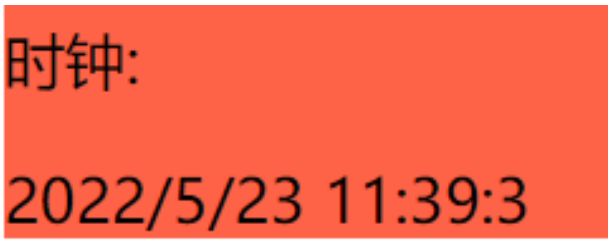
```
<div id="after">
……
Clock:
<p id="clock"></p>
</div>
<script>
  ……
  var date = new Date;
  var year = date.getFullYear();
  var month = date.getMonth() + 1;
  var day = date.getDate();
  var hour = date.getHours();
  var min = date.getMinutes();
  var second = date.getSeconds();
  var time = year + "/" + month + "/" + day + " " + hour + ":  " + min + ":
" + second;
  var clock = document.getElementById("clock");
  clock.innerHTML = time;
</script>
```

get time:year, month, day, hour, minute, second

当操作数之一的类型为字符串时，+运算符表示字符串连接（数字回先转换为字符串）

更改id属性为clock标签之间的内容

效果：



第三周 提纲

- 添加静态的时钟
  - JavaScript Date 对象
- 让时钟动起来
  - JavaScript 动画
- Debug 方法
  - console.log 函数
- 对比 JS 和 Go

JavaScript 动画

- 王小二写了一个时钟，但是这个时钟只是显示了一条日期，并不是一个动态的时钟，为了让这个时钟动起来，王小二学习了下面的函数：
- setTimeout函数
  - 用法：setTimeout(函数名，时间（单位毫秒）,参数1,参数2,…,);
  - 含义：告知浏览器，在指定的毫秒过后，执行由第一个参数指定的函数（以第三个参数以及后面的参数作为此函数的参数）
  - 仅仅告知，不需要等待浏览器执行所告知的函数，告知完继续执行后面的语句
- 使用举例：

```
var a = 0;
setTimeout(console.log,1000,a);  // 1000ms后执行
console.log(a);
```

- 把显示时间的代码放到一个函数里，每 1000ms 就调用一遍：

只需要调用一次 run_clock 函数

```
run_clock();
function run_clock(){
    …(原显示时间代码)
    setTimeout(run_clock,
1000);
}
```

每次函数执行完，都会告诉浏览器，1000ms后再调用“我”一遍

效果：

时钟:

2022/5/23 11:41:47

时钟:

2022/5/23 11:41:57

时钟:

2022/5/23 11:42:12

JavaScript 动画

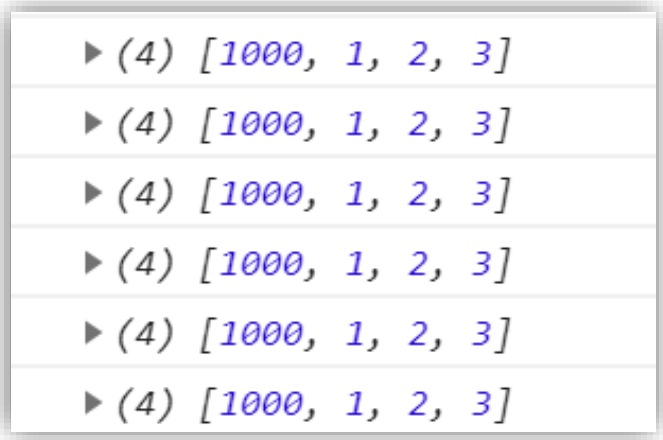
- 至此，王小二成功地完成了网页版时钟，但是王小二在使用setTimeout的时候，巧妙地避开了setTimeout的一个坑：
- 另一个使用setTimeout的例子：

```
var ar=[0,1,2,3];
for(var i=0;i<1000;i++){
        setTimeout(console.log, 1000*i, ar);
        ar[0]++;
}
```

- 控制台上会打印什么？
  - 会逐渐打印[0,1,2,3],[1,1,2,3],…吗?

JavaScript 动画

- 控制台打印：



- 当传递给函数的参数是一个对象时，实际传递的只是对象的地址，而非复制整个对象。
- setTimeout每次把数组ar的地址传给浏览器，而且由于ar只有一个，当setTimeout告知过浏览器1000次后，ar已经变成了[1000,1,2,3]。
- 即使时间达到，setTimeout告知浏览器要执行的函数必须在正常的代码执行完后再执行。
- 于是后面console.log(ar)打印的全部是[1000,1,2,3]。

JavaScript 动画

- 其它使用setTimeout的例子
  - 传递数字
    - 传递的是数字的副本，而非地址
  - 传递对象后，修改保存对象的变量
    - 传递的是原对象的地址
      - 共定义了4个对象，先后赋值给x
      - 每次setTimeout的第3个参数是不同的对象
  - 传递对象后，修改对象的属性
    - 传递的是原对象的地址
      - 共定义了1个对象

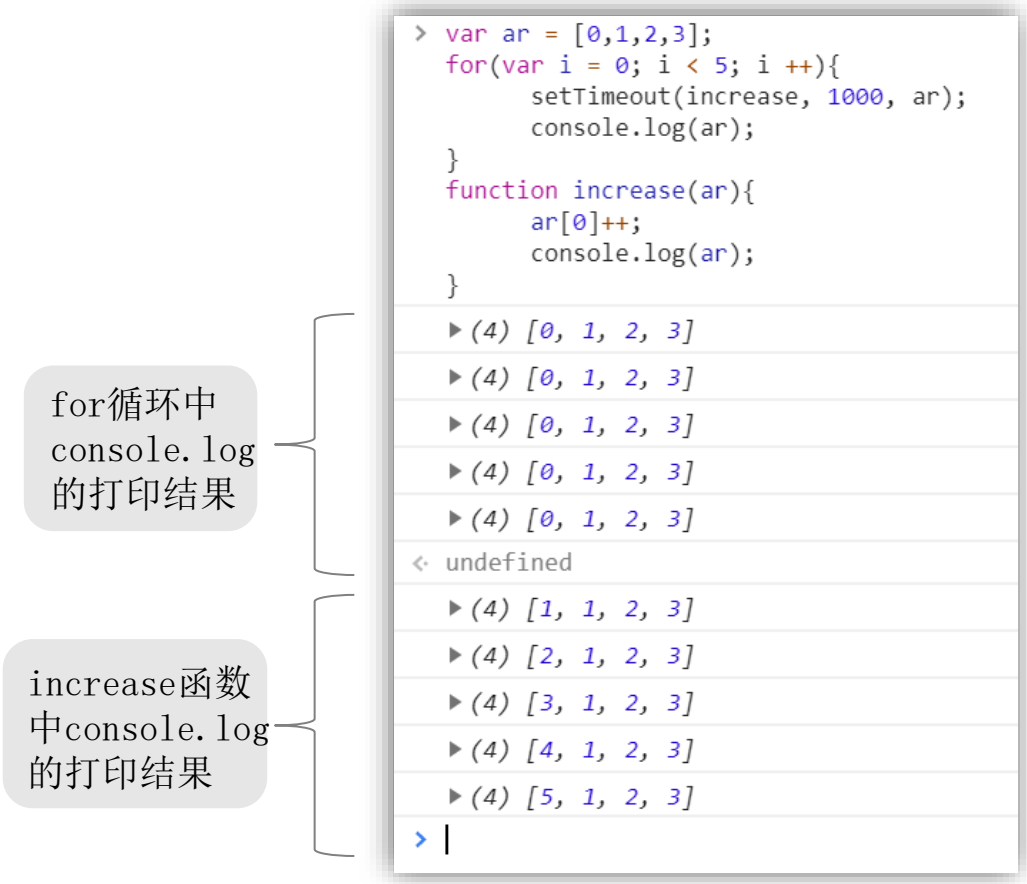
```
> x=2; setTimeout(console.log, 1000, x); x=3; x=4;
< 4    // x=4; 的返回值
  2    // console.log 打印的结果
```

```
> var x=[2,3,4];setTimeout(console.log,1000,x);x=
  [3,3,4];setTimeout(console.log,1000,x);x=
  [4,3,4];setTimeout(console.log,1000,x);x=
  [5,3,4];setTimeout(console.log,1000,x);
< 7 // 最后一次setTimeout的返回值
  ▶ (3) [2, 3, 4]  // 第1次console.log的打印结果
  ▶ (3) [3, 3, 4]  // 第2次console.log的打印结果
  ▶ (3) [4, 3, 4]  // 第3次console.log的打印结果
  ▶ (3) [5, 3, 4]  // 第4次console.log的打印结果
```

```
> var x=[2,3,4]; setTimeout(console.log, 1000, x); x[0]++; x[0]++;
< 3      // x[0]++; 的返回值
  ▶ (3) [4, 3, 4]  // console.log 打印的结果
```

JavaScript 动画

- 其它使用setTimeout的例子
  - for循环
    - 告知浏览器，1000ms后执行increase函数
    - 打印ar
      - increase尚未执行，ar不变
      - increase会在for循环执行完毕后再执行
  - increase函数
    - 1000ms后执行
    - 每次把ar[0]加1
      - 打印出的ar[0]递增



第三周 提纲

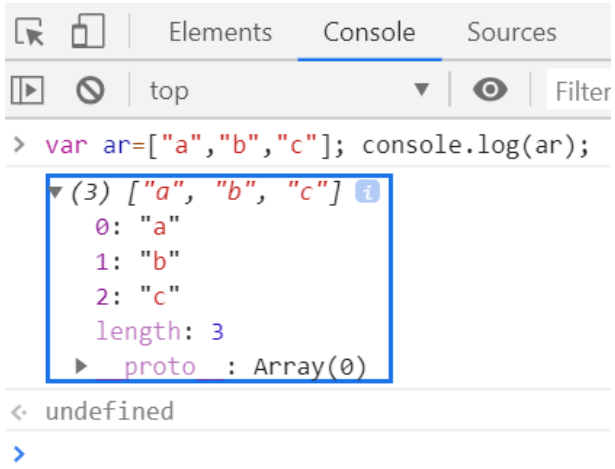
- 添加静态的时钟
  - JavaScript Date 对象
- 让时钟动起来
  - JavaScript 动画
- Debug 方法
  - console.log 函数
- 对比 JS 和 Go

Debug: console.log 函数

- console.log可以打印对象
  - 有时会打印对象的详细信息（属性、方法）
  - 很多JS内置函数的返回值是对象
  - 当不确定某函数的返回值时，可用console.log打印它
- 控制台可以执行用户直接输入的JS语句
- 在控制台输入下面的代码，然后按回车执行，观察控制台输出：

```
var ar = ["a", "b", "c"];
console.log(ar);
```

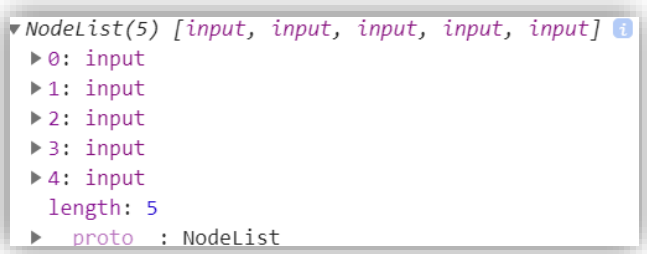
  - 图示使用Chrome浏览器



使用 console.log 的场景

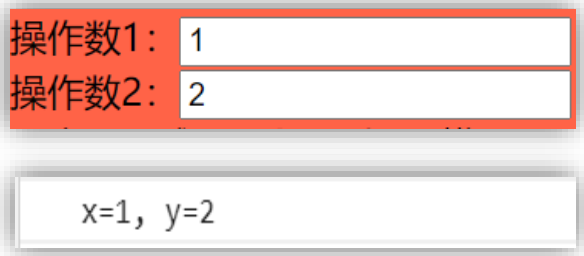
- 查看函数的返回值

```
var options =document.getElementsByName("op");
console.log(options);
```



- 打印中间变量的值

```
var operand1 = document.getElementById("operand1");
var x = operand1.value - 0;
var operand2 = document.getElementById("operand2");
var y = operand2.value - 0;
console.log("x=%d, y=%d", x, y);
```



可以使用 %d (round down),%f,%s 和%o(object)

第三周 提纲

- 添加静态的时钟
  - JavaScript Date 对象
- 让时钟动起来
  - JavaScript 动画
- Debug 方法
  - console.log 函数
- 对比 JS 和 Go

Go vs JS

- 要点
  - Go 是用来做“本地编程”的
    - 直接执行 I/O 操作
      - 操作硬盘中的文件
      - 读取终端的命令行输入
      - 可直接向终端输出信息
    - 所访问的资源均在本地
  - JS 是用来做web（网络）编程的
    - 通过浏览器执行 I/O 操作
      - 操作展示在浏览器里的网页
      - 向浏览器中的 console 里输出信息
    - 所访问的文件可在万维网（WWW）中

Go vs JS

	Go	JavaScript
变量	需要类型，一旦声明，类型不能变 如：var a int = 1	无类型，声明后可以把不同类型的值赋值给它，如：var a=1; a=“JS”;
数据类型	比JS多了数组、切片、字符	JS里没有切片，数组是通过“对象”实现的 JS里只有字符串类型，没有字符类型
函数	需要参数类型，返回值类型，可以有多个返回值	不需要参数类型，不需要写返回值类型，只能有一个返回值 函数是一种特殊“对象”，可组成函数数组
算术运算符	5/2=2　5.0/2=2.5 3.2 % 0.7 编译时报错	5/2=2.5 3.2 % 0.7 ≈ 0.4 (损失精度)
比较运算符	不同类型值的比较编译时会报错	如果用==(或!=)比较，不同类型的值先自动转换成相同类型，再比较 如果用===(或!==)比较，不同类型的值比较的结果是false(或true)
右移运算符 >>	A>>B A是无符号数：高位补0 A是有符号数：高位补符号位	A>>B：高位补0 A>>>B：高位补符号位

Go vs JS

	Go	JavaScript

控制流	条件不用加括号
	if x<10 {
	……
	}
		条件需要加括号
		if (x<10){
		……
		}

对象	没有讲	Array对象、Date对象、代表HTML元素的对象等
打印函数	fmt.Println, fmt.Printf	console.log
数组长度	len(array)	array.length
语言类型	编译型，需先编译成可执行文件，再运行可执行文件	解释型，浏览器内置的解释器一句句解释执行JS代码
语句	结尾不需要分号	结尾分号可有可无

Quicksort – Go vs JS

- Go
  - Go中创建切片、数组需要指定长度
  - Go可以传递切片，切片有长度，所以不需要传递待排数据的边界
  - fmt.Println 可直接输出到终端

```
func main() {
	rand.Seed
(time.Now().UnixNano())
	n :  = 1*1024*1024
	var da = make([]int,n)
	for i:  =0; i<n; i++ {
		da[i] = rand.Int()
	}
	quicksort (da[0:  n])
	fmt.Printf(" %d\n %d\n
%d\n",da[0],da[1],n-1,da[n-1])
}
```

- JS
  - JS里创建数组不需要指定长度
  - JS里生成随机数方式和Go不同
  - JS里没有切片，所以需要传递待排数据的边界
  - console.log 只能输出到控制台

Math.random返回[0,1)之间的随机数

```
<script>
  var ar = new Array();
  var n = 1*1024*1024;
  for(var i=0; i<n; i++){
    ar[i] = Math.floor(Math.random()*1000000);
  }
    quicksort(ar, 0, ar.length-1);
    console.log(" %d\n %d\n %d\n", ar[0],
ar[1], ar[ar.length-1]);
      ……
```

Quicksort – Go vs JS

- Go
  - if后面的条件不需要圆括号
  - array是一个切片，代表了原始数组的一段，可以直接用len(array)获取待排数据的长度
  - 划分函数partition只需要当前数组作参数
- JS
  - if后面的条件需要圆括号
  - 由于传递了数组的边界，所以需要通过边界来计算出待排数据长度
  - 划分函数partition还需要当前数组的边界作参数

```
func quicksort(array []int) {
  if len(array) < 2 {
    return
  }
  left_array, right_array := partition(array)
  quicksort(left_array)
  quicksort(right_array)
}
```

```
function quicksort(ar, left, right){
  if (right - left + 1 < 2) {
    return;
  }
  var pivotal_index=partition(ar, left, right);
  quicksort(ar, left, pivotal_index-1);
  quicksort(ar, pivotal_index+1, right);
}
```

Quicksort—Go vs JS

- Go

生成随机数的范围是[0, len(array)]

```
func partition(array []int) ([]int, []int)
{
 pivotal_index := rand.Intn(len(array))
 pivotal_value := array[pivotal_index]
 next := 0
 array[pivotal_index], array[len(array)-1]
=array[len(array)-1], array[pivotal_index]

 for i:  = 0; i < len(array); i++ {
   if (array[i] < pivotal_value) {
     array[next], array[i] =
     array[i], array[next]
     next++
   }
 }
 array[next], array[len(array)-1]=
 array[len(array)-1], array[next]
 return array[0: next], array[next+1:
len(array)]
}
```

可以使用同时给两个数赋值的语句来达到交换两个值的目的

返回两个切片

- JS

生成随机数的范围是[left, right]

```
function partition(ar, left, right){
      var len = right - left + 1;
      var pivotal_index =
Math.floor(Math.random()*len) + left;
      var pivotal_value =
ar[pivotal_index];
      var next=left;
      // pivotal to last
      ar[pivotal_index] = ar[right];
      ar[right] = pivotal_value;
      // smaller than pivotal :   left
      for(var i=left; i<=right; i++){
            if(ar[i] < pivotal_value) {
                  // swap
                  var temp = ar[next];
                  ar[next] = ar[i];
                  ar[i] = temp;
                  next++;
            }
      }
      // swap pivotal back
      ar[right] = ar[next];
      ar[next] = pivotal_value;
      return next;
}
```

交换两个值的时候，需要先把一个值存到另一个临时变量里

返回标杆下标

评分标准

- 教师评分+同学互评，每个同学评7份作品
  - 同学互评开放时间：6月28日 14:00 - 7月2日 23:55
  - 要在7份作品中明确指明最优秀和最不优秀作品

分数	评判标准
4	独立完成且正确，少量使用他人的素材（有标注致谢），有创意
3	独立完成且大部分正确，少量使用他人的素材（有标注致谢）
2	完成且基本正确（如代码运行但结果有错），并标注致谢了使用他人的素材
1	截止时间前提交，但有明显错误（如代码不能运行）或缺失 50%标注致谢
0	未在截止时间前提交，或抄袭

注意事项

- 致谢以文字的形式在页面底部显示（而不是注释）
  - 如果引用外部图片等资源，必须明确指明图片等资源链接
- 标注创造性：在<head>部分以注释的形式说明创造性
- **不允许使用Vue等框架，外部代码占比<10%**
  - 如果已经使用并且完成了网页，请联系实验课助教

补充材料

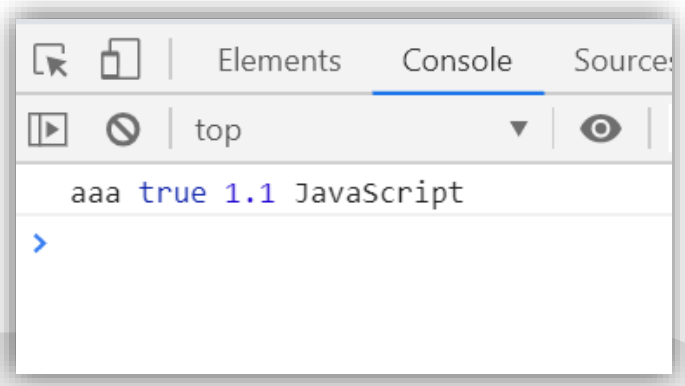
补充材料 提纲

- JavaScript 变量 & 函数
- JavaScript 操作符 & 表达式
- JavaScript 控制流
  - while 循环
  - if-else 语句
  - switch-case 语句

JavaScript 基础–基本数据类型

类型	说明	举例
布尔	表示真、假，只有两个值：true、false	true、false
数字	64bit长，可以同时表示浮点数和整数，相当于Go语言中float64	-1，0，1，0.1，-0.1
字符串	表示一个字符串	"abc"，""（空串）
null类型	null值属于一个只包含该值的特殊类型。 null表示此处不应该有值	null
undefined类型	undefined值属于一个只包含该值的特殊类型。 undefined表示缺少值： （1）变量声明后未赋值 （2）函数调用时没提供的参数 （3）函数没有返回值	undefined

JavaScript 编程知识–变量



- <script> 部分：

```
var x,y,z;      变量声明（不需要类型）
var s = "JavaScript";   声明并赋值
x = 0;          变量赋值
x = "aaa";      变量赋值（先后给同一个变量赋不同类型的值）
y= true;        变量赋值
z = 1.1;        变量赋值
console.log(x,y,z,s);  ⟶ 同时打印多个变量值，用逗号隔开
```

JavaScript 编程知识–函数

- 函数定义

相比Go，无需定义参数和返回值类型

JS:

```
function
add(x,y){
    return x
+ y;
}
```

Go:

```
func add(x,y int)
int{
    return x + y
}
```

函数体，只能返回至多一个值（而Go中可返回多个）

- 函数调用

```
var a=1,b=1;
var z = add(a,b); // z=2;
```

补充材料 提纲

- JavaScript 变量 & 函数
- JavaScript 操作符 & 表达式
- JavaScript 控制流
  - while 循环
  - if-else 语句
  - switch-case 语句

JavaScript 运算符 & 表达式

- 与Go相同的算术运算符
  - +,-,*,++,--
- 与Go不同的算术运算符
  - /（除法）
    - JS中：3/2=1.5，Go中：3/2=1（向下取整）
    - 如果JS中要实现向下取整，需要使用Math.floor(3/2)
  - %（取余）
    - JS中 % 的两操作数可以有小数，而Go不允许这样做。
    - JS中：3.2 % 0.7 ≈ 0.4(有精度损失)，Go中：不允许 3.2%0.7

JavaScript 运算符 & 表达式

- JS里运算符不只有算术运算符，Go里有的几乎都能在JS里找到对应的：
- 赋值运算符
  - =，+=，-=，*=，/=，%=
  - x op= y相当于x=x op y，如x+=y相当于x=x+y
    - op可以是+、-、*、/、%

```
> console.log(false==0)
  true
< undefined
> console.log(false===0)
  false
< undefined
```

- 比较运算符
  - 比较运算的结果是true或false
    - 与Go相同：>,>=,<,<=
    - 与Go不同：==,!=,===,!==
      - 使用==或!=比较时，若两被比较数类型不同，先自动转换成相同类型再比较
      - 使用===(!==)比较时，若两被比较数类型不同，则比较结果为false(true)

JavaScript 运算符 & 表达式

- 逻辑运算符
  - 与Go中相同：&&（与），||（或），!（非）
  - 经常在**条件表达式**里使用
  - Eg: `if( x<10 && x>0 ) { … }`

- 位运算符
  - 与Go相同：&（按位与），|（按位或），^（按位异或），~（按位取反），<<（左移）
  - 与Go不同：
    - >>（右移，左边补符号位，负数补1，非负数补0）
    - >>>（右移，左边补0）
  - 当进行位运算时，参与运算的整数被当作32位整数

JavaScript 运算符 & 表达式

- 条件运算符（一种三元运算符）
  - 条件 ？条件为真时表达式的值 ： 条件为假时表达式的值
  - 比如：`var y = x>0?x:-x; //y=x的绝对值`

- 字符串连接运算符
  - +（加号，和加法用同一个）
  - 当两个操作数至少有一个是字符串时，将不是字符串的操作数转化成字符串，然后连接：

```
var x = "Java" + "Script"; //x="JavaScript"
var x = "0" + 1;  //x= "01" （1先被转换成字符串）
```

补充材料 提纲

- JavaScript 变量 & 函数
- JavaScript 操作符 & 表达式
- JavaScript 控制流
  - while 循环
  - if-else 语句
  - switch-case 语句

JavaScript 控制流—while 循环

- while 循环：

```
while(表达式){
    // 一些代码
}
```

- 每次循环，若表达式为真，则执行循环体；否则跳过循环体，停止循环。

JavaScript 控制流–if-else 语句

- if-else 语句：
  - 同Go类似，但注意表达式需要加圆括号。

```
if(表达式1){
    //一系列语句
}else if (表达式2){
    //一系列语句
}else{
    //一系列语句
}
```

可以有零个或多个else分支

JavaScript 控制流—switch-case语句

- switch-case 语句:
  - 可代替 if-else 语句

```
switch(表达式){
    case 值1: //一系列代码，当表达式的值等于值1时执行
    break;    //不要忘记break;否则，已匹配case语句后面所有语句都会无条件执行，
              //直到碰到后面的break或者到达整个switch-case语句末尾
    case 值2: //一系列代码，当表达式的值不等于值1且等于值2时执行
    break;
    default:  //一系列代码，当表达式的值既不等于值1也不等于值2时执行
                  //default可以省略
}
```