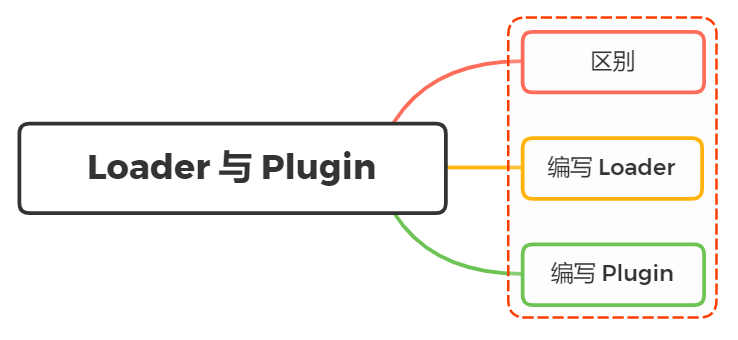
# 面试官：说说Loader和Plugin的区别？编写Loader，Plugin的思路？

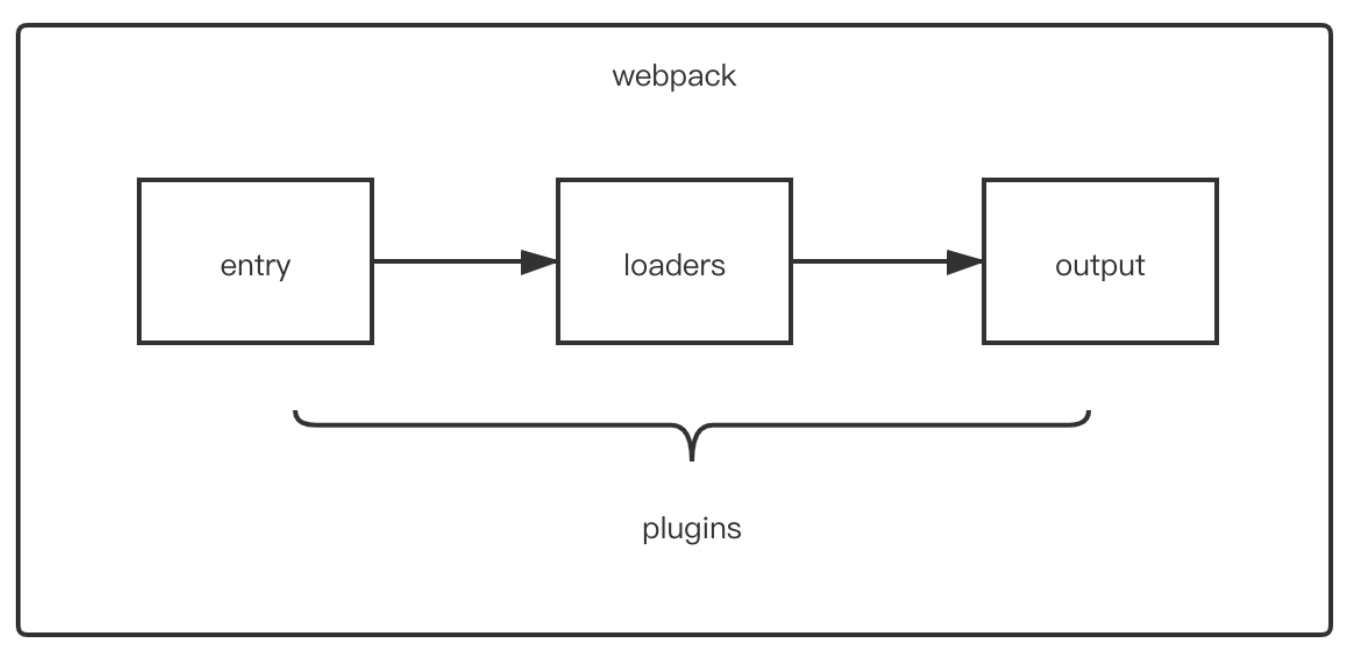


## 一、区别

前面两节我们有提到Loader与Plugin对应的概念，先来回顾下

* loader 是文件加载器，能够加载资源文件，并对这些文件进行一些处理，诸如编译、压缩等，最终一起打包到指定的文件中
* plugin 赋予了 webpack 各种灵活的功能，例如打包优化、资源管理、环境变量注入等，目的是解决 loader 无法实现的其他事

从整个运行时机上来看，如下图所示：



可以看到，两者在运行时机上的区别：

* loader 运行在打包文件之前
* plugins 在整个编译周期都起作用

在Webpack 运行的生命周期中会广播出许多事件，Plugin 可以监听这些事件，在合适的时机通过Webpack提供的 API改变输出结果

对于loader，实质是一个转换器，将A文件进行编译形成B文件，操作的是文件，比如将A.scss或A.less转变为B.css，单纯的文件转换过程

## 二、编写loader

在编写 loader 前，我们首先需要了解 loader 的本质

其本质为函数，函数中的 this 作为上下文会被 webpack 填充，因此我们不能将 loader设为一个箭头函数

函数接受一个参数，为 webpack 传递给 loader 的文件源内容

函数中 this 是由 webpack 提供的对象，能够获取当前 loader 所需要的各种信息

函数中有异步操作或同步操作，异步操作通过 this.callback 返回，返回值要求为 string 或者 Buffer

代码如下所示：

// 导出一个函数，source为webpack传递给loader的文件源内容  
module.exports = function(source) {  
 const content = doSomeThing2JsString(source);  
   
 // 如果 loader 配置了 options 对象，那么this.query将指向 options  
 const options = this.query;  
   
 // 可以用作解析其他模块路径的上下文  
 console.log('this.context');  
   
 /\*  
 \* this.callback 参数：  
 \* error：Error | null，当 loader 出错时向外抛出一个 error  
 \* content：String | Buffer，经过 loader 编译后需要导出的内容  
 \* sourceMap：为方便调试生成的编译后内容的 source map  
 \* ast：本次编译生成的 AST 静态语法树，之后执行的 loader 可以直接使用这个 AST，进而省去重复生成 AST 的过程  
 \*/  
 this.callback(null, content); // 异步  
 return content; // 同步  
}

一般在编写loader的过程中，保持功能单一，避免做多种功能

如less文件转换成 css文件也不是一步到位，而是 less-loader、css-loader、style-loader几个 loader的链式调用才能完成转换

## 三、编写plugin

由于webpack基于发布订阅模式，在运行的生命周期中会广播出许多事件，插件通过监听这些事件，就可以在特定的阶段执行自己的插件任务

在之前也了解过，webpack编译会创建两个核心对象：

* compiler：包含了 webpack 环境的所有的配置信息，包括 options，loader 和 plugin，和 webpack 整个生命周期相关的钩子
* compilation：作为 plugin 内置事件回调函数的参数，包含了当前的模块资源、编译生成资源、变化的文件以及被跟踪依赖的状态信息。当检测到一个文件变化，一次新的 Compilation 将被创建

如果自己要实现plugin，也需要遵循一定的规范：

* 插件必须是一个函数或者是一个包含 apply 方法的对象，这样才能访问compiler实例
* 传给每个插件的 compiler 和 compilation 对象都是同一个引用，因此不建议修改
* 异步的事件需要在插件处理完任务时调用回调函数通知 Webpack 进入下一个流程，不然会卡住

实现plugin的模板如下：

class MyPlugin {  
 // Webpack 会调用 MyPlugin 实例的 apply 方法给插件实例传入 compiler 对象  
 apply (compiler) {  
 // 找到合适的事件钩子，实现自己的插件功能  
 compiler.hooks.emit.tap('MyPlugin', compilation => {  
 // compilation: 当前打包构建流程的上下文  
 console.log(compilation);  
   
 // do something...  
 })  
 }  
}

在 emit 事件发生时，代表源文件的转换和组装已经完成，可以读取到最终将输出的资源、代码块、模块及其依赖，并且可以修改输出资源的内容

## 参考文献

* https://webpack.docschina.org/api/loaders/
* https://webpack.docschina.org/api/compiler-hooks/
* https://segmentfault.com/a/1190000039877943
* https://vue3js.cn/interview