蒋大师的MVC框架解析确实是越学越有趣，即使是跟着学写些示例代码也是收获良多，尤其是关于类型、反射和委托等方面，平时在应用开发中确实很少会有机会写这样的代码。今天学习的ASP.NET MVC中的Model的验证，刚开时会以为这一章会比较简单，因为之前已经学习过了Model元数据的解析、Model绑定，Model的验证可能就只是DataAnnotation相关类的介绍。但实际学习的过程中，尤其是自定义用于修饰Action的验证特性让我到现在仍然感觉是比较萌萌哒，毕竟这一块对于框架的扩展基本上涉及到了验证相关的所有类型。除此之外，昨晚也是我第一次从<https://aspnetwebstack.codeplex.com/>上用git下载到了到MVC的源码，本以为会比较艰难，但实际却非常的方便，怒赞下。之后在VS2012打开Nuget会自动下载依赖组件，就可以编译通过了。记得今年一直听到各种关于微软的开源计划，自己接触的知识领域还是比较低端，也不太清楚到底有些什么源代码可以看，当时首先想到的就是到目前为止仍然掌握很弱的WCF，然后查查居然也有源码了，顿时觉得压力山大，因为以后再做不好.NET就不能和妈妈说我看不到源码了。原来一直关于.NET的彷徨，至少在这一刻得到很好的坚定，虽然由于市场的原因.NET在国内的发展比较飘忽，但从自身技术发展的角度，有了源码，只要努力，我就可以生活大师的身边，知道什么是对的了，这个一直困惑我多年。不知道大家有没有这样的感受，即使对自己的代码风格、设计理念非常认同，但重来没有说应该这样做的底气。见笑了，言归正传，回到Model的验证，内容比较对，篇幅很能比较长，望见谅。

首先介绍最为核心的ModelValidator抽象类，该类的主要的成员方法包括：GetClientValidationRules()，返回值为客户端验证规则，最终由HtmlHelper的模板方法渲染为html语句，由于未来项目中并不打算使用Razor引擎，这部分会略过一些内容，但之后有一部分关于JQuery-validate组件的扩展还是很有价值的；Validate(object container)，返回值为ModelValidationResult集合，需要注意的是该方法的参数container说明验证过程是包含类型本身和其所辖的属性成员的。接下来用图表简要介绍几个MVC中的Model验证解决方案：

|  |  |
| --- | --- |
| 验证解决方案 | 简介 |
| DataAnnotationsModelValidator | 最主要的验证方案，包括常见的验证特性：RequiredAttribute,RangeAttribute等 |
| ClientModelValidator | 客户端验证。 |
| DataErrorInfoModelValidator | 实现IDataErrorInfo接口，包括:  DataErrorInfoClassModelValidator,  DataErrorInfoPropertyModelValidator |
| ValidatableObjectAdapter | 实现IValidatableObject接口，也称为”自我验证“，比较少使用。 |

这儿仍然使用Provider模式来提供相应的组件，ModelValidatorProvider类具有GetValidators(ModelMetadata metadata, ControllerContext context)

方法，前一个参数描述被验证类型或熟悉的元数据对象，另一个为当前的ControllerContext。同时，具体的Provider与之前介绍的验证解决方案的中类型相对应，在此就不一一介绍，需要注意的是在验证一个类型时，是先验证它的属性，然后才验证它自身，因此会出现验证的短路现象，即属性出错，就不会继续验证和反馈容器类型的错误了。与之前一样，这儿也会使用注册表模式来管理Provider，使用上ModelValidatorProviders来进行注册，框架默认会加载DataAnnotationXXX，ClientXXX，DataErrorInfoPropertyXXX，也可以把自定义的Provider加入其中。在框架中真正负责验证工作的是一个CompositeModelValidator私有类，查看源码确定是ModelValidator中的一个内部类，但为什么这样使用还有一些困惑，为什么这样需要完全隐藏掉该类？

接下来，介绍Model绑定与验证的关系，在前文”Model的绑定”的介绍中提到Controller对象的ViewData包含ModelState集合，用于表示Model的状态，其中既包括ValueProvider提供的值，也包括Errors验证结果。验证结果的呈现通过ValidationMessage，ValidationMessageFor扩展方法对单个属性进行验证，输出html形式为(class=”field-validation-error” data-valmsg-for=“xxx”,data-valmsg-replace=“true”)，ValidationSummary呈现容器整体的验证结果，可以设置excludePropertyErrors参数。同时注意可以通过ModelState的AddModelError方法添加错误信息，EditorForModel扩展方法在使用时会默认的显示验证错误时的信息。

Model绑定中的验证解释起来比较拗口，但简单说来就是DefaultModelBinder在递归的绑定复杂对象的过程中对绑定后的对象实施验证，如下图所示。



为了更加了解Model绑定和验证的关联，自己跟着蒋大师的源码基本原样敲了一遍，主要里面有一些用法自己还是不够熟悉，多练练了，大家可以无视。

|  |
| --- |
| public class CompositeModelValidator : ModelValidator  {  public CompositeModelValidator(ModelMetadata metadata, ControllerContext controllerContext)  : base(metadata, controllerContext)  {  }  public override IEnumerable<ModelValidationResult> Validate(object container)  {  bool isPropertiesValid = true;  //验证属性  foreach (var propertyMetadata in Metadata.Properties)  {  foreach (var validator in propertyMetadata.GetValidators(this.ControllerContext))  {  var results = validator.Validate(propertyMetadata.Model);  if (results.Any())  {  isPropertiesValid = false;  }  foreach (var result in results)  {  yield return new ModelValidationResult  {  MemberName = DefaultModelBinder.CreateSubPropertyName(propertyMetadata.PropertyName, result.MemberName),  Message = result.Message  };  }  }  }  //验证容器类  if (isPropertiesValid)  {  foreach (var validator in Metadata.GetValidators(this.ControllerContext))  {  var results = validator.Validate(Metadata.Model);  foreach (var result in results)  {  yield return result;  }  }  }  }  } |

|  |
| --- |
| public class DefaultModelBinder : IModelBinder  {  internal static string CreateSubPropertyName(string prefix, string propertyName)  {  prefix = prefix ?? "";  propertyName = propertyName ?? "";  return (prefix + "." + propertyName).Trim('.');  }  protected virtual object GetComplexModel(ControllerContext controllerContext, Type modelType, IValueProvider valueProvider, string prefix) {  object model = CreateModel(modelType);  foreach(PropertyDescriptor property in TypeDescriptor.GetProperties(modelType)){  if (property.IsReadOnly) {  continue;  }  string key = string.IsNullOrEmpty(prefix) ? property.Name : prefix + "." + property.Name;  property.SetValue(model, GetModel(controllerContext, property.PropertyType, valueProvider, key));  }  //Model验证  var metadata = ModelMetadataProviders.Current.GetMetadataForType(()=>model, modelType);  var validator = new CompositeModelValidator(metadata, controllerContext);  foreach(var result in validator.Validate(model)){  string key = CreateSubPropertyName(prefix, result.MemberName);  controllerContext.Controller.ViewData.ModelState.AddModelError(key, result.Message);  }  return model;  }  } |

注：本文主要供自己学习，不妥之处望见谅。

参考资料：

[1]蒋金楠. ASP.NET MVC4框架揭秘[M]. 上海:电子工业出版社, 2012. 254-282