中国人民大学 信息学院
RENMIN UNIVERSITY OF CHINA SCHOOL OF INFORMATION



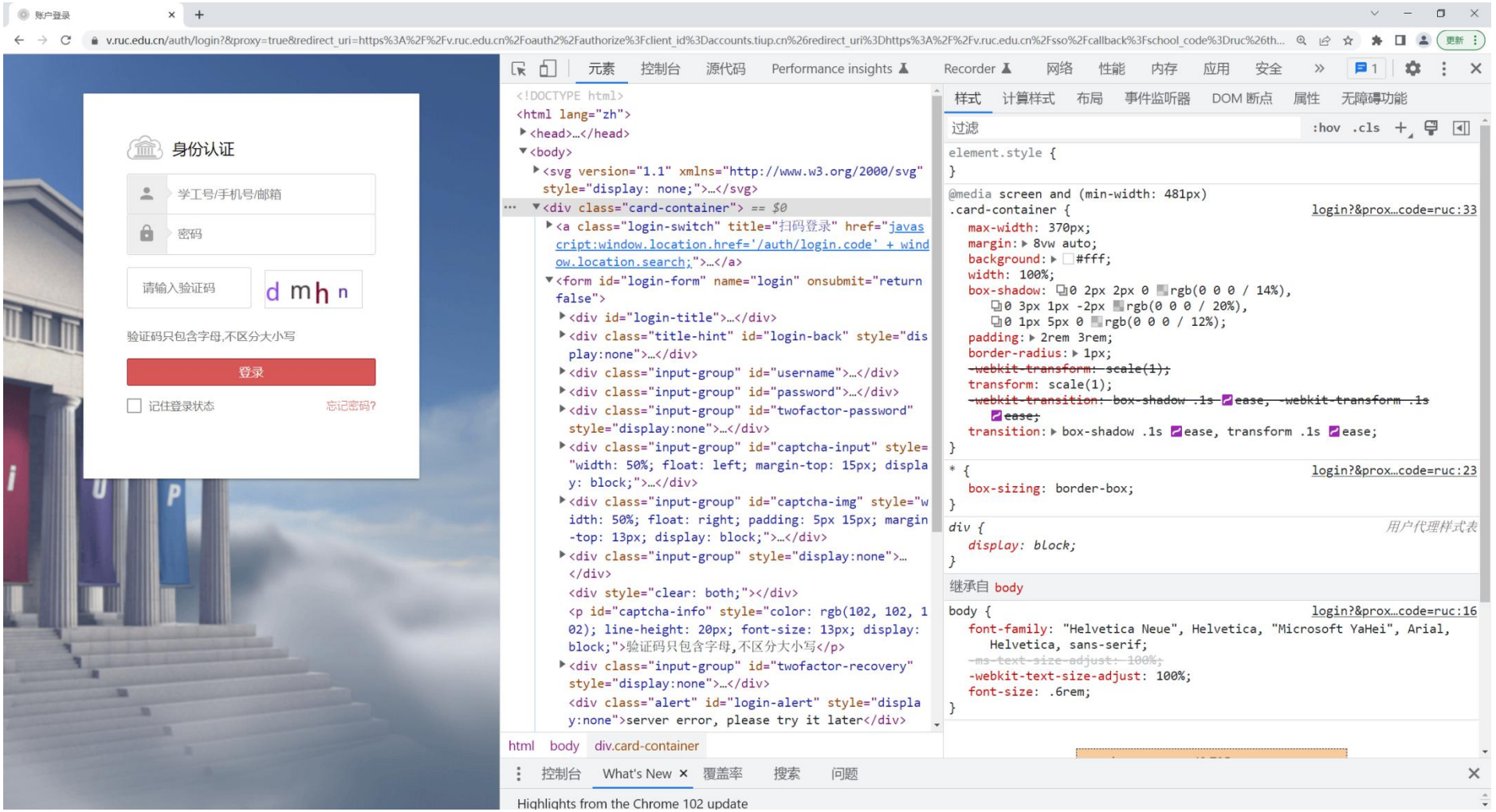
# 2. Web应用开发——前端

授课教师：游伟 副教授

授课时间：周五16:00 – 17:30（教二2406）

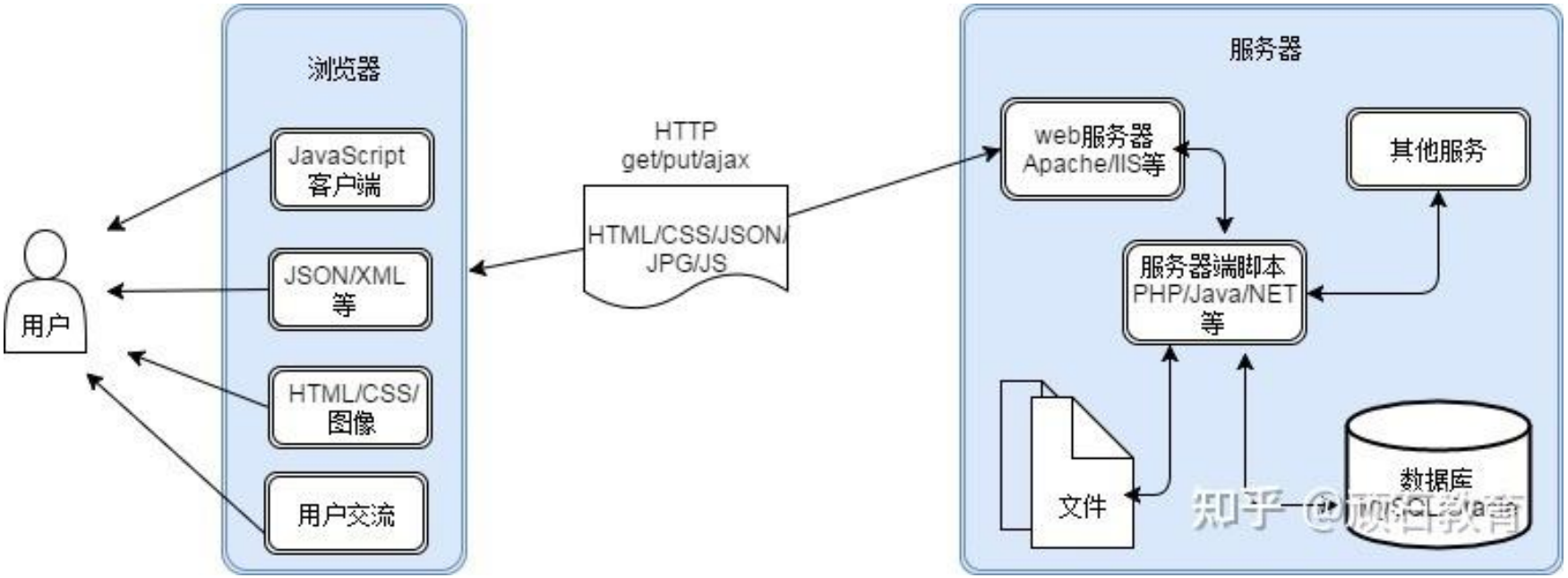
课程主页：https://www.youwei.site/course/websecurity

# 引子

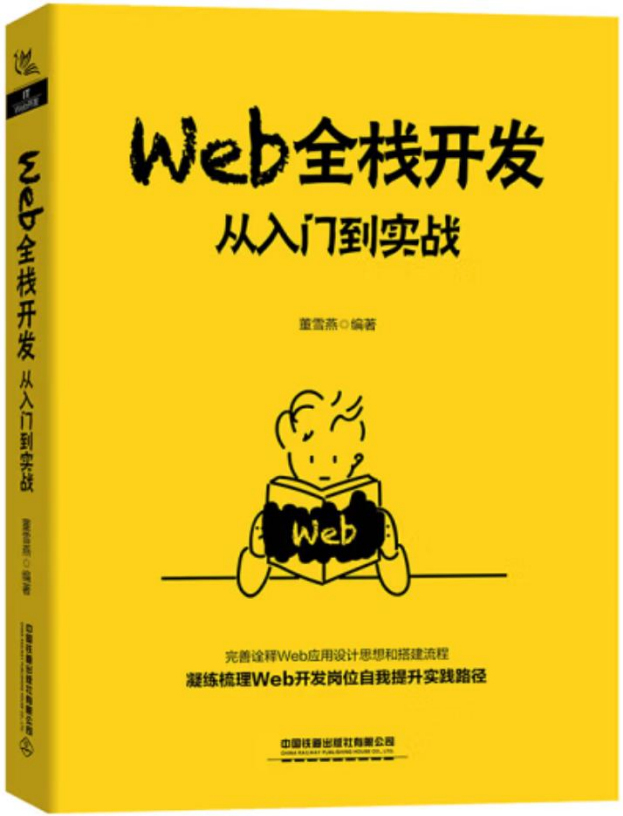

账户登录
v.ruc.edu.cn/auth/login?&proxy=true&redirect_uri=https%3A%2F%2Fv.ruc.edu.cn%2Foauth2%2Fauthorize%3Fclient_id%3Daccounts.tiup.cn%26redirect_uri%3Dhttps%3A%2F%2Fv.ruc.edu.cn%2Fsso%2Fcallback%3Fschool_code%3Druc%26th...
身份认证
学工号/手机号/邮箱
密码
请输入验证码
d mh n
验证码只包含字母,不区分大小写
登录
记住登录状态
忘记密码?
元素 控制台 源代码 Performance insights Recorder 网络 性能 内存 应用 安全
样式 计算样式 布局 事件监听器 DOM 断点 属性 无障碍功能
过滤 :hov .cls
element.style {
}
@media screen and (min-width: 481px)
.card-container { login?&prox…code=ruc:33
max-width: 370px;
margin: 8vw auto;
background: #fff;
width: 100%;
box-shadow: 0 2px 2px 0 rgb(0 0 0 / 14%), 0 3px 1px -2px rgb(0 0 0 / 20%), 0 1px 5px 0 rgb(0 0 0 / 12%);
padding: 2rem 3rem;
border-radius: 1px;
-webkit-transform: scale(1);
transform: scale(1);
-webkit-transition: box-shadow .1s ease, -webkit-transform .1s ease;
transition: box-shadow .1s ease, transform .1s ease;
}
* { login?&prox…code=ruc:23
box-sizing: border-box;
}
div { 用户代理样式表
display: block;
}
继承自 body
body { login?&prox…code=ruc:16
font-family: "Helvetica Neue", Helvetica, "Microsoft YaHei", Arial, Helvetica, sans-serif;
-ms-text-size-adjust: 100%;
-webkit-text-size-adjust: 100%;
font-size: .6rem;
}
html body div.card-container
控制台 What's New 覆盖率 搜索 问题
Highlights from the Chrome 102 update

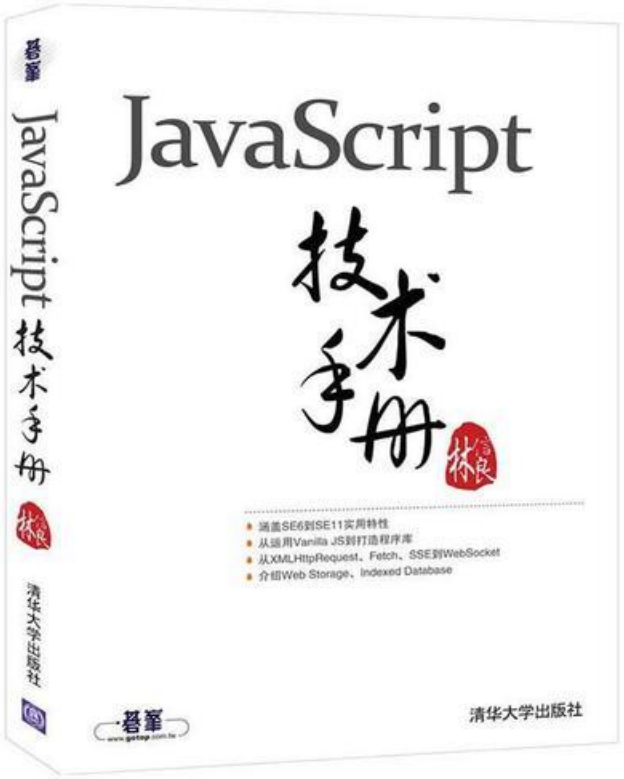
# Web应用开发模式

- 前端技术：HTML + CSS + JavaScript
- 后端技术：PHP/ASP/JSP + 数据库 + 数据处理
- 前后端通信：HTTP + AJAX



# 推荐图书

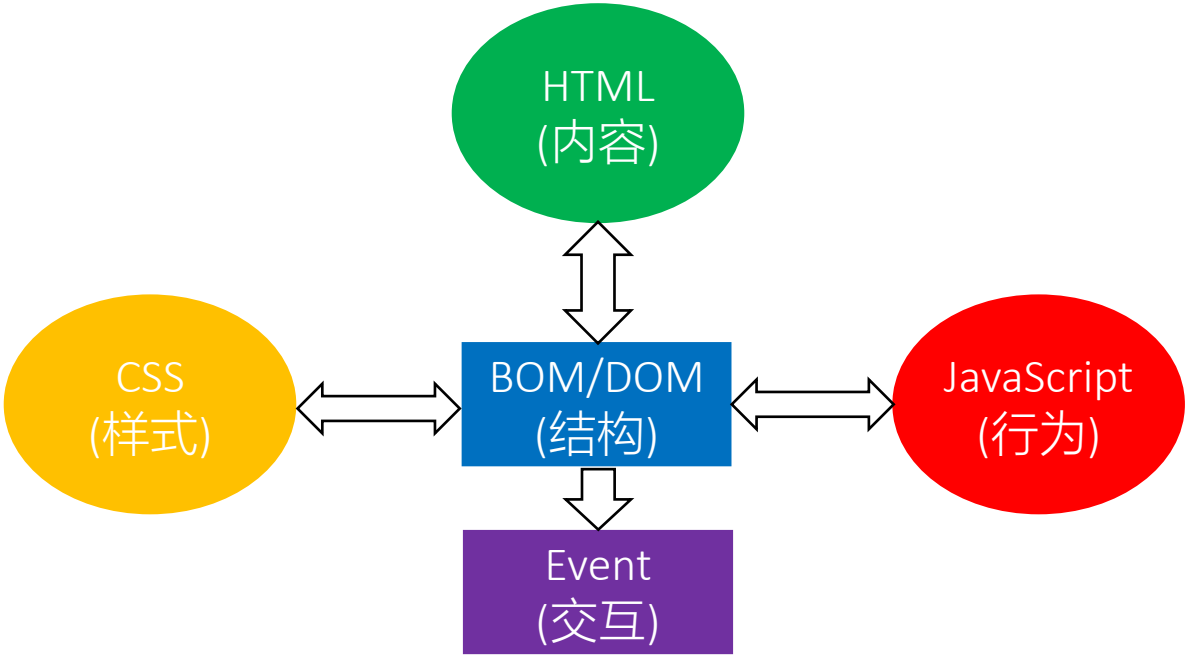

Web全栈开发
从入门到实战
董雪燕 编著
Web
完善诠释Web应用设计思想和搭建流程
凝练梳理Web开发岗位自我提升实践路径
中国铁道出版社有限公司


JavaScript
技术手册
林良
涵盖SE6到SE11实用特性
从运用Vanilla JS到打造程序库
从XMLHttpRequest、Fetch、SSE到WebSocket
介绍Web Storage、Indexed Database
碁峰
清华大学出版社


第2版
PHP核心技术与最佳实践
PHP Core Technology and Best Practice, Second Edition
PHP核心技术
与最佳实践
机械工业出版社
China Machine Press

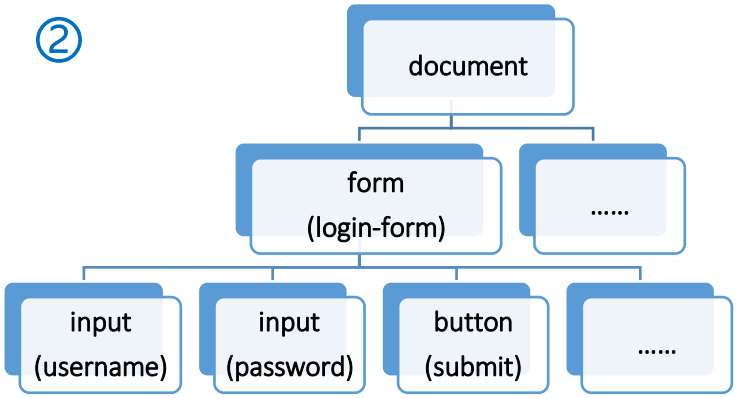
# Web前端架构

- 三种语言：HTML + CSS + JavaScript
- 两个机制：BOM/DOM (浏览器文档对象模型) + Event (事件)



# Web前端示例

①


② 


③

| 对象 | 事件 | 响应 |
|---|---|---|
| button (submit) | onclick | login() |
| ..... | ..... | ..... |

④
```
<form id="login-form">
  <input id="username" ... />
  <input id="password" ... />
  <button id="submit" ... />
  .....
</form>
```

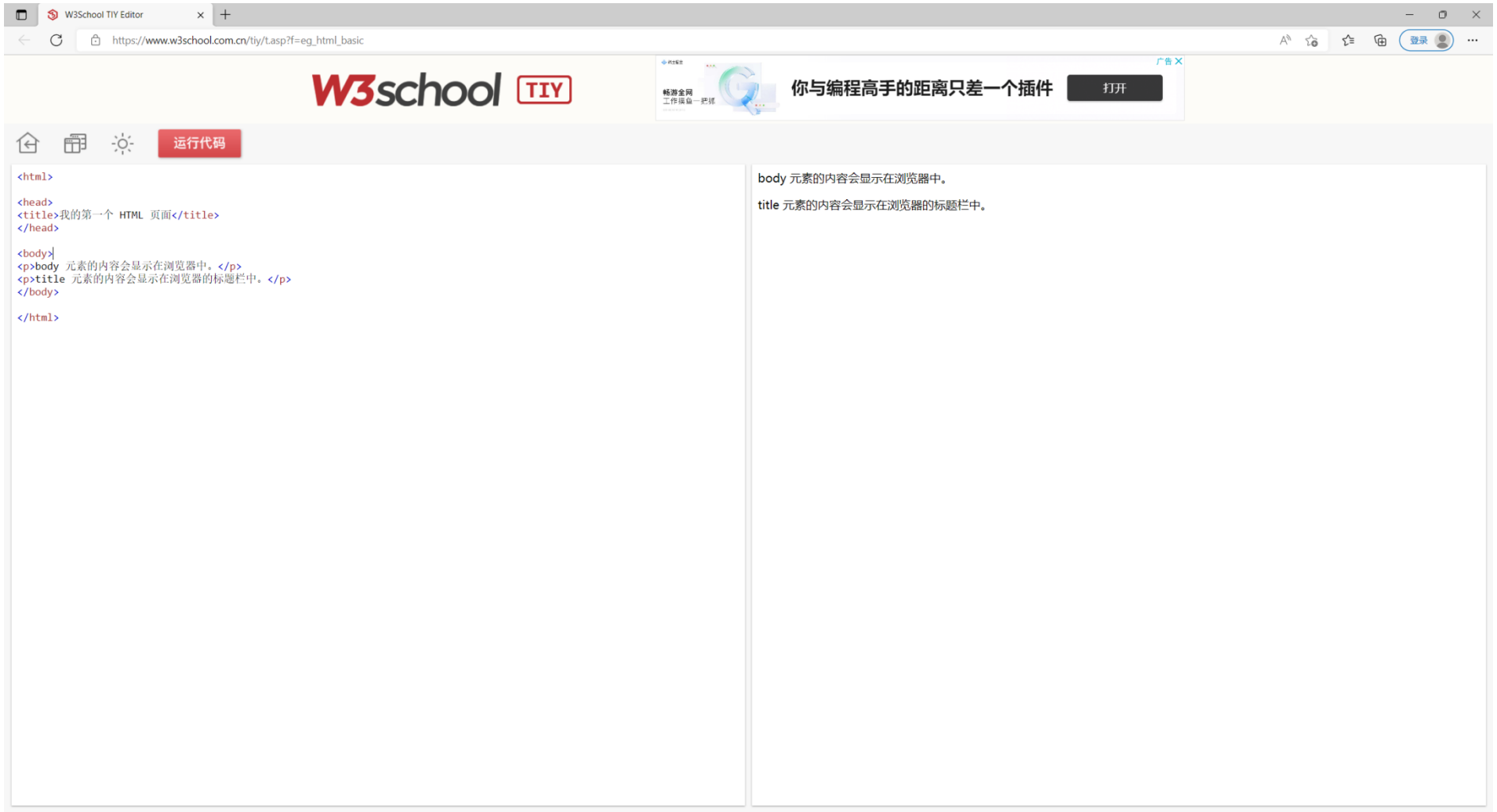
⑤
```
<style>
  .input { background-color: white;
           height: 45px; .....
         }
  .....
</style>
```

⑥
```
<script>
 function login() {
   result = check_verification();
   .....
 }
 button=getElementById("submit");
 button.onclick = login;
</script>
```

① 页面显示
② BOM/DOM
③ Event
④ HTML
⑤ CSS
⑥ JavaScript

# 前端在线编辑平台

- https://www.w3school.com.cn/tiy/t.asp?f=eg_html_basic



# 目录

# 2.1 HTML

- HTML是HyperText Markup Language（超文本标记语言）的缩写，是一种为普通文件中某些字句加上标签的语言，其目的是运用标签（tag）对文件达到预期的效果

## HTML 代码

```
<html>
  <head>
    <title>学习 HTML</title>
  </head>
  <body>
    <h1>欢迎来到HTML的世界</h1>
  </body>
</html>
```

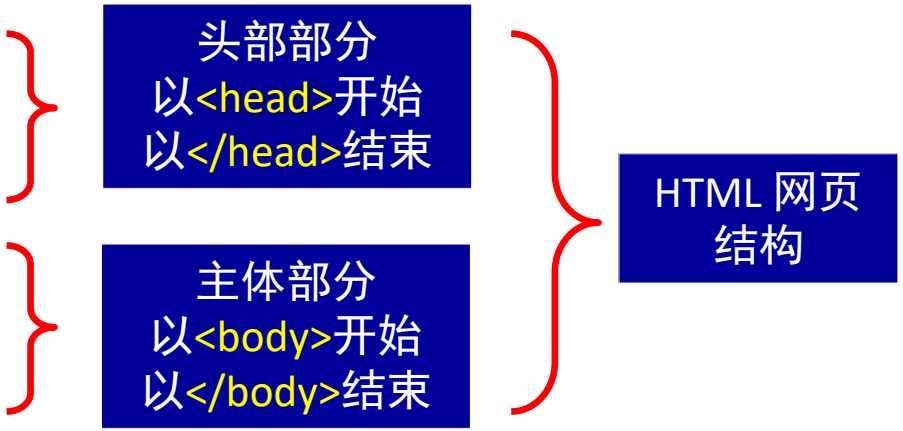
浏览器处理代码并显示


学习更多：https://www.w3school.com.cn/html/index.asp

# 2.1.1 HTML文档结构

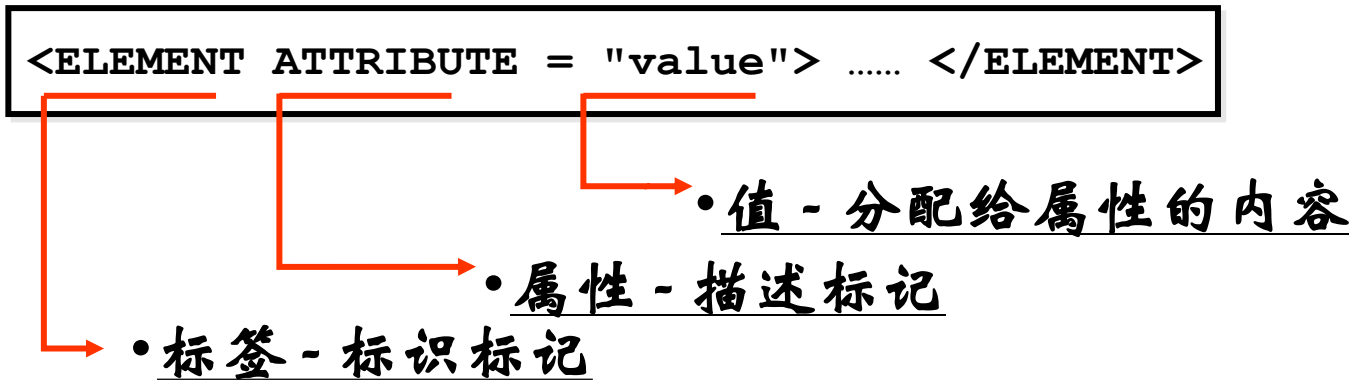
- HTML 文档以<html>...</html>标签标记开始和结束
  - 头部部分：<head>...</head>，包括标题和其他说明信息
  - 主体部分：<body>...</body>，包括文本、图像、链接、表单等

```
<html>
  <head>
    <title>学习 HTML</title>
  </head>
  <body>
    <h1>欢迎来到HTML的世界</h1>
  </body>
</html>
```



# 2.1.2 标签、属性、元素

- 标签是由尖括号包围的关键词，是HTML的基本组成单位
  - 大部分标签是双标记标签，由起始标记和闭合标记组成，如<h1></h1>
  - 一些标签是单标记标签，在起始标记中进行闭合，如<br/>
- 标签内部可以添加属性，属性的定义方式为一个键值对
  - 一个标签可以有多个属性
  - 属性值应该被包含在引号中
- 元素指的是从开始标签到结束标签所有的符号

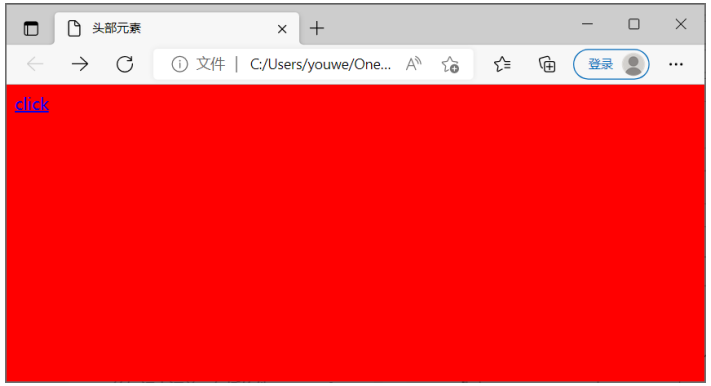


# 2.1.3 常见的头部元素

```
<html>
  <head>
    <title>头部元素</title>
    <base href="http://www.youwei.site/base/"
          target="_blank"/>
    <link rel="stylesheet" href="mystyle.css"/>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8"/>
    <script type="text/javascript">
      alert('hi');
    </script>
    <style type="text/css">
      body { background-color: red; }
    </style>
  </head>
  <body>
    <a href="test.html">click</a>
  </body>
</html>
```

由于头部base元素的设置，点击超链接将在空白页面（_blank），打开http://www.youwei.site/base/test.html。

| 标签 | 描述 |
| --- | --- |
| <head> | 定义关于文档的信息。 |
| <title> | 定义文档标题。 |
| <base> | 定义页面上所有链接的默认地址或默认目标。 |
| <link> | 定义文档与外部资源之间的关系。 |
| <meta> | 定义关于 HTML 文档的元数据。 |
| <script> | 定义客户端脚本。 |
| <style> | 定义文档的样式信息。 |



# 2.1.4 常见的主体元素

- 标头
- 段落
- 换行
- 字体
- 文本格式化
- 预格式化文本
- 水平分割线
- 图片
- 超链接
- 列表
- 表格
- 表单
- 区块
- 框架

# 标头

- 使用<h1>到<h6>标签，指定不同级别的标头（heading）

```
<html>
  <head>
    <title>动物世界</title>
  </head>
  <body>
    <h1>从大自然获得灵感</h1>
    <h2>从大自然获得灵感</h2>
    <h3>从大自然获得灵感</h3>
    <h4>从大自然获得灵感</h4>
    <h5>从大自然获得灵感</h5>
    <h6>从大自然获得灵感</h6>
  </body>
</html>
```

# 段落

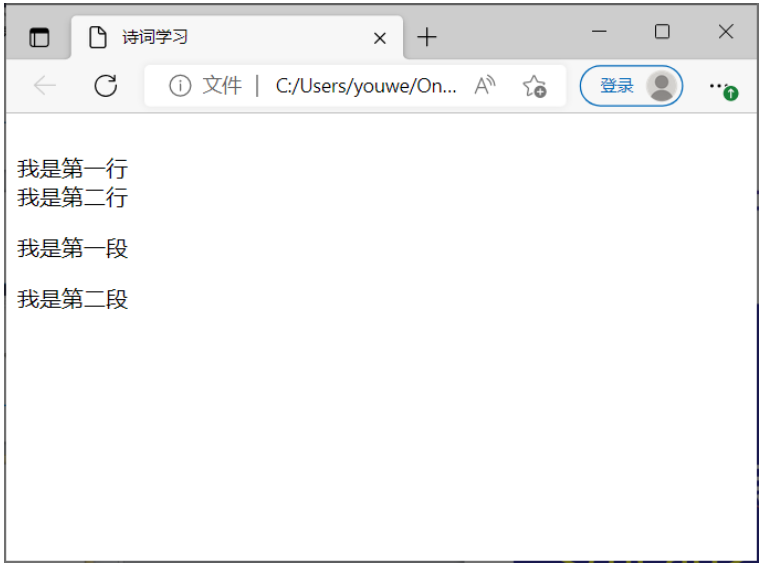
- 使用 ... 标签用于标记段落（paragraph）
- 属性align用于控制段落对齐方式

```
<html>
  <head>
    <title>诗词学习</title>
  </head>
  <body>
    <p>我是第一段</p>
    <p>我是第二段</p>
    <p align="left">左对齐显示</p>
    <p align="center">居中显示</p>
    <p align="right">右对齐显示</p>
  </body>
</html>
```

# 换行

- 文本中用<br/>标签，会强制换行（break）

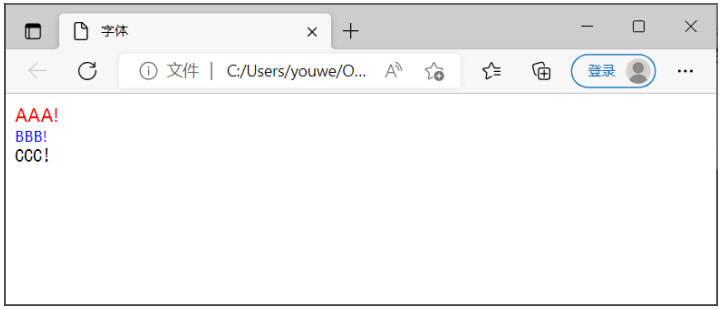
```
<html>
  <head>
    <title>诗词学习</title>
  </head>
  <body>
    <br/>我是第一行<br/>我是第二行
    <p>我是第一段</p>
    <p>我是第二段</p>
  </body>
</html>
```



# 字体

- 使用<font>标签，规定文本的字体、字体尺寸、字体颜色

```
<html>
  <head>
    <title>字体</title>
  </head>

  <body>
    <font size="3" color="red">AAA!</font>
    <br/>

    <font size="2" color="#0000FF">BBB!</font>
    <br/>

    <font face="SimHei">CCC!</font>
    <br/>
  </body>
</html>
```



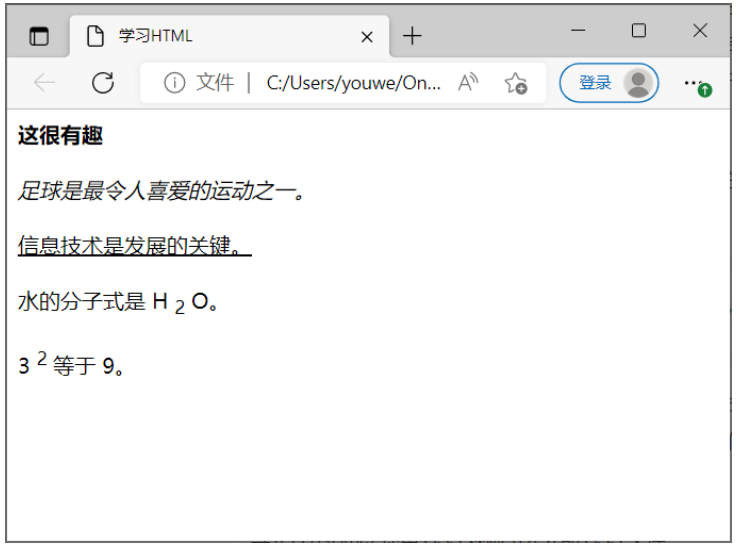

| 属性 | 值 | 描述 |
| --- | --- | --- |
| color | *rgb(x,x,x)*<br>*#xxxxxx*<br>*colorname* | 不赞成使用。请使用样式取代它。<br>规定文本的颜色。 |
| face | *font_family* | 不赞成使用。请使用样式取代它。<br>规定文本的字体。 |
| size | *number* | 不赞成使用。请使用样式取代它。<br>规定文本的大小。 |

| 颜色 | 3位十六进制颜色值 | 6位十六进制颜色值 | RGB |
| --- | --- | --- | --- |
| | #000 | #000000 | rgb(0,0,0) |
| | #F00 | #FF0000 | rgb(255,0,0) |
| | #0F0 | #00FF00 | rgb(0,255,0) |
| | #00F | #0000FF | rgb(0,0,255) |
| | #FF0 | #FFFF00 | rgb(255,255,0) |
| | #0FF | #00FFFF | rgb(0,255,255) |
| | #F0F | #FF00FF | rgb(255,0,255) |
| | #888 | #888888 | rgb(136,136,136) |
| | #FFF | #FFFFFF | rgb(255,255,255) |

# 文本格式化

- HTML使用标签对输出的文本进行格式

```
<html>
  <head>
    <title>学习HTML</title>
  </head>

  <body>
    <p>
      <b>这很有趣</b>
      <br/><br/>

      <i>足球是最令人喜爱的运动之一。</i>
      <br/><br/>

      <u>信息技术是发展的关键。</u>
      <br/><br/>

      水的分子式是 H <sub>2</sub> O。
      <br/><br/>

      3 <sup>2</sup> 等于 9。
      <br/><br/>
    </p>
  </body>
</html>
```

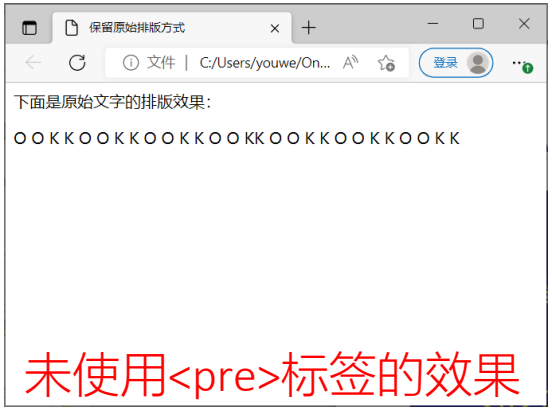



| 名称 | HTML代码 |
|---|---|
| 黑体 |  |
| 斜体 |  |
| 下划线 | <u></u> |
| 中划线 | <s></s> |
| 闪烁 | <blink></blink> |
| 上标 | <sup></sup> |
| 下标 | <sub></sub> |

# 预格式化文本

- <pre> 标签用于显示具有预先定义格式的文本（Preformatted）

```
<html>
  <head>
    <title>保留原始排版方式</title>
  </head>

  <body>
    <p>下面是原始文字的排版效果：</p>

    <pre>
            O  O                  K      K
         O        O               K    K
       O            O             K  K
      O               O            KK
       O             O             K  K
         O         O               K    K
            O  O                  K      K
    </pre>
  </body>
</html>
```

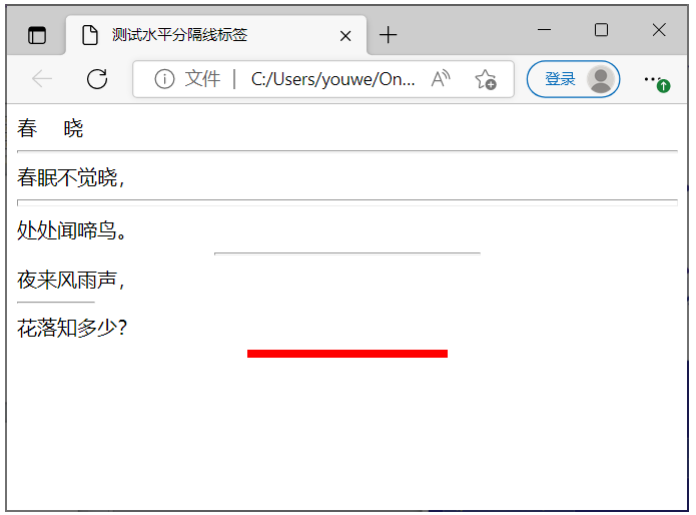



# 水平分割线

| 属性 | 功能 | 单位 | 默认值 |
| --- | --- | --- | --- |
| size | 设置水平分隔线的粗细 | pixel | 2 |
| width | 设置水平分隔线的宽度 | pixel，% | 100% |
| align | 设置水平分隔线的对齐方式 | | center |
| color | 设置水平分隔线的颜色 | | black |

- <hr/>用于段落与段落之间的分隔（Horizontal Rule）
- 通过设置<hr/>标签的属性值，可以控制水平分隔线的样式

```
<html>
  <head>
    <title>测试水平分隔线标签</title>
  </head>
  <body>
    春　晓
    <hr/>
    春眠不觉晓，
    <hr size="6"/>
    处处闻啼鸟。
    <hr width="40%"/>
    夜来风雨声，
    <hr width="60" align="left"/>
    花落知多少？
    <hr size="6" width="30%" align="center" color="red"/>
  </body>
</html>
```



# 图片

| 属性 | 功能 | 单位 | 默认值 |
|---|---|---|---|
| src | 指定图像的URL地址 | | |
| width | 设置图像宽度 | pixel, % | |
| height | 设置图像高度 | pixel, % | |
| align | 指定图像与文字的对齐关系 | | bottom |

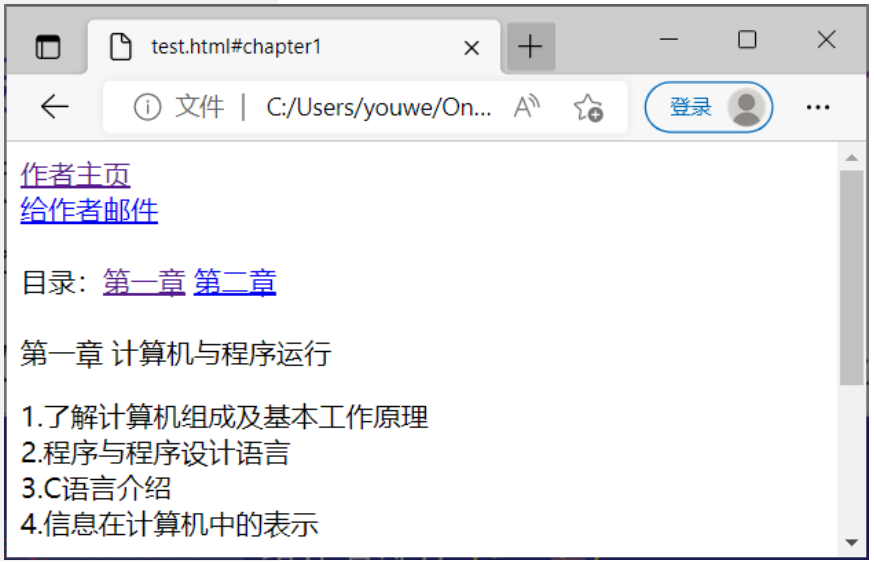
- 图像由<img>标签定义（Image）

```
<html>
  <head>
    <title>仙剑世界</title>
  </head>

  <body>
    <h1>
      <font size="3" color="green">
        <b>李逍遥和赵灵儿</b>
      </font>
    </h1>

    <p><img align="bottom" src="pic.jpg" height=100/>底部对齐</p>
    <p><img align="top" src="pic.jpg" height=100/>顶部对齐</p>
    <p><img align="middle" src="pic.jpg" height=100/>居中对齐</p>
    <p><img src="pic.jpg" height=100>默认对齐</p>
  </body>
</html>
```

# 超级链接

- <a>标签用于创建超级链接（Anchor）
  - 网页链接：跳转到另一个页面
  - 电子邮件链接：发送电子邮件
  - 长文档内部链接：跳转到文档特定书签处

```
<body>
  <a href="https://www.youwei.site">作者主页</a> <br/>
  <a href="mailto:youwei@ruc.edu.cn">给作者邮件</a> <br/>
  目录：
  <a href="#chapter1">第一章</a>
  <a href="#chapter2">第二章</a>
  <br/><br/>
  <a name="chapter1">第一章 计算机与程序运行</a>
  <p>
    1.了解计算机组成及基本工作原理<br/>
    2.程序与程序设计语言<br/>
    ……
  </p>
  <a name="chapter2">第二章 基本概念</a>
  <p>
    1.数据存储、数据类型、变量、常量<br/>
    2.运算符和表达式<br/>
    ……
  </p>
</body>
```



# 列表

- 列表用于按逻辑方式对数据分组
  - <ul>标签用于创建无序列表（Unordered List）
  - <ol>标签用于创建有序列表（Ordered List）

```
…
<body>
  <h1>星期中的每一天</h1>
  <ul>
    <li>星期日</li>
    <li>星期一</li>
    <li>星期二</li>
    <li>星期三</li>
    <li>星期四</li>
    <li>星期五</li>
    <li>星期六</li>
  </ul>
</body>
…
```

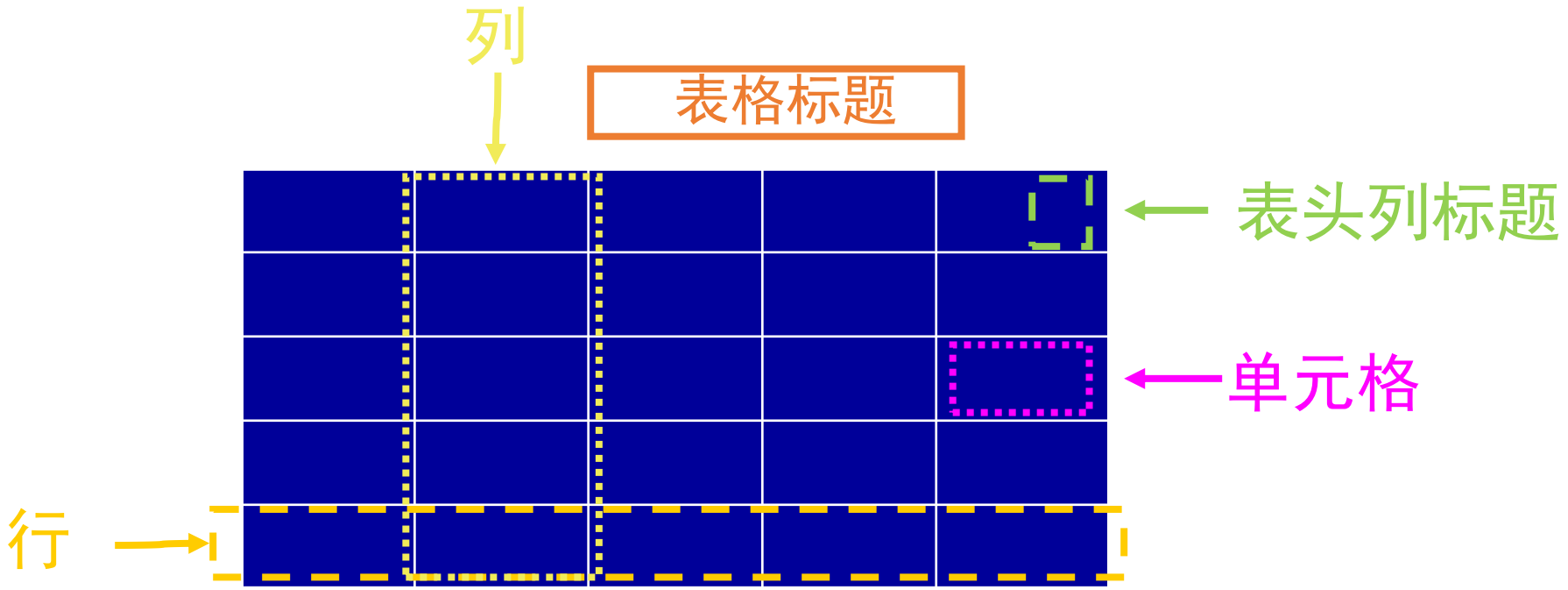
无序列表

```
…
<body>
  <h1>星期中的每一天</h1>
  <ol>
    <li>星期日</li>
    <li>星期一</li>
    <li>星期二</li>
    <li>星期三</li>
    <li>星期四</li>
    <li>星期五</li>
    <li>星期六</li>
  </ol>
</body>
…
```

有序列表


# 表格

- 表格由 <table> 标签来定义
  - 每个表格均有若干行，由 <tr> 标签定义（Table Row）
  - 每行分割为若干单元格，由 <td> 标签定义（Table Data Cell）
  - 表头可以设置列标题，由<th>标签定义（Table Header Cell）
  - 整个表格的标题，由<caption>标签定义



# 表格

学员档案信息

| 姓名 | 性别 | 分数 |
| --- | --- | --- |
| Robert | M | 80 |
| Mary | F | 18 |

```
<html>
  <head>
    <title>使用表格</title>
  </head>
  <body>
    <table border="2">
      <caption align="top">学员档案信息</caption>
      <tr>
        <th>姓名</th> <th>性别</th> <th>分数</th>
      </tr>
      <tr>
        <td>Robert</td> <td>M</td> <td>80</td>
      </tr>
      <tr>
        <td align="left">Mary</td>
        <td align="center">F</td>
        <td align="right">18</td>
      </tr>
    </table>
  </body>
</html>
```

<table>标签代表表格的开始，border=2表示边框尺寸为2

<caption>表示表格标题

<tr>表示行，这是表格的第一行，有三列数据，<th>代表标题单元格

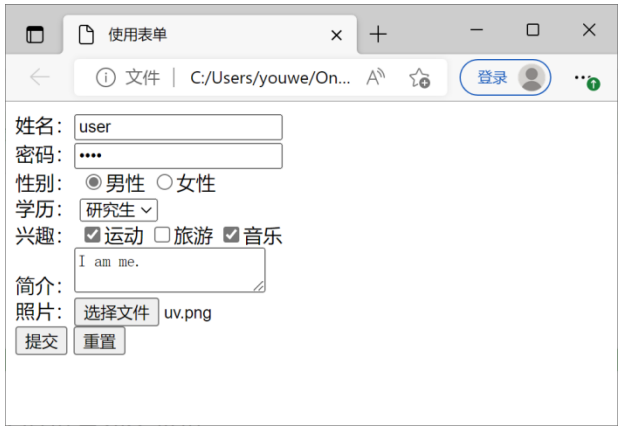
<td>代表数据单元格

不同的对齐方式

# 表单

- 表单由<form>标签定义，用于收集用户的输入信息

```
<form action="submit.php" method="POST" enctype="multipart/form-data">
  姓名：<input type="text" name="name"/> <br/>
  密码：<input type="password" name="password"/> <br/>
  性别：
      <input type="radio" name="sex" value="male" checked/>男性
      <input type="radio" name="sex" value="female"/>女性
      <br/>
  学历：
      <select name="education">
        <option value="undergraduate"/>本科</option>
        <option value="graduate" selected/>研究生</option>
      </select>
      <br/>
  兴趣：
      <input type="checkbox" name="hobby" value="sports"/>运动
      <input type="checkbox" name="hobby" value="travel"/>旅游
      <input type="checkbox" name="hobby" value="music"/>音乐
      <br/>
  简介：<textarea name="intro" placeholder="不超过100字"></textarea> <br/>
  照片：<input type="file" name="photo" id="photo"/> <br/>

  <input type="submit"/>
  <input type="reset"/>
</form>
```

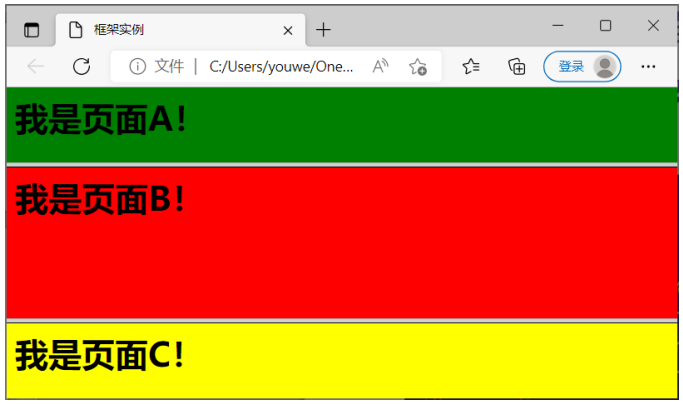



# 框架

- 将浏览器的页面显示区域进行划分，分成多个独立可控制区域，这些区域称为框架，每个框架中都可以打开网页

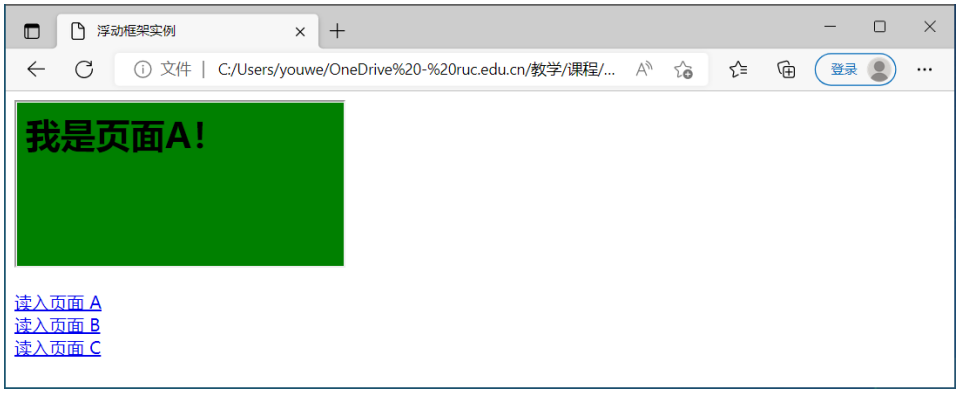
## 普通框架（框架集）

```
<html>
  <head><title>框架实例</title></head>
  <frameset rows="25%,50%,25%">
    <frame src="frame_a.htm"/>
    <frame src="frame_b.htm"/>
    <frame src="frame_c.htm"/>
  </frameset>
</html>
```




## 浮动框架（窗口嵌套）

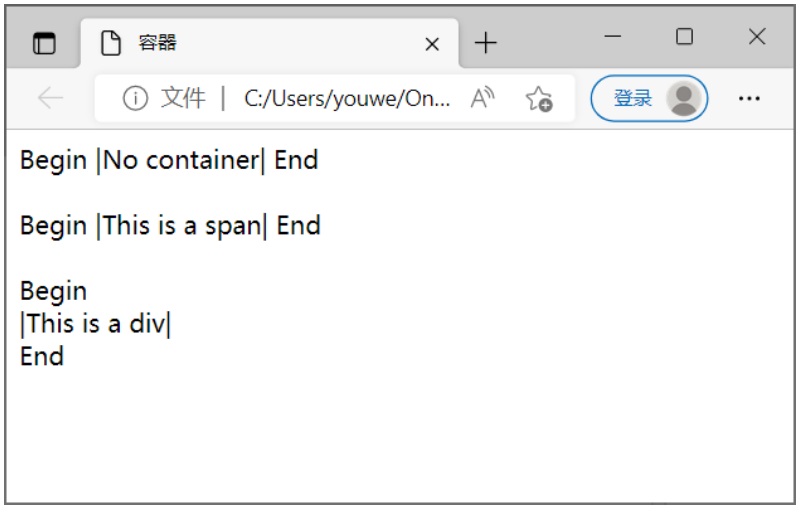
```
<html>
  <head><title>浮动框架实例</title></head>
  <body>
    <iframe src="A.html" name="window"></iframe><br/>
    <a href="A.html" target="window">读入页面 A</a><br/>
    <a href="B.html" target="window">读入页面 B</a><br/>
    <a href="C.html" target="window">读入页面 C</a><br/>
  </body>
</html>
```



# 容器

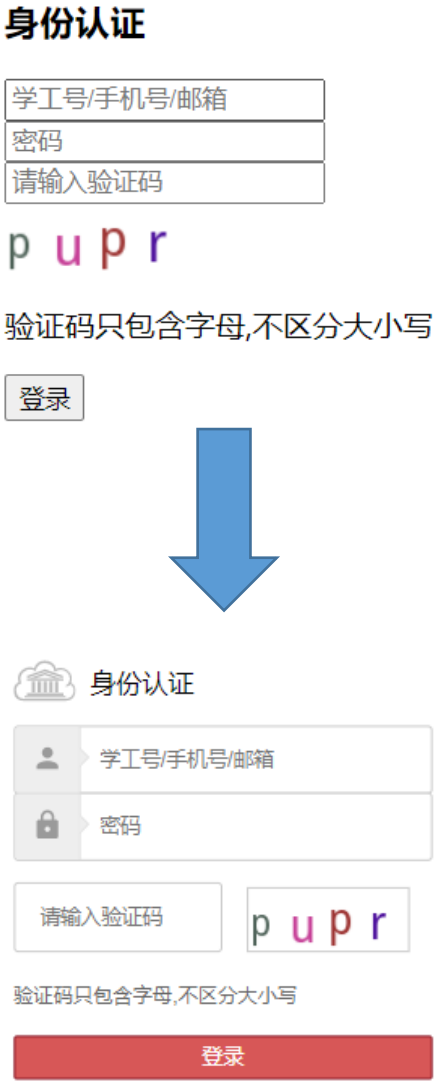
- 容器本身无意义，其存在是配合CSS为其内部内容定义显示规则
  - 是行内元素，在它的前后不会换行
  - 是块级元素，它包含的元素会自动换行

```
<html>
<head>
  <title>容器</title>
</head>
<body>
  Begin |No container| End
  <br/><br/>


  Begin <span class="myclass">|This is a span|</span>
  End
  <br/><br/>


  Begin <div id="myid"> |This is a div| </div> End
  <br/><br/>
</body>
</html>
```



# 实例：微人大认证页面HTML代码

```
<html>
  <head>
    <meta charset="utf-8"/>
    <title>账户登录</title>
    <link rel="stylesheet" href="style.css"/>
    <script type="text/javascript" src="script.js"></script>
  </head>
  <body onload="body_onload_handler()">
    <div class="card-container">
      <form id="login-form" onsubmit="return false">
        <div id="login-title"><h3>身份认证</h3></div>
        <div class="input-group" id="username">
          <input type="text" name="username" placeholder="学工号/手机号/邮箱" required>
        </div>
        <div class="input-group" id="password">
          <input type="password" name="password" placeholder="密码" required>
        </div>
        <div class="input-group" id="captcha-input">
          <input type="text" name="code" placeholder="请输入验证码">
        </div>
        <div class="input-group" id="captcha-img"><img src="img/code_0.png"/></div>
        <p id="captcha-info">验证码只包含字母,不区分大小写</p>
        <div class="alert" id="login-alert"></div>
        <div class="form-group">
          <button id="login-submit" type="submit"><span>登录</span></button>
        </div>
      </form>
    </div>
  </body>
</html>
```



# 2.2 CSS

- CSS是Cascading Style Sheets（层叠样式表）的缩写，是一种样式语言，用于修饰HTML元素，从而告诉浏览器如何将这些元素精美地显示在网页中

```
<html>
  <head>
    <style>
    body { background-color: lightblue; }
    #myid { color: white; text-align: center; }
    .myclass { font-family: verdana;
               font-size: 20px; }
    </style>
  </head>
  <body>
    <h1 id="myid">我的第一个 CSS 实例</h1>
    <p class="myclass">这是一个段落。</p>
  </body>
</html>
```


学习更多：https://www.w3school.com.cn/css/index.asp

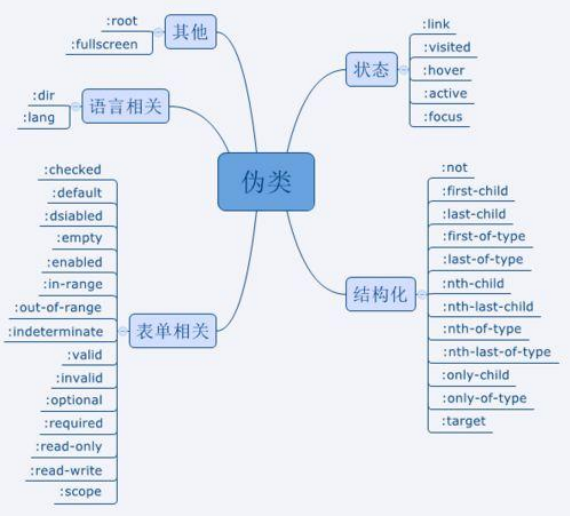
# 2.2.1 选择器

- 指示浏览器该条规则应用于HTML文档中的哪一个（些）标签
  - class选择器：为同一个class的多个元素设置样式
  - id选择器：为指定id的唯一元素设置样式
  - 元素选择器：为同一标签的全部元素设置样式
  - 伪类和伪元素：选择处于特殊状态的元素或者元素中特别的内容

| | 选择器 | 例子 | 例子描述 |
|---|---|---|---|
| | * | * | 选取所有元素。 |
| 属性选择器 | .class | .intro | 选取所有 class="intro" 的元素。 |
| 属性选择器 | #id | #firstname | 选取 id="firstname" 的那个元素。 |
| 元素选择器 | element | p | 选取所有 <p> 元素。 |
| 元素选择器 | element1,element2,.. | div, p | 选取所有  元素和所有 <p> 元素。 |
| 上下文选择器 | element1 element2 | div p | 选取  元素的所有后代 <p> 元素。 |
| 上下文选择器 | element1>element2 | div>p | 选取  元素子代的所有 <p> 元素。 |

# 伪类和伪元素：实现基于文档树外信息的格式化

- 伪类：为处于某个状态的已有元素添加对应的样式，这个状态是根据用户行为而动态改变的
  - 状态伪类：基于元素当前状态进行选择的
  - 结构性伪类：通过文档结构的互相关系来匹配元素




:link 应用于未被访问过的链接；

:hover 应用于鼠标悬停到的元素；

:active 应用于被激活的元素；

:visited 应用于被访问过的链接，与:link互斥。

:focus 应用于拥有键盘输入焦点的元素。

状态伪类

结构性伪类

1. :first-child 选择某个元素的第一个子元素；
2. :last-child 选择某个元素的最后一个子元素；
3. :nth-child(n) 匹配属于其**父元素**的第 n个子元素，**不论元素的类型**；
4. :nth-last-child() 从这个元素的最后一个子元素开始算,选匹配属于其父元素的第 n个子元素，不论元素的类型；
5. :nth-of-type() 规定属于其父元素的第n个 指定 元素；
6. :nth-last-of-type() 从元素的最后一个开始计算,规定属于其父元素的指定 元素；
7. :first-of-type 选择一个上级元素下的第一个同类子元素；
8. :last-of-type 选择一个上级元素的最后一个同类子元素；
9. :only-child 选择它的父元素的唯一一个子元素；
10. :only-of-type 选择一个元素是它的上级元素的唯一一个相同类型的子元素；
11. :checked匹配被选中的input元素，这个input元素包括radio和checkbox。
12. :empty 选择的元素里面没有任何内容。
13. :disabled匹配禁用的表单元素。
14. :enabled匹配没有设置disabled属性的表单元素。
15. :valid匹配条件验证正确的表单元素。
16. :in-range选择具有指定范围内的值的 <input> 元素。
17. :optional选择不带 "required" 属性的 <input> 元素。
18. :focus选择获得焦点的 <input> 元素。


```
<html>
  <head>
    <style>
      body { background-color: black; }
      .nav-class {
        margin: 15px 20px; padding: 0 15px; font-size: 14px;
        display: -webkit-box; display: -ms-flexbox; display: flex;
        background: hsla(0,0%,100%,.9); border-radius: 6px;
      }
      .nav-item {
        padding: 8px 0; color: #000; text-align: center;
        -webkit-box-flex: 1; -ms-flex: 1; flex: 1;
      }
      .nav-class :first-child       { border-top: 2px solid #cf4a65; }
      .nav-class :first-child :hover { background: #cf4a65; color: #f2f2f2; }
      .nav-class :nth-child(2)      { border-top: 2px solid #e68654; }
      .nav-class :nth-child(2) :hover { background: #e68654; color: #f2f2f2; }
      .nav-class :nth-child(3)      { border-top: 2px solid #2db0d2; }
      .nav-class :nth-child(3) :hover { background: #2db0d2; color: #f2f2f2; }
      .nav-class :nth-last-child(2)      { border-top: 2px solid #dcbf55; }
      .nav-class :nth-last-child(2) :hover { background: #dcbf55; color: #f2f2f2; }
      .nav-class :last-child        { border-top: 2px solid #64b36a; }
      .nav-class :last-child :hover { background: #64b36a; color: #f2f2f2; }
    </style>
  </head>
  <body>
    <div class="nav-class">
      <div class="nav-item class-border-0"><span>全部</span></div>
      <div class="nav-item class-border-1"><span>投稿</span></div>
      <div class="nav-item class-border-2"><span>表白</span></div>
      <div class="nav-item class-border-3"><span>吐槽</span></div>
      <div class="nav-item class-border-4"><span>闲置</span></div>
    </div>
  </body>
</html>
```

注意：伪类与前置选择器之间不能有空格，此处为了排版增加空格

状态伪类:hover应用于鼠标悬停的元素

结构性伪类:first-child, :nth-child, :nth-last-child, :last-child应用于.nav-item类的第几个元素

# 伪类和伪元素

| 选择器 | 例子 | 例子描述 |
| --- | --- | --- |
| ::after | p::after | 在每个 <p> 元素之后插入内容。 |
| ::before | p::before | 在每个 <p> 元素之前插入内容。 |
| ::first-letter | p::first-letter | 选择每个 <p> 元素的首字母。 |
| ::first-line | p::first-line | 选择每个 <p> 元素的首行。 |
| ::selection | p::selection | 选择用户选择的元素部分。 |

- 伪元素：创建一些不在文档树中的元素，并为其添加样式

```
<html>
  <head>
    <style>
      h1::before {
        content: url("http://www.youwei.site/ruc.png");
      }
      h2::after {
        content: url("http://www.youwei.site/vruc.png ");
      }
      h3::selection {
        color: red; background: yellow;
      }
    </style>
  </head>

  <body>
    <h1>这个标题前面会插入中国人民大学的logo</h1>
    <br/>

    <h2>这个标题后面会插入微人大的logo</h2>
    <br/>

    <h3>这个标题选中的文本会被高亮显示</h3>
    <br/>
  </body>
</html>
```

伪元素::before, ::after, ::selection


# 2.2.2 样式定义与冲突处理

- 样式定义方式及优先级（优先级越高，对最终样式影响越大）

```
/*所有span元素的默认字体颜色样式*/

span { color: black; }
```

浏览器默认样式（优先级=1）

```
<head>
  <link rel="stylesheet" href="mystyle.css">
</head>
```

外部样式mystyle.css（优先级=2）

```
<head>
  span { color: limegreen; }
</head>
```

内部样式（优先级=3）

```
<body>
  <span style="color:limegreen">周冲</span>
</head>
```

内联样式（优先级=4）

| 样式名称 | 定义 |
| --- | --- |
| 浏览器默认样式 | 如果显式没有给出样式声明，HTML元素本身也有样式的默认设置 |
| 外部样式 | 写在独立的css文件中，例如mystyle.css |
| 内部样式 | 在html文档的<head>结构内定义 |
| 内联样式 | 在标签内部直接定义 |

# 2.2.2 样式定义与冲突处理

- 同一级别下，重复定义样式：谁后定义听谁的

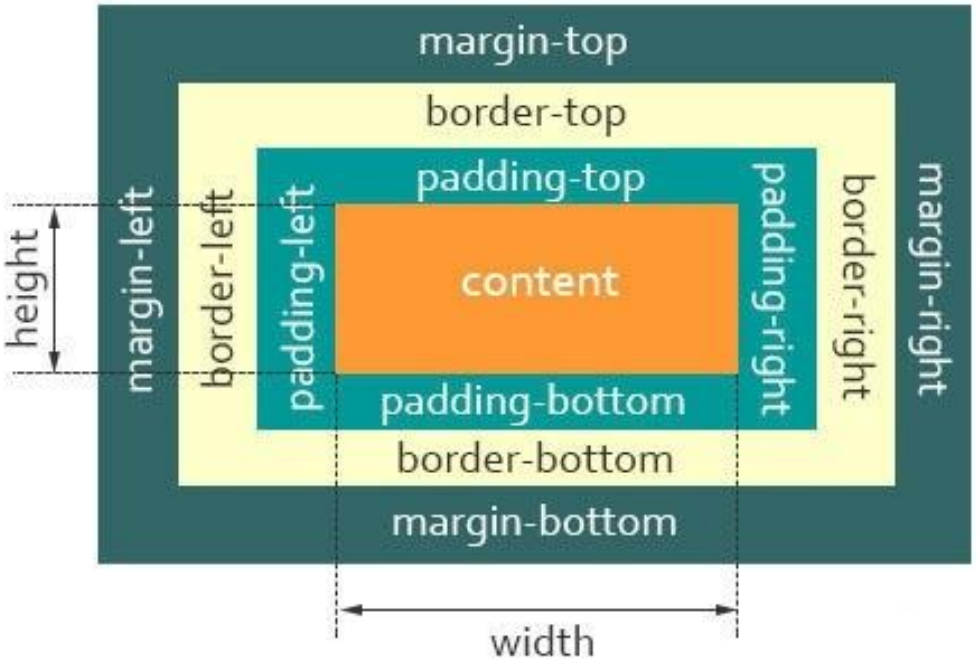
```
<head>
  <style>
    h2 { color: yellow; }
    h2 { color: blue; }
  </style>
</head>
<body> <h2>最终是蓝色</h2> </body>
<!-- 类比：相同变量的多次赋值，以最后一次赋值为准 -->
```

- 不同级别下，重复定义样式：谁优先级高听谁的

```
<head>
  <link rel="stylesheet" href="my.css">    <!--第一种：外部样式 -->
  <style> h2 { color: yellow; } </style>   <!--第二种：内部样式 -->
</head>
<body>
  <h2 style="color:green;"> 最终是绿色</h2> <!--第三种：内联样式 -->
</body>
<!-- 类比：局部变量与全局变量重名；子类对父类方法的重写 -->
```

# 2.2.3 盒子模型

- 每一个可以包裹内容的元素都可以定义盒子模型
  - 该模型分为内外两层盒子：外层盒子边框不可见，内层盒子边框可见
  - margin：定义两个盒子之间的距离（外边距）
  - padding：定义盒子内边框与内容之间的距离（内边距）

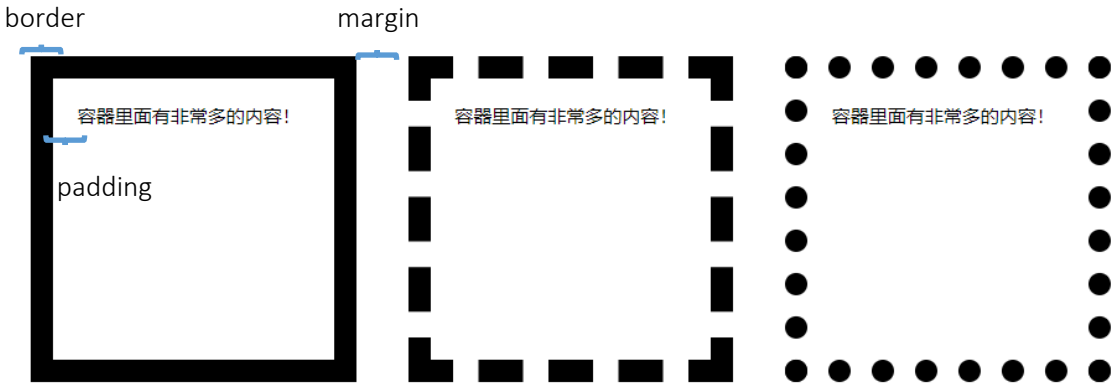


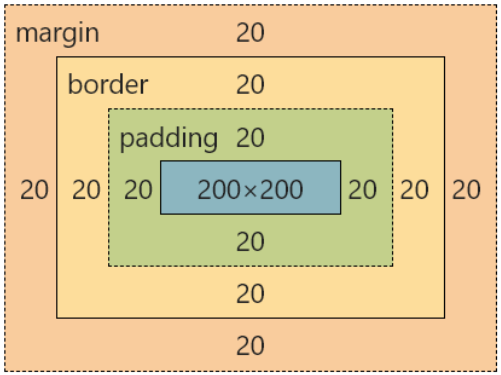

盒子宽度 = margin_left + margin_right
+ padding_left + padding_right
+ border_left + border_right
+ width

盒子高度 = margin_top + margin_bottom
+ padding_top + padding_bottom
+ border_top + border_bottom
+ height

# 2.2.3 盒子模型

```
<html>
  <head>
    <title>盒子模型</title>
    <style>
      .demo { display: inline-block; width:200px; height:200px;
              border:20px; margin:20px; padding: 20px; }
      #demo1 { border-style: solid; }
      #demo2 { border-style: dashed; }
      #demo3 { border-style: dotted; }
    </style>
  </head>
  <body>
    <div class="demo" id="demo1">容器里面的内容</div>
    <div class="demo" id="demo2">容器里面的内容</div>
    <div class="demo" id="demo3">容器里面的内容</div>
  </body>
</html>
```

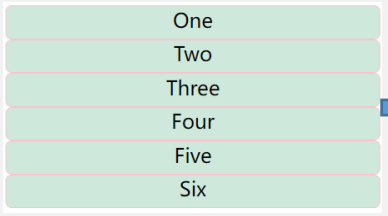


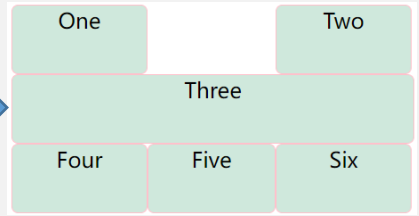
# 2.2.4 布局

```html
<html>
  <head>
    <style>
      .wrapper { width: 300px; }
      .wrapper > div {
        border-radius: 5px; border: 1px solid pink;
        background-color: rgb(207,232,220);
        padding: 0; margin: 0; text-align: center;
      }
    </style>
  </head>
  <body>
    <div class="wrapper">
      <div class="box1">One</div>
      <div class="box2">Two</div>
      <div class="box3">Three</div>
      <div class="box4">Four</div>
      <div class="box5">Five</div>
      <div class="box6">Six</div>
    </div>
  </body>
</html>
```

<!-- 默认布局 -->




<!-- 目标布局 -->




```css
<!-- Float布局 -->
.wrapper > div { height: 50px; }
.box1 { float: left; width: 100px; }
.box2 { float: right; width: 100px; }
.box3 { clear: both; width: 300px; }
.box4 { float: left; width: 100px; }
.box5 { float: left; width: calc(100% - 206px); }
.box6 { float: left; width: 100px; }
```

```css
<!-- Position布局: -->
.wrapper {position: relative;}
.wrapper > div { height: 50px; }
.box1 { position: absolute; left: 0px; top: 0px; width: 100px; }
.box2 { position: absolute; right: 0px; top: 0px; width: 100px; }
.box3 { position: absolute; left: 0px; top: 50px; width: 100%; }
.box4 { position: absolute; left: 0px; top: 100px; width: 100px; }
.box5 { position: absolute; left: 100px; top: 100px; width: 100px; }
.box6 { position: absolute; right: 0px; top: 100px; width: 100px; }
```

```css
<!- Flex布局: -->
.wrapper {
  width: 300px; display:flex;
  flex-wrap: wrap;
  flex-direction: row;
  justify-content: space-between;
}
.wrapper > div { height: 50px; }
.box1 { width: 100px; }
.box2 { width: 100px; }
.box3 { width: 100%; }
.box4 { width: 100px; }
.box5 { width: calc(100% - 206px); }
.box6 { width: 100px; }
```

```css
<!-- Grid布局: -->
.box2 { grid-column: 3; }
.box3 { grid-column: 1 / span 3; }
.wrapper {
  display: grid;
  grid-template-rows: 50px 50px 50px;
  grid-template-columns: 100px 100px 100px;
}
```

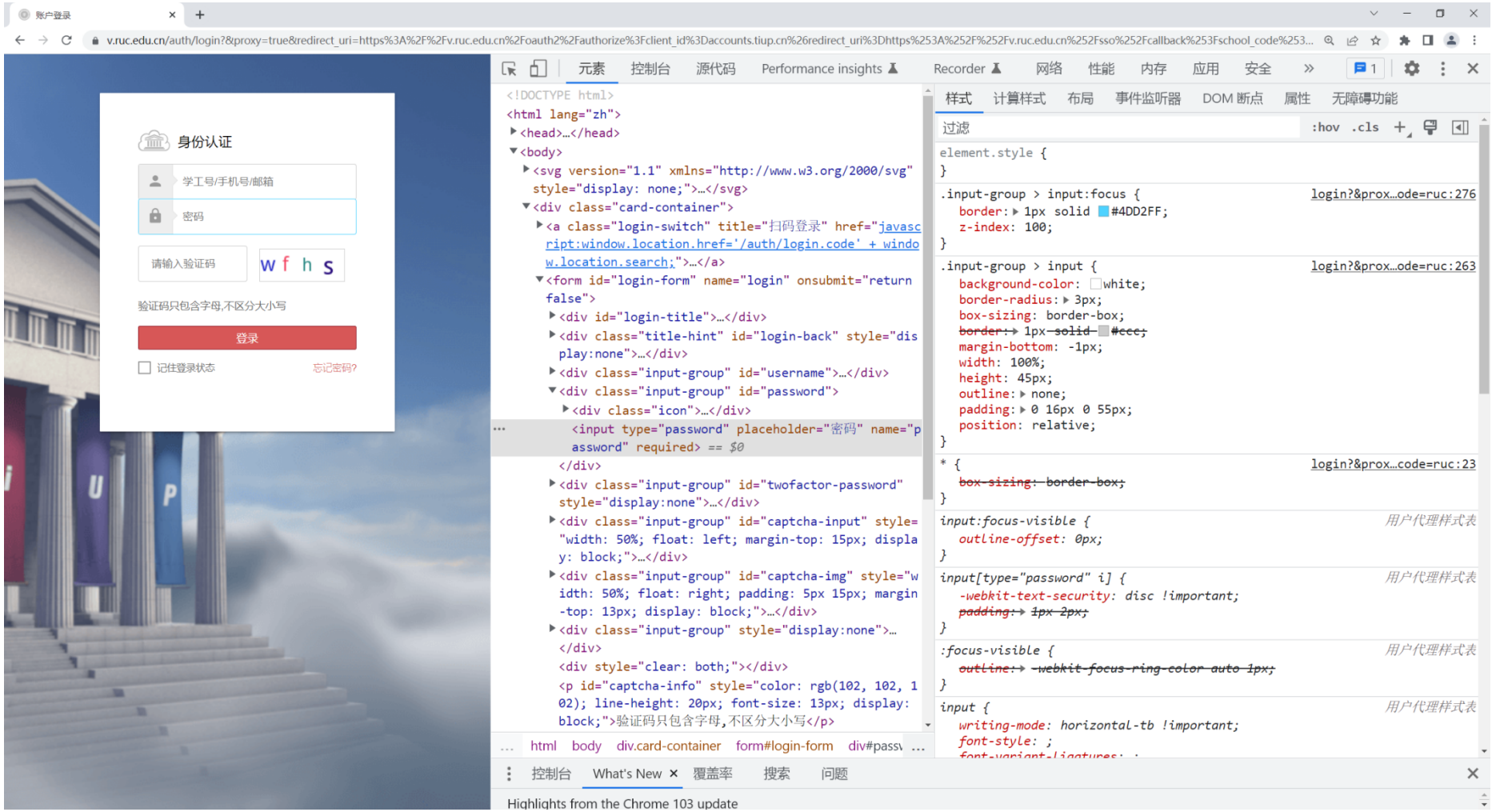
# 2.2.5 常用的样式


文字相关的样式
文本颜色
color:#333333;十六进制
color:red;关键字
color:rgb(255,255,0)；
文字大小
font-size:16px;
浏览器默认的字号16px 最小字号12px
文字背景色
background: red;
文字加粗
font-weight:bold;
文字字体
font-family: '微软雅黑'
文字下划线
text-decoration:none;去掉下划线
text-decoration:underline;加下划线
边框线
border: 1px solid #333333; (常用)
solid实线
dashed虚线
dottded点状
默认加粗文字样式去掉,如h1
font-weight:normal;
文字水平居中
text-align:center;作用是让块元素里的行内元素居中
文字垂直居中
height = line-height时
段落首行缩进实际（应用场景文章详情页面）
text-indent:2em;
首行缩进了2个文字的宽度
省略号
overflow:hidden;----溢出隐藏
overflow: auto;----自适应（内容过多会自动出滚动条）
text-overflow: ellipsis;---省略号
自动出省略号：四点
overflow:hidden;
text-overflow: ellipsis;
容器的宽度width；
white-space: nowrap;不换行
无序里li标签的点去掉
list-style:none;
背景的相关样式
背景图片的写法
background:url(*img/a.jpg*);
背景不平铺
background-repeat: no-repeat;
背景图片的位置 x,y
background-position: x,y;
x：水平方向 单位可以是px、% —— left居左/right居右/center居中
y：垂直方向 单位可以是px、% —— top居上/bottom居下/center居中
综合
background: url(*img/a.jpg*) no-repeat left center;
边框的相关样式
上边框
border-top:1px solid #333;
下边框
border-bottom:1px solid #333;
左边框
border-left:1px solid #333;
右边框
border-right:1px solid #333;
图片如何实现居中
法一：因为img行内块
将img转成块 display: block;
一定要给图片宽度width: npx;
居中margin: 0 auto; —— （margin: auto auto;是错误的，auto不能上下居中）
法二：img外层套块级标签
将text-align: center;这个属性给在套img的块级标签上
img标签会有留白如何解决？
将img行内块标签转化为块就可以 ，用disply:block;
但是注意转成块以后的居中就不能用行居中的方式text-align:center;了，得用块居中的方式magin:auto;
行标签和块标签的转换化
特点
行标签
宽度和高度设置无效
span a
块标签
宽度和高度设置有效
div、p、h1-h6、ul、li、dl、dt、dd
行转块
display:block;
块转行
display:inline;
块元素转成行内块
display:inline-block;
本身存在bug（默认有空隙）
解决办法：把用了的标签与上一标签写在编译器同一行即可
清空默认
* {padding: 0;margin: 0;}
浏览器默认的外边距：8px
ul, li { list-style-type: none; }
a{text-decoration: none;}

# 2.2.5 常用的样式

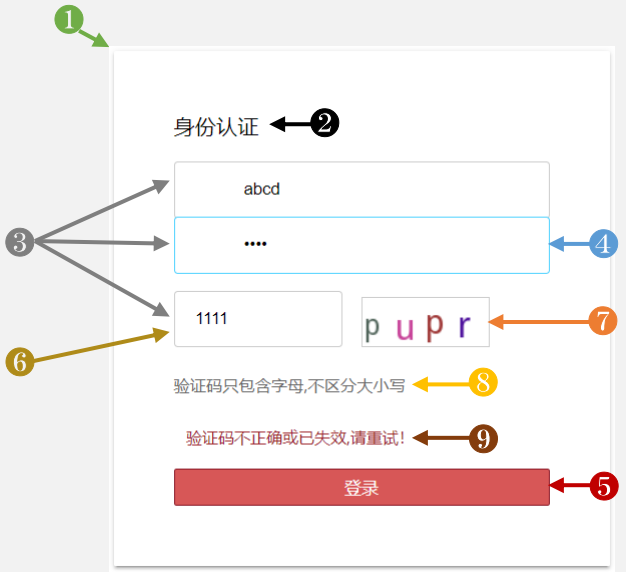

盒模型
padding内边距
应用场景：嵌套的
margin外边距
应用场景：两个独立的块与块之间的距离
值的意义
一个值
padding-top:20px;上边距
padding-bottom:20px;下边距
padding-left: 20px;左内边距
padding-right:20px;右内边距
两个值
margin: 10px 20px;
第一个值指的是上下
第二个值指的是左右
三个值
margin: 5px 8px 15px;
第一个值指的是上
第二个值指的是左右
第三个值指的是下
四个值
margin: 5px 10px 8px 20px;
第一个值是上
第二个值是右
第三个值是下
第四个值是左
padding和margin的区别
外边距能够调盒子的位置
内边距能够改变盒子的大小
margin自身存在外边距合并问题
取最大值进行合并
如何解决
一般在开发页面中，给其中一个盒子下边距，那么给其他盒子也一直下边距
注意事项
块元素-------支持上下左右 padding margin
行标签-------只支持左右padding margin上下padding margin不支持（跟转成块无关）
想让上下内边距生效----------把行标签转成块标签 display：block；
防止父代随子代中的margin塌陷
给父代加一个透明border
border: 1px solid transparent;
浮动
应用场景
横向导航
左浮动
float:left;
右浮动
float:right;
不浮动
float:none;
清除浮动
理由
只要元素浮动，那么就会出现问题（外层容器无法包裹住里层元素）
方法
.clearfix:after{display: block;clear: both;content:''; } 给在浮动元素父级
圆形
先做正方形
（宽高一样）
加border-radius:50%;宽度和高度一致
如何让一个盒子在当前的浏览器中：水平居中、垂直居中
绝对定位加负margin就可以实现： 负的盒子宽度的一半、负的盒子高度的一半
position: absolute;left:50%;top:50%;
使左上角的点定到浏览器正中
margin-left: -100px;margin-top: -100px;
定位
与浮动一样，都脱离了文档流，即加了定位后，转成行内块
绝对定位
position:absolute;
给了绝对定位，一定要给父代相对定位
（因为有时我们只给了绝对没有相对，在ie浏览器上会出现问题）
如果没有给相对，那么就会逐层向上查找，直到找到body标签为止
不占原来位置，盒子有父级时使用，定位指相对于body
相对定位
position:relative;
占原来位置，定位指相对于自己原来位置
固定定位
position:fixed;
相对于页面，但不滚动
只要有position就用top/bottom/right/left
top:0;距离上方bottom:0;距离下方left:0;距离左方right:0;距离右方
z-index：数字;
值越大层级越高
但是需要注意的是一定要与position连用
子绝父相
父亲是相对position: relative;
孩子是绝对position: absolute;

# 实例：使用CSS美化微人大认证页面



# 实例：使用CSS美化微人大认证页面

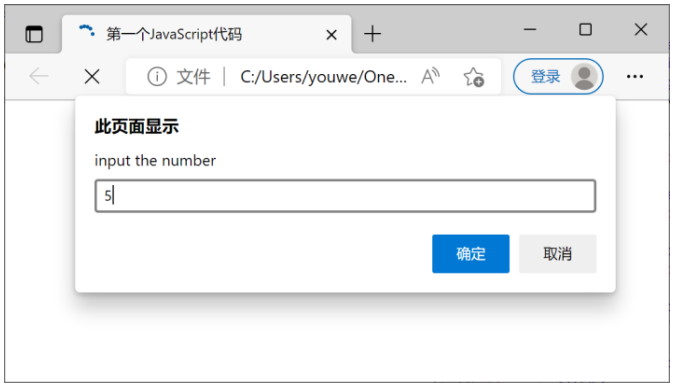
```css
/* style.css */

.card-container {  ❶
  max-width: 300px; margin: 8vw auto; background: #fff; width: 100%; padding: 2rem 3rem; border-radius: 1px;
  box-shadow: 0 2px 2px 0 rgb(0 0 0 / 14%), 0 3px 1px -2px rgb(0 0 0 / 20%), 0 1px 5px 0 rgb(0 0 0 / 12%);
}

h3 { font-size: 17px; font-weight: 400; }  ❷

.input-group > input {  ❸
  background-color: white; border-radius: 3px; box-sizing: border-box;
  border: 1px solid #ccc; margin-bottom: -1px; width: 100%; height: 45px;
  outline: none; padding: 0 16px 0 55px; position: relative;
}

.input-group > input:focus { border: 1px solid #4DD2FF; z-index: 100; }  ❹

.button {
  position: relative; font-size: 0.85rem; background: #d75757;
  border: 1px solid #891524; padding: .3rem 2rem;
  border-radius: 2px; width: 100%; cursor: pointer; overflow: hidden;
}  ❺
.button span { color: #fff; position: relative; z-index: 1;  }

#captcha-input { width: 45%; float: left; margin-top: 15px; display: block; }  ❻
#captcha-input input { padding-left: 16px; }

#captcha-img { width: 45%; float: right; padding: 5px 15px; margin-top: 15px; display: block; border: 1px solid #ccc;}  ❼
#captcha-img img { padding-left: 16px; border: 1px solid #ccc; }

#captcha-info { clear: both; color: rgb(102, 102, 102); line-height: 20px; font-size: 13px; display: block; margin-top: 80px; }  ❽

.alert { position: relative; margin: 0.5em,0,-1em,0; padding: 10px; border-radius: 2px; font-size: .8rem; color: #9c232a; }  ❾
```

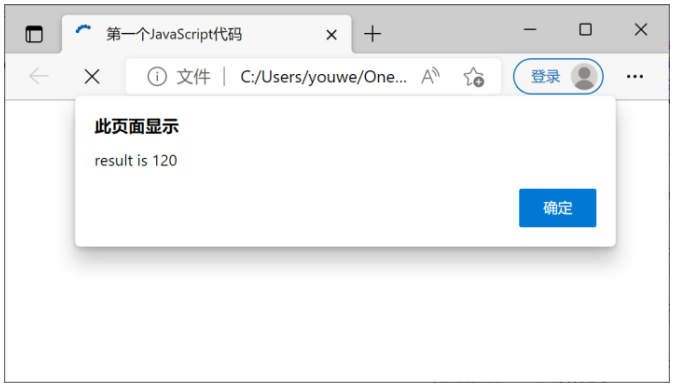


# 2.3 JavaScript

- JavaScript是一门基于对象的、弱类型、解释性语言，其运行于客户端浏览器，主要用于与页面元素进行动态交互

```
<html>
  <head>
    <title>第一个JavaScript代码</title>
  </head>
  <body>
    <script type="text/javascript">
      function factorial(n) {          //递归函数计算阶乘
        if (n == 1) return n;
        else return n * factorial(n-1);
      }
      n = prompt("input the number"); //类似于scanf
      var res = factorial(n);          //函数调用
      res = "result is " + res;        //变量类型自动转变
      alert(res);                      //类似于printf
      console.log(res);                //在控制台显示(F12)
    </script>
  </body>
</html>
```




学习更多：https://www.w3school.com.cn/js/index.asp

# 2.3.1 在网页中引入JavaScript代码

- 在页面中直接嵌入JavaScript

```
<html>
  <head>
    <script type="text/javascript"> alert("JavaScript in head"); </script>
  </head>
  <body>
    <script type="text/javascript"> alert("JavaScript in body"); </script>
  </body>
</html>
```

- 链接外部JavaScript文件

```
<html>
  <head>
    <script type="text/javascript" src="javascript_head.js"></script>
  </head>
  <body>
    <script type="text/javascript" src="javascript_head.js"></script>
  </body>
</html>
```

```
/* javascript_head.js */
alert("JavaScript in head");
```

```
/* javascript_body.js */
alert("JavaScript in body");
```

注意：在<head>...</head>中的脚本运行时机先于在<body>...</body>中的脚本，此时页面元素尚未完成装载。

# 2.3.1 在网页中引入JavaScript代码

- 作为标签的属性值使用
  - 通过“javascript:”调用JavaScript函数

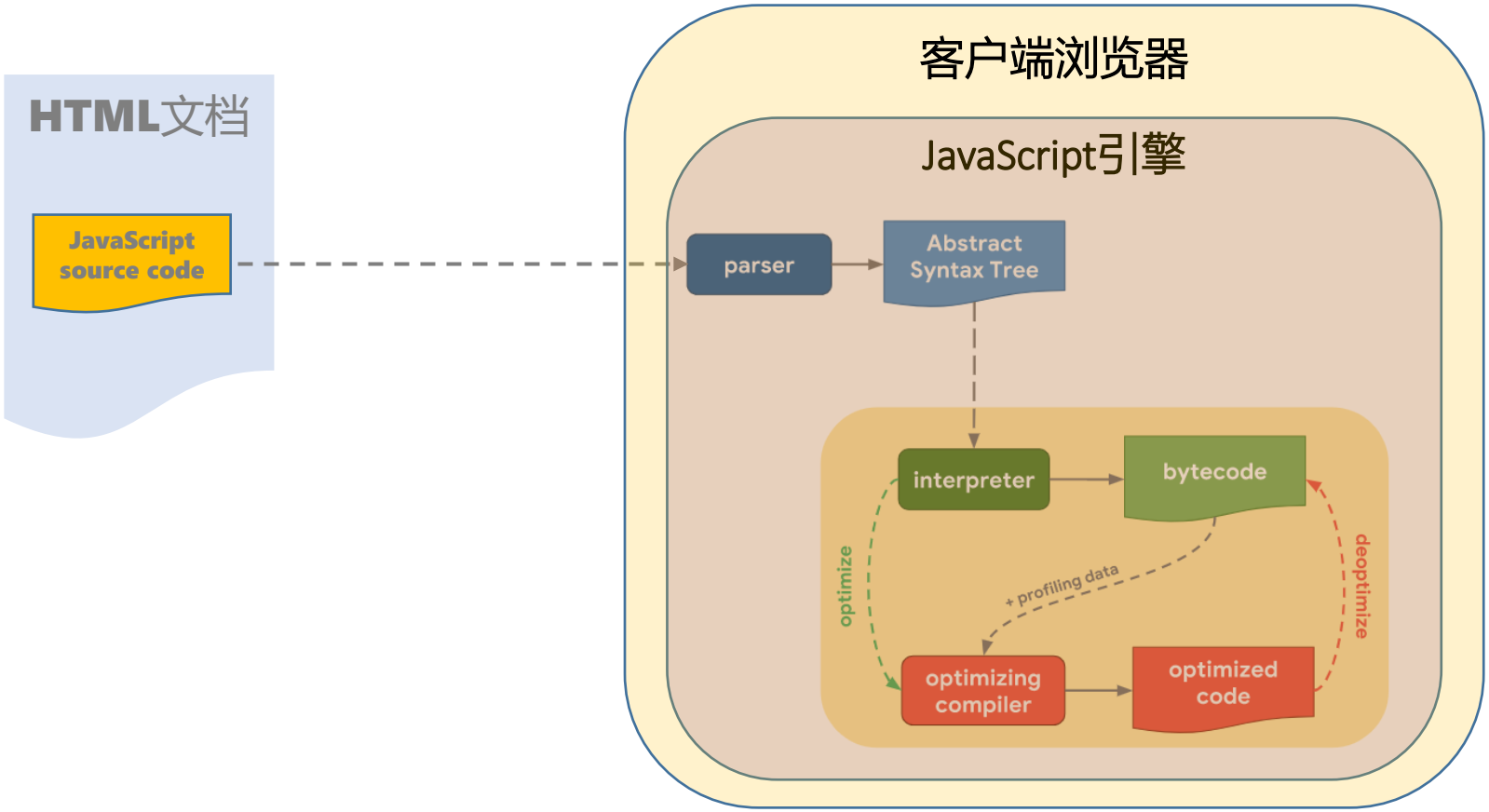
```
<html>
  <body>
    <a href="javascript:alert('你好JavaScript')">测试</a>
  </body>
</html>
```

  - 在事件中调用JavaScript函数

```
<html>
  <body>
    <input type="button" value="测试" onclick="alert('你好JavaScript')" />
  </body>
</html>
```

# 2.3.2 JavaScript代码执行流程

- 浏览器将嵌入HTML文档的JavaScript代码交由JavaScript引擎
- 解析->解释执行（非热点代码）->JIT编译执行（热点代码）



# 2.3.3 变量

- 定义方式
  - 使用var、let关键字，后跟变量名进行声明
  - 不使用var、let关键字声明，而是直接赋值
- 作用域：变量起作用的地方（空间）
  - 全局作用域：最外层函数定义的变量
    未声明而直接赋值的变量
  - 局部作用域：函数作用域（函数内部）
    块级作用域（代码块内部）
- 生命周期：变量生存的时段（时间）
  - 全局变量：从定义位置开始，一直到整个代码结束才会销毁
  - 局部变量：在函数中定义，在函数中使用，函数结束时自动销毁

```
var global_a = 1; //最外层函数定义的变量

function fun(x) {
  global_b = 2; //未声明而直接赋值的变量
  var local_a = 3; //函数作用域

  if (x > 0) {
    //块作用域，使用var声明提升至函数作用域
    var local_b = 4;
    //块作用域，使用let声明不发生变量提升
    let local_c = 5;
  }

  alert(local_a); //3
  alert(local_b); //4

  //ReferenceError: local_c is not defined
  try { alert(local_c); }
  catch(e) { alert(e); }
}

fun(1);
alert(global_a); //1
alert(global_b); //2
```

# 2.3.4 数据类型

- JavaScript是弱类型脚本语言，无需声明变量类型，运行时根据需要，自动确定变量的数据类型并进行数据类型转换
  - 基本类型存储在栈内存，把一个基本类型的变量赋值给另一个变量实际上进行的是值拷贝
  - 引用类型指针（地址）存储在栈内存，这个指针指向的对象存在堆内存，如果进行拷贝，拷贝的是栈内存中存储的指针，而不是对象真正的值

| 大类 | 数据类型 | 说明 |
| --- | --- | --- |
| 基本类型 | 数值类型（Number） | 包含整数和浮点数，特殊值：NaN, ±Infinity |
| | 布尔类型（Boolean） | 只有true和false两个值 【两个值小写】 |
| | 字符串类型（String） | 表示一个字符序列，必须使用单引号（''）或双引号（""）括起来 |
| | Undefined类型（Undefined） | 用来确定一个已经创建但还没有赋初值的变量，只有一个值为：undefined 【小写】 |
| | Null类型（Null） | 表名某个变量的值为空，只有一个值为：null【小写】 |
| 引用类型 | Object类型（Object） | 对象由一些列属性（变量）和方法（函数）构成的集合，访问它们时用. (英文点号) |
| | 数组（Array） | 一系列的变量组成，数量元素的类型无需一致 |
| | 函数（Function） | 包含一段可执行的代码 |

# 2.3.4 数据类型

```
n = 5.5;
alert(typeof(n)); //number

n = 5;
alert(typeof(n)); //number

n = true;
alert(typeof(n)); //boolean

n = "hello javascript";
alert(typeof(n)); //string

var m; //变量定义但未初始化
alert(typeof(m)); //undefined

try {alert(x);} //引用一个未定义的变量
catch(e) {alert(e);} //ReferenceError: x is not defined
```

```
n = null; //特殊值null被认为是一个空的对象引用
alert(typeof(n)); //object
alert(n instanceof Object); //null不是Object类实例

n = new Object(); //实例化一个对象
alert(typeof(n)); //object
alert(n instanceof Object); //Object类实例

n = [1, 2, 3];
alert(typeof(n)); // object
alert(n instanceof Array); //Array类实例
alert(n instanceof Object); //Object类实例

n = function() {};
alert(typeof(n)); // function
alert(n instanceof Function); //Function类实例
alert(n instanceof Object); //Object类实例
```

- 在对变量赋值时会自动改变变量的类型
- 引用未定义的变量，会引发异常
- typeof：判断某个变量/表达式的数据类型
- instancof：判断某个变量是否是指定类的实例，相当于C++的dynamic_cast

# 2.3.5 类型转换

- 类型转换分为：隐式类型转换和显式类型转换
  - 显式类型转换：开发人员在编写代码时，强制指定对值的类型进行转换
  - 隐式类型转换：对不同类型值使用运算符时，值可在类型之间自动转换

```
//ToNumber
str1 = "123", str2 = "123.456", str3 = "12xyz";
num11 = Number(str1), num12 = Number(str2), num13 = Number(str3);              //显式转换
num21 = parseInt(str1), num22 = parseInt(str2), num23 = parseInt(str3);        //显式转换
num31 = parseFloat(str1), num32 = parseFloat(str2), num33 = parseFloat(str3);  //显式转换
num41 = str1 - 3, num42 = str1 - str2, num43 = str1 - str3;                    //隐式转换
alert(num11); /* 123 */          alert(num12); /* 123.456 */      alert(num13); /* NaN */
alert(num21); /* 123 */          alert(num22); /* 123 */          alert(num23); /* 12 */
alert(num31); /* 123 */          alert(num32); /* 123.456 */      alert(num33); /* 12 */
alert(num41); /* 120 */          alert(num42); /* -0.4560... */   alert(num43); /* NaN */

//ToString
boola = true, numb = 123;
stra = String(boola); strb = numb.toString();                                  //显式转换
strc = numb + "456";                                                           //隐式转换
alert(stra); /* true */          alert(strb); /* 123 */          alert(strc); /* 123456 */

//ToBoolean
strx = "123", numy = 0, strz = "";
boolx = Boolean(strx), booly = Boolean(numy);                                  //显式转换
boolz = !!strz;                                                                //隐式转换
alert(boolx); /* true */          alert(booly); /* false */        alert(boolz); /* false */
```

### ToNumber Conversions

| Argument Type | Result |
|---|---|
| Undefined | Return **NaN**. |
| Null | Return **+0**$_{\mathbb{F}}$. |
| Boolean | If *argument* is **true**, return **1**$_{\mathbb{F}}$. If *argument* is **false**, return **+0**$_{\mathbb{F}}$. |
| Number | Return *argument* (no conversion). |
| String | Return ! StringToNumber(*argument*). |
| Object | Apply the following steps:<br>1. Let *primValue* be ? ToPrimitive(*argument*, number).<br>2. Return ? ToNumber(*primValue*). |

### ToString Conversions

| Argument Type | Result |
|---|---|
| Undefined | Return **"undefined"**. |
| Null | Return **"null"**. |
| Boolean | If *argument* is **true**, return **"true"**.<br>If *argument* is **false**, return **"false"**. |
| Number | Return Number::toString(*argument*). |
| String | Return *argument*. |
| Object | Apply the following steps:<br>1. Let *primValue* be ? ToPrimitive(*argument*, string).<br>2. Return ? ToString(*primValue*). |

### ToBoolean Conversions

| Argument Type | Result |
|---|---|
| Undefined | Return **false**. |
| Null | Return **false**. |
| Boolean | Return *argument*. |
| Number | If *argument* is **+0**$_{\mathbb{F}}$, **-0**$_{\mathbb{F}}$, or **NaN**, return **false**;<br>otherwise return **true**. |
| String | If *argument* is the empty String (its length is 0), return **false**;<br>otherwise return **true**. |
| Object | Return **true**. |

| 原始值 | 转换为数值 | 转换为字符串 | 转换为布尔值 |
|---|---|---|---|
| false | 0 | "false" | false |
| true | 1 | "true" | true |
| 0 | 0 | "0" | false |
| 1 | 1 | "1" | true |
| "0" | 0 | "0" | **true** |
| "000" | 0 | "000" | **true** |
| "1" | 1 | "1" | true |
| NaN | NaN | "NaN" | false |
| Infinity | Infinity | "Infinity" | true |
| -Infinity | -Infinity | "-Infinity" | true |
| "" | **0** | "" | **false** |
| "20" | 20 | "20" | true |

| 原始值 | 转换为数值 | 转换为字符串 | 转换为布尔值 |
|---|---|---|---|
| "Runoob" | NaN | "Runoob" | true |
| [ ] | **0** | "" | true |
| [20] | **20** | "20" | true |
| [10,20] | NaN | "10,20" | true |
| ["Runoob"] | NaN | "Runoob" | true |
| ["Runoob","Google"] | NaN | "Runoob,Google" | true |
| function(){} | NaN | "function(){}" | true |
| { } | NaN | "[object Object]" | true |
| null | **0** | "null" | false |
| undefined | NaN | "undefined" | false |

学习更多：
https://tc39.es/ecma262/#sec-type-conversion

# 2.3.6 运算符

- 与C/C++的运算符类似
  - 优先级：决定了表达式中运算执行的先后顺序
  - 结合性：多个具有同样优先级的运算符表达式中的运算顺序

| 优先级 | 运算符 | 说明 | 结合性 |
|---|---|---|---|
| 1 | []、.、() | 字段访问、数组索引、函数调用和表达式分组 | 从左向右 |
| 2 | ++ -- -~!delete new typeof void | 一元运算符、返回数据类型、对象创建、未定义的值 | 从右向左 |
| 3 | *、/、% | 相乘、相除、求余数 | 从左向右 |
| 4 | +、- | 相加、相减、字符串串联 | 从左向右 |
| 5 | <<、>>、>>> | 左位移、右位移、无符号右移 | 从左向右 |
| 6 | <、<=、>、>=、instanceof | 小于、小于或等于、大于、大于或等于、是否为特定类的实例 | 从左向右 |
| 7 | ==、!=、===、!== | 相等、不相等、全等，不全等 | 从左向右 |

| 优先级 | 运算符 | 说明 | 结合性 |
|---|---|---|---|
| 8 | & | 按位“与” | 从左向右 |
| 9 | ^ | 按位“异或” | 从左向右 |
| 10 | \| | 按位“或” | 从左向右 |
| 11 | && | 短路与（逻辑“与”） | 从左向右 |
| 12 | \|\| | 短路或（逻辑“或”） | 从左向右 |
| 13 | ?: | 条件运算符 | 从右向左 |
| 14 | =、+=、-=、*=、/=、%=、&=、\|=、^=、<、<=、>、>=、>>= | 混合赋值运算符 | 从右向左 |
| 15 | , | 多个计算 | 按优先级计算，然后从右向左 |

使用（）改变默认的优先级和结合性

# 2.3.6 运算符

- +：既是字符串运算符又是加法运算符（类比C++操作符重载）
  - 左值/右值中只要有一个是string类型，则进行字符串拼接
  - 否则尝试把左值/右值均转化内成number类型，进行算术加法

**ApplyStringOrNumericBinaryOperator ( *lval*, *opText*, *rval* )**

The abstract operation ApplyStringOrNumericBinaryOperator takes arguments *lval* (an ECMAScript language value), *opText* (**, *, /, %, +, -, <<, >>, >>>, &, ^, or |), and *rval* (an ECMAScript language value) and returns either a normal completion containing either a String, a BigInt, or a Number, or a throw completion. It performs the following steps when called:

1. If *opText* is +, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*).
   c. If Type(*lprim*) is String or Type(*rprim*) is String, then
      i. Let *lstr* be ? ToString(*lprim*).
      ii. Let *rstr* be ? ToString(*rprim*).
      iii. Return the string-concatenation of *lstr* and *rstr*.
   d. Set *lval* to *lprim*.
   e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.
3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is different from Type(*rnum*), throw a **TypeError** exception.
6. If Type(*lnum*) is BigInt, then
   a. If *opText* is **, return ? BigInt::exponentiate(*lnum*, *rnum*).
   b. If *opText* is /, return ? BigInt::divide(*lnum*, *rnum*).
   c. If *opText* is %, return ? BigInt::remainder(*lnum*, *rnum*).
   d. If *opText* is >>>, return ? BigInt::unsignedRightShift(*lnum*, *rnum*).
7. Let *operation* be the abstract operation associated with *opText* and Type(*lnum*) in the following table:
8. Return *operation*(*lnum*, *rnum*).

```
a = "3.145";          //字符串
b = 3.145;            //数值

c = a+2;              //字符串拼接
d = b+2;              //算数加法

//使用显式类型转换改变默认的隐式类型转换规则
e = String(Number(a) + 2);

alert(c); //3.1452
alert(d); //5.145
alert(e); //5.145
```

学习更多：
https://tc39.es/ecma262/#sec-additive-operators

# 2.3.6 运算符

- ==/!=和===/!==区别
  - ==/!=：先进行类型转换，再比较值（值相等，则为true） <= LooselyEqual
  - ===/!==：比较类型和值（值相等且类型相同，则为true）<= StrictlyEqual

**IsLooselyEqual ( *x*, *y* )**

The abstract operation IsLooselyEqual takes arguments *x* (an ECMAScript language value) and *y* (an ECMAScript language value) and returns either a normal completion containing a Boolean or a throw completion. It provides the semantics for the comparison *x* == *y*. It performs the following steps when called:

1. If Type(*x*) is the same as Type(*y*), then
   a. Return IsStrictlyEqual(*x*, *y*).
2. If *x* is **null** and *y* is **undefined**, return **true**.
3. If *x* is **undefined** and *y* is **null**, return **true**.
4. NOTE: This step is replaced in section B.3.6.2.
5. If Type(*x*) is Number and Type(*y*) is String, return ! IsLooselyEqual(*x*, ! ToNumber(*y*)).
6. If Type(*x*) is String and Type(*y*) is Number, return ! IsLooselyEqual(! ToNumber(*x*), *y*).
7. If Type(*x*) is BigInt and Type(*y*) is String, then
   a. Let *n* be StringToBigInt(*y*).
   b. If *n* is **undefined**, return **false**.
   c. Return ! IsLooselyEqual(*x*, *n*).
8. If Type(*x*) is String and Type(*y*) is BigInt, return ! IsLooselyEqual(*y*, *x*).
9. If Type(*x*) is Boolean, return ! IsLooselyEqual(! ToNumber(*x*), *y*).
10. If Type(*y*) is Boolean, return ! IsLooselyEqual(*x*, ! ToNumber(*y*)).
11. If Type(*x*) is either String, Number, BigInt, or Symbol and Type(*y*) is Object, return ! IsLooselyEqual(*x*, ? ToPrimitive(*y*)).
12. If Type(*x*) is Object and Type(*y*) is either String, Number, BigInt, or Symbol, return ! IsLooselyEqual(? ToPrimitive(*x*), *y*).
13. If Type(*x*) is BigInt and Type(*y*) is Number, or if Type(*x*) is Number and Type(*y*) is BigInt, then
    a. If *x* or *y* are any of **NaN**, $+\infty_{\mathbb{F}}$, or $-\infty_{\mathbb{F}}$, return **false**.
    b. If $\mathbb{R}(x) = \mathbb{R}(y)$, return **true**; otherwise return **false**.
14. Return **false**.

**IsStrictlyEqual ( *x*, *y* )**

The abstract operation IsStrictlyEqual takes arguments *x* (an ECMAScript language value) and *y* (an ECMAScript language value) and returns a Boolean. It provides the semantics for the comparison *x* === *y*. It performs the following steps when called:

1. If Type(*x*) is different from Type(*y*), return **false**.
2. If Type(*x*) is Number, then
   a. Return Number::equal(*x*, *y*).
3. If Type(*x*) is BigInt, then
   a. Return BigInt::equal(*x*, *y*).
4. Return SameValueNonNumeric(*x*, *y*).

```
alert(5=="5");          //true
alert(5==="5");         //false

alert(1==true);         //true
alert(1===true);        //false

alert(false=="");       //true
alert(false==="");      //false
```

学习更多：
https://tc39.es/ecma262/#sec-equality-operators

# 2.3.7 函数

- 函数的主要功能是将代码组织为可复用的单位，可以完成特定的任务并返回数据
- 函数是JavaScript组织代码的基本单位

```
function 函数名( [ 参数1, [ 参数2, [ 参数N ] ] ] )
{
        [ 语句组 ];
        [ return [表达式] ];
}
```

- **function**：必选项，定义函数用的关键字。
- 函数名：必选项，合的**JavaScript**标识符。
- 参数：可选项，合法的**JavaScript**标识符，外部的数据可以通过参数传送到函数内部。
- 语句组：可选项，**JavaScript**程序语句，当为空时函数没有任何动作
- **return**：可选项，遇到此指令函数执行结束并返回，当省略该项时函数将在右花括号处结束。
- 表达式：可选项，其值作为函数返回值。

# 函数表达式

- 将声明的函数赋值给一个变量，通过变量完成函数调用和参数的传递，它也是JavaScript中另一种实现自定义函数的方式

① 函数的定义方式不同

② 函数的调用方式不同

③ 函数定义与调用顺序不同

```
var fn = function sum(num1, num2) {
  return num1 + num2;
};
fn();
```

函数表达式

```
sum();
function sum(num1, num2) {
  return num1 + num2;
};
```

函数声明方式

# 匿名函数

- 定义：没有函数名称的函数，既是函数表达式的另一种表示形式，又可以通过函数声明的方式实现调用
- 作用：可以有效避免全局变量污染以及函数名的冲突问题

① 函数表达式中省略函数名

```
var fn = function (num1, num2) {
  return  num1 + num2;
};
fn(1, 2);
```

② 自调用方式

```
(function (num1, num2) {
  return  num1 + num2;
})(2, 3);
```

③ 处理事件

```
document.body.onclick  = function () {
  alert('Hi, everybody!');
};
```

# 回调函数

- 定义：一个函数A作为参数传递给一个函数B，然后在B函数体内调用函数A，此时，我们称函数A为回调函数
- 作用：函数体中某部分功能由调用者决定，此时可用回调函数

| 方法名称 | 功能描述 |
| --- | --- |
| find() | 返回数组中满足回调函数的第一个元素的值，否则返回undefined |
| every() | 测试数组的所有元素是否都通过了回调函数的测试 |
| some() | 测试数组中的某些元素是否通过由回调函数实现的测试 |
| forEach() | 对数组的每个元素执行一次提供的函数 |
| map() | 创建一个新数组，其结果是该数组中的每个元素都调用一次提供的回调函数后返回的结果 |
| reduce() | 对累加器和数组中的每个元素（从左到右）应用一个函数，将其减少为单个值 |
| reduceRight() | 接收一个函数作为累加器（accumulator）和数组的每个值（从右到左）将其减少为单个值 |

在JavaScript中为数组提供了很多利用回调函数实现具体功能的方法

# 回调函数

- 以map()方法为例进行演示，对arr数组中的每个元素都按顺序调用一次回调函数。

```
var arr = ['a', 'b', 'c'];
arr.map(function(value, index) {
  console.log(value, index);
});
```

- 参数：

map()的参数是一个回调函数fn。

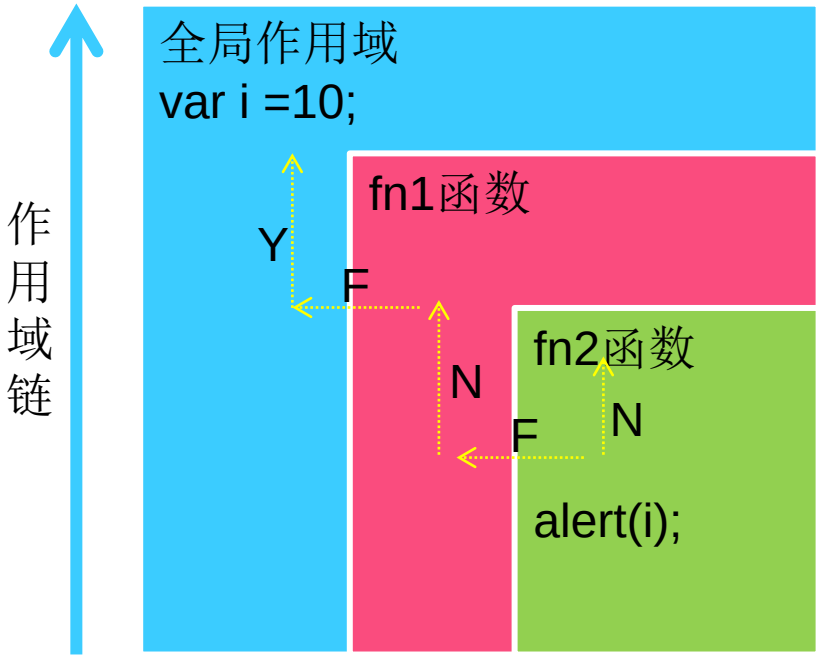
fn的第1个参数表示当前数组的元素。

fn的第2个参数表示对应元素的索引下标。

- 返回值：回调函数每次执行后的返回值会组合起来形成一个新数组。
- 示例结果：在控制台依次可查看到，“a 0”、“b 1”和“c 2”。

# 函数嵌套与作用域链

- 函数嵌套：在一个函数内部存在另一个函数的声明
- 作用域链：各层嵌套函数局部作用域由内到外构成的链
  - 内层函数只能在外层函数作用域内执行
  - 在内层函数执行过程中，若需要引入某个变量，会在当前作用域中寻找
  - 若未找到，则继续向上一层级的作用域中寻找，直到全局作用域链




符号含义参考

F：表示向上查找；N：没有找到；Y：找到

```
<script type="text/javascript">
  var i = 10; //全局变量i

  function fn1() { //在全局作用域定义函数fn1
    function fn2() { //在fn1局部作用域定义函数fn2
      alert(i);
    }
    fn2(); //在fn1局部作用域调用函数fn2
  }
  fn1();
</script>
```

# 2.3.8 对象

- JavaScript是基于对象的语言，JavaScript中的对象本质是由名-值对构成的集合
- 分类：
  - 内置对象
  - 浏览器对象
  - 自定义对象

# 内置对象

- Math数学对象：所有基本数学计算的功能和常量的属性和方法
- Date日期对象：获取、设置日期和时间的属性和方法
- String字符串对象：对字符串进行处理的属性和方法
- Array数组对象：数组操作相关的属性和方法

| 方法 | 作用 |
|---|---|
| abs | 返回某数的绝对数 |
| ceil | 返回大于或等于指定数的最小整数 |
| floor | 返回小于等于其数值参数的最大整数 |
| random | 返回[0-1)之间的随机数 |
| round | 返回四舍五入后的整数 |

Math对象

String对象

| 属性/方法 | 说明 | 用法 |
|---|---|---|
| length | 字符串长度 | strObj.length |
| charAt | 返回指定索引位置的字符 | strObj.charAt(index) |
| concat | 进行字符串连接 | str1.concat(str2,str3) |
| indexOf | String 对象内第一次出现子字符串的字符位置 | str1.indexOf(str2[, startIndex]) |
| lastIndexOf | 返回 String 对象中子字符串最后出现的位置 | str1.lastIndexOf(str2[, startindex]) |
| substring | 返回 String 对象中指定位置的子字符串 | **str1.substring(start, end)** |
| toLowerCase<br>toUpperCase | 大小写转换 | str1.toLowerCase( ) |

| 方法 | 作用 | 返回值 |
|---|---|---|
| getYear | 返回年度 | |
| getMonth | 返回月份 | 0-11 |
| getDate | 返回日期 | 1-31 |
| getDay | 返回星期几 | 0-6 |
| getHours | 返回小时数 | 0-23 |
| getMinutes | 返回分钟数 | 0-59 |
| getSeconds | 返回秒数 | 0-59 |

Data对象

Array对象

| 属性/方法 | 作用 | 格式 |
|---|---|---|
| length | 获得数组元素个数 | 数组.length |
| toString | 数组转换为字符串,各个数组元素用逗号连接 | 数组.toString() |
| join | 将数组组合为字符串,由分隔符隔开 | 数组.join(分隔符) |
| push | 添加若干个元素到数组末端 | 数组.push(元素1,元素2) |
| pop | 删除最后一个元素 | 数组.pop() |
| reverse | 将数组内元素的索引次序翻转 | 数组.reverse() |

# 内置对象

```
<!-- Math对象使用实例：随机生成一个0-9之间的整数 -->

<script type="text/javascript">
  x = Math.random(); //返回一个[0,1)之间的随机浮点数x
  y = x * 10;        //y是一个[0, 10)之间的随机浮点数
  z = floor(y);      //对y下取整，得一个[0, 10)之间的整数
  alert(z);
</script>
```

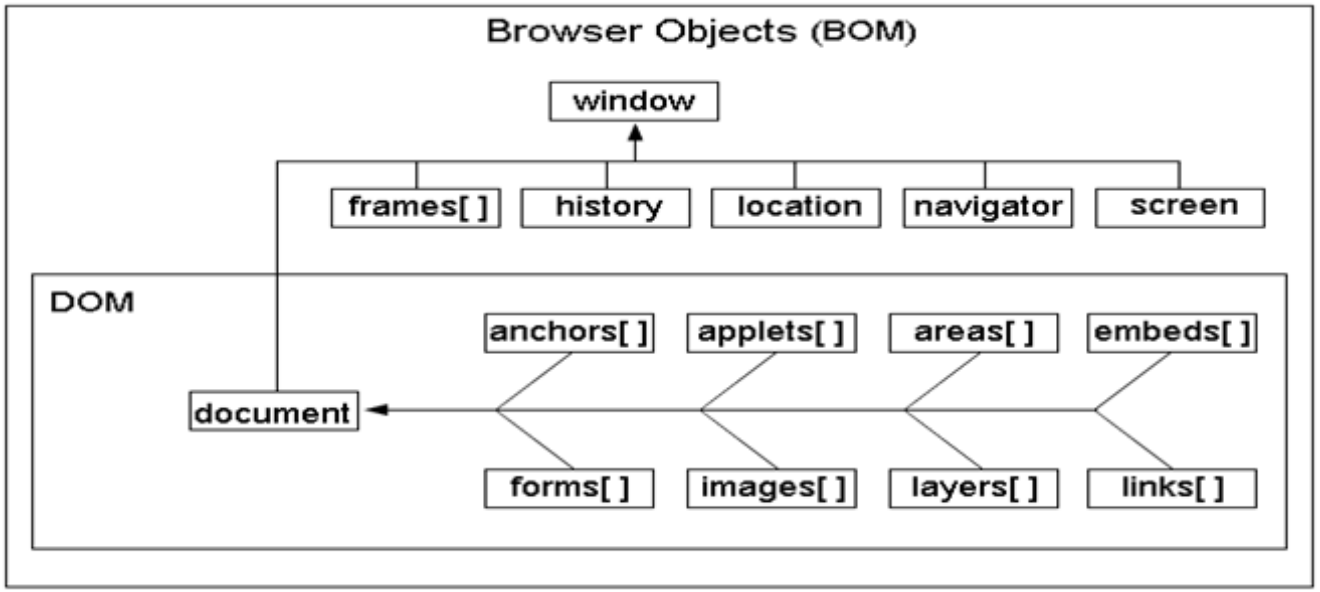
```
<!-- String对象使用实例：获得URL字符串中的主机名 -->

<script type="text/javascript">
  url = "http://www.youwei.site/ruc.png";
  start = url.indexOf("//");       //查找“//”所在位置
  end = url.indexOf("/", start+2); //查找“/”所在位置
  host = url.substring(start+2,end); //[start+2,end)子串
</script>
```

```
<!-- Date对象使用实例：获取当前日期的年/月/日和时/分/秒 -->

<script type="text/javascript">
  date = new Date();
  year = date.getYear()+1900; //getYear()+1900为当前年份
  month = date.getMonth()+1;  //getMonth()+1为当前月份
  day = date.getDay();        //getDay()为当前日期
  hour = date.getHours();     //getDay()为当前时点
  minute = date.getMinutes(); //getDay()为当前分点
  second = date.getSeconds(); //getDay()为当前秒点
</script>
```

```
<!-- Array对象使用实例：用数组实现栈/队列操作 -->

<script type="text/javascript">
  stack = [];      //模拟栈，后进先出
  stack.push(1); stack.push(2);
  alert(stack.pop()); alert(stack.pop()); //2 1

  queue = Array(); //模拟队列，先进先出
  queue.unshift(1); queue.unshift(2);
  alert(queue.pop()); alert(pop.shift()); //1 2
</script>
```

# 2.4 BOM/DOM

- BOM（Browser Object Model）：浏览器对象模型，定义了操作浏览器的标准方法，根对象是window
- DOM（Document Object Model）：文档对象模型，定义了操作HTML文档的标准方法，根对象是document



# 2.4.1 BOM

- BOM：浏览器对象模型，用于访问和网页无关的浏览器功能。

☑**window**：处于对象层次的最顶端,提供处理窗口的方法和属性；

☑**document**：装入到当前窗口的文档，包含一个文档所有的标签元素，将其封装供使用；

☑**location**：用于创建、访问或修改URL对象；

☑**history**：包含客户机先前已经请求过的URL；

☑**screen**：提供关于用户屏幕的信息，如大小颜色等；

☑**navigator**：提供关于浏览器对象的信息,包括浏览器版本、型号等信息

学习更多：https://www.w3school.com.cn/js/js_window.asp

# window

- 功能：表示浏览器的一个实例，同时也是一个Global对象

| 属性 | 描述 |
| --- | --- |
| closed | 返回窗口是否已被关闭。 |
| defaultStatus | 设置或返回窗口状态栏中的默认文本。 |
| innerHeight | 返回窗口的文档显示区的高度。 |
| innerWidth | 返回窗口的文档显示区的宽度。 |
| length | 设置或返回窗口中的框架数量。 |
| name | 设置或返回窗口的名称。 |
| opener | 返回对创建此窗口的窗口的引用。 |
| outerHeight | 返回窗口的外部高度，包含工具条与滚动条。 |
| outerWidth | 返回窗口的外部宽度，包含工具条与滚动条。 |
| pageXOffset | 设置或返回当前页面相对于窗口显示区左上角的 X 位置。 |

| 属性 | 描述 |
| --- | --- |
| pageYOffset | 设置或返回当前页面相对于窗口显示区左上角的 Y 位置。 |
| parent | 返回父窗口。 |
| screenLeft | 返回相对于屏幕窗口的x坐标 |
| screenTop | 返回相对于屏幕窗口的y坐标 |
| screenX | 返回相对于屏幕窗口的x坐标 |
| screenY | 返回相对于屏幕窗口的y坐标 |
| self | 返回对当前窗口的引用。等价于 Window 属性。 |
| status | 设置窗口状态栏的文本。 |
| top | 返回最顶层的父窗口。 |

| 方法 | 描述 |
| --- | --- |
| alert() | 显示带有一段消息和一个确认按钮的警告框。 |
| atob() | 解码一个 base-64 编码的字符串。 |
| btoa() | 创建一个 base-64 编码的字符串。 |
| blur() | 把键盘焦点从顶层窗口移开。 |
| clearInterval() | 取消由 setInterval() 设置的 timeout。 |
| clearTimeout() | 取消由 setTimeout() 方法设置的 timeout。 |
| close() | 关闭浏览器窗口。 |
| confirm() | 显示带有一段消息以及确认按钮和取消按钮的对话框。 |
| createPopup() | 创建一个 pop-up 窗口。 |
| focus() | 把键盘焦点给予一个窗口。 |
| getSelection() | 返回一个 Selection 对象，表示用户选择的文本范围或光标的当前位置。 |
| getComputedStyle() | 获取指定元素的 CSS 样式。 |
| matchMedia() | 该方法用来检查 media query 语句，它返回一个 MediaQueryList对象。 |
| moveBy() | 可相对窗口的当前坐标把它移动指定的像素。 |

| 方法 | 描述 |
| --- | --- |
| moveTo() | 把窗口的左上角移动到一个指定的坐标。 |
| open() | 打开一个新的浏览器窗口或查找一个已命名的窗口。 |
| print() | 打印当前窗口的内容。 |
| prompt() | 显示可提示用户输入的对话框。 |
| resizeBy() | 按照指定的像素调整窗口的大小。 |
| resizeTo() | 把窗口的大小调整到指定的宽度和高度。 |
| scroll() | 已废弃。 该方法已经使用了 scrollTo() 方法来替代。 |
| scrollBy() | 按照指定的像素值来滚动内容。 |
| scrollTo() | 把内容滚动到指定的坐标。 |
| setInterval() | 按照指定的周期（以毫秒计）来调用函数或计算表达式。 |
| setTimeout() | 在指定的毫秒数后调用函数或计算表达式。 |
| stop() | 停止页面载入。 |
| postMessage() | 安全地实现跨源通信。 |

# 窗口操作

- 打开新窗口：win = window.open(url, name, features, replace)
- 关闭窗口：win.close()

| 参数 | 说明 |
| --- | --- |
| URL | 可选。打开指定的页面的URL。如果没有指定URL，打开一个新的空白窗口 |
| name | 可选。指定target属性或窗口的名称。支持以下值：<br>• _blank - URL加载到一个新的窗口。这是默认<br>• _parent - URL加载到父框架<br>• _self - URL替换当前页面<br>• _top - URL替换任何可加载的框架集<br>• *name* - 窗口名称 |
| replace | Optional.Specifies规定了装载到窗口的 URL 是在窗口的浏览历史中创建一个新条目，还是替换浏览历史中的当前条目。支持下面的值：<br>• true - URL 替换浏览历史中的当前条目。<br>• false - URL 在浏览历史中创建新的条目。 |

```
<html>
<body>
  <input type="submit" value="打开"
         onclick="window.win = open('http://www.ruc.edu.cn',
                                    '_blank','width=200,height=200')">
  <input type="submit" value="关闭"
         onclick="win.close();")>
</body>
</html>
```

| 参数 | 说明 |
| --- | --- |
| specs | 可选。一个逗号分隔的项目列表。支持以下值： |

| | |
| --- | --- |
| channelmode=yes\|no\|1\|0 | 是否要在影院模式显示 window。默认是没有的。仅限IE浏览器 |
| directories=yes\|no\|1\|0 | 是否添加目录按钮。默认是肯定的。仅限IE浏览器 |
| fullscreen=yes\|no\|1\|0 | 浏览器是否显示全屏模式。默认是没有的。在全屏模式下的 window，还必须在影院模式。仅限IE浏览器 |
| height=pixels | 窗口的高度。最小.值为100 |
| left=pixels | 该窗口的左侧位置 |
| location=yes\|no\|1\|0 | 是否显示地址字段.默认值是yes |
| menubar=yes\|no\|1\|0 | 是否显示菜单栏.默认值是yes |
| resizable=yes\|no\|1\|0 | 是否可调整窗口大小.默认值是yes |
| scrollbars=yes\|no\|1\|0 | 是否显示滚动条.默认值是yes |
| status=yes\|no\|1\|0 | 是否要添加一个状态栏.默认值是yes |
| titlebar=yes\|no\|1\|0 | 是否显示标题栏.被忽略，除非调用HTML应用程序或一个值得信赖的对话框.默认值是yes |
| toolbar=yes\|no\|1\|0 | 是否显示浏览器工具栏.默认值是yes |
| top=pixels | 窗口顶部的位置.仅限IE浏览器 |
| width=pixels | 窗口的宽度.最小.值为100 |

# 延时执行与周期性执行

- 延迟代码执行：tid = setTimeout(code, delay)
- 取消延迟执行：clearTimeout(tid)
- 周期性执行：tid = setInterval(code, interval)
- 取消周期性执行：clearInterval(tid)

```
<html>
<head>
  <script>
    function showClock() //自定义函数
    {
      d = new Date(); //创建一个时间对象
      document.title = d.toLocaleString(); //标题上显示当前时间
      tid = window.setTimeout("showClock()",1000); //1秒更新一次
    }
  </script>
</head>
<body>
  <input type="submit" value="开始"
         onclick="showClock()">
  <input type="submit" value="取消"
         onclick="window.clearTimeout(tid);")>
</body>
</html>
```

```
<html>
<head>
  <script>
    function showClock() //自定义函数
    {
      d = new Date(); //创建一个时间对象
      document.title = d.toLocaleString(); //标题上显示当前时间
    }
  </script>
</head>

<body>
  <input type="submit" value="开始"
  onclick="window.tid = window.setInterval('showClock()',1000")>
  <input type="submit" value="取消"
         onclick="window.clearTimeout(tid);")>
</body>
</html>
```

# location

- 功能：表示窗口中当前显示文档的URL地址，并提供对URL进行解析和操作的属性和方法

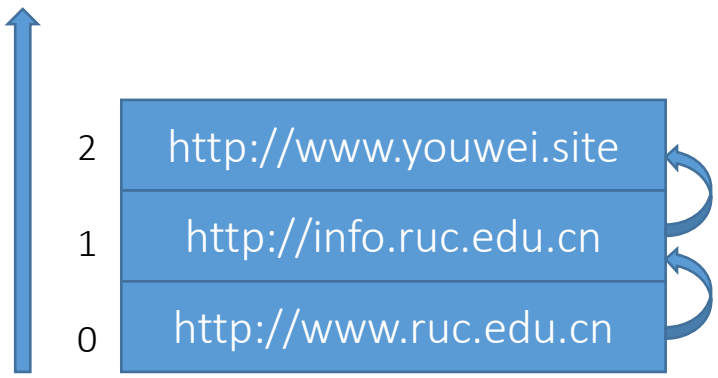
| 属性和方法 | 说明 |
| --- | --- |
| **href** | 获取或设置当前页面的URL,是location的默认属性 |
| **host** | 服务器名字 |
| **pathname** | URL中主机后面的名字 |
| **port** | URL请求中的端口号 |
| **protocol** | URL中使用的协议 |
| **search** | 执行get请求的URL中的？后面部分 |
| **reload()** | 重新载入当前页面 |

```
//当前窗口URL：https://www.youwei.site:8080/uv/game/rank.php?action=show_rank

alert(location.href); /* https://www.youwei.site:8080/uv/game/rank.php?action=show_rank */
alert(location.host); /* www.youwei.site */
alert(location.pathname); /* /uv/game/rank.php */
alert(location.port); /* 8080 */
alert(location.protocol); /* https */
alert(location.search); /* ?action=show_rank */
location.href = "http://www.ruc.edu.cn"; /* 页面跳转 */
```

# history

- 功能：表示用户浏览过的URL历史，并提供对浏览历史URL记录的访问操作

| 属性和方法 | 说明 |
| --- | --- |
| **go(i)** | 加载 history 列表中的某个具体页面，i代表相对位置：-1上一个页面，1前进一个页面 |
| **back()** | 加载 history 列表中的前一个 URL（等价于点击后退按钮或调history.go(-1)） |
| **forward()** | 加载 history 列表中的下一个 URL（等价于点击前进按钮或 history.go(1)） |



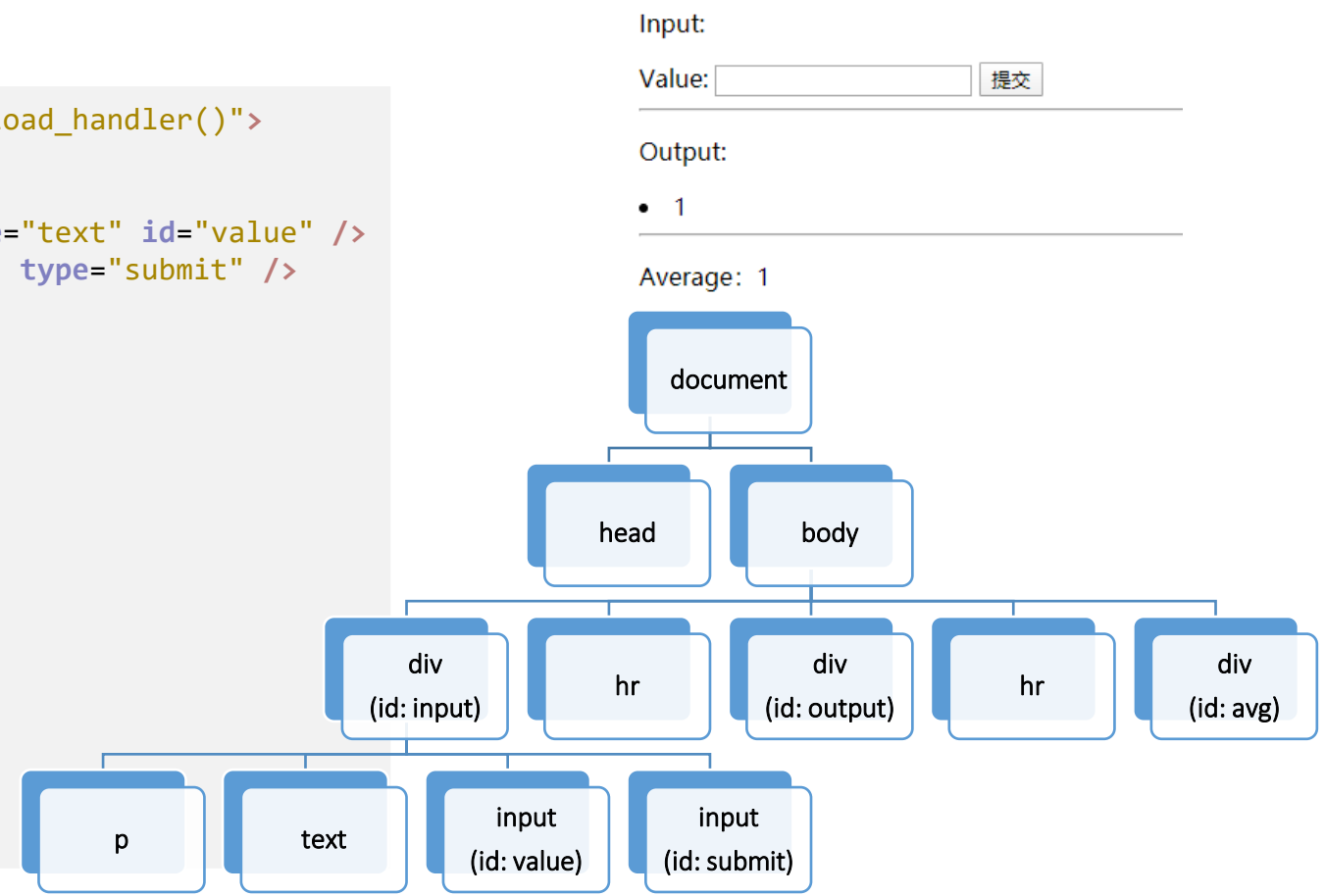

浏览历史栈

```
//当前窗口URL：http://info.ruc.edu.cn

/*
欲访问http://www.ruc.edu.cn：history.back(); 或者 history.go(-1);
欲访问http://www.youwei.site：history.forward(); 或者 history.go(1);
*/
```

# 2.4.2 DOM

- 将整个HTML文档描述成一棵树，文档中的每个标签对应DOM树的一个节点

```
<body onload="body_onload_handler()">
  <div id="input">
    <p>Input:</p>
    Value: <input type="text" id="value" />
    <input id="submit" type="submit" />
  </div>

  <hr />

  <div id="output">
    <p>Output:</p>
    <li>1</li>
  </div>

  <hr />

  <div id="avg">
    <p>Average: 1</p>
  </div>
</body>
```




学习更多：https://www.w3school.com.cn/js/js_htmldom.asp

# DOM操作

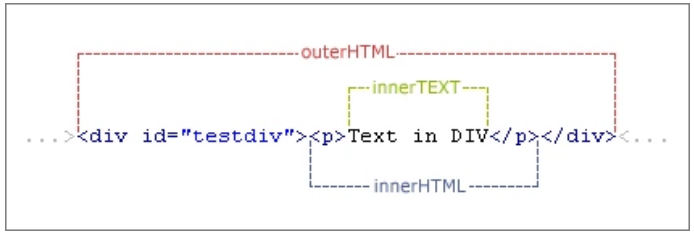

定位输入框：

- document.getElementById("value")
- document.body.childNodes[1].firstChild.nextSibling.nextSibling.nextSibling

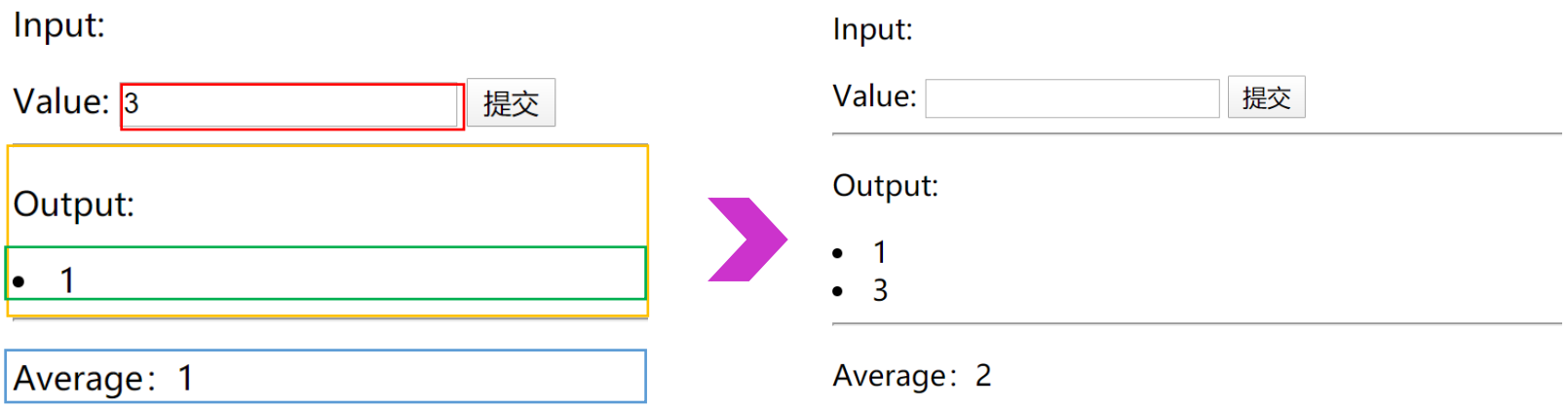
- 获取节点
  - 按id查询：node = document.getElementById(id)
  - 按类名查询：node = document.getElementsByClass(class)
  - 按标签名查询：node = document.getElementsByTagName(tag)
- 节点指针
  - 返回目标节点的所有子节点的列表：node.childNodes
  - 指向在childNodes列表中的第一个节点：node.firstChild
  - 指向在childNodes列表中的最后一个节点：node.lastChild
  - 指向父节点：node.parentNode
  - 指向前一个兄弟节点：node.previousSibling
  - 指向后一个兄弟节点：node.nextSibling
  - 指向这个节点所属的文档：node.ownerDocument

# DOM操作

- 节点操作
  - 创建节点：document.createElement(tag)
  - 添加到指定位置：parentNode.appendChild(newNode);<br>parentNode.insertBefore(newNode, targetNode);
  - 删除节点：parentNode.removeChild(childNode)
  - 替换节点：parentNode.replaceChild(newNode, oldNode)
- 属性操作
  - 获取属性：node.getAttribute(attrName)
  - 设置属性：node.setAttribute(attrName, value)
  - 删除属性：node.removeAttribute(attrName)
- 文本操作
  - innerText：节点内部内容(包括目标元素标签)
  - innerHTML：节点内部内容(不含目标元素标签)
  - outerHTML：含目标元素标签的节点完整内容



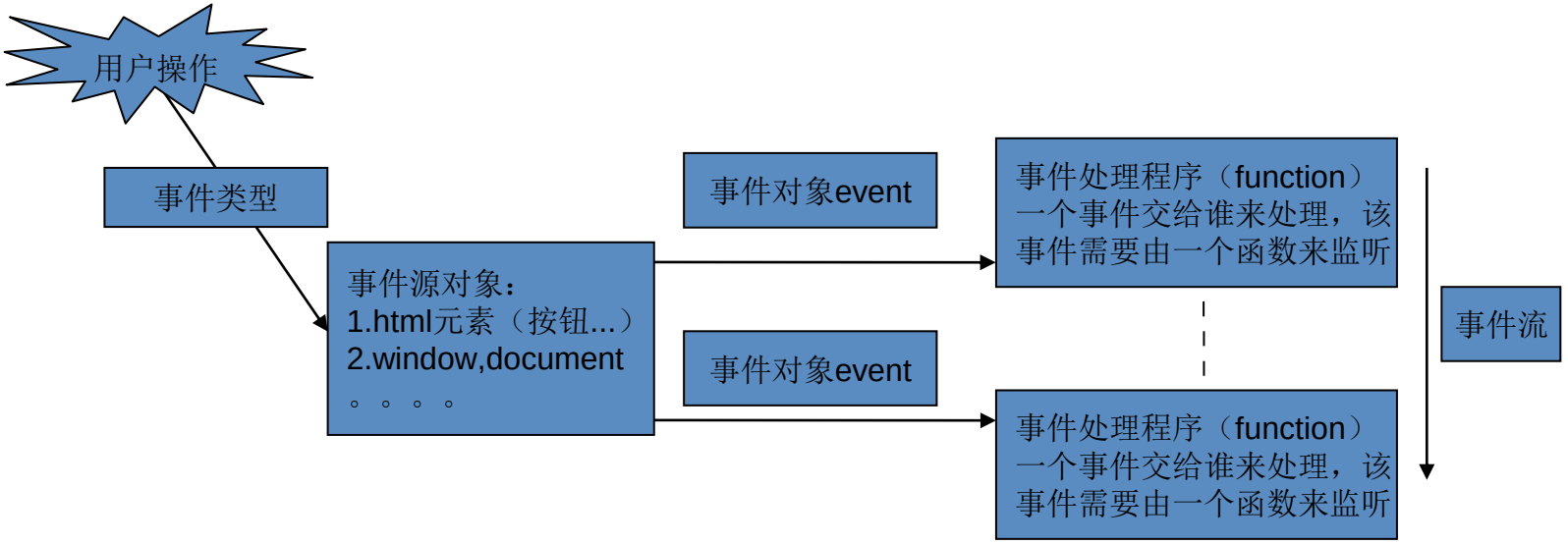
# DOM操作示例




```
/* 当提交按钮被点击时触发执行onclick_handler()函数 */
function onclick_handler() {
  value = document.getElementById("value");          //找到value输入框对象
  output = document.getElementsByTagName("div")[1];  //找到output区块对象
  li = document.createElement("li");                 //创建li列表对象
  li.innerHTML = value.value;                        //设置li列表对象的内容为value输入框对象的值
  output.appendChild(li);                            //将新创建的li列表对象添加为output区块对象的儿子节点
  value.value = "";                                  //清空value输入框对象的值
}
/* 当output区块插入DOM节点时触发执onDOMNodeInserted_handler函数 */
function onDOMNodeInserted_handler() {
  avg = document.getElementById("avg");              //找到avg区块对象
  lis = document.getElementsByTagName("li");         //找到所有的li列表对象
  sum = 0;                                           //设置变量sum的初始值为0
  for (i = 0; i < lis.length; i++) {                 //遍历每一个li列表对象
    sum += Number(lis[i].innerText);                 //将li列表对象的内容显示转化为数值内容，累加到sum变量
  }
  avg.innerText = "Average: " + sum/lis.length;      //设置avg区块对象的内容为sum与li列表对象个数的商（平均值）
}
```

# 2.5 Event

- 事件驱动编程机制：事件是一种异步编程的实现方式，是一种传递信息的机制，为Web程序提供了互动性
- 用户在某个页面元素（事件源对象）上进行某种类型的操作（事件类型）时产生事件对象，浏览器会按照特定顺序（事件流）将事件分派给目标页面元素及其祖先元素上绑定的事件处理程序

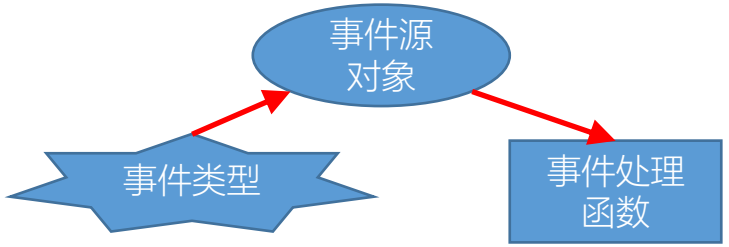



学习更多：https://www.w3school.com.cn/js/js_htmldom_events.asp

# 2.5.1 基本概念

- 事件：可以被JavaScript侦测到的行为，如用户与页面交互时产生的操作（单击鼠标、按下键盘上的某个键）或页面自身的一些行为（如页面加载）
- 事件类型：不同事件类型对应不同的用户交互行为
- 事件源对象：引发事件的元素（事件发生在谁身上）
- 事件处理程序：对事件进行处理的程序或响应某个事件的函数
- 事件对象：在触发某个事件时，会产生一个事件对象event，其中包含所有与事件有关的信息（包括导致事件的元素、事件类型及其他与特定事件相关的信息）
- 事件流：事件流描述的是从页面中接收事件的顺序。事件流开发者认为，当有页面元素触发事件时，该元素的容器控件以及整个页面都按照特定顺序响应该元素触发的事件，事件发生顺序就是事件流

# 2.5.2 绑定事件处理函数




- 行内绑定：<标签 属性列表 事件类型=“事件处理函数的调用” >

```
<body onload="body_onload_handler()">
```

- 动态绑定：事件源对象.事件类型 = 事件处理函数的名字

```
submit = document.getElementById("submit");
submit.onclick = submit_onclick_handler;
```

- 事件监听：事件源对象.addEventListener( “事件类型”,事件处理函数的名字)

```
output = document.getElementById("output");
output.addEventListener(
    "DOMNodeInserted",
    output_onDOMNodeInserted_handler,
    false);
```

Input:

Value: 3 提交

Output:

- 1

Average：1

# 2.5.2 绑定事件处理函数

```
<html>
  <head>
    <script>
      function body_onload_handler(e) {
        submit = document.getElementById("submit");
        submit.onclick = submit_onclick_handler;      //动态绑定
        output = document.getElementById("output");
        output.addEventListener("DOMNodeInserted",    //事件监听
                                output_onDOMNodeInserted_handler, false);
      }
      function button_onclick_handler(e) {
        ……
      }
      function output_onDOMNodeInserted_handler(e) {
        ……
      }
    </script>
  </head>
  <body onload="body_onload_handler()">  <!--行内绑定，浏览器加载事件 -->
    ……
  </body>
<html>
```

# 2.5.3 Web浏览器常见事件

| 事件类型 | 事件名称 | 事件说明 |
| --- | --- | --- |
| 浏览器窗口事件 | load | 页面加载完成时触发 |
| | unload | 窗口关闭时触发 |
| 鼠标事件 | click | 在元素上单击鼠标左键时触发 |
| | mousedown | 在元素上按下鼠标时触发 |
| | mouseup | 在元素上释放鼠标时触发 |
| 键盘事件 | keydown | 当键盘按键按下时触发 |
| | keyup | 当键盘按键释放时触发 |
| 表单事件 | focus | 当表单元素获得焦点时触发 |
| | submit | 当表单提交时触发 |
| 剪贴板事件 | oncopy | 在用户拷贝元素内容时触发 |
| | onpaste | 在用户粘贴元素内容时触发 |
| DOM事件 | DOMNodeInserted | 有DOM元素被添加时触发 |
| | DOMNodeRemoved | 有DOM元素被删除时触发 |
| …… | | |

# 2.5.3 Web浏览器常见事件

```
<html>
  <body>
    <button id="button">点击</button>
    <script type="text/javascript">
      button = document.getElementById("button");
      button.onclick = function(e) { console.log("click"); }          //单击鼠标左键
      button.onmouseup = function(e) { console.log("mouseup"); }      //释放鼠标
      button.onmousedown = function(e) { console.log("mousedown"); }  //按下鼠标
    </script>
  </body>
</html>
```

```
mousedown
mouseup
click
```

```
<html>
  <body>
    <button id="button">点击</button>
    <script type="text/javascript">
      document.onkeydown = function(e) { console.log("keydown: " + e.code); } //键盘按键按下
      document.onkeyup = function(e) { console.log("keyup" + e.code);         //键盘按键释放
      document.oncopy = function(e) { console.log("copy"); }                  //复制行为发生
    </script>
  </body>
</html>
```

```
keydown: ControlLeft
keydown: KeyC
copy
```

# 2.5.4 事件对象

- 事件对象只有事件发生时才会产生，且只能在事件处理函数内访问，在所有事件处理函数运行结束后，事件对象就被销毁

| 属性和方法 | 描述 |
| --- | --- |
| bubbles | 返回布尔值，指示事件是否是起泡事件类型。 |
| cancelable | 返回布尔值，指示事件是否可拥可取消的默认动作。 |
| currentTarget | 返回其事件监听器触发该事件的元素。 |
| eventPhase | 返回事件传播的当前阶段。 |
| target | 返回触发此事件的元素（事件的目标节点）。 |
| timeStamp | 返回事件生成的日期和时间。 |
| type | 返回当前 Event 对象表示的事件的名称。 |
| initEvent() | 初始化新创建的 Event 对象的属性。 |
| preventDefault() | 通知浏览器不要执行与事件关联的默认动作。 |
| stopPropagation() | 不再派发事件。 |

通用事件对象的常见属性和方法

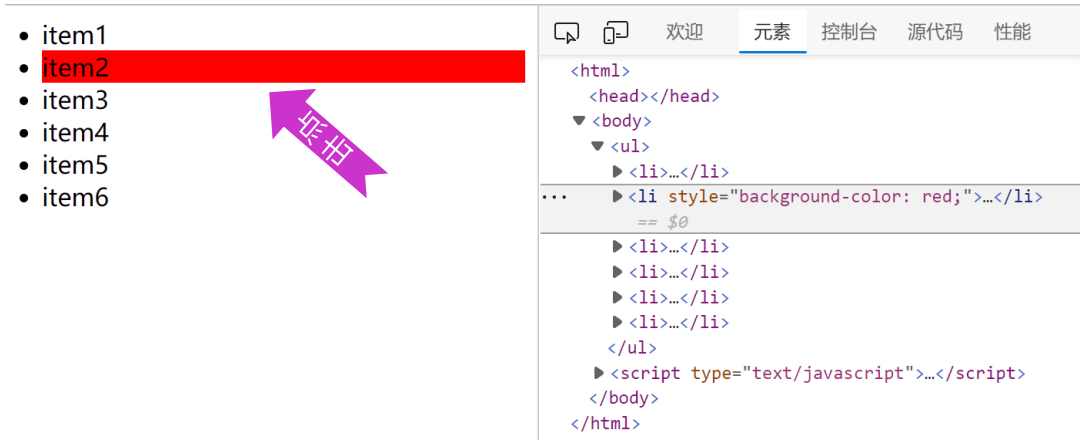
| 事件类型 | 属性和方法 | 描述 |
| --- | --- | --- |
| 键盘事件 | keyCode | 键盘中对应键位的键值 |
| | charCode | 键盘中对应键位的 Unicode 编码 |
| | shiftKey | 是否按下 Shift 键 |
| | ctrlKey | 是否按下 Ctrl 键 |
| | altKey | 是否按下 Alt 键 |
| 鼠标事件 | button | 数值代表按下了鼠标左/中/右键 |
| | clientX | 相对于浏览器窗口左上角的水平坐标 |
| | clientY | 相对于浏览器窗口左上角的垂直坐标 |
| | offsetX | 鼠标位置距离事件源对象左边缘的水平距离 |
| | offsetY | 鼠标位置距离事件源对象左边缘的垂直距离 |
| 剪贴板事件 | clickBoard.getData() | 获得剪贴板中的内容 |
| | clickBoard.setData | 设置剪贴板中的内容 |
| | clickBoard.clearData() | 从剪贴板中删除数据 |

特定事件对象的常见属性和方法

# 2.5.4 事件对象

```html
<html>
  <body>
    <ul>
      <li>item1</li>
      <li>item2</li>
      <li>item3</li>
      <li>item4</li>
      <li>item5</li>
      <li>item6</li>
    </ul>

    <script type="text/javascript">
      ul = document.getElementsByTagName("ul")[0];
      ul.onclick = function(e) {
        alert(e.target); //[object HTMLLIElement]
        alert(e.currentTarget); //[object HTMLUListElement]
        e.target.style.backgroundColor = "red";
      }
    </script>
  </body>
</html>
```

```html
<html>
  <body>
    <ul>
      <li>item1</li>
      <li>item2</li>
      <li>item3</li>
      <li>item4</li>
      <li>item5</li>
      <li>item6</li>
    </ul>

    <script type="text/javascript">
      lis = document.getElementsByTagName("li");
      for (li of lis) {
        li.onclick = function() {
          this.style.backgroundColor = "red";
        }
      }
    </script>
  </body>
</html>
```




左边代码：只在<ul>元素上绑定一个事件处理函数

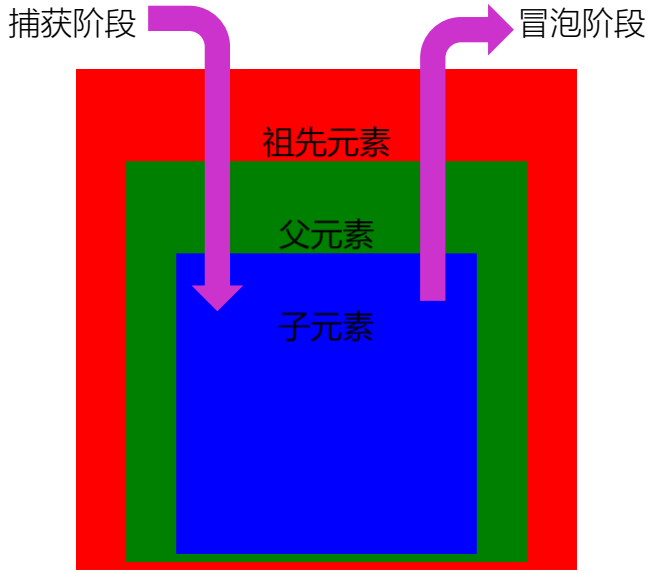
- target是引起触发事件的元素（鼠标点击在item2这个<li>元素上）
- currentTarget是事件绑定的元素（事件处理函数绑定在<ul>上）
- 事件委托：当有多个类似元素（<li>）需要绑定事件时，给其共同父级元素（<ul>）添加一个事件函数，处理所有元素的事件

右边代码：在每一个<li>元素上都绑定一个事件处理函数

- target和currentTarget都是每一个被点击的<li>元素
- 相比于事件委托，对多个类似元素逐一绑定事件处理函数，既浪费时间，又不利于性能

# 2.5.5 事件流

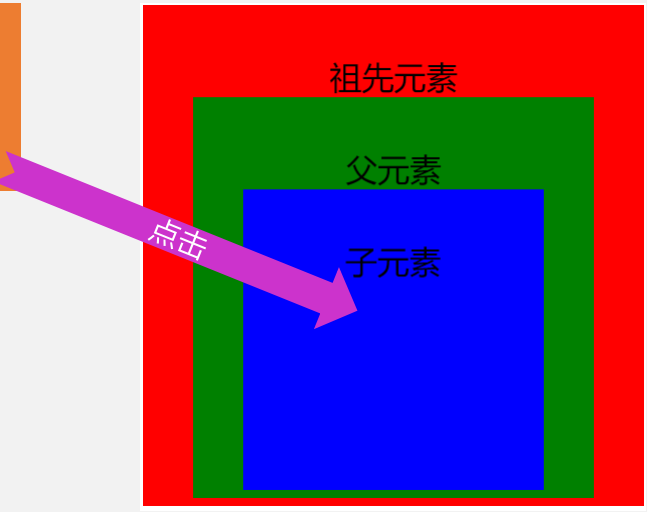
- 当事件发生在某个DOM节点时，事件在DOM结构中进行一级一级的传递，事件流便描述了从页面中接收事件的顺序
- 事件流处理的三个阶段：
  - 捕获过程：事件从Document节点自上而下向目标节点传播的阶段
  - 触发过程：真正的目标节点正在处理事件的阶段
  - 冒泡过程：事件从目标节点自下而上向Document节点传播的阶段



# 2.5.5 事件流

```
<html>
  <head>
    <style>
      div { padding:25px; border:2px; text-align:center; }
      #ancestor { width:200px; height:200px; background-color:red; }
      #parent { width:150px; height:150px; background-color:green; }
      #child { width:100px; height:100px; background-color:blue; }
    </style>
  </head>
  <body>
    <div id="ancestor">
      <span>祖先元素</span>
      <div id="parent">
        <span>父元素</span>
        <div id="child">
          <span>子元素</span>
        </div>
      </div>
    </div>
    <script type="text/javascript">
      ancestor = document.getElementById("ancestor");
      parent = document.getElementById("parent");
      child = document.getElementById("child");
      ancestor.addEventListener("click", function(e){alert("click-ancestor");}, false);
      parent.addEventListener("click", function(e){alert("click-parent");}, false);
      child.addEventListener("click", function(e){alert("click-child");}, false);
    </script>
  </body>
</html>
```

点击子元素时的输出：
click-child
click-parent
click-ancestor



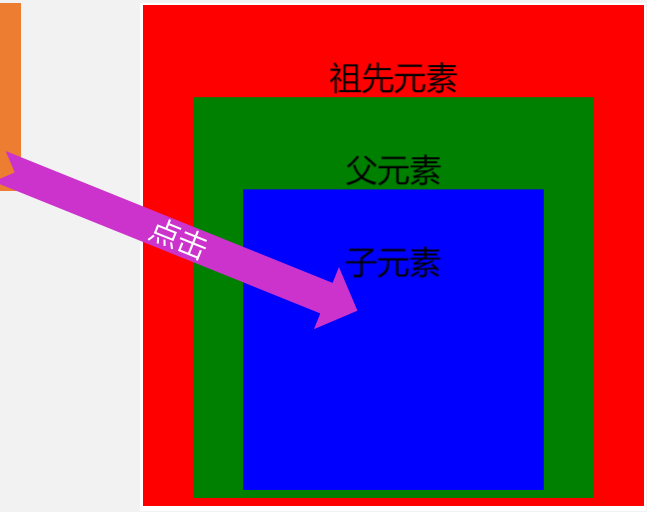

addEventListener的第三个参数指定在哪个阶段调用事件处理函数：

- false: 在冒泡阶段调用事件处理函数，默认值
- true: 在捕获阶段调用事件处理函数

# 2.5.5 事件流

```
<html>
  <head>
    <style>
      div { padding:25px; border:2px; text-align:center; }
      #ancestor { width:200px; height:200px; background-color:red; }
      #parent { width:150px; height:150px; background-color:green; }
      #child { width:100px; height:100px; background-color:blue; }
    </style>
  </head>
  <body>
    <div id="ancestor">
      <span>祖先元素</span>
      <div id="parent">
        <span>父元素</span>
        <div id="child">
          <span>子元素</span>
        </div>
      </div>
    </div>
    <script type="text/javascript">
      ancestor = document.getElementById("ancestor");
      parent = document.getElementById("parent");
      child = document.getElementById("child");
      ancestor.addEventListener("click", function(e){alert("click-ancestor");}, true);
      parent.addEventListener("click", function(e){alert("click-parent");}, true);
      child.addEventListener("click", function(e){alert("click-child");}, true);
    </script>
  </body>
</html>
```

点击子元素时的输出：
click-ancestor
click-parent
click-child



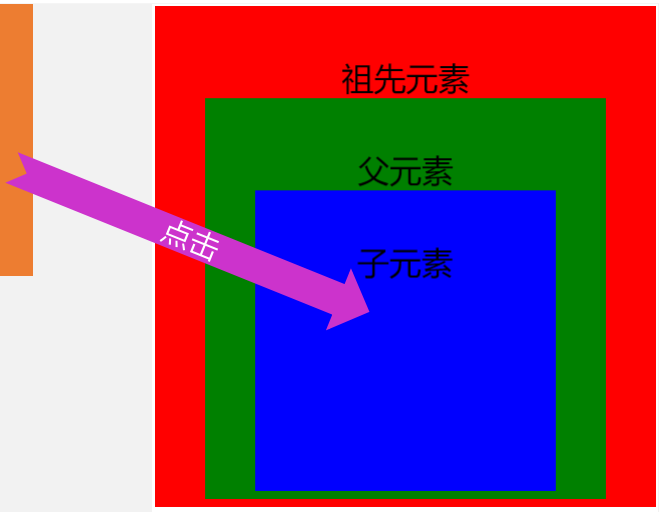

addEventListener的第三个参数指定在哪个阶段调用事件处理函数：
- false: 在冒泡阶段调用事件处理函数，默认值
- true: 在捕获阶段调用事件处理函数

# 2.5.5 事件流

```
<html>
  <head>
    <style> …… </style>
  </head>
  <body>
    <div id="ancestor">
      <span>祖先元素</span>
      <div id="parent">
        <span>父元素</span>
        <div id="child">
          <span>子元素</span>
        </div>
      </div>
    </div>
    <script type="text/javascript">
      ancestor = document.getElementById("ancestor");
      parent = document.getElementById("parent");
      child = document.getElementById("child");
      ancestor.addEventListener("click", function(e){alert("click-ancestor-cap");}, true);
      parent.addEventListener("click", function(e){alert("click-parent-cap");}, true);
      child.addEventListener("click", function(e){alert("click-child-cap");}, true);
      ancestor.addEventListener("click", function(e){alert("click-ancestor-bub");}, false);
      parent.addEventListener("click", function(e){alert("click-parent-bub");
                                                   e.stopPropagation(); }, false);
      child.addEventListener("click", function(e){alert("click-child-bub");}, false);
    </script>
  </body>
</html>
```

点击子元素时的输出：

click-ancestor-cap
click-parent-cap
click-child-cap
click-child-bub
click-parent-bub

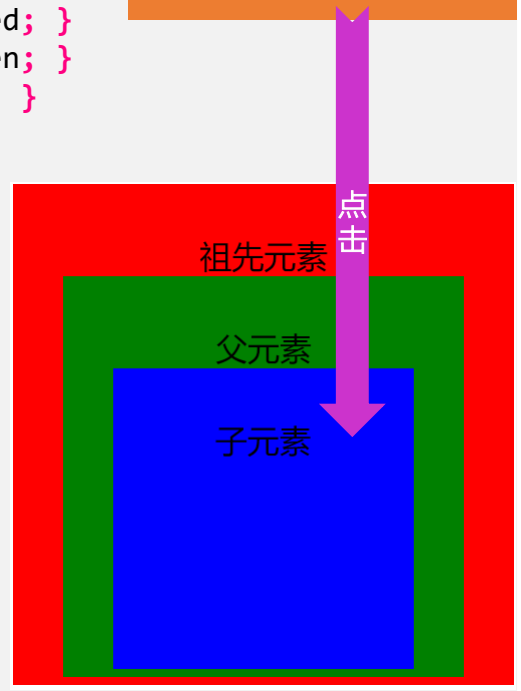



同一个对象可以在不同阶段注册不同的事件处理函数
使用stopPropagatoin可以终止事件的传播

# 2.5.5 事件流

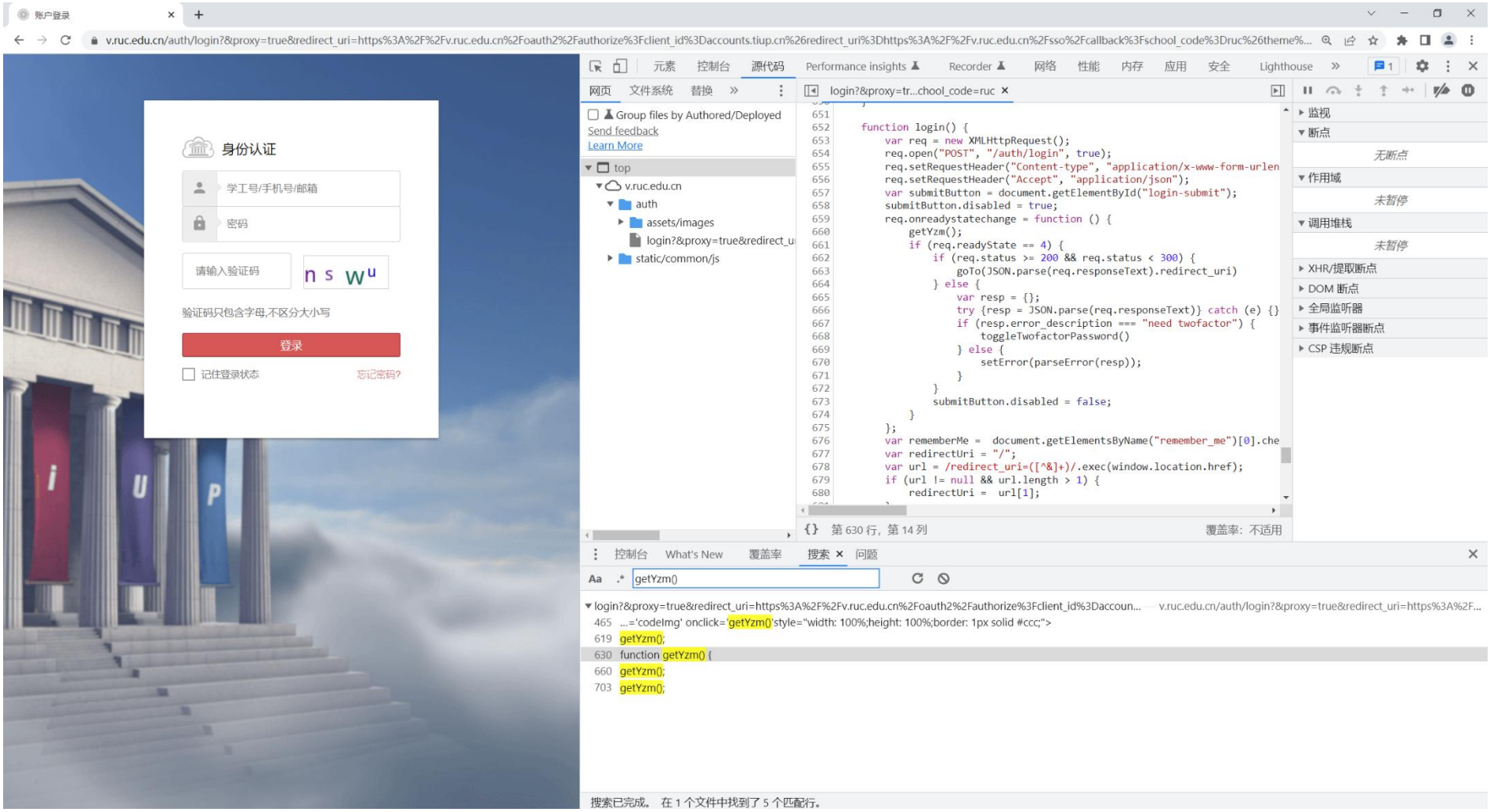
```
<html>
  <head>
    <style>
      div { padding:25px; border:2px; text-align:center; }
      #ancestor { width:200px; height:200px; background-color:red; }
      #parent { width:150px; height:150px; background-color:green; }
      #child { width:100px; height:100px; background-color:blue; }
    </style>
  </head>
  <body>
    <div id="ancestor"  onclick="alert('click-ancestor1')">
      <span>祖先元素</span>
      <div id="parent"  onclick="alert('click-parent1')">
        <span>父元素</span>
        <div id="child" onclick="alert('click-child1')">
          <span>子元素</span>
        </div>
      </div>
    </div>
    <script type="text/javascript">
      ancestor = document.getElementById("ancestor");
      parent = document.getElementById("parent");
      child = document.getElementById("child");
      ancestor.onclick = function(e){ alert("click-ancestor2");};
      parent.onclick = function(e){alert("click-parent2");};
      child.onclick = function(e){alert("click-child2");};
    </script>
  </body>
</html>
```

点击子元素时的输出：
click-child2
click-parent2
click-ancestor2




以行内绑定和动态绑定方式注册的事件处理函数，将在冒泡阶段被调用
同一个对象在同一个阶段注册多个事件处理函数，以最后一次注册为准

# 实例：为微人大认证页面添加前端交互



# 实例：为微人大认证页面添加前端交互

```
/* script.js */

function body_onload_handler() { //页面加载时触发
  button = document.getElementById("login-submit");
  button.onclick = login; //注册登录按钮的鼠标点击事件处理函数

  img = document.getElementById("captcha-img").childNodes[1];
  img.onclick = getYzm; //注册验证码图片的鼠标点击事件处理函数
  img.click();
}

function getYzm(e) { //随机生成一个0-9的整数x，使用图片code_x.png作为验证码图片
  x = Math.floor(Math.random()*10);
  img = e.target;
  img.src = "img/code_" + x + ".png";
}

function login() { //登录时检验验证码/用户名/密码
  img = document.getElementById("captcha-img").childNodes[1];
  form = document.getElementById("login-form");
  info = document.getElementById("login-alert");

  //维护一个验证码正确值和验证码图片下标的映射
  checks = ["pupr", "khrb", "huks", "egwb", "ekdv", "khns", "wuxc", "tcyk", "nupf", "brpk"];
  correct_code = checks[img.src.charAt(img.src.length-5)];

  username = form.username.value;
  password = form.password.value;
  code = form.code.value;

  if (code != correct_code) { info.innerText = "验证码不正确或已失效,请重试！"; }
  else if (username != "abcd" || password != "1234") { info.innerText = "用户名或密码不正确,请重试！"; }
  else { info.innerText = "验证成功！"; }
}
```