# 《Web应用基础》课程结业报告

网站主要写了5个页面，首页，电影名言，电影历史，电影评论，美图欣赏。

首页的上面是一个轮换图，自动的隔几秒换一个图片。下面是显示几个热门电影的信息。

电影名言是找了一些著名电影的名言，然后展现出来。

电影历史就是简单的讲解一下电影的由来。

电影评论是找了一些热门电影的网友的评论，，然后展现出来。

美图欣赏就是展示一些好看的电影场景或者海报。

# 首页

顶部导航栏:

导航栏位于<header>标签内的<ul class="menu-bar">中。

导航栏包含了五个链接，分别是"首页"、"电影名言"、"电影历史"、"电影评论"和"美图欣赏"。

链接使用<a>标签表示，并且每个链接都有一个data-id属性用于标识不同的页面。

首页内容:

主要内容区域位于<div class="main main-checked">中。

main-checked类用于突出显示当前所选页面。

首页内容包括一个轮播图和一些热门电影介绍。

轮播图使用<div class="sideshow">包裹，并在其中使用<img>标签展示多张图片。

图片的URL通过src属性指定。

轮播图下方有圆点和箭头用于切换图片。

热门电影介绍:

热门电影介绍位于<div class="items">中。

每个电影使用<div class="item">包裹，其中包括电影海报和介绍文字。

电影海报通过<img>标签展示，海报的URL通过src属性指定。

电影介绍文字位于<div class="introduction">中，包括电影标题和简要介绍。

样式部分：

代码的开头设置了一些全局样式，将所有元素的外边距和内边距都设置为0。

接下来定义了链接样式，包括字体大小、颜色、过渡效果和文本装饰。

鼠标悬停链接时的样式设置了光标为指针。

html 元素设置了平滑滚动的行为。

body 元素设置了背景颜色为棕色。

导航部分：

.menu-bar 类表示导航栏容器，通过 display: flex 实现水平布局。

链接样式通过 .menu-bar li 设置，包括宽度、高度、居中对齐和过渡效果。

当链接被选中时，通过添加 .nav-checked 类名改变样式，包括底部边框和颜色。

.main 类表示内容区块，默认设置为不显示，通过 .main-checked 类名控制显示。

.container 类表示顶部容器，设置了宽度和自动外边距。

.header 类表示顶部区域的背景颜色。

图片轮播部分：

.sideshow 类表示轮播图容器，通过 position: relative 设置相对定位和固定高度。

图片通过 .sideshow-img > img 选择器设置样式，包括光标为指针、绝对定位、居中对齐、过渡效果和透明度。

第一张图片默认为显示状态，通过 opacity: 1 设置。

圆点分页器通过 .pagination > div 选择器设置样式，包括宽度、高度、圆角、背景颜色和过渡效果。

第一个圆点默认为选中状态，通过添加 .pagination0 类名改变样式。

左右箭头按钮通过 .sideshow .btn 类设置样式，包括相对定位、宽度、高度、透明度和光标样式。

鼠标悬停按钮时的样式改变了透明度。

首页区域部分：

.content 类表示内容区域，设置了自动外边距。

图片通过 .content img 选择器设置样式，包括宽度和过渡效果。

.items 类表示项目容器，通过 display: flex 实现项目的水平布局，并设置了间距。

.item-img 类表示每个项目的图片容器，设置了固定宽度、高度、溢出隐藏、过渡效果和圆角。

鼠标悬停在图片上时使用 transform: scale(1.2) 实现放大效果。

.introduction 类表示项目的介绍文本，设置了字体大小、宽度和内边距。

导航栏切换：

通过 querySelector 方法获取具有 .menu-bar 类名的导航栏元素，并为其添加了一个点击事件监听器。

在点击事件处理函数中，首先判断被点击的元素是否是 <a> 标签，通过检查 ev.target.tagName 的值。

如果是 <a> 标签，则获取它的 data-id 属性值，该属性保存了当前导航项的索引。

使用 querySelector 方法获取当前选中的导航项，并移除其 nav-checked 类名。

使用 querySelector 方法获取当前点击的导航项，并添加 nav-checked 类名。

通过 querySelector 方法获取原来被选中的内容区块，并移除其 main-checked 类名。

使用 querySelector 方法获取新的内容区块，并添加 main-checked 类名。

图片轮播：

通过 querySelector 方法获取具有 .sideshow-img 类名的图片父容器，并获取其子元素集合。

通过 querySelector 方法获取具有 .pagination 类名的分页父容器，并获取其子元素集合。

初始化变量 i 为 0，用于表示当前显示的图片索引。

定义了一个 change 函数，用于切换图片。

在 change 函数中，首先将所有分页按钮的背景设置为白色。

然后将所有图片元素的透明度设置为 0。

将当前索引对应的图片元素的透明度设置为 1。

将当前索引对应的分页按钮的背景设置为蓝色。

使用 setInterval 方法每隔 3000ms 执行一次 change 函数，实现自动切换图片的效果。

当鼠标移入图片容器时，清除定时器停止自动播放。

当鼠标移出图片容器时，重新设置定时器以继续自动播放。

给右侧按钮添加点击事件监听器，点击时执行 change 函数以切换到下一张图片。

给左侧按钮添加点击事件监听器，点击时递减索引并执行 change 函数以切换到上一张图片。

# 电影名言

代码包含一个<div class="main">，表示主要内容区域。

主要内容区域内包含一个<div class="container famouss">，用于容纳电影名言的内容。

电影名言部分有四个模块，每个模块使用<div class="famous">包裹，其中包括电影海报和名言文字。

每个模块下包含两个子元素：

第一个子元素是包含电影海报的<div>，其中使用<img>标签展示了电影的海报图片。海报的URL通过src属性指定。

第二个子元素是包含电影名言的<div class="information">，其中包括电影标题和名言文字。标题使用<h2>标签表示，名言文字使用<p>标签表示。

每个模块之间使用<hr>标签进行分隔，表示水平线。

.famouss .famous表示选中每个电影名言模块的样式，具体样式属性有：

display: flex;使用弹性布局，将子元素排列为一行。

justify-content: space-around;设置子元素的对齐方式为环绕分布，子元素之间的间距会均匀分配。

margin: 10px 0;设置上下边距为10像素，左右边距为0。

.famouss .famous .famous-img表示选中电影海报图片的样式，具体样式属性有：

width: 300px;设置图片宽度为300像素。

cursor: pointer;当鼠标悬停在图片上时，鼠标样式变为手型。

.famouss .famous .information表示选中电影名言文字信息的样式，具体样式属性有：

padding: 10px;设置内边距为10像素，文字与边框之间有一定的空隙。

width: 500px;设置宽度为500像素。

background-color: #f2f2f288;设置背景颜色为浅灰色，透明度为0.53。

box-shadow: 5px 5px 3px #a6a6a688;添加一个阴影效果，阴影颜色为浅灰色。

.famouss .famous .information > p表示选中电影名言文字段落的样式，具体样式属性有：

margin: 5px 0;设置上下边距为5像素，左右边距为0。

font-size: 20px;设置字体大小为20像素。

# 电影历史

1. 页面结构：整个页面使用了 `<div>`、`<p>`、`<img>`、`<ol>` 等标签进行内容组织和排版。

2. 图片展示：使用 `<img>` 标签嵌入了几张电影相关的图片，用来视觉上补充和支持文字内容。

3. 文字内容：使用了多个 `<p>` 标签包裹了一系列关于电影的描述性文字，分别介绍了电影的起源、制作过程、艺术特点、分类以及广告等相关内容。

4. 列表展示：使用了 `<ol>` 和 `<li>` 标签定义了两个有序列表，分别列举了美国和英国电影广告的分类标记。

5. 结束语：最后部分提到了电影是一种表演艺术，介绍了3D电影和多个国际电影节的最高奖项。

.history .img：该选择器用于设置历史部分中的图片样式。height: auto; 表示图片高度自适应，margin-bottom: 20px; 表示图片下方留有20像素的外边距。

.history p：该选择器用于设置历史部分中的段落文本样式。font-size: x-large; 表示段落文本的字体大小为较大尺寸。

.history .img-container：该选择器用于设置历史部分中图片容器的样式。text-align: center; 表示图片容器内的内容居中对齐。

.history li：该选择器用于设置历史部分中有序列表项（li）的样式。font-size: large; 表示列表项文本的字体大小为较大尺寸，font-style: italic; 表示列表项文本以斜体显示。

# 电影评论

页面结构：整个页面的主要内容包裹在<div class="main">里面，其中电影评论部分包裹在<div class="container comments">中。

电影评论：每个电影评论都被包裹在一个<div class="comment">的<div>标签中，其中包含两个子<div>标签，一个用于显示评论者的头像，另一个用于显示电影信息和评论内容。

电影信息：每个电影信息部分包含一个标题<h2>，用于显示电影名称，以及一个水平线<hr>，用于分隔标题和评论内容。

评论内容：每个评论的具体内容都包含在一个<p>标签中，用于显示文字描述。

图片链接：每个电影图使用<img>标签来显示，其中src属性指定了图片链接。

.comments .comment：这个样式规则用于设置每个评论的外部容器。设置了display: flex;来让评论以弹性盒子布局的方式显示，flex-direction: row-reverse;使得布局的顺序从右到左排列，justify-content: space-around;用于在水平方向上均匀分布评论。

.comments .comment .comment-img：这个样式规则用于设置电影图。设置了width: 300px;来限制图的宽度为300像素，cursor: pointer;鼠标悬浮时显示为手型。

.comments .comment .information：这个样式规则用于设置评论信息的样式。设置了padding: 10px;来设置内边距为10像素，width: 500px;来限制评论信息的宽度为500像素，background-color: #f2f2f288;设置背景颜色为浅灰色，box-shadow: 5px 5px 3px #a6a6a688;添加了一个阴影效果。

.comments .comment .information > p：这个样式规则用于设置评论内容的样式。设置了margin: 5px 0;来设置上下边距为5像素，font-size: 20px;设置字体大小为20像素。

# 美图欣赏

使用div元素作为图片的容器，通过给div元素添加class="main"来设置其样式。

在div容器内部，使用class="pictures"的div元素来包裹所有的图片。

使用img元素来插入图片，每个img元素都有一个class="picture"的类名，并且通过src属性指定了图片的链接地址。

.pictures：这个样式规则用于设置图片容器的样式。设置了display: grid;以网格布局方式显示图片，font-size: xx-large;设置字体大小为相当大的尺寸，text-align: center;设置居中对齐，grid-template-columns: 1fr 1fr;设置网格的列数为两列，每列的宽度平均分配，grid-gap: 2rem;设置网格间距为2rem，margin: 1rem;设置外边距为1rem。

.pictures img：这个样式规则用于设置图片的样式。设置了width: 45rem;来设置图片的宽度为45rem。

# 遇到的问题

在写首页的轮换图的时候，遇到图片始终都只是显示最后那一张图片，前面的图片都被覆盖遮住了，一直都不能实现轮换的效果，也就是js始终都没有生效。通过查阅信息后发现原因是我将js的代码放在了body中靠前的位置。

将JavaScript脚本放在<body>标签的尾部的原因主要与网页加载和渲染的性能有关。

避免阻塞渲染：浏览器在解析HTML文档时，会从上到下逐行读取和执行。当浏览器遇到一个JavaScript文件时，它会停止渲染页面，先下载并执行这个脚本。这意味着如果你把JavaScript放在<head>标签中，或者在<body>标签的顶部，它可能会阻止HTML页面的渲染。这可能导致页面的可见部分出现延迟，从而影响用户体验。

确保DOM元素已加载：JavaScript代码通常会操作或修改页面上的DOM元素。如果脚本在DOM元素之前加载，当脚本试图访问这些元素时，它们可能还不存在。将脚本放在<body>标签的尾部可以确保在脚本执行时，DOM元素已经加载完毕，从而避免这类问题。尽管如此，在某些情况下，您可能需要在页面加载时立即运行一些JavaScript代码，例如检测用户代理或设置一些全局变量。在这种情况下，可以将这些脚本放在<head>标签内。然而，大多数情况下，为了获得更好的性能和避免潜在的DOM操作问题，建议将JavaScript脚本放在<body>标签的尾部。

js代码在载入完后，是立即执行的。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<script>

var item = document.getElementById("item");

console.log(item); //null

</script>

</head>

<body>

<div id="item">

我是html结构，我要展示到页面上！

</div>

</body>

</html>

通过这个简单的例子我们可以看到，js代码在加载完后，是立即执行的，执行完后，body才开始解析渲染，所以是找不到item，所以为null。

js代码执行时会阻塞页面渲染（由于GUI渲染线程和js引擎线程互斥。具体原理可以看这）

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<script>

while (true) {

console.log(1);

}

</script>

</head>

<body>

<div id="item">

我是html结构，我要展示到页面上！

</div>

</body>

</html>

测试结果，浏览器一直处于加载过程，一会就卡死了。html整个结构（包括html标签）都无法渲染出来。js的下载和执行会阻塞Dom树的构建。

所以，我们一般采用以下方法

将所有的script标签放到页面底部，也就是body闭合标签之前，这能确保在脚本执行前页面已经完成了DOM树渲染。