如何自己编写一个Plugin

暗号：做人嘛，最重要的是开心

作业：实现一个打包清单插件

功能：在emit阶段，生成一个fileList.txt，记录output目录下的文件数量和文件名称

fileList:4

- index.html
- main-6666.js
- main-66666.js.map
- kkb.txt

方式：插件代码截图

webpack打包bundle原理

生命周期概念的

启动webpack

读取配置

A插件告知webpack运行到哪个阶段 触发A

webpack 在编译代码过程中，会触发一系列 Tapable 钩子事件，插件所做的，就是找到相应的钩子，往上面挂上自己的任务，也就是注册事件，这样，当 webpack 构建的时候，插件注册的事件就会随着钩子的触发而执行了。

Plugin: 开始打包，在某个时刻，帮助我们处理一些什么事情的机制

plugin要比loader稍微复杂一些，在webpack的源码中，用plugin的机制还是占有非常大的场景，可以说plugin是webpack的灵魂

设计模式

事件驱动

发布订阅

plugin是一个类，里面包含一个apply函数，接受一个参数，compiler

官方文档：https://webpack.js.org/contribute/writing-a-plugin/

Webpack 接收配置 走完整个构建过程（有很多阶段）

插件基本结构

```
// 插件名称
class MyPlugin {
  constructor(options) {}
  // 插件运行方法apply
  apply(compiler) {
    // 插件hooks
    compiler.hooks.done.tap('My Plugin', (/* xxx */) => {
      // 插件处理逻辑
    })
  }
}
```

基本流程

Webpack 的基本流程可以分为 3 个阶段：

- 准备阶段：主要任务是创建 `Compiler` 和 `Compilation` 对象；
- 编译阶段：这个阶段任务是完成 modules 解析，并且生成 chunks；
- module 解析：包含了三个主要步骤，创建实例、loaders 应用和依赖收集；
- chunks 生成，主要步骤是找到每个 `chunk` 所需要包含的 `modules`。
- 产出阶段：这个阶段的主要任务是根据 `chunks` 生成最终文件，主要有三个步骤：模板 Hash 更新，模板渲染 chunk，生成文件。

Compiler

`Compiler` 模块是 Webpack 最核心的模块。每次执行 Webpack 构建的时候，在 Webpack 内部，会首先实例化一个 Compiler 对象，然后调用它的 run 方法来开始一次完整的编译过程。我们直接使用 Webpack API `webpack(options)` 的方式得到的就是一个 `Compiler` 实例化的对象，这时候 Webpack 并不会立即开始构建，需要我们手动执行 `comipler.run()` 才可以。

```
const webpack = require('webpack');
const webpackConfig = require('../webpack.config.js');

// 只传入 config
const compiler = webpack(webpackConfig);
// 开始执行
compiler.run(callback);

// 上面两句等价于
webpack(webpackConfig, callback);
```

> Tips：使用 webpack-dev-server API 方式时，只需要传入 compiler 对象给 dev server 即可，不需要手动执行 `compiler.run()`。

我们如果要手动实例化一个 `Compiler` 对象，可以通过 `const Compiler = webpack.Compiler` 来获取它的类，一般只有一个父 `Compiler`，而子 `Compiler` 可以用来处理一些特殊的事件。

在 webpack plugin 中，每个插件都有个 `apply` 方法。这个方法接收到的参数就是 `Compiler` 对象，我们可以通过在对应的钩子时机绑定处理函数来编写插件，下面主要介绍下 `Compiler` 对象的钩子。

Compiler 钩子

在Webpack 工作流程中，我们通过下面的代码，获取了对应的钩子名称：

```
const compiler = webpack(config);
// 遍历hooks，添加回调，输出`hookName`
Object.keys(compiler.hooks).forEach(hookName => {
    if (compiler.hooks[hookName].tap) {
        compiler.hooks[hookName].tap('anyString', () => {
            console.log(`run -> ${hookName}`);
        });
    }
});
// 触发webpack的编译流程
compiler.run();
```

得到`compiler.run()`之后的工作流程：

```
run -> beforeRun
run -> run
run -> normalModuleFactory
run -> contextModuleFactory
run -> beforeCompile
run -> compile
run -> thisCompilation
run -> compilation
run -> make
run -> afterCompile
run -> shouldEmit
run -> emit
run -> afterEmit
run -> done
```

Compiler Hooks

Compiler 编译器模块是创建编译实例的主引擎。大多数面向用户的插件都首先在 Compiler 上注册。

compiler上暴露的一些常用的钩子：

钩子	类型	什么时候调用
run	AsyncSeriesHook	在编译器开始读取记录前执行
compile	SyncHook	在一个新的compilation创建之前执行
compilation	SyncHook	在一次compilation创建后执行插件
make	AsyncParallelHook	完成一次编译之前执行
emit	AsyncSeriesHook	在生成文件到output目录之前执行，回调参数：`compilation`
afterEmit	AsyncSeriesHook	在生成文件到output目录之后执行
assetEmitted	AsyncSeriesHook	生成文件的时候执行，提供访问产出文件信息的入口，回调参数：`file`，`info`
done	AsyncSeriesHook	一次编译完成后执行，回调参数：`stats`

原材料 --- 源代码

webpack工厂处理

流水线 ----1 ------2---------3----------4-------5---------6---------emit

原材料(compilation)

暗号：做人嘛，最重要的是开心

作业：实现一个打包文件清单插件

webpack构建成功后，会在dist目录生成一个fileList.txt

- 记录打包文件的数量
- 记录打包文件的名称

方式：代码截图

案例：

- 创建copyright-webpack-plugin.js

```
class CopyrightWebpackPlugin {
  constructor() {
  }

  //compiler：webpack实例
  apply(compiler) {

  }
}
module.exports = CopyrightWebpackPlugin;
```

- 配置文件里使用

```
const CopyrightWebpackPlugin = require("./plugin/copyright-webpack-plugin");

plugins: [new CopyrightWebpackPlugin()]
```

- 如何传递参数

```
//webpack配置文件
plugins: [
    new CopyrightWebpackPlugin({
      name: "开课吧"
    })
  ]

//copyright-webpack-plugin.js
class CopyrightWebpackPlugin {
  constructor(options) {
    //接受参数
    console.log(options);
  }

  apply(compiler) {}
}
module.exports = CopyrightWebpackPlugin;
```

- 配置plugin在什么时刻进行

```
class CopyrightWebpackPlugin {
  constructor(options) {
    // console.log(options);
  }

  apply(compiler) {
    //hooks.emit 定义在某个时刻
    compiler.hooks.emit.tapAsync(
      "CopyrightWebpackPlugin",
      (compilation, cb) => {
        compilation.assets["copyright.txt"] = {
          source: function() {
            return "hello copy";
          },
          size: function() {
            return 20;
          }
        };
        cb();
      }
    );

    //同步的写法
    //compiler.hooks.compile.tap("CopyrightWebpackPlugin", compilation => {
    //  console.log("开始了");
    //});
  }
}
module.exports = CopyrightWebpackPlugin;
```

开发一个文件清单插件

我希望每次webpack打包后，自动产生一个打包文件清单，上面要记录文件名、文件数量等信息。

思路：

- 显然这个操作需要在文件生成到dist目录之前进行，所以我们要注册的是`Compiler`上的`emit`钩子。
- `emit` 是一个异步串行钩子，我们用 `tapAsync` 来注册。
- 在 `emit` 的回调函数里我们可以拿到 `compilation` 对象，所有待生成的文件都在它的 `assets` 属性上。
- 通过 `compilation.assets` 获取我们需要的文件信息，并将其整理为新的文件内容准备输出。
- 然后往 `compilation.assets` 添加这个新的文件。

插件完成后，最后将写好的插件放到 webpack 配置中，这个包含文件清单的文件就会在每次打包的时候自动生成了。

实现：

```
// plugins/FileListPlugin.js

class FileListPlugin {
    constructor (options) {
      // 获取插件配置项
        this.filename = options && options.filename ? options.filename :
'FILELIST.md';
    }

    apply(compiler) {
        // 注册 compiler 上的 emit 钩子
        compiler.hooks.emit.tapAsync('FileListPlugin', (compilation, cb) => {

            // 通过 compilation.assets 获取文件数量
            let len = Object.keys(compilation.assets).length;

            // 添加统计信息
            let content = `# ${len} file${len>1?'s':''} emitted by
webpack\n\n`;

            // 通过 compilation.assets 获取文件名列表
            for(let filename in compilation.assets) {
                content += `- ${filename}\n`;
            }

            // 往 compilation.assets 中添加清单文件
            compilation.assets[this.filename] = {
              // 写入新文件的内容
                source: function() {
                    return content;
                },
                // 新文件大小（给 webapck 输出展示用）
                size: function() {
                    return content.length;
                }
            }

            // 执行回调，让 webpack 继续执行
            cb();
        })
    }
}
```

```
module.exports = FileListPlugin;
复制代码
```

测试：

在 webpack.config.js 中配置我们自己写的plugin：

```
plugins: [
    new MyPlugin(),
    new FileListPlugin({
        filename: '_filelist.md'
    })
]
复制代码
```

`npm run build` 执行，可以看到生成了 `_filelist.md` 文件：

```
{
  a:"./src/a.js",
  b:"./src/b.js"
  c:"./src/c.js"
}

===>{
  a.js
  b.js
  c.js
}

new htmlwebpack{
  temp
}
new htmlwebpack{
  temp
}
new htmlwebpack{
  temp
}
```

参考：compiler-hooks

https://webpack.js.org/api/compiler-hooks