**IServer**

IServer代表一个服务器

public interface IServer : IDisposable {

    IFeatureCollection Features { get; }

    Task StartAsync<TContext> (IHttpApplication<TContext> application, CancellationToken cancellationToken);

    ...

}

Features：用于保存服务器的特性，可以看作一个键值对

StartAsync：启动服务器，该方法在webHost.Run(); 中被调用

其参数IHttpApplication<TContext> application封装了我们注册的中间件

StartAsync要完成的任务：

监听服务器请求

将请求发生至中间间处理

**IHttpApplication<TContext>**

IHttpApplication代表一个应用，ProcessRequestAsync代表应用的处理流程

public interface IHttpApplication<TContext>

{

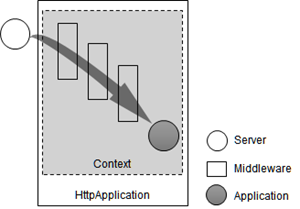
    TContext CreateContext(IFeatureCollection contextFeatures);

    Task ProcessRequestAsync(TContext context);

    void DisposeContext(TContext context, Exception exception);

}

IHttpApplication包含了我们的中间件，当我们调用ProcessRequestAsync时，其会调用我们注册的中间件

[](http://images2015.cnblogs.com/blog/19327/201610/19327-20161012225303500-287888110.png)

**IServer监听的消息如何传给IHttpApplication的中间件**

1. 调用CreateContext生成上下文（注：不是我们的HttpContext）
2. 调用ProcessRequestAsync处理上下文
3. 调用DisposeContext销毁上下文

**CreateContext做了什么**

var httpContext = \_httpContextFactory.Create(contextFeatures);

CreateContext将传进来的特征集合（IFeatureCollection contextFeatures）利用IHttpContextFactory生成HttpContext，并将其保存到Context中返回

**ProcessRequestAsync做了什么**

只有一行代码，即调用我们的中间件

我们注册的所有中间件处理流程其被封装为一个RequestDelegate方法

public Task ProcessRequestAsync (Context context)

{

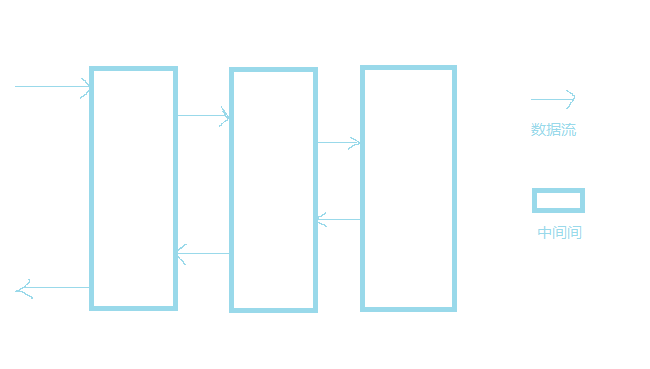
    return \_application (context.HttpContext);

}

private readonly RequestDelegate \_application;

public delegate Task RequestDelegate(HttpContext context);

RequestDelegate可以看作一个入口中间件，其会调用后续注册的中间件，后续注册的中间件又会调用后后续注册的中间件



**IApplicationBuilder中间件生成器**

IApplicationBuilder将我们注册的中间件组装成一个RequestDelegate

public interface IApplicationBuilder

{

    // 生成入口中间件 RequestDelegate

    RequestDelegate Build();

    // 注册中间件

    IApplicationBuilder Use(Func<RequestDelegate, RequestDelegate> middleware);

}

**中间件Func<RequestDelegate, RequestDelegate>**

我们注册的中间件以Func<RequestDelegate, RequestDelegate>展现

函数的第一个RequestDelegate是我们后一个中间件的处理流程，第二个RequestDelegate是我们当前中间件的处理流程，示例如下：

app.Use (next => async httpContext => {

    // ...当前中间件处理....

    // 调用后一个中间件处理

await next (httpContext);

    // ...当前中间件处理...

});

**中间件如何组装起来**

当我们调用最后一个中间件的Func，传入一个默认的RequestDelegate，于是我们得到了最后一个中间件的处理流程RequestDelegate，我们将最后一个中间件的处理流程RequestDelegate传入倒数第二个中间件的Func，于是我们得到了封装了两个中间件处理流程的RequestDelegate