**Summary**

We want to stop using Traceur to compile the Angular2 codebase and instead compile to both ES5 and Dart using the TypeScript compiler.

The same toolchain will eventually be the canonical one for writing Angular2 apps as well.

This means the Angular2 codebase needs to be valid TypeScript. TypeScript is adopting some additional syntax and language features to allow continued use of metadata annotations.

**Details**

The Angular2 codebase has several types of sources:

* .js files are written in ‘AtScript’ and need to be valid TypeScript. They are transpiled into both Dart and ES5 for distribution (and possibly ES6 as well).  
  **Proposal**: rename files to .ts as they become valid TypeScript, to indicate what language syntax they use, and also for tooling (tools like tslint and Webstorm expect that extension.)
* .dart files are for interop with the core Dart APIs. They are merely packaged with the Dart distribution, and are not read by any transpiler.
* .es6 files are for interop with the core browser APIs, but still need to be transpiled to ES5 for current browsers. These files need to be valid TypeScript as well, but the transpiled version is packaged only with the JS distribution, not the Dart distribution. The suffix is misleading; these files are not valid ES6.  
  **Proposal**: these files should end with .ts as well. Other ways to provide the hint for filtering them out of the Dart package:
  + end them with .js.ts or such (convenient for glob-style expression in gulpfile)
  + include a file-level comment or metadata annotation which is easy to parse out

**Rough plan**

1. Write a [TypeScript to Dart transpiler](https://github.com/angular/ts2dart) [angular-team]
2. Pick a package to start with (possibly di) which is close to valid TS already.
3. Create a gulp task which accepts all .js files and in this package and runs them through ts2dart. *[WIP* [*https://github.com/angular/angular/compare/ts2dart*](https://github.com/angular/angular/compare/ts2dart)*]* This will break the build if any of these files don’t parse. The output dart code can be thrown away or written to an experimental output directory, and we still use the traceur transpiled code in the dart distribution.
4. Fix the build by modifying ts2dart and the Angular codebase as needed, so that it is a language subset which parses in TS but also correctly executes in ES6 and Dart via traceur.
5. Expand one package at a time.
6. **First milestone**: all .js files are **valid TS syntax** as determined by ts2dart. Still can’t emit or type-check with tsc. Still using traceur as primary.
7. Run the output of ts2dart through the dart analyzer to find more errors, including type errors. Fix them.
8. Transpile the .es6 files to ES5 using tsc
9. Pick a starting package again, this time with the constraint that it must be a root in the TS-dependency tree (depends on no other package).
10. Create a gulp task which runs tsc with emitOnError:true and ignore the return value. Run this on the chosen package. Emits both ES6 and Dart. Exclude this package from the traceur compile.
11. Fix the build by modifying both ts2dart and the Angular codebase as needed, so that the compile succeeds, and the emitted ES6 and Dart code can pass the existing tests. Also, the Dart runtime will still do the type-checking even if the compile step doesn’t.
12. Expand one package at a time, following the dependency tree to the root.
13. **Second milestone**: whole codebase is **incorrectly typed TypeScript.** We no longer use Traceur.
14. Big bang change to all the file extensions, as proposed above.
15. Pick a starting package again, at random.
16. Change the build to enable the tsc type checking in this package.
17. Fix the build by correcting the types.
18. Expand one package at a time.
19. **Third milestone**: whole codebase is **non-idiomatic correct TypeScript**.
20. Ad-hoc: update the codebase to introduce Typescript idioms
    1. Replace pure-abstract classes with interfaces, eg. modules/angular2/src/change\_detection/interfaces.js
    2. Remove comments like ‘// TODO: add support for object type literals to traceur!’
    3. more ?
21. **Fourth milestone: all done.**

**Possible issues:**

Does the RTTS library work with TypeScript? ([comment](https://github.com/angular/angular/blob/e05079f4a8102cfa1f06d14203f9dfbbaab8f66e/modules/rtts_assert/test/rtts_assert_spec.es6#L2) says it is designed for Traceur)

**Alternatives:**

* 1. compile with both side-by-side during transition
     + update traceur to understand the 5 syntax differences Mohamed listed [[1]](#footnote-0)
     + strip down the syntax to a subset that both compilers understand
  2. Implement the dart output in tsc and switch over in a big bang change
  3. Run the compilers one-after-the-other (but I think it wouldn't work because the types are lost before the second one runs)

1. Things that traceur needs to support:

   - Optional properties and optional functional arguments (e.g. function foo (a?: number, b?:string))

   - Interface declarations (e.g. interface IFoo { a: string, method(): void; }

   - Type alias syntax (type int = number;)

   - Generic signatures (e.g. then<T>(promise: Promise<T>, success: Function, rejection: Function): Promise<T>)

   - Type cast syntax ( e.g. var x = <number> y;) [↑](#footnote-ref-0)