

Projektowanie aplikacji ASP.NET

Wykład 10/15

ASP.NET MVC rozszerzenia

Wiktor Zychła 2019/2020

Spis treści

2	WebGrid	2
3	Filtry.....	4
4	Własny model binder	5
5	Własny atrybut walidacyjny	7
6	Własny HTML helper	8
7	Programowanie asynchroniczne	10
8	Własna fabryka kontrolerów.....	11
9	Testy jednostkowe.....	13

2 WebGrid

W implementacji stronicowania w MVC przydaje się pomocniczy model, opisujący stronicowany zbiór danych.

```
public class PagedEnumerable<T>
{
    public int CurrentPage { get; set; }
    public int PageSize { get; set; }
    public int TotalCount { get; set; }

    public IEnumerable<T> Items { get; set; }
}
```

Taki pomocniczy model może być następnie składową modelu głównego:

```
public class IndexModel
{
    public PagedEnumerable<User> Users { get; set; }
}
```

Helper **WebGrid** funkcjonalnie jest tylko namiastką GridView/ListView z **WebForms**, brakuje tu zwłaszcza rozbudowanych mechanizmów edycji ale obsługuje poprawnie stronicowanie i sortowanie.

Niestety, nie dzieje się to automatycznie. Kontroler musi wspierać odczytywanie porządku sortowania oraz numeru strony, jak również przygotowywać dane w modelu dla grida. Grid potrafi poprawnie wyrenderować stronę danych oraz pager. Inaczej niż w przypadku WebForms, tu kolumna sortowania i porządek sortowania są przez grid obsługiwane na dwóch rozłącznych parametrach:

```
public ActionResult Index(
    int page = 1,
    string sort = "GivenName", string sortdir = "ASC")
{
    var model = new IndexModel();
    var dataLayer = new DataLayer();

    model.Users = new PagedEnumerable<DataLibrary.User>()
    {
        Items = dataLayer.GetUsers(string.Format( "{0} {1}", sort, sortdir ), (page - 1) * 10, 10),
        TotalCount = dataLayer.TotalUsers()
    };

    return View(model);
}
```

Warstwa danych może tu być całkowicie przykładowa:

```
public class DataLayer
{
    public IEnumerable<User> GetUsers( string OrderBy, int StartRow, int RowCount )
    {
        IEnumerable<User> model = StaticModel.Users;

        model = model.AsQueryable().OrderBy(OrderBy).ToList();
    }
}
```

```

        return model.Skip(StartRow).Take(RowCount);
    }

    public int TotalUsers()
    {
        return StaticModel.Users.Count();
    }
}

public class StaticModel
{
    static StaticModel()
    {
        Users = new List<User>();
        Enumerable.Range(1, 100).ToList().ForEach(i =>
        {
            Users.Add(new User()
            {
                ID = i,
                GivenName = "GivenName " + i.ToString(),
                Surname = "Surname " + (100-i).ToString()
            });
        });
    }

    public static List<User> Users { get; private set; }
}

```

Renderowanie WebGrida polega na wskazaniu mu źródła danych oraz na użyciu metod renderujących tabelę oraz pager. Na uwagę zasługuje możliwość użycia funkcji typu lambda do tworzenia zawartości kolumn:

```

@{
    var grid = new WebGrid(canPage: true, rowsPerPage: 10);
    grid.Bind(source: Model.Users.Items,
              rowCount: Model.Users.TotalCount, autoSortAndPage: false);
}

<div>
    @grid.Table(
        columns: grid.Columns(
            grid.Column("GivenName", "Given name"),
            grid.Column("Surname", "Surname"),
            grid.Column(format:
                item =>
                    Html.ActionLink("Edycja", "Edit",
                        new { id = item.ID }), canSort: false)
        )
    )
</div>
<div>
    @grid.Pager(WebGridPagerModes.All, "<<", "<", ">", ">>")
</div>

```

3 Filtry

Mechanizm filtrów pozwala [rozszerzyć potok przetwarzania](#) o własną logikę w obszarach:

- Filtry akcji
- Filtry autentykacji
- Filtry wyniku
- Filtry wyjątków

Przykład [filtra akcji](#):

```
public class CustomActionFilter : ActionFilterAttribute, IActionFilter
{
    #region IActionFilter Members

    void IActionFilter.OnActionExecuted(ActionExecutedContext filterContext)
    {
        //throw new NotImplementedException();
    }

    void IActionFilter.OnActionExecuting(ActionExecutingContext filterContext)
    {
        string controllerName = filterContext.ActionDescriptor.ControllerDescriptor.ControllerName;
        string actionName      = filterContext.ActionDescriptor.ActionName;

        // np. logowanie
        // lub wręcz nadpisanie odpowiedzi
        // filterContext.Result = ...
    }

    #endregion
}
```

oraz jego użycie

```
[CustomActionFilter]
public ActionResult Index()
{
    return Content("foo");
}
```

4 Własny model binder

Własny binder modelu ma zastosowanie w przypadkach nietypowych konstrukcji formularzy, które miałyby się niestandardowo mapować na model. Przykładowy binder zadziała tak że wartość w polu tekstowym wpisana z przecinkami jako „a,b,c” zostanie zbindowana nie do zmiennej typu **string**, standardowo, tylko do zmiennej typu **string[]**, gdzie poszczególne elementy tablicy będą kolejnymi wartościami z napisu: a b i c.

```
public class CustomBinder : IModelBinder
{
    #region IModelBinder Members

    public object BindModel(
        ControllerContext controllerContext,
        ModelBindingContext bindingContext)
    {
        // nazwa zmiennej modelu (parametr metody akcji)
        string name = bindingContext.ModelName;

        var result = bindingContext.ValueProvider.GetValue(name);
        if (result != null)
        {
            string value = result.AttemptedValue;

            return value.Split(new[] { ',' });
        }
        else
            return Enumerable.Empty<string>();
    }

    #endregion
}
```

Bindowanie musi być wskazane, na przykład atrybutem

```
[HttpPost]
public ActionResult Index(
    [ModelBinder(typeof(CustomBinder))] IEnumerable<string> foo
)
{
    IndexModel model = new IndexModel();
    model.Foo = string.Join(",", foo);
    return View(model);
}
```

W przykładzie tym jest standardowy model i widok

```
public class IndexModel
{
    public string Foo { get; set; }
}
```

```
@model WebApplication2.Models.IndexModel
@{
```

```
    ViewBag.Title = "Index";  
}  
@using (var form = Html.BeginForm())  
{  
    @Html.TextBoxFor( m => m.Foo )  
    <button>Zapisz</button>  
}
```

5 Własny atrybut walidacyjny

Własny atrybut walidacyjny pozwala na rozszerzenie dostarczonego zestawu atrybutów walidacyjnych. Poniższy, przykładowy, pozwala na wskazanie oczekiwanej wartości pola tekstowego:

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = true)]
public class OurCustomValidationAttribute : ValidationAttribute
{
    private string _value { get; set; }
    public OurCustomValidationAttribute(string value)
    {
        this._value = value;
    }

    protected override ValidationResult IsValid(object value, ValidationContext
validationContext)
    {
        object property = validationContext.ObjectInstance.GetType().InvokeMemb
er(validationContext.DisplayName, System.Reflection.BindingFlags.GetProperty | Sy
stem.Reflection.BindingFlags.Public | System.Reflection.BindingFlags.Instance, n
ull, validationContext.ObjectInstance, null);

        if (property is string && ((string)property) == _value)
            return ValidationResult.Success;
        else
            return new ValidationResult("wrong value");
    }
}
```

Model udekorowany walidatorem

```
public class IndexModel
{
    [OurCustomValidation("foo")]
    public string Foo { get; set; }
}
```

6 Własny HTML helper

Ten mechanizm rozszerzający pozwala na tworzenie własnych helperów, w tym rozbudowanych jak WebGrid

```
public static class CustomHtmlHelper
{
    /// <summary>
    /// Usage Html.CustomTextBox( "foo" )
    /// </summary>
    /// <param name="htmlHelper"></param>
    /// <param name="name"></param>
    /// <returns></returns>
    public static HtmlString CustomTextBox(this HtmlHelper htmlHelper, string name, object value)
    {
        TagBuilder tb = new TagBuilder("input");

        tb.MergeAttribute("type", "text");
        tb.MergeAttribute("name", name);
        if (value != null)
        {
            tb.MergeAttribute("value", value.ToString());
        }

        return new HtmlString(tb.ToString());
    }

    public static HtmlString CustomTextBoxFor<TModel, TProperty>(
        this HtmlHelper<TModel> htmlHelper,
        Expression<Func<TModel, TProperty>> Property)
    {
        TagBuilder tb = new TagBuilder("input");

        tb.MergeAttribute("type", "text");

        string name = ExpressionHelper.GetExpressionText(Property);

        tb.MergeAttribute("name", name);

        object value = ModelMetadata.FromLambdaExpression(Property, htmlHelper.ViewData).Model;

        if (value != null)
        {
            tb.MergeAttribute("value", value.ToString());
        }

        return new HtmlString(tb.ToString());
    }
}
```

Użycie:

```
@model WebApplication2.Models.IndexModel
@using WebApplication2.Models
@{
```



```
    ViewBag.Title = "Index";  
}  
  
@using (var form = Html.BeginForm())  
{  
    @Html.CustomTextBoxFor( m => m.Foo )  
    <button>Zapisz</button>  
}
```

7 Programowanie asynchroniczne

Serwer aplikacji przydziela przychodzącym żądaniom wątki według [ściśle określonych reguł](#). Jeśli żądania blokują wykonanie na oczekiwaniu na zewnętrzne I/O (pliki, baza danych), to możemy myśleć o nieoptymalnym wykorzystaniu zasobów serwera.

Dlatego MVC wprowadza możliwość tworzenia kodu asynchronicznego na serwerze, co w praktyce oznacza, że z punktu widzenia klienta (protokół http) żądanie nadal jest synchroniczne, ale na serwerze wątek roboczy można było uwolnić dla innych obliczeń:

```
[HttpGet]
public async Task<ActionResult> Index()
{
    IndexModel model = new IndexModel();
    await Task.Delay(1000);
    return View(model);
}
```

8 Własna fabryka kontrolerów

MVC pozwala na przejęcie kontroli nad fabryką kontrolerów. Dzięki temu możliwe jest np. zaimplementowanie wstrzykiwania zależności do klas kontrolerów przez kontener IoC:

```
public class CustomControllerFactory :
    DefaultControllerFactory,
    IControllerFactory
{
    private IUnityContainer _container { get; set; }

    public CustomControllerFactory( IUnityContainer container )
    {
        this._container = container;
    }

    public override IController CreateController(RequestContext requestContext,
        string controllerName)
    {
        var controllerType = this.GetControllerType(requestContext, controllerName);

        if (controllerType != null)
        {
            IController controller = (IController)_container.Resolve(controllerType);

            if (controller != null)
                return controller;
        }

        throw new HttpException(404, string.Format("Nie odnaleziono zasobu dla żądania '{0}'", new object[]
        {
            requestContext.HttpContext.Request.Path
        }));
    }
}
```

Fabrykę należy wskazać i skonfigurować w potoku:

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        RouteConfig.RegisterRoutes(RouteTable.Routes);

        IUnityContainer container = new UnityContainer();
        container.RegisterType<IService, ServiceImpl>();

        ControllerBuilder.Current.SetControllerFactory(new CustomControllerFactory(container));
    }
}

public interface IService
{
}
```

```
    string DoWork();
}

public class ServiceImpl : IService
{
    public string DoWork()
    {
        return string.Format("serviceimpl: {0}", DateTime.Now);
    }
}
```

dzięki czemu możliwe jest wstrzykiwanie zależności do kontrolera, np. przez konstruktor (czyli tak jak wspiera to wykorzystany kontener: tu Unity):

```
public class HomeController : Controller
{
    private IService _service;
    public HomeController( IService service )
    {
        this._service = service;
    }
}
```

Taka architektura wspiera właściwą modularność aplikacji i jest chętnie wykorzystywana w praktyce.

9 Testy jednostkowe

Z uwagi na możliwość wykonywania metod kontrolerów poza kontekstem aktualnych żądań HTTP, framework MVC jest lepiej przystosowany do [pisania testów jednostkowych](#).