عنوان: **مثالی از الگوی Delegate Dictionary**
نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۸/۱۹ ۱۵:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, Design patterns, Refactoring

این الگو چیز جدیدی نیست و قبلا تو سری مطالب « مروری بر کاربردهای Action و Func » دربارش مطلب نوشته شده و...
البته با توجه به جدید بودن این الگو اسم واحدی براش مشخص نشده ولی تو این مطلب «الگوی Delegate Dictionary» معرفی شده که بنظرم از بقیه بهتره.
به طور خلاصه این الگو میگه اگه قراره براساس شرایط (ورودی) خاصی کار خاصی انجام بشه بجای استفاده از IF و Switch از FuncوDictionary یا Action استفاده کنیم.

برای مثال فرض کنید مدلی به شکل زیر داریم

```
public class Person
{
    public int Id { get; set; }
    public Gender Gender { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

قراره براساس جنسیت( **شرایط** ) شخص اعتبارسنجی متفاوتی( **کار خاص** ) رو انجام بدیدم.مثلا در اینجا قراره چک کنیم اگه شخص مرد بود اسم زنونه انتخاب نکرده باشه و...
خب روش معمول به این شکل میتونه باشه

```
switch (person.Gender)
{
    case Gender.Male:
        if (IsMale(person.FirstName))
        {
            //Isvalid
        }
        break;
    case Gender.Female:
        if (IsFemale(person.FirstName))
        {
            //Isvalid
        }
        break;
}
```

خب این روش خوب جواب میده ولی باید در حد توان استفاده از IF و Switch رو کم کرد.مثلا تو همین مثال ما اصل Open/Closed رو نقض کردیم فکر کنید قرار باشه اعتبارسنجی دیگه ای از همین دست به این کد(کلاس) اضافه بشه باید تغیرش بدیم پس این کد(کلاس) برای تغییر بسته نیست.در اینجور موارد «الگوی Delegate Dictionary» به کار ما میاد.
ما میایم توابع مورد نظرمون رو داخل یک Dictionary ذخیره میکنیم.

```
var genderFuncs = new Dictionary<Gender, Func<string, bool>>
                {
                    {Gender.Male , (x) => IsMale(x)},
                    {Gender.Female , (x) => IsFemale(x)}
                };
```

فرض کنید پیاده سازی توابع به شکل زیر باشه

```
public static bool IsMale(string name)
{
    //check...
    return true;
}
public static bool IsFemale(string name)
```

```
{
    //check...
    if (name == "Farzad")
    {
        return false;
    }
    return true;
}
```

نحوه استفاده

```
var dummyPerson = new List<Person>
                {
                    new Person
                        {Id = 1, Gender = Gender.Male, FirstName = "Mohammad", LastName = "Saheb"},
                    new Person
                        {Id = 2, Gender = Gender.Female, FirstName = "Farzad", LastName = "Mojidi"}
                };

foreach (var person in dummyPerson)
{
    bool isValid = genderFuncs[person.Gender].Invoke(person.FirstName);
}
```

با همین روش میشه قسمت آخر مقاله ی خوب آقای کیاست رو هم Refactor کرد.

```
var query = context.Students.AsQueryable();
  if (searchByName)
  {
      query= query.FindStudentsByName(name);
  }
  if (orderByAge)
  {
      query = query.OrderByAge();
  }
  if (paging)
  {
     query =  query.SkipAndTake(skip, take);
  }
  return query.ToList();
```

توابع رو داخل یک دیکشنری ذخیره میکنیم

```
var searchTypeFuncs = new Dictionary<SearchType, Func<IQueryable<Student>, string,
IQueryable<Student>>>
                    {
                        {SearchType.FirstName, (x, y) => x.FindStudentsByName(y)},
                        {SearchType.LastName, (x, y) => x.FindStudentsByLastName(y)}
                    };
```

نحوه استفاده

```
public static IList<Student> SearchStudents(IQueryable<Student> students, SearchType type, string
keyword)
{
    var result = searchTypeFuncs[type].Invoke(students, keyword);
    return result.ToList();
}
```