Shims

The shims package deals with the specification and generation of "shim files". These are files which are not part of the user's original program, but are added by the compiler by user request or in support of certain features. For example, users can request that the compiler produce `.ngfactory` files alongside user files to support migration from View Engine (which used `.ngfactory` files) to Ivy which does not.

API

Shim generation is exposed through two interfaces: `TopLevelShimGenerator` and `PerFileShimGenerator`. Each implementation of one of these interfaces produces one or more shims of a particular type.

A top-level shim is a shim which is a "singleton" with respect to the program - it's one file that's generated and added in addition to all the user files.

A per-file shim is a shim generated from the contents of a particular file (like how `.ngfactory` shims are generated for each user input file, if requested).

Shims from either kind of generator can be emittable, in which case their `ts.SourceFile`s will be transpiled to JS and emitted alongside the user's code, or non-emittable, which means the user is unlikely to be aware of their existence.

This API is used both by the shim generators in this package as well as for other types of shims generated by other compiler subsystems.

Implementation

The shim package exposes two specific pieces of functionality related to the integration of shims into the creation of a `ts.Program`:

- A `ShimReferenceTagger` which "tags" `ts.SourceFile`s prior to program creation, and creates links from each original file to all of the per-file shims which need to be created for those file.
- A `ShimAdapter` which is used by an implementation of `ts.CompilerHost` to include shims in any program created via the host.

`ShimAdapter`

The shim adapter is responsible for recognizing when a path being loaded corresponds to a shim, and producing a `ts.SourceFile` for the shim if so.

Recognizing a shim filename involves two steps. First, the path itself must match a pattern for a particular `PerFileShimGenerator`'s shims (for example, NgFactory shims end in `.ngfactory.ts`). From this filename, the "source" filename can be inferred (actually several source filenames, since the source file

might be `.ts` or `.tsx`). Even if a path matches the pattern, it's only a valid shim if the source file actually exists.

Once a filename has been recognized, the `ShimAdapter` caches the generated shim source file and can quickly produce it on request.

Shim loading in practice As TS starts from the root files and walks imports and references, it discovers new files which are part of the program. It will discover shim files in two different ways:

- As references on their source files (those added by `ShimReferenceTagger`).
- As imports written by users.

This means that it's not guaranteed for a source file to be loaded before its shim.

`ShimReferenceTagger`

During program creation, TypeScript enumerates the `.ts` files on disk (the original files) and includes them into the program. However, each original file may have many associated shim files, which are not referenced and do not exist on disk, but still need to be included as well.

The mechanism used to do this is "reference tagging", which is performed by the `ShimReferenceTagger`.

`ts.SourceFile`s have a `referencedFiles` property, which contains paths extracted from any `/// <reference>` comments within the file. If a `ts.SourceFile` with references is included in a program, so are its referenced files.

This mechanism is (ab)used by the `ShimReferenceTagger` to create references from each original file to its shims, causing them to be loaded as well.

Once the program has been created, the `referencedFiles` properties can be restored to their original values via the `cleanup()` operation. This is necessary as `ts.SourceFile`s may live on in various caches for much longer than the duration of a single compilation.

Expando

The shim system needs to keep track of various pieces of metadata for `ts.SourceFile`s:

- Whether or not they're shims, and if so which generator created them.
- If the file is not a shim, then the original `referenceFiles` for that file (so it can be restored later).

Instead of `Map`s keyed with `ts.SourceFile`s which could lead to memory leaks, this information is instead patched directly onto the `ts.SourceFile` instances using an expando symbol property `NgExtension`.

Usage

Factory shim generation

Generated factory files create a catch-22 in ngtsc. Their contents depends on static analysis of the current program, yet they're also importable from the current program. This importability gives rise to the requirement that the contents of the generated file must be known before program creation, so that imports of it are valid. However, until the program is created, the analysis to determine the contents of the generated file cannot take place.

ngc used to get away with this because the analysis phase did not depend on program creation but on the metadata collection / global analysis process.

ngtsc is forced to take a different approach. A lightweight analysis pipeline which does not rely on the `ts.TypeChecker` (and thus can run before the program is created) is used to estimate the contents of a generated file, in a way that allows the program to be created. A transformer then operates on this estimated file during emit and replaces the estimated contents with accurate information.

It is important that this estimate be an overestimate, as type-checking will always be run against the estimated file, and must succeed in every case where it would have succeeded with accurate info.

Summary shim generation

Summary shim generation is simpler than factory generation, and can be generated from a `ts.SourceFile` without needing to be cleaned up later.

Other uses of shims

A few other systems in the compiler make use of shim generation as well.

- `entry_point` generates a flat module index (in the way View Engine used to) using a shim.
- `typecheck` includes template type-checking code in the program using a shim generator.