

Утверждения (Claims) в ASP.NET Identity

[ASP.NET](#) --- [ASP.NET Identity](#) --- [Утверждения \(Claims\)](#)

1 В старых системах управления пользователями, таких как Membership API, приложение считается авторитетным источником всей информации о пользователе. По сути приложение рассматривается как некий «закрытый мир» и система аутентификации доверяет данным, содержащимся в нем. Это укоренившийся подход к разработке программного обеспечения, который вы могли видеть на примере нашего приложения — мы аутентифицируем пользователей на основе учетных записей, хранящихся в базе данных и предоставляем доступ на основе ролей, связанных с этими учетными записями.

ASP.NET Identity предоставляет также альтернативный подход для работы с пользователями, который хорошо работает в рамках MVC-приложения, использующего несколько источников информации о пользователях, и который может быть использован для авторизации пользователей более гибким способом, чем традиционные роли.

Этот альтернативный подход использует **утверждения (claims)**, и в этой статье я покажу как использовать такой тип авторизации. Давайте ответим на ряд стандартных вопросов, которые могут возникнуть у вас в данный момент:

Что это?

Утверждения (claims) – это дополнительная информация о пользователях, на основе которой можно принимать решения по авторизации. Эти утверждения могут быть получены из внешних источников данных, а также из локальной базы данных Identity.

Зачем нужно использовать?

Утверждения обеспечивают гибкий подход к системе авторизации. В отличие от ролей, утверждения авторизуют пользователя на основе информации, которая описывает этого пользователя. Пройди тесты C# тест (легкий) .NET тест (средний)

Как использовать в рамках MVC?

Утверждения не используются непосредственно в рамках платформы MVC, но они интегрированы в стандартный функционал авторизации, такой как атрибут `Authorize`.

Описание концепции утверждений

Утверждение (claim) — это информация о пользователе, а также информация откуда взялась эта информация (да, получилась тавтология). В ASP.NET Identity оно представлено классом **Claim**. Самым простым способом описать утверждения, является наглядный пример. Для начала добавьте в проект контроллер Claims со следующим содержимым:

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

namespace Users.Controllers
{
    public class ClaimsController : Controller
    {
        [Authorize]
        public ActionResult Index()
        {
            ClaimsIdentity ident = HttpContext.User.Identity as ClaimsIdentity;

            if (ident == null)
            {
                return View("Error", new string[] { "Нет доступных требований" });
            }
            else
            {
                return View(ident.Claims);
            }
        }
    }
}
```

Вы можете по-разному получить данные об утверждениях. В этом примере мы использовали **свойство Claims**, определенное в классе пользователя **ClaimsIdentity**. Экземпляр этого класса можно получить из свойства `Identity` объекта `IPrincipal`, который, в свою очередь, можно получить из свойства `HttpContext.User`. Свойство `Identity` возвращает реализацию `IIdentity`, которую в ASP.NET Identity реализует класс `ClaimsIdentity`. Следующая таблица содержит описание свойств и методов этого класса:

Свойства и методы класса ClaimsIdentity

Название	Описание
<i>Claims</i>	Возвращает коллекцию объектов Claim, описывающих утверждения для пользователя.
<i>AddClaim(claim)</i>	Добавляет новое утверждение к информации о пользователе.
<i>AddClaims(claims)</i>	Добавляет список утверждений к информации о пользователе.
<i>HasClaim(predicate)</i>	Возвращает true, если информация о пользователе содержит утверждение с заданным предикатом.
<i>RemoveClaim(claim)</i>	Удаляет утверждение для пользователя.

Перечисленные в таблице члены класса ClaimsIdentity, являются самыми широко используемыми. С помощью этих свойств и методов я продемонстрирую, как утверждения вписываются в более общую платформу ASP.NET.

В примере выше мы привели реализацию IIdentity к типу ClaimsIdentity, и вернули коллекцию объектов Claim в представление с помощью свойства Claims. Объект Claim описывает один элемент данных о пользователе и его свойства перечислены в таблице ниже:

Пройди тесты C# тест (легкий) .NET тест (средний)

Свойства класса Claim

Название	Описание
<i>Issuer</i>	Возвращает название источника, откуда поступают данные о пользователе
<i>Subject</i>	Возвращает базовый объект ClaimsIdentity, связанный с текущим утверждением
<i>Type</i>	Возвращает тип информации об утверждении
<i>Value</i>	Возвращает информацию об утверждении

Давайте используем эти свойства в представлении Index.cshtml, которое добавим в папку /Views/Claims. Это представление содержит таблицу с описанием каждого утверждения, связанного с пользователем:

```
@using System.Security.Claims
@using Users.Infrastructure

@model IEnumerable<Claim>

@{ ViewBag.Title = "Требования"; }

<div class="panel panel-primary">
    <div class="panel-heading">
        Требования
    </div>
    <table class="table table-striped">
        <tr>
            <th>Subject</th>
            <th>Issuer</th>
            <th>Type</th>
            <th>Value</th>
        </tr>
        @foreach (Claim claim in Model.OrderBy(x => x.Type))
        {
            <tr>
                <td>@claim.Subject.Name</td>
                <td>@claim.Issuer</td>
                <td>@Html.ClaimType(claim.Type)</td>
                <td>@claim.Value</td>
            </tr>
        }
    </table>
</div>
```

Свойство Claim.Type возвращает URI для схемы Microsoft, который не очень полезен. Популярные схемы описываются специальным **классом ClaimTypes** из пространства имен System.Security.Claims. Чтобы извлечь полезную информацию из объектов этого класса, в представлении Index.cshtml мы использовали вспомогательный метод Html.ClaimType(). Реализацию этого метода необходимо добавить в файл IdentityHelper.cs, содержащий расширяющие методы для класса HtmlHelper:

Пройди тесты C# тест (легкий) .NET тест (средний)

```
using System;
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity.Owin;
using System.Reflection;
using System.Security.Claims;
using System.Linq;

namespace Users.Infrastructure
{
    public static class IdentityHelpers
    {
        public static MvcHtmlString GetUserName(this HtmlHelper html, string id)
        {
            // ...
        }

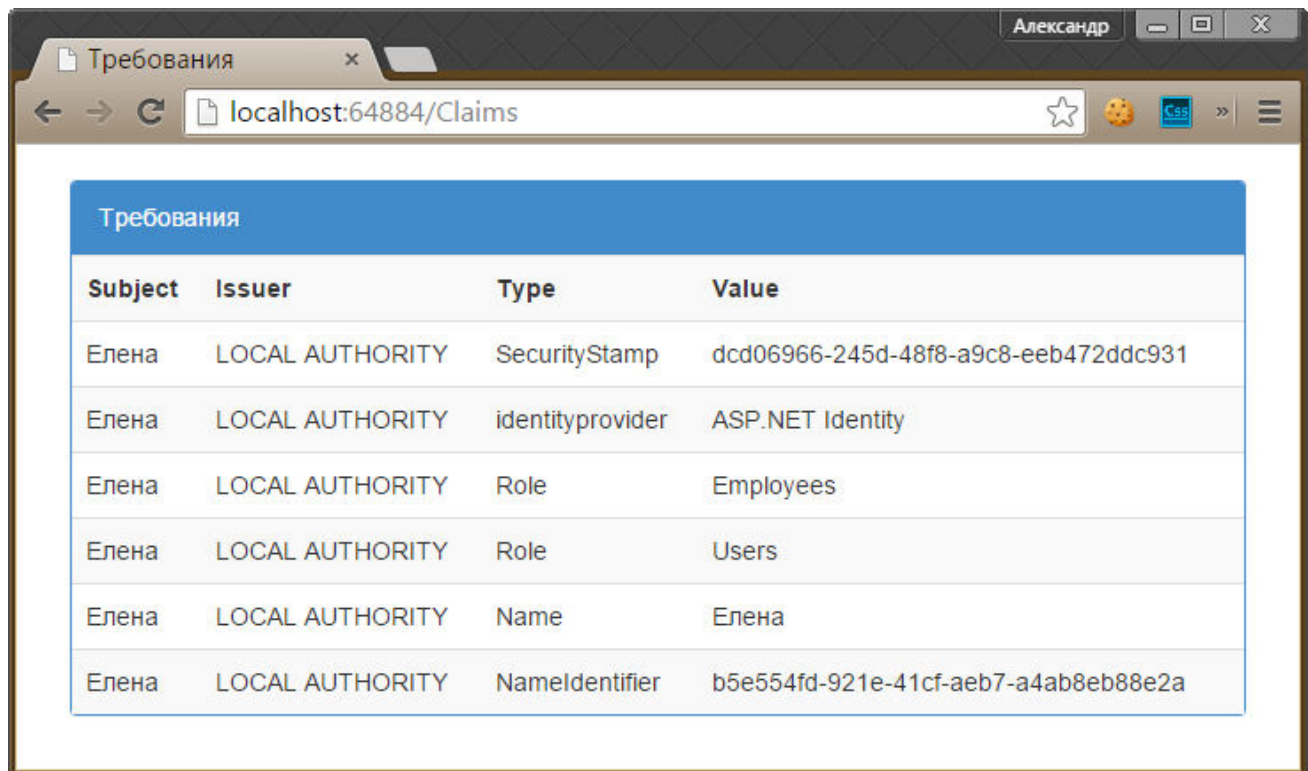
        public static MvcHtmlString ClaimType(this HtmlHelper html, string claimType)
        {
            FieldInfo[] fields = typeof(ClaimTypes).GetFields();

            foreach (FieldInfo field in fields)
            {
                if (field.GetValue(null).ToString() == claimType)
                {
                    return new MvcHtmlString(field.Name);
                }
            }

            return new MvcHtmlString(string.Format("{0}",
                claimType.Split('/', '.').Last()));
        }
    }
}
```

Теперь запустите наше приложение и перейдите по адресу /Claims/Index, предварительно авторизовавшись в приложении:

Пройди тесты C# тест (легкий) .NET тест (средний)



Требования			
Subject	Issuer	Type	Value
Елена	LOCAL AUTHORITY	SecurityStamp	dcd06966-245d-48f8-a9c8-eeb472ddc931
Елена	LOCAL AUTHORITY	identityprovider	ASP.NET Identity
Елена	LOCAL AUTHORITY	Role	Employees
Елена	LOCAL AUTHORITY	Role	Users
Елена	LOCAL AUTHORITY	Name	Елена
Елена	LOCAL AUTHORITY	NameIdentifier	b5e554fd-921e-41cf-aeb7-a4ab8eb88e2a

В этой таблице приведены базовые утверждения, которые автоматически добавляются при реализации системы аутентификации ASP.NET Identity, как в нашем приложении. Обратите внимание, что некоторые утверждения основаны на идентификаторе пользователя (Name содержит имя пользователя, а NameIdentifier его уникальный идентификатор в базе данных).

Также утверждения показывают членство в ролях — в таблице показано два утверждения типа Role, которые отражают тот факт, что пользователь «Елена» находится в ролях Users и Employees. Утверждение IdentityProvider описывает систему, с помощью которой был аутентифицирован пользователь — в нашем случае ASP.NET Identity.

Свойство Issuer описывает источник, откуда поступают данные о требованиях. В нашем случае это свойство содержит значение LOCAL AUTHORITY, которое говорит о том, что данные пользователя устанавливаются приложением.

Итак, теперь, когда вы увидели некоторые примеры утверждений, я могу описать что же они из себя представляют. Утверждение содержит любую информацию о пользователе, доступную для приложения. Например, идентификатор пользователя или членство в ролях. Эта информация автоматически формируется, когда мы добавляем какой-то функционал в приложение. Например, когда мы добавили систему ролей, платформа ASP.NET

Identity автоматически создала утверждения типа Role для каждого из пользователей.

Создание и использование утверждений

Как говорилось ранее, приложение может получать данные пользователя из разных источников, а не полагаться только на локальную базу данных Identity. Давайте добавим файл класса LocationClaimsProvide.cs в папку Infrastructure нашего проекта:

```
using System.Collections.Generic;
using System.Security.Claims;

namespace Users.Infrastructure
{
    public class LocationClaimsProvider
    {
        public static IEnumerable<Claim> GetClaims(ClaimsIdentity user)
        {
            List<Claim> claims = new List<Claim>();

            if (user.Name.ToLower() == "елена")
            {
                claims.Add(CreateClaim(ClaimTypes.PostalCode, "DC 20500"));
                claims.Add(CreateClaim(ClaimTypes.StateOrProvince, "DC"));
            }
            else
            {
                claims.Add(CreateClaim(ClaimTypes.PostalCode, "NY 10036"));
                claims.Add(CreateClaim(ClaimTypes.StateOrProvince, "NY"));
            }

            return claims;
        }

        private static Claim CreateClaim(string type, string value)
        {
            return new Claim(type, value, ClaimValueTypes.String, "RemoteClaims");
        }
    }
}
```

Пройди тесты

C# тест (легкий)

.NET тест (средний)

Метод GetClaims() принимает параметр типа ClaimsIdentity и использует свойство Name для создания утверждений, описывающих почтовый индекс и область

проживания. В реальном приложении эти данные могут быть получены, например, из базы данных HR, никак не связанной с текущим приложением.

Утверждения, связанные с идентификацией пользователя, можно добавить в процессе аутентификации. В примере ниже мы изменили метод действия Login контроллера Account, и добавили список утверждений с помощью метода LocationClaimsProvide.GetClaims():

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Login(LoginViewModel details, string returnUrl)
{
    AppUser user = await UserManager.FindAsync(details.Name, details.Password);

    if (user == null)
    {
        ModelState.AddModelError("", "Некорректное имя или пароль.");
    }
    else
    {
        ClaimsIdentity ident = await UserManager.CreateIdentityAsync(user,
            DefaultAuthenticationTypes.ApplicationCookie);

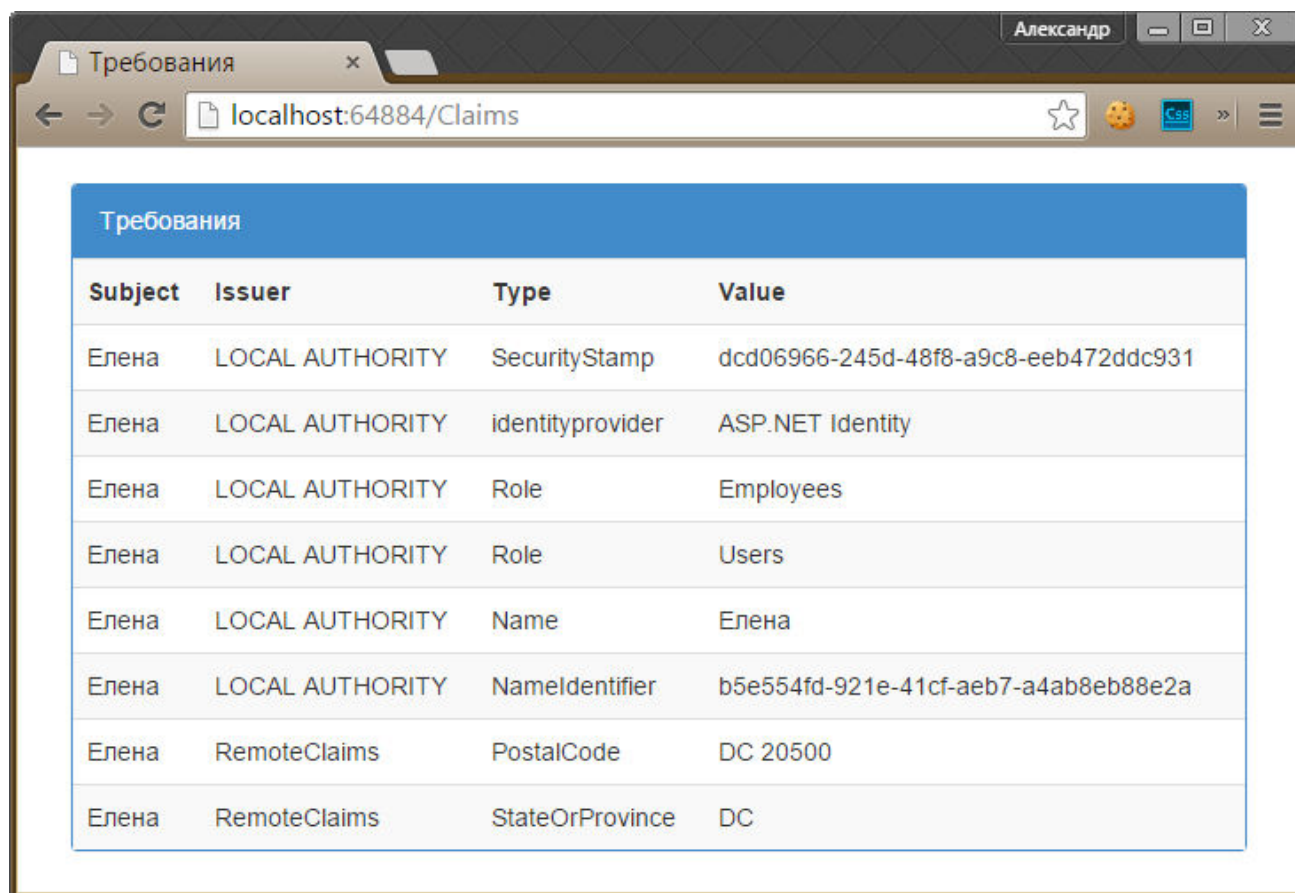
        // Мы добавили только эту строку
        ident.AddClaims(LocationClaimsProvider.GetClaims(ident));

        AuthManager.SignOut();
        AuthManager.SignIn(new AuthenticationProperties
        {
            IsPersistent = false
        }, ident);
        return Redirect(returnUrl);
    }

    return View(details);
}
```

Вы можете увидеть эффект от применения утверждений о местоположении пользователя, если запустите приложение, авторизуетесь и перейдете по адресу /Claims/Index:

Пройди тесты C# тест (легкий) .NET тест (средний)



Требования			
Subject	Issuer	Type	Value
Елена	LOCAL AUTHORITY	SecurityStamp	dcd06966-245d-48f8-a9c8-eeb472ddc931
Елена	LOCAL AUTHORITY	identityprovider	ASP.NET Identity
Елена	LOCAL AUTHORITY	Role	Employees
Елена	LOCAL AUTHORITY	Role	Users
Елена	LOCAL AUTHORITY	Name	Елена
Елена	LOCAL AUTHORITY	NameIdentifier	b5e554fd-921e-41cf-aeb7-a4ab8eb88e2a
Елена	RemoteClaims	PostalCode	DC 20500
Елена	RemoteClaims	StateOrProvince	DC

Получение утверждений из нескольких источников означает, что приложение не должно дублировать данные, сохраняемые в другом месте и позволяет легко интегрироваться со сторонними системами. Свойство `Claim.Issuer` всегда подскажет вам, в какой системе было создано утверждение, благодаря чему можно судить о точности получаемых данных. Например, данные о местоположении сотрудников, полученные из центральной базы данных HR, вероятно, будут более точными и надежными по сравнению с данными, полученными от внешнего поставщика списка рассылки.

Применение утверждений

Используя утверждения, можно более гибко настроить авторизацию в вашем приложении, нежели чем с ролями. Проблема с ролями заключается в том, что они статичны, и после того, как пользователю была назначена роль, он остается членом этой роли пока явно не будет удален из нее. Утверждения могут быть использованы для авторизации пользователей непосредственно на основе той информации, что известна о них. Самым простым способом реализации такой системы авторизации является генерация требований `Role` на основе пользовательских данных. Давайте добавим файл `ClaimsRoles.cs` в папку `Infrastructure` проекта со следующим содержимым:

```

using System.Collections.Generic;
using System.Security.Claims;

namespace Users.Infrastructure
{
    public class ClaimsRoles
    {
        public static IEnumerable<Claim> CreateRolesFromClaims(ClaimsIdentity user)
        {
            List<Claim> claims = new List<Claim>();

            if (user.HasClaim(x => x.Type == ClaimTypes.StateOrProvince
                && x.Issuer == "RemoteClaims" && x.Value == "DC")
                && user.HasClaim(x => x.Type == ClaimTypes.Role
                && x.Value == "Employees"))
            {
                claims.Add(new Claim(ClaimTypes.Role, "DCStaff"));
            }

            return claims;
        }
    }
}

```

Метод `CreateRolesFromClaims()` использует лямбда-выражение для определения того, имеет ли пользователь утверждение о местоположении `StateOrProvince` поступающее из источника `RemoteClaims` со значением «DC» (проживает ли пользователь в штате Вашингтон) и проверяет утверждение `Role` со значением «Employees» (является ли сотрудником компании). Если пользователь соответствует этим утверждениям, то ему задается роль-утверждение `DCStaff`. Следующий код демонстрирует как мы будем использовать метод `CreateRoleFromClaims()` в методе действия `Login` контроллера `Account`:

```

// ...
ident.AddClaims(LocationClaimsProvider.GetClaims(ident));
ident.AddClaims(ClaimsRoles.CreateRolesFromClaims(ident));
// ...

```

Теперь мы можем ограничить доступ к оп...
 ПроЙди тесты C# тест (легкий) .NET тест (средний)

```
public class ClaimsController : Controller
{
    // ...

    [Authorize(Roles = "DCStaff")]
    public string OtherAction()
    {
        return "Это защищенный метод действия";
    }
}
```

Пользователи смогут получить доступ к методу действия `OtherAction()` только если их утверждения соответствуют членству в роли `DCStaff`. Членство в этой роли генерируется динамически, поэтому изменение данных о должности пользователя или его местоположения будет автоматически отражаться на авторизации.

Добавление фильтра авторизации

Предыдущий пример является эффективной демонстрацией того, как разрешения могут быть использованы для точечной настройки авторизации, потому что мы создали специальную роль на основе разрешений, а затем ограничили доступ к определенным методам используя проверку на членство в этой роли. Более прямым и гибким подходом к реализации системы авторизации на основе разрешений, является создание [настраиваемого атрибута фильтра](#) авторизации.

Добавьте файл `ClaimsAccessAttribute.cs` в папку `Infrastructure` со следующим содержимым:

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

namespace Users.Infrastructure
{
    public class ClaimsAccessAttribute : AuthorizeAttribute
    {
        public string Issuer { get; set; }
        public string ClaimType { get; set; }
        public string Value { get; set; }

        protected override bool AuthorizeCore(HttpContextBase context)
        {
            return context.User.Identity.IsAuthenticated
                && context.User.Identity is ClaimsIdentity
                && ((ClaimsIdentity)context.User.Identity).HasClaim(x =>
                    x.Issuer == Issuer && x.Type == ClaimType && x.Value == Value
                );
        }
    }
}
```

Класс атрибута в этом примере является производным от **AuthorizeAttribute**, который позволяет легко создавать пользовательские политики авторизации в приложении MVC, путем переопределения *метода* **AuthorizeCore()**. Наша реализация этого метода предоставляет доступ, если пользователь был аутентифицирован, реализация интерфейса **IIdentity** является объектом **ClaimsIdentity** (т. е. имеет поддержку разрешений) и пользователь соответствует разрешению, которое настраивается путем инициализации свойств **Issue**, **ClaimType** и **Value** атрибута **ClaimsAccessAttribute**.

Следующий пример демонстрирует использование этого атрибута к методу действия **OtherAction()** контроллера **Claims**:

```
// ...
using Users.Infrastructure;

namespace Users.Controllers
{
    public class ClaimsController : Controller
    {
        // ...

        [ClaimsAccess(Issuer = "RemoteClaims", ClaimType = ClaimTypes.PostalCode,
            Value = "DC 20500")]
        public string OtherAction()
        {
            return "Это защищенный метод действия";
        }
    }
}
```

Фильтр авторизации в примере выше гарантирует, что доступ к методу действия `OtherAction` получают лишь те пользователи, почтовый индекс которых равен «DC 20500».

