

模块化原理和source-map

Mode配置

- 前面我们一直没有讲mode，但是在这里我们要简单讲一下，后面还会提到它的其他用法。
- Mode配置选项，可以告知webpack使用响应模式的内置优化：
 - 默认值是production（什么都不设置的情况下）；
 - 可选值有：'none' | 'development' | 'production'；
- 这几个选项有什么样的区别呢？

选项	描述
development	会将 DefinePlugin 中 process.env.NODE_ENV 的值设置为 development。为模块和 chunk 启用有效的名。
production	会将 DefinePlugin 中 process.env.NODE_ENV 的值设置为 production。为模块和 chunk 启用确定性的混淆名称，FlagDependencyUsagePlugin，FlagIncludedChunksPlugin，ModuleConcatenationPlugin，NoEmitOnErrorsPlugin 和 TerserPlugin。
none	不使用任何默认优化选项

Mode配置代表更多

```
// webpack.development.config.js
module.exports = {
  + mode: 'development',
  - devtool: 'eval',
  - cache: true,
  - performance: {
  -   hints: false
  - },
  - output: {
  -   pathinfo: true
  - },
  - optimization: {
  -   moduleIds: 'named',
  -   chunkIds: 'named',
  -   mangleExports: false,
  -   nodeEnv: 'development',
  -   flagIncludedChunks: false,
  -   occurrenceOrder: false,
  -   concatenateModules: false,
  -   splitChunks: {
  -     hidePathInfo: false,
  -     minSize: 10000,
  -     maxAsyncRequests: Infinity,
  -     maxInitialRequests: Infinity,
  -   },
  -   emitOnErrors: true,
  -   checkWasmTypes: false,
  -   minimize: false,
  -   removeAvailableModules: false
  - },
  - plugins: [
  -   new webpack.DefinePlugin({ "process.env.NODE_ENV": JSON.stringify("development") })
  - ]
}
```

```
// webpack.production.config.js
module.exports = {
  + mode: 'production',
  - performance: {
  -   hints: 'warning'
  - },
  - output: {
  -   pathinfo: false
  - },
  - optimization: {
  -   moduleIds: 'deterministic',
  -   chunkIds: 'deterministic',
  -   mangleExports: 'deterministic',
  -   nodeEnv: 'production',
  -   flagIncludedChunks: true,
  -   occurrenceOrder: true,
  -   concatenateModules: true,
  -   splitChunks: {
  -     hidePathInfo: true,
  -     minSize: 30000,
  -     maxAsyncRequests: 5,
  -     maxInitialRequests: 3,
  -   },
  -   emitOnErrors: false,
  -   checkWasmTypes: true,
  -   minimize: true,
  - },
  - plugins: [
  -   new TerserPlugin(/* ... */),
  -   new webpack.DefinePlugin({ "process.env.NODE_ENV": JSON.stringify("production") }),
  -   new webpack.optimize.ModuleConcatenationPlugin(),
  -   new webpack.NoEmitOnErrorsPlugin()
  - ]
}
```

Webpack的模块化

■ Webpack打包的代码，允许我们使用各种各样的模块化，但是最常用的是**CommonJS**、**ES Module**。

□ 那么它是如何帮助我们实现了代码中**支持模块化**呢？

■ 我们来研究一下它的原理，包括如下原理：

□ CommonJS模块化实现原理；

□ ES Module实现原理；

□ CommonJS加载ES Module的原理；

□ ES Module加载CommonJS的原理；

■ 这里不再给出代码，查看课堂代码的注释解析；

认识source-map

- 我们的代码通常运行在浏览器上时，是通过**打包压缩**的：
 - 也就是**真实跑在浏览器上的代码**，和**我们编写的代码**其实是有差异的；
 - 比如**ES6的代码**可能被转换成**ES5**；
 - 比如**对应的代码行号、列号**在经过编译后肯定会不一致；
 - 比如代码进行**丑化压缩**时，会将**编码名称**等修改；
 - 比如我们使用了**TypeScript**等方式编写的代码，最终转换成**JavaScript**；
- 但是，当代码报错需要**调试时（debug）**，调试**转换后的代码**是很困难的
- 但是我们能保证代码不出错吗？**不可能**。
- 那么如何可以**调试这种转换后不一致**的代码呢？答案就是**source-map**
 - source-map是从**已转换的代码**，映射到**原始的源文件**；
 - 使浏览器可以**重构原始源**并在调试器中**显示重建的原始源**；

如何使用source-map

■ 如何可以使用source-map呢？两个步骤：

- **第一步**：根据源文件，生成source-map文件，webpack在打包时，可以通过配置生成source-map；
- **第二步**：在转换后的代码，最后添加一个注释，它指向sourcemap；

```
//# sourceMappingURL=common.bundle.js.map
```

■ 浏览器会根据我们的注释，查找响应的source-map，并且根据source-map还原我们的代码，方便进行调试。

■ 在Chrome中，我们可以按照如下的方式打开source-map：

Default indentation:	<input type="text" value="4 spaces"/>	Show whitespace characters:	<input type="text" value="None"/>
<input type="checkbox"/> Search in anonymous and content scripts		<input checked="" type="checkbox"/> Display variable values inline while debugging	
<input type="checkbox"/> Automatically reveal files in sidebar		<input checked="" type="checkbox"/> Enable CSS source maps	
<input checked="" type="checkbox"/> Enable JavaScript source maps		<input checked="" type="checkbox"/> Allow scrolling past end of file	
<input type="checkbox"/> Enable tab moves focus			

分析source-map

■ 最初source-map生成的文件带下是原始文件的10倍，第二版减少了约50%，第三版又减少了50%，所以目前一个133kb的文件，最终的source-map的大小大概在300kb。

■ 目前的source-map长什么样子呢？

□ **version**：当前使用的版本，也就是最新的第三版；

□ **sources**：从哪些文件转换过来的source-map和打包的代码（最初始的文件）；

□ **names**：转换前的变量和属性名称（因为我目前使用的是development模式，所以不需要保留转换前的名称）；

□ **mappings**：source-map用来和源文件映射的信息（比如位置信息等），一串base64 VLQ（variable-length quantity可变长度值）编码；

□ **file**：打包后的文件（浏览器加载的文件）；

□ **sourceContent**：转换前的具体代码信息（和sources是对应的关系）；

□ **sourceRoot**：所有的sources相对的根目录；

source-map文件

- 参考文档 (MDN) : https://developer.mozilla.org/en-US/docs/Mozilla/JavaScript_code_modules/SourceMap.jsm

```
{
  "version": 3,
  "sources": [ ...
  ],
  "names": [],
  "mappings": ";;;;;;;;;AAAA;AACA;AACA;;AAEA;AACA;AACA,C;;;;;;;;;;ACNA,OAAO,MAAM,GAAG,mBAAO,CAAC,8BAAO;;AAE/B;AACA;AACA;;AAEA;;AAEA;AACA;AACA;AACA,C;;;;;;;;;;ACXO;AACP;AACA;;AAEO;AACP;AACA;;;;;;;;;;UCNA;UACA;;UAEA;UACA;UACA;UACA;UACA;UACA;UACA;UACA;UACA;UACA;UACA;;UAEA;UACA;;UAEA;UACA;UACA;;;;WCrBA;WACA;WACA;WACA;WACA,wCAAwC,yCAAyC;WACjF;WACA;WACA,E;;;;;;;;;WCPA,sF;;;;;;;;WCAA;WACA;WACA;WACA,sDAAsD,kBAAkB;WACxE;WACA,+CAA+C,cAAc;WAC7D,E;;;;;;;;;;ACNA,OAAO,aAAa,GAAG,mBAAO,CAAC,uCAAa;AAC5C,OAAO,MAAM,GAAG,mBAAO,CAAC,mCAAW;;AAEnC;AACA",
  "file": "common.bundle.js",
  "sourcesContent": [ ...
  ],
  "sourceRoot": ""
}
```


生成source-map

■ 如何在使用webpack打包的时候，生成对应的source-map呢？

□ webpack为我们提供了非常多的选项（目前是26个），来处理source-map；

□ <https://webpack.docschina.org/configuration/devtool/>

□ 选择不同的值，生成的source-map会稍微有差异，打包的过程也会有性能的差异，可以根据不同的情况进行选择；

■ 下面几个值不会生成source-map

■ **false**：不使用source-map，也就是没有任何和source-map相关的内容。

■ **none**：production模式下的默认值，不生成source-map。

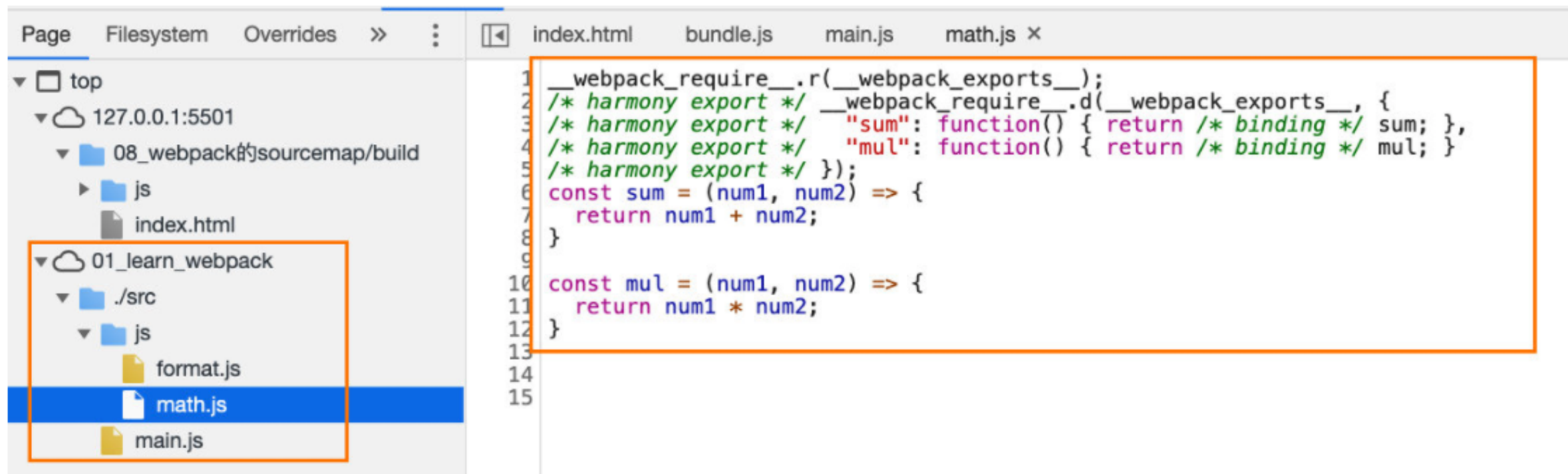
■ **eval**：development模式下的默认值，不生成source-map

□ 但是它会在eval执行的代码中，添加 `//# sourceMappingURL=`；

□ 它会被浏览器在执行时解析，并且在调试面板中生成对应的一些文件目录，方便我们调试代码；

eval的效果

```
eval("__webpack_require__.r(__webpack_exports__);\n/* harmony export */ __webpack_require__.d(__webpack_exports__, {\n/* harmony export */   \"sum\": () => /* binding */ sum,\n/* harmony export */   \"mul\": () => /* binding */ mul\n/* harmony export */ });\nconst sum = (num1, num2) => {\n  return num1 + num2;\n}\nconst mul = (num1, num2) => {\n  return num1 * num2;\n}\n\n\n// # sourceMappingURL=webpack://webpack_module_principle/./src/js/math.js?");
```



source-map值

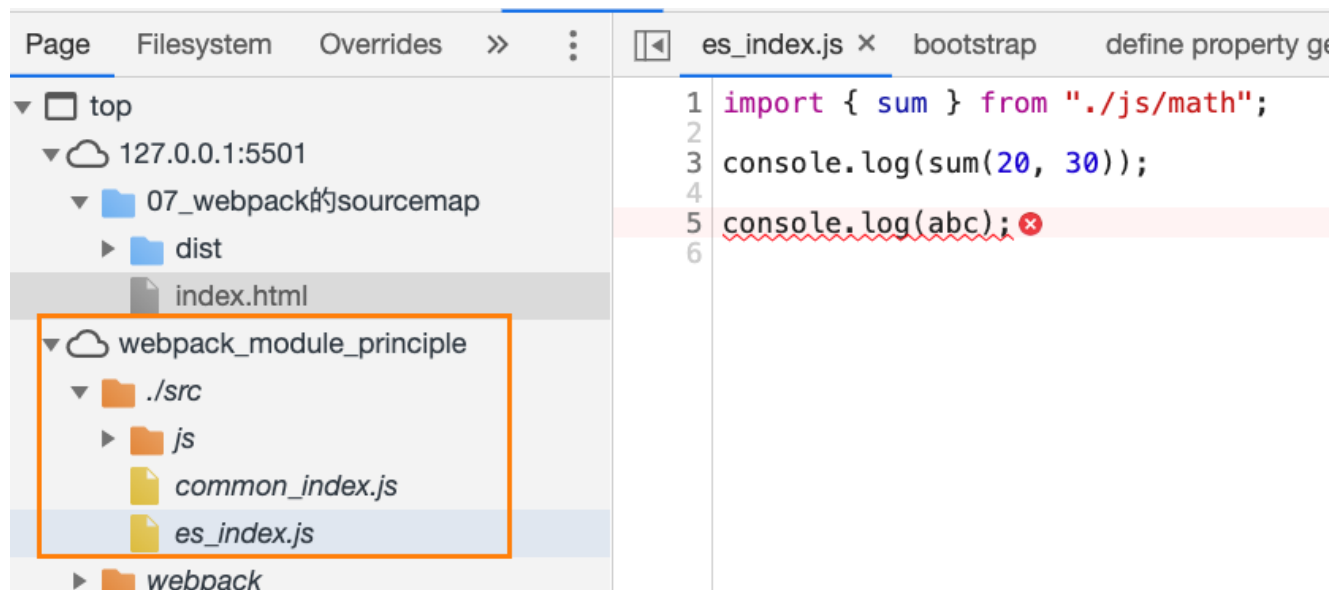
■ source-map :

□ 生成一个独立的source-map文件，并且在bundle文件中有一个注释，指向source-map文件；

■ bundle文件中有如下的注释：

□ 开发工具会根据这个注释找到source-map文件，并且解析；

```
// # sourceMappingURL=bundle.js.map
```



eval-source-map值

- **eval-source-map**：会生成sourcemap，但是source-map是以DataUrl添加到eval函数的后面

```
eval("const { dateFormat } = __webpack_require__(/*! ./js/format */ \"./src/js/format.js\");  
\nconst { sum } = __webpack_require__(/*! ./js/math */ \"./src/js/math.js\");\n\nconsole.log(  
dateFormat(\"aaaa\"));\nconsole.log(sum(20, 30));\n// # sourceURL=[module]\n// # sourceMappingURL=data:application/json;charset=utf-8;base64,  
eyJ2ZXJzaW9uIjozLCJzb3VyY2VzIjpbIndlbnBhY2s6Ly93ZWJwYWNRX21vZHVzZV9wcmluY2lwbgUvLi9zcmMvY29tbW9uX  
2luZGV4LmpzPzg0YjYiXSswibmFtZXMiOltdLCJtYXBwaW5ncyI6IkFBQUEstT0FBTyxhQUFhLEdBQUcsbUJBQU8sQ0FBQyx1Q0  
FBYTtBQUUM1QyxPQUFPLE1BQU0sR0FBRyxtQkFBTyxDQUFDLG1DQUFXOztBQUVuQztBQUUNBIiwilZmlsZSI6Ii4vc3JjL2NvbW1  
vbl9pbmRleC5qcy5qcyIsInNvdXJjZXNDb250ZW50IjpbImNvbnN0IHsgZGF0ZUZvcmlhdCB9ID0gcmlvdWlyZSgnLi9qcy9m  
b3JtYXQnKTtcbmNvbnN0IHsgc3VtIH0gPSByZXFlaXJlKFWiLi9qcy9tYXR0XCipO1xuXG5jb25zb2xlLmxvZyhkYXRlRm9yb  
WF0KFwiYWFhYVwiKSsk7XG5jb25zb2xlLmxvZyhzdW0oMjAsIDMwKSsk7XG4iXSswic291cmNlUm9vdCI6IiJ9\n// # sourceURL=webpack-internal:///./src/common_index.js\n");
```

inline-source-map值


- **inline-source-map** : 会生成sourcemap , 但是source-map是以DataUrl添加到bundle文件的后面

[illegible]

cheap-source-map


■ cheap-source-map :

- 会生成sourcemap , 但是会更加高效一些 (cheap低开销) , 因为它没有生成列映射 (Column Mapping)
- 因为在开发中 , 我们只需要行信息通常就可以定位到错误了



```
console.log(abc);
```

cheap-source-map的错误提示



```
console.log(abc);
```

source-map的错误提示

cheap-module-source-map值

■ cheap-module-source-map :

□ 会生成sourcemap , 类似于cheap-source-map , 但是对源自loader的sourcemap处理会更好。

■ 这里有一个很模糊的概念 : 对源自loader的sourcemap处理会更好 , 官方也没有给出很好的解释

□ 其实是如果loader对我们的源码进行了特殊的处理 , 比如babel ;

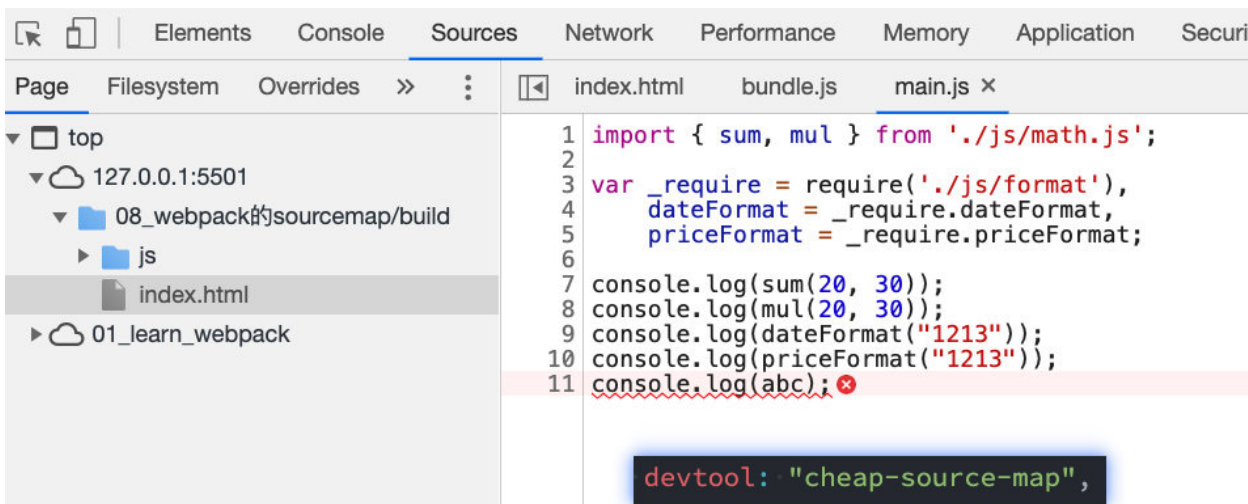
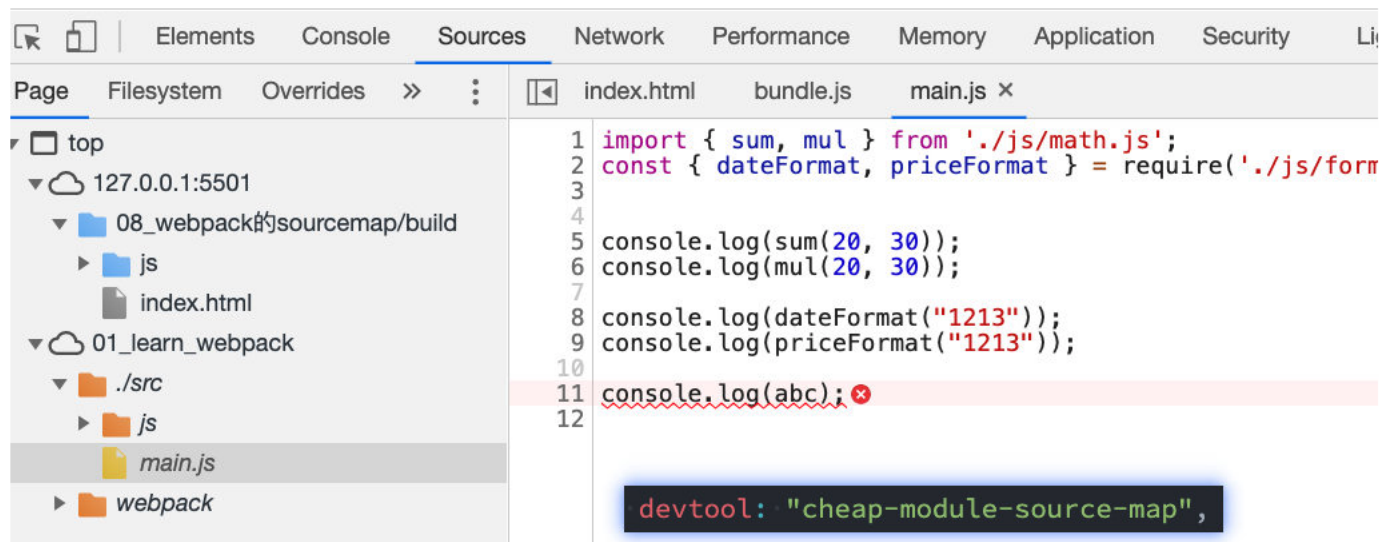
■ 如果我这里使用了babel-loader (注意 : 目前还没有讲babel)

□ 可以先按照我的babel配置演练 ;

```
rules: [
  {
    test: /\.js$/,
    use: {
      loader: "babel-loader",
      options: {
        presets: [
          ['@babel/preset-env', { targets: "defaults" }]
        ]
      }
    }
  }
]
```

cheap-source-map和cheap-module-source-map

■ cheap-source-map和cheap-module-source-map的区别：



hidden-source-map值

■ hidden-source-map :

- 会生成sourcemap , 但是不会对source-map文件进行引用 ;
- 相当于删除了打包文件中对sourcemap的引用注释 ;

```
// 被删除掉的  
//# sourceMappingURL=bundle.js.map
```

■ 如果我们手动添加进来 , 那么sourcemap就会生效了

nosources-source-map值

■ nosources-source-map :

□ 会生成sourcemap , 但是生成的sourcemap只有错误信息的提示 , 不会生成源代码文件 ;

■ 正确的错误提示 :

```
Uncaught ReferenceError: abc is not defined                                main.js:11
    at Object../src/main.js (main.js:11)
    at __webpack_require__ (bootstrap:18)
    at startup:3
    at startup:4
```

■ 点击错误提示 , 无法查看源码 :



多个值的组合

■ 事实上，webpack提供给我们的26个值，是可以进行多组合的。

■ 组合的规则如下：

□ inline-|hidden-|eval：三个值时三选一；

□ nosources：可选值；

□ cheap可选值，并且可以跟随module的值；

`[inline-|hidden-|eval-][nosources-][cheap-[module-]]source-map`

■ 那么在开发中，最佳的实践是什么呢？

□ 开发阶段：推荐使用 source-map或者cheap-module-source-map

✓ 这分别是vue和react使用的值，可以获取调试信息，方便快速开发；

□ 测试阶段：推荐使用 source-map或者cheap-module-source-map

✓ 测试阶段我们也希望在浏览器下看到正确的错误提示；

□ 发布阶段：false、缺省值（不写）