现在我们已经按照指南里的步骤写好一个声明文件，是时候把它发布到npm了。
有两种主要方式用来发布声明文件到npm：

1. 与你的npm包捆绑在一起，或
2. 发布到npm上的@types organization。

如果你能控制要使用你发布的声明文件的那个npm包的话，推荐第一种方式。
这样的话，你的声明文件与JavaScript总是在一起传递。

包含声明文件到你的npm包

如果你的包有一个主`.js`文件，你还是需要在`package.json`里指定主声明文件。
设置`types`属性指向捆绑在一起的声明文件。
比如：

```
{
    "name": "awesome",
    "author": "Vandelay Industries",
    "version": "1.0.0",
    "main": "./lib/main.js",
    "types": "./lib/main.d.ts"
}
```

注意`"typings"`与`"types"`具有相同的意义，也可以使用它。

同样要注意的是如果主声明文件名是`index.d.ts`并且位置在包的根目录里（与`index.js`并列），你就不需要使用`"types"`属性指定了。

依赖

所有的依赖是由npm管理的。
确保所依赖的声明包都在`package.json`的`"dependencies"`里指明了
比如，假设我们写了一个包它依赖于Browserify和TypeScript。

```
{
    "name": "browserify-typescript-extension",
    "author": "Vandelay Industries",
    "version": "1.0.0",
    "main": "./lib/main.js",
    "types": "./lib/main.d.ts",
    "dependencies": [
        "browserify@latest",
        "@types/browserify@latest",
        "typescript@next"
    ]
}
```

这里，我们的包依赖于`browserify`和`typescript`包。
`browserify`没有把它的声明文件捆绑在它的npm包里，所以我们需要依赖于`@types/browserify`得到它的声明文件。
`typescript`相反，它把声明文件放在了npm包里，因此我们不需要依赖额外的包。

我们的包要从这两个包里暴露出声明文件，因此`browserify-typescript-extension`的用户也需要这些依赖。
正因此，我们使用`"dependencies"`而不是`"devDependencies"`，否则用户将需要手动安装那些包。
如果我们只是在写一个命令行应用，并且我们的包不会被当做一个库使用的话，那么我就可以使用`devDependencies`。

危险信号

`/// <reference path="..." />`

*不要*在声明文件里使用`/// <reference path="..." />`。

```
/// <reference path="../typescript/lib/typescriptServices.d.ts" />
....
```

*应该*使用`/// <reference types="..." />`代替

```
/// <reference types="typescript" />
....
```

务必阅读[使用依赖](./Library Structures.md#consuming-dependencies)一节了解详情。

打包所依赖的声明

如果你的类型声明依赖于另一个包：

- *不要*把依赖的包放进你的包里，保持它们在各自的文件里。
- *不要*将声明拷贝到你的包里。
- *应该*依赖于npm类型声明包，如果依赖包没包含它自己的声明的话。

公布你的声明文件

在发布声明文件包之后，确保在DefinitelyTyped外部包列表里面添加一条引用。
这可以让查找工具知道你的包提供了自己的声明文件。

发布到@types

@types下面的包是从DefinitelyTyped里自动发布的，通过types-publisher工具。
如果想让你的包发布为@types包，提交一个pull request
到https://github.com/DefinitelyTyped/DefinitelyTyped。
在这里查看详细信息contribution guidelines page。