阿不

在

讯微博和

新浪微博

@hjf1223

，最新的

技术观点

在那



公告

## NVelocity for ASP.NET MVC

在我的这篇博文中，有这么一段话：“我一直在想，有没有办法可以单独限制View中的代码的访问权限，类似于trust level，只是这个trust level是用来限制模板中的代码。”。有读者johngeng问，为什么要用trust level来锁住view，他不是很理解。我的本意是，希望在view中，开发人员只能写某一些特定功能的代码，调用某一些特定开放的API，对于大部分安全级比较高的代码，比如读写文件等API或类库，不允许在view当中使用。这对于我们将模板开放出来，在线提供给我们的用户去修改的需求下是非常重要的。而目前，不管WebForm还是Razor，都是非常自由的模板，在View能做的事情等同于Controller或其它地方所写的代码，这样View就不允许开放出来由用户在线修改。

在相同的博文里面，还是那位读者johngeng提到它更喜欢$而不是@，由于我之前并不了解NVelocity，所以我误解为它是在说客户端开发包jquery。现在看来，他说的应该就是NVelocity，也许他觉得此人不可教，他并没有直接回复我的疑问，这也只能怪自己知识面太窄了。 

若不是最近在为项目添加多模板引擎的支持，或许我永远也无法得到以上两个问题的答案，而这两个答案都与NVelocity有关。虽然我平常肯定也见过NVelocity这个词，但到要选择除WebForm以外的模板引擎，我还是完完全全没有记起他，还是同事@浪子提醒我NVelocity这个模板引擎值得一试。看了官方的语法介绍后，我不得不说它是一种非常简洁且实用的模板，同时又不失它的灵活性和安全性。我所指的灵活性是它不像StringTemplate那样，限制的那么死，连个对象的函数都不允许调用。安全性方面又可以满足我希望模板上限制开发人员只能在模板上调用指定的API。到目前为止，NVelocity仍然让我非常满意。

在ASP.NET MVC切换视图引擎非常简单，在ASP.NET MVC1.0出来以后，MvcContrib就曾经提供了多种视图引擎的切换选择，但是在最近的版本中，我却始终没有找到相关的代码，应该是这些代码已经被移出去了，但它的介绍文档中还没有删掉相关的主题。还好在@重典童鞋的博客上找到了他从MvcContrib中提取出来的实现。但是这个实现相对于MVC3来说，已经相对过时了，有些接口已经改变或被移除了，比如IViewLocator这个接口就已经不存在了。还有就是，它去掉了原先支持的调用HtmlHelper扩展方法的功能，而我最重要的就是要支持扩展函数，因为我自定义了一些必须的扩展方法。下面我们就来看看NVelocity for ASP.NET MVC几个类的详细情况：

### NVelocityViewEngine

在之前的实现中，直接实现了IViewEngine这个接口，查找View的路径是通过实现IViewLocator来定位。在MVC2当中，修改了这部分的实现，MVC内部提供

了VirtualPathProviderViewEngine这个模板方法类，在子类当中，我们中需要设置一下我们要查找的路径格式，其它的事件就可以交给模板方法类来完成，这样一方面可以简化我们的实现，另一方面还可以和默认的路径查找方式统一。

同时，由于我使用Nvelocity内置的相对文件路径的方式来查找模板，而使用VirtualPath的风格，因此在找到VirtualPath后，我们需要转换成实际的物理路径，直接通过物理路径来加载模板内容，而内置的FileResourceLoader并不支持从物理路径加载模板，所以我们还要额外实现一下FileResourceLoader，让支持从物理路径的加载方法。这两个类的代码如下:

```
public class FileResourceLoaderEx : FileResourceLoader
{
    public FileResourceLoaderEx() : base() { }
    private Stream FindTemplate(string filePath)
    {
        try
        {
            FileInfo file = new FileInfo(filePath);
            return new BufferedStream(file.OpenRead());
        }
        catch (Exception exception)
        {

base.runtimeServices.Debug(string.Format("FileResourceLoader : {0}",
exception.Message));
            return null;
        }
    }


    public override long
GetLastModified(global::NVelocity.Runtime.Resource.Resource resource)
    {
        if (File.Exists(resource.Name))
        {
            FileInfo file = new FileInfo(resource.Name);
            return file.LastWriteTime.Ticks;
        }
        return base.GetLastModified(resource);
    }
    public override Stream GetResourceStream(string templateName)
    {
        if (File.Exists(templateName))
        {
            return FindTemplate(templateName);
        }
        return base.GetResourceStream(templateName);
    }
    public override bool
IsSourceModified(global::NVelocity.Runtime.Resource.Resource
resource)
    {
        if (File.Exists(resource.Name))
```

```
        {
            FileInfo file = new FileInfo(resource.Name);
            return (!file.Exists || (file.LastWriteTime.Ticks != 
resource.LastModified));
        }
        return base.IsSourceModified(resource);
    }
}
public class NVelocityViewEngine : VirtualPathProviderViewEngine, 
IViewEngine
{
    public static NVelocityViewEngine Default = null;

    private static readonly IDictionary DEFAULT_PROPERTIES = new 
Hashtable();
    private readonly VelocityEngine _engine;

    static NVelocityViewEngine()
    {
        string targetViewFolder = 
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "views");
        //DEFAULT_PROPERTIES.Add(RuntimeConstants.RESOURCE_LOADER, 
"file");

DEFAULT_PROPERTIES.Add(RuntimeConstants.FILE_RESOURCE_LOADER_PATH, 
targetViewFolder);
        DEFAULT_PROPERTIES.Add("file.resource.loader.class", 
"NVelocityEngine.FileResourceLoaderEx\\,NVelocityEngine");


        Default = new NVelocityViewEngine();
    }

    public NVelocityViewEngine()
        : this(DEFAULT_PROPERTIES)
    {
    }

    public NVelocityViewEngine(IDictionary properties)
    {
        base.MasterLocationFormats = new string[] { 
"~/Views/{1}/{0}.vm", "~/Views/Shared/{0}.vm" };
        base.AreaMasterLocationFormats = new string[] { 
"~/Areas/{2}/Views/{1}/{0}.vm", "~/Areas/{2}/Views/Shared/{0}.vm" };
        base.ViewLocationFormats = new string[] { 
"~/Views/{1}/{0}.vm", "~/Views/Shared/{0}.vm" };
        base.AreaViewLocationFormats = new string[] { 
"~/Areas/{2}/Views/{1}/{0}.vm", "~/Areas/{2}/Views/Shared/{0}.vm" };
        base.PartialViewLocationFormats = base.ViewLocationFormats;
        base.AreaPartialViewLocationFormats = 
base.AreaViewLocationFormats;
```

```
        base.FileExtensions = new string[] { "vm" };


        if (properties == null) properties = DEFAULT_PROPERTIES;

        ExtendedProperties props = new ExtendedProperties();
        foreach (string key in properties.Keys)
        {
            props.AddProperty(key, properties[key]);
        }

        _engine = new VelocityEngine();
        _engine.Init(props);
    }

    protected override IView CreateView(ControllerContext
controllerContext, string viewPath, string masterPath)
    {
        Template viewTemplate = GetTemplate(viewPath);
        Template masterTemplate = GetTemplate(masterPath);
        NVelocityView view = new NVelocityView(controllerContext,
viewTemplate, masterTemplate);
        return view;
    }
    protected override IView CreatePartialView(ControllerContext
controllerContext, string partialPath)
    {
        Template viewTemplate = GetTemplate(partialPath);
        NVelocityView view = new NVelocityView(controllerContext,
viewTemplate, null);
        return view;
    }
    public Template GetTemplate(string viewPath)
    {
        if (string.IsNullOrEmpty(viewPath))
        {
            return null;
        }
        return
_engine.GetTemplate(System.Web.Hosting.HostingEnvironment.MapPath(vie
wPath));
    }

}
```

## NVelocityView

主要实现IView接口，实现Render方法来将模板和当前的上下文结合之后输出出来。这个类还实现了，IViewDataContainer好像不是特别必要。NVelocity的Render也很简单，只是把所需要的对像塞到NVelocity执行的上下文当中，然后调用一下Merge方法就OK了。这里要特别说明的是，在NVelocity模板上面，我们可以调用上下文对象的中的任何属性和方法，但是

没有办法调用到对象上的扩展方法，这时候，我们就需要借助NVelocity所提供的IDuck这个接口来提供扩展方法的支持，如下代码的：new HtmlExtensionDuck(context, this)；。完全代码如下：

```
public class NVelocityView : IViewDataContainer, IView
{
    private ControllerContext _controllerContext;
    private readonly Template _masterTemplate;
    private readonly Template _viewTemplate;

    public NVelocityView(ControllerContext controllerContext, string
viewPath, string masterPath)
        : this(controllerContext,
NVelocityViewEngine.Default.GetTemplate(viewPath),
NVelocityViewEngine.Default.GetTemplate(masterPath))
    {

    }
    public NVelocityView(ControllerContext controllerContext,
Template viewTemplate, Template masterTemplate)
    {
        _controllerContext = controllerContext;
        _viewTemplate = viewTemplate;
        _masterTemplate = masterTemplate;
    }

    public Template ViewTemplate
    {
        get { return _viewTemplate; }
    }

    public Template MasterTemplate
    {
        get { return _masterTemplate; }
    }

    private VelocityContext CreateContext(ViewContext context)
    {
        Hashtable entries = new
Hashtable(StringComparer.InvariantCultureIgnoreCase);
        if (context.ViewData != null)
        {
            foreach (var pair in context.ViewData)
            {
                entries[pair.Key] = pair.Value;
            }
        }
        entries["viewdata"] = context.ViewData;
        entries["tempdata"] = context.TempData;
        entries["routedata"] = context.RouteData;
        entries["controller"] = context.Controller;
        entries["httpcontext"] = context.HttpContext;
```

```
        entries["viewbag"] = context.ViewData;
        CreateAndAddHelpers(entries, context);

        return new VelocityContext(entries);
    }

    private void CreateAndAddHelpers(Hashtable entries, ViewContext
context)
    {
        entries["html"] = entries["htmlhelper"] = new
HtmlExtensionDuck(context, this);
        entries["url"] = entries["urlhelper"] = new
UrlHelper(context.RequestContext);
        entries["ajax"] = entries["ajaxhelper"] = new
AjaxHelper(context, this);
    }

    public void Render(ViewContext viewContext, TextWriter writer)
    {
        this.ViewData = viewContext.ViewData;

        bool hasLayout = _masterTemplate != null;

        VelocityContext context = CreateContext(viewContext);

        if (hasLayout)
        {
            StringWriter sw = new StringWriter();
            _viewTemplate.Merge(context, sw);

            context.Put("childContent",
sw.GetStringBuilder().ToString());

            _masterTemplate.Merge(context, writer);
        }
        else
        {
            _viewTemplate.Merge(context, writer);
        }
    }

    private ViewDataDictionary _viewData;
    public ViewDataDictionary ViewData
    {
        get
        {
            if (_viewData == null)
            {
                return _controllerContext.Controller.ViewData;
            }
            return _viewData;
```

```
        }
        set
        {
            _viewData = value;
        }
    }
}
```

## ExtensionDuck

ExtensionDuck就是对IDuck接口的实现，它是我们需要提供扩展方法支持的Duck对象的基类。所有需要接供扩展方法的对象，通过继承该方法可以简化大部分的工作:

```
public class ExtensionDuck : IDuck
{
    private readonly object _instance;
    private readonly Type _instanceType;
    private readonly Type[] _extensionTypes;
    private Introspector _introspector;

    public ExtensionDuck(object instance)
        : this(instance, Type.EmptyTypes)
    {
    }

    public ExtensionDuck(object instance, params Type[]
extentionTypes)
    {
        if(instance == null) throw new
ArgumentNullException("instance");

        _instance = instance;
        _instanceType = _instance.GetType();
        _extensionTypes = extentionTypes;
    }

    public Introspector Introspector
    {
        get
        {
            if(_introspector == null)
            {
                _introspector = RuntimeSingleton.Introspector;
            }
            return _introspector;
        }
        set { _introspector = value; }
    }

    public object GetInvoke(string propName)
    {
        throw new NotSupportedException();
```

```
    }

    public void SetInvoke(string propName, object value)
    {
        throw new NotSupportedException();
    }

    public object Invoke(string method, params object[] args)
    {
        if(string.IsNullOrEmpty(method)) return null;

        MethodInfo methodInfo = Introspector.GetMethod(_instanceType,
method, args);
        if(methodInfo != null)
        {
            return methodInfo.Invoke(_instance, args);
        }

        object[] extensionArgs = new object[args.Length + 1];
        extensionArgs[0] = _instance;
        Array.Copy(args, 0, extensionArgs, 1, args.Length);

        foreach(Type extensionType in _extensionTypes)
        {
            methodInfo = Introspector.GetMethod(extensionType,
method, extensionArgs);
            if(methodInfo != null)
            {
                return methodInfo.Invoke(null, extensionArgs);
            }
        }

        return null;
    }
}
```

接下，我们就可以来实现一个HtmlExtensionDuck，指定一下，View中可以调用到HtmlHelper的哪些扩展方法，需要被开放的扩展方法可以在HTML_EXTENSION_TYPES中提供扩展方法所在的静态类名:

```
public class HtmlExtensionDuck : ExtensionDuck
{
    public static readonly Type[] HTML_EXTENSION_TYPES =
        new Type[]
            {
                typeof(DisplayExtensions),
                typeof(DisplayTextExtensions),
                typeof(EditorExtensions),
                typeof(FormExtensions),
                typeof(InputExtensions),
                typeof(LabelExtensions),
                typeof(LinkExtensions),
                typeof(MvcForm),
```

```
                typeof(PartialExtensions),
                typeof(RenderPartialExtensions),
                typeof(SelectExtensions),
                typeof(TextAreaExtensions),
                typeof(ValidationExtensions)
            };

    public HtmlExtensionDuck(ViewContext viewContext,
IViewDataContainer container)
        : this(new HtmlHelper(viewContext, container))
    {
    }

    public HtmlExtensionDuck(HtmlHelper htmlHelper)
        : this(htmlHelper, HTML_EXTENSION_TYPES)
    {
    }

    public HtmlExtensionDuck(HtmlHelper htmlHelper, params Type[]
extentionTypes)
        : base(htmlHelper, extentionTypes)
    {
    }
}
```

完整的NVelocity for ASP.NET MVC的实现就是以上几个类就可以完成。然后，我们就可以直接注册到系统中来。我们不需要重写任何Conroller，我们直接把ViewEngine注册到MVC中来就可以被用到，也可以支持一个程序支持多种视图引擎共存的和谐场面。简单的注册代码放在Global.asax文件中：

```
ViewEngines.Engines.Add(NVelocityViewEngine.Default);
```

详细的使用示例，下载附件查看详细。

最后总结一下，NVelocity确实是一种简单实用的模板引擎，特别是它的语法非常简洁，而且API的可扩展性还是挺强的。Razor的语法的基本风格应该是有借鉴了它的语法风格。我现在虽然也很喜欢NVelocity，但如果不是特殊情况的特殊需要，在普通的ASP.NET MVC程序中，我还是会侧向于使用Razor。它更自由一点，由于是直接的C#编译支持，所以我们可以做任何的事情，这对于很多开发人员来说很重要。另一个不可忽视的就是它的IDE支持，特别是代码提示确实是让人相当的舒服。

posted on 2011-01-13 23:37 阿不 阅读(3383) 评论(8) 编辑 收藏

Powered by:

博客园