愿景："让编程不再难学，让技术与生活更加有趣"

更多课程请访问 xdclass.net

第一章 课程简介和初始Webpack5

第1集 Webpack零基础到实战课程简介

简介：零基础到实战课程简介

- 适合人群
  - 前端工程师
  - 全栈工程师

- 课程大纲速览

- 学后水平
  - 深入理解 webpack5 配置文件的原理和使用
  - 掌握常用的加载器 loader和插件 plugins 安装使用
  - 灵活应用 webpack5 中的资源模块加载图片、字体资源
  - 掌握在开发环境配置本地服务器实现热更新、接口跨域
  - 利用代码压缩、分离和tree shaking，优化构建的性能
  - 掌握区分不同环境、热更新和source map提高开发效率

- 学习形式
  - 视频讲解+文字笔记+代码分析+交互流程图
  - 配套源码 + 笔记 + 课程软件 + 技术交流群 + 答疑

小滴课堂🌴 大钊-前端架构 👤

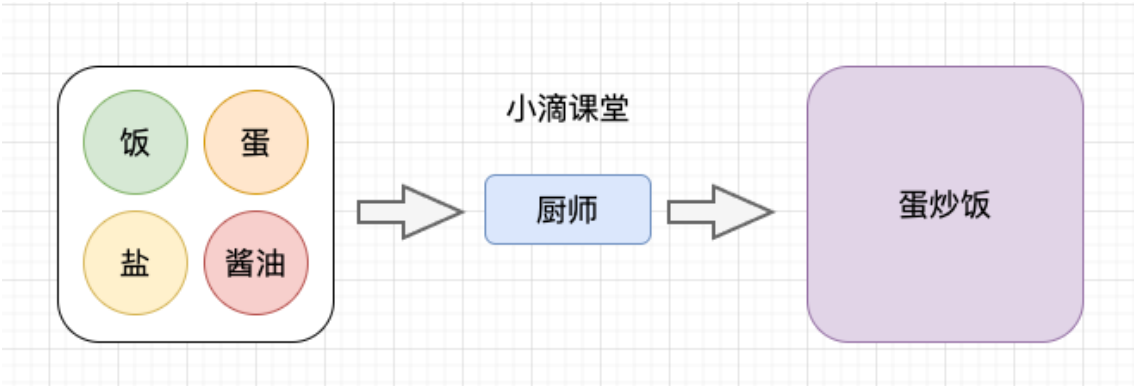
广东 广州

扫一扫上面的二维码图案，加我微信

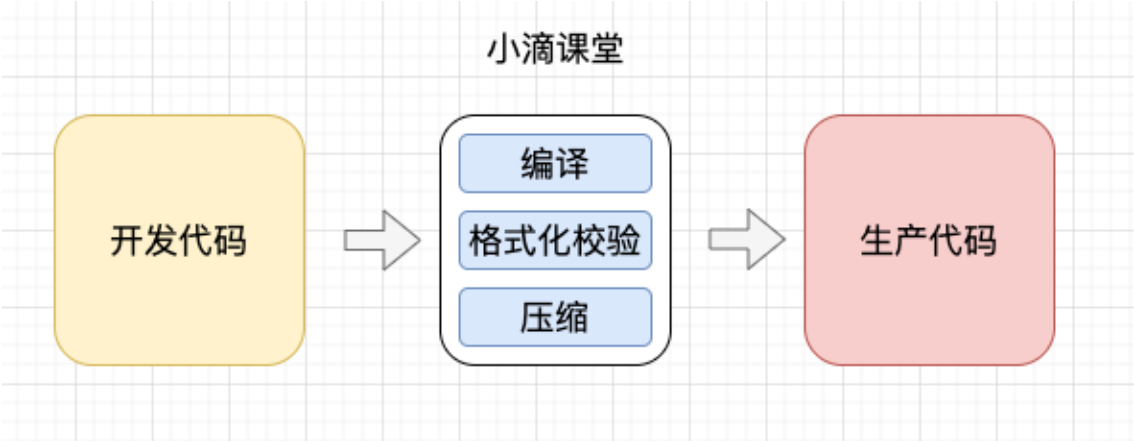
第2集 走进Webpack5的世界—初始

简介：Webpack初始

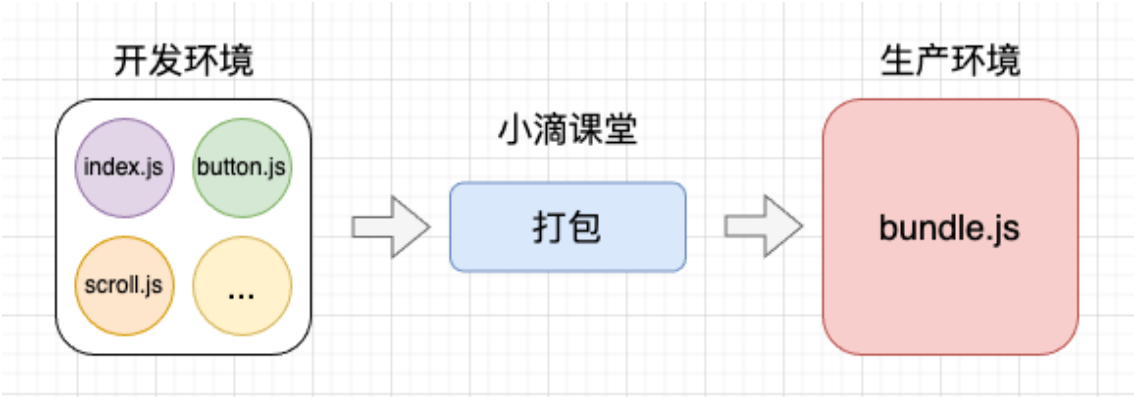
- Webpack初始
  - 版本历史
    - 2012.3—webpack（问世）
    - 2014.2—webpack1
    - 2016.12—webpack2
    - 2017.6—webpack3
    - 2018.2—webpack4
    - 2020.10—webpack5（要求node版本10.13+）
  - 官网
    - https://webpack.docschina.org/
  - 定义
    - **webpack** 是一个用于现代 JavaScript 应用程序的**静态模块打包**工具
    - 静态模块
      - 模块化开发，避免重复代码、 逻辑，提高开发效率
    - 打包
      - 将各个模块，按照一定的规则组装起来



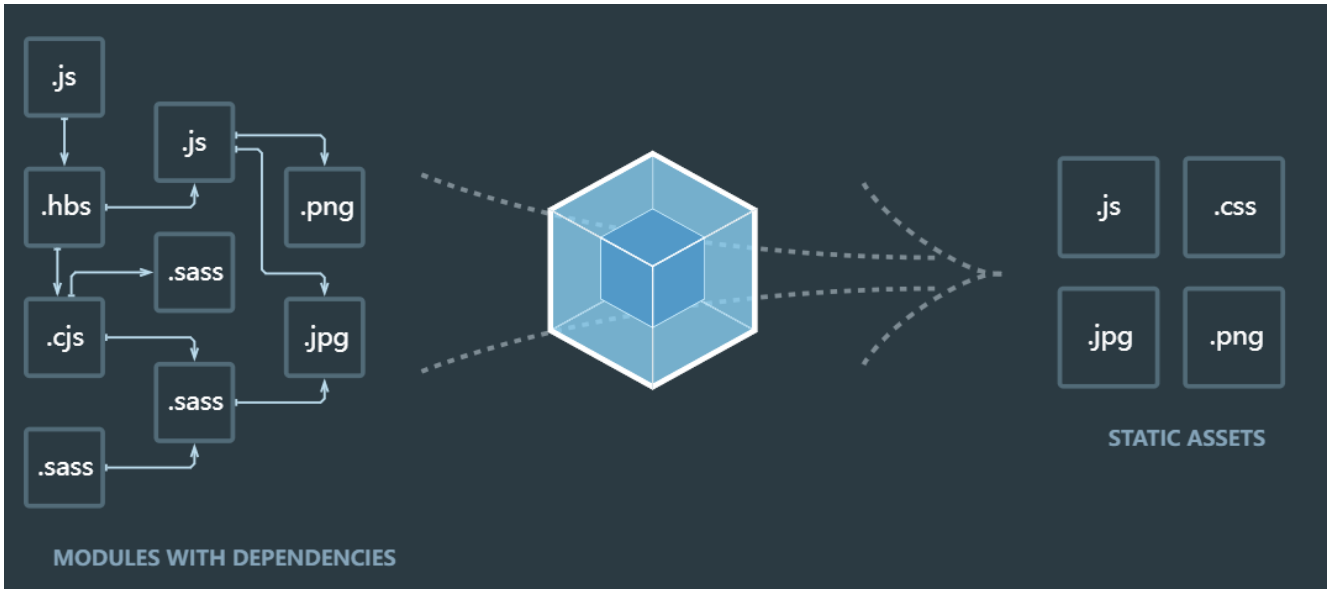
  - 特点
    - 构建（不支持的代码转换成支持的代码）
      - 源代码编译成浏览器能解析的生产代码（如：es6=>es5，scss=>css）
      - 格式化校验
      - 压缩



- 打包
  - 把多个文件合并成一个文件，减少http请求的次数，提高生产环境的运行效率



- 发布 `web` 服务
  - 生成本地的服务器，监听开发代码的修改，自动更新页面的显示

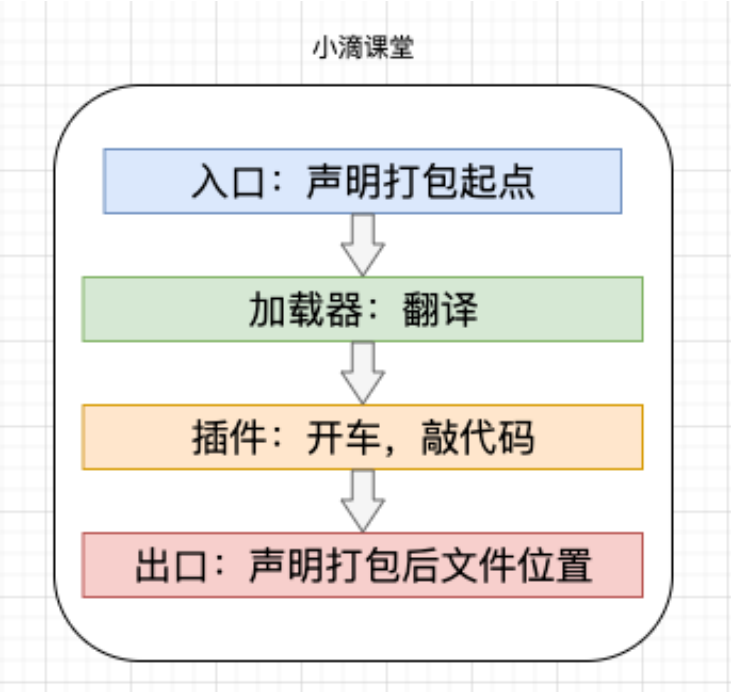


-
  - 功能
    - 将多个不同类型文件打包
    - 编译代码，确保浏览器能解析
    - 对代码进行压缩，减少文件体积，提高加载速度

- 对源代码进行格式化校验
- 有热更新功能，提高开发效率
- 不同环境，提供不同的打包策略

第3集 详解Webpack五个核心配置

简介：Webpack核心概念



- 入口（Entry）
  - 告诉 `webpack` 应该使用哪个模块，来作为构建整个项目的开始

- 出口（Output）
  - 告诉 `webpack` 在哪里输出它所创建的 `bundle`，以及如何命名这些文件（默认为 `main.js`）

- 加载器（Loader）

    - webpack 只能理解 JS 和 JSON 文件，loader 让 webpack 能够去处理其他类型的文件，并将它们转换为有效模块，以供应用程序使用
    - 例子：`css-loader | html-loader`

- 插件（Plugins）
    - 插件是 webpack 的支柱功能，能够执行范围更广的任务，包括：打包优化、压缩、重新定义环境中的变量等
    - 例子：`html-webpack-plugin`

- 模式（Mode）
    - 区分环境，不同的环境打包逻辑不同
        - `development`（优化打包速度，提供调试的辅助功能）
        - `production`（优化打包结果，打包之后代码的运行效率和性能优化）
        - `none`（最原始的打包）

第4集 上手webpack的使用—初体验

简介：上手webpack使用

- 初始化项目

```
cnpm init -y
```

- 安装

```
cnpm i webpack webpack-cli -D
```

- 创建入口文件

```
// ./src/index.js
console.log(hello webpack)
```

- 打包命令（指定mode）

```
webpack ./src/index.js -o ./dist --mode=development
```

愿景："让编程不再难学，让技术与生活更加有趣"

更多课程请访问 xdclass.net

第二章 Webpack5基础—HTML、CSS资源打包

第1集 深入理解webpack配置文件

简介：webpack配置文件

- 特点
  - 简化命令行的选项
  - 默认配置文件名称为webpack.config.js
  - 操作webpack大部分都是在配置webpack.config.js文件

- 配置详情
  - mode（模式）

```
// 打包模式配置
mode: 'development',
```

  - entry（入口）

```
// 入口文件配置
entry: './src/index.js',
```

  - output（出口）

```
const { resolve } = require('path');

// 出口文件配置
output: {
  // 所有输出文件的目标路径，必须是绝对路径
  path: path.resolve(__dirname, 'dist'),
  // 出口文件名配置
  filename: "bundle.js",
},
```

  - module（模块-loader配置）

```
// 模块配置
module: {
  rules: [
    // 配置多个模块规则（配置loader、解析器等选项）
  ],
},
```

- devServer（用于快速开发应用程序）

```
// 开发服务器
devServer: {},
```

- plugin（插件）

```
// 插件配置
plugins: [],
```

- 打包命令设置

```
// package.json
"scripts": {
  "build": "webpack",
},
```

.

第2集 HTML资源打包—自动生成HTML文件和指定模板

简介：详解打包自动生成HTML文件

- 自动生成HTML文件（html-webpack-plugin）
  - 定义
    - 该插件可以生成一个 HTML 文件，并在HTML中加载所有打包资源，便于服务器访问
  - 安装

```
cnpm i html-webpack-plugin -D
```

  - 引入

```
// webpack.config.js
const HtmlWebpackPlugin = require('html-webpack-plugin')
```

- 配置

```
// webpack.config.js
plugins: [
  new HtmlWebpackPlugin(),
],
```

- 指定生成HTML模板
  - 新建HTML文件

```
// ./src/index.html
```

  - 配置

```
// webpack.config.js
plugins: [
  new HtmlWebpackPlugin({
    template:'./src/index.html',  //指定 html 模板
    filename:'index.html',        //指定 html 名称
    title:'hello webpack'         //指定 html title
  }),
],
```

  - `title`使用

```
// 使用EJS语法，是JS的模板引擎
<title><%=htmlWebpackPlugin.options.title%></title>
```

第3集 HTML资源打包—打包多个HTML页面和压缩

简介：详解打包多个HTML页面

- 打包多个HTML页面
  - 配置

```
// webpack.config.js
plugins: [
  new HtmlWebpackPlugin({
    template:'./src/index.html',
    filename:'index.html',
    title:'首页'
  }),
  new HtmlWebpackPlugin({
    template:'./src/index.html',
    filename:'mine.html',
    title:'我的'
  }),
],
```

  - 跳转设置

```
// index.html
<body>
  <div>小滴课堂</div>
  <a href="index.html">首页</a>
  <a href="mine.html">我的</a>
</body>
```

- 压缩
  - 配置

```
new HtmlWebpackPlugin({
  template: './src/index.html',
  title: '我的',
  filename: 'mine.html',
  minify: {
    collapseWhitespace: true,  // 清除空格换行
    removeComments: true,      // 清除注释
  },
}),
```

第4集 CSS资源打包—剖析CSS打包的原理

简介：剖析CSS打包的原理

- 打包css的loader
  - css-loader
    - 将css代码转化为js代码，合并到打包后的js文件中
  - style-loader
    - 将包含css内容的js代码，插入到html中style标签中

- 安装

```
cnpm i css-loader style-loader -D
```

- css文件引入

```
// index,js
import './css/main.css'
```

- webpack配置文件中配置规则

```
// 模块配置
module: {
  rules: [
    // 配置多个模块规则（配置loader、解析器等选项）
    {
      // 匹配css文件
      test: /\.css$/i,
      // 指定加载器，加载顺序是从左到右或者是从下到上
      use: ['style-loader', 'css-loader'],
    },
  ],
},
```

第5集 CSS资源打包—将CSS代码抽离成单独文件

简介：将CSS代码抽离成单独文件

- mini-css-extract-plugin
  - 安装

    ```
    cnpm i mini-css-extract-plugin -D
    ```

  - 引入

    ```
    // webpack.config.js
    const MiniCssExtractPlugin=require('mini-css-extract-plugin')
    ```

  - loader配置（替换style-loader）

    ```
    // webpack.config.js
    use:[MiniCssExtractPlugin.loader,'css-loader']
    ```

  - 实例化插件

    ```
    // webpack.config.js
    new MiniCssExtractPlugin({
      filename:'./css/main.css'
    })
    ```

- 作用
  - 从js文件中抽离出css代码，减少js文件体积
  - 当js文件比较庞大时，可以避免阻碍页面的渲染
  - 提高渲染速度

第6集 CSS资源打包—玩转打包CSS预编译语言

简介：剖析CSS预编译语言

- css预编译语言的安装
  - less

```
cnpm install less less-loader -D
```

  - sass

```
cnpm install node-sass sass-loader -D
```

  - stylus

```
cnpm install stylus stylus-loader -D
```

- 使用

```
// index.js
import './css/main.less'
```

- webpack配置文件中配置规则

```
 // 模块配置
module: {
  rules: [
    {
      test: /\.less$/i,
      use: ['style-loader', 'css-loader', 'less-loader'],
    },
  ],
},
```

第7集 CSS资源打包—对特殊的CSS样式添加兼容前缀

简介：对特殊的CSS样式添加兼容前缀

- 安装

```
cnpm i postcss-loader autoprefixer -D
```

- 配置
  - webpack.config.js

```
use: [
  MiniCssExtractPlugin.loader,
  'css-loader',
  'postcss-loader'   // 处理css兼容
]
```

- postcss.config.js

```
module.exports = {
  plugins: [require('autoprefixer')], // 添加浏览器前缀
};
```

- package.json

```
"browserslist":[
  "last 2 version",    // 兼容浏览器的最近两个版本
  "> 1%"               // 全球占有率超过1%的浏览器
]
```

- 作用
  - 让特殊的css样式兼容各个浏览器

第8集 CSS资源打包—CSS压缩

简介：压缩CSS代码

- optimize-css-assets-webpack-plugin
  - 安装

```
cnpm i optimize-css-assets-webpack-plugin -D
```

  - 引入

```
// webpack.config.js
const OptimizeCssAssetsPlugin = require('optimize-css-assets-webpack-plugin')
```

  - 配置

```
// webpack.config.js
plugins: [
  new OptimizeCssAssetsPlugin(),
],
```

- 作用
  - 压缩后的代码去除了空格和换行
  - 文件体积更小，提高请求的速度

愿景："让编程不再难学，让技术与生活更加有趣"

更多课程请访问 xdclass.net

第三章 webpack5基础— JS、其他资源打包和热更新

第1集 JS资源打包—编译

简介：详解编译

- 目的
  - 将ES5以上的语法转成ES5，保证在低版本浏览器的兼容性
- 安装

```
cnpm i babel-loader @babel/core @babel/preset-env core-js -D
```

- 配置

```
{
  test: /\.m?js$/,
  exclude: /node_modules/,
  use: {
    loader: 'babel-loader',
    options: {
      presets: [
        ['@babel/preset-env',
         {
           useBuiltIns:'usage', //按需加载
           corejs:3,            //指定版本
```

```
            targets: "defaults"
          }
        ]
      ]
    }
  }
}
```

第2集 JS资源打包—代码格式校验

简介：详解代码格式校验

- 安装

```
cnpm i eslint eslint-config-airbnb-base eslint-webpack-plugin eslint-plugin-import -D
```

  - eslint
    - 检验代码格式的工具
  - eslint-config-airbnb-base
    - js代码格式规范
    - 校验的依据：https://github.com/airbnb/javascript
  - eslint-webpack-plugin
    - webpack的eslint插件
  - eslint-plugin-import
    - 用于在package.json中读取eslintConfig配置项
- 使用

```
// webpack.config.js
const ESlintPlugin=require('eslint-webpack-plugin')

...
plugins:[
  new ESlintPlugin({
    fix:true          // 自动修正不符合规范的代码
  })
]
...
```

```
// package.json
"eslintConfig":{
  "extends":"airbnb-base"
}
```

第3集 玩转资源模块Asset Modules

简介：详解资源模块Asset Modules

- Asset Modules
  - 定义
    - Asset Modules 是一种模块，允许我们在不配置额外 loader 的情况下使用资源文件（字体、图片、图标、html等）
  - 官方地址：https://webpack.docschina.org/guides/asset-modules/

- 在 webpack 4 版本，通常使用：
  - `raw-loader` 将文件作为字符串输出
  - `file-loader` 将文件发送到输出目录
  - `url-loader` 设定一个临界值（文件大小），大于该值将文件发送到输出目录，否则将文件转为base64合并到 js 文件当中

- 在webpack5版本，可以使用Asset Modules 的 4 种新的模块类型来替换这些 `loader`
  - `asset/resource`
    - 发出一个单独的文件并导出 URL。替换 `file-loader`.

- `asset/inline`
  - 导出资源的data URI。替换`url-loader`.
- `asset/source`
  - 导出资源的源代码。替换`raw-loader`.
- `asset`
  - 自动在导出data URI 和发出单独文件之间进行选择。之前是使用`url-loader`资产大小限制来实现。

第4集 实现图片和字体资源打包

简介：详解图片和字体资源打包

- 图片资源打包

```
{
  test: /\s/i,
  // asset可以在asset/inline和asset/resource之间进行切换，文件小于8kb时使用asset/inline，否则使用asset/resource
  type: 'asset',
  parser: {
    dataUrlCondition: {
      maxSize: 8 * 1024,
    },
  },
  generator: {
    filename: 'images/[name][ext]',
  },
},
```

- 字体资源打包

```
{
  test: /\.(eot|svg|ttf|woff|woff2)$/i,
  // asset可以在asset/inline和asset/resource之间进行切换，文件小于8kb时使用asset/inline，否则使用asset/resource
  type: 'asset',
  parser: {
    dataUrlCondition: {
      maxSize: 8 * 1024,
    },
  },
  generator: {
    filename: 'fonts/[name][ext]',
  },
},
```

第5集 提高开发效率—开发服务器

简介：详解开发服务器

- 热更新
  - 安装

```
cnpm i webpack-dev-server -D
```

  - 配置

```
devServer:{
  // 告诉服务器从哪里提供内容
  static: {
    directory: resolve(__dirname, 'output'),
  },
  // 打开自动更新
  liveReload: true,
  // 打开gzip压缩
  compress: true,
  // 指定端口号
  port: 8888,
},
// 指定构建的环境
target: 'web',
```

- 配置接口代理（proxy）
  - 目的
    - 解决接口跨域问题
  - 配置

```
proxy: {
  '/api': {
    // 访问的目标地址
    target: 'https://api.xdclass.net',
    // 如果不希望传递/api，则需要重写路径
    pathRewrite: { '^/api': '' },
    // 访问https时需要配置
    secure: false,
  },
  // 覆盖源主机名
  changeOrigin: true,
},
```

愿景："让编程不再难学，让技术与生活更加有趣"

更多课程请访问 xdclass.net

第四章 webpack5进阶—环境区分和代码分离

第1集 开发环境区分—变量区分打包环境

简介：详解开发环境区分

- 目的
  - 不同的环境需要不同打包策略
  - 生产环境需要压缩，但是可读性差，在开发环境一般不开启
  - 生产环境与开发环境的调试接口不一样
- 变量区分打包环境
  - 配置

- 打包命令传参
  - 生产环境打包命令设置

```
// package.json
"scripts": {
  "build:prod": "webpack --env production"
},
```

  - 开发环境

```
npm run build
```

  - 生产环境

```
npm run build:prod
```

第2集 开发环境区分—配置文件区分打包环境

简介：详解开发环境区分

- 配置文件区分打包环境
  - 合并插件
    - webpack-merge
      - 将公共配置文件分别与两个环境的配置文件合并
    - 安装

      ```
      cnpm i webpack-merge -D
      ```

  - 开发环境配置文件

    ```
    // webpack.dev.config.js
    const {merge} = require('webpack-merge')
    const baseWebpackConfig=require('./webpack.base.conf')
    const devWebpackConfig=merge(baseWebpackConfig,{
      // 开发环境的配置
    })
    ```

  - 生产环境配置文件

    ```
    // webpack.prod.config.js
    const {merge} = require('webpack-merge')
    const baseWebpackConfig=require('./webpack.base.conf')
    const prodWebpackConfig=merge(baseWebpackConfig,{
      // 生产环境的配置
    })
    ```

  - 公共配置文件
    - 抽离需要区分环境的配置后的配置文件

- 修改打包命令

第3集 开发环境区分—环境区分全局变量

简介：环境区分不同的全局变量

- DefinePlugin
  - 为项目注入全局变量
- 不同环境注入不同的接口地址
  - 开发环境配置

```
// webpack.dev.config.js
const webpack = require('webpack')
plugins:[
  new webpack.DefinePlugin({
    API_BASE_URL:JSON.stringify('https://apidev.xdclass.com')
  })
]
```

  - 获取接口地址

```
// index.js
console.log('配置的接口：',API_BASE_URL)
```

第4集 代码分离—多入口打包

简介：详解代码分离多入口打包

- 目的
  - 将代码分离到不同打包后的文件中即 `bundle` 文件
  - 可以按需加载或并行加载这些文件
  - 获取更小的 bundle，控制资源加载优先级，降低加载时间
- 分离方式
  - 多入口打包

    - 抽离出公共文件

      ```
      optimization.splitChunks.chunks:all
      ```

    - 动态导入
      - 按需加载
      - 预加载
- 多入口打包
    - 入口 `entry` 配置多个文件
    - 配置

      ```
      // webpack.config.js
      entry: {
        index: './src/index.js',
        mine: './src/mine.js',
      },

      output: {
        filename: '[name].bundle.js',
      },

      HtmlWebpackPlugin({
        chunks:['index']
      })
      HtmlWebpackPlugin({
        chunks:['mine']
      })
      ```

第5集 代码分离—抽离出公共代码防止重复打包

简介：详解代码分离抽离出公共文件

- 目的
    - 将多个页面重复引入的模块抽离成公共的模块，避免重复打包，减少包体积
- 配置

  ```
  optimization: {
    splitChunks: {
      chunks: 'all',
    },
  },
  ```

- 可视化工具
  - 安装

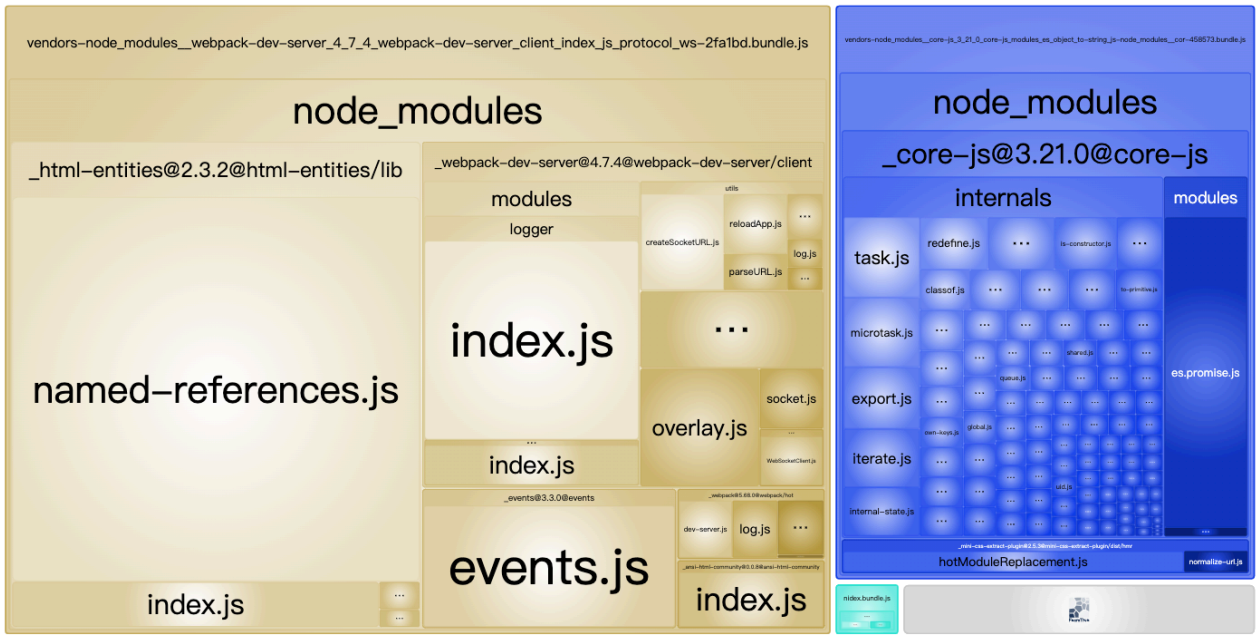
```
cnpm install webpack-bundle-analyzer -D
```

  - 配置

```
// webpack.config.js
const BundleAnalyzerPlugin = require('webpack-bundle-
analyzer').BundleAnalyzerPlugin;

module.exports = {
    plugins: [new BundleAnalyzerPlugin()]
}
```

  - 图示



第6集 代码分离—动态导入

简介：详解代码分离动态导入

- 按需加载（懒加载）
  - 默认不加载，只有页面展示或者事件触发后才加载
  - 指定打包后的文件名称

```
webpackChunkName:'xxx'
```

- 预加载
  - 先等待其他资源加载完成之后再加载
  - 指定需要预加载的内容

```
webpackPrefetchz:true
```

愿景："让编程不再难学，让技术与生活更加有趣"

更多课程请访问 xdclass.net

第五章 webpack5进阶—提高开发效率和性能优化

第1集 掌握代码映射Source Map定位问题

简介：掌握代码映射Source Map定位问题

- Source Map
  - 定义
    - 源代码和构建后代码的映射
  - 目的
    - 当项目运行出现问题或者报错时，通过控制台能够快速定位到具体出错的代码
  - 配置

```
devtool:"source-map"
```

  - 推荐配置

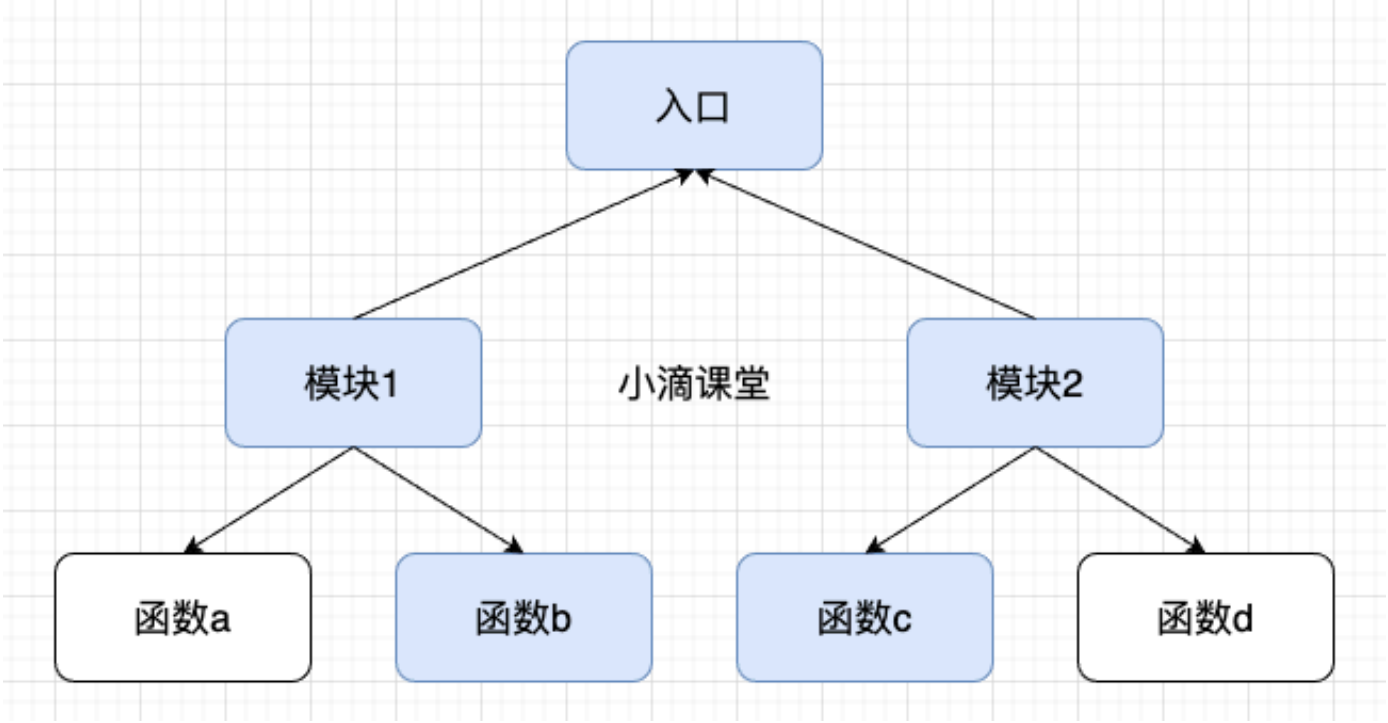
```
devtool:"eval-cheap-module-source-map"
```

    - `eval` 具有最好的性能，但并不能帮助你转译代码。
    - `cheap` 配置的map 质量会稍微差一点（保留行报错，没有列报错），但是不影响问题定位，还可以提高性能
    - `module` 包含第三方模块的报错

第2集 深入理解Tree Shaking《上》

简介：深入理解Tree Shaking

- Tree Shaking（摇树）
    - 描述移除 JavaScript 上下文中的未引用代码
        - 函数return后的代码
        - 只声明未使用的代码
        - 只引入未使用的代码



- 注意
    - Tree-shaking只对ES Module规范的模块起作用
    - 针对静态结构进行分析，只有import和export是静态的导入和导出。而commonjs有动态导入和导出的功能，无法进行静态分析。
- 与Source Map有兼容问题
    - devtool 只能设置以下四种

```
devtool:source-map | inline-source-map | hidden-source-map | nosources-source-map
```

  - eval 模式是将 js 输出为字符串不是 ES Modules规范，导致Tree Shaking失效

- 使用
  - 生产模式
    - 自动开启
  - 开发模式
    - usedExports

```
const TerserPlugin=require('terser-webpack-plugin')
optimization:{
  // 标记未使用的代码
  usedExports:true,
  // 删除已经标记未使用的代码
  minimize:true,
  minimizer:[new TerserPlugin()]
}
```

    - sideEffects

第3集 深入理解Tree Shaking《下》

简介：深入理解Tree Shaking

- 副作用
  - 引入一个模块，调用了模块中的函数，或者修改当前模块、全局的数据，就有副作用
    - 修改全局的变量
    - 在原型上扩展方法
    - css的引入

- 开启副作用

```
optimization:{
  // 开启副作用标识
  sideEffects:true,
}
```

- 标识代码是否有副作用

```
// package.json
"sideEffects":false（告诉webpack所有代码都没有副作用）
"sideEffects":true（告诉webpack所有代码都有副作用）
"sideEffects":['xxx.js','*.less']（告诉webpack哪些有副作用，不移除）
```

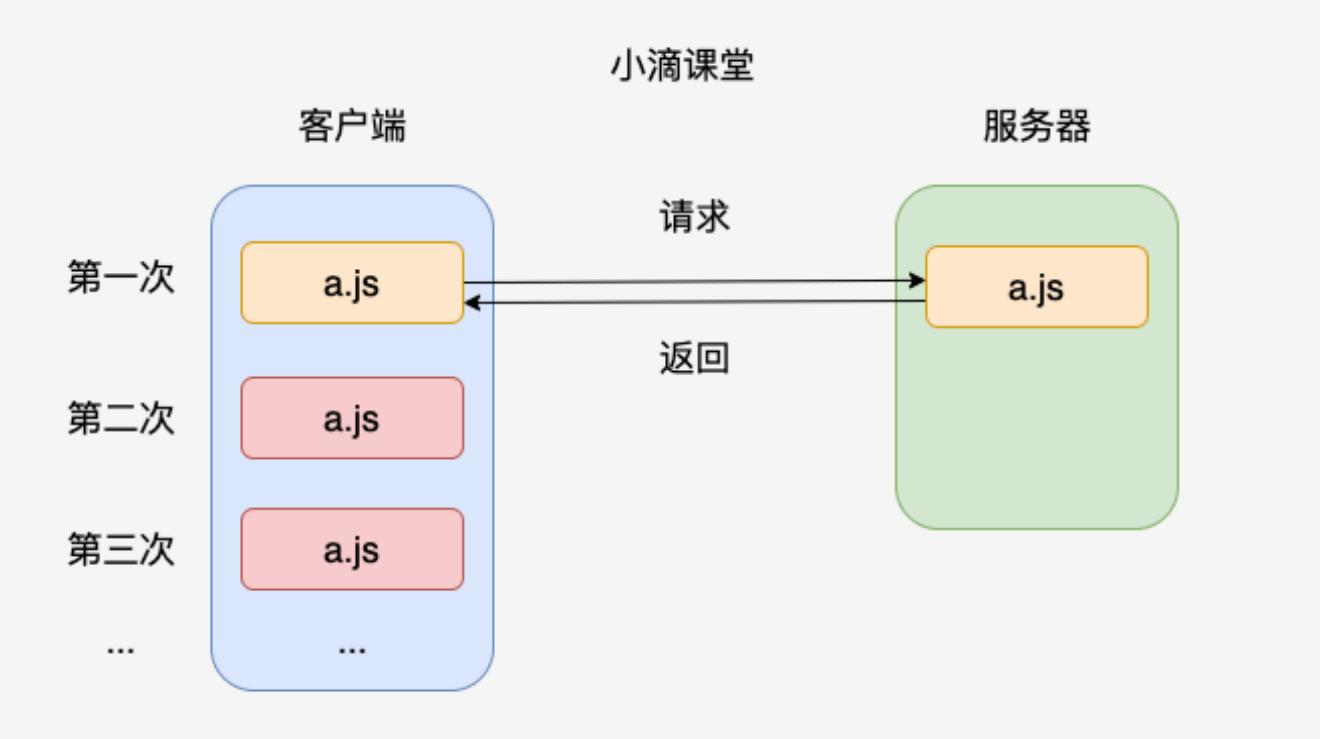
第4集 剖析webpack中的缓存

简介：剖析webpack中的缓存

- babel缓存
  - 特点
    - 第二次构建时，会读取之前的缓存
  - 配置

```
use:[
  options:{
    cacheDirectory:true
  }
]
```

- 文件资源缓存



- 配置webpack哈希值
  - hash
  - chunkhash
  - contenthash

第5集 剖析webpack中的模块解析和排除依赖打包

简介：剖析webpack中的模块解析

- 模块解析
  - 配置

```
// webpack.config.js
resolve:{
  alias:{
    // 指定路径的别名
    '@':resolve('src')
  },
  // 自动解析模块的后缀名
  extensions: ['.js', '.json', '.less'],
}
```

- 排除依赖打包
  - 配置

```
// webpack.config.js
externals:{
  'jquery':'jQuery'
}
```

```
// index.
<script src="https://cdn.bootcdn.net/ajax/libs/jquery/3.6.0/jquery.min.js">
</script>
```