---

title: SurviveJs-Webpack中文翻译文档-05浏览器自动刷新.md date: 2016-10-13 16:38:16 author: Bing tags:

- Webpack
- 学习
- SurviveJs
- javascript categories: []

浏览器自动刷新

*LiveReload*或者*Browsersync*这些工具，让我们可以在开发中自动刷新浏览器，甚至可以在更改CSS后自动刷新浏览器。

开发环境中一个很好的尝试就是直接使用Webpack的**watch**模式。可以在启动webpack的时候用webpack --watch来配置。配置后，webpack会发现文件的更改并自动重新编译。另外一中方式就是使用webpack-dev-server,它也是以watch模式为基础，但是功能要更强大。

webpack-dev-server是一个运行在内存中的开发服务。当你更改你的应用后，将会更新内容。它也支持Webapck其它更先进的特性如热加载Hot Module Replacement (HMR)，采用对浏览器打补丁的方式而不是全部刷新。当与React技术结合起来的时候，就特别强大了。

> 特别需要提醒的是，*webpack-dev-server*只能用在开发环境，如果想启用正式的应用环境，建议采用其它标准的模式，如Apache或者Nginx。

> 一个叫做安全写入的IDE特性会在使用热加载时变得不可控。在创建基于热加载的架构时将该特性关闭是非常明智的做法。

开始使用*webpack-dev-server*

安装 *webpack-dev-server*:

```
npm i webpack-dev-server --save-dev
```

首先，这个命令将会在 npm bin目录下生成一个命令。你可以尝试用它启动 *webpack-dev-server*。最快速实现浏览器为我们的项目自动刷新的方式就是直接执行：

```
webpack-dev-server --inline
```

--inling,在inline模式下启动服务，将webpack-dev-server客户端加入到webpack入口文件的配置中。

添加webpack-dev-server到项目中

为了将*webpack-dev-server*整合到我们的项目中，我们可以按照前一章的想法，在pakage.json启动脚本script配置中定义一个新的命令: **package.json**

```
...
"scripts": {
leanpub-start-insert
  "start": "webpack-dev-server",
leanpub-end-insert
  "build": "webpack"
},
...
```

--inline将被添加在Webpack配置文件中。我喜欢尽量保持npm命令的简洁，将复杂的配置项都交给webpack配置文件。尽管这样要写更多的代码，，但是却更容易理解所发生的事情，而且维护起来也更简单。

现在不论是执行npm start还是npm run start,你都会看到如下结果：

```
> Webpack@1.0.0 start
/Users/bing/Documents/workspase/Webpack
> webpack-dev-server
```

```
 http://localhost:8080/webpack-dev-server/
webpack result is served from /
content is served from
/Users/bing/Documents/workspase/Webpack
Hash: 87c1e5f935ae062f6501
Version: webpack 1.13.2
```

这表示我们的开发服务已经运行了。打开http://localhost:8080/，你将看到和之前一样的hello world。

当你修改了你的代码，终端将会输出新的信息。但是，到目前为止，浏览器还不能捕获到更改进而不能自动刷新，只有当你手动刷新时，才会更新。接下来我们将解决这个问题。

> 如果你看到浏览器出现了一些错误，你可能需要更换一个端口，因为其它的应用程序可能已经占用了这个端口。你可以使用`netstat -na | grep 8080`来确认。若果有其他程序在使用8080端口，将会有信息展示出来。当然，不同的系统，可能命令有差别。

配置热加载

热加载搭建在*webpack-dev-server*之上，通过增加接口来实现对模块儿的动态扫描。例如：CSS加载器style-loader可以在不刷新的情况下更新css。因为它不关注应用的其它状态，所以能很好的处理CSS的热加载。

热加载也可以应用在JavaScript文件上，但是由于应用的状态，这将稍微负责一些。在**React配置（Configuring React）**章节我们会详细讲解。

我们可以使用webpack-dev-server --inline --hot来实现客户端的热加载。--hot可以使Webpack利用一个特定插件来将入口信息写到一个相关的javascript文件中。

热加载的配置项

为了可以更好的管理我们的配置文件，我将把如热加载这样的功能拆分到新的文件

以保持其独立性。这样使得webpack.config.js文件简洁且可复用。我们可以将这样的组合作为一个独立的npm包。我们甚至可以将它们 We could push a collection like this to a npm package of its own. We could even turn them into presets to use across projects. Functional composition allows that.

我将会提交所有的配置信息到libs.parts.js文件，在使用时再从这里引用。下面的就是热加载配置的部分：

libs/parts.js

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

leanpub-start-insert
const parts = require('./libs/parts');
leanpub-end-insert

...

// Detect how npm is run and branch based on that
switch(process.env.npm_lifecycle_event) {
  case 'build':
    config = merge(common, {});
    break;
  default:
leanpub-start-delete
    config = merge(common, {});
leanpub-end-delete
leanpub-start-insert
    config = merge(
      common,
      parts.devServer({
```

```
        // Customize host/port here if needed
        host: process.env.HOST,
        port: process.env.PORT
      })
    );
leanpub-end-insert
}

module.exports = validate(config);
```

啊哈，的确是比较多的代码，主要是因为我们希望让配置文件看起来容易理解，并可以在以后复用。幸运的是当我们在主配置文件中引入时就变的是很简单了。

weback.config.js

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


const parts = require('./libs/parts');


...

// Detect how npm is run and branch based on that
switch(process.env.npm_lifecycle_event) {
  case 'build':
    config = merge(common, {});
    break;
  default:

--    config = merge(common, {});
```

```
    config = merge(
      common,
      parts.devServer({
        // Customize host/port here if needed
        host: process.env.HOST,
        port: process.env.PORT
      })
    );

}
```

现在终端执行npm start,在浏览器中打开localhost:8080,再更改`app/component.js'文件。浏览器便会自动刷新。要注意的是，当你改变了javascript代码后，会硬（？）刷新浏览器。CSS的更改和应用更加简单利索，我们家辉在下一章讲解。

> *webpack-dev-server*对路径要求比较严。如果提供的include路径与系统不匹配，可能会操成错误。具体请参考***issue#675***

> 你也可以使用另外一个路径来访问应用;localhost:8080/webpack-dev-server/ 。它将会在应用的上方展示浏览器的相关操作信息。

在Windows,Ubuntu和Vagrant上配置热加载

上面的配置可能在一些Windows,Ubuntu和Vagrant环境上有问题。下面的配置也许可以解决：

libs/parts.js

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

exports.devServer = function(options) {
  return {
leanpub-start-insert
```

```
    watchOptions: {
      // Delay the rebuild after the first change
      aggregateTimeout: 300,
      // Poll using interval (in ms, accepts boolean too)
      poll: 1000
    },
leanpub-end-insert
    devServer: {
      ...
    },
    ...
  };
}
```

由于这个的配置会去轮询文件系统，所以在默认情况不好使时需要给予权限。

> 更多请见*** # 155***

通过网络连接其它机器的开发服务

如果你想访问另外一台机器上的开发服务，你可以通过配置ip和端口来实现。

Webpack-dev-server的其它用法

我们也可以使用Express自己搭建一个服务器，仅将webpack-dev-server作为一个***中间件(middleware)***。也可以使用Node.js API。这样你可以自己控制，并足够灵活。

> ***dotenv***可以让你通过一个.envfile文件来定义环境环境变量。在开发时比较方便。

> 请注意，CLI和Node.js API之间有细小的差别。所以有些人喜欢全部用Nde.js API。

小结

本章，学习了搭建一个可以自动刷新浏览器的Webpack框架。为了使其更完美的处理CSS文件，我们将在下一章更上一层楼！

> 相关博客可参考：
> http://www.cnblogs.com/hhhyaaon/p/5664002.html