作为Controller基类ControllerBase的Execute方法的核心在于对Action方法的执行和作为方法返回的ActionResult的执行，两者的执行是通过一个叫做ActionInvoker的组件来完成的。

**一、ActionInvoker**

我们同样为ActionInvoker定义了一个接口IActionInvoker。如下面的代码片断所示，该接口定义了一个唯一的方法InvokeAction用于执行指定名称的Action方法，该方法的第一个参数是一个表示基于当前Controller上下文的ControllerContext对象。

1: public interface IActionInvoker

2: {

3: void InvokeAction(ControllerContext controllerContext, string actionName);

4: }

ControllerContext类型在真正的ASP.NET MVC框架中要负责一些，在这里我们对它进行了简化，仅仅将它表示成对当前Controller和请求上下文的封装，而这两个要素分别通过如下所示的Controller和RequestContext属性表示。

1: public class ControllerContext

2: {

3: public ControllerBase Controller { get; set; }

4: public RequestContext RequestContext { get; set; }

5: }

ControllerBase中表示ActionInvoker的同名属性在构造函数中被初始化。在Execute方法中，通过作为方法参数的RequestContext对象创建ControllerContext对象，并通过包含在RequestContext中的RouteData得到目标Action的名称，然后将这两者作为参数调用ActionInvoker的InvokeAction方法。

从前面给出的关于ControllerBase的定义我们可以看到在构造函数中默认创建的ActionInvoker是一个类型为ControllerActionInvoker的对象。如下所示的代码片断反映了整个ControllerActionInvoker的定义，而InvokeAction方法的目的在于实现针对Action方法的执行。由于Action方法具有相应的参数，在执行Action方法之前必须进行参数的绑定。ASP.NET MVC将这个机制成为Model的绑定，而这又涉及到另一个重要的组件ModelBinder。

1: public class ControllerActionInvoker : IActionInvoker

2: {

3: public IModelBinder ModelBinder { get; private set; }

4: public ControllerActionInvoker()

5: {

6: this.ModelBinder = new DefaultModelBinder();

7: }

8: public void InvokeAction(ControllerContext controllerContext, string actionName)

9: {

10: MethodInfo method = controllerContext.Controller.GetType().GetMethods().First(m => string.Compare(actionName, m.Name, true) == 0);

11: List<object> parameters = new List<object>();

12: foreach (ParameterInfo parameter in method.GetParameters())

13: {

14: parameters.Add(this.ModelBinder.BindModel(controllerContext, parameter.Name, parameter.ParameterType));

15: }

16: ActionResult actionResult = method.Invoke(controllerContext.Controller, parameters.ToArray()) as ActionResult;

17: actionResult.ExecuteResult(controllerContext);

18: }

19: }

**二、ModelBinder**

我们为ModelBinder提供了一个如下一个简单的定义，这与在真正的ASP.NET MVC中的同名接口的定义不尽相同。该接口具有唯一的BindModel根据ControllerContext和Model名称（在这里实际上是参数名称）和类型得到一个作为参数的对象。

1: public interface IModelBinder

2: {

3: object BindModel(ControllerContext controllerContext, string modelName, Type modelType);

4: }

通过前面给出的关于ControllerActionInvoker的定义我们可以看到在构造函数中默认创建的ModelBinder对象是一个DefaultModelBinder对象。由于仅仅是对ASP.NET MVC的模拟，定义在自定义的DefaultModelBinder中的Model绑定逻辑比ASP.NET MVC中同名类型中实现的要简单得多。

如下面的代码片断所示，绑定到参数上的数据具有三个来源：HTTP-POST Form、RouteData和Values和DataTokens，它们都是字典结构的数据集合。如果参数类型为字符串或者简单的值类型，我们直接根据参数名称和Key进行匹配；对于复杂类型（比如之前例子中定义的包含Contrller和Action名称的数据类型SimpleModel），则通过反射根据类型创建新的对象并根据属性名称与Key的匹配关系对相应的属性进行赋值。

1: public class DefaultModelBinder : IModelBinder

2: {

3: public object BindModel(ControllerContext controllerContext, string modelName, Type modelType)

4: {

5: if (modelType.IsValueType || typeof(string) == modelType)

6: {

7: object instance;

8: if (GetValueTypeInstance(controllerContext, modelName, modelType, out instance))

9: {

10: return instance;

11: };

12: return Activator.CreateInstance(modelType);

13: }

14: object modelInstance = Activator.CreateInstance(modelType);

15: foreach (PropertyInfo property in modelType.GetProperties())

16: {

17: if (!property.CanWrite || (!property.PropertyType.IsValueType && property.PropertyType!= typeof(string)))

18: {

19: continue;

20: }

21: object propertyValue;

22: if (GetValueTypeInstance(controllerContext, property.Name, property.PropertyType, out propertyValue))

23: {

24: property.SetValue(modelInstance, propertyValue, null);

25: }

26: }

27: return modelInstance;

28: }

29: private bool GetValueTypeInstance(ControllerContext controllerContext, string modelName, Type modelType, out object value)

30: {

31: var form = HttpContext.Current.Request.Form;

32: string key;

33: if (null != form)

34: {

35: key = form.AllKeys.FirstOrDefault(k => string.Compare(k, modelName, true) == 0);

36: if (key != null)

37: {

38: value = Convert.ChangeType(form[key], modelType);

39: return true;

40: }

41: }

42:

43: key = controllerContext.RequestContext.RouteData.Values

44: .Where(item => string.Compare(item.Key, modelName, true) == 0)

45: .Select(item => item.Key).FirstOrDefault();

46: if (null != key)

47: {

48: value = Convert.ChangeType(controllerContext.RequestContext.RouteData.Values[key], modelType);

49: return true;

50: }

51:

52: key = controllerContext.RequestContext.RouteData.DataTokens

53: .Where(item => string.Compare(item.Key, modelName, true) == 0)

54: .Select(item => item.Key).FirstOrDefault();

55: if (null != key)

56: {

57: value = Convert.ChangeType(controllerContext.RequestContext.RouteData.DataTokens[key], modelType);

58: return true;

59: }

60: value = null;

61: return false;

62: }

63: }

在ControllerActionInvoker的InvokeAction方法中，我们直接将传入的Action名称作为方法名从Controller类型中得到表示Action操作的MethodInfo对象。然后遍历MethodInfo的参数列表，对于每一个ParameterInfo对象，我们将它的Name和ParameterType属性表示的参数名称和类型连同创建ControllerContext作为参数调用ModelBinder的BindModel方法并得到对应的参数值。最后通过反射的方式传入参数列表并执行MethodInfo。和真正的ASP.NET MVC一样，定义在Contrller的Action方法返回一个ActionResult对象，我们通过指定它的Execute方法是先对请求的响应。

**三、ActionResult**

我们为具体的ActionResult定义了一个ActionResult抽象基类。如下面的代码片断所示，该抽象类具有一个参数类型为ControllerContext的抽象方法ExecuteResult，我们最终对请求的响应就实现在这里。

1: public abstract class ActionResult

2: {

3: public abstract void ExecuteResult(ControllerContext context);

4: }

在之前创建的例子中，Action方法返回的是一个类型为RawContentResult的对象。顾名思义，RawContentResult将初始化时指定的内容（字符串）原封不动地写入针对当前请求的HTTP回复中，具体的实现如下所示。

1: public class RawContentResult: ActionResult

2: {

3: public string RawData { get; private set; }

4: public RawContentResult(string rawData)

5: {

6: RawData = rawData;

7: }

8: public override void ExecuteResult(ControllerContext context)

9: {

10: context.RequestContext.HttpContext.Response.Write(this.RawData);

11: }

12: }

[ASP.NET MVC是如何运行的[1]: 建立在“伪”MVC框架上的Web应用](http://www.cnblogs.com/artech/archive/2012/03/11/mvc-how-to-work-01.html)   
[ASP.NET MVC是如何运行的[2]: URL路由](http://www.cnblogs.com/artech/archive/2012/03/11/mvc-how-to-work-02.html)   
[ASP.NET MVC是如何运行的[3]: Controller的激活](http://www.cnblogs.com/artech/archive/2012/03/12/mvc-how-to-work-03.html)   
[ASP.NET MVC是如何运行的[4]: Action的执行](http://www.cnblogs.com/artech/archive/2012/03/12/mvc-how-to-work-04.html)