**IHttpContextFactory**

IHttpContextFactory表示HttpContext工厂，其默认实现为HttpContextFactory

public interface IHttpContextFactory

{

    // 创建 HttpContext

    HttpContext Create(IFeatureCollection featureCollection);

    void Dispose(HttpContext httpContext);

}

IHttpContextFactory的Create方法在IHttpApplication的CreateContext中被调用，用于创建HttpContext

**HttpContext**

HttpContext是一个抽象类，其包含了一些重要的对象，如下：

public abstract class HttpContext

{

    // 请求类

    public abstract HttpRequest Request { get; }

    // 响应类

    public abstract HttpResponse Response { get; }

    // 认证管理器

    public abstract AuthenticationManager Authentication { get; }

    // 身份标识

    public abstract ClaimsPrincipal User { get; set; }

    // Session

    public abstract ISession Session { get; set; }

    // 依赖注入容器

    public abstract IServiceProvider RequestServices { get; set; }

    // 特性集合

    public abstract IFeatureCollection Features { get; }

}

**用于生成HttpContext的原始保存在哪？**

HttpContext的默认实现DefaultHttpContext，其接收原始数据作为参数，Features属性就是我们的原始数据

public class DefaultHttpContext : HttpContext

{

    public DefaultHttpContext (IFeatureCollection features);

    ...

}

HttpContext并不加工原始数据，只是作为一个访问原始数据的桥梁而已

**原始数据集合IFeatureCollection**

public interface IFeatureCollection : IEnumerable<KeyValuePair<Type, object>>

{

    // 获取特性

    TFeature Get<TFeature>();

    // 设置特性的实例

    void Set<TFeature>(TFeature instance);

}

IServer收到请求后，会生成一系列的原始数据，如：

var httpRequestFeature = new HttpRequestFeature();

然后将这些数据保存到IFeatureCollection中

featureCollection.Set<IHttpRequestFeature>(httpRequestFeature);

然后将featureCollection作为参数调用IHttpApplication的CreateContext方法

CreateContext方法将featureCollection作为参数调用IHttpContextFactory的Create

**原始数据的接口**

这里列举几个重要的接口

IHttpRequestFeature：获取描述请求的基本信息。

IHttpResponseFeature：控制对请求的响应。

IHttpAuthenticationFeature：

提供完成用户认证的AuthenticationHandler对象和表示当前用户的ClaimsPrincipal对象

IServiceProvidersFeature：提供根据服务注册创建的ServiceProvider。

ISessionFeature：提供描述当前会话的Session对象。

**HttpContext的属性如何访问原始数据**

以Request属性为例

在DefaultHttpContext的构造函数中实例了Request

\_request = new DefaultHttpRequest(this);

DefaultHttpRequest的构造函数通过乱七八糟的机制提取并保存其需要的原始数据

public DefaultHttpRequest (DefaultHttpContext context)

{

    \_context = context;

    // 提取并保存 DefaultHttpRequest 所需的 Features

    \_features.Initalize (context.Features);

}

当我们访问数据时，如访问Request.PathBase时

private IHttpRequestFeature HttpRequestFeature =>

    // 从乱七八糟的机制中取得 IHttpRequestFeature

    \_features.Fetch (ref \_features.Cache.Request, \_nullRequestFeature);

// 通过访问 IHttpRequestFeature 获取或设置数据

public override PathString PathBase {

    get { return new PathString (HttpRequestFeature.PathBase); }

    set { HttpRequestFeature.PathBase = value.Value; }

}

**最终关系图看起来如下**

