第14章 开发工具

React 使用了若干的抽象层来帮助你更轻松地开发组件、推导程序的状态。然而，在调试、构建及分发应用时，这样的设计就会产生负面影响了。

幸运的是，我们拥有一些非常棒的开发工具能在开发及构建过程中为我们提供帮助。在本章中我们将探讨这些构建工具和调试工具，它们可以让开发 React 程序的过程更加高效。

构建工具

构建工具帮助你优化重复性的工作使运行代码更加轻松。在 React 程序开发中，最具重复性的工作之一就是对所有的 React 组件运行 JSX 解释器。另一复杂的任务是将所有模块打包成一个或多个文件以便分发到浏览器中使用。

让我们看看 React 是如何与两款流行的 JavaScript 构建工具——Browserify 和 Webpack ——一起工作的。

我们的示例程序问卷制作工具使用了 Browserify，因为它更善于打包 JavaScript 文件。同时，由于无须额外的配置文件，单独使用 Browserify 成了一件很简单的事。

> 我们有一个贯穿全书的示例项目，一个问卷制作工具，你可以在 *https://github.com/backstopmedia/bleeding-edge-sample-app* 阅读全部源码。

Browserify

Browserify 是一个 JavaScript 打包工具，支持在浏览器中使用 Node.js 风格的 `require()` 方法。不需要了解太多的细节也不必不知所措，Browserify 会自动将所有的依赖打包到一个文件中，以支持模块在浏览器环境中使用。任何包含 `require` 语句的 JavaScript 文件运行 Browserify 都会自动打包所有的依赖项。

尽管十分强大，Browserify 仅支持 JavaScript 文件，不像 Bower、Webpack 或其他打包工具支持多种文件格式。

建立一个 Browserify 项目

想要让 Broswerify 良好地运行起来，你必须初始化一个 node 项目。假设已经安装了 node 和 npm，你可以通过在终端运行下面的命令来初始化一个新项目。这个命令会创建一个含有必要资源的 `package.json` 文件。

```
npm init
# ... answer questions as necessary to complete init
npm install --save-dev browserify reactify react uglify-js
```

在 `package.json` 文件的末尾增加如下构建脚本：

```
    ...
    "devDependencies": {
      "browserify": "^5.11.2",
      "reactify": "^0.14.0",
      "react": "^0.11.1",
      "uglify-js": "^2.4.15"
    },
    "scripts": {
      "build": "browserify --debug index.js > bundle.js",
      "build-dist": "NODE_ENV=production browserify index.js | uglifyjs -m
       > bundle.min.js"
    },
    "browserify": {
      "transform": ["reactify"]
    }
}
```

通过运行 npm run build 来执行默认的任务，这个命令会创建一个打包好的 JavaScript 文件和对应的源代码映射文件（source map）。这样的配置能够让你像引用多个独立文件那样查看错误信息和添加断点，而实际上你只引用了一个文件。同时，你也会看到原来的 JSX 代码而不是被编译成原生 JavaScript 的版本。

对于构建生产环境的代码，我们需要指明当前是生产环境。React 使用了一个叫作 envify 的转换工具，当它和代码压缩工具如 uglify 一起使用时，可以移除所有的调试代码和详细的错误信息，以此来提升效率并缩减文件体积。

如果你想要使用一些 ES6 的特性，如箭头函数或类，你可以把 transform 那一行改成这样：

```
"transform": [["reactify", {"harmony": true}]]
```

现在你就可以写点 React 组件并将其打包了。

对代码做出修改

让我们创建一个名为 index.js 的 React + JSX 文件。

```
var React = require('react');
React.render(<h1>Hello World</h1>, document.body);
```

再增加一个简单的 index.html 文件：

```
<html>
  <head>
    <title>React + Browserify Demo</title>
  </head>
  <body>
    This text should not appear in the browser
    <script src="bundle.js"></script>
  </body>
</html>
```

现在你的项目结构看起来大致是这样的：

- index.html
- index.js
- node_modules/
- package.json

如果现在尝试打开 index.html 你会发现页面没有加载任何的 JavaScript，因为我们还没有打包出最终的文件。运行 `npm run build` 命令然后再刷新该页面，这个示例程序就能成功加载了。

Watchify

你可以选择增加一个监控（watch）任务，它对开发工作大有帮助。Watchify 是对 Browserify 的一个封装，当你改动了文件的时候它会自动帮你重新打包。同时 Watchify 还使用了缓存来加快重新打包的速度。

```
npm install --save-dev watchify
```

把下面这行添加到 package.json 中的 scripts 对象中。

```
"watch": "watchify --debug index.js -o bundle.js"
```

这样你就不再需要运行 `npm run build`，运行 `npm run watch` 即可，它能给你带来更流畅的开发体验。

构建

现在，你只需要简单运行一下构建命令就能将 React + JSX 代码打包到一个文件中供浏览器使用了。

```
npm run-script build
```

你会看到多了一个新的 bundle.js 文件。打开 bundle.js 你会发现在文件头部有一些被压缩过的 JavaScript 代码，后续跟着的是经过 JSX 转换的组件代码。这个文件包含了你在 index.js 中需要的所有依赖，它可以在浏览器中运行。再打开 index.html 你会发现一切都正常工作了。

Webpack

Webpack 和 Browserify 很像，它也会把你的 JavaScript 代码打包到一个文件中。老实说，把 Webpack 和 Browserify 放在一起进行对比是不公平的，因为 Webpack 有更多的特色功能。而这些功能在 Browserify 中是不怎么使用的。

Webpack 还能：

- 将 CSS、图片以及其他资源打包到同一个包中。
- 在打包之前对文件进行预处理（less、coffee、jsx 等）。

- 根据入口文件的不同把你的包拆分成多个包。
- 支持开发环境的特性标志位。
- 支持模块代码“热”替换。
- 支持异步加载。

因此，Webpack 能够实现 Browserify 混合其他构建工具如 gulp、grunt 的功能。

Webpack 是一个模块系统，通过增加或替换插件来实现功能。默认情况下，它启用了一个 CommonJS 解释器插件。

在这里我们不会详细介绍 Webpack 的每一种特性，不过我们会介绍基本的功能以及让它与 React 一起工作需要做的配置。

Webpack 与 React

React 帮助你开发应用程序组件。Webpack 不仅帮助你打包所有的 JavaScript 文件，还拥有其他所有应用需要的资源。这样的设计让你能创建一个自动包含所有类型依赖的组件。由于可以自动包含所有依赖，组件也变得更加方便移植。更妙的是，随着应用不断地开发并修改，当你移除某个组件的时候，它的所有依赖也会自动被移除。这意味着不会再有未被使用的 CSS 或图片遗留在代码目录中。

让我们看一下 React 组件是怎样加载资源依赖的。

```
//logo.js
require('./logo.css');

var React = require('react');

var Logo = React.createClass({
    render: function () {
      return <img className="Logo" src={require('./logo.png')} />
    }
});

module.exports = Logo;
```

我们需要一个应用的入口文件来打包这个组件。

```
//app.js
```

```
var React = require('react');
var Logo = require('./logo.js');

React.render(<Logo/>, document.body);
```

现在我们需要创建一个 Webpack 配置文件，以通知 Webpack 对不同的文件类型应该使用哪种加载器。同时，还要定义应用的入口文件以及打包后文件的存放位置。

```
//webpack.config.js
module.exports = {
  // 程序的入口文件
  entry: './app.js',
  output: {
    // 所有打包好的资源的存放位置
    path: './public/build',

    // 使用 url-loader 的资源的前缀
    publicPath: './build/',

    // 生成的打包文件名
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      {
        // 用于匹配加载器支持的文件格式的正则表达式
        test: /\.(js)$/,

        // 要使用的加载器类型
        // 加载器支持通过查询字符串的方式接收参数
        loader: 'jsx-loader?harmony'
      },
      {
        test: /\.(css)$/,

        // 多个加载器通过“!”连接
        loader: 'style-loader!css-loader'
```

```
      },
      {
        test: /\.(png|jpg)$/,

        // url-loader 支持 base64 编码的行内资源
        loader: 'url-loader?size=8192'
      }
    ]
  }
};
```

现在，你需要安装 Webpack 及一系列加载器。你可以选择在控制台使用 npm 或修改 package.json 来完成安装。

确保你把这些加载器安装到了本地，而不是全局（使用 -g 参数）。

```
npm install webpack react
npm install url-loader jsx-loader style-loader css-loader
```

当所有的准备工作完成后，运行 Webpack：

```
// 在开发环境构建一次
webpack

// 构建并生成源代码映射文件
webpack -d

// 在生成环境构建，压缩、混淆代码，并移除无用代码
webpack -p

// 快速增量构建，可以和其他选项一起使用
webpack --watch
```

调试工具

无论你多么小心，总是会犯这样那样的错误。我们不会讨论如何调试 JavaScript，但会提到一些让调试 React 应用更加简单的工具。

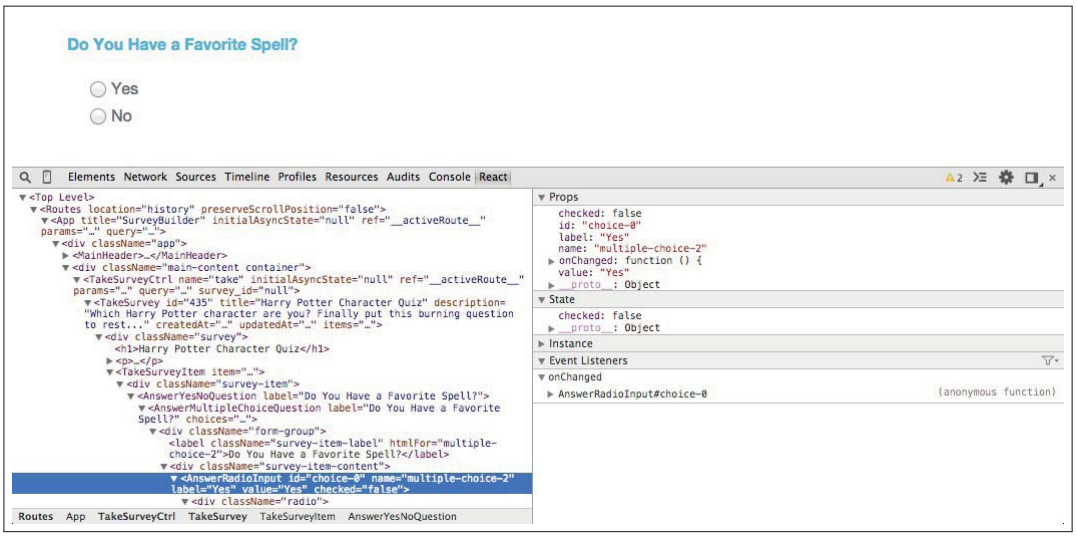
基础工具

对于本章的内容，打开 Chrome 并安装 React Developer Tool 扩展。另外，还需要设置 window.React。

```
window.React = require('react');
```

在元素处右击，并选择审查元素。你会看到在 Elements 面板显示着熟悉的 DOM 结构。

不过你不是来看 DOM 结构的。你想要看到的是组件，还有它们的 props 及 state。如果完成了上面的配置工作，你应该可以在面板列表的最右边看到名为 React 的面板。



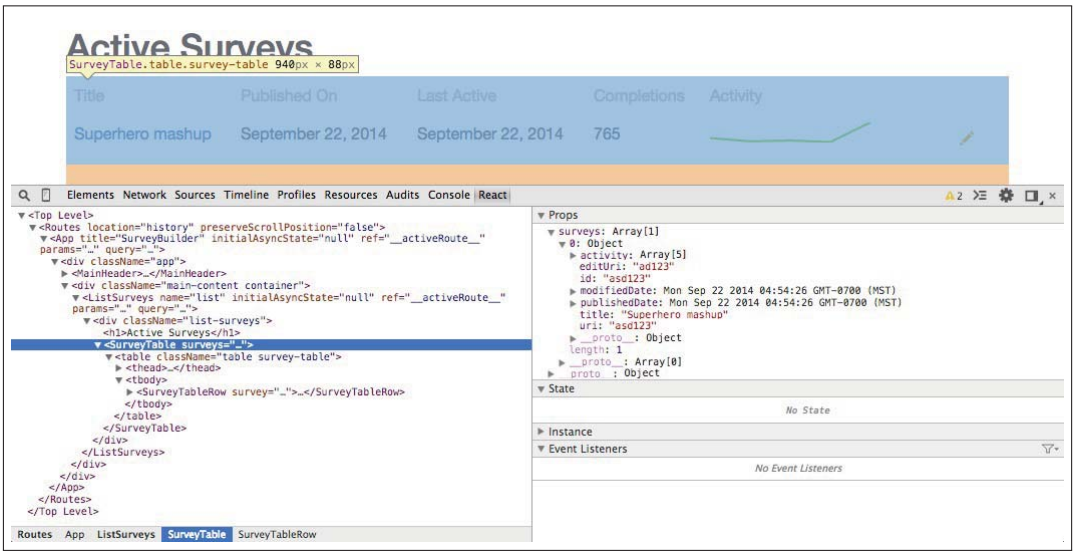
displayName

JSX 转换器会尝试猜测组件的 displayName 并插入到组件中。如果你没有使用 JSX，那么你应该手动给组件添加一个 displayName。

```
React.createClass({displayName: "MyComponent", ...});
```

组件的层级结构显示在左边，而选中组件的信息在右边。

只看这些信息就能告诉你很多关于 React 组件的 state、props 以及事件处理器的信息。你会看到 onDragStart 事件的处理器是 ModuleButton::handleDragStart，还有按钮拥有的 class 以及其他你感兴趣的内容。React 开发者工具的能力不止于此。



你能看到一组包含了时间戳的问卷（survey）数据被传入了 SurveyTab 组件。它们以更友好的方式展现在屏幕上。双击某个时间戳并输入一个新的值，组件会自动更新并重新渲染。

开发者工具能帮你缩小问题范围，帮助团队中的新成员找到完成任务需要修改的组件。

JSBin 与 JSFiddle

在调试或只是在头脑风暴时，在线调试站点如 JSFiddle 及 JSBin 是很好的资源。在求助或与他人分享原型时，可以用它们来创建测试用例。

总结

现在，你已经见识到了在开发 React 程序时提供给你的调试及构建工具的好处。下一章，我们会详细谈谈如何在 React 中使用自动测试。