عنوان: **آموزش TypeScript #4**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۵/۲۸ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, TypeScript, Client Side Framework

در پست‌های قبل با کلیات و primitive types در زبان TypeScript آشنا شدیم:
کلیات TypeScript
انواع داده ای اولیه در TypeScript

در این پست به مفاهیم شی گرایی در این زبان می‌پردازیم.

ماژول ها:

تعریف یک ماژول: برای تعریف یک ماژول باید از کلمه کلیدی module استفاده کنید. یک ماژول معادل یک ظرف است برای نگهداری کلاس‌ها و اینترفیس‌ها و سایر ماژول ها. کلاس‌ها و اینترفیس‌ها در TypeScript می‌توانند به صورت internal یا public باشند(به صورت پیش فرض internal است؛ یعنی فقط در همان ماژول قابل استفاده و فراخوانی است). هر چیزی که در داخل یک ماژول تعریف می‌شود محدوده آن در داخل آن ماژول خواهد بود. اگر قصد توسعه یک پروژه در مقیاس بزرگ را دارید می‌توانید همانند دات نت که در آن امکان تعریف فضای نام‌های تودرتو امکان پذیر است در TypeScript نیز، ماژول‌های تودرتو تعریف کنید. برای مثال:

```
module MyModule1 {
    module  MyModule2 {
     }
}
```

اما به صورت معمول سعی می‌شود هر ماژول در یک فایل جداگانه تعریف شود. استفاده از چند ماژول در یک فایل به مرور، درک پروژه را سخت خواهد کرد و در هنگام توسعه امکان برخورد با مشکل وجود خواهد داشت. برای مثال اگر یک فایل به نام MyModule.ts داشته باشیم که یک ماژول به این نام را شامل شود بعد از کامپایل یک فایل به نام MyModule.js ایجاد خواهد شد.

کلاس ها:
برای تعریف یک کلاس می‌توانیم همانند دات نت از کلمه کلیدی class استفاده کنیم. بعد از تعریف کلاس می‌توانیم متغیرها و توابع مورد نظر را در این کلاس قرار داده و تعریف کنیم.

```
module Utilities {
   export class Logger {
      log(message: string): void{
       if(typeofwindow.console !== 'undefined') {
           window.console.log(message);
        }
      }
   }
}
```

نکته مهم و جالب قسمت بالا کلمه export است. export معادل public در دات نت است و کلاس logger را قابل دسترس در خارج ماژول Utilities خواهد کرد. اگر از export در هنگام تعریف کلاس استفاده نکنیم این کلاس فقط در سایر کلاس‌های تعریف شده در داخل همان ماژول قابل دسترس است.
تابع log که در کلاس بالا تعریف کردیم به صورت پیش فرض public یا عمومی است و نیاز به استفاده export نیست.
برای استفاده از کلاس بالا باید این کلمه کلیدی new استفاده کنیم.

```
window.onload = function() {
  varlogger = new Utilities.Logger();
  logger.log('Logger is loaded');
};
```

برای تعریف سازنده برای کلاس بالا باید از کلمه کلیدی constructor استفاده نماییم:

```
export class Logger{
constructor(private num: number) {
}
```

با کمی دقت متوجه تعریف متغیر num به صورت private خواهید شد که برخلاف انتظار ما در زبان‌های دات نتی است. بر خلاف دات نت در زبان TypeScript، دسترسی به متغیر تعریف شده در سازنده با کمک اشاره گر this در هر جای کلاس ممکن می‌باشد. در نتیجه نیازی به تعریف متغیر جدید و پاس دادن مقادیر این متغیرها به این فیلدها نمی‌باشد.
اگر به تابع log دقت کنید خواهید دید که یک پارامتر ورودی به نام message دارد که نوع آن string است. در ضمن Typescript از پارامترهای اختیاری(پارامتر با مقدار پیش فرض) نیز پشتیبانی می‌کند. مثال:

```
pad(num: number, len: number= 2, char: string= '0')
```

استفاده از پارامترهای Rest

منظور از پارامترهای Rest یعنی در هنگام فراخوانی توابع محدودیتی برای تعداد پارامترها نیست که معادل params در دات نت است. برای تعریف این گونه پارامترهاکافیست به جای params از ... استفاده نماییم.

```
function addManyNumbers(...numbers: number[]) {
  var sum = 0;
  for(var i = 0; i < numbers.length; i++) {
    sum += numbers[i];
 }
  returnsum;
}
var result = addManyNumbers(1,2,3,5,6,7,8,9);
```

تعریف توابع خصوصی

در TypeScript امکان توابع خصوصی با کلمه کلیدی private امکان پذیر است. همانند دات نت با استفاده از کلمه کلیدی private می‌توانیم کلاسی تعریف کنیم که فقط برای همان کلاس قابل دسترس باشد(به صورت پیش فرض توابع به صورت عمومی هستند).

```
module Utilities {
    Export class Logger {
     log(message: string): void{
                 if(typeofwindow.console !== 'undefined') {
                    window.console.log(this.getTimeStamp() + ' -'+ message);
                    window.console.log(this.getTimeStamp() + ' -'+ message);
                }
        }
  private getTimeStamp(): string{
      var now = newDate();
      return now.getHours() + ':'+
      now.getMinutes() + ':'+
      now.getSeconds() + ':'+
      now.getMilliseconds();
  }
 }
}
```

از آن جا که تابع getTimeStamp به صورت خصوصی تعریف شده است در نتیجه امکان استفاده از آن در خارج کلاس وجود ندارد. اگر سعی بر استفاده این تابع داشته باشیم توسط کامپایلر با یک warning مواجه خواهیم شد.

```
window.onload = function () {
    var logger = new Utilities.Logger();
    logger.getTimeStamp();
};
```

یک نکته مهم این است که کلمه private فقط برای توابع و متغیرها قابل استفاده است.

تعریف توابع static:

در TypeScript امکان تعریف توابع static وجود دارد. همانند دات نت باید از کلمه کلیدی static استفاده کنیم.

```
classFormatter {
static pad(num: number, len: number, char: string): string{
      var output = num.toString();
         while(output.length < len) {
         output = char + output;
      }
   returnoutput;
   }
  }
}
```

و استفاده از این تابع بدون وهله سازی از کلاس :

```
Formatter.pad(now.getSeconds(), 2, '0') +
```

Function Overload

همان گونه که در دات نت امکان overload کردن توابع میسر است در TypeScript هم این امکان وجود دارد.

```
static pad(num: number, len?: number, char?: string);
static pad(num: string, len?: number, char?: string);
static pad(num: any, len: number= 2, char: string= '0') {
 var output = num.toString();
 while(output.length < len) {
 output = char + output;
 }
 returnoutput;
}
```

ادامه دارد...