عنوان: **بررسی مدیریت دسترسی در جوملا 2.5-1.6**
نویسنده: علی علوی
تاریخ: ۱۳۹۲/۰۳/۰۶ ۲۱:۳۵
آدرس: www.dotnettips.info
گروه‌ها: joomla, Authorization

مطابق با ویکی پدیا، سطوح دسترسی مشخص می‌کند که کدام کاربران یا سیستم پردازش اجازه دسترسی به اشیاء را دارند(Authentication)، همچنین چه عملیات‌هایی بر روی اشیاء مجازند که اجرا شوند(Authorization) .

در مورد جوملا ، ما دو جنبه جدا برای سطوح دسترسی داریم:

1. کدام کاربران به چه بخش‌هایی می‌توانند دسترسی داشته باشند؟ برای مثال، انتخاب یک منو برای کدام کاربر فعال خواهد بود؟

2. چه عملیات (یا اقداماتی) کاربر می‌تواند بر روی اشیاء داشته باشد؟ برای مثال، آیا کاربر می‌تواند یک مطلب را ارسال یا ویرایش کند؟

ماهیت‌های موجود در سیستم :

· کاربران

کاربر می‌تواند به گروه‌های مختلفی اختصاص یابد .

· گروه‌ها کاربری

شامل مجوزهایی به صورت پیش فرض می‌باشند که این مجوزها را از سطوح بالایی نیز به ارث می‌برند.

· سطوح دسترسی

شامل یک یا چند گروه کاربری می‌باشد و سطوح دسترسی به محتواهای سایت نسبت داده می‌شود یعنی اگر یک مطلب دارای سطح دسترسی **عمومی** باشد آنگاه تمامی گروه‌های کاربری که در **عمومی** وجود دارند می‌توانند مطلب را مشاهده کنند.

· عملیات و مجوزها

به صورت پیش فرض یک سری عملیات در سیستم تعریف شده است شامل ویرایش ، حذف و غیره که برای هر گروه کاربری (تعدادی گروه کاربری به صورت پیش فرض در سیستم تعریف شده است) به صورت پیش فرض مجوزهایی در نظر گرفته شده است که این مجوزها قابلیت ارث بری از والد گروه به فرزند رانیز دارا میباشد پس با این حساب همیشه در جوملا والد از سطح دسترسی پایین‌تری نسبت به فرزند برخوردار می‌باشد.

اما باید گفت در جوملا ب ه ازای هر کامپوننت نیز می‌توان این مجوزها را به ازای گروه‌های مختلف تغییر داد در این جا هم هر کامپوننت دارای مجوزهای پیش فرضی می‌باشد که در هنگام نصب کامپوننت برای آن در نظر گرفته می‌شود.

جداول این سیستم :

users : جدول کاربران

usergroups : جدول گروه‌های کاری یا همان نقش‌های کاربری

user_usergroup_map : جدول واسط بین کاربران و گروه‌های کاری به منظور ایجاد رابطه‌ی چند به چند (n:n)

assets : این جدول که از جوملا 1.6 به بعد به دیتابیس جوملا افزوده شده است مهمترین جدول در این سیستم می‌باشد . که در آن به ازای هر جز که سطح دسترسی باید برای آن لحاظ گردد یک سطر در نظر گرفته می‌شود که این سطر باتوجه به افزایش اجزا سیستم تغییر و به صورت داینامیک به جدول اضافه می‌گردد ضمنا این سطور قابلیت ارث بری از یکدیگر را نیز دارا می‌باشند. در هر بک از سطرها فیلدی به نام **ruls** وجود دارد محتوای این فیلد از نوع داده ای json می‌باشد با یک مثال شاید بهتر بتوان توضیح داد :

محتوای فیلد کامپوننت بنر :

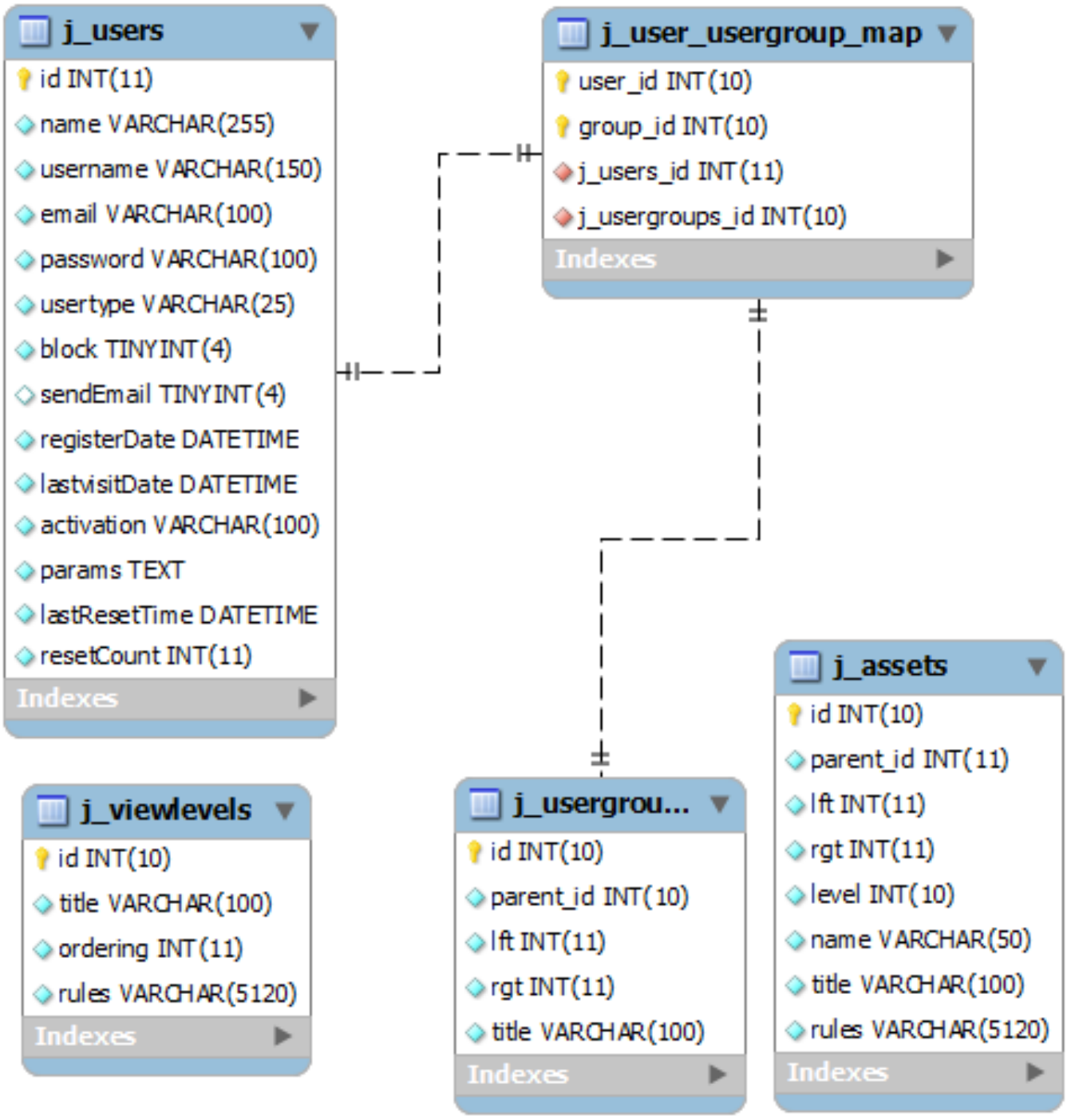
```
{core.admin":{"9":1,"7":1},"core.manage":{"6":1},"core.create":[],"core.delete":[],"core.edit " }
```

در این جا "core.admin" مجوز دسترسی مدیریتی به این کامپوننت می‌باشد که گروه‌های کاری شماره 7 و 9 دارای چنین دسترسی می‌باشند . ضمنا عملیات‌های " core.create " از سطوح بالاتر یا همان سطر والد خود ارث بری می‌کند.

Viewlevel : در این جدول سطوح دسترسی تعریف شده اند مهمترین فیلد این جدول نیز **ruls** نام دارد و حاوی id گروه هایی است که به این سطح دسترسی ، دسترسی دارند.

به طور مثال سطح دسترسی **ثبت نام شده** حاوی [6,2,8] می‌باشد یعنی گروه‌های کاری با id های مورد نظر می‌توانند به محتواهای با سطح دسترسی **ثبت نام شده** دسترسی داشته باشند.

دیاگرام جداول :


j_users
id INT(11)
name VARCHAR(255)
username VARCHAR(150)
email VARCHAR(100)
password VARCHAR(100)
usertype VARCHAR(25)
block TINYINT(4)
sendEmail TINYINT(4)
registerDate DATETIME
lastvisitDate DATETIME
activation VARCHAR(100)
params TEXT
lastResetTime DATETIME
resetCount INT(11)
Indexes
j_user_usergroup_map
user_id INT(10)
group_id INT(10)
j_users_id INT(11)
j_usergroups_id INT(10)
Indexes
j_assets
id INT(10)
parent_id INT(11)
lft INT(11)
rgt INT(11)
level INT(10)
name VARCHAR(50)
title VARCHAR(100)
rules VARCHAR(5120)
Indexes
j_usergrou...
id INT(10)
parent_id INT(10)
lft INT(11)
rgt INT(11)
title VARCHAR(100)
Indexes
j_viewlevels
id INT(10)
title VARCHAR(100)
ordering INT(11)
rules VARCHAR(5120)
Indexes

عنوان: **پیاده سازی authorization به روش AOP به کمک کتابخانه های SNAP و StructureMap**
نویسنده: کاوه شهبازی
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۷:۵۵
آدرس: www.dotnettips.info
برچسب‌ها: Authorization, AOP, SNAP, StructureMap

همانطور که پیشتر در این مقاله بحث شده است، بوسیله AOP می‌توان قابلیت‌هایی که قسمت عمده‌ای از برنامه را تحت پوشش قرار می‌دهند، کپسوله کرد. یکی از قابلیت‌هایی که در بخشهای مختلف یک سیستم نرم‌افزاری مورد نیاز است، Authorization یا اعتبارسنجی‌ست. در ادامه به بررسی یک پیاده‌سازی به این روش می‌پردازیم.

کتابخانه SNAP

کتابخانه SNAP به گفته سازنده آن، با یکپارچه‌سازی AOP با IoC Containerهای محبوب، برنامه‌نویسی به این سبک را ساده می‌کند. این کتابخانه هم اکنون علاوه بر structureMap از IoC Providerهای Autofac, Ninject, LinFu و Castle Windsor نیز پشتیبانی میکند.

دریافت SNAP.StructureMap

برای دریافت آن نیاز است دستور پاورشل ذیل را در کنسول نیوگت ویژوال استودیو اجرا کنید:

```
PM> Install-Package snap.structuremap
```

پس از اجرای دستور فوق، کتابخانه SNAP.StructureMap که در زمان نگارش این مطلب نسخه 1.8.0 آن موجود است به همراه کلیه نیازمندی‌های آن که شامل موارد زیر می‌باشد نصب خواهد شد.

```
StructureMap (≥ 2.6.4.1)
CommonServiceLocator.StructureMapAdapter (≥ 1.1.0.3)
SNAP (≥ 1.8)
fasterflect (≥ 2.1.2)
Castle.Core (≥ 3.1.0)
CommonServiceLocator (≥ 1.0)
```

تنظیمات SNAP

از آنجا که تنظیمات SNAP همانند تنظیمات StructureMap تنها باید یک بار اجرا شود، بهترین جا برای آن در یک برنامه وب، Application_Start فایل Global.asax است.

```
namespace Framework.UI.Asp
{
    public class Global : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            initSnap();
            initStructureMap();
        }

        private static void initSnap()
        {
            SnapConfiguration.For<StructureMapAspectContainer>(c =>
            {
                // Tell Snap to intercept types under the "Framework.ServiceLayer..." namespace.
                c.IncludeNamespace("Framework.ServiceLayer.*");
                // Register a custom interceptor (a.k.a. an aspect).
                c.Bind<Framework.ServiceLayer.Aspects.AuthorizationInterceptor>()
                .To<Framework.ServiceLayer.Aspects.AuthorizationAttribute>();
            });
        }

        void Application_EndRequest(object sender, EventArgs e)
        {
            ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects();
```

```
        }

        private static void initStructureMap()
        {
            var thread = StructureMap.Pipeline.Lifecycles.GetLifecycle(InstanceScope.HttpSession);
            ObjectFactory.Configure(x =>
            {
                x.For<IUserManager>().Use<EFUserManager>();
                x.For<IAuthorizationManager>().LifecycleIs(thread)
                 .Use<EFAuthorizationManager>().Named("_AuthorizationManager");
                x.For<Framework.DataLayer.IUnitOfWork>()
                 .Use<Framework.DataLayer.Context>();

                x.SetAllProperties(y =>
                {
                    y.OfType<IUserManager>();
                    y.OfType<Framework.DataLayer.IUnitOfWork>();
                    y.OfType<Framework.Common.Web.IPageHelpers>();
                });
            });
        }
    }
}
```

بخش اعظم کدهای فوق در مقاله‌های « استفاده از StructureMap به عنوان یک IoC Container » و « تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET Web forms » شرح داده شده‌اند، تنها بخش جدید متد initSnap() است، که خط اول آن به snap می‌گوید همه کلاس‌هایی که در فضای نام Framework.ServiceLayer و زیرمجموعه‌های آن هستند را پوشش دهد. خط دوم نیز کلاس AuthorizationInterceptor را به عنوان مرجعی برای handle کردن AuthorizationAttribute معرفی می‌کند.

در ادامه به بررسی کلاس AuthorizationInterceptor می‌پردازیم.

```
namespace Framework.ServiceLayer.Aspects
{
    public class AuthorizationInterceptor : MethodInterceptor
    {
        public override void InterceptMethod(IInvocation invocation, MethodBase method, Attribute
attribute)
        {
            var AuthManager = StructureMap.ObjectFactory
.GetInstance<Framework.ServiceLayer.UserManager.IAuthorizationManager>();
            var FullName = GetMethodFullName(method);
            if (!AuthManager.IsActionAuthorized(FullName))
                throw new Common.Exceptions.UnauthorizedAccessException("");

            invocation.Proceed(); // the underlying method call
        }

        private static string GetMethodFullName(MethodBase method)
        {
            var TypeName = (((System.Reflection.MemberInfo)(method)).DeclaringType).FullName;
            return TypeName + "." + method.Name;
        }
    }

    public class AuthorizationAttribute : MethodInterceptAttribute
    { }
```

کلاس مذکور از کلاس MethodInterceptor کتابخانه snap ارث بری کرده و متد InterceptMethod را تحریف میکند. این متد، کار اجرای متد اصلی ای که با این Aspect تزئین شده را بر عهده دارد. بنابراین می‌توان پیش از اجرای متد اصلی، اعتبارسنجی را انجام داد. **کلاس MethodInterceptor**

کلاس MethodInterceptor شامل چندین متد دیگر نیز هست که میتوان برای سایر مقاصد از جمله مدیریت خطا و Event logging از آنها استفاده کرد.

```
namespace Snap
{
```

```
    public abstract class MethodInterceptor : IAttributeInterceptor, IInterceptor, IHideBaseTypes
    {
        protected MethodInterceptor();

        public int Order { get; set; }
        public Type TargetAttribute { get; set; }

        public virtual void AfterInvocation();
        public virtual void BeforeInvocation();
        public void Intercept(IInvocation invocation);
        public abstract void InterceptMethod(IInvocation invocation, MethodBase method, Attribute
attribute);
        public bool ShouldIntercept(IInvocation invocation);
    }
}
```

یک نکته

نکته مهمی که در اینجا پیش می‌آید این است که برای اعتبارسنجی، کد کاربری شخصی که لاگین کرده، باید به طریقی در اختیار متد IsActionAuthorized() قرار بگیرد. برای این کار می‌توان در یک HttpMudole به عنوان مثال همان ماژولی که برای تسهیل در کار تزریق خودکار وابستگی‌ها در سطح فرم‌ها استفاده می‌شود، با استفاده از امکانات structureMap به وهله‌ی ایجاد شده از AuthorizationManager (که با کمک structureMap با طول عمر InstanceScope.HttpSession ساخته شده است) دسترسی پیدا کرده و خاصیت مربوطه را مقداردهی کرد.

```
private void Application_PreRequestHandlerExecute(object source, EventArgs e)
        {
            var page = HttpContext.Current.Handler as BasePage; // The Page handler
            if (page == null)
                return;

            WireUpThePage(page);
            WireUpAllUserControls(page);

            var UsrCod = HttpContext.Current.Session["UsrCod"];
            if (UsrCod != null)
            {
                var _AuthorizationManager = ObjectFactory

.GetNamedInstance<Framework.ServiceLayer.UserManager.IAuthorizationManager>("_AuthorizationManager");

                ((Framework.ServiceLayer.UserManager.EFAuthorizationManager)_AuthorizationManager)
                    .AuditUserId = UsrCod.ToString();
            }
        }
```

روش استفاده

نحوه استفاده از Aspect تعریف شده در کد زیر قابل مشاهده است:

```
namespace Framework.ServiceLayer.UserManager
{
    public class EFUserManager : IUserManager
    {
        IUnitOfWork _uow;
        IDbSet<User> _users;

        public EFUserManager(IUnitOfWork uow)
        {
            _uow = uow;
            _users = _uow.Set<User>();
        }

        [Framework.ServiceLayer.Aspects.Authorization]
        public List<User> GetAll()
        {
            return _users.ToList<User>();
        }
    }
}
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۹:۷

با تشکر از شما. چند سؤال: متد AuthManager.IsActionAuthorized چطور تعریف شده؟ و همچنین EFAuthorizationManager حدودا چه تعریفی داره؟

نویسنده: کاوه شهبازی
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۹:۵۱

۱- متد IsActionAuthorized نام کامل متدی که قرار است اجرا شود را به عنوان پارامتر گرفته و در دیتابیس (در این پیاده سازی به وسیله‌ی EntityFramework) چک میکند که کاربری که Id اش در AuthManager.AuditUserId است (یعنی کاربری که درخواست اجرای متد را داده است) اجازه اجرای این متد را دارد یا نه. بسته به نیازمندی برنامه شما این دسترسی میتواند به طور ساده فقط مستقیما برای کاربر ثبت شود و یا ترکیبی از دسترسی خود کاربر و دسترسی گروه هایی که این کاربر در آن عضویت دارد باشد.

۲- EFAuthorizationManager کلاس ساده ایست

```
namespace Framework.ServiceLayer.UserManager
{
    public class EFAuthorizationManager : IAuthorizationManager
    {
        public String AuditUserId { get; set; }
        IUnitOfWork _uow;

        public EFAuthorizationManager(IUnitOfWork uow)
        {
            _uow = uow;
        }

        public bool IsActionAuthorized(string actionName)
        {
            var res = _uow.Set<User>()
            .Any(u => u.Id == AuditUserId &&
                    u.AllowedActions.Any(a => a.Name == actionName));
            return res;
        }

        public bool IsPageAuthorized(string pageURL)
        {
            //TODO: بررسی وجود دسترسی باید پیاده سازی شود
            //فقط برای تست
            return true;
        }
    }
}
```

خلاصه ای از کلاسهای مدل مرتبط را هم در زیر مشاهده می‌کنید:

```
namespace Framework.DataModel
{
    public class User : BaseEntity
    {
        public string UserName { get; set; }
        public string Password { get; set; }

        //...

        [Display(Name = "عملیات مجاز")]
        public virtual ICollection<Action> AllowedActions { get; set; }
    }

    public class Action:BaseEntity
    {
        public string Name { get; set; }
        public Entity RelatedEntity { get; set; }
```

```
        //...

        public virtual ICollection<User> AllowedUsers { get; set; }
    }

    public abstract class BaseEntity
    {
        [Key]
        public int Id { get; set; }
        //...
    }
}
```