typescript

Source code of released version | Source code of development version

---

This plugin package enables the use of TypeScript modules with `.ts` or `.tsx` file extensions in Meteor applications and packages, alongside `.js` modules (or whatever other types of modules you might be using).

Usage

The `typescript` package registers a compiler plugin that transpiles TypeScript to plain ECMAScript, which is then compiled by Babel for multiple targets (server, modern browsers, legacy browsers, cordova). By default, the `typescript` package is included in the `.meteor/packages` file of all new Meteor applications.

To add the `typescript` package to an existing app, run the following command from your app directory:

```
meteor add typescript
```

To add the `typescript` package to an existing package, include the statement `api.use('typescript');` in the `Package.onUse` callback in your `package.js` file:

```
Package.onUse(function (api) {
  api.use('typescript');
});
```

Supported TypeScript features

Almost all TypeScript syntax is supported, though this plugin currently does not attempt to provide type checking (just compilation).

Since the Meteor `typescript` package runs the official TypeScript compiler before running Babel, it does not suffer from the same caveats known to affect the Babel TypeScript transform. In particular, `namespace` s are fully supported.

However, as of this writing, the Meteor `typescript` package compiles TypeScript modules individually, using the `transpileModule` function, which means that certain cross-file analysis and compilation will not work. For example, `export const enum {...}` is not fully supported, though `const enum {...}` works when confined to a single module.

Unlike the Babel implementation of TypeScript, there is nothing fundamentally preventing Meteor from compiling all TypeScript modules together, rather than individually, which would enable full support for features like `export const enum`. That's an area for future improvement, though we will have to weigh the performance implications carefully.

As of this writing, `tsconfig.json` files are ignored by the plugin. There's nothing fundamentally preventing the Meteor `typescript` plugin from accepting configuration options from `tsconfig.json`, but for now we've kept things simple by avoiding configuration complexities. You may still want to have a `tsconfig.json` file in your application root directory to configure the behavior of editors like VSCode, but it will not be respected by Meteor.

Finally, since the TypeScript compiler runs first, syntax that is not understood by the TypeScript compiler, such as experimental ECMAScript proposals, may cause the TypeScript parser to reject your code. You can use `.babelrc` files to configure Babel compilation, but TypeScript still has to accept the code first.

Areas for improvement

- Compile all TypeScript modules at the same time, rather than compiling them individually, to enable cross-module analysis and compilation. In the meantime, if you need this behavior, consider using `adornis:typescript`.
- Use the version of `typescript` installed in the application `node_modules` directory, rather than the one bundled with `meteor-babel`. We will attempt to keep `meteor-babel`'s version of `typescript` up-to-date, but it would be better to leave this decision to the application developer.
- Generate `.d.ts` declaration files that can be consumed by external tools. In particular, a Meteor package that uses TypeScript could generate `.d.ts` files in the `/node_modules/meteor/package-name/` directory, which would allow tools like VSCode to find the right types for the package when importing from `meteor/package-name`.
- Cache the output of the TypeScript compiler separately from the output of Meteor's Babel compiler pipeline. The TypeScript compiler does not compile code differently for different targets (server, modern, legacy, etc.), so its results could theoretically be reused for all targets.
- Make the `typescript` plugin look up `tsconfig.json` files (similar to how `babel-compiler` looks up `.babelrc` files) and obey (some of) the configuration options.