这篇快速上手指南将教你如何使用Gulp构建TypeScript，和如何在Gulp管道里添加Browserify，uglify或Watchify。
它还包涵了Babel的功能，通过使用Babelify。

这里假设你已经在使用Node.js和npm了。

创建简单工程

我们首先创建一个新目录。
命名为`proj`，也可以使用任何你喜欢的名字。

```
mkdir proj
cd proj
```

我们将以下面的结构开始我们的工程：

```
proj/
   ├─ src/
   └─ dist/
```

TypeScript文件放在`src`文件夹下，经过TypeScript编译器编译生成的目标文件放在`dist`目录下。

下面让我们来创建这些文件夹：

```
mkdir src
mkdir dist
```

初始化工程

现在让我们把这个文件夹转换成npm包：

```
npm init
```

你将看到有一些提示操作。
除了入口文件外，其余的都可以使用默认项。
入口文件使用`./dist/main.js`。
你可以随时在`package.json`文件里更改生成的配置。

安装依赖项

现在我们可以使用`npm install`命令来安装包。
首先全局安装TypeScript和Gulp。
（如果你正在使用Unix系统，你可能需要使用`sudo`命令来启动`npm install`命令行。）

```
npm install -g gulp-cli
```

然后安装`typescript`，`gulp`和`gulp-typescript`到开发依赖项。
Gulp-typescript是TypeScript的一个Gulp插件。

```
npm install --save-dev typescript gulp gulp-typescript
```

写一个简单的例子

让我们写一个Hello World程序。
在src目录下创建main.ts文件：

```
function hello(compiler: string) {
    console.log(`Hello from ${compiler}`);
}
hello("TypeScript");
```

在工程的根目录proj下新建一个tsconfig.json文件：

```
{
    "files": [
        "src/main.ts"
    ],
    "compilerOptions": {
        "noImplicitAny": true,
        "target": "es5"
    }
}
```

新建gulpfile.js文件

在工程根目录下，新建一个gulpfile.js文件：

```
var gulp = require("gulp");
var ts = require("gulp-typescript");
var tsProject = ts.createProject("tsconfig.json");

gulp.task("default", function () {
    return tsProject.src()
        .pipe(tsProject())
        .js.pipe(gulp.dest("dist"));
});
```

测试这个应用

```
gulp
node dist/main.js
```

程序应该能够打印出“Hello from TypeScript!”。

向代码里添加模块

在使用Browserify前，让我们先构建一下代码然后再添加一些混入的模块。
这个结构将是你在真实应用程序中会用到的。

新建一个src/greet.ts文件：

```
export function sayHello(name: string) {
```

```
    return `Hello from ${name}`;
}
```

更改`src/main.ts`代码，从`greet.ts`导入`sayHello`：

```
import { sayHello } from "./greet";

console.log(sayHello("TypeScript"));
```

最后，将`src/greet.ts`添加到`tsconfig.json`：

```
{
    "files": [
        "src/main.ts",
        "src/greet.ts"
    ],
    "compilerOptions": {
        "noImplicitAny": true,
        "target": "es5"
    }
}
```

确保执行`gulp`后模块是能工作的，在Node.js下进行测试：

```
gulp
node dist/main.js
```

注意，即使我们使用了ES2015的模块语法，TypeScript还是会生成Node.js使用的CommonJS模块。
我们在这个教程里会一直使用CommonJS模块，但是你可以通过修改`module`选项来改变这个行为。

Browserify

现在，让我们把这个工程由Node.js环境移到浏览器环境里。
因此，我们将把所有模块捆绑成一个JavaScript文件。
所幸，这正是Browserify要做的事情。
更方便的是，它支持Node.js的CommonJS模块，这也正是TypeScript默认生成的类型。
也就是说TypeScript和Node.js的设置不需要改变就可以移植到浏览器里。

首先，安装Browserify，tsify和vinyl-source-stream。
tsify是Browserify的一个插件，就像gulp-typescript一样，它能够访问TypeScript编译器。
vinyl-source-stream会将Browserify的输出文件适配成gulp能够解析的格式，它叫做vinyl。

```
npm install --save-dev browserify tsify vinyl-source-stream
```

新建一个页面

在`src`目录下新建一个`index.html`文件：

```
<!DOCTYPE html>
<html>
    <head>
```

```
        <meta charset="UTF-8" />
        <title>Hello World!</title>
    </head>
    <body>
        <p id="greeting">Loading ...</p>
        <script src="bundle.js"></script>
    </body>
</html>
```

修改main.ts文件来更新这个页面：

```
import { sayHello } from "./greet";

function showHello(divName: string, name: string) {
    const elt = document.getElementById(divName);
    elt.innerText = sayHello(name);
}

showHello("greeting", "TypeScript");
```

showHello调用sayHello函数更改页面上段落的文字。
现在修改gulpfile文件如下：

```
var gulp = require("gulp");
var browserify = require("browserify");
var source = require('vinyl-source-stream');
var tsify = require("tsify");
var paths = {
    pages: ['src/*.html']
};

gulp.task("copy-html", function () {
    return gulp.src(paths.pages)
        .pipe(gulp.dest("dist"));
});

gulp.task("default", ["copy-html"], function () {
    return browserify({
        basedir: '.',
        debug: true,
        entries: ['src/main.ts'],
        cache: {},
        packageCache: {}
    })
    .plugin(tsify)
    .bundle()
    .pipe(source('bundle.js'))
    .pipe(gulp.dest("dist"));
});
```

这里增加了copy-html任务并且把它加作default的依赖项。
这样，当default执行时，copy-html会被首先执行。
我们还修改了default任务，让它使用tsify插件调用Browserify，而不是gulp-typescript。
方便的是，两者传递相同的参数对象到TypeScript编译器。

调用bundle后，我们使用source（vinyl-source-stream的别名）把输出文件命名为bundle.js。

测试此页面，运行gulp，然后在浏览器里打开dist/index.html。
你应该能在页面上看到“Hello from TypeScript”。

注意，我们为Broswerify指定了debug: true。
这会让tsify在输出文件里生成source maps。
source maps允许我们在浏览器中直接调试TypeScript源码，而不是在合并后的JavaScript文件上调试。
你要打开调试器并在main.ts里打一个断点，看看source maps是否能工作。
当你刷新页面时，代码会停在断点处，从而你就能够调试greet.ts。

Watchify，Babel和Uglify

现在代码已经用Browserify和tsify捆绑在一起了，我们可以使用Browserify插件为构建添加一些特性。

- Watchify启动Gulp并保持运行状态，当你保存文件时自动编译。
  帮你进入到编辑-保存-刷新浏览器的循环中。

- Babel是个十分灵活的编译器，将ES2015及以上版本的代码转换成ES5和ES3。
  你可以添加大量自定义的TypeScript目前不支持的转换器。

- Uglify帮你压缩代码，将花费更少的时间去下载它们。

Watchify

我们启动Watchify，让它在后台帮我们编译：

```
npm install --save-dev watchify gulp-util
```

修改gulpfile文件如下：

```
var gulp = require("gulp");
var browserify = require("browserify");
var source = require('vinyl-source-stream');
var watchify = require("watchify");
var tsify = require("tsify");
var gutil = require("gulp-util");
var paths = {
    pages: ['src/*.html']
};

var watchedBrowserify = watchify(browserify({
    basedir: '.',
    debug: true,
    entries: ['src/main.ts'],
    cache: {},
    packageCache: {}
}).plugin(tsify));

gulp.task("copy-html", function () {
    return gulp.src(paths.pages)
        .pipe(gulp.dest("dist"));
});
```

```
function bundle() {
    return watchedBrowserify
        .bundle()
        .pipe(source('bundle.js'))
        .pipe(gulp.dest("dist"));
}

gulp.task("default", ["copy-html"], bundle);
watchedBrowserify.on("update", bundle);
watchedBrowserify.on("log", gutil.log);
```

共有三处改变，但是需要你略微重构一下代码。

1. 将`browserify`实例包裹在`watchify`的调用里，控制生成的结果。
2. 调用`watchedBrowserify.on("update", bundle);`，每次TypeScript文件改变时Browserify会执行`bundle`函数。
3. 调用`watchedBrowserify.on("log", gutil.log);`将日志打印到控制台。

(1)和(2)在一起意味着我们要将`browserify`调用移出`default`任务。
然后给函数起个名字，因为Watchify和Gulp都要调用它。
(3)是可选的，但是对于调试来讲很有用。

现在当你执行`gulp`，它会启动并保持运行状态。
试着改变`main.ts`文件里`showHello`的代码并保存。
你会看到这样的输出：

```
proj$ gulp
[10:34:20] Using gulpfile ~/src/proj/gulpfile.js
[10:34:20] Starting 'copy-html'...
[10:34:20] Finished 'copy-html' after 26 ms
[10:34:20] Starting 'default'...
[10:34:21] 2824 bytes written (0.13 seconds)
[10:34:21] Finished 'default' after 1.36 s
[10:35:22] 2261 bytes written (0.02 seconds)
[10:35:24] 2808 bytes written (0.05 seconds)
```

Uglify

首先安装Uglify。
因为Uglify是用于混淆你的代码，所以我们还要安装vinyl-buffer和gulp-sourcemaps来支持sourcemaps。

```
npm install --save-dev gulp-uglify vinyl-buffer gulp-sourcemaps
```

修改gulpfile文件如下：

```
var gulp = require("gulp");
var browserify = require("browserify");
var source = require('vinyl-source-stream');
var tsify = require("tsify");
var uglify = require('gulp-uglify');
var sourcemaps = require('gulp-sourcemaps');
var buffer = require('vinyl-buffer');
var paths = {
```

```
    pages: ['src/*.html']
};

gulp.task("copy-html", function () {
    return gulp.src(paths.pages)
        .pipe(gulp.dest("dist"));
});

gulp.task("default", ["copy-html"], function () {
    return browserify({
        basedir: '.',
        debug: true,
        entries: ['src/main.ts'],
        cache: {},
        packageCache: {}
    })
    .plugin(tsify)
    .bundle()
    .pipe(source('bundle.js'))
    .pipe(buffer())
    .pipe(sourcemaps.init({loadMaps: true}))
    .pipe(uglify())
    .pipe(sourcemaps.write('./'))
    .pipe(gulp.dest("dist"));
});
```

注意`uglify`只是调用了自己—`buffer`和`sourcemaps`的调用是用于确保sourcemaps可以工作。
这些调用让我们可以使用单独的sourcemap文件，而不是之前的内嵌的sourcemaps。
你现在可以执行`gulp`来检查`bundle.js`是否被压缩了：

```
gulp
cat dist/bundle.js
```

Babel

首先安装Babelify和ES2015的Babel预置程序。
和Uglify一样，Babelify也会混淆代码，因此我们也需要vinyl-buffer和gulp-sourcemaps。
默认情况下Babelify只会处理扩展名为`.js`，`.es`，`.es6`和`.jsx`的文件，因此我们需要添加`.ts`扩展名到Babelify选项。

```
npm install --save-dev babelify babel-preset-es2015 vinyl-buffer gulp-sourcemap
```

修改gulpfile文件如下：

```
var gulp = require('gulp');
var browserify = require('browserify');
var source = require('vinyl-source-stream');
var tsify = require('tsify');
var sourcemaps = require('gulp-sourcemaps');
var buffer = require('vinyl-buffer');
var paths = {
    pages: ['src/*.html']
};

gulp.task('copyHtml', function () {
    return gulp.src(paths.pages)
```

```
        .pipe(gulp.dest('dist'));
});

gulp.task('default', ['copyHtml'], function () {
    return browserify({
        basedir: '.',
        debug: true,
        entries: ['src/main.ts'],
        cache: {},
        packageCache: {}
    })
    .plugin(tsify)
    .transform('babelify', {
        presets: ['es2015'],
        extensions: ['.ts']
    })
    .bundle()
    .pipe(source('bundle.js'))
    .pipe(buffer())
    .pipe(sourcemaps.init({loadMaps: true}))
    .pipe(sourcemaps.write('./'))
    .pipe(gulp.dest('dist'));
});
```

我们需要设置TypeScript目标为ES2015。
Babel稍后会从TypeScript生成的ES2015代码中生成ES5。
修改`tsconfig.json`:

```
{
    "files": [
        "src/main.ts"
    ],
    "compilerOptions": {
        "noImplicitAny": true,
        "target": "es2015"
    }
}
```

对于这样一段简单的代码来说，Babel的ES5输出应该和TypeScript的输出相似。