

# 前端基础

---

**提醒：**还没有申请到IDEA专业版本授权的同学要抓紧了，很快就需要用到。

经过前面基础内容的学习，现在我们就可以正式地进入Web开发的学习当中啦~

本章节会讲解前端基础内容（如果已经学习过，可以直接跳到下一个大章节了）那么什么是前端，什么又是后端呢？

- 前端：我们网站的页面，包括网站的样式、图片、视频等一切用户可见的内容都是前端的内容。
- 后端：处理网站的所有数据来源，比如我们之前从数据库中查询数据，而我们查询的数据经过处理最终会被展示到前端，而用于处理前端数据的工作就是由后端来完成的。

相当于，前端仅仅是一层皮，它直接决定了整个网站的美观程度，我们可以自由地编排页面的布局，甚至可以编写好看的特效；而灵魂则是后端，如何处理用户的交互、如何处理数据查询是后端的职责所在，我们前面学习的都是后端内容，而Java也是一门专注于后端开发的语言。

对于前端开发我们需要学习一些新的内容，只有了解了它们，我们才能编写出美观的页面。

本教程并不会过多地去讲解前端知识，我们只会提及一些必要的内容，我们主要学习的是JavaWeb，更倾向于后端开发，学习前端的目的是为了同学们了解前后端的交互方式，在进行后端开发时思路能够更加清晰，有关前端的完整内容学习，可以浏览其他前端知识教程。

我们在最开始讲解网络编程时，提到了浏览器访问服务器，实际上浏览器访问服务器就是一种B/S结构，而我们使用Java代码编写的客户端连接服务器就是一种C/S结构。

Web开发还要从HTML开始讲起，这个语言非常简单，很好学习，看完视频如果你觉得前端简单自己更喜欢一些，建议马上转前端吧，还来得及，工资还比后端高，不像后端那么枯燥乏味。

## HTML页面

---

我们前面学习了XML语言，它是一种标记语言，我们需要以成对标签的格式进行填写，但是它是专用于保存数据，而不是展示数据，而HTML恰恰相反，它专用于展示数据，由于我们前面已经学习过XML语言了，HTML语言和XML很相似，所以我们学习起来会很快。

### 第一个HTML页面

我们前面知道，通过浏览器可以直接浏览XML文件，而浏览器一般是用于浏览HTML文件的，以HTML语言编写的内容，会被浏览器识别为一个页面，并根据我们编写的内容，将对应的组件添加到浏览器窗口中。

我们一般使用Chrome、Safari、Microsoft Edge等浏览器进行测试，IE浏览器已经彻底淘汰了！

比如我们可以创建一个Html文件来看看浏览器会如何识别，使用IDEA也能编写HTML页面，我们在IDEA中新建一个web模块，进入之后我们发现，项目中没有任何内容，我们右键新建一个HTML文件，选择HTML5文件，并命名为index，创建后出现：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

</body>
</html>
```

我们发现，它和XML基本长得一样，并且还自带了一些标签，那么现在我们通过浏览器来浏览这个HTML文件（这里推荐使用内置预览，不然还得来回切换窗口）

我们发现现在什么东西都没有，但是在浏览器的标签位置显示了网页的名称为 `Title`，并且显示了一个IDEA的图标作为网页图标。

现在我们稍微进行一些修改：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>lbw的直播间</title>
</head>
<body>
  现在全体起立
</body>
</html>
```

再次打开浏览器，我们发现页面中出现了我们输入的文本内容，并且标题也改为了我们自定义的标题。

我们可以在设置->工具->Web浏览器和预览中将重新加载页面规则改为 `变更时`，这样我们使用内置浏览器或是外部浏览器，可以自动更新我们编写的内容。

我们还可以在页面中添加一个图片，随便将一张图片放到html文件的同级目录下，命名为 `image.xxx`，其中xxx是后缀名称，不要修改，我们在body节点中添加以下内容：

```

<!-- 注意xxx替换成对应的后缀名称 -->
```

我们发现，我们的页面中居然能够显示我们添加的图片内容。因此，我们只需要编写对应的标签，浏览器就能够自动识别为对应的组件，并将其展示到我们的浏览器窗口中。

我们再来看看插入一个B站的视频，很简单，只需要到对应的视频下方，找到分享，我们看到有一个嵌入代码：

```
<iframe src="//player.bilibili.com/player.html?
aid=333231998&bvid=BV1rA41lg7q8&cid=346917516&page=1" scrolling="no" border="0"
frameborder="no" framespacing="0" allowfullscreen="true" width="800"
height="500"> </iframe>
```

每一个页面都是通过这些标签来编写的，几乎所有的网站都是使用HTML编写页面。

# HTML语法规范

一个HTML文件中一般分为两个部分：

- 头部：一般包含页面的标题、页面的图标、还有页面的一些设置，也可以在这里导入css、js等内容。
- 主体：整个页面所有需要显示的内容全部在主体编写。

我们首先来看头部，我们之前使用的HTML文件中头部包含了这些内容：

```
<meta charset="UTF-8">
<title>lbw的直播间</title>
```


首先 `meta` 标签用于定义页面的一些元信息，这里使用它来定义了一个字符集（编码格式），一般是 UTF-8，下面的 `title` 标签就是页面的标题，会显示在浏览器的上方。我们现在来给页面设置一个图标，图标一般可以在字节跳动的IconPark网站找到：<https://iconpark.oceanengine.com/home>，选择一个自己喜欢的图标下载即可。

将图标放入到项目目录中，并命名为icon.png，在HTML头部添加以下内容：

```
<link rel="icon" href="icon.png" type="image/x-icon" />
```

`link` 标签用于关联当前HTML页面与其他资源的关系，关系通过 `rel` 属性指定，这里使用的是icon表示这个文件是当前页面图标。

现在访问此页面，我们发现页面的图标已经变成我们指定的图标样式了。

现在我们再来看主体，我们可以在主体内部编写该页面要展示的所有内容，比如我们之前就用到了 标签来展示一个图片，其中每一个标签都称为一个元素：

```

```

我们发现，这个标签只存在一个，并没有成对出现，HTML中有些标签是单标签，也就是说只有这一个，还有一些标签是双标签，必须成对出现，HTML中，也不允许交叉嵌套，但是出现交叉嵌套时，浏览器并不会提示错误，而是仍旧尝试去解析这些内容，甚至会帮助我们进行一定程度的修复，比如：

```
<body>

  <iframe src="//player.bilibili.com/player.html?
aid=333231998&bvid=BV1rA41lg7q8&cid=346917516&page=1" width="800" height="500"
    scrolling="no" border="0" frameborder="no" framespacing="0"
allowfullscreen="true">
</body>
</iframe>
```

很明显上面的代码已经出现交叉嵌套的情况了，但是依然能够在浏览器中正确地显示。

在主体中，我们一般使用div标签来分割页面：

```
<body>
  <div>我是第一块</div>
  <div>我是第二块</div>
</body>
```

通过使用 `div` 标签，我们将整个页面按行划分，而高度就是内部元素的高度，那么如果只希望按元素划分，也就是说元素占多大就划分多大的空间，那么我们就可以使用 `span` 标签来划分：

```
<body>
  <div>
    <span>我是第一块第一个部分</span>
    <span>我是第一块第二个部分</span>
  </div>
  <div>我是第二块</div>
</body>
```

我们也可以使用 `p` 段落标签，它一般用于文章分段：

```
<body>
  <p>
    你看这个彬彬啊，才喝几罐就醉了，真的太逊了。 这个彬彬就是逊呀！
    听你这么说，你很勇哦？ 开玩笑，我超勇的，超会喝的啦。
    超会喝，很勇嘛。身材不错哦，蛮结实的嘛。
  </p>
  <p>
    哎，杰哥，你干嘛啊。都几岁了，还那么害羞！我看你，完全是不懂哦！
    懂，懂什么啊？ 你想懂？我房里有一些好康的。
    好康，是玩游戏哦！ 什么新游戏，比游戏还刺激！
  </p>
  <p>
    杰哥，这是什么啊？ 哎呦，你脸红啦！来，让我看看。
    不要啦！！ 让我看看嘛。 不要啦，杰哥，你干嘛啊！
    让我看看你法语正不正常啊！
  </p>
</body>
```

那么如果遇到特殊字符该怎么办呢？和XML一样，我们可以使用转义字符：

最常用的字符实体			
Character Entities			
显示	说明	实体名称	实体编号
	半方大的空白	<code>&amp;ensp;</code>	<code>&amp;#8194;</code>
	全方大的空白	<code>&amp;emsp;</code>	<code>&amp;#8195;</code>
	不断行的空白	<code>&amp;nbsp;</code>	<code>&amp;#160;</code>
<	小于	<code>&amp;lt;</code>	<code>&amp;#60;</code>
>	大于	<code>&amp;gt;</code>	<code>&amp;#62;</code>
&	&符号	<code>&amp;amp;</code>	<code>&amp;#38;</code>
"	双引号	<code>&amp;quot;</code>	<code>&amp;#34;</code>
©	版权	<code>&amp;copy;</code>	<code>&amp;#169;</code>
®	已注册商标	<code>&amp;reg;</code>	<code>&amp;#174;</code>
™	商标（美国）	<code>™</code>	<code>&amp;#8482;</code>
×	乘号	<code>&amp;times;</code>	<code>&amp;#215;</code>
÷	除号	<code>&amp;divide;</code>	<code>&amp;#247;</code>

**注意：**多个连续的空格字符只能被识别为一个，如果需要连续多个必须使用转义字符，同时也不会识别换行，换行只会变成一个空格，需要换行必须使用 `br` 标签。

通过了解了HTML的一些基础语法，我们现在就知道一个页面大致是如何编写了。

## HTML常用标签

前面我们已经了解了HTML的基本语法规则，那么现在我们就来看看，有哪些常用的标签吧，首先是换行和分割线：

- br 换行
- hr 分割线

```
<body>
  <div>
    我是一段文字<br>我是第二段文字
  </div>
  <hr>
  <div>我是底部文字</div>
</body>
```

标题一般用h1到h6表示，我们来看看效果：

```
<body>
<h1>一级标题</h1>
<h2>二级标题</h2>
<h3>三级标题</h3>
<h4>四级标题</h4>
<h5>五级标题</h5>
<h6>六级标题</h6>
<p>我是正文内容，真不错。</p>
</body>
```

现在我们来看看超链接，我们可以添加一个链接用于指向其他网站：

```
<a href="https://www.bilibili.com">点击访问小破站</a>
```

我们也可以指定页面上的一个锚点进行滚动：

```
<body>
<a href="#test">跳转锚点</a>




<div id="test">我是锚点</div>



</body>
```

每个元素都可以有一个id属性，我们只需要给元素添加一个id属性，就使用a标签可以跳转到一个指定锚点。

我们接着来看看列表元素，这是一个无序列表，其中每一个li表示一个列表项：

```
<ul>
  <li>一号选项</li>
  <li>二号选项</li>
  <li>三号选项</li>
  <li>四号选项</li>
  <li>五号选项</li>
</ul>
```

我们也可以使用 `ol` 来显示一个有序列表：

```
<ol>
  <li>一号选项</li>
  <li>二号选项</li>
  <li>三号选项</li>
  <li>四号选项</li>
  <li>五号选项</li>
</ol>
```

表格也是很重要的一种元素，但是它编写起来相对有一点麻烦：

```
<table>
  <thead>
    <tr>
      <th>学号</th>
      <th>姓名</th>
      <th>性别</th>
      <th>年级</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>0001</td>
      <td>小明</td>
      <td>男</td>
      <td>2019</td>
    </tr>
    <tr>
      <td>0002</td>
      <td>小红</td>
      <td>女</td>
      <td>2020</td>
    </tr>
  </tbody>
</table>
```

虽然这样生成了一个表格，但是这个表格并没有分割线，并且格式也不符合我们想要的样式，那么如何才能修改这些基础属性的样式呢，我们就需要聊聊CSS了。

# HTML表单

表单就像其名字一样，用户在页面中填写了对应的内容，点击按钮就可以提交到后台，比如登陆界面，就可以使用表单来实现：

一个网页中最重要的当属输入框和按钮了，那么来看看如何创建一个输入框和按钮：

```
<label>
    我是输入框
    <input type="text">
</label>
```

对于一个输入框，我们一般会将其包括在一个 `label` 标签中，它和 `span` 效果一样，但是我们点击前面文字也能快速获取输入框焦点。

```
<body>
<div>登陆我们的网站</div>
<hr>
<div>
    <label>
        账号：
        <input type="text">
    </label>
</div>
<div>
    <label>
        密码：
        <input type="password">
    </label>
</div>
</body>
```

输入框可以有很多类型，我们来试试看 `password`，现在输入内容就不会直接展示原文了。

创建一个按钮有以下几种方式，在学习 `JavaWeb` 时，我们更推荐第二种方式，我们后面进行登陆操作需要配合表单使用：

```
<button>登陆</button>
<input type="submit" value="登陆">
<input type="button" value="登陆">
```

现在我们可以写一个大致的登陆页面了：

```
<body>
    <h1>登陆我们的网站</h1>
    <form>
        <div>
            <label>
                账号：
                <input type="text" placeholder="Username...">
            </label>
        </div>
        <div>
```

```

        <label>
            密码:
            <input type="password" placeholder="Password...">
        </label>
    </div>
    <br>
    <a href="https://www.baidu.com">忘记密码</a>
    <br>
    <br>
    <div>
        <input type="submit" value="登陆">
    </div>
</form>
</body>

```

表单一般使用 `form` 标签将其囊括，但是现在我们还用不到表单提交，因此之后我们再来讲解表单的提交。

`input` 只能实现单行文本，那么如何实现多行文本呢？

```

<label>
    这是我们的文本框<br>
    <textarea placeholder="文本内容..." cols="10" rows="10"></textarea>
</label>

```

我们还可以指定默认的行数和列数，拖动左下角可以自定义文本框的大小。

我们还可以在页面中添加勾选框：

```

<label>
    <input type="checkbox">
    我同意本网站的隐私政策
</label>

```

上面演示的是一个多选框，那么来看看单选框：

```

<label>
    <input type="radio" name="role">
    学生
</label>
<label>
    <input type="radio" name="role">
    教师
</label>

```

这里需要使用 `name` 属性进行分组，同一个组内的选项只能选择一个。

我们也可以添加列表让用户进行选择，创建一个下拉列表：



```
<label>
  登陆身份:
  <select>
    <option>学生</option>
    <option>教师</option>
  </select>
</label>
```

默认选取的是第一个选项，我们可以通过 `selected` 属性来决定默认使用的是哪个选项。

当然，HTML的元素远不止我们所提到的这些，有关更多HTML元素的内容，可以自行了解。

---

## CSS样式

之前我们编写的页面非常基础，我们只能通过一些很基本的属性来排列我们的页面元素，那么如何实现更高度的自定义呢，我们就需要用到CSS来自定义样式，首先我们创建一个名为 `style.css` 的文件。

首先在我们HTML文件的头部添加：

```
<link href="style.css" rel="stylesheet">
```

我们在CSS文件中添加以下内容：

```
body {
  text-align: center;
}
```

我们发现，网页的内容全部变为居中显示了，这正是css在生效，相当于我们现在给页面添加了自定义的样式规则。

当然，我们也可以选择不使用CSS，而是直接对某个元素添加样式：

```
<body style="text-align: center;">
  ...
```

这样的效果其实是等同于上面的css文件的，相当于我们直接把样式定义在指定元素上。

也可以在头部直接定义样式，而不是使用外部文件：

```
<style>
  body {
    text-align: center;
  }
</style>
```

使用以上三种方式都可以自定义页面的样式，我们推荐使用还是第一种，不然我们的代码会很繁杂。

样式的属性是非常多的，我们不可能一个一个全部讲完，视频中用到什么再来讲解什么，如果同学们感兴趣，可以自行下去了解。

## CSS选择器

我们首先来了解一下选择器，那么什么是选择器呢？我们想要自定义一个元素的样式，那么我们肯定要去选择某个元素，只有先找到要自定义的元素，我们才能开始编写样式。

我们上面的例子中使用的就是标签名选择器，它可以快速选择页面中所有指定的的标签，比如我们之前使用的就是 `body` 标签，那么就相当于页面中所有的 `body` 元素全都使用此样式，那么我们现在来试试看选择页面中所有的 `input` 标签：

```
input {  
  width: 200px;  
}
```

我们发现，页面中所有的 `input` 元素宽度全部被设定为了200个像素（`px` 是单位大小，代表像素，除了 `px` 还有 `em` 和 `rem`，他们是根据当前元素字体大小决定的相对大小，一般用于适配各种大小的浏览器窗口，这里暂时不用）

样式编写完成后，如果只有一个属性，可以不带；若多个属性则每个属性后面都需要添加一个；

因此，一个标签选择器的格式为：

```
标签名称 {  
  属性名称: 属性值  
}
```

我们还可以设定输入框的字体大小、行高等：

```
input {  
  width: 200px;  
  font-size: 20px;  
  line-height: 40px;  
}
```

我们现在可以通过选择器快速地去设置某个元素样式了，那么如何实现只设置某个元素的样式呢，现在让我们来看看，`id`选择器，我们之前已经讲解过了，每个元素都可以有一个`id`属性，我们可以将其当做一个跳转的锚点使用，而现在，我们可以使用`css`来进行定位：

我们先为元素添加`id`属性：

```
<h1 id="title">登陆我们的网站</h1>
```

现在使用`CSS`选择我们的元素，并设定一个属性，选择某个`id`需要在前面加上一个`#`：

```
#title {  
  color: red;  
}
```

虽然`id`选择器已经可以很方便的指定某个元素，但是如果我们希望`n`个但不是元素都被选择，`id`选择器就无法实现了，因为每个元素的`id`是唯一的，不允许出现重复`id`的元素，因此接着我们来讲解一下类选择器。

每个元素都可以有一个 `class` 属性，表示当前元素属于某个类（注意这里的类和我们java中的类概念完全不同）一个元素可以属于很多个类，一个类也可以被很多个元素使用：

```
<form>
  <div >
    <label class="test">
      账号：
      <input type="text" placeholder="Username...">
    </label>
  </div>
  <div>
    <label class="test">
      密码：
      <input type="password" placeholder="Password...">
    </label>
  </div>
</form>
```

上面的例子中，两个 `label` 元素都使用了 `test` 类（类名称是我们自定义的），现在我们在css文件中编写以下内容来以类进行选择：

```
.test{
  color: blue;
}
```

我们发现，两个标签的文本内容都变为了蓝色，因此使用类选择器，能够对所有为此类的元素添加样式。注意在进行类选择时，我们需要在类名前面加上 `.` 来表示。

## 组合选择器和优先级问题

我们也可以让多个选择器，共用一个css样式：

```
.test, #title {
  color: red;
}
```

只需要并排写即可，注意中间需要添加一个英文的逗号用于分割，我们也可以使用 `*` 来一次性选择所有的元素：

```
* {
  color: red;
}
```

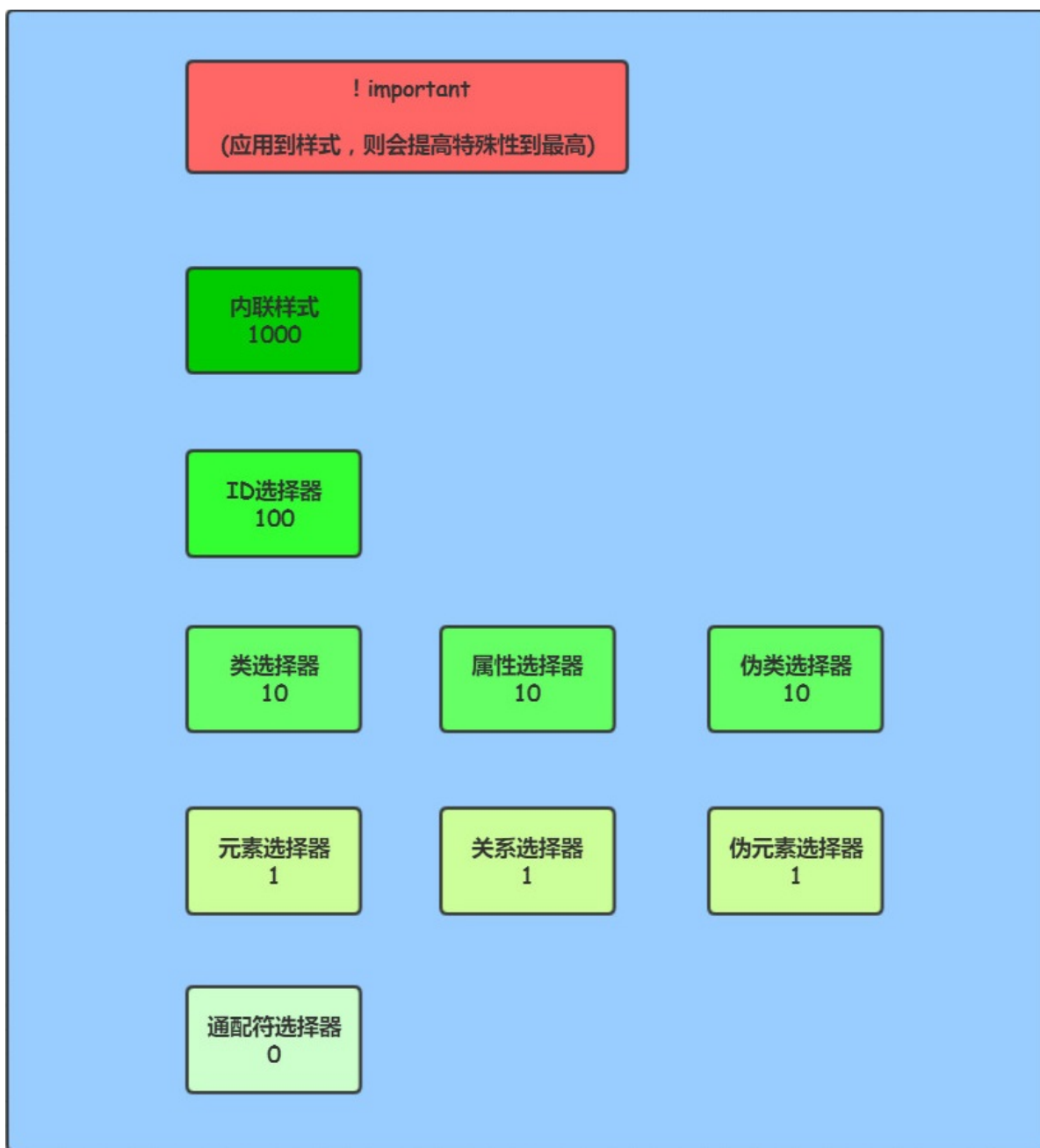
我们还可以选择位于某个元素内的某个元素：

```
div label {
  color: red;
}
```

这样的话，就会选择所有位于div元素中的label元素。

当然，我们这里只介绍了一些常用的选择器，有关详细的CSS选择器可以查阅：<https://www.runoob.com/cssref/css-selectors.html>

我们接着来看一下选择器的优先级：



我们根据上面的信息，来测试一下，首先编写一下HTML文件：

```
<body>  
  <div class="test" id="simple" style="color: blue">我是测试文本内容</div>  
</body>
```

现在我们来编写一下css文件：

```
.test {  
  color: yellow;  
}  
  
#simple {  
  color: red;  
}  
  
* {  
  color: palegreen;  
}
```

那么现在我们可以看到，实际上生效的是我们直接编写在标签内部的内联属性，那么现在我们依次进行移除，来看看它们的优先级。

那么如果我们希望某个属性无视任何的优先级，我们可以在属性后面添加 `!important` 标记，表示此属性是一个重要属性，它的优先级会被置为最高。

**思考：**那要是我每个选择器的这个属性后面都加一个 `!important` 会怎么样？

## 自定义边距

我们来看看，如何使用css控制一个div板块的样式，首先编写以下代码，相当于一个div嵌套了一个div元素：

```
<div id="outer">  
  <div id="inner">  
  
  </div>  
</div>
```

现在编写一下自定义的css样式，我们将div设定为固定大小，并且背景颜色添加为绿色：

```
#outer {  
  background: palegreen;  
  width: 300px;  
  height: 300px;  
}
```

我们发现左侧快速预览页面存在空隙，这是因为浏览器给我们添加了一个边距属性，我们只需要覆盖此属性并将其设定为0即可：

```
body {  
  margin: 0;  
}
```

现在我们给内部嵌套的div也设定一个大小，并将颜色设定为橙色：

```
#inner {  
  background: darkorange;  
  width: 100px;  
  height: 100px;  
}
```

现在我们发现内部的div元素位于右上角，我们还可以以百分比的形式来指定大小：

```
#inner {  
  background: darkorange;  
  width: 100%;  
  height: 100%;  
}
```

百分比会依照当前可用大小来进行分配，比如当前位于一个div内部，并且外部div元素是固定大小300px，因此100%就相当于使用了外部的全部大小，也是300px，现在内部元素完全将外部元素覆盖了，整个元素现在呈现为橙色。

我们可以为一个元素设定边距，边距分为外边距和内边距，外部元素内边距决定了内部元素与外部元素之间的间隔，我们来修改一下css样式：

```
#outer {  
  background: palegreen;  
  width: 300px;  
  height: 300px;  
  padding: 10px;  
}
```

我们发现，内部的div元素小了一圈，这是因为外部div元素设定了内边距，上下左右都被设定为10px大小。

而我们发现，实际上我们在一开始也是将body的外边距设定为了0，整个页面跟浏览器窗口直接间隔0px的宽度。

## 编写一个漂亮的登陆界面

现在我们就来尝试编写一个漂亮的登陆界面吧！

---

## JavaScript语言

也称为js，是我们整个前端基础的重点内容，只有了解了JavaScript语言，我们才能了解前端如何与后端交互。

JavaScript与Java没有毛关系，仅仅只是名字中包含了Java而已，跟Java比起来，它更像Python，它是一门解释型语言，不需要进行编译，它甚至可以直接在浏览器的命令窗口中运行。

它相当于前端静态页面的一个补充，它可以让一个普通的页面在后台执行一些程序，比如我们点击一个按钮，我们可能希望执行某些操作，比如下载文件、页面跳转、页面弹窗、进行登陆等，都可以使用JavaScript来帮助我们实现。

我们来看看一个简单的JavaScript程序：

```
const arr = [0, 2, 1, 5, 9, 3, 4, 6, 7, 8]

for (let i = 0; i < arr.length; i++) {
  for (let j = 0; j < arr.length - 1; j++) {
    if(arr[j] > arr[j+1]){
      const tmp = arr[j]
      arr[j] = arr[j+1]
      arr[j+1] = tmp
    }
  }
}

window.alert(arr)
```

这段代码实际上就是实现了一个冒泡排序算法，我们可以直接在页面的头部中引用此js文件，浏览器会在加载时自动执行js文件中编写的内容：

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

我们发现JS的语法和Java非常相似，但是它还是和Java存在一些不同之处，而且存在很多阴间语法，那么我们来看看JS的语法。

## JavaScript基本语法

在js中，定义变量和Java中有一些不同，定义一个变量可以使用 `let` 关键字或是 `var` 关键字，IDEA推荐我们使用 `let` 关键字，因为 `var` 存在一定的设计缺陷（这里就不做讲解了，之后一律使用let关键字进行变量声明）：

```
let a = 10;
a++;
window.alert(a)
```

上面的结果中，我们得到了a的结果是11，也就是说自增和自减运算在JS中也是支持的，并且JS每一句结尾可以不用加分号。

js并不是Java那样的强类型语言（任意变量的类型一定是明确的），它是一门弱类型语言，变量的类型并不会在一开始确定，因此我们在定义变量时无需指定变量的确切类型，而是在运行时动态解析类型：

```
let a = 10;
a = "HelloWorld! "
console.info(a)
```

我们发现，变量a已经被赋值为数字类型，但是我们依然在后续能将其赋值一个字符串，它的类型是随时可变的。

很多人说，这种变态的类型机制是JS的一大缺陷。

世界上只有两种语言：一种是很多人骂的，一种是没人用的。

我们接着来看看，JS中存在的基本数据类型：

- Number：数字类型（包括小数和整数）
- String：字符串类型（可以使用单引号或是双引号）
- Boolean：布尔类型（与Java一致）

还包括一些特殊值：

- undefined：未定义 - 变量声明但不赋值默认为undefined
- null：空值 - 等同于Java中的null
- NaN：非数字 - 值不是合法数字，比如：

```
window.alert(100/'xx')
```

我们可以使用 `typeof` 关键字来查看当前变量值的类型：

```
let a = 10;
console.info(typeof a)
a = 'Hello world'
console.info(typeof a)
```

## JavaScript逻辑运算和流程控制

我们接着来看看js中的关系运算符，包括如下8个关系运算符：大于（>）,小于（<）,小于等于（<=）,大于等于（>=）,相等（==），不等（!=），全等（===），不全等（!==）

其实关系运算符大致和Java中的使用方法一致，不过它还可以进行字符串比较，有点像C++的语法：

```
console.info(666 > 777)
console.info('aa' > 'ab')
```

那么，相等和全等有什么区别呢？

```
console.info('10' == 10)
console.info('10' === 10)
```

我们发现，在Java中，若运算符两边是不同的基本数据类型，会直接得到false，而JS中却不像这样，我们发现字符串的10居然等于数字10，而使用全等判断才是我们想要的结果。

`==` 的比较规则是：当操作数类型一样时，比较的规则和恒等运算符一样，都相等才相等，如果两个操作数是字符串，则进行字符串的比较，如果里面有一个操作数不是字符串，那两个操作数通过`Number()`方法进行转换，转成数字进行比较。

因此，我们上面进行的判断实际上是运算符两边都进行了数字转换的结果进行比较，自然也就得到了true，而全等判断才是我们在Java中认识的相等判断。

我们接着来看逻辑运算，JS中包括&&、||、&、|、?:等，我们先来看看位运算符：

```
console.info(4 & 7)
console.info(4 | 7)
```

实际上和Java中是一样的，那么我再来看看逻辑运算：

```
console.info(true || false)
```

对于boolean变量的判断，是与Java一致的，但是JS也可以使用非Boolean类型变量进行判断：



```
console.info(!0)
console.info(!1)
```

和C/C++语言一样，0代表false，非0代表true，那么字符串呢？

```
console.info(!"a")
console.info(!" ")
```

我们发现，空串为false，非空串为true，我们再来看看：

```
console.info(true || 7)
console.info(7 || true)
```

我们发现，前者得到的结果为true，而后者得到的结果却是7，真是滑天下之大稽，什么鬼玩意，实际上是因为，默认非0都是true，而后者又是先判断的7，因此会直接得到7而不是被转换为true

那么我们再来看看几个特殊值默认代表什么：

```
console.info(!undefined)
console.info(!null)
console.info(!NaN)
```

最后来使用一下三元运算符，实际上和java中是一样的：

```
let a = true ? "xx" : 20
console.info(a)
```

得益于JS的动态类型，emmm，三元运算符不一定需要固定的返回值类型。

JS的分支结构，实际上和java是一样的，也是使用if-else语句来进行：

```
if("lwnb"){ //非空串为true
  console.info("!!!")
} else {
  console.info("??")
}
```

同理，多分支语句也能实现：

```
if(""){
  console.info("!!!")
} else if(-666){
  console.info("??")
} else {
  console.info("0.0")
}
```

当然，多分支语句也可以使用switch来完成：

```
let a = "a"
switch (a){
  case "a":
```

```

        console.info("1")
        break
    case "b":
        console.info("2")
        break
    case "c":
        console.info("3")
        break
    default:
        console.info("4")
}

```

接着我们来看看循环结构，其实循环结构也和java相差不大：

```

let i = 10
while(i--){
    console.info("100")
}

```

```

for (let i = 0; i < 10; i++) {
    console.info("??")
}

```

## JavaScript函数定义

JS中的方法和java中的方法定义不太一样，JS中一般称其为函数，我们来看看定义一个函数的格式是什么：

```

function f() {
    console.info("有一个人前来买瓜")
}

```

定义一个函数，需要在前面加上 `function` 关键字表示这是一个函数，后面跟上函数名称和 `()`，其中可以包含参数，在 `{}` 中编写函数代码。我们只需要直接使用函数名+ `()` 就能调用函数：

```

f();

```

我们接着来看一下，如何给函数添加形式参数以及返回值：

```

function f(a) {
    console.info("得到的实参为: "+a)
    return 666
}

f("aa");

```

由于JS是动态类型，因此我们不必指明参数a的类型，同时也不必指明返回值的类型，一个函数可能返回不同类型的结果，因此直接编写return语句即可。同理，我们可以在调用函数时，不传参，那么默认会使用undefined：

```
function f(a) {  
    console.info("得到的实参为: "+a)  
    return 666  
}  
  
f();
```

那么如果我们希望不传参的时候使用我们自定义的默认值呢？

```
function f(a = "6666") {  
    console.info("得到的实参为: "+a)  
    return 666  
}  
  
f();
```

我们可以直接在形参后面指定默认值。

函数本身也是一种类型，他可以被变量接收，所有函数类型的变量，也可以直接被调用：

```
function f(a = "6666") {  
    console.info("得到的实参为: "+a)  
    return 666  
}  
  
let k = f;  
k();
```

我们也可以直接将匿名函数赋值给变量：

```
let f = function (str) {  
    console.info("实参为: "+str)  
}
```

既然函数是一种类型，那么函数也能作为一个参数进行传递：

```
function f(test) {  
    test();  
}  
  
f(function () {  
    console.info("这是一个匿名函数")  
})
```

对于所有的匿名函数，可以像Java的匿名接口实现一样编写lambda表达式：

```
function f(test) {
  test();
}

f(() => {
  console.info("可以，不跟你多bb")
})
```

```
function f(test) {
  test("这个是回调参数");
}

f(param => {
  console.info("接受到回调参数: "+param)
})
```

## JavaScript数组和对象

JS中的数组定义与Java不同，它更像是Python中的列表，数组中的每个元素并不需要时同样的类型：

```
let arr = [1, "lbwnb", false, undefined, NaN]
```

我们可以直接使用下标来访问：

```
let arr = [1, "lbwnb", false, undefined, NaN]
console.info(arr[1])
```

我们一开始编写的排序算法，也是使用了数组。

数组还可以动态扩容，如果我们尝试访问超出数组长度的元素，并不会出现错误，而是得到 undefined，同样的，我们也可以直接往超出数组长度的地方设置元素：

```
let arr = [1, "lbwnb", false, undefined, NaN]
arr[5] = "???"
console.info(arr)
```

也可以使用 push 和 pop 来实现栈操作：

```
let arr = [1, "lbwnb", false, undefined, NaN]
arr.push("bbb")
console.info(arr.pop())
console.info(arr)
```

数组还包括一些其他的方法，这里就不一一列出了：

```
let arr = [1, "lbwnb", false, undefined, NaN]
arr.fill(1)
console.info(arr.map(o => {
  return 'xxx'+o
}))
```

我们接着来看对象，JS中也能定义对象，但是这里的对象有点颠覆我们的认知：

```
let obj = new Object()
let obj = {}
```

以上两种写法都能够创建一个对象，但是更推荐使用下面的一种。

JS中的对象也是非常随意的，我们可以动态为其添加属性：

```
let obj = {}
obj.name = "伞兵一号"
console.info(obj)
```

同理，我们也可以给对象动态添加一个函数：

```
let obj = {}
obj.f = function () {
  console.info("我是对象内部的函数")
}

obj.f()
```

我们可以在函数内使用this关键字来指定对象内的属性：

```
let name = "我是外部变量"
let obj = {}
obj.name = "我是内部变量"
obj.f = function () {
  console.info("name属性为: "+this.name)
}

obj.f()
```

**注意：**如果使用lambda表达式，那么this并不会指向对象。

除了动态添加属性，我们也可以在一开始的时候指定对象内部的成员：

```
let obj = {
  name: "我是内部的变量",
  f: function () {
    console.info("name属性为: "+this.name)
  }
}

obj.f()
```

注意如果有多行属性，需要在属性定义后添加一个逗号，进行分割！

# JavaScript事件

当我们点击一个页面中的按钮之后，我们希望之后能够进行登陆操作，或是执行一些JS代码来实现某些功能，那么这个时候，就需要用到事件。

事件相当于一个通知，我们可以提前设定好事件发生时需要执行的内容，当事件发生时，就会执行我们预先设定好的JS代码。

事件有很多种类型，其中常用的有：

- onclick：点击事件
- oninput：内容输入事件
- onsubmit：内容提交事件

那么如何为事件添加一个动作呢？

```
<input type="password" oninput="console.info('正在输入文本')">
```

我们可以直接为一个元素添加对应事件的属性，比如 oninput 事件，我们可以直接在事件的值中编写js代码，但是注意，只能使用单引号，因为双引号用于囊括整个值。

我们也可以单独编写一个函数，当事件发生时直接调用我们的函数：

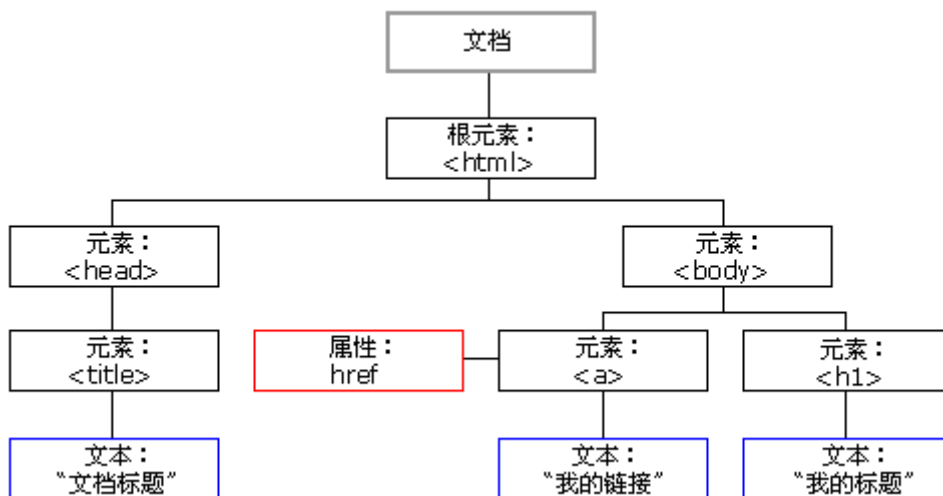
```
function f() {  
    window.alert("你输入了一个字符")  
}
```

```
<input type="password" oninput="oninput()">
```

仅仅了解了事件，还不足以实现高度自定义，我们接着来看DOM。

## Document对象

当网页被加载时，浏览器会创建页面的文档对象模型（Document Object Model），它将整个页面的所有元素全部映射为JS对象，这样我们就可以在JS中操纵页面中的元素。



比如我现在想要读取页面中某个输入框中的内容，那么我们就需要从DOM中获取此输入框元素的对象：

```
document.getElementById("pwd").value
```

通过document对象就能够快速获取当前页面中对应的元素，并且我们也可以快速获取元素中的一些属性。

比如现在我们可以结合事件，来进行密码长度的校验，密码长度小于6则不合法，不合法的密码，会让密码框边框变红，那么首先我们先来编写一个css样式：

```
.illegal-pwd{
  border: red 1px solid !important;
  box-shadow: 0 0 5px red;
}
```

接着我们来编写一下js代码，定义一个函数，此函数接受一个参数（元素本身的对象）检测输入的长度是否大于6，否则就将当前元素的class属性设定为css指定的class：

```
function checkIllegal(e) {
  if(e.value.length < 6) {
    e.setAttribute("class", "illegal-pwd")
  }else {
    e.removeAttribute("class")
  }
}
```

最后我们将此函数绑定到 `oninput` 事件即可，注意传入了一个this，这里的this代表的是输入框元素本身：

```
<input id="pwd" oninput="checkIllegal(this)" type="password">
```

现在我们在输入的时候，会自动检查密码是否合法。

既然oninput本身也是一个属性，那么实际上我们可以动态进行修改：

```
document.getElementById("pwd").oninput = () => console.info("???)
```

那么，我们前面提及的window对象又是什么东西呢？

实际上Window对象范围更加广阔，它甚至直接代表了整个窗口，当然也包含我们的Document对象，我们一般通过Window对象来弹出提示框之类的东西。

## 发送XHR请求

JS的大致内容我们已经全部学习完成了，那么如何使用JS与后端进行交互呢？

我们知道，如果我们需要提交表单，那么我们就需要将表单的信息全部发送给我们的服务器，那么，如何发送给服务器呢？

通过使用XMLHttpRequest对象，来向服务器发送一个HTTP请求，下面是一个最简单的请求格式：

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'https://www.baidu.com');
xhr.send();
```

上面的例子中，我们向服务器发起了一次网络请求，但是我们请求的是百度的服务器，并且此请求的方法为GET请求。

我们现在将其绑定到一个按钮上作为事件触发：

```
function http() {  
    let xhr = new XMLHttpRequest();  
    xhr.open('GET', 'https://www.baidu.com');  
    xhr.send();  
}
```

```
<input id="button" type="button" onclick="http()">
```

我们可以在网络中查看我们发起的HTTP请求并且查看请求的响应结果，比如上面的请求，会返回百度这个页面的全部HTML代码。

实际上，我们的浏览器在我们输入网址后，也会向对应网站的服务器发起一次HTTP的GET请求。

在浏览器得到页面响应后，会加载当前页面，如果当前页面还引用了其他资源文件，那么会继续向服务器发起请求，直到页面中所有的资源文件全部加载完成后，才会停止。