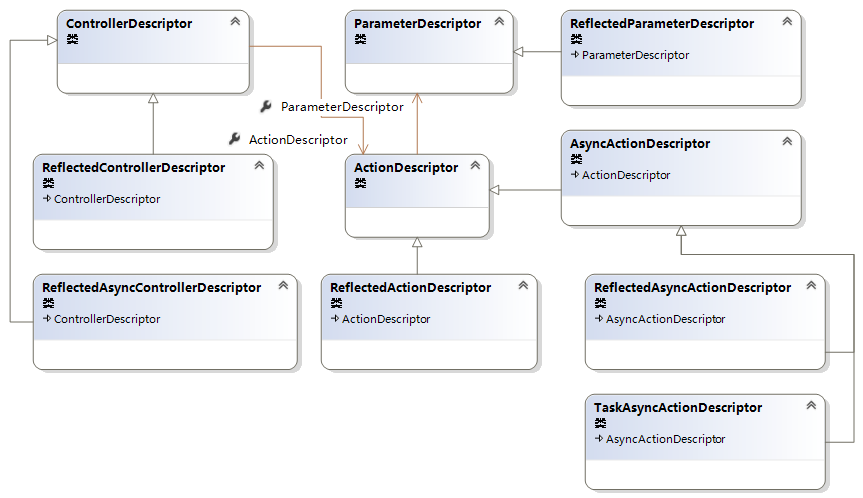
数据绑定对于长期进行.NET相关开发的技术人员来说，非常的熟悉。无论是最开始的WinForm， WebForm， 还是现在的WPF，ASP.NET MVC，只要是与前台页面数据源相关的内容，都离不开这个概念，该概念的实现极大的简化了相关的开发工作。即使不使用Razor视图引擎，直接使用静态的HTML5页面，该模块仍然不可获取。现在简单的介绍一下ASP.NET MVC中相关的Model绑定。

在ASP.NET MVC框架中，Model绑定本质上就是为目标Action方法生成参数列表的过程，这些参数列表的来源可能是请求的URL，可能是HTTP的请求头或请求体中，通过参数的元数据信息可以得到相关内容。主要有ControllerDescriptor，ActionDescriptor，ParameterDescriptor三个抽象类，他们均实现了System.Reflection.ICustomAttributeProvider接口，其实主要就是对类元数据信息的包装类，方便元数据信息的使用，其具体实现类与关系如下图所示。



通过上图可以看到，所有Reflected作为前缀的类都是实现类，同时可以看到Controller、Action描述类的异步版本，比较特殊的是TaskAsyncActionDescriptor，它可以在普通的（非异步）的Controller中使用，在自己试图搭建框架时可以模仿该方式，抽象类，同步/异步版本，接下来通过一个表格简要介绍和比较以上三个描述类。

|  |  |
| --- | --- |
| 类型 | 简介 |
| ControllerDescriptor | 比较特殊是GetFilterAttributes方法，用于获取该控制器上的所有过滤器特性，ActionMethodSelectorAttribute特性包含GET, POST, PUT, DELETE, Head, Options, Patch等七个Http方法。 |
| ActionDescriptor | GetFilters方法返回FilterInfo类型，包含ActionFilter，AuthorizationFilter，ExceptionFilter，ResultFilter等四种类型的筛选器，与J2EE类似。 |
| ParameterDescriptor | 其中属性ParameterBindingInfo最为关键，实际包含ModelBinder对象，该对象是整个绑定模块的核心，同时Include，Exclude集合用于显示设置参与/不参与绑定的属性，Prefix属性主要用于复杂的类型的绑定。 |

由于数据绑定的来源各不相同，框架通过接口IValueProvider类来统一提供数据。首先介绍NameValueCollectionValueProvider，属于key/Value的形式，需要加强理解的是，在之前介绍的描述一个复杂数据类型的ModelMetadata具有树型层次化结构，而NameValueColletion对象却是一个“扁平”的结构，两者的匹配通过前缀来完成。“扁平化”这个概念在现在数据呈现中出现的非常多，无论是这儿MVC中的ViewModel， 还是WPF中MVVM框架下的ViewModel。该类型的Provider主要包括FormValueProvider和QueryStringProvider，顾名思义，关联HttpRequest中的QueryString属性。

接下来介绍DictionaryValueProvider，与以前数据提供器的主要区别是其数据值不仅支持字符串，还支持任意对象，可以使用泛型约束。主要包括RouteDataValueProvider，HttpFileCollectionProvider，ChildActionValueProvider。这里想重点提及的是ChildActionProvider，它主要用于子Action中，由于子Action不同独立用于响应客户端的请求，只是用于生成部分的HTML。框架中使用ValueProviderFactory工厂类用于创建一系列的值提供器，同时使用ValueProviderFactories这个静态类通过注册的方式管理以上工厂。

之前有提过这部分最重要的类型就是ModelBinder，有了之前的基础，现在是时候介绍它了。借用蒋大师的原话，“Model的绑定体现在从当前请求提取相应的数据并生成相应的对象作为调用目标Action方法的参数列表”。不知道大家还记不记得，之前在ActionDescriptor中提到的ParameterBindingInfo类型的对象，其中就有ModelBinder，为了便于理解仍然使用表格进行介绍。

|  |  |
| --- | --- |
| 类型 | 简介 |
| IModelBinder | 包含BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)方法。 |
| ModelBinderAttribute | 用于自己定义模型的绑定器，之后会介绍默认绑定器，在未指定绑定时使用，也是最常见的。 |
| ModelBinders | 静态类用于注册模型绑定器，可以在Application\_Start方法中为指定类型设置绑定器 |
| ModelBinderProvider | GetBinder(Type modelType)方法用于根据指定的数据类型获取相应的ModelBinder对象。 |
| ModelState | \*Model绑定除了设置参数列表以外，还将数据通过ModelState的形式存储于Controller的ViewData中。sw |
| ModelBindingContext | 简述Model绑定的过程，首先Action调用ActionInvoker执行Action，此时获得ActionDescriptor对象，然后遍历ActionDescriptor参数列表，根据ParameterDescriptor等对象创建ModelBindingContext。然后获得指定的ModelBinder，调用GetModel得到由ValueProvider提供的相应参数值，最后以ModelState的形式保存。 |

最后，介绍默认的Model绑定DefaultModelBinder类，这块内容比较多，但由于是框架默认提供的，了解即可，主要思路如以下代码所示。

|  |
| --- |
| public class DefaultModelBinder : IModelBinder  {  public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)  {  return this.GetModel(controllerContext, bindingContext.ModelType, bindingContext.ValueProvider, bindingContext.ModelName);  }  public object GetModel(ControllerContext controllerContext, Type modeType, IValueProvider valueProvider, string key)  {  if (!valueProvider.ContainsPrefix(key))  return null;  return valueProvider.GetValue(key).ConvertTo(modeType);  }  } |

以上可以比较明确的看到类型的绑定过程体现在GetModel方法中，所用参数均来自于绑定上下文对象，通过模型名称获得值并转化为指定模型类型，在简单类型的处理上，已完全满足。接下来是复杂类型，通过GetComplexModel方法获得复杂对象，其实也比较好理解，绑定的过程是一个递归的过程，它通过反射根据数据类型创建对象，并将相关值赋到其属性上。每一次的递归都是将属性名称作为前缀附加到现有前缀作为下一次递归的前缀。之后还有数组，集合，字典等类型的绑定，其中都包含一个深复刻的过程，有部分会涉及泛型方法的反射应用，都比较相似就不一一介绍了。

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. 183-253