TypeScript

Wednesday, May 15, 2024 2:46 PM

Setup

- Install Node.js
- In CLI:
  npm install -g typescript
  npm install -g ts-node
  - ts-node is used to skip the transpilation step. You can go straight to *ts-node <filepath to .ts file>* to run it
- Develop in VS Code
  - Recommended to add Live Server extension
  - With this extension, you can run the webpage and whenever you transpile the code, it'll automatically update in the browser instead of having to refresh the browser

TypeScript is a superset of JavaScript

- A .ts file is transpiled into a .js file with:
  tsc <filepath to .ts file>
- This then creates a .js file that can be run with Node.js
- The goal of TS is to have code errors pop up at or before compile time rather than run time
- VS Code's hover tooltips should be more descriptive when the type is known and offer a helpful list of available methods and properties when applicable

Better Objects

- When trying to reference an object property it is easy to misspell it
  - An error like this in JS shows up as undefined during runtime and can someimes go unnoticed

interface <interface_name>{
    <property_name>: <property_type>,
    etc.
};

- This syntax is one of the improvements TS introduces: interfaces
- When misspelling an interface's property (thereby referencing a non-existent property), VS Code will highlight the error

Better functions

- Because JS parameters are non-typed, it is easy to feed in arguments in the wrong order
- You can specify what data type a function accepts and returns:
  function <functionName>(<varName>: <varType>, etc.) : <return_type>
  - The return_type can be an Object or Interface

Data Types

Primitive

- string
- number
- boolean
- null
- undefined
- symbol

Objects

- functions
- arrays
- classes
- etc.

Declaration Syntax

- *js* = Basic JavaScript tool

Basic

- *<js_declarator> <varNm>: <varType>[= value];*

Array

- *<js_declarator> <arrNm>: <arrType>[ ] = [<arrData>];*
- Note that the assignment value is optional

Object

- *<js_declarator> <objNm>: {*
      <propNm>: <propType>;
  };

Function

- *<js_declarator> <fnNm> = function (<paramNm>: <paramType>): <returnType> { <fnBody>; }*
- The returnType is optional, but I disagree with removing clarity
- *void* can be used for returnType to indicate a function has no return
- All parameters are required by default in TS and the number of arguments given must always match the number of parameters
  - You must still account for *null* or *undefined* as arguments to required parameters
- Optional parameters are defined by adding '?' to the end of their names:
  function (<paramNm>?: <paramType>)
  - Optional parameters must always be declared after any required parameters
- Default values:
  function(<paramNm> = <defaultVal>)
  - Function like optional parameters if placed after all required parameters:
    If the user does not provide an argument or passes in undefined, the default value is used
  - Functions like required parameters if placed before any required parameter:
    The user will be required to pass in some value or undefined
- Rest parameter:
  function (...<paramNm>: <paramType>[ ])
  - Allows the user to pass in 0 to as many arguments as they wish
  - A function can only ever have 1 of these and they must be placed at the end of the parameter list
  - Inside the function body, the arguments will be in an array that goes by *paramNm*