عنوان: **ذخیره تنظیمات متغیر مربوط به یک وب اپلیکیشن ASP.NET MVC با استفاده از EF**
نویسنده: غلامرضا ربال
تاریخ: ۱۳۹۴/۰۴/۲۷ ۹:۱۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MVC, Reflection, ASP.NET Identity, Caching

طی این مقاله، **نحوه‌ی ذخیره سازی تنظیمات متغیر و پویای یک برنامه را به صورت Strongly Typed** ارائه خواهم داد. برای این منظور، یک API را که از Reflection ، Cache ، Lazy Loading و Entity Framework بهره میگیرد، خواهیم ساخت.
برنامه‌ی هدف ما که از این API استفاده می‌کند، یک اپلیکیشن Asp.net MVC است. قبل از شروع به ساخت API مورد نظر، یک دید کلی در مورد آنچه که قرار است در نهایت توسعه یابد، در زیر مشاهده میکنید:

```
public SettingsController(ISettings settings)
{
  // example of saving
  _settings.General.SiteName = "دات نت تیپس";
  _settings.Seo.HomeMetaTitle = ".Net Tips";
  _settings.Seo.HomeMetaKeywords = "Asp.net MVC,Entity Framework,Reflection";
  _settings.Seo.HomeMetaDescription = "ذخیره تنظیمات برنامه";
  _settings.Save();
}
```

همانطور که در کدهای بالا مشاهده میکنید، شی setting_ ما دارای دو پراپرتی فقط خواندنی بنام‌های General و Seo است که شامل تنظیمات مورد نظر ما هستند و این دو کلاس از کلاس پایه‌ی SettingBase ارث بری کرده‌اند. دو دلیل برای انجام این کار وجود دارد:
تنظیمات به صورت گروه بندی شده در کنار هم قرار گرفته‌اند و یافتن تنظیمات برای زمانی که نیاز به دسترسی به آنها داریم، راحت‌تر و ساده‌تر خواهد بود.

به این شکل تنظیمات قابل دسترس در یک گروه، از دیتابیس بازیابی خواهند شد.

اصلا چرا باید این تنظیمات را در دیتابیس ذخیره کنیم؟

شاید فکر کنید چرا باید تنظیمات را در دیتابیس ذخیره کنیم در حالی که فایل web.config در درسترس است و می‌توان توسط کلاس ConfigurationManager به اطلاعات آن دسترسی داشت. **جواب:** دلیل این است که با تغییر فایل web.config، برنامه‌ی وب شما ری استارت خواهد شد (چه زمان‌هایی یک برنامه Asp.net ری استارت میشود).
برای جلوگیری از این مساله، راه حل مناسب برای ذخیره سازی اطلاعاتی که نیاز به تغییر در زمان اجرا دارند، استفاده از از دیتابیس می‌باشد. در این مقاله از Entity Framework و پایگاه داده Sql Sever استفاده می‌کنم.

مراحل ساخت Setting API مورد نظر به شرح زیر است:
ساخت یک Asp.net Web Application

ساخت مدل Setting و افزودن آن به کانتکست Entity Framework

ساخت کلاس SettingBase برای بازیابی و ذخیره سازی تنظیمات با رفلکشن

ساخت کلاس GenralSettins و SeoSettings که از کلاس SettingBase ارث بری کرده‌اند.

ساخت کلاس Settings به منظور مدیریت تمام انواع تنظیمات

یک برنامه‌ی Asp.Net Web Application را از نوع MVC ایجاد کنید. تا اینجا مرحله‌ی اول ما به پایان رسید؛ چرا که ویژوال استودیو کارهای مورد نیاز ما را انجام خواهد داد.
لازم است مدل خود را به ApplicationDbContext موجود در فایل IdentityModels.cs معرفی کنیم. به شکل زیر:

```
namespace DynamicSettingAPI.Models
{
    public interface IUnitOfWork
    {
        DbSet<Setting> Settings { get; set; }
        int SaveChanges();
    }
}

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>,IUnitOfWork
    {
        public DbSet<Setting> Settings { get; set; }
        public ApplicationDbContext()
            : base("DefaultConnection", throwIfV1Schema: false)
        {
        }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }
    }


namespace DynamicSettingAPI.Models
{
    public class Setting
    {
        public string Name { get; set; }
        public string Type { get; set; }
        public string Value { get; set; }
    }
}
```

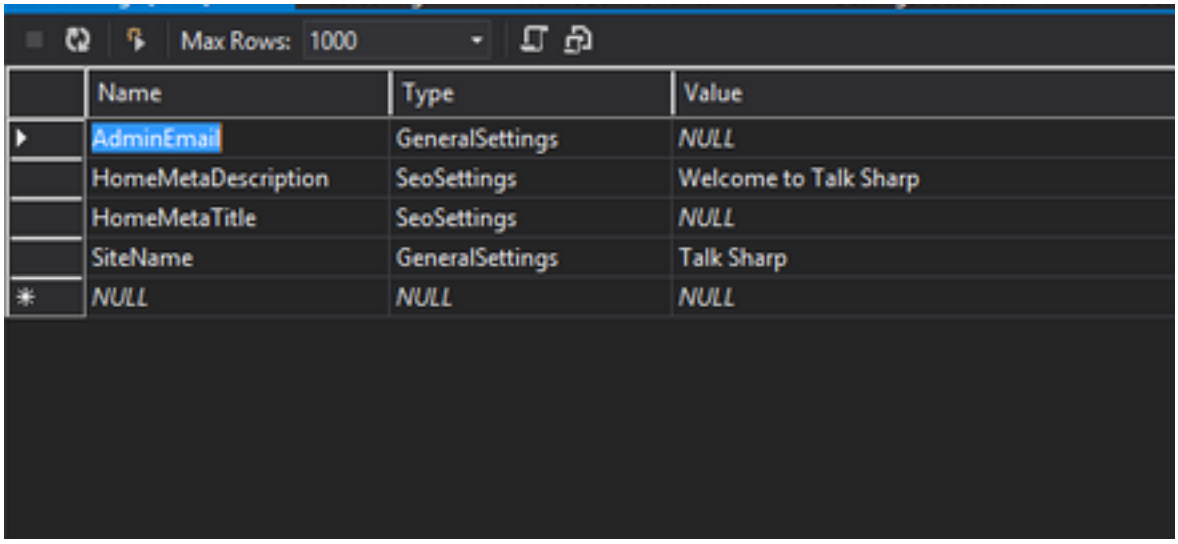
مدل تنظیمات ما خیلی ساده است و دارای سه پراپرتی به نام‌های Value ، Type ، Name هست که به ترتیب برای دریافت مقدار تنظیمات، نام کلاسی که از کلاس SettingBase ارث برده و نام تنظیمی که لازم داریم ذخیره کنیم، در نظر گرفته شده‌اند.
لازم است تا متد OnModelCreating مربوط به ApplicationDbContext را نیز تحریف کنیم تا کانفیگ مربوط به مدل خود را نیز اعمال نمائیم.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Setting>()
                    .HasKey(x => new { x.Name, x.Type });

            modelBuilder.Entity<Setting>()
                        .Property(x => x.Value)
                        .IsOptional();

            base.OnModelCreating(modelBuilder);
        }
```

ساختاری به شکل زیر مد نظر ماست:



کلاس SettingBase ما همچین ساختاری را خواهد داشت:

```
namespace DynamicSettingAPI.Service
{
    public abstract class SettingsBase
    {
        //1
        private readonly string _name;
        private readonly PropertyInfo[] _properties;

        protected SettingsBase()
        {
            //2
            var type = GetType();
            _name = type.Name;
            _properties = type.GetProperties();
        }

        public virtual void Load(IUnitOfWork unitOfWork)
        {
            //3 get setting for this type name
            var settings = unitOfWork.Settings.Where(w => w.Type == _name).ToList();

            foreach (var propertyInfo in _properties)
            {
                //get the setting from setting list
                var setting = settings.SingleOrDefault(s => s.Name == propertyInfo.Name);
                if (setting != null)
                {
                    //4 set
                    propertyInfo.SetValue(this, Convert.ChangeType(setting.Value,
propertyInfo.PropertyType));
                }
            }
        }
        public virtual void Save(IUnitOfWork unitOfWork)
        {
            //5 get all setting for this type name
            var settings = unitOfWork.Settings.Where(w => w.Type == _name).ToList();

            foreach (var propertyInfo in _properties)
            {
                var propertyValue = propertyInfo.GetValue(this, null);
                var value = (propertyValue == null) ? null : propertyValue.ToString();

                var setting = settings.SingleOrDefault(s => s.Name == propertyInfo.Name);
                if (setting != null)
                {
                    // 6 update existing value
                    setting.Value = value;
                }
                else
                {
```

```
                    // 7 create new setting
                    var newSetting = new Setting()
                    {
                        Name = propertyInfo.Name,
                        Type = _name,
                        Value = value,
                    };
                    unitOfWork.Settings.Add(newSetting);
                }
            }
        }
    }
}
```

این کلاس قرار است توسط کلاس‌های تنظیمات ما به ارث برده شود و در واقع کارهای مربوط به رفلکشن را در این کلاس کپسوله کرده‌ایم. همانطور که مشخص است ما دو فیلد را به نام‌های _name و _properties به صورت فقط خواندنی در نظر گرفته ایم که نام کلاس مورد نظر ما که از این کلاس به ارث خواهد برد، به همراه پراپرتی‌های آن، در این ظرف‌ها قرار خواهند گرفت.
متد Load وظیفه‌ی واکشی تمام تنظیمات مربوط به Type و ست کردن مقادیر به دست آمده را به خصوصیات کلاس ما، برعهده دارد. کد زیر مقدار دریافتی از دیتابیس را به نوع داده پراپرتی مورد نظر تبدیل کرده و نتیجه را به عنوان Value پراپرتی ست میکند.

```
propertyInfo.SetValue(this, Convert.ChangeType(setting.Value, propertyInfo.PropertyType));
```

متد Save نیز وظیفه‌ی ذخیره سازی مقادیر موجود در خصوصیات کلاس تنظیماتی را که از کلاس SettingBase ما به ارث برده است، به عهده دارد.
این متد دیتاهای موجود دردیتابیس را که متعلق به کلاس ارث برده مورد نظر ما هستند، واکشی میکند و در یک حلقه، اگر خصوصیتی در دیتابیس موجود بود، آن را ویرایش کرده وگرنه یک رکورد جدید را ثبت میکند.

کلاس‌های تنظیمات شخصی سازی شده خود را به شکل زیر تعریف میکنیم :

```
public class GeneralSettings : SettingsBase
    {
        public string SiteName { get; set; }
        public string AdminEmail { get; set; }
        public bool RegisterUsersEnabled { get; set; }
    }

 public class GeneralSettings : SettingsBase
    {
        public string SiteName { get; set; }
        public string AdminEmail { get; set; }
    }
```

نیازی به توضیح ندارد.
برای اینکه تنظیمات را به صورت یکجا داشته باشیم و Abstraction ای را برای استفاده از این API ارائه دهیم، یک اینترفیس و یک کلاس که اینترفیس مذکور را پیاده کرده است در نظر میگیریم:

```
public interface ISettings
{
    GeneralSettings General { get; }
    SeoSettings Seo { get; }
    void Save();
}

public class Settings : ISettings
{
    // 1
    private readonly Lazy<GeneralSettings> _generalSettings;
    // 2
    public GeneralSettings General { get { return _generalSettings.Value; } }

    private readonly Lazy<SeoSettings> _seoSettings;
    public SeoSettings Seo { get { return _seoSettings.Value; } }
```

```
    private readonly IUnitOfWork _unitOfWork;
    public Settings(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
        // 3
        _generalSettings = new Lazy<GeneralSettings>(CreateSettings<GeneralSettings>);
        _seoSettings = new Lazy<SeoSettings>(CreateSettings<SeoSettings>);
    }

    public void Save()
    {
        // only save changes to settings that have been loaded
        if (_generalSettings.IsValueCreated)
            _generalSettings.Value.Save(_unitOfWork);

        if (_seoSettings.IsValueCreated)
            _seoSettings.Value.Save(_unitOfWork);

        _unitOfWork.SaveChanges();
    }
    // 4
    private T CreateSettings<T>() where T : SettingsBase, new()
    {
        var settings = new T();
        settings.Load(_unitOfWork);
        return settings;
    }
}
```

این اینترفیس مشخص می‌کند که ما به چه نوع تنظیماتی، دسترسی داریم و متد Save آن برای آپدیت کردن تنظیمات، در نظر گرفته شده است. هر کلاسی که از کلاس SettingBase ارث بری کرده را به صورت فیلد فقط خواندنی و با استفاده از کلاس Lazy درون آن ذکر میکنیم و به این صورت کلاس تنظیمات ما زمانی ساخته خواهد شد که برای اولین بار به آن دسترسی داشته باشیم.
متد CreateSetting وظیفه‌ی لود دیتا را از دیتابیس، بر عهده دارد که برای این منظور، متد لود Type مورد نظر را فراخوانی میکند. این متد وقتی به کلاس تنظیمات مورد نظر برای اولین بار دسترسی پیدا کنیم، فراخوانی خواهد شد.

حتما امکان این وجود دارد که شما از امکان Caching هم بهره ببرید برای مثال همچین متد و سازنده‌ای را در کلاس Settings در نظر بگیرید:

```
private readonly ICache _cache;
public Settings(IUnitOfWork unitOfWork, ICache cache)
{
    // ARGUMENT CHECKING SKIPPED FOR BREVITY
    _unitOfWork = unitOfWork;
    _cache = cache;
    _generalSettings = new Lazy<GeneralSettings>(CreateSettingsWithCache<GeneralSettings>);
    _seoSettings = new Lazy<SeoSettings>(CreateSettingsWithCache<SeoSettings>);
}

private T CreateSettingsWithCache<T>() where T : SettingsBase, new()
{
    // this is where you would implement loading from ICache
    throw new NotImplementedException();
}
```

در آخر هم به شکل زیر میتوان (به عنوان دمو فقط) از این API استفاده کرد.

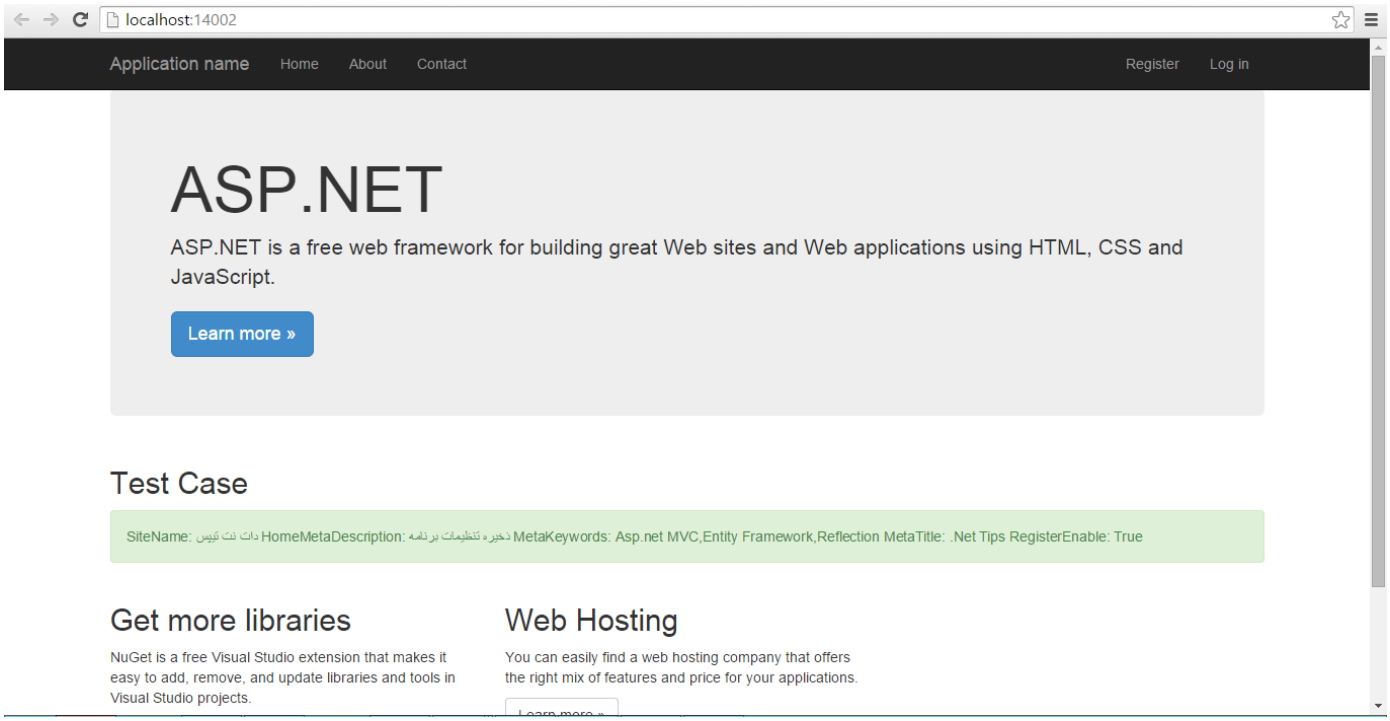
```
public ActionResult Index()
        {
            using (var uow = new ApplicationDbContext())
            {
                var _settings = new Settings(uow);
                _settings.General.SiteName = "دات نت تیپس";
                _settings.General.AdminEmail = "admin@gmail.com";
                _settings.General.RegisterUsersEnabled = true;
                _settings.Seo.HomeMetaTitle = ".Net Tips";
                _settings.Seo.MetaKeywords = "Asp.net MVC,Entity Framework,Reflection";
                _settings.Seo.HomeMetaDescription = "ذخیره تنظیمات برنامه";

                var settings2 = new Settings(uow);
```

```
                var output = string.Format("SiteName: {0} HomeMetaDescription: {1}  MetaKeywords:  {2}
MetaTitle:  {3}  RegisterEnable:  {4}",
                    settings2.General.SiteName,
                    settings2.Seo.HomeMetaDescription,
                    settings2.Seo.MetaKeywords,
                    settings2.Seo.HomeMetaTitle,
                    settings2.General.RegisterUsersEnabled.ToString()
                    );
                return Content(output);
            }

        }
```

خروجی :



نکته: در پروژه ای که جدیدا در سایت ارائه داده‌ام و در حال تکمیل آن هستم، از بهبود یافته‌ی این مقاله استفاده می‌شود. حتی برای اسلاید شوهای سایت هم میشود از این روش استفاده کرد و از فرمت json بهره برد برای این منظور. حتما در پروژه‌ی مذکور همچین امکانی را هم در نظر خواهم گرفتم.
پیشنها میکنم سورس SmartStore را بررسی کنید. آن هم به شکل مشابهی ولی پیشرفته‌تر از این مقاله، همچین امکانی را دارد.
DynamicSettingAPI.zip