تعریف ماژول ها در TypeScript

ماژول ها جهت سازمان دهی کدهای نوشته شده در تایپ اسکریپت طراحی شده اند. ماژول ها را می توان به دو دسته ی کلی زیر تقسیم کرد.

- ماژول های داخلی
- ماژول های خارجی

ماژول های داخلی

این ماژول ها در ورژن های قبلی تایپ اسکریپت وجود داشتند. کار آن ها دسته بندی منطقی کلاس ها، رابط ها و توابع در یک واحد و export کردن آن ها در ماژول دیگری بود. در نسخه های جدیدتر تایپ اسکریپت اسم این گروه بندی منطقی به namespace تغییر کرد. در نتیجه ماژول های داخلی دیگر منسوخ شده و در عوض آن می توان از namespace استفاده کرد. با وجود اینکه از این ماژول ها هنوز پشتیبانی می شود، اما بهتر است که به جای آن ها از namespace استفاده کرد.

سینتکس ماژول داخلی (قدیمی)

```
module TutorialPoint {
  export function add(x, y) {
    console.log(x+y);
  }
}
```

سینتکس Namespace (جدید)

```
namespace TutorialPoint {
  export function add(x, y) { console.log(x + y);}
}
```

جاوا اسکریپت ایجاد شده برای هر دو حالت بالا یکی است.

```
var TutorialPoint;

(function (TutorialPoint) {

  function add(x, y) {

    console.log(x + y);

  }

  TutorialPoint.add = add;
```

```
})(TutorialPoint || (TutorialPoint = {}));
```

ماژول خارجی

وظیفه ی این ماژول ها مشخص کردن و بارگیری وابستگی ها بین فایل های js خارجی متعدد است. اگر تنها از یک فایل js استفاده شده باشد، در این صورت استفاده از ماژول های خارجی چندان مطلوب نیست. در گذشته مدیریت وابستگی بین فایل های جاوا اسکریپت با استفاده از تگ های script مرورگر انجام می شد (<script></script>). اما با توجه به اینکه این روش طی بارگیری ماژول ها تا حد زیادی خطی است، درنتیجه چندان قابل توسعه نیست. این یعنی به جای بارگیری فایل ها یکی پس از دیگری، هیچ گزینه ای برای بارگیری ناهمگام ماژول ها وجود ندارد. زمانی که برای سرور در حال برنامه نویسی جاوا اسکریپت هستید (مثلا NodeJs)، حتی خبری از تگ های script نیست.

دو سناریو برای بارگیری مستقل فایل های js از یک فایل اصلی جاوا اسکریپت وجود دارد.

- سمت کلاینت ـ RequireJs
- سمت سرور ـ NodeJs

انتخاب Module Loader

برای پشتیبانی از بارگیری فایل های خارجی جاوا اسکریپت به یک module loader نیاز است. module loader کتابخانه جاوا اسکریپت دیگری خواهد بود. پرکاربرد ترین کتابخانه ای که در مرورگرها استفاده می شود، RequireJS است. در این کتابخانه از AMD (تعریف ناهمگام ماژول) پیاده سازی شده است. AMD این قابلیت را دارد تا به جای اینکه فایل ها را یکی پس از دیگری بارگیری کند، آن ها به صورت کاملا مجزا بارگیری کند حتی اگر این فایل ها به یکدیگر وابسته باشند.

تعریف ماژول های خارجی

در صورتی که بخواهیم ماژولی خارجی را با استفاده از CommonJS یا AMD در تایپ اسکریپت تعریف کنیم، هر یک از فایل ها به عنوان یک ماژول لحاظ می شوند. در نتیجه استفاده از ماژول های داخلی درون ماژول های خارجی اختیاری است.

در صورتی که بخواهیم سیستم ماژول AMD تایپ اسکریپت را به سیستم CommonJS تغییر دهیم، در این صورت به هیچ کار اضافه ی دیگری نیاز نداریم. تنها کاری که باید انجام دهیم این است که پرچم کامپایلر تغییر دهیم. این در حالی است که در جاوا اسکریپت در صورتی که بخواهیم از CommonJS به AMD یا برعکس برویم، باید کارهای بیشتری انجام دهیم.

با استفاده از عبارات کلیدی ’export‘و’import‘ می توان ماژول های خارجی را اعلان کرد.

سینتکس

```
//FileName : SomeInterface.ts
export interface SomeInterface {
  //code declarations
}
```

برای اینکه بتوان ماژول اعلان شده را در فایل دیگری استفاده کرد، مانند زیر از عبارت کلیدی import استفاده می شود. تنها اسم فایل مشخص شده و از extension دیگری استفاده نشده است.

```
import someInterfaceRef = require("./SomeInterface");
```

مثال

برای درک بهتر این مطلب به مثال زیر توجه کنید.

```
// IShape.ts

export interface IShape {

  draw();

}


// Circle.ts

import shape = require("./IShape");

export class Circle implements shape.IShape {

  public draw() {

    console.log("Cirlce is drawn (external module)");

  }

}


// Triangle.ts

import shape = require("./IShape");
```

```
export class Triangle implements shape.IShape {

  public draw() {

    console.log("Triangle is drawn (external module)");

  }

}


// TestShape.ts

import shape = require("./IShape");

import circle = require("./Circle");

import triangle = require("./Triangle");


function drawAllShapes(shapeToDraw: shape.IShape) {

  shapeToDraw.draw();

}


drawAllShapes(new circle.Circle());

drawAllShapes(new triangle.Triangle());
```

برای کامپایل کردن فایل اصلی تایپ اسکریپت در سیستم های AMD از دستور زیر استفاده می شود.

```
tsc --module amd TestShape.ts
```

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر برای AMD ایجاد می شود.

File:IShape.js

```
//Generated by typescript 1.8.10

define(["require", "exports"], function (require, exports) {

});
```

File:Circle.js

```
//Generated by typescript 1.8.10
define(["require", "exports"], function (require, exports) {
  var Circle = (function () {
    function Circle() {
    }
    Circle.prototype.draw = function () {
      console.log("Cirlce is drawn (external module)");
    };
    return Circle;
  })();
  exports.Circle = Circle;
});
```

File:Triangle.js

```
//Generated by typescript 1.8.10
define(["require", "exports"], function (require, exports) {
  var Triangle = (function () {
    function Triangle() {
    }
    Triangle.prototype.draw = function () {
      console.log("Triangle is drawn (external module)");
    };
    return Triangle;
  })();
  exports.Triangle = Triangle;
});
```

File:TestShape.js

```
//Generated by typescript 1.8.10
define(["require", "exports", "./Circle", "./Triangle"],
  function (require, exports, circle, triangle) {


  function drawAllShapes(shapeToDraw) {
    shapeToDraw.draw();
  }
  drawAllShapes(new circle.Circle());
  drawAllShapes(new triangle.Triangle());
});
```

برای کامپایل کردن فایل اصلی تایپ اسکریپت در سیستم های Commonjs از دستور زیر استفاده می شود.

```
tsc --module commonjs TestShape.ts
```

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر برای Commonjs ایجاد می شود.

File:Circle.js

```
//Generated by typescript 1.8.10
var Circle = (function () {
  function Circle() {
  }
  Circle.prototype.draw = function () {
    console.log("Cirlce is drawn");
  };
  return Circle;
})();
```

```
exports.Circle = Circle;
```

File:Triangle.js

```
//Generated by typescript 1.8.10
var Triangle = (function () {
  function Triangle() {
  }
  Triangle.prototype.draw = function () {
    console.log("Triangle is drawn (external module)");
  };
  return Triangle;
})();
exports.Triangle = Triangle;
```

File:TestShape.js

```
//Generated by typescript 1.8.10
var circle = require("./Circle");
var triangle = require("./Triangle");
function drawAllShapes(shapeToDraw) {
  shapeToDraw.draw();
}
drawAllShapes(new circle.Circle());
drawAllShapes(new triangle.Triangle());
```

خروجی

```
Cirlce is drawn (external module)
Triangle is drawn (external module)
```