عنوان: **خطا ها و مدیریت خطا (Exception Handling)**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۸ ۰:۴۰
آدرس: www.dotnettips.info
برچسب‌ها: F#, Programming

مدیریت خطا در F# شبیه به الگوی try catch finally در C# است. برای تعریف خطا از کلمه کلیدی exception استفاده می‌کنیم و یک نام رو به اون اختصاص می‌دهیم و می‌تونیم به صورت اختیاری یک نوع داده رو هم برای این خطا با استفاده از کلمه کلیدی of تعیین کنیم.

```
exception myError of int
```

با استفاده از دستور raise می‌تونیم یک exception رو پرتاب کنیم.(به دلیل اینکه در دات نت از دستور throw به معنی پرتاب کردن استفاده می‌کنیم این جا نیز از همین لغت استفاده کردم کما اینکه در F# دستور raise جایگزین throw شده است). البته در جاهایی که قصد ما از پرتاب exception فقط متوقف کردن عملیات و نمایش یک خطا است می‌تونیم از دستور failwith به همراه یک پیغام نیز استفاده کنیم.(یک نمونه از آن را در فصل‌های قبلی مشاهده کردید)
ساختار کلی try catch finally در F# به صورت زیر است.(تنها تفاوت در کلمه with به جای catch است)

```
try
// try code here
with
//catch statement here
```

یا به صورت

```
try
// try code here
finally
//finally statement here
```

*نکته مهم: در F# شما اجازه استفاده از finally رو به همراه with ندارید.به همین دلیل من این ساختارو به دو صورت بالا نوشتم.

یک مثال از try with:

```
exception WrongSecond of int//یک exception تعریف می‌کنیم

let primes =
[ 2; 3; 5; 7; 11; 13; 17; 19; 23; 29; 31; 37; 41; 43; 47; 53; 59 ]

// وجود دارد یا نه prime یک تابع برای تست اینکه آیا ثانیه الان در لیست
let testSecond() =
try
let currentSecond = System.DateTime.Now.Second in

// شرط برای اینکه مشخص شود که ثانیه در لیست است یا خیر
if List.exists (fun x -> x = currentSecond) primes then

// اگر بود یک خطا تولید می‌شود
failwith "A prime second"

else

// پرتاب میشود wrongSecond اگر نبود یک استثنا از نوع
raise (WrongSecond currentSecond)

with
// catch استثناها کردن

WrongSecond x ->
printf "The current was %i, which is not prime" x
```

در کد با در هر خط توضیحات لازم داده شده است. نکته قابل ذکر این است که در #C زمانی که قصد داشته باشیم یک استثنا جدید ایجاد کنیم باید کلاسی جدیدی که از کلاس System.Exception ارث برده باشد(یا هر کلاس دیگری که خود از این System.Exception ارث برده است) ایجاد کنیم و کدهای مورد نظر رو در اون قرار بدیم. ولی در اینجا (در قسمتی که رنگ آن متفاوت است) به راحتی توانستیم یک استثنا جدید بر اساس نیاز بسازیم.

یک مثال از try finally :

```
// تابعی برای نوشتن فایل
let writeToFile() =
//ابتدا فایل به صورت متنی ساخته می‌شود
let file = System.IO.File.CreateText("test.txt")
try
// متن مورد نظر در فایل نوشته می‌شود
file.WriteLine("Hello F# users")
finally
//فایل مورد نظر بسته می‌شود.این دستور حتی اگر در هنگام نوشتن فایل استثنا هم رخ بدهد اجرا خواهد شد
file.Dispose()
```

عملکرد finally در #F دقیقا مشابه با عملکرد finally در #C است. یعنی دستورات بلوک finally همواره (چه استثنا رخ بدهد و چه رخ ندهد) اجرا خواهد شد.

***توجه :** برنامه نویسانی که قبلا با OCaml کدنویسی کرده اند هنگام برنامه نویسی #F از raise کردن‌های زیاد و بی مورد استثناها خودداری کنند. به دلیل نوع معماری CLR پرتاب کردن استثنا و مدیریت آن کمی هزینه بر است (بیشتر از زبان Ocaml). البته این مسئله در زبان‌های تحت دات نت نیز مطرح است کما اینکه در #C نیز مدیریت استثناها رو در بالاترین لایه انجام می‌دهیم و از catch کردن بی مورد استثنائات در لایه‌های زیرین خودداری می‌کنیم.

یک مثال از الگوی Matching در try with

```
let getNumber msg =
    printf msg;
    try
        int32(System.Console.ReadLine())
    with
        | :? System.FormatException -> -1
        | :? System.OverflowException -> System.Int32.MinValue
        | :? System.ArgumentNullException -> 0
```