这个快速上手指南将会教你如何将TypeScript和React还有webpack连结在一起使用。

我们假设已经在使用Node.js和npm。

初始化项目结构

让我们新建一个目录。
将会命名为`proj`，但是你可以改成任何你喜欢的名字。

```
mkdir proj
cd proj
```

我们会像下面的结构组织我们的工程：

```
proj/
├─ dist/
└─ src/
   └─ components/
```

TypeScript文件会放在`src`文件夹里，通过TypeScript编译器编译，然后经webpack处理，最后生成一个`bundle.js`文件放在`dist`目录下。
我们自定义的组件将会放在`src/components`文件夹下。

下面来创建基本结构：

```
mkdir src
cd src
mkdir components
cd ..
```

Webpack会帮助我们生成`dist`目录。

初始化工程

现在把这个目录变成npm包。

```
npm init
```

你会看到一些提示。
你可以使用默认项除了开始脚本。
当然，你也可以随时到生成的`package.json`文件里修改。

安装依赖

首先确保已经全局安装了Webpack。

```
npm install -g webpack
```

Webpack这个工具可以将你的所有代码和可选择地将依赖捆绑成一个单独的`.js`文件。

现在我们添加React和React-DOM以及它们的声明文件到`package.json`文件里做为依赖：

```
npm install --save react react-dom @types/react @types/react-dom
```

使用`@types/`前缀表示我们额外要获取React和React-DOM的声明文件。
通常当你导入像`"react"`这样的路径，它会查看`react`包；
然而，并不是所有的包都包含了声明文件，所以TypeScript还会查看`@types/react`包。
你会发现我们以后将不必在意这些。

接下来，我们要添加开发时依赖awesome-typescript-loader和source-map-loader。

```
npm install --save-dev typescript awesome-typescript-loader source-map-loader
```

这些依赖会让TypeScript和webpack在一起良好地工作。
awesome-typescript-loader可以让Webpack使用TypeScript的标准配置文件`tsconfig.json`编译TypeScript代码。
source-map-loader使用TypeScript输出的sourcemap文件来告诉webpack何时生成*自己的* sourcemaps。
这就允许你在调试最终生成的文件时就好像在调试TypeScript源码一样。

注意我们安装TypeScript为一个开发依赖。
我们还可以使用`npm link typescript`来链接TypeScript到一个全局拷贝，但这不是常见用法。

添加TypeScript配置文件

我们想将TypeScript文件整合到一起 - 这包括我们写的源码和必要的声明文件。

我们需要创建一个`tsconfig.json`文件，它包含了输入文件列表以及编译选项。
在工程根目录下新建文件`tsconfig.json`文件，添加以下内容：

```
{
    "compilerOptions": {
        "outDir": "./dist/",
        "sourceMap": true,
        "noImplicitAny": true,
        "module": "commonjs",
        "target": "es5",
        "jsx": "react"
    },
    "include": [
        "./src/**/*"
    ]
}
```

你可以在这里了解更多关于`tsconfig.json`文件的说明。

写些代码

下面使用React写一段TypeScript代码。
首先，在`src/components`目录下创建一个名为`Hello.tsx`的文件，代码如下：

```
import * as React from "react";

export interface HelloProps { compiler: string; framework: string; }

export const Hello = (props: HelloProps) => <h1>Hello from {props.compiler} and
```

注意这个例子使用了[无状态的功能组件]，我们可以让它更像一点*类*。

```
import * as React from "react";

export interface HelloProps { compiler: string; framework: string; }

// 'HelloProps' describes the shape of props.
// State is never set so we use the 'undefined' type.
export class Hello extends React.Component<HelloProps, undefined> {
    render() {
        return <h1>Hello from {this.props.compiler} and {this.props.framework}!
    }
}
```

接下来，在`src`下创建`index.tsx`文件，源码如下：

```
import * as React from "react";
import * as ReactDOM from "react-dom";

import { Hello } from "./components/Hello";

ReactDOM.render(
    <Hello compiler="TypeScript" framework="React" />,
    document.getElementById("example")
);
```

我们仅仅将`Hello`组件导入`index.tsx`。
注意，不同于`"react"`或`"react-dom"`，我们使用`index.tsx`的*相对路径* - 这很重要。
如果不这样做，TypeScript只会尝试在`node_modules`文件夹里查找。

我们还需要一个页面来显示`Hello`组件。
在根目录`proj`创建一个名为`index.html`的文件，如下：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Hello React!</title>
    </head>
    <body>
        <div id="example"></div>

        <!-- Dependencies -->
        <script src="./node_modules/react/dist/react.js"></script>
        <script src="./node_modules/react-dom/dist/react-dom.js"></script>

        <!-- Main -->
        <script src="./dist/bundle.js"></script>
    </body>
</html>
```

需要注意一点我们是从node_modules引入的文件。
React和React-DOM的npm包里包含了独立的.js文件，你可以在页面上引入它们，这里我们为了快捷就直接引用了。
可以随意地将它们拷贝到其它目录下，或者从CDN上引用。
Facebook在CND上提供了一系列可用的React版本，你可以在这里查看更多内容。

创建一个webpack配置文件

在工程根目录下创建一个webpack.config.js文件。

```
module.exports = {
    entry: "./src/index.tsx",
    output: {
        filename: "bundle.js",
        path: __dirname + "/dist"
    },

    // Enable sourcemaps for debugging webpack's output.
    devtool: "source-map",

    resolve: {
        // Add '.ts' and '.tsx' as resolvable extensions.
        extensions: [".ts", ".tsx", ".js", ".json"]
    },

    module: {
        rules: [
            // All files with a '.ts' or '.tsx' extension will be handled by 'a
            { test: /\.tsx?$/, loader: "awesome-typescript-loader" },

            // All output '.js' files will have any sourcemaps re-processed by
            { enforce: "pre", test: /\.js$/, loader: "source-map-loader" }
        ]
    },

    // When importing a module whose path matches one of the following, just
    // assume a corresponding global variable exists and use that instead.
    // This is important because it allows us to avoid bundling all of our
    // dependencies, which allows browsers to cache those libraries between bui
    externals: {
        "react": "React",
        "react-dom": "ReactDOM"
    },
};
```

大家可能对externals字段有所疑惑。
我们想要避免把所有的React都放到一个文件里，因为会增加编译时间并且浏览器还能够缓存没有发生改变的库文件。

理想情况下，我们只需要在浏览器里引入React模块，但是大部分浏览器还没有支持模块。
因此大部分代码库会把自己包裹在一个单独的全局变量内，比如：jQuery或_。
这叫做“命名空间”模式，webpack也允许我们继续使用通过这种方式写的代码库。
通过我们的设置"react": "React"，webpack会神奇地将所有对"react"的导入转换成从

`React`全局变量中加载。

你可以在这里了解更多如何配置webpack。

整合在一起

执行：

```
webpack
```

在浏览器里打开`index.html`，工程应该已经可以用了！
你可以看到页面上显示着“Hello from TypeScript and React!”