عنوان: **آموزش TypeScript #5**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۵/۳۰ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, TypeScript, Client Side Framework

در ادامه مباحث شی گرایی در TypeScript قصد داریم به مباحث مربوط به interface و طریقه استفاده از آن بپردازیم. همانند زبان‌های دات نتی در TypeScript نیز به راحتی می‌توانید interface تعریف کنید. در یک اینترفیس اجازه پیاده سازی هیچ تابعی را ندارید و فقط باید عنوان و پارامترهای ورودی و نوع خروجی آن را تعیین کنید. برای تعریف اینترفیس از کلمه کلیدی interface به صورت زیر استفاده خواهیم کرد.

```
export interface ILogger {
  log(message: string): void;
}
```

همان طور در پست‌های قبلی مشاهده شد از کلمه کلیدی export برای عمومی کردن اینترفیس استفاده می‌کنیم. یعنی این اینترفیس از بیرون ماژول خود نیز قابل دسترسی است.
حال نیاز به کلاسی داریم که این اینترفیس را پیاده سازی کند. این پیاده سازی به صورت زیر انجام می‌گیرد:

```
export class Logger implements ILogger
{
}
```

یا:

```
export class AnnoyingLogger implements ILogger {
   log(message: string): void{
        alert(message);
    }
}
```

همانند دات نت یک کلاس می‌تواند چندین اینترفیس را پیاده سازی کند.(اصطلاحا به این روش explicit implementation یا پیاده سازی صریح می‌گویند)

```
export class MyClass implements IFirstInterface, ISecondInterface
{

}
```

*یکی از قابلیت جالب و کارآمد زبان TypeScript این است که در هنگام کار با اینترفیس‌ها حتما نیازی به پیاده سازی صریح نیست. اگر یک object تمام متغیرها و توابع مورد نیاز یک اینترفیس را پیاده سازی کند به راحتی همانند روش explicit emplementation می‌توان از آن object استفاده کرد. به این قابلیت **Duck Typing** می‌گویند. مثال:

```
IPerson {
   firstName: string;
   lastName: string;
}
class Person implements IPerson {
  constructor(public firstName: string, public lastName: string) {
  }
}

varpersonA: IPerson = newPerson('Masoud', 'Pakdel'); //expilict
varpersonB: IPerson = { firstName: 'Ashkan', lastName: 'Shahram'}; // duck typing
```

همان طور که می‌بینید object دوم به نام personB تمام متغیرها ی مورد نیاز اینترفیس IPerson را پیاده سازی کرده است در

نتییجه کامپایلر همان رفتاری را که با object اول به نام personA دارد را با آن نیز خواهد داشت.

پیاده سازی چند اینترفیس به صورت همزمان

همانند دات نت که یک کلاس فقط می‌تواند از یک کلاس ارث ببرد ولی می‌تواند n تا اینترفیس را پیاده سازی کند در TypeScript نیز چنین قوانینی وجود دارد. یعنی یک اینترفیس می‌تواند چندین اینترفیس دیگر را توسعه دهد(extend) و کلاسی که این اینترفیس را پیاده سازی می‌کند باید تمام توابع اینترفیس‌ها را پیاده سازی کند. مثال:

```
interface IMover {
 move() : void;
}

interface IShaker {
 shake() : void;
}

interface IMoverShaker extends IMover, IShaker {
}
class MoverShaker implements IMoverShaker {
 move() {
 }
 shake() {
 }
}
```

*به کلمات کلیدی extends و implements و طریقه به کار گیری آن‌ها دقت کنید.

instanceof

از instanceof زمانی استفاده می‌کنیم که قصد داشته باشیم که یک instance را با یک نوع مشخص مقایسه کنیم. اگر instance مربوطه از نوع مشخص باشد یا از این نوع ارث برده باشد مقدار true برگشت داده می‌شود در غیر این صورت مقدار false خواهد بود.

یک مثال:

```
var isLogger = logger instanceof Utilities.Logger;
var isLogger = logger instanceof Utilities.AnnoyingLogger;
var isLogger = logger instanceof Utilities.Formatter;
```

Method overriding

در TypeScript می‌توان مانند زبان‌های شی گرای دیگر Method overriding را پیاده سازی کرد. یعنی می‌توان متدهای کلاس پایه را در کلاس مشتق شده تحریف کرد. با یک مثال به شرح این مورد خواهم پرداخت.

فرض کنید یک کلاس پایه به صورت زیر داریم:

```
class BaseEmployee
{
    constructor (public fname: string,public lname: string)
    {
    }
    sayInfo()
    {
       alert('this is base class method');
    }
}
```

کلاس دیگری به نام Employee می‌سازیم که کلاس بالا را توسعه می‌دهد(یا به اصطلاح از کلاس بالا ارث می‌برد).

```
class Employee extends BaseEmployee
{
   sayInfo()
     {
        alert('this is derived class method');
     }
}
```

```
window.onload = () =>
{
    var first: BaseEmployee= new Employee();
    first.sayInfo();
    var second: BaseEmployee = new BaseEmployee();
    second.sayInfo();
}
```

نکته مهم این است که دیگر خبری از کلمه کلیدی virtual برای مشخص کردن توابعی که قصد overriding آن‌ها را داریم نیست. تمام توابع که عمومی هستند را می‌توان override کرد.

*اگر در کلاس مشتق شده قصد داشته باشیم که به توابع و فیلدهای کلاس پایه اشاره کنیم باید از کلمه کلیدی super استفاده کنیم.(معادل base در #C).

مثال:

```
class Animal {
    constructor (public name: string) {
    }
}

class Dog extends Animal {
      constructor (public name: string, public age:number)
      {
        super(name);
      }

    sayHello() {


      alert(super.name);


 }
}
```

اگر به سازنده کلاس مشتق شده دقت کنید خواهید دید که پارامتر name را به سازنده کلاس پایه پاس دادیم: کد معادل در #C به صورت زیر است:

```
public class Dog : Animal
{
      public Dog (string name, int age):base(name)
      {
      }
}
```

در تابع sayHello نیز با استفاده از کلمه کلیدی super به فیلد name در کلاس پایه دسترسی خواهیم داشت.

*دقت کنید که مباحث مربوط به interface و private modifier و Type safety که پیش‌تر در مورد آن‌ها بحث شد، فقط در فایل‌های TypeScript و در هنگام کد نویسی و طراحی معنی دار هستند، زیرا بعد از کامپایل فایل‌های ts این مفاهیم در Javascript پشتیبانی نمی‌شوند در نتیجه هیچ مورد استفاده هم نخواهد داشت.

ادامه دارد...