Summary

Angular users need to get the typings for the version of the library they are on. We have three options:

DefinitelyTyped
The current recommendation is fetching from DefinitelyTyped using tsd. However, we don't want to keep publishing to DT because

- It's too easy to get a version of the typings that conflicts with the version of the library being used (eg. there is an unversioned file in the directory which is "latest" but users don't know to update this when they update the library)
- We have an extra step in our release process to publish these, requiring a review in a foreign repository
- The tsd program adds a bit of complexity in our Getting Started path that is outside our control
- We like the simplicity of the story that DefinitelyTyped exists as a side-channel for JavaScript libraries which cannot (or choose not to) publish typings along with their distribution.

tsd link
The next logical step is to continue to use tsd, but tell it to "link" the .d.ts files from the node_modules folder into the "typings" directory that tsd uses. However, this requires users use both tsd install (for dependencies) as well as tsd link. Version conflicts are still possible here.
An example doing this: https://github.com/johnpapa/angular2-tour-of-heroes

TypeScript module resolver
TS 1.6 includes a node-compatible module resolution feature described in https://github.com/Microsoft/TypeScript/issues/2338 which we would like to use.
This is the most convenient for users, since the typescript compiler can now understand a statement like
`import {symbol} from 'my_module';`
to mean both the runtime concept of module loading, as well as module resolution to fetch the typings, via a single syntax and no setup required ahead of time aside from `npm install`.

The distinguishing characteristic of this approach is that beginners do not even need to *know* that there are typings that have to be presented to the Typescript language tools.

Status

This document describes how Angular 2 is moving to the TypeScript resolver.

A proof of concept is at https://github.com/alexeagle/angular2-distro/blob/master/README.md

This was merged into angular2 in
https://github.com/angular/angular/commit/95f984615bff680f800f07575a19ad8ca28925a9

For now, the old typings bundle (produced by custom system in dgeni, our documentation tool) is still a build artifact, and we can continue distributing it to DefinitelyTyped until we're confident that all users should move to the new typings model. We might then decide to stop using DefinitelyTyped, in which case we would publish a file at https://github.com/borisyankov/DefinitelyTyped/blob/master/angular2/angular2.d.ts explaining what to do instead.

How it works

We have moved all the triple-slash reference statements to a single pair of top-level typings files, one for ES5 and one for ES6. This one file must always be provided as a root input to the compiler, both for internal compilation and for users, so that transitive typings like es6-shim are available.

In our broccoli build step, we set "declaration" to true, asking the compiler to produce a .d.ts output for each .ts input. We also set "stripInternal" to true, so that the @internal JSDoc annotations cause symbols to be omitted from the .d.ts outputs. These are in browser_tree.ts

In the same file, we use a trick thanks to Wes on the TypeScript team, where we monkey-patch the top-level angular2.d.ts to include that single top-level typings using an "import" statement. This means that any user who brings that file into the compiler will get the transitive typings as well. This patch relies on a bug in the typescript compiler, which would typically disallow triple-slash references in exported external package typings files, but only checks the entry point file. It has the downside that there is no package scoping yet, so if the user also brings their own copy of these typings to the compiler (possibly a different version), they will get an error because angular2 has already declared those symbols. TypeScript team @microsoft is working on this problem (https://github.com/Microsoft/TypeScript/issues/4665).

When the user does `npm install angular2`, they get this structure:

```
node_modules/
  angular2/
    angular2.d.ts
    core.d.ts
    manual_typings/
      globals.d.ts
    typings/
      es6-shim/
```

```
es6-shim.d.ts
```

this means that if their code says
`import {Component} from ‘angular2/angular2’;`
it says “look in the npm module ‘angular2’, and find a .d.ts or .ts file called ‘angular2’. So this resolves correctly, both for runtime import and for typings.

What about DefinitelyTyped/tsd?
We will still need to train TypeScripters on these topics, because if they choose to use a third-party javascript library, they will almost certainly need to fetch the typings for it to avoid typechecker warnings.

More complete API shape
Instead of a handful of selected .d.ts bundles, we now export the complete modules/angular2 API. This includes APIs which are not re-exported through ‘angular2/angular2’, including router and http.
So users can write

```
import {Component, bootstrap, View} from 'angular2/angular2';
import {RouteConfig, ROUTER_DIRECTIVES, ROUTER_BINDINGS} from
'angular2/router';
import {ConnectionBackend} from 'angular2/http';
import {AsyncTestCompleter} from 'angular2/test_lib';
import {AnimationBuilder} from 'angular2/animate';
import {DoCheck} from 'angular2/lifecycle_hooks';
import {MockViewResolver} from 'angular2/mock';
import {AbstractControl} from 'angular2/web_worker/worker';
import {WebWorkerApplication} from 'angular2/web_worker/ui';
```

They can even import from files under angular2/src. That means that atypical users, such as someone writing a custom renderer, can import and get API typechecking, completion, etc.

Testing
A new gulp task is added, `test.typings.npm`
This relies on `build.js.dev`, and lays out a directory under /tmp which looks like a user’s project, including the angular2 distro under node_modules. Then it just runs the typescript compiler against it. (but this does *not* verify that the application works!)
If the test fails, you can just open the generated tmp folder in an editor to see what is broken.

Note that this test constrains our typings more than the old test.typings did. This is because we produce .d.ts files even for APIs which are not re-exported to users (see above). So these need to be --noImplicitAny and a non-@internal symbol cannot depend on an @internal symbol from another file.

I have a TODO to verify that methods starting with underscore (dart semantics for file visibility) match with @internal (new TypeScript semantics for module visibility).

New getting started instructions

It should simply say this:

```
$ npm install typescript angular2
$ ./node_modules/.bin/tsc -init --experimentalDecorators --target es5
# Open VSCode
# Start coding in .ts - completion, error-highlighting, nav-to-docs all work!
$ ./node_modules/.bin/tsc --watch
```

Worklog

- ~~Currently users import from "angular2/angular2" but the node module is named "angular2". But Typescript 1.6 did not ship with the "path mappings" feature described in the proposal. So these names do not match.~~ **Update: this is not causing any problem in practice, because the resolver understands the first "angular2" is node_modules/angular2 and the second "angular2" is the angular2.d.ts in that directory.**
- DONE: Need to remove relative references. TypeScript gives this error if the angular2.d.ts includes any ///<reference>'s
  `node_modules/angular2/bundles/typings/angular2/angular2.d.ts(16,1): error TS2654: Exported external package typings file cannot contain tripleslash references.`
- ~~TypeScript requires that an exported external package typings file be a module, meaning that it provides top-level exports. Simply defining ambient module declarations does not work. This might be a bug?~~
  ~~`app.ts(1,42): error TS2306: File 'node_modules/angular2/bundles/typings/angular2/angular2.d.ts' is not a module.`~~ **Update: use the .d.ts files written by the compiler, works**
- If user asks for --noImplicitAny, then Angular has to do this too. Action: clean up our missing typings. DONE and change:
- DONE(change, change): Members starting with underscore are emitted, but we want to hide them. Possible solution: use the /**@internal*/ JSDoc annotation documented on the `--stripInternal` experimental compiler option.
  steps:
    - Change @private to @internal (#4477)

    - Add private modifier on underscore methods (ts2dart#278)
    - add @internal on any symbol we wish to hide that cannot be private for some reason
- DONE: We expose the StringMap type in our .d.ts bundle template, and in a globals.d.ts that is not re-exported to the angular2.ts entry point. Solution: remove it and use TS index signatures instead. (it's better for users not to have that type exposed anyway) (#4483)
- ~~Need to strip private (may expose a symbol that is otherwise not exported)~~ This is working as intended: users need to know about private members when subtyping, and none of the referenced symbols are exposed. See https://github.com/Microsoft/TypeScript/issues/5106
- DONE: Need to introduce abstract classes as a supertype of current Angular APIs with @private constructor. Otherwise the concrete class implies that a no-arg constructor is available.
- (note) somebody can compress these introduced abstract superclasses a bit by just returning an anonymous function (avoids increasing angular codesize)