Writing a TypeScript Language Service Plugin

In TypeScript 2.2 and later, developers can enable *language service plugins* to augment the TypeScript code editing experience. The purpose of this guide is to help you write your own plugin.

What's a Language Service Plugin?

TypeScript Language Service Plugins ("plugins") are for changing the *editing experience* only. The core TypeScript language remains the same. Plugins can't add new language features such as new syntax or different typechecking behavior, and plugins aren't loaded during normal commandline typechecking or emitting, (so are not loaded by `tsc`).

Instead, plugins are for augmenting the editing experience. Some examples of things plugins might do:

- Provide errors from a linter inline in the editor
- Filter the completion list to remove certain properties from `window`
- Redirect "Go to definition" to go to a different location for certain identifiers
- Enable new errors or completions in string literals for a custom templating language

Examples of things language plugins cannot do:

- Add new custom syntax to TypeScript
- Change how the compiler emits JavaScript
- Customize the type system to change what is or isn't an error when running `tsc`

Developers using the plugin will `npm install --save-dev your_plugin_name` and edit their `tsconfig.json` file to enable your plugin.

Kickstart your plugin

There is a template repo which shows a working development environment for a TSServer Plugin here: https://github.com/orta/TypeScript-TSServer-Plugin-Template

Overview: Writing a Simple Plugin

Let's write a simple plugin. Our plugin will remove a user-configurable list of property names from the completion list. You might use this sort of plugin on your team to help remind you which APIs are 'banned' (for example, using the `caller` property of `function` is discouraged).

Setup and Initialization

When your plugin is loaded, it's first initialized as a factory function with its first parameter set to `{typescript: ts}`. It's important to use *this* value, rather than the imported `ts` module, because any version of TypeScript might be loaded by tsserver. If you use any other object, you'll run into compatibility problems later because enum values may change between versions.

Here's the minimal code that handles this injected `ts` value:

```
function init(modules: { typescript: typeof import("typescript/lib/tsserverlibrary")
}) {
  const ts = modules.typescript;
  /* More to come here */
```

```
}

export = init;
```

Decorator Creation

TypeScript Language Service Plugins use the Decorator Pattern to "wrap" the main TypeScript Language Service. When your plugin is initialized, it will be given a Language Service instance to wrap, and should return a new decorator wrapping this instance. This is exposed through the `create` function returned from your outer factory function.

Let's fill in some more code to properly set up a decorator:

```
function init(modules: { typescript: typeof import("typescript/lib/tsserverlibrary")
}) {
  const ts = modules.typescript;

  function create(info: ts.server.PluginCreateInfo) {
    // Set up decorator object
    const proxy: ts.LanguageService = Object.create(null);

    for (let k of Object.keys(info.languageService) as Array<keyof
ts.LanguageService>) {
      const x = info.languageService[k]!;
      // @ts-expect-error - JS runtime trickery which is tricky to type tersely
      proxy[k] = (...args: Array<{}>) => x.apply(info.languageService, args);
    }

    return proxy;
  }

  return { create };
}

export = init;
```

This sets up a "pass-through" decorator that invokes the underlying language service for all methods.

Enabling a plugin

To enable this plugin, users will add an entry to the `plugins` list in their `tsconfig.json` file:

```
{
    "compilerOptions": {
        "noImplicitAny": true,
        "plugins": [{ "name": "sample-ts-plugin" }]
    }
}
```

This name can only be an NPM package name.

Customizing Behavior

Let's modify the above pass-through plugin to add some new behavior.

We'll change the `getCompletionsAtPosition` function to remove certain entries named `caller` from the completion list:

```
// Remove specified entries from completion list
proxy.getCompletionsAtPosition = (fileName, position, options) => {
  const prior = info.languageService.getCompletionsAtPosition(fileName, position,
options);
  prior.entries = prior.entries.filter(e => e.name !== "caller");
  return prior;
};
```

Handling User Configuration

Users can customize your plugin behavior by providing additional data in their `tsconfig.json` file. Your plugin is given its enabling entry from the `tsconfig.json` file in the `info.config` property.

Let's allow the user to customize the list of names to remove from the completion list:

```
function create(info: ts.server.PluginCreateInfo) {
  // Get a list of things to remove from the completion list from the config object.
  // If nothing was specified, we'll just remove 'caller'
  const whatToRemove: string[] = info.config.remove || ["caller"];

  const proxy: ts.LanguageService = Object.create(null);
  // ... (set up decorator here) ...

  // Remove specified entries from completion list
  proxy.getCompletionsAtPosition = (fileName, position, options) => {
    const prior = info.languageService.getCompletionsAtPosition(
      fileName,
      position,
      options
    );
    prior.entries = prior.entries.filter(e => whatToRemove.indexOf(e.name) < 0);
    return prior;
  };

  return proxy;
}
```

The new `tsconfig.json` file might look like this:

```
{
    "compilerOptions": {
        "noImplicitAny": true,
        "plugins": [{
```

```
            "name": "sample-ts-plugin",
            "remove": ["caller", "callee", "getDay"]
        }]
    }
}
```

Debugging

You'll probably want to add some logging to your plugin to help you during development. The TypeScript Server Log allows plugins to write to a common log file.

Setting up TypeScript Server Logging

Your plugin code runs inside the TypeScript Server process. Its logging behavior can be enabled by setting the `TSS_LOG` environment variable. To log to a file, set `TSS_LOG` to:

```
-logToFile true -file C:\SomeFolder\MyTypeScriptLog.txt -level verbose
```

Ensure that the containing directory (`C:\SomeFolder` in this example) exists and is writable.

Logging from your plugin

You can write to this log by calling into the TypeScript project's logging service:

```
function create(info: ts.server.PluginCreateInfo) {
  info.project.projectService.logger.info(
    "I'm getting set up now! Check the log for this message."
  );
}
```

Putting it all together

```
function init(modules: { typescript: typeof import("typescript/lib/tsserverlibrary")
}) {
  const ts = modules.typescript;

  function create(info: ts.server.PluginCreateInfo) {
    // Get a list of things to remove from the completion list from the config
object.
    // If nothing was specified, we'll just remove 'caller'
    const whatToRemove: string[] = info.config.remove || ["caller"];

    // Diagnostic logging
    info.project.projectService.logger.info(
      "I'm getting set up now! Check the log for this message."
    );

    // Set up decorator object
    const proxy: ts.LanguageService = Object.create(null);
    for (let k of Object.keys(info.languageService) as Array<keyof
```

```
ts.LanguageService>) {
      const x = info.languageService[k]!;
      // @ts-expect-error - JS runtime trickery which is tricky to type tersely
      proxy[k] = (...args: Array<{}>) => x.apply(info.languageService, args);
    }

    // Remove specified entries from completion list
    proxy.getCompletionsAtPosition = (fileName, position, options) => {
      const prior = info.languageService.getCompletionsAtPosition(fileName,
position, options);
      if (!prior) return

      const oldLength = prior.entries.length;
      prior.entries = prior.entries.filter(e => whatToRemove.indexOf(e.name) < 0);

      // Sample logging for diagnostic purposes
      if (oldLength !== prior.entries.length) {
        const entriesRemoved = oldLength - prior.entries.length;
        info.project.projectService.logger.info(
          `Removed ${entriesRemoved} entries from the completion list`
        );
      }

      return prior;
    };

    return proxy;
  }

  return { create };
}

export = init;
```

Testing Locally

To locally test your plugin, set up a sample project and use a `file:` dependency (e.g. `"tsserver-plugin": "file:.."`) to link your plugin via the module name in the tsconfig.json file. For example:

```
your_plugin/index.ts
your_plugin/index.js (compiled by tsc)
your_plugin/sample_project/package.json
your_plugin/sample_project/tsconfig.json
```

where `your_plugin/sample_project/package.json` contains

```
{
    "name": "sample_project",
    "dependencies": {
        "your_plugin": "file:..",
        "typescript": "^4.4.3"
```

```
    }
}
```

and `your_plugin/sample_project/tsconfig.json` contains

```
{
    "compilerOptions": {
        "plugins": [{
            "name": "your_plugin",
        }]
    }
}
```

Alternatively, you can test your plugin similarly to how you would test other node modules. To set up a sample project where you can easily test plugin changes:

- Run `npm link` from your plugin directory
- In your sample project, run `npm link your_plugin_name`
- Add an entry to the `plugins` field of the `tsconfig.json`
- Rebuild your plugin and restart your editor to pick up code changes

Note: If you're using Visual Studio Code, you'll have to use the first approach above, with a path to the module, or run the "TypeScript: Select TypeScript Version" command and choose "Use Workspace Version", or click the version number between "TypeScript" and 😃 in the lower-right corner. Otherwise, VS Code will not be able to find your plugin.

Real-world Plugins

Some other TypeScript Language Service Plugin implementations you can look at for reference:

- https://github.com/angular/angular/blob/master/packages/language-service/src/ts_plugin.ts
- https://github.com/Quramy/ts-graphql-plugin
- https://github.com/Microsoft/typescript-styled-plugin
- https://github.com/angelozerr/tslint-language-service/
- https://github.com/xialvjun/ts-sql-plugin
- https://github.com/HearTao/ts-string-literal-enum-plugin
- https://github.com/ngnijland/typescript-todo-or-die-plugin