动手实验：使用 Webpack 构建和部署网站

预计所需时间： 30 分钟

在这个实验中，您将学习如何使用 Webpack 打包您的 web 应用程序。

该实验是水果市场的延续。

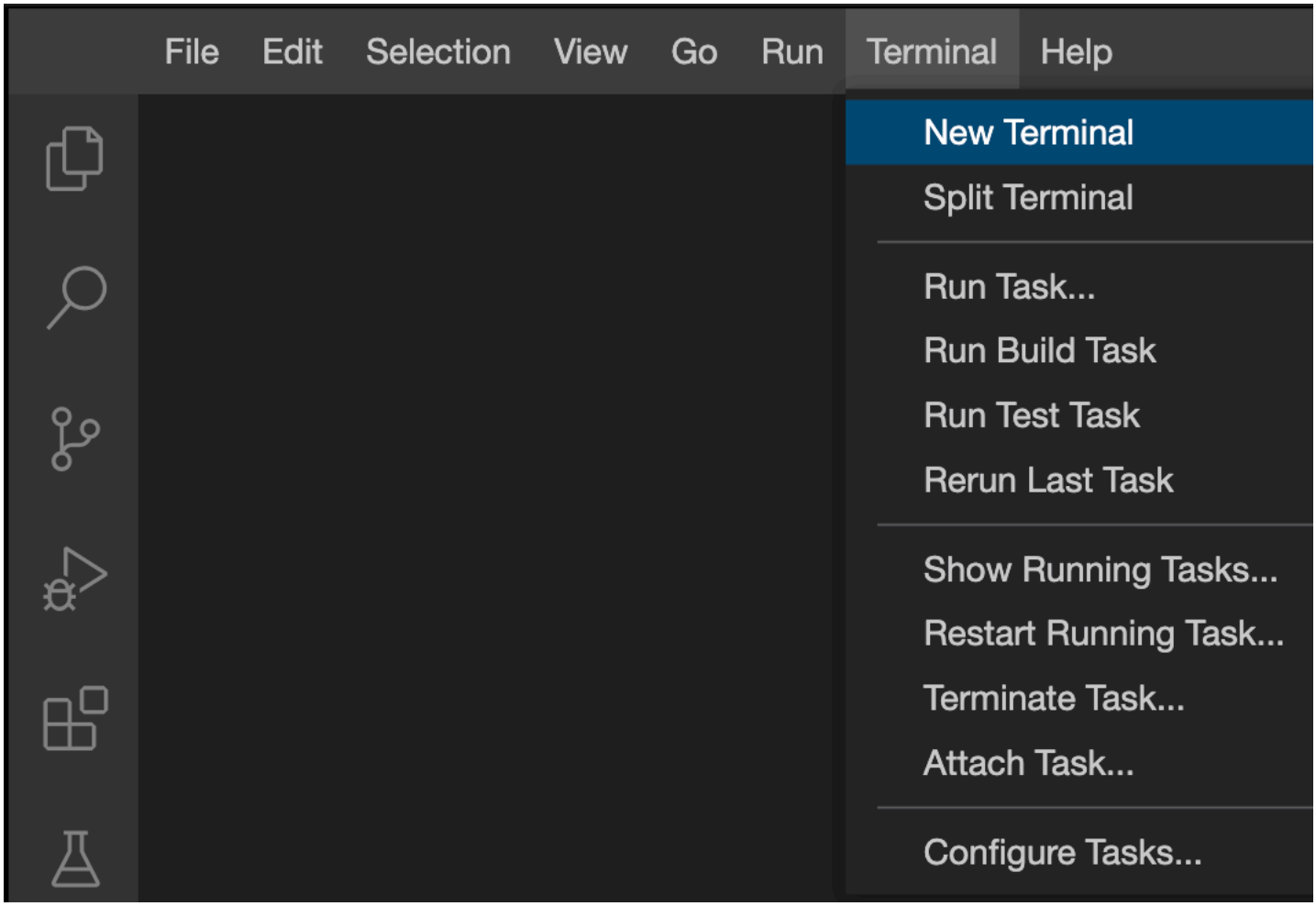
目标：

完成此实验后，您将能够：

- 使用 Webpack 打包图像和 CSS 文件。

设置：创建应用程序

1. 通过编辑器中的菜单打开终端窗口：**终端 > 新建终端**。



2. 如果您当前不在项目文件夹中，请复制并粘贴以下代码以切换到您的项目文件夹。

   ```
   cd /home/project
   ```

3. 运行以下命令以克隆包含本实验所需起始代码的 Git 仓库，如果 Git 仓库尚不存在。

   ```
   [ ! -d 'ghwey-build-deploy-webpack' ] && git clone https://github.com/ibm-developer-skills-network/ghwey-build-deploy-webpack.git
   ```

4. 切换到目录 **ghwey-build-deploy-webpack** 开始进行实验。

   ```
   cd ghwey-build-deploy-webpack
   ```

5. 列出此目录的内容以查看本实验的工件。

   ```
   ls
   ```

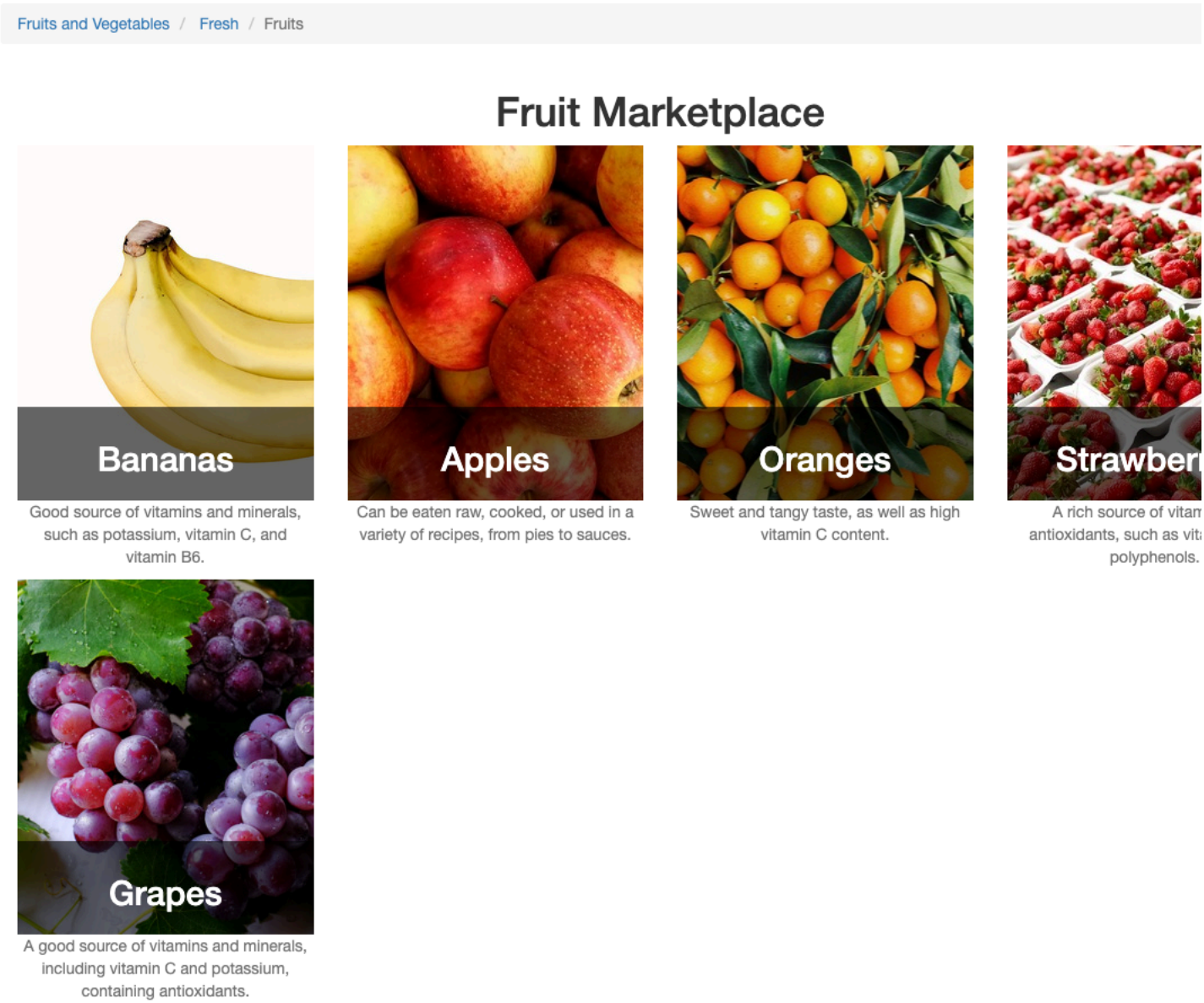
6. 在终端运行以下命令以托管您的网页。

```
python3 -m http.server
```

7. 要在浏览器中测试您的应用程序，请先运行应用程序。

Launch Application

8. 它看起来像这样：

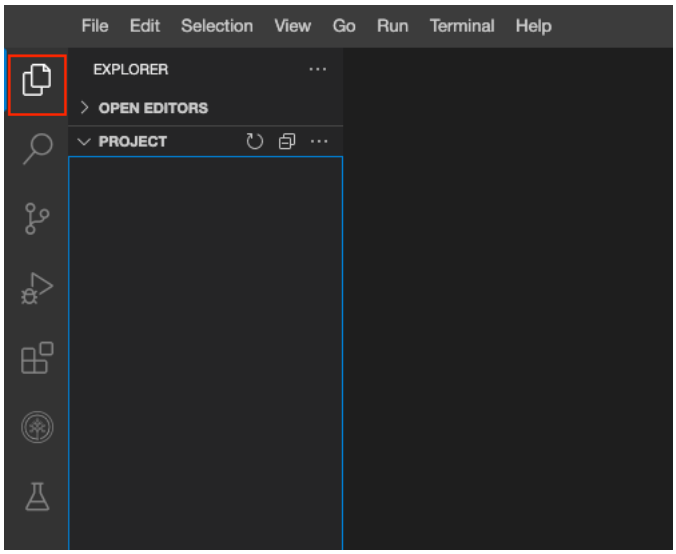


9. 您可以复制URL并在自己的浏览器中运行。

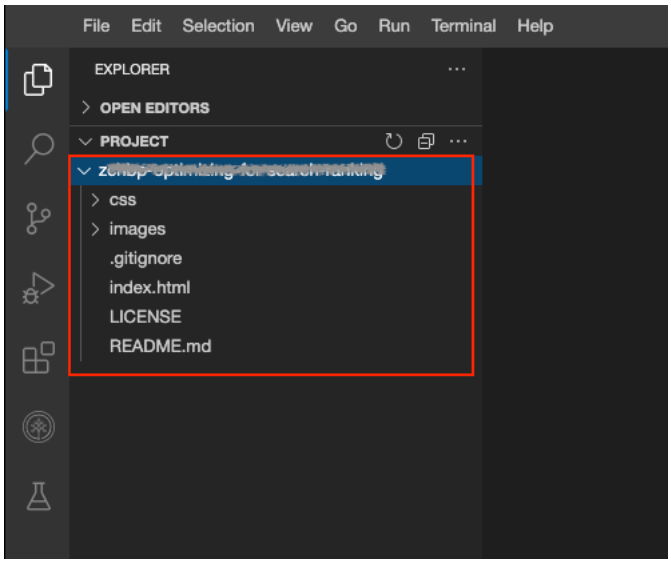
在 Cloud IDE 中处理文件

如果您是 Cloud IDE 的新手，本节将向您展示如何在 Cloud IDE 中创建和编辑项目中的文件。

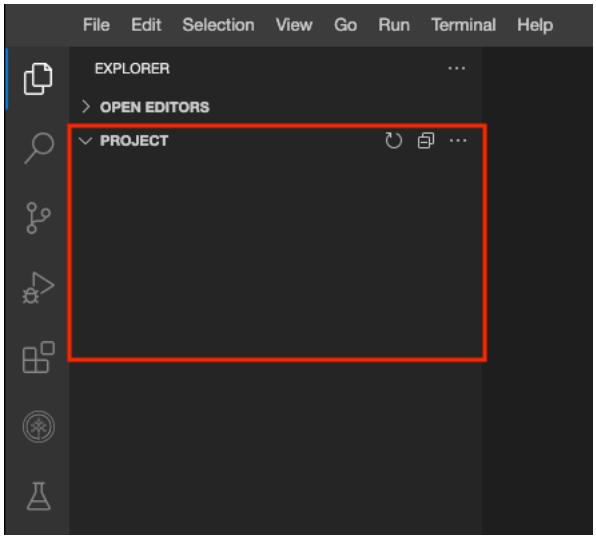
要查看 Cloud IDE 中的文件和目录，请单击此文件图标以显示它。



如果您已克隆（使用 `git clone` 命令）模板/起始代码，则它看起来如下：

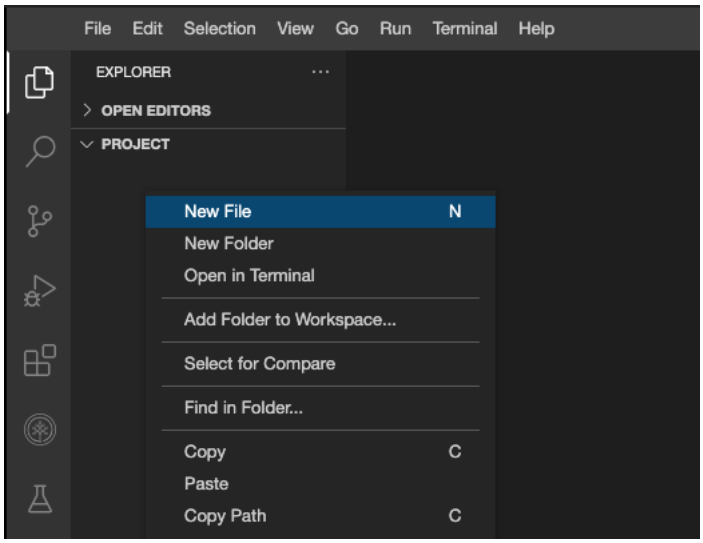


否则，空项目看起来是这样的：



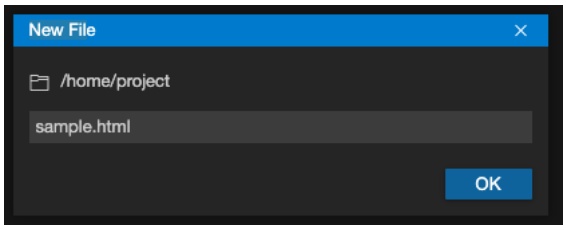
创建新文件

您可以右键单击并选择“新建文件”选项在项目中创建文件。

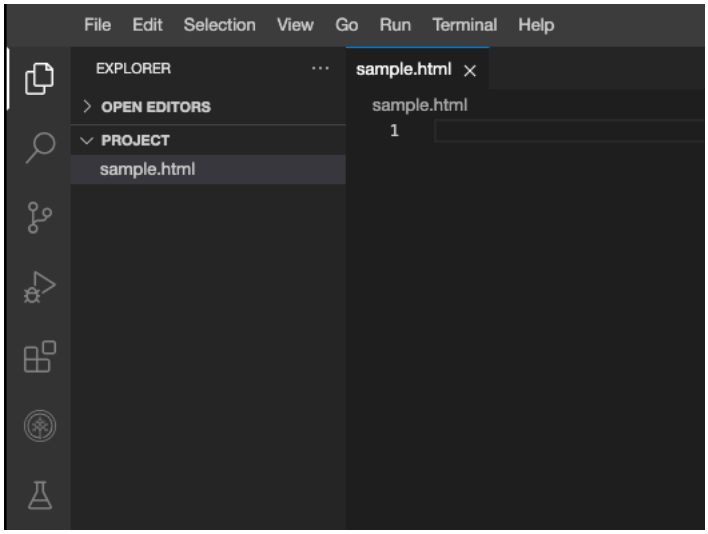


您也可以选择 文件 -> 新建文件 来执行相同操作。

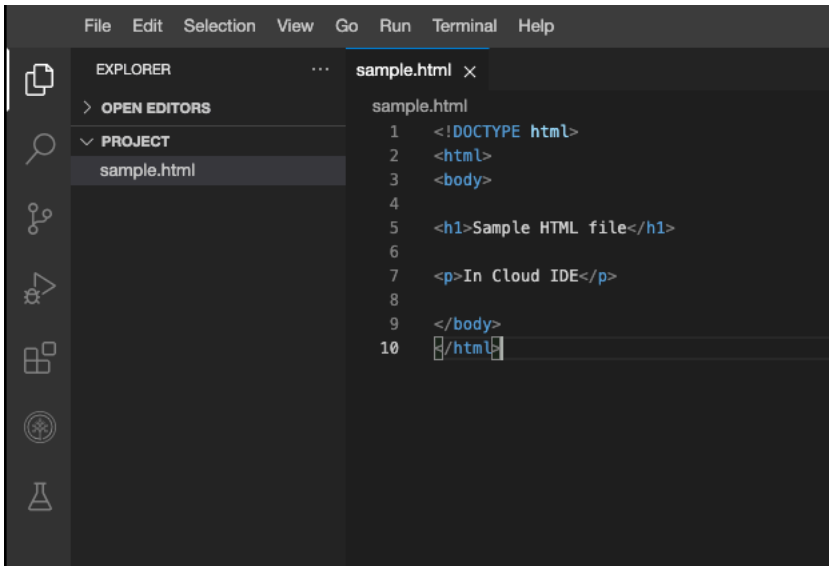
然后，它会提示您输入新文件的名称。在下面的示例中，我们正在创建 `sample.html`。



单击目录结构中的文件名 `sample.html` 将在右侧窗格中打开该文件。您可以创建所有不同类型的文件；例如，JavaScript 文件的名称为 `FILE_NAME.js`。

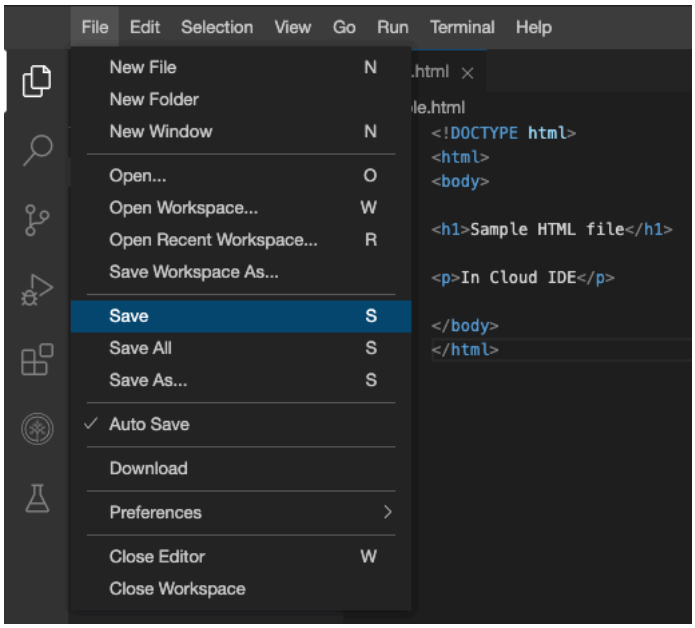


在示例中，我们只是粘贴了一些基本的 HTML 代码，然后保存了文件。



并通过以下方式保存它：

- 通过菜单进行操作。
- 在 Mac 上按 ⌘ Cmd + S 或在 Windows 上按 CTRL + S。
- 或者，它也可以为您自动保存。

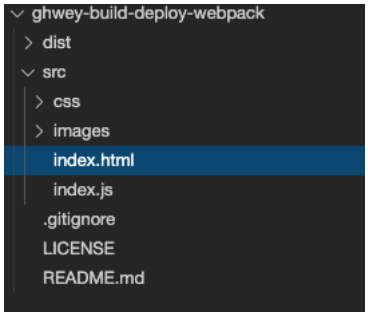


项目结构

我们重新构建了 HTML 项目，并保留了 src（源文件）和 dist（输出）目录。

和之前一样，src 目录下有 images 和 css 目录用于存放我们的资源。

dist 目录目前是空的。



练习 1：添加依赖

我们首先安装所需的包，以帮助我们实现 Webpack。在终端中运行以下命令。

```
npm install webpack webpack-cli html-webpack-plugin --save-dev
```

Where:

- `webpack` 是一个模块打包器。它的主要目的是将 JavaScript 文件打包以供浏览器使用，但它也能够转换、打包或打包几乎任何资源或资产。
- `webpack-cli` 为开发者提供了一套灵活的命令，以提高设置自定义 Webpack 项目的速度。
- `html-webpack-plugin` 简化了创建 HTML 文件以服务于 Webpack 打包的过程。

您还会注意到在您的目录结构中创建了两个新文件 `package.json` 和 `package-lock.json`。

- `package.json` 保存有关项目的重要信息。它包含可读的元数据，例如项目名称和描述，以及功能性元数据，例如包版本号和应用程序所需的依赖列表。
- `package-lock.json` 是在 npm 修改 `node_modules` 树或 `package.json` 的任何操作中自动生成的。它描述了生成的确切树，允许后续安装生成相同的树，而不管中间依赖的更新。

练习 2：第一次运行 Webpack

让我们运行以下命令，这将以 `src/index.js` 作为入口点，并生成 `dist/main.js` 作为输出。`npx` 命令随 Node 8.2/npm 5.2.0 或更高版本一起提供，它运行我们一开始安装的 Webpack 包的 Webpack 二进制文件 (./node_modules/.bin/webpack)。

```
npx webpack
```

让我们使用 Python 服务器托管 Webpack 的输出。首先，您需要进入 `dist` 目录（在终端中）。

```
cd dist
python3 -m http.server
```

但是我们的输出 HTML 在哪里?

让我们为 Webpack 添加所需的配置！

练习 3：定义 Webpack 配置

在我们项目的根目录中，我们需要创建 `webpack.config.js`，并使用以下配置：

```
const HtmlWebpackPlugin = require("html-webpack-plugin");
module.exports = {
  plugins: [
    new HtmlWebpackPlugin({
      hash: true,
      title: "Fresh Fruit Marketplace",
      metaDesc:
        "Find the freshest and highest quality fruits from local and global sources at our fruit marketplace. Shop for a variety of
      header: "Fruit Marketplace",
      template: "./src/index.html",
      filename: "index.html",
      inject: "body",
    }),
  ],
  mode: "development",
  output: {
    clean: true,
  },
  module: {
    rules: [
      {
        test: /\.(jpg)$/i,
        type: "asset/resource",
        generator: {
          outputPath: "images/",
          publicPath: "images/",
          filename: "[name][ext]",
        },
      },
      {
        test: /\.(css)$/i,
        type: "asset/resource",
        generator: {
          outputPath: "css/",
          publicPath: "css/",
          filename: "[name][ext]",
        },
      },
    ],
  },
};
```

插件

插件是 Webpack 的核心。Webpack 本身是基于您在 Webpack 配置中使用的相同插件系统构建的！

HTML-Webpack-plugin

名称	类型	默认值	描述
hash	{Boolean}	false	如果为 true，则会将唯一的 webpack 编译哈希添加到所有包含的脚本和 CSS 文件中。这对于清除缓存非常有用。
title	{String}	Webpack 应用	用于生成的 HTML 文档。
metaDesc	{String}	‘’	允许通过自定义配置将元描述添加到 HTML 中。
header	{String}	‘’	允许通过自定义配置将页面描述添加到 HTML 中。
template	{String}	‘’	webpack 模板的相对或绝对路径。如果存在，默认使用 src/index.ejs。有关更多信息，请参见 docs。
filename	{String|Function}	'index.html'	HTML 将写入的文件。index.html 是默认名称。您还可以在此处指定子目录（例如：assets/admin.html）。[name] 占位符将被入口名称替换。它也可以是一个函数，例如 (entryName) => entryName + '.html'。
inject	{Boolean|String}	true	true || 'head' || 'body' || false 将所有资产注入到给定的 template 或 templateContent 中。当传递 'body' 时，所有 JavaScript 资源将放置在 body 元素的底部。'head' 将把脚本放在头元素中。传递 true 将根据 scriptLoading 选项将其添加到头/体。传递 false 将禁用自动注入。有关更多详细信息，请参见 inject:false 示例。

模式

使用模式配置选项指示 Webpack 适当地使用其内置优化。

默认为 production，可能的值为：none、development 和 production。

有关更多详细信息，请参见 docs。

输出

顶级 output 键包含一组选项，告诉 Webpack 如何以及在哪里输出捆绑、资产以及与 Webpack 一起捆绑或加载的其他内容。

output.clean

如果设置为 true，它将在发出之前清理输出目录。

有关更多详细信息，请参见 docs。

模块

这些选项决定了项目中不同类型的模块将如何处理。

module.rules

一个 规则 数组，当创建模块时与请求匹配。这些规则可以修改模块的创建方式。它们可以将加载器应用于模块，或修改解析器。

名称	类型	描述
test	{String}	包含所有通过测试断言的模块。
type	{String}	为匹配的模块设置类型。这防止默认规则及其默认导入行为发生。例如，如果您想通过自定义加载器加载 .json 文件，则需要将类型设置为 javascript/auto 以绕过 Webpack 内置的 json 导入。
generator.outputPath	{String}	在相对于 output.path 的指定文件夹中发出资产。
generator.publicPath	{String}	自定义资产模块的 publicPath。
generator.filename	{String}	asset/resource 模块默认将 [hash][ext][query] 文件名发出到输出目录。此参数覆盖这些值。

练习 4：使用 Webpack 在 index.html 中注入值

我们现在需要以不同的方式导入我们的资产（CSS 和图像），并进行必要的更改以注入其他文本值。

1. CSS 导入：

```
<link rel="stylesheet" href=<%=require('./css/main.css')%> type="text/css" media="all" />
```

标题

```
<title><%= htmlWebpackPlugin.options.title %></title>
```

3. 元描述：

```
<meta name="description" content="<%= htmlWebpackPlugin.options.metaDesc %>" />
```

水果市场

```
<h1 class="text-center"><%= htmlWebpackPlugin.options.header %></h1>
```

5. 水果图片：

```
<img src=<%=require('./images/banana.jpg')%> alt="...." />
```

在 `index.js`（`src` 目录）中，添加以下代码：

```
console.log('Hello world!');
```

让我们重新打包并托管它。打开两个终端会更方便，一个指向项目根目录，另一个指向 `dist` 目录。

在终端中，在根目录下运行：

```
npx webpack
```

在另一个上，运行：

```
python3 -m http.server
```

练习 5：查看 dist 目录中的输出

1. `main.js`：Webpack 从 index.js 文件获取源代码并在这里打包最终代码。
2. `index.html`：与之前的类似，但这次是由 Webpack 生成的。我们能够在原本静态的 HTML 中动态注入值。
3. `images`（目录）：包含在我们的 HTML 页面中使用/引用的图像。你不会在这里找到 `mangos.jpg`。
4. `css`（目录）：类似于 `images`，仅包含那些被引用的 CSS 文件。

恭喜！你已完成使用 Webpack 构建和部署网站的实验。

总结：

在这个实验中，你使用 Webpack 从静态 HTML 网站动态创建了一个可分发的版本。这为你提供了一个优化的引用资产包以及参数化的文本。

Author(s)

Muhammad Yahya

(C) IBM Corporation. All rights reserved.