Авторизация на основе Claims

Последнее обновление: 06.12.2015

Инструмент Claims в рамках ASP.NET Identity представляет нам прекрасный способ для создания системы авторизации, которая основана на самых различных свойствах объекта или так называемая **Claims-Based Authorization**. Например, если мы хотим разграничить доступ к ресурсам в зависимости от возраста пользователя, то с помощью Claims мы это легко можем сделать.

Итак, создадим новый проект с типом авторизации Individual User Accounts. Пусть нам надо добавить разграничение доступа по возрасту пользователя. Для этого прежде всего изменим подсистему регистрации.

В модель RegisterModel из файла *Models/AccountViewModels.cs* добавим свойство Year для хранения года рождения пользователя:

```
public class RegisterViewModel
{
    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    public string ConfirmPassword { get; set; }

    [Required]
    [Range(1910, 2015)]
    [Display(Name = "Year")]
    public int Year { get; set; }
}
```

И также добавим в представление дл регистрации *Register.cshtml* код для создания поля ввода года рождения:

```
<div class="form-group">
    @Html.LabelFor(m => m.Year, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.TextBoxFor(m => m.Year, new { @class = "form-control" })
    </div>
</div>
```

Итак, в модели RegisterModel определено свойство Year, а в представлении ля него имеется поле ввода. Теперь изменим post-версию метода Register, чтобы использовать это свойство:

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
```

```
    {
        var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            // создаем claim для хранения года рождения
            var identityClaim = new IdentityUserClaim { ClaimType = "Year", ClaimValue = model.Year.ToString() };
            // добавляем claim пользователю
            user.Claims.Add(identityClaim);
            // сохраняем изменения
            await UserManager.UpdateAsync(user);

            await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

            return RedirectToAction("Index", "Home");
        }
        AddErrors(result);
    }
    return View(model);
}
```

После создания пользователя вначале формируем объект **IdentityUserClaim**:

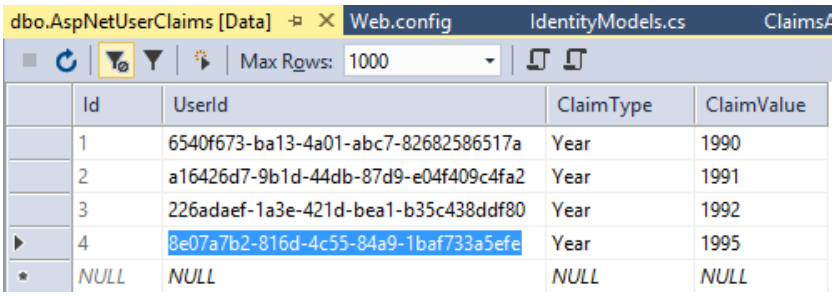
```
var identityClaim = new IdentityUserClaim { ClaimType = "Year", ClaimValue = model.Year.ToString() };
```

Чтобы задействовать этот класс, надо подключить пространство имен `Microsoft.AspNet.Identity.EntityFramework`. Свойство `ClaimType` отвечает за тип данных. Поскольку здесь сохраняется год, то и тип называется Year. А свойство `ClaimValue` хранит значение для этого типа в виде строки.

Далее claim добавляется пользователю, и происходит обновление:

```
user.Claims.Add(identityClaim);
await UserManager.UpdateAsync(user);
```

В итоге claim будет сохранен в базе данных. Для хранения claim в ASP.NET Identity используется таблица **AspNetUserClaims**. А сохраненые данные будут выглядеть примерно так:



Id	UserId	ClaimType	ClaimValue
1	6540f673-ba13-4a01-abc7-82682586517a	Year	1990
2	a16426d7-9b1d-44db-87d9-e04f409c4fa2	Year	1991
3	226adaef-1a3e-421d-bea1-b35c438ddf80	Year	1992
4	8e07a7b2-816d-4c55-84a9-1baf733a5efe	Year	1995
NULL	NULL	NULL	NULL

В данном случае определяется claim "Year" для хранения года рождения, но также мы можем сохранять и любую другую информацию о пользователе.

Затем изменим класс **ApplicationUser**, чтобы он сохранял данные своего claim в куках:

```
public class ApplicationUser : IdentityUser
{
    public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
    {
        var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);

        var yearClaim = Claims.FirstOrDefault(c => c.ClaimType == "Year");
        if (yearClaim != null)
            userIdentity.AddClaim(new Claim(yearClaim.ClaimType, yearClaim.ClaimValue));
        return userIdentity;
    }
}
```

И в конце нам надо создать атрибут, который бы позволил устанавливать ограничения по доступу. Для этого добавим в проект новую папку Util, и затем в нее добавим новый класс **ClaimsAuthorizeAttribute**:

```
using System;
using System.Security.Claims;
using System.Web;
using System.Web.Mvc;
```

```
namespace AspClaimsApp.Util
{
    public class ClaimsAuthorizeAttribute : AuthorizeAttribute
    {
        public int Age { get; set; }

        protected override bool AuthorizeCore(HttpContextBase httpContext)
        {
            ClaimsIdentity claimsIdentity;
            if (!httpContext.User.Identity.IsAuthenticated)
                return false;

            claimsIdentity = httpContext.User.Identity as ClaimsIdentity;
            var yearClaims = claimsIdentity.FindFirst("Year");
            if (yearClaims == null)
                return false;

            int year; // получаем год
            if (!Int32.TryParse(yearClaims.Value, out year))
                return false;

            // проверяем возраст относительно текущей даты
            if ((DateTime.Now.Year - year) < this.Age)
                return false;

            // обращаемся к методу базового класса
            return base.AuthorizeCore(httpContext);
        }
    }
}
```

Данный класс наследуется от атрибута авторизации **AuthorizeAttribute** и добавляет свойство `Age`, которое позволит устанавливать ограничения по возрасту.

В самом классе с помощью получаемого контекста запроса httpContext извлекаем claim и устанавливаем, имеет ли текущий пользователь доступ к ресурсу. Если пользователь проходит авторизацию, то возвращается `true`, если нет, то `false`.

И после этого мы сможем использовать данный атрибут в контроллере. Например:

```
public class HomeController : Controller
{
    [ClaimsAuthorize(Age=18)]
    public ActionResult Index()
    {
        return View();
    }

    [ClaimsAuthorize(Age = 22)]
    public ActionResult About()
    {
        return View();
    }
}
```

Назад **Содержание** **Вперед**

17 Комментариев metanit.com **Войти**

Рекомендовать **Поделиться** **Лучшее в начале**

Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ ИЛИ ЧЕРЕЗ DISQUS ?

Имя

AI • месяц назад

А у Вас есть статьи про SignInManager? В каких случаях его стоит использовать в реальных приложениях?

Ответить • Поделиться ›

Metanit Модератор ➚ Al • месяц назад

по сути он просто устанавливает аутентификационные куки и больше ничего, особо то и нечего добавить

^ | v • Ответить • Поделиться ›

Артем • 2 года назад

Подскажите, а как сделать в Identiny объект типа User (авторизованный пользователь) доступным из всех контроллеров? А то приходится в каждом методе каждого контроллера его вытаскивать из БД. Существуют какие-то глобальные переменные для этого?

^ | v • Ответить • Поделиться ›

Metanit Модератор ➚ Артем • 2 года назад

сохраняйте все необходимые данные в claims - собственно они для этого и нудны, другого способа нет

^ | v • Ответить • Поделиться ›

Артем ➚ Metanit • 2 года назад

А как сохранить пользовательский объект в Claim? Я имею в виду прямо весь объект User со всеми полями, которые я сам ему придумал.

^ | v • Ответить • Поделиться ›

Metanit Модератор ➚ Артем • 2 года назад

по отдельности сохранять все поля

^ | v • Ответить • Поделиться ›

Anon • 2 года назад

Не совсем понятно зачем добавлять клейм в методе регистрации, если он добавляется и в методе GenerateUserIdentityAsync.

^ | v • Ответить • Поделиться ›

Anon ➚ Anon • 2 года назад

Вернее наоборот: зачем добавлять клейм в методе GenerateUserIdentityAsync, если он уже добавился в методе регистрации?

^ | v • Ответить • Поделиться ›

Metanit Модератор ➚ Anon • 2 года назад

в методе регистрации добавляется объект IdentityUserClaim в базу данных, он там хранится - хранение на сервере.
в методе GenerateUserIdentityAsync добавляется объект Claim (это другой уже тип) в куки в браузере - хранение на клиенте

^ | v • Ответить • Поделиться ›

Anon ➚ Metanit • 2 года назад

Просто у вас есть тема http://metanit.com/sharp/mv... где, как я понял, клейм в бд не заносится, а просто устанавливается в куки. Вот я и спросил, т.к. подумал что операция лишняя.

^ | v • Ответить • Поделиться ›

Metanit Модератор ➚ Anon • 2 года назад

ага, ну в этом и разница, там в принципе это тоже хранится в бд, но в таблице самих пользователей. Здесь используется непосредственно отведенная для claims таблица

^ | v • Ответить • Поделиться ›

Ярослав Орлов ➚ Metanit • 8 месяцев назад

var yearClaim = Claims.FirstOrDefault(c => c.ClaimType == "Year");

Claims - это какой-то объект из предка пренадлежащий именно текущему пользователю?

^ | v • Ответить • Поделиться ›

Metanit Модератор ➚ Ярослав Орлов • 8 месяцев назад

не совсем из предка. Пользователь в системе аутентификации представляет объект ClaimsIdentity, у которого есть коллекция Claims. В эту коллекцию мы можем сохранять какие-то данные из бд из таблицы пользователей. Например, id текущего пользователя

^ v • Ответить • Поделиться ›

Ярослав Орлов ➚ Metanit • 8 месяцев назад

А разве не объект IPrincipal?

^ v • Ответить • Поделиться ›

Metanit Модератор ➚ Ярослав Орлов • 8 месяцев назад

Не, IPrincipal хранит ссылку на IIdentity, А ClaimsPrincipal - реализация IPrincipal хранит объект ClaimsIdentity - реализацию интерфейса IIdentity. Смотрите тут- http://metanit.com/sharp/as... - хотя для asp net core, но в данном случае то же самое будет и для asp net mvc 5 с Owin+Identity 2.0

^ v • Ответить • Поделиться ›

Ярослав Орлов ➚ Metanit • 8 месяцев назад

Ага, то есть как я понял, свойство User контекста возврщает объект IPrincipal дефолтной реализацией которого является ClaimsPrincipal, в свою очередь у этого интерфейса есть свойство Identity типа IIdentity дефолтной реализацией которого есть ClaimsIdentity - объект хранящий клэймы пользователя. Верно?

^ v • Ответить • Поделиться ›

Metanit Модератор ➚ Ярослав Орлов • 8 месяцев назад