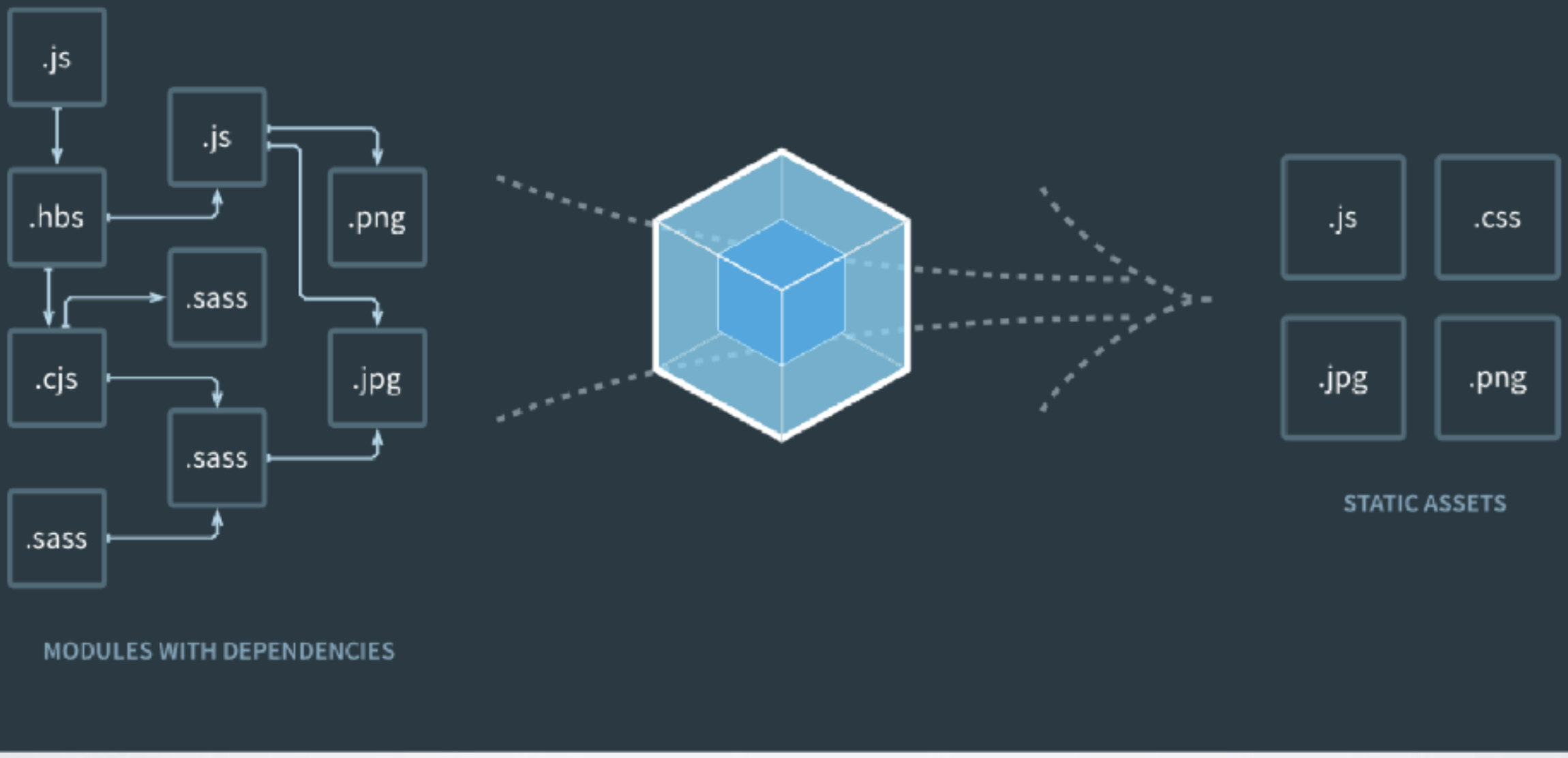

.js
.js
.hbs
.png
.sass
.cjs
.jpg
.sass
.sass
MODULES WITH DEPENDENCIES
.js
.css
.jpg
.png
STATIC ASSETS

什么是webpack

WebPack可以看做是模块打包机:它做的事情是，分析你的项目 结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的 拓展语言(Scss，TypeScript等)，并将其转换和打包为合适的格 式供浏览器使用。

什么是webpack

现今的很多网页其实可以看做是功能丰富的应用，它们拥有着复杂的JavaScript代码和一大堆依赖包。为了简化开发的复杂度，前 端社区涌现出了很多好的实践方法 模块化，让我们可以把复杂的程序细化为小的文件; 类似于TypeScript这种在JavaScript基础上拓展的开发语言: 使我 们能够实现目前版本的JavaScript不能直接使用的特性，并且之后还能转换为JavaScript文件使浏览器可以识别; Scss，less等CSS预处理器 ...

这些改进确实大大的提高了我们的开发效率，但是利用它们开发 的文件往往需要进行额外的处理才能让浏览器识别,而手动处理 又是非常繁琐的，这就为WebPack类的工具的出现提供了需求。

安装webpack

前提条件

在开始之前，请确保安装了 Node.js 的最新版本。使用 Node.js 最新的长期支持版本(升级方法)，是理想的起步。使用旧版本，你可能遇到各种问题，因为它们可能缺少 webpack 功能以及/或者缺少相关 package 包。

本地安装

要安装最新版本或特定版本，请运行以下命令之一：

```
npm install --save-dev webpack
npm install --save-dev webpack@<version>
```

如果4.0后的版本还需安装 CLI

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

对于大多数项目，我们建议本地安装。这可以使我们在引入破坏式变更(breaking change)的依赖时，更容易分别升级项目。通常，webpack 通过运行一个或多个 npm scripts，会在本地 node_modules 目录中查找安装的 webpack

全局安装

以下的 NPM 安装方式，将使 webpack 在全局环境下可用：

```
npm install --global webpack
```

不推荐全局安装 webpack。这会将你项目中的 webpack 锁定到指定版本，并且在使用不同的 webpack 版本的项目中，可能会导致构建失败。

最佳安装

首先我们创建一个目录，初始化 npm，以及在本地安装 webpack：

```
mkdir webpack-demo && cd webpack-demo
npm init -y
npm install webpack webpack-cli --save-dev
```

初始化配置

package.json

```
{
  "name": "webpack-demo",
  "version": "1.0.0",
  "description": "",
  "private": true,
  "scripts": {

  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^4.5.0",
    "webpack-cli": "^2.0.14"
  }
}
```

JS打包

npx默认打包模式

执行 npx webpack，会将我们的脚本作为入口起点，然后输出为 main.js。Node 8.2+ 版本提供的 npx 命令，可以运行在初始安装的 webpack 包(package)的 webpack 二进制文件

```
npx webpack
```

webpack.config.js配置

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  }
```

```
npx webpack --config webpack.config.js
```

如果 webpack.config.js 存在，则 webpack 命令将默认选择使用它。比起 CLI 这种简单直接的使用方式，配置文件具有更多的灵活性。

NPM 脚本(NPM Scripts)

```
{
  ...
  "scripts": {
    "build": "webpack"
  },
  ...
}
```

```
npm run build
```

```
"scripts": {
  "build":"webpack --mode production"
},
```

production和development模式

production模式下，默认对打包的进行minification(文件压缩)，Tree Shaking(只导入有用代码)，scope hoisting（作用域提升）等等操作。
总之是让打包文件更小。
development模式下，对打包文件不压缩，同时打包速度更快。如果没指定任何模式，默认是production模式。

config文件说明

```
module.exports = {
  //入口文件配置
  entry:{
    index:'./src/index.js',
  },
  //出口文件配置
  output: {
    filename: '[name].js',
    path: path.resolve(__dirname, 'dist')
  }
};
```

```
//模块：例如解读css 图片转换等
module:{},
//插件，用于生产模板和各种功能
plugins:[],
//配置webpack开发服务
devServer:{}
```

devServer

webpack-dev-server 为你提供了一个简单的 web 服务器，并且能够实时重新加载(live reloading)

```
npm install --save-dev webpack-dev-server
```

config文件配置

```
//配置webpack开发服务
devServer:{
    contentBase: path.resolve(__dirname, 'dist'),
    host: '127.0.0.1',
    port: '8081',
    compress: true
}
;
```

package.json配置

```
"scripts": {
    "build":"webpack --mode production",
    "start":"webpack-dev-server --open"
},
```

html文件打包

到目前为止，我们在 index.html 文件中手动引入所有资源，然而随着应用程序增长，并且一旦开始输出多个 bundle，手动地对 index.html 文件进行管理，一切就会变得困难起来。然而，可以通过一些插件，会使这个过程更容易操控。

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

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: {
    app: './src/index.js',
    print: './src/print.js'
  },
  plugins: [
    new HtmlWebpackPlugin({
      title: 'Output Management'
    })
  ],
```

```
//插件，用于生产模板和各种功能
plugins:[
    new HtmlWebpackPlugin({
        title: 'Output Management',
        minify:{
            removeAttributeQuotes:true
        },
        hash:true,
        template:'./src/index.html'
    })
],
```

CSS文件打包

在入口js文件中: import css from './css/index.css';

style-loader:用来处理css文件中的url()等，url挂在到js中

css-loader:用来将css插入到页面的style标签

安装style-loader:npm install --save-dev style-loader

安装css-loader:npm install --save-dev css-loader

```
npm install --save-dev style-loader css-loader
```

```
//模块: 例如解读css 图片转换等
module: {
  rules: [
    {
      test: /\.css$/,
      use: ['style-loader','css-loader']
    }
  ]
}
```

CSS分离

安装: npm install --save-dev extract-text-webpack-plugin@next

```
const ExtractTextPlugin = require("extract-text-webpack-plugin");
```

```
plugins:[
    new ExtractTextPlugin("./css/main.css")
]
```

```
module: {
    rules: [
        {
            test: /\.css$/,
            use: ExtractTextPlugin.extract({
                fallback: "style-loader",
                use: "css-loader"
            })
        }
```

css图片打包

npm install --save-dev file-loader url-loader

- test:/\.(png|jpg|gif)/是匹配图片文件后 缀名称。
- **use**:是指定使用的**loader**和**loader**的配 置参数。
- limit:是把小于500000B的文件打成 Base64的格式，写入css。

```
},{
  test: /\.(jpg|png|gif)$/,
  use: [{
      loader: 'file-loader',
      options: {
          limit: 500,
          outputPath: 'images/'
      }
  }]
```

```
rules: [
  {
    test: /\.css$/,
    use: ExtractTextPl
      fallback: "style
      use: "css-loader
      publicPath:'../'
    })
  },{
```

html图片打包

npm install html-withimg-loader --save-dev

```
},{
    test: /\.(html|htm)$/i,
    loader: 'html-withimg-loader'
}
```

sass打包

npm install --save-dev node-sass sass-loader

```
},{
  test: /\.scss$/,
  use: ['style-loader',"css-loader",'sass-loader']
 }
```

```
},{
  test: /\.scss$/,
  // use: ['style-loader',"css-loader",'sass-loader']
  use: ExtractTextPlugin.extract({
    fallback: "style-loader",
    use: ["css-loader", 'sass-loader']
  })
}
```

css前缀

npm install --save-dev postcss-loader autoprefixer

新建文件:postcss.config.js

postcss.config.js

```
module.exports = {
    plugins: [
        require('autoprefixer')
    ]
};
```

```
{
  test: /\.css$/,
  use: ExtractTextPlugin.extract({
    fallback: "style-loader",
    use: [{
      loader: "css-loader",
      options: {importLoaders: 1}
      }, 'postcss-loader'],
    publicPath:'../'
  })
},{
```

Watch

```
build": "webpack --mode development --watch",
```

```
watchOptions: {
  //检测修改时间，以毫秒为单位。
  poll: 1000,
  //防止重复保存，此处设置500毫秒内保存不进行打包操作。
  aggregateTimeout: 500,
  //设置不监听的文件
  ignored: /node_modules/
}
```

清除未使用css样式

npm install --save-dev purifycss-webpack purify-css

```
const glob = require('glob');
const PurifyCSSPlugin = require('purifycss-webpack');
```

```
new PurifyCSSPlugin({
  paths: glob.sync(path.join(__dirname, 'src/*.html')),
})
```

对标签选择器和#div开头的选择器失效

Babel

npm install --save-dev babel-core babel-loader babel-preset-env

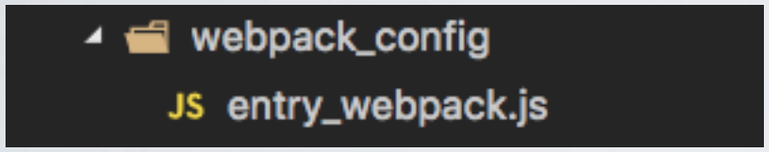
```
},{
    test: /\.js$/,
    use: {
        loader: 'babel-loader',
        options: {presets: 'env'}
    }
}
```

打包注释

```
const webpack = require('webpack');
```

```
new webpack.BannerPlugin('唯创所有'),
```

模块化配置




```
let entry = {
    index: './src/index.js'
};
module.exports = entry;
```

```
const entry = require('./webpack_config/entry_webpack.js');

module.exports = {
  //入口文件配置
  entry:entry,
```

开发环境 与 生产环境

devDependencies 存放测试代码依赖的包或构建工具的包
dependencies 存放项目或组件代码中依赖到的

安装全部项目依赖包:**npm install**
安装生产环境依赖包:**npm install --production**

例：**npm install jquery --save**

引入第三方库

npm install --save jquery

Type 1：

```
import $ from 'jquery';
```

Type 2:

```
const webpack = require('webpack');
```

```
new webpack.ProvidePlugin({
    $: 'jquery'
}),
```

根据页面是否
使用插件
才去引用插件

Type 3：分离文件并指明路径

```
optimization:{
  splitChunks:{
    //必须三选一：  "initial" | "all"(默认就是all) | "async"
    chunks:'initial',
    // 入口的entry的key
    name:'jquery',
    //文件存储路径
    filename :'./assets/js/jquery.js'
  }
}
```