Modules and Namespaces

Brice Wilson

@brice_wilson   www.BriceWilson.net

Overview

History of modules in TypeScript

Modules versus namespaces

Creating and using namespaces

Creating and using modules

Changes in TypeScript 1.5

"Internal modules" became "namespaces"

"External modules" became "modules"

Support for ECMAScript2015 modules

Modules Versus Namespaces

Modules	Namespaces
Tool for organizing code	Tool for organizing code
Native support in Node.js	No special loader required
Browsers supported with module loader	Prevents global namespace pollution
Supports ES2015 module syntax	Best for smaller client applications
Facilitates code reuse	
Modules are the future!	

Defining Namespaces

```
namespace Membership {









}
```

Defining Namespaces

```
namespace Membership {
    export function AddMember(name: string) {
        // add a new member
    }
    export namespace Cards {
        export function IssueCard(memberNumber: number) {
            // issue new card
        }
    }
}
Membership.AddMember('Garrett');
Membership.Cards.IssueCard(1234);
```

```
/// <reference path="membership.ts" />
```

“Triple-Slash” References

Enhances editor support for referenced files

TypeScript compiler will compile all required references

```
/// <reference path="membership.ts" />

let memberName: string = 'Elaine';
let memberNumber: number = 789;
Membership.AddMember(memberName);
Membership.Cards.IssueCard(memberNumber);
```

“Triple-Slash” References

Enhances editor support for referenced files

TypeScript compiler will compile all required references

Use –outFile compiler option to generate a single JS output file

Demo

Using namespaces

Reasons to Use Modules

They're modular!!!

Maintainable

Reusable

Native to Node and ES2015

Organized simply in files and folders

Supported Module Formats

CommonJS

Asynchronous Module Definition (AMD)

Universal Module Definition (UMD)

System

ES2015

Module Loaders

Require.js

http://requirejs.org

SystemJS

https://github.com/systemjs/systemjs

Exporting from a Module

```
// periodicals.ts
export interface Periodical {
    issueNumber: number;
}
export class Magazine implements Periodical {
    issueNumber: number;
}
export function GetMagazineByIssueNumber(issue: number): Magazine {
    // retrieve and return a magazine
}
```

Exporting from a Module

```
// periodicals.ts
interface Periodical {
    issueNumber: number;
}
class Magazine implements Periodical {
    issueNumber: number;
}
function GetMagazineByTitle(title: string): Magazine {
    // retrieve and return a magazine
}
export { Periodical, Magazine, GetMagazineByTitle as GetMag}
```

Importing from a Module

```
// news.ts

import { Magazine, GetMag as GetMagazine } from './periodicals';
```

Importing from a Module

```
// news.ts

import { Magazine, GetMag as GetMagazine } from './periodicals';

let newsMag: Magazine = GetMagazine('Weekly News');


// kids.ts

import * as mag from './periodicals’;
```

Importing from a Module

```
// news.ts
import { Magazine, GetMag as GetMagazine } from './periodicals';
let newsMag: Magazine = GetMagazine('Weekly News');


// kids.ts
import * as mag from './periodicals';
let kidMag: mag.Magazine = mag.GetMag('Games and Stuff!');
```

Demo

Export and import basics

Demo

Importing an entire module

Default Exports

```
// movie.ts
export default class {
    title: string;
    director: string;
}
```

Default Exports

```
// movie.ts
export default class {
    title: string;
    director: string;
}
// kids.ts
import AnimatedMovie from './movie’;
```

Default Exports

```
// movie.ts
export default class {
    title: string;
    director: string;
}
// kids.ts
import AnimatedMovie from './movie';
let cartoon = new AnimatedMovie();
```

Demo

Using default exports

Summary

History of modules

Choosing between modules and namespaces

Creating and using namespaces

Exporting and importing code with modules