Lazy Loading Module

In this lesson , you will see how to lazy load a moduue.

TypeScript analyzes the code while transpiling into JavaScript and can detect if an import is being used only for its type, or not. Removing code is an important notion which helps to remove the size of the bundle of dependencies. It is also important if you want to lazy load a module. Lazy loading is the principle of loading on-demand when the module is required. Lazy loading gives a performance boost by initially only loading what is needed. Hereafter, the lazy loading pattern can load additional modules for a particular route or for particular actions. Support for lazy loading is still in an infantile stage with a syntax that requires being closer to a specific module target.

For example, a way to do this with TypeScript is to import using `require`. Inside the function that uses the lazily loaded module, the code calls `require` you to create an element of the module and mark its type with `typeof`, with the imported type at the top of the file. This line of code must set the value by calling require again. The code below was for TypeScript prior to version 2.4

```
import module1_variable1 = require("./module1");
export function lazyLoadWhenInvoked() {
  const _foo: typeof module1_variable1 = require("./module1");
  console.log(_foo.module1_variable1);
}
lazyLoadWhenInvoked();
```

Lazy loading can also use a dynamic import expression which is a recent feature of ECMAScript. However, this time, promise syntax is supported which allows having the module in the `then` or the exception detail in the `catch`. To do so, TypeScript's configurations must specify `esnext` as a module and target `es5` or higher. Some modules still do not completely support the syntax and require you to have TypeScript configured to use the `moduleResolution` to `node`.

```
async function getVariableLazyLoaded1(): Promise<string> {
  const mod1 = await import("./module1");
  const varFromOtherModule = await mod1.module1_variable1;
    return Promise.resolve(varFromOtherModule);
}
```

Or without async and only using promise.

```
function getVariableLazyLoaded2(): Promise<string> {
    return import("./module1").then(mod1 => {
        return mod1.module1_variable1;
    });
  }
```

It’s also possible to combine lazily loaded package with Webpack. The use of `import` can use the comment approach to specify in which bundle to pack the desired module that is specified in the import statement.

```
return import(/* webpackChunkName: "bundleAbc" */"./module1").then(…);
```