第5章

前端开发基础

在人工智能应用的开发中，前端是用户与系统交互的纽带，也是展示算法价值与创新成果的重要窗口。一个良好的前端设计能够将复杂的模型结果以直观、生动的方式呈现，增强用户体验和理解，促进技术的实际落地与推广。本章将带领读者系统学习前端开发的核心概念、常用技术栈以及与人工智能项目结合的实践方法，帮助大家掌握从界面设计到数据可视化的完整流程，为智能产品打造更加人性化、交互性强的前端界面奠定坚实基础。

5.1 前端开发概述

前端开发是构建用户友好、交互性强的应用程序不可或缺的一环。当前，人工智能研究多集中于算法优化和模型性能提升，而在实际应用场景中，如何让最终用户高效、直观地理解和使用 AI 结果同样重要。因此，除了熟悉后端开发技能外，了解和掌握前端开发技能，对人工智能创新应用开发同样重要。

5.1.1 前端开发的定义与作用

1. 狭义前端与广义前端

在技术发展与实践中，前端这一术语逐渐衍生出狭义与广义两种理解。狭义的前端，通常专指运行在浏览器端的用户界面(UI)开发，即用户直接看到、交互操作的部分。它主要包含以下三大技术核心：

- HTML(HyperText Markup Language)：用于定义网页内容的结构；
- CSS(Cascading Style Sheets)：负责网页的样式设计与布局；
- JavaScript：赋予网页动态行为，实现数据交互与逻辑控制。

这些技术通过渲染静态页面、处理用户事件、调用后端接口等方式，负责实现信息的展示与交互逻辑。典型场景包括网页表单、产品展示页、单页应用(SPA)中的组件式界面等。狭义前端更强调“可视化”与“交互体验”，关注的是如何让页面美观、流畅并易用。

随着技术不断演化，前端逐渐扩展到更大的范畴，形成了广义前端的概念。它不仅指浏览器端的可视界面，还包括：

- 移动端，例如基于 React Native、Flutter 构建的应用程序；
- 桌面端，例如基于 Electron、Tauri 的跨平台桌面应用；

- 运行在客户端的脚本，例如浏览器扩展、边缘计算的前端推理等；
- 可视化控制面板、嵌入式设备上的人机交互界面；
- 包含图形渲染、模型可视化、实时数据监控等高级前端任务。

在广义理解中，前端不仅仅负责“展示”，还可能直接参与部分业务逻辑和数据处理，如前端模型推理（TensorFlow.js、ONNX.js）、离线数据缓存、本地加密与验证等。尤其在人工智能应用中，前端甚至可实现部分轻量级 AI 功能，提供即时反馈和增强交互。

在本书中，“前端”主要指浏览器端的 UI 开发，但这部分技能对于学习移动端、桌面端开发同样具有助益。

2. 前端的技术演化

前端技术的发展史是一部互联网技术创新和用户体验不断进化的缩影。从最初的静态网页到如今的多端智能应用，前端经历了多个重要阶段，技术边界和功能定位持续扩展。

1）静态页面时代（20 世纪 90 年代初）

最早的网页是由纯 HTML 编写的静态文档，只能展示静态文本和简单图片，没有交互功能。这一阶段，前端的任务主要是“信息发布”，网页更像是电子公告板。静态页面时代的前端技术单一，仅使用 HTML，内容更新需要重新编辑 HTML 文件并重新发布，也就是说需要对页面进行人工替换。

2）动态交互初步阶段（1995—2005 年）

随着 JavaScript 的出现和 DOM（文档对象模型）的标准化，网页拥有了动态行为，开发者可以通过 JavaScript 操作 DOM 来实现动态修改页面内容、结构和样式，用户能够在页面中进行简单交互（如表单验证、动态菜单）。与此同时，CSS 的加入使得前端界面更美观、样式更丰富。

3）Web 2.0 与 Ajax 时代（2005—2010 年）

2005 年前后，Ajax 技术（Asynchronous JavaScript and XML）的兴起使网页可以在不刷新整页的情况下与服务器异步通信，带来了“单页应用”的雏形，提升了用户体验和响应速度。例如，Gmail、Google Maps 等应用实现了页面局部刷新和即时数据交互。同时，jQuery 等工具库进一步简化了 DOM 操作和浏览器兼容问题，前端从“静态展示”迈向“动态应用”。

4）现代化框架与组件化（2010 年至今）

自 2010 年起，前端迎来了以组件化和模块化为核心的革命。React、Angular、Vue 等现代框架先后出现，推动了前端从页面驱动向组件驱动的转变。开发者倾向于将页面拆解为独立、可复用的小单元（组件），提高可维护性；前端框架引入了 Redux、Vuex 等全局状态管理方案，支持更加复杂的交互逻辑；在移动端，通过虚拟 DOM 和路由控制，前端可实现接近原生应用程序的体验。

同时，随着项目规模的扩大，前端也开始朝“工程化”方向演进，出现了丰富的构建工具和自动化方案；Web 3.0 快速兴起，前端将越来越多地接入区块链、智能合约和去中心化身份，实现更安全、可信和用户主权的数据交互方式。

3. 前端在人工智能项目中的角色

前端在人工智能项目中具有多方面的作用。

首先，前端负责将复杂的算法输出以可视化、交互式的方式呈现，帮助用户快速理解 AI 推理结果。例如，在医学影像诊断中，前端可以将模型识别出的病灶区域用高亮方式标注；

在自然语言处理应用中，前端可以以对话或图表形式呈现摘要或情感分析结果。这种直观的呈现大大降低了 AI 技术的理解门槛，提升了信任度和使用体验。

其次，前端作为数据输入的重要入口，承担用户信息收集、交互反馈以及模型参数设置等功能。通过表单、滑块、拖曳组件等方式，前端可以引导用户完成更丰富的交互操作，从而让 AI 系统实现更个性化、更精准的服务。

最后，现代前端技术的发展，如实时渲染、响应式布局、跨平台适配等，为 AI 模型的实时推理结果提供了更好的展示载体。前端还能通过与后端服务的紧密配合，实现模型推理结果的快速刷新和动态交互，保证系统整体的流畅性和可靠性。

5.1.2　HTML 与 CSS：页面结构与样式

在前端开发中，HTML 与 CSS 是网页开发的两大基石，分别用于构建页面结构与视觉样式。

1. HTML：页面的骨架

HTML（超文本标记语言）用于描述网页的内容和结构，就像一栋楼的框架，决定了页面包含哪些元素、这些元素之间的关系。

下面是一段简单的 HTML 页面内容定义：


代码

```
<!DOCTYPE html>
<html>
  <head>
    <title>我的第一个网页</title>
  </head>
  <body>
    <h1>欢迎来到人工智能创新实践课堂！</h1>
    <p>这是一个演示基础结构的示例。</p>
  </body>
</html>
```

新建一个 HTML 格式的文本文件，将上面的内容粘贴到文件中，并用浏览器打开该文件，你将看到类似图 5-1 所示的页面。

图 5-1　一个简单的 HTML 页面

在上面的 HTML 代码中，<!DOCTYPE html>是文件类型声明，这里表示“该文件是一个 HTML5 文档”。<html>标签是整个页面的根元素，开始标签<html>和结束标签</html>之间的内容都用于定义页面结构。其中，<head>标签用于放置页面的元数据，如标题、样式表、脚本等；<body>标签则包含了所有页面实际显示的内容。<title>标签用于定义网页的标题，它将显示在浏览器的标题栏；<h1>标签表示“一级标题”；<p>标签则表示“文本内容”。

通过这个例子可以了解，HTML 文档是由很多标签（Tag）构成，每个标签都是网页中的一个元素。title、h1 等尖括号中的内容是标签的名称，用来标识元素的类型。大部分标

签分为开始标签和结束标签，结束标签比开始标签多了一个斜线，例如＜p＞是开始标签，而＜/p＞是结束标签。开始和结束标签之间的部分则是标签内容。标签可以嵌套，这种嵌套关系表示了网页元素间的包含关系，例如，＜h1＞和＜p＞标签被包含在＜body＞标签内，表示它们是 HTML 文档的主体组成部分。此外，标签还可以通过属性(Attribute)提供额外信息，例如：

```
<a href="https://example.com" target="_blank" title="示例网站">单击这里</a>
```

这里的＜a＞标签用于创建超链接，我们为其指定了 3 个属性：

- href 表示链接地址，也就是超链接指向的地址；
- target 表示链接的打开方式，_blank 是指在新窗口打开；
- title 表示鼠标在链接上悬停时显示的提示文本。

属性值通常用双引号包裹，也可以用单引号，但不建议省略引号。

表 5-1 中列出了常用的 HTML 标签，你可以将它们添加到 HTML 文档中，观察它们在页面中呈现的外观。

表 5-1　常用的 HTML 标签

标　签	用　途	示　例	备　注
＜h1＞～＜h6＞	标题	＜h1＞一级标题＜/h1＞	数字越大，标题层级越低，字号越小
＜p＞	段落	＜p＞这是一个段落＜/p＞	自动在前后换行
＜a＞	超链接	＜a href＝"https://example.com"＞单击这里＜/a＞	可加属性 target＝"_blank" 在新窗口打开
＜img＞	图片	＜img src＝"logo.png" alt＝"Logo" /＞	必须设置 alt 以提升可访问性
＜ul＞/＜ol＞	列表(无序/有序)	＜ul＞＜li＞项目＜/li＞＜/ul＞	列表项使用 ＜li＞
＜div＞	块级容器	＜div＞内容＜/div＞	常用于布局和包裹多个元素
＜span＞	行内容器	＜span＞文本＜/span＞	不会换行，常用于局部样式
＜input＞	表单输入框	＜input type＝"text" placeholder＝"请输入姓名" /＞	可根据 type 定义不同类型输入
＜button＞	按钮	＜button＞提交＜/button＞	可与表单或脚本结合
＜form＞	表单容器	＜form action＝"/submit" method＝"post"＞＜/form＞	用于收集和提交数据
＜table＞	表格	＜table＞＜tr＞＜td＞单元格＜/td＞＜/tr＞＜/table＞	表格需要 ＜tr＞、＜td＞
＜tr＞	表格行	＜tr＞＜td＞数据＜/td＞＜/tr＞	必须在 ＜table＞ 内
＜td＞	表格单元格	＜td＞内容＜/td＞	表示数据单元格
＜th＞	表头单元格	＜th＞标题＜/th＞	内容通常加粗、居中
＜br＞	换行	第一行＜br＞第二行	无结束标签
＜hr＞	水平分隔线	＜hr /＞	无结束标签

2. CSS：页面的样式和美化

CSS(层叠样式表)用于定义网页的外观，包括颜色、字体、布局和动画效果等。它可以让页面变得更具吸引力，更易于用户理解和操作。有多种方式可以引入 CSS。首先，我们可以直接使用标签的 style 属性引入，例如：

```
<p style="color: blue;">这是蓝色文字</p>
```

这里的 color：blue;就是样式表的内容，表示将标签的文字颜色设置为蓝色。我们也可以使用＜style＞标签集中引入样式表，称为**内部样式表**。例如，如果我们希望把所有＜p＞标签中的文字统一设置为蓝色，那么可以在 HTML 头部区域写入以下内容：

```
<style>
  p {
    color: blue;
  }
</style>
```

多数情况下，我们更希望把 HTML 的页面结构定义与 CSS 样式定义分开，以方便管理，此时可以使用**外部样式表**。例如，我们可以单独创建一个 CSS 样式表文件 styles.css，并写入以下内容：

```
body {
  background-color: #f5f5f5;
}
p {
  font-size: 16px;
  line-height: 1.6;
}
```

接下来，在 HTML 文件中引入该样式表：

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

这样，HTML 页面的背景将被设置为白烟色，＜p＞标签中的文字将被设置为 16px，行高将被设置为字体大小的 1.6 倍。

3. 选择器与属性

CSS 通过**选择器**来指定网页中哪些元素应用特定的样式规则。选择器可以精确地定位到某一个或某一类 HTML 元素，使开发者能够灵活地控制页面的视觉表现与交互行为。选择器的类型主要包括基础选择器、组合选择器、伪类与伪元素选择器。

1）基础选择器

基础选择器直接通过标签名、类名(以.开头)、ID(以＃开头)等定位元素。例如：

```
p { color: black; }                    /* 所有段落标签 */
.highlight { font-weight: bold; }      /* 类名为 highlight 的元素 */
#main-title { font-size: 24px; }       /* ID 为 main-title 的元素 */
```

2）组合选择器

组合选择器通过组合多种条件，精确匹配嵌套结构或相邻元素。例如：

```
div p { line-height: 1.5; }            /* 所有位于 div 内的 p 元素 */
h1 + p { margin-top: 0; }              /* 紧跟在 h1 后的第一个 p 元素 */
```

3）伪类与伪元素选择器

伪类与伪元素选择器用于选择元素的特定状态或部分。例如：

```
a:hover { color: red; }                    /* 鼠标悬停时的链接 */
p::first-line { font-style: italic; }      /* 段落的首行 */
```

在样式定义中，可以为元素设置各种属性(Property)，每个属性都控制某个视觉或结构方面的表现，例如颜色(Color)、边距(Margin)、字体(Font-family)等。属性值的设定可以是固定值(如 16px)，也可以是相对值(如 em、%)或函数(如 calc())。例如：

```
.container {
  width: 80%;                          /* 设置宽度 */
  background-color: #f9f9f9;           /* 设置背景颜色 */
  padding: 20px;                       /* 设置边距 */
  border-radius: 8px;                  /* 设置圆角 */
}
```

4. 布局与响应式设计

网页布局决定了内容的结构与展示形式，是前端开发中的核心内容之一。随着设备种类和屏幕尺寸的多样化，现代网页布局不仅要追求视觉上的美观与逻辑上的清晰，还必须具备良好的响应能力，适配不同终端，如桌面浏览器、平板计算机和智能手机等。

1）传统布局方式

早期的网页布局多采用浮动方式、定位技术，或基于表格(<table>)的结构模拟复杂页面，但这些方法存在灵活性不足、维护成本高等问题，已逐渐被现代布局模型所取代。

2）弹性盒布局(Flexbox)

Flexbox(Flexible Box)是一种一维布局模型，适合用于处理横向或纵向的元素对齐与分布问题。它的核心思想是将容器设为弹性布局模型，并通过其子元素的“可伸缩性”来自动调整排布方式。使用时，只需将容器的 display 属性设置为 flex 或 inline-flex。


代码

例如，我们首先创建一个 CSS 文件 styles.css：

```
/* 设置容器为弹性布局，启用 Flexbox */
.card-container {
  display: flex;
  flex-wrap: wrap;                     /* 允许内容换行，便于响应式 */
  justify-content: space-between;      /* 卡片之间留出空隙 */
  padding: 20px;
  gap: 20px;                           /* 控制卡片间距 */
}

/* 卡片样式 */
.card {
  flex: 1 1 calc(33.333% - 40px);      /* 三列布局，预留边距 */
  background-color: #f2f2f2;
  border-radius: 10px;
  padding: 16px;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
  min-width: 260px;                    /* 防止过小 */
  box-sizing: border-box;
  transition: transform 0.3s ease;
}
```

接下来，创建 HTML 文件 flexbox.html：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Flexbox 卡片布局示例</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="card-container">
    <div class="card">
      <h2>卡片一</h2>
      <p>这是第一张卡片的内容。</p>
    </div>
    <div class="card">
      <h2>卡片二</h2>
      <p>这是第二张卡片的内容。</p>
    </div>
    <div class="card">
      <h2>卡片三</h2>
      <p>这是第三张卡片的内容。</p>
    </div>
  </div>
</body>
</html>
```

这样一来，我们就实现了一个简单的弹性盒布局页面，该页面能够根据宽度自动调整各个卡片的位置。图 5-2 展示了页面在不同宽度下的显示效果：当页面足够宽时，三个卡片横向排列；而当页面宽度不足时，三个卡片则自动调整为纵向排列。

图 5-2　弹性盒布局页面在不同宽度下的显示效果

3）网格布局（Grid）

Grid 布局是一种强大的二维布局系统，允许我们在水平和垂直两个维度上同时组织页面内容。以下代码展示了一个典型的三行三列的网格布局结构，适用于仪表盘、信息面板等场景。作为案例，我们首先创建 CSS 样式表文件 grid.css 和 HTML 页面文件 grid.html，前者写入以下样式配置信息：


代码

```
.container {
  display: grid;                                  /* 启用 Grid 布局 */
  grid-template-columns: repeat(3, 1fr);          /* 定义三列，等宽 */
  grid-template-rows: auto auto 1fr;              /* 三行，前两行内容自适应 */
  gap: 20px;                                      /* 网格单元之间的间距 */
  padding: 20px;
  background-color: #f5f5f5;
}

.item {
  background-color: #4caf50;
  color: white;
  padding: 20px;
  font-size: 1.2em;
  text-align: center;
  border-radius: 8px;
}

.header {
  grid-column: 1 / 4;                             /* 跨越全部三列 */
  background-color: #2196f3;
}

.sidebar {
  grid-row: 2 / 4;                                /* 占据第 2、3 行 */
  background-color: #ff9800;
}

.content {
  grid-column: 2 / 4;                             /* 占据右侧两列 */
}

.footer {
  grid-column: 2 / 4;                             /* 底部栏也占据右侧两列 */
  background-color: #9c27b0;
}
```

后者写入以下内容：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
  <title>Grid 布局示例</title>
  <link rel="stylesheet" href="grid.css">
</head>
<body>
  <div class="container">
    <div class="item header">头部 Header(跨三列)</div>
    <div class="item sidebar">侧边栏 Sidebar(跨两行)</div>
    <div class="item content">主内容区 Content</div>
    <div class="item footer">底部 Footer</div>
  </div>
</body>
</html>
```

用浏览器打开 grid.html 文件，你将看到如图 5-3 所示的页面布局。

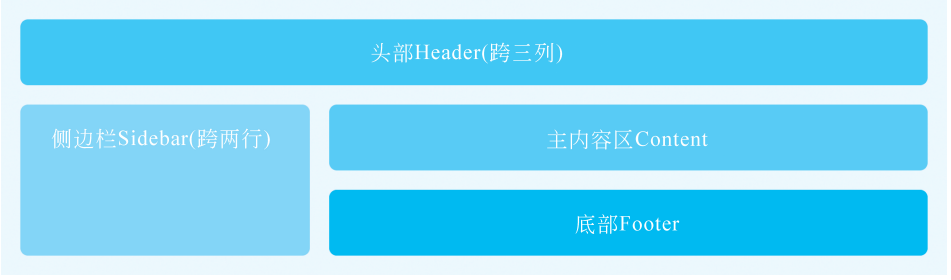

图 5-3　网格布局的显示效果

4）响应式设计与媒体查询

随着移动设备的普及，用户可能通过智能手机、平板计算机、笔记本计算机或大屏显示器访问网页。为了提供一致且良好的用户体验，**响应式设计**(Responsive Design)应运而生。响应式设计的核心思想是根据设备的屏幕尺寸、分辨率等特征，自动调整网页的布局和样式。

CSS 的媒体查询(Media Query)是实现响应式设计的主要手段。它允许开发者为不同的屏幕尺寸编写不同的样式规则，基本语法是：

```
@media 条件 {
  /* 针对满足条件的设备应用的样式 */
}
```

条件通常应指定媒体特性，常见的媒体特性(Media Features)包括最大宽度(Max-width)、最小宽度(Min-width)、屏幕方向(Orientation)、分辨率(Resolution)等。

为简化响应式开发，业界也出现了许多成熟的 CSS 框架，如 Bootstrap、Tailwind CSS、Foundation 等，它们封装了大量的响应式组件和网格系统，开发者只需使用类名即可快速实现适配布局。

5.1.3　JavaScript：实现动态页面

在前端开发中，JavaScript 用于为网页提供动态交互与逻辑控制能力。本节将介绍 JavaScript 的基础用法，并展示其在网页交互、事件处理、DOM 操作以及浏览器行为控制等方面的强大能力，帮助你迈出构建真正“活”的网页的第一步。

1. 基本语法

JavaScript 是一门以事件驱动、动态类型、解释执行为特点的脚本语言，其语法规则与其他编程语言类似，主要语法规则包括变量定义、运算符、流程控制、函数定义等。

1）变量定义

在 JavaScript 中，let 和 const 是 ES6（ECMAScript 2015）引入的两种变量声明方式，它们相较于传统的 var 声明方式提供了更清晰的作用域和行为规则。使用 let 和 const 声明的变量只在其所在的代码块（{}）内有效，let 声明的变量可以被重新赋值，而 const 声明的变量不能被重新赋值。

JavaScript 的数据类型可分为基本数据类型和引用数据类型。基本数据类型包括数值（Number）、字符串（String）、布尔值（Boolean），还支持未定义类型（Undefined）、空值（Null）、唯一值（Symbol）等；引用数据类型则包括对象（Object）、数组（Array）、函数（Function）、日期（Date）、正则表达式（RegExp）等。此外，需要注意的是，在 JavaScript 中变量是动态类型的，声明时不需要显式指定类型，变量的类型会根据赋值的内容动态确定。以下是定义常见数据类型变量的示例。

```
let name = "Alice";                              // 字符串
let isStudent = true;                            // 布尔型
const age = 25;                                  // 数值型
let person = {                                   // 对象
  name: "Bob",
  age: 30,
  isEmployed: true
};
let colors = ["red", "green", "blue"];           // 数组
```

2）运算符

JavaScript 提供了丰富的运算符，包括算术运算符、赋值运算符、比较运算符、逻辑运算符等，与 Java、Python 等常见编程语言基本一致。需要特别关注的是 JavaScript 中的相等运算符，当使用==（相等）运算符时，运算符两端会先进行类型转换再比较，因此'5' == 5 的运算结果为 true；如果需要同时进行值和类型的严格相等判定，需使用===（全等）运算符，例如'5' === 5 的运算结果为 false。类似地，不等比较运算符也分为!=和!==两种，两者的区别参照相等和全等运算符。

3）流程控制

JavaScript 中的流程控制语句决定代码执行的顺序和逻辑。当需要进行条件判断时，可使用 if/else 结构，例如：

```
let score = 85;
if (score >= 90) {
    console.log("优秀");          // 在控制台输出成绩等级
} else if (score >= 60) {
    console.log("及格");
} else {
    console.log("不及格");
}
```