在webpack配置object中是通过devtool来设置sourcemap的，这里有几个选择，下面我们来具体解释他们之间的区别吧

* source-map 这个是产生一个完整的全面的source map，这个选项的效果最好，但是它会降低build的效率
* cheap-module-source-map 这个是选项也是单独产生一个source map文件，但是去掉了具体列的信息，所以会降低调试的方便，却提高了build的效率。
* **eval-source-map bundle源代码是利用'eval', source map完整的和bundle的结果在同一个文件中。这个有和好的调试效果，同时不影响build的效率，但是可能会有执行效率和安全的缺点，但是在开发的过程是一个很好的选择。**
* cheap-module-eval-source-map 这个是build效率最高的方式，和eval-source-map相似，但是去掉了具体的列的信息。和eval-source-map相似，它有执行效率和安全性的缺点，所以不适合用在生产环境中。

正想你看到的一样，这四个选项是从上而下build速度越来越快，但是越上面的越产生没有缺点的source map。

特别在中小型的项目中，eval-source-map是一个不错的选择，它build效率比较高，同时调试比较方便，同时我们可以写一个专门用于生产环境的webpack配置文件。后面会讲到具体的应用。

module.exports = {

devtool: 'eval-source-map',

entry: \_\_dirnam + '/app/main.js',

output: {

path: \_\_dirname + '/public',

filename: "bundle.js"

}

}

### webpack development server

webpack development server 是一个webpack可选的本地开发的server。它通过nodejs的express 来起一个server提供静态文件服务，同时它根据配置信息（xxx.config.js）来打包资源，存在内存中，同时当你的代码发生改变的时候，它还可以刷新你的浏览器。

它是一个单独的npm module，通过npm install webpack-dev-server --save-dev来给项目安装依赖。

webpack dev server可以通过webpack.config.js的devServer选项来配置，具体配置包括：

* contentBase： 默认webpack dev server是从项目的根目录提供服务，如果要从不同的目录提供服务，可以通过contentBase来配置，比如rails中可以把contentBase配置成'./public'。
* port: 默认webpack是用 8080端口起的，通过port可以配成其他的端口。
* inline: 设置为true，代码有变化，浏览器端刷新。
* colors: 这个是当server跑的时候，terminal输出带颜色。
* historyApiFallback: 这个是干嘛用的嘞？对于单页面程序，浏览器的brower histroy可以设置成html5 history api或者hash，而设置为html5 api的，如果刷新浏览器会出现404 not found，原因是它通过这个路径（比如： /activities/2/ques/2）来访问后台，所以会出现404，而把historyApiFallback设置为true那么所有的路径都执行index.html。

module.exports = {

devtool: 'eval-source-map',

entry: \_\_dirname + "/app/main.js",

output: {

path: \_\_dirname + '/public',

filename: 'bundle.js'

},

devServer: {

contentBase: './public',

color: true,

historyApiFallback: true,

inline: true

}

}

webpack通过loader来加载各种各样的资源，不同的资源应用的不同的loader ，举个例子：打包es6会用到babel-loader，打包css用到style-loader和css-loader等等。  
loaders是通过单独的npm来安装的，然后在webpack.config.js中通过module来配置。loader的配置包括：

* test: 一个正则表达式，用于检测不同的文件的后缀名，然后配置配置不同的loader。
* loader: loader的名字，比如'babel-loader'
* include/exclude: 一个选项来配置哪些目录和文件需要排除掉或者加上
* query: 这个query settings可以用于传递不同的参数给loader

module: {

loaders: [

{

test: /\.json$/,

loader: "json"

}

]

},

##### babel的安装和配置

babel是一个模块化的并且分发到不同的npm modules。核心的功能 babel-core 是通过babel-loader按装来直接使用的。但是，对于一些其他的功能和扩展要另外的安装（最常用的是babel-preset-es2015和babel-preset-react分别用于支持es6和react jsx）。  
我们来安装所需的包npm install --save-dev babel-core babel-loader babel-preset-es2015 babel-preset-react

##### babel的配置文件

babel可以通过webpack的配置文件直接配置，但是它有很多的配置信息，都放到同一个webpack的配置文件中会使得配置文件不好维护。因为这个原因很多的开发者选择了单独的babel配置文件'.babelrc'，来配置bebel的选项等等。  
目前我们对babel的配置只是presets，所以我们先把这个配置放到'.babelrc'中吧

去掉了对babel loader的presets配置，那么.babelrc文件如下：

{

"presets": ["react", "es2015"]

}

##### 样式表

webpack提供了css-loader和style-loader来处理样式表。不同的loader处理不同的任务，其中css-loader处理@import和url值来解决他们的依赖关系，然后style-loader把这些计算后的样式表加到页面上。总结来说呢，就是这两个loader共同实现了把样式表嵌入到webpack的js bundle中。

现在把这两个loader添加到我们的小项目中。

首先通过npm来安装 npm install --save-dev style-loader css-loader

##### 样式表

webpack提供了css-loader和style-loader来处理样式表。不同的loader处理不同的任务，其中css-loader处理@import和url值来解决他们的依赖关系，然后style-loader把这些计算后的样式表加到页面上。总结来说呢，就是这两个loader共同实现了把样式表嵌入到webpack的js bundle中。

现在把这两个loader添加到我们的小项目中。

首先通过npm来安装 npm install --save-dev style-loader css-loader

module: {

loaders: [

{test: /\.json$/, loader: "json"},

{test: /\.js$/, exclude: /node\_modules/, loader: 'babel'},

{test: /\.css$/, loader: 'style!css'}

]

},

注意 loader中的(!)这个符号用于连接不同的loader的，在这里就是.css文件要被style-loader和css-loader同时处理。

import './main.css';

##### css 模块

一个项目叫做'css modules'目的就是把模块化等等的特点带给css。通过css modules，所有的css的classname和animation name都是本地scoped。webpack从一开始就加入了这个项目在css loader中，只是我们需要显示的开启它。有了这个功能，你可以export class来给指定的component。

module: {

loaders: [

{ test: /\.json$/, loader: "json" },

{ test: /\.js$/, exclude: /node\_modules/, loader: 'babel' }, {

test: /\.css$/,

loader: 'style!css?modules' }

]

},

import styles from './Greeter.css';

<div className={styles.root}>

.root {

background-color: #eee,

padding: 10px;

border: 3px solid #ccc;

}

##### css的预处理器

css的预处理比如sass less stylus 都扩展扩展了css的语法。他们让我们可以利用一些css中没有的功能来写css，比如变量，函数，嵌套，mixins等等。

PostCSS是一个css转译工具。通过连接不同的插件，来应用不用的转译到你的css文件。你可以了解更多通过[这个](https://github.com/postcss/postcss" \t "_blank)。  
这里我们通过PostCSS和autoprefixer插件来举例子，其中autoprefixer是给我们的css自动添加浏览器供应商前缀。  
首先安装这些包， npm install --save-dev postcss-loader autoprefixer

module: {

loaders: [

{ test: /\.json$/, loader: "json" },

{ test: /\.js$/, exclude: /node\_modules/, loader: 'babel' },

{

test: /\.css$/,

loader: 'style!css?modules!postcss'

} ]

},

postcss: [ require('autoprefixer')],

devServer: {...}