08 Dependency Typings

Until now we've seen that our dependencies don't get typed automatically. In the previous sections we've seen a workaround in the form of installir
the node_modules folder (using "npm install" or manually), but there are two problems with that:

1. Installing into the node_modules only works for npm dependencies. If the dependency comes straight from Github or a custom repository
   installed using "npm install"
2. Some dependencies (like Redux) do not come with their definition files (d.ts) either because they weren't created in TypeScript or becaus
   process doesn't produce these definitions

To handle these cases there is another technique described below. Remember that this is only required for the IDE and standalone TypeScript co
these definitions. JSPM doesn't need them when compiling our files or bundling our app.

The Typings Project

The Typing project was original created in the name "Definitely Typed" and its purpose was to solve the exact problems listed above, especially be
supporting npm packages. It contains a huge registry for known libraries and includes a tool for creating d.ts for unknown or private libraries. The l
significant breaking changes, thus changing its name and a new repository that is now called Typings (https://github.com/typings/typings). Read he
differences between TSD and Typings: https://github.com/typings/typings/issues/72.

We will use the Typings tool to solve the problems listed above and find a way to do that in a reproducible manner.

Install Typings globally using npm:

```
npm install typings -g
```

Using the Typings Registry

We have two known libraries that don't have a d.ts. These are **redux** and **redux-thunk**. The original redux project (https://github.com/rackt/redux/)
and so it redux-thunk (https://github.com/gaearon/redux-thunk).

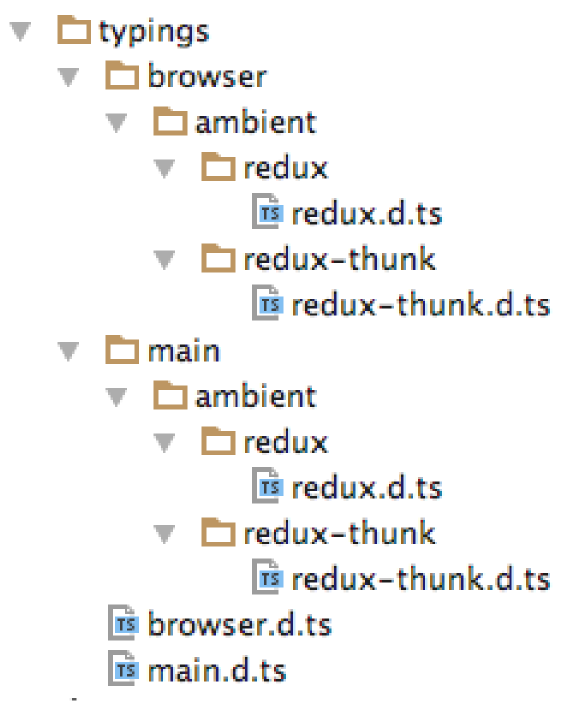
Fortunately, both of these libraries exist on the DefinitelyTyped registry:

- https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/redux
- https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/redux-thunk

Install these library definitions using the following command:

```
typings install redux --ambient
typings install redux-thunk --ambient
```

You should now have a new folder in your project called "typings". Inside you'll find two versions of the definitions, for both libraries, one for the bro
server (using node.js):



```
typings
  browser
    ambient
      redux
        redux.d.ts
      redux-thunk
        redux-thunk.d.ts
  main
    ambient
      redux
        redux.d.ts
      redux-thunk
        redux-thunk.d.ts
  browser.d.ts
  main.d.ts
```

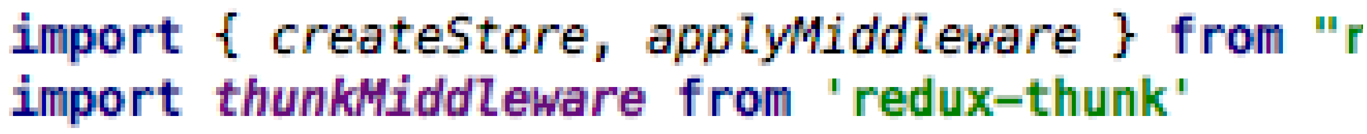
Please avoid committing these to our repository. Make sure you add the "typings" folder to your .gitignore file.

When compiling the project using tsc we need to tell it to ignore the "main" typings, otherwise TypeScript would complain about duplicate typings. I
"main" to our tsconfig.json exclusion list:

tsconfig.json

```
"exclude": [
  "jspm_packages",
  "node_modules",
  "dist",
  "typings/main.d.ts",
  "typings/main"
]
```

Now, you can look at your TypeScript files that use redux or redux-think and your IDE should be happier.



```
import { createStore, applyMiddleware } from "r
import thunkMiddleware from 'redux-thunk'
```

Installing Typings for Custom Dependencies

The other dependency we have is on **angular2-redux**. This is a little library created by Infomedia and published to npm: https://www.npmjs.com/pa
. Even though it's a library that comes with d.ts files it's not recognised because it's in a custom location (under jspm/npm/...). Even though we can
learn how to include its typings as part of the set of typings we install anyway for other dependencies.

When installing a library in a custom location inside our project we need to tell typings where to find it and how to name it:

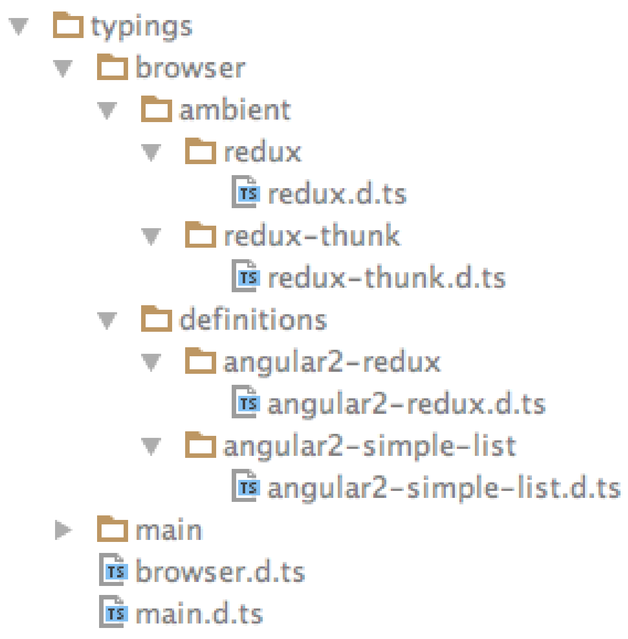
```
typings install file:jspm_packages/npm/angular2-redux@1.0.8/dist/index.d.ts --name angular2
```

Notice that the version is important here as this is the actual directory where this library resides in your JSPM folder dependency tree.

Let's also do that for the angular2-simple-list:

```
typings install file:jspm_packages/github/InfomediaLtd/angular2-simple-list@master/app/inde
--name angular2-simple-list
```

The typings directory should now have definitions for all the above dependencies:


```
typings
  browser
    ambient
      redux
        redux.d.ts
      redux-thunk
        redux-thunk.d.ts
    definitions
      angular2-redux
        angular2-redux.d.ts
      angular2-simple-list
        angular2-simple-list.d.ts
  main
  browser.d.ts
  main.d.ts
```

Configurable Typings Installation and Dealing with Version Changes

This method is not ideal because it required an extra step when dependencies change. It also requires knowing which dependencies need typings
them.

The Typings tool allows creation of a typings.json file with all the information required for installing typings. You can drop a --save at the end of the
will then add the required installation information to this file. Then, when running "typings install" it will look up that info and perform the appropriate
all dependencies specified in there. This solves the problem of knowing which dependencies need typings and how to install then - it can be done
as we add it to JSPM.

Delete the typings folder and run the following commands one by one:

```
typings install redux --ambient --save
typings install redux-thunk --ambient --save
typings install file:jspm_packages/npm/angular2-redux@1.0.8/dist/index.d.ts --name angular2
--save
typings install file:jspm_packages/github/InfomediaLtd/angular2-simple-list@master/app/inde
--name angular2-simple-list --save
```

You should now see a typings.json file in the top directory of your project:

typings.json

```
{
    "devDependencies": {
        "angular2-redux": "file:jspm_packages/npm/angular2-redux@1.0.8/dist/index.d.ts",
        "angular2-simple-list":
"file:jspm_packages/github/InfomediaLtd/angular2-simple-list@master/app/index.d.ts"
    },
    "ambientDevDependencies": {
        "redux": "github:DefinitelyTyped/DefinitelyTyped/redux/redux.d.ts",
        "redux-thunk": "github:DefinitelyTyped/DefinitelyTyped/redux-thunk/redux-thunk.d.ts
    }
}
```

Now, delete the typings folder and run "typings install". This should recreate the folder with the expected d.ts files.

Dealing with Version Changes

Another problem with this approach is the versioning of dependencies. If we run "jspm update" and any dependency version changes then the abo
correct anymore. The folder specified in there has the old version and thus the typings installation will fail.

We can solve this with an automated script that will generate the typings.json file from a configuration. Here's an example of how this could be don

Add the following to your package.json file:

package.json

```
"typingsDependencies": {
  "registry": [
    "redux",
    "redux-thunk"
  ],
  "file": [
    "jspm_packages/npm/angular2-redux@VERSION/dist/index.d.ts",
    "jspm_packages/github/InfomediaLtd/angular2-simple-list@VERSION/app/index.d.ts"
  ]
},
```

This is a custom section that we create for ourselves to know which dependencies need typings and how to retrieve them. The "VERSION" in there with the version we have at the time of installation.

Now, create this script file:

typings.js

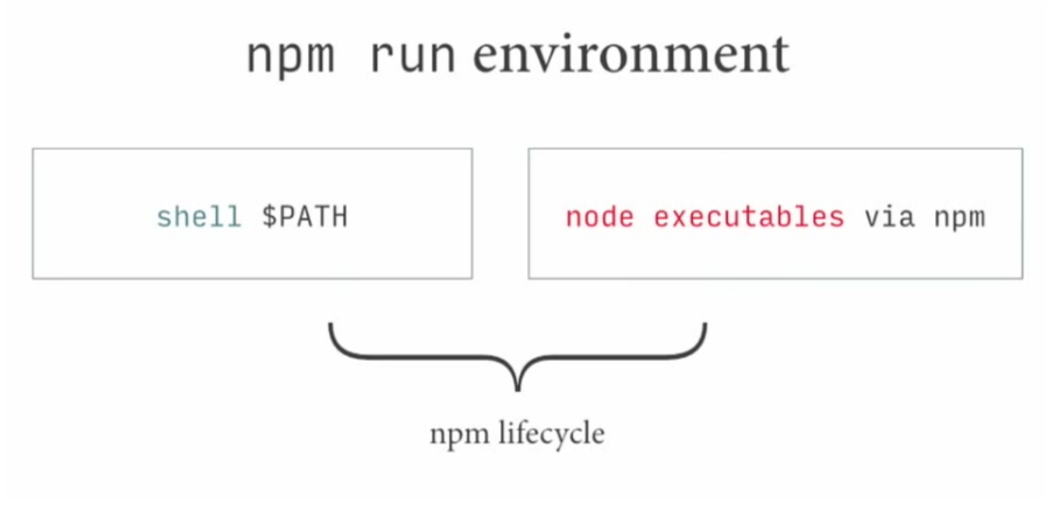
```
var fs = require("fs");
var typingsDependencies = JSON.parse(fs.readFileSync("package.json")).typingsDependencies;
var result = {"devDependencies": {},"ambientDevDependencies": {}};
// generate dt dependencies
typingsDependencies && (typingsDependencies.registry||[]).forEach(value => {
    result.ambientDevDependencies[value] = "github:DefinitelyTyped/DefinitelyTyped/" + valu
value + ".d.ts";
});
// generate local d.ts dependencies
typingsDependencies && (typingsDependencies.file||[]).forEach((path) => {
    var dependencyName = path.replace(/@VERSION.*/, "");
    var parentPath = dependencyName.replace(/[^\/]*$/, "");
    var namePrefix = dependencyName.replace(parentPath,"");
    var matchingFolders = fs.readdirSync(parentPath)
                            .filter(name => name.startsWith(namePrefix+"@"))
                            .filter(name => fs.lstatSync(parentPath + "/" + name).isDirecto
    if (matchingFolders.length > 0) {
        result.devDependencies[namePrefix] = "file:" + path.replace(/VERSION/,
matchingFolders[0].replace(/.*@/,""));
    } else {
        console.log("Couldn't find a single match for '" + path + "' in " + parentPath);
    }
});
fs.writeFile("typings.json", JSON.stringify(result, null, 4), err => console.log(err||"Crea
typings.json"));
```

You can test it out by running it:

```
node typings.js
```

This should override the typings.json with a new content based on the dependencies added to our package.json and the appropriate versions from
versions were changed then it should be the same as the file created manually above.

The "npm run" environment is a powerful tool for automating simple tasks and make them cross compatible:



You can automate the creation of the file with the typings installation by adding some scripts to our package.json:

package.json

```
"scripts": {
 "postinstall": "jspm install && npm run typings",
 "typings": "node typings.js && rm -rf typings && typings install"
},
```

This adds two scripts you can run:

- "npm run typings": This will run the script to create typings.json, delete the typings folder and then run the typings installation
- "npm install": after installing all npm dependencies it will run the "postinstall" script which also adds the jspm dependencies and executes t set up the additional typings

```
npm install
```

Having this process in place means we don't need to check in the typings.json or worry about versioning. Every time we want to ensure all depend their typings versions are installed we would run the above script.

Summary

This section was all about workarounds that hopefully will be resolved soon in the next TypeScript version. These workaround are required only fo the types and provide the important IntelliSense that aids during development.

It also introduces reproducible dependency installation in the form of npm scripting, which is very important for automating tasks and keeping every development workflow.