AppManagerAppDomainFactory.Create

说明：IIS（非托管代码）使用COM方式调用AppManagerAppDomainFactory.Create方法，该方法返回ISAPIRuntime给IIS

namespace System.Web.Hosting  
{  
 using System;  
 using System.Globalization;  
 using System.IO;  
 using System.Runtime.InteropServices;  
 using System.Runtime.Remoting;  
 using System.Security.Permissions;  
 using System.Web.Compilation;  
 using System.Web.Util;  
  
 [SecurityPermission(SecurityAction.LinkDemand, Unrestricted=true)]  
 public sealed class AppManagerAppDomainFactory : IAppManagerAppDomainFactory  
 {  
 private ApplicationManager \_appManager = ApplicationManager.GetApplicationManager();  
  
 public AppManagerAppDomainFactory()  
 {  
 this.\_appManager.Open();  
 }  
  
 internal static string ConstructSimpleAppName(string virtPath, bool isDevEnvironment)  
 {  
 if (virtPath.Length > 1)  
 {  
 return virtPath.Substring(1).ToLower(CultureInfo.InvariantCulture).Replace('/', '\_');  
 }  
 if (!BuildManagerHost.InClientBuildManager & isDevEnvironment)  
 {  
 return "vs";  
 }  
 return "root";  
 }  
  
 [return: MarshalAs(UnmanagedType.Interface)]  
 public object Create(string appId, string appPath)  
 {  
 object obj2;  
 try  
 {  
 if (appPath[0] == '.')  
 {  
 appPath = new FileInfo(appPath).FullName;  
 }  
 if (!StringUtil.StringEndsWith(appPath, '\\'))  
 {  
 appPath = appPath + @"\";  
 }  
 ISAPIApplicationHost appHost = new ISAPIApplicationHost(appId, appPath, false);  
 ISAPIRuntime o = (ISAPIRuntime) this.\_appManager.CreateObjectInternal(appId, typeof(ISAPIRuntime), appHost, false, null);  
 o.StartProcessing();  
 obj2 = new ObjectHandle(o);  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
 return obj2;  
 }  
  
 public void Stop()  
 {  
 this.\_appManager.Close();  
 }  
 }  
}

ISAPIRuntime.ProcessRequest

说明：IIS（非托管代码）使用COM方式调用（异步）ISPAIRuntime.ProcessRequest方法。参数ecb实现了ISAPI和ISAPIRuntime的交互

[SecurityPermission(SecurityAction.LinkDemand, Unrestricted=true)]  
public int ProcessRequest(IntPtr ecb, int iWRType)  
{  
 IntPtr zero = IntPtr.Zero;  
 if (iWRType == 2)  
 {  
 zero = ecb;  
 ecb = UnsafeNativeMethods.GetEcb(zero);  
 }  
 ISAPIWorkerRequest wr = null;  
 try  
 {  
 bool useOOP = iWRType == 1;  
 wr = ISAPIWorkerRequest.CreateWorkerRequest(ecb, useOOP);  
 wr.Initialize();  
 string appPathTranslated = wr.GetAppPathTranslated();  
 string appDomainAppPathInternal = HttpRuntime.AppDomainAppPathInternal;  
 if ((appDomainAppPathInternal == null) || StringUtil.EqualsIgnoreCase(appPathTranslated, appDomainAppPathInternal))  
 {  
 HttpRuntime.ProcessRequestNoDemand(wr);  
 return 0;  
 }  
 object[] args = new object[] { appDomainAppPathInternal, appPathTranslated };  
 HttpRuntime.ShutdownAppDomain(ApplicationShutdownReason.PhysicalApplicationPathChanged, SR.GetString("Hosting\_Phys\_Path\_Changed", args));  
 return 1;  
 }  
 catch (Exception exception)  
 {  
 try  
 {  
 WebBaseEvent.RaiseRuntimeError(exception, this);  
 }  
 catch  
 {  
 }  
 if ((wr == null) || !(wr.Ecb == IntPtr.Zero))  
 {  
 throw;  
 }  
 if (zero != IntPtr.Zero)  
 {  
 UnsafeNativeMethods.SetDoneWithSessionCalled(zero);  
 }  
 if (exception is ThreadAbortException)  
 {  
 Thread.ResetAbort();  
 }  
 return 0;  
 }  
}

HttpRuntime.ProcessRequestNoDemand

说明：HttpRuntime. ProcessRequestNoDemand调用HttpRuntime.ProcessRequestInternal方法

private void ProcessRequestInternal(HttpWorkerRequest wr)  
{  
 Interlocked.Increment(ref this.\_activeRequestCount);  
 if (this.\_disposingHttpRuntime)  
 {  
 try  
 {  
 wr.SendStatus(0x1f7, "Server Too Busy");  
 wr.SendKnownResponseHeader(12, "text/html; charset=utf-8");  
 byte[] bytes = Encoding.ASCII.GetBytes("<html><body>Server Too Busy</body></html>");  
 wr.SendResponseFromMemory(bytes, bytes.Length);  
 wr.FlushResponse(true);  
 wr.EndOfRequest();  
 }  
 finally  
 {  
 Interlocked.Decrement(ref this.\_activeRequestCount);  
 }  
 }  
 else  
 {  
 HttpContext context;  
 try  
 {  
 context = new HttpContext(wr, false);  
 }  
 catch  
 {  
 try  
 {  
 wr.SendStatus(400, "Bad Request");  
 wr.SendKnownResponseHeader(12, "text/html; charset=utf-8");  
 byte[] data = Encoding.ASCII.GetBytes("<html><body>Bad Request</body></html>");  
 wr.SendResponseFromMemory(data, data.Length);  
 wr.FlushResponse(true);  
 wr.EndOfRequest();  
 return;  
 }  
 finally  
 {  
 Interlocked.Decrement(ref this.\_activeRequestCount);  
 }  
 }  
 wr.SetEndOfSendNotification(this.\_asyncEndOfSendCallback, context);  
 HostingEnvironment.IncrementBusyCount();  
 try  
 {  
 try  
 {  
 this.EnsureFirstRequestInit(context);  
 }  
 catch  
 {  
 if (!context.Request.IsDebuggingRequest)  
 {  
 throw;  
 }  
 }  
 context.Response.InitResponseWriter();  
 IHttpHandler applicationInstance = HttpApplicationFactory.GetApplicationInstance(context);  
 if (applicationInstance == null)  
 {  
 throw new HttpException(SR.GetString("Unable\_create\_app\_object"));  
 }  
 if (EtwTrace.IsTraceEnabled(5, 1))  
 {  
 EtwTrace.Trace(EtwTraceType.ETW\_TYPE\_START\_HANDLER, context.WorkerRequest, applicationInstance.GetType().FullName, "Start");  
 }  
 if (applicationInstance is IHttpAsyncHandler)  
 {  
 IHttpAsyncHandler handler2 = (IHttpAsyncHandler) applicationInstance;  
 context.AsyncAppHandler = handler2;  
 handler2.BeginProcessRequest(context, this.\_handlerCompletionCallback, context);  
 }  
 else  
 {  
 applicationInstance.ProcessRequest(context);  
 this.FinishRequest(context.WorkerRequest, context, null);  
 }  
 }  
 catch (Exception exception)  
 {  
 context.Response.InitResponseWriter();  
 this.FinishRequest(wr, context, exception);  
 }  
 }  
}

ApplicationFactory.GetApplicationInstance

说明：Application\_Start方法在这里会被调用

internal static IHttpHandler GetApplicationInstance(HttpContext context)  
{  
 if (\_customApplication != null)  
 {  
 return \_customApplication;  
 }  
 if (context.Request.IsDebuggingRequest)  
 {  
 return new HttpDebugHandler();  
 }  
 \_theApplicationFactory.EnsureInited();  
 \_theApplicationFactory.EnsureAppStartCalled(context);  
 return \_theApplicationFactory.GetNormalApplicationInstance(context);  
}

HttpApplication.InitInternal

说明：override Init方法在这里被调用

internal void InitInternal(HttpContext context, HttpApplicationState state, MethodInfo[] handlers)  
{  
 this.\_state = state;  
 PerfCounters.IncrementCounter(AppPerfCounter.PIPELINES);  
 try  
 {  
 try  
 {  
 this.\_initContext = context;  
 this.\_initContext.ApplicationInstance = this;  
 context.ConfigurationPath = context.Request.ApplicationPathObject;  
 using (new DisposableHttpContextWrapper(context))  
 {  
 if (HttpRuntime.UseIntegratedPipeline)  
 {  
 try  
 {  
 context.HideRequestResponse = true;  
 this.\_hideRequestResponse = true;  
 this.InitIntegratedModules();  
 goto Label\_006B;  
 }  
 finally  
 {  
 context.HideRequestResponse = false;  
 this.\_hideRequestResponse = false;  
 }  
 }  
 this.InitModules();  
 Label\_006B:  
 if (handlers != null)  
 {  
 this.HookupEventHandlersForApplicationAndModules(handlers);  
 }  
 this.\_context = context;  
 if (HttpRuntime.UseIntegratedPipeline && (this.\_context != null))  
 {  
 this.\_context.HideRequestResponse = true;  
 }  
 this.\_hideRequestResponse = true;  
 try  
 {  
 this.Init();  
 }  
 catch (Exception exception)  
 {  
 this.RecordError(exception);  
 }  
 }  
 if (HttpRuntime.UseIntegratedPipeline && (this.\_context != null))  
 {  
 this.\_context.HideRequestResponse = false;  
 }  
 this.\_hideRequestResponse = false;  
 this.\_context = null;  
 this.\_resumeStepsWaitCallback = new WaitCallback(this.ResumeStepsWaitCallback);  
 if (HttpRuntime.UseIntegratedPipeline)  
 {  
 this.\_stepManager = new PipelineStepManager(this);  
 }  
 else  
 {  
 this.\_stepManager = new ApplicationStepManager(this);  
 }  
 this.\_stepManager.BuildSteps(this.\_resumeStepsWaitCallback);  
 }  
 finally  
 {  
 this.\_initInternalCompleted = true;  
 context.ConfigurationPath = null;  
 this.\_initContext.ApplicationInstance = null;  
 this.\_initContext = null;  
 }  
 }  
 catch  
 {  
 throw;  
 }  
}

HttpApplication.BeginProcessRequest

说明：请求开始进入管道，BeginRequest等事件均在ResumeSteps中被执行

IAsyncResult IHttpAsyncHandler.BeginProcessRequest(HttpContext context, AsyncCallback cb, object extraData)  
 {  
 this.\_context = context;  
 this.\_context.ApplicationInstance = this;  
 this.\_stepManager.InitRequest();  
 this.\_context.Root();  
 HttpAsyncResult result = new HttpAsyncResult(cb, extraData);  
 this.AsyncResult = result;  
 if (this.\_context.TraceIsEnabled)  
 {  
 HttpRuntime.Profile.StartRequest(this.\_context);  
 }  
 this.ResumeSteps(null);  
 return result;  
 }

HttpApplication.PipelineStepManager

using System;  
using System.Diagnostics;  
using System.Threading;  
using System.Web;  
using System.Web.Hosting;  
using System.Web.Util;  
  
internal class PipelineStepManager : HttpApplication.StepManager  
{  
 private WaitCallback \_resumeStepsWaitCallback;  
 private bool \_validateInputCalled;  
 private bool \_validatePathCalled;  
  
 internal PipelineStepManager(HttpApplication app) : base(app)  
 {  
 }  
  
 internal override void BuildSteps(WaitCallback stepCallback)  
 {  
 HttpApplication app = base.\_application;  
 HttpApplication.IExecutionStep step = new HttpApplication.MaterializeHandlerExecutionStep(app);  
 app.AddEventMapping("ManagedPipelineHandler", RequestNotification.MapRequestHandler, false, step);  
 app.AddEventMapping("ManagedPipelineHandler", RequestNotification.ExecuteRequestHandler, false, app.CreateImplicitAsyncPreloadExecutionStep());  
 HttpApplication.IExecutionStep step2 = new HttpApplication.CallHandlerExecutionStep(app);  
 app.AddEventMapping("ManagedPipelineHandler", RequestNotification.ExecuteRequestHandler, false, step2);  
 HttpApplication.IExecutionStep step3 = new HttpApplication.TransitionToWebSocketsExecutionStep(app);  
 app.AddEventMapping("ManagedPipelineHandler", RequestNotification.EndRequest, true, step3);  
 HttpApplication.IExecutionStep step4 = new HttpApplication.CallFilterExecutionStep(app);  
 app.AddEventMapping("AspNetFilterModule", RequestNotification.UpdateRequestCache, false, step4);  
 app.AddEventMapping("AspNetFilterModule", RequestNotification.LogRequest, false, step4);  
 this.\_resumeStepsWaitCallback = stepCallback;  
 }  
  
 internal override void InitRequest()  
 {  
 base.\_requestCompleted = false;  
 this.\_validatePathCalled = false;  
 this.\_validateInputCalled = false;  
 }  
  
 [DebuggerStepperBoundary]  
 internal override void ResumeSteps(Exception error)  
 {  
 HttpContext context = base.\_application.Context;  
 IIS7WorkerRequest workerRequest = context.WorkerRequest as IIS7WorkerRequest;  
 AspNetSynchronizationContextBase syncContext = context.SyncContext;  
 RequestNotificationStatus status = RequestNotificationStatus.Continue;  
 ThreadContext indicateCompletionContext = null;  
 bool flag = false;  
 bool synchronous = false;  
 bool flag3 = false;  
 bool completedSynchronously = false;  
 bool isReentry = false;  
 int eventCount = -1;  
 base.\_application.GetNotifcationContextProperties(ref isReentry, ref eventCount);  
 CountdownTask applicationInstanceConsumersCounter = base.\_application.ApplicationInstanceConsumersCounter;  
 using (context.RootedObjects.WithinTraceBlock())  
 {  
 if (!isReentry)  
 {  
 syncContext.AssociateWithCurrentThread();  
 }  
 try  
 {  
 if (applicationInstanceConsumersCounter != null)  
 {  
 applicationInstanceConsumersCounter.MarkOperationPending();  
 }  
 bool locked = false;  
 try  
 {  
 if (!isReentry)  
 {  
 if (context.InIndicateCompletion && (context.ThreadInsideIndicateCompletion == Thread.CurrentThread))  
 {  
 indicateCompletionContext = context.IndicateCompletionContext;  
 if (context.UsesImpersonation)  
 {  
 indicateCompletionContext.SetImpersonationContext();  
 }  
 }  
 else  
 {  
 indicateCompletionContext = base.\_application.OnThreadEnter(context.UsesImpersonation);  
 flag = true;  
 }  
 }  
 Label\_00C6:  
 if (syncContext.Error != null)  
 {  
 error = syncContext.Error;  
 syncContext.ClearError();  
 }  
 if (error != null)  
 {  
 base.\_application.RecordError(error);  
 error = null;  
 }  
 if (!this.\_validateInputCalled || !this.\_validatePathCalled)  
 {  
 error = this.ValidateHelper(context);  
 if (error != null)  
 {  
 goto Label\_00C6;  
 }  
 }  
 if (!isReentry && syncContext.PendingCompletion(this.\_resumeStepsWaitCallback))  
 {  
 base.\_application.AcquireNotifcationContextLock(ref locked);  
 context.NotificationContext.PendingAsyncCompletion = true;  
 }  
 else  
 {  
 bool flag7 = (((context.NotificationContext.Error != null) || context.NotificationContext.RequestCompleted) && (context.CurrentNotification != RequestNotification.LogRequest)) && (context.CurrentNotification != RequestNotification.EndRequest);  
 if (flag7 || (context.CurrentModuleEventIndex == eventCount))  
 {  
 status = flag7 ? RequestNotificationStatus.FinishRequest : RequestNotificationStatus.Continue;  
 if (context.NotificationContext.PendingAsyncCompletion)  
 {  
 context.Response.SyncStatusIntegrated();  
 context.NotificationContext.PendingAsyncCompletion = false;  
 synchronous = false;  
 flag3 = true;  
 goto Label\_037A;  
 }  
 if (flag7 || (UnsafeIISMethods.MgdGetNextNotification(workerRequest.RequestContext, RequestNotificationStatus.Continue) != 1))  
 {  
 synchronous = true;  
 flag3 = true;  
 goto Label\_037A;  
 }  
 int currentModuleIndex = 0;  
 bool isPostNotification = false;  
 int currentNotification = 0;  
 UnsafeIISMethods.MgdGetCurrentNotificationInfo(workerRequest.RequestContext, out currentModuleIndex, out isPostNotification, out currentNotification);  
 context.CurrentModuleIndex = currentModuleIndex;  
 context.IsPostNotification = isPostNotification;  
 context.CurrentNotification = (RequestNotification) currentNotification;  
 context.CurrentModuleEventIndex = -1;  
 eventCount = base.\_application.CurrentModuleContainer.GetEventCount(context.CurrentNotification, context.IsPostNotification) - 1;  
 }  
 context.CurrentModuleEventIndex++;  
 HttpApplication.IExecutionStep step = base.\_application.CurrentModuleContainer.GetNextEvent(context.CurrentNotification, context.IsPostNotification, context.CurrentModuleEventIndex);  
 context.SyncContext.Enable();  
 completedSynchronously = false;  
 error = base.\_application.ExecuteStep(step, ref completedSynchronously);  
 if (!completedSynchronously)  
 {  
 base.\_application.AcquireNotifcationContextLock(ref locked);  
 context.NotificationContext.PendingAsyncCompletion = true;  
 }  
 else  
 {  
 context.Response.SyncStatusIntegrated();  
 goto Label\_00C6;  
 }  
 }  
 }  
 finally  
 {  
 if (locked)  
 {  
 base.\_application.ReleaseNotifcationContextLock();  
 }  
 if (indicateCompletionContext != null)  
 {  
 if (context.InIndicateCompletion)  
 {  
 if (synchronous)  
 {  
 indicateCompletionContext.Synchronize();  
 indicateCompletionContext.UndoImpersonationContext();  
 }  
 else if (!indicateCompletionContext.HasBeenDisassociatedFromThread)  
 {  
 ThreadContext context3;  
 bool lockTaken = false;  
 try  
 {  
 context3 = indicateCompletionContext;  
 Monitor.Enter(context3, ref lockTaken);  
 if (!indicateCompletionContext.HasBeenDisassociatedFromThread)  
 {  
 indicateCompletionContext.DisassociateFromCurrentThread();  
 flag = false;  
 if (context.ThreadInsideIndicateCompletion == Thread.CurrentThread)  
 {  
 context.IndicateCompletionContext = null;  
 }  
 }  
 }  
 finally  
 {  
 if (lockTaken)  
 {  
 Monitor.Exit(context3);  
 }  
 }  
 }  
 }  
 else if (synchronous)  
 {  
 indicateCompletionContext.Synchronize();  
 context.IndicateCompletionContext = indicateCompletionContext;  
 flag = false;  
 indicateCompletionContext.UndoImpersonationContext();  
 }  
 else  
 {  
 indicateCompletionContext.DisassociateFromCurrentThread();  
 flag = false;  
 }  
 if (flag)  
 {  
 indicateCompletionContext.DisassociateFromCurrentThread();  
 }  
 }  
 }  
 Label\_037A:  
 if (flag3)  
 {  
 base.\_application.AsyncResult.Complete(synchronous, null, null, status);  
 }  
 }  
 finally  
 {  
 if (!isReentry)  
 {  
 syncContext.DisassociateFromCurrentThread();  
 }  
 if (applicationInstanceConsumersCounter != null)  
 {  
 applicationInstanceConsumersCounter.MarkOperationCompleted();  
 }  
 }  
 }  
 }  
  
 private Exception ValidateHelper(HttpContext context)  
 {  
 if (!this.\_validateInputCalled)  
 {  
 this.\_validateInputCalled = true;  
 try  
 {  
 context.Request.ValidateInputIfRequiredByConfig();  
 }  
 catch (Exception exception1)  
 {  
 return exception1;  
 }  
 }  
 if (!this.\_validatePathCalled)  
 {  
 this.\_validatePathCalled = true;  
 try  
 {  
 context.ValidatePath();  
 }  
 catch (Exception exception2)  
 {  
 return exception2;  
 }  
 }  
 return null;  
 }  
}

HttpApplication.ApplicationStepManager

using System;  
using System.Collections;  
using System.Diagnostics;  
using System.Threading;  
using System.Web;  
using System.Web.Configuration;  
using System.Web.Util;  
  
internal class ApplicationStepManager : HttpApplication.StepManager  
{  
 private int \_currentStepIndex;  
 private int \_endRequestStepIndex;  
 private HttpApplication.IExecutionStep[] \_execSteps;  
 private int \_numStepCalls;  
 private int \_numSyncStepCalls;  
 private WaitCallback \_resumeStepsWaitCallback;  
  
 internal ApplicationStepManager(HttpApplication app) : base(app)  
 {  
 }  
  
 internal override void BuildSteps(WaitCallback stepCallback)  
 {  
 ArrayList steps = new ArrayList();  
 HttpApplication app = base.\_application;  
 UrlMappingsSection urlMappings = RuntimeConfig.GetConfig().UrlMappings;  
 steps.Add(new HttpApplication.ValidateRequestExecutionStep(app));  
 steps.Add(new HttpApplication.ValidatePathExecutionStep(app));  
 if (urlMappings.IsEnabled && (urlMappings.UrlMappings.Count > 0))  
 {  
 steps.Add(new HttpApplication.UrlMappingsExecutionStep(app));  
 }  
 app.CreateEventExecutionSteps(HttpApplication.EventBeginRequest, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventAuthenticateRequest, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventDefaultAuthentication, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventPostAuthenticateRequest, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventAuthorizeRequest, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventPostAuthorizeRequest, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventResolveRequestCache, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventPostResolveRequestCache, steps);  
 steps.Add(new HttpApplication.MapHandlerExecutionStep(app));  
 app.CreateEventExecutionSteps(HttpApplication.EventPostMapRequestHandler, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventAcquireRequestState, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventPostAcquireRequestState, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventPreRequestHandlerExecute, steps);  
 steps.Add(app.CreateImplicitAsyncPreloadExecutionStep());  
 steps.Add(new HttpApplication.CallHandlerExecutionStep(app));  
 app.CreateEventExecutionSteps(HttpApplication.EventPostRequestHandlerExecute, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventReleaseRequestState, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventPostReleaseRequestState, steps);  
 steps.Add(new HttpApplication.CallFilterExecutionStep(app));  
 app.CreateEventExecutionSteps(HttpApplication.EventUpdateRequestCache, steps);  
 app.CreateEventExecutionSteps(HttpApplication.EventPostUpdateRequestCache, steps);  
 this.\_endRequestStepIndex = steps.Count;  
 app.CreateEventExecutionSteps(HttpApplication.EventEndRequest, steps);  
 steps.Add(new HttpApplication.NoopExecutionStep());  
 this.\_execSteps = new HttpApplication.IExecutionStep[steps.Count];  
 steps.CopyTo(this.\_execSteps);  
 this.\_resumeStepsWaitCallback = stepCallback;  
 }  
  
 internal override void InitRequest()  
 {  
 this.\_currentStepIndex = -1;  
 this.\_numStepCalls = 0;  
 this.\_numSyncStepCalls = 0;  
 base.\_requestCompleted = false;  
 }  
  
 [DebuggerStepperBoundary]  
 internal override void ResumeSteps(Exception error)  
 {  
 bool flag = false;  
 bool completedSynchronously = true;  
 HttpApplication application = base.\_application;  
 CountdownTask applicationInstanceConsumersCounter = application.ApplicationInstanceConsumersCounter;  
 HttpContext context = application.Context;  
 ThreadContext context2 = null;  
 AspNetSynchronizationContextBase syncContext = context.SyncContext;  
 try  
 {  
 if (applicationInstanceConsumersCounter != null)  
 {  
 applicationInstanceConsumersCounter.MarkOperationPending();  
 }  
 using (syncContext.AcquireThreadLock())  
 {  
 try  
 {  
 context2 = application.OnThreadEnter();  
 }  
 catch (Exception exception)  
 {  
 if (error == null)  
 {  
 error = exception;  
 }  
 }  
 try  
 {  
 try  
 {  
 Label\_004E:  
 if (syncContext.Error != null)  
 {  
 error = syncContext.Error;  
 syncContext.ClearError();  
 }  
 if (error != null)  
 {  
 application.RecordError(error);  
 error = null;  
 }  
 if (!syncContext.PendingCompletion(this.\_resumeStepsWaitCallback))  
 {  
 if ((this.\_currentStepIndex < this.\_endRequestStepIndex) && ((context.Error != null) || base.\_requestCompleted))  
 {  
 context.Response.FilterOutput();  
 this.\_currentStepIndex = this.\_endRequestStepIndex;  
 }  
 else  
 {  
 this.\_currentStepIndex++;  
 }  
 if (this.\_currentStepIndex >= this.\_execSteps.Length)  
 {  
 flag = true;  
 }  
 else  
 {  
 this.\_numStepCalls++;  
 syncContext.Enable();  
 error = application.ExecuteStep(this.\_execSteps[this.\_currentStepIndex], ref completedSynchronously);  
 if (completedSynchronously)  
 {  
 this.\_numSyncStepCalls++;  
 goto Label\_004E;  
 }  
 }  
 }  
 }  
 finally  
 {  
 if (flag)  
 {  
 context.RaiseOnRequestCompleted();  
 }  
 if (context2 != null)  
 {  
 try  
 {  
 context2.DisassociateFromCurrentThread();  
 }  
 catch  
 {  
 }  
 }  
 }  
 }  
 catch  
 {  
 throw;  
 }  
 }  
 if (flag)  
 {  
 context.RaiseOnPipelineCompleted();  
 context.Unroot();  
 application.AsyncResult.Complete(this.\_numStepCalls == this.\_numSyncStepCalls, null, null);  
 application.ReleaseAppInstance();  
 }  
 }  
 finally  
 {  
 if (applicationInstanceConsumersCounter != null)  
 {  
 applicationInstanceConsumersCounter.MarkOperationCompleted();  
 }  
 }  
 }  
}