**ASP.NET WebForm页面生命周期**

1. HTTP请求进入到HttpRuntime Pipeline的时候，当流程进入到第八个管道【**PostMapRequestHandler**】，会根据当前请求资源的后缀名，在machine.config(默认配置文件，文件名和其所在位置可能根据.NET Framework的版本而具有不同的体现，可以参考这篇文章：<https://stackoverflow.com/questions/24056966/cant-find-httpmodules-and-httphandlers-inside-machine-config>)或项目的webconfig配置文件下的<httpmodules>节点找到所对应的【**HttpHandler**】并创建之，WebForm项目隶属于【System.Web.UI.Page】，所以ASP.NET会根据在访问 \*.aspx 时所临时创建的dll，找到并实例化其中所对应的、继承于【System.Web.UI.Page】且实现【**IHttpHandler**】的【aspx网页文件名\_aspx】类



1. 需要注意的是【aspx网页文件名\_aspx】有两个需要我们关注的函数

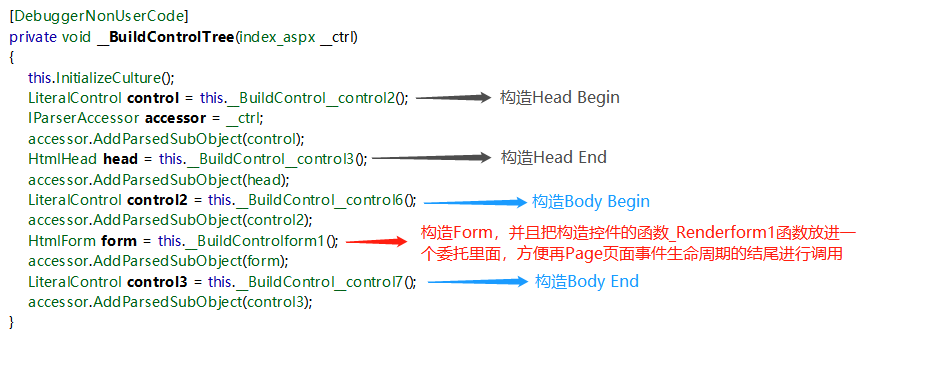
-> 其一是它重写与父类【System.Web.UI.TemplateControl】的函数【[FrameworkInitialize](http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://System.Web:4.0.0.0:b03f5f7f11d50a3a/System.Web.UI.TemplateControl/FrameworkInitialize()" \o )】该函数主要完成以下几件事

1. 构造ASP.NET控件树
2. 我们使用ASP.NET内联表达式所编写的C#代码也会整合在这里

-> 在HTML【Head】部分编写的C#代码则整合在【构造Head Begin】和【构造Head End】当中

-> 在HTML【Body】部分编写的C#代码则整合在【构造Body Begin】和【构造Body End】当中

-> 在HTML【Form】部分编写的C#代码则整合在【\_BuildControlfrom1】函数当中



-> 其二是实现于【**IHttpHandler**】的函数【ProcessRequest】



1. 我们可以把【ProcessRequest】函数视为页面生命周期的一个入口，不以为然，更准确的说法是在该函数的基础上所调用的【ProcessRequestMain】函数上面，从这里，我们就真正意义上的进入到了页面生周期之中



1. 在生命周期的开始，会先通过【DeterminPostBackMode】函数检查请求报文中是否带有【\_\_VIEWSTATE】键的参数，并通过该参数是否存在来确认当前Page的【IsPostBack】属性

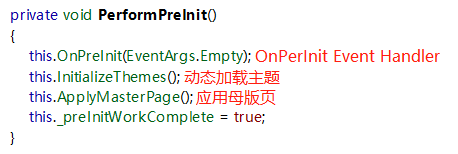


1. 调用【PerformPreInit】函数，那么在【PerformPreInit】函数的内部中又干了以下几件事

-> 调用Page的【Page\_PreInit】Event Handler

-> 动态加载主题

-> 应用母版页



1. 调用【InitRecursive】函数，在其内部干了以下几件事

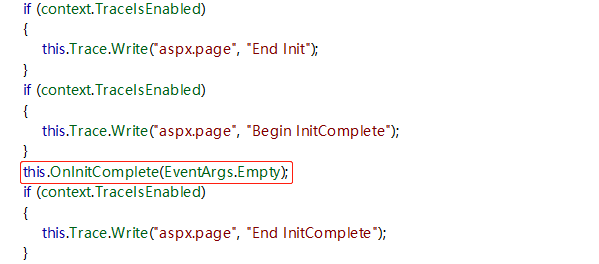
-> 通过递归方法调用Page页上所有控件的【Control\_Init】

-> 调用Page的【Page\_Init】

-> 标记ViewState为可追踪状态

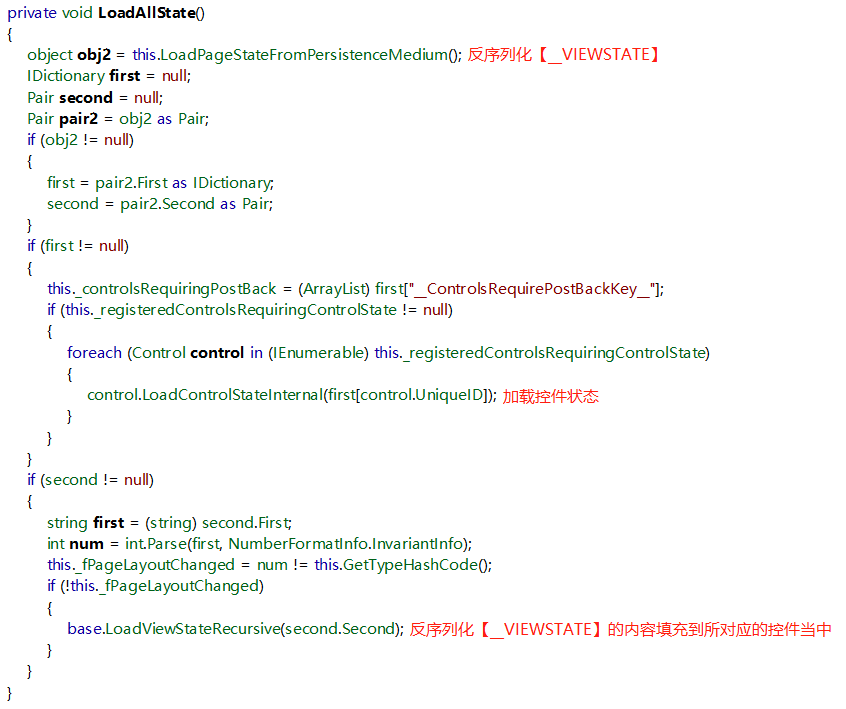


1. 通过函数【OnInitComplete】内部去调用Page的【Page\_InitComplete】



1. 紧接着，会通过判断【IsPostBack】，来确定是否调用【LoadAllState】和【ProcessPostData】，那么这两个函数的具体职责是干什么的呢？

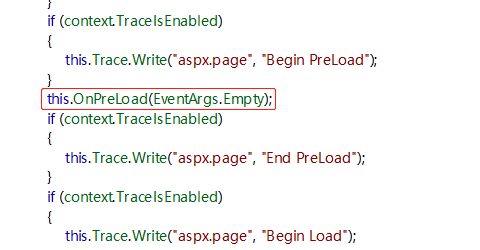
-> 在【LoadAllState】一共干了三件事，第一件事就是通过函数【LoadPageStateFromPersistenceMedium】将所提交上来的【\_\_VIEWSTATE】的值进行反序列化（【LoadPageStateFromPersistenceMedium】函数是一个虚函数，如果我们对该函数进行了重写，那么相应的进行页面控件树序列化的函数【SavePageStateToPersistenceMedium】也要进行重写），第二件事是递归调用【LoadControlState】函数进行【控件状态】的加载，第三件事就是将刚刚反序列化后的内容逐一填充到所对应的控件当中



-> 【ProcessPostData】其主要主责是负责处理控件的回传数据，我们知道每一个ASP.NET控件其实就是一个对象，当我们在一个 \*.aspx 页使用了控件后，在后面的【RenderControl】函数中会为其生成所对应的HTML标签（标签的ID和Name属性则为控件的Name属性），那么在当前 \*.aspx 页面上用户进行了数据提交之后，这些标签自然而然地也会被提交至服务器，【ProcessPostData】函数的作用就是将那些HTML标签所提交的数据都根据其ID一一加载到所对应的ASP.NET控件当中，这么做的目的是为了让我们能够在后面的页面生命周期事件中能够获取我们所使用的某个ASP.NET控件对象在上一步进行数据提交后的值



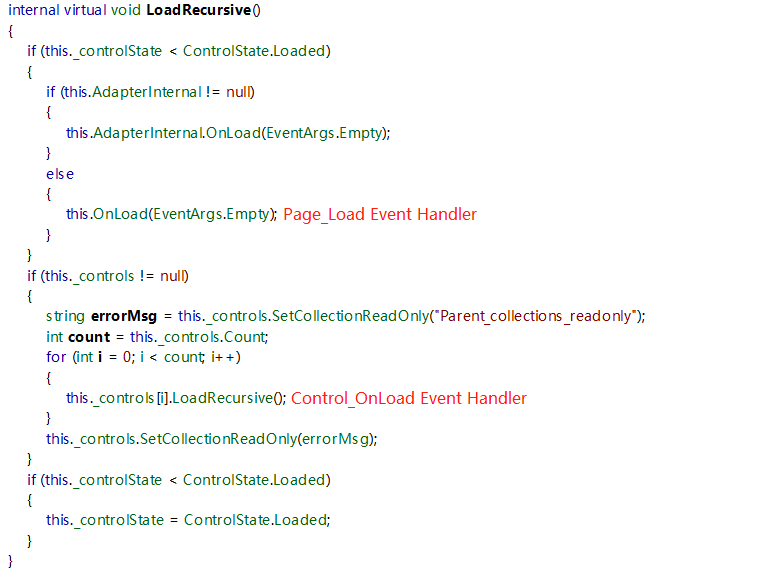
1. 会调用【OnPreLoad】函数，在该函数内部又调用了【Page\_PreLoad】Event Handler



1. 调用【LoadRecursive】在其内部干了以下几件事

-> 调用Page的【Page\_Load】Event Handler

-> 通过递归方法调用Page页上所有控件的【Control\_OnLoad】Event Handler

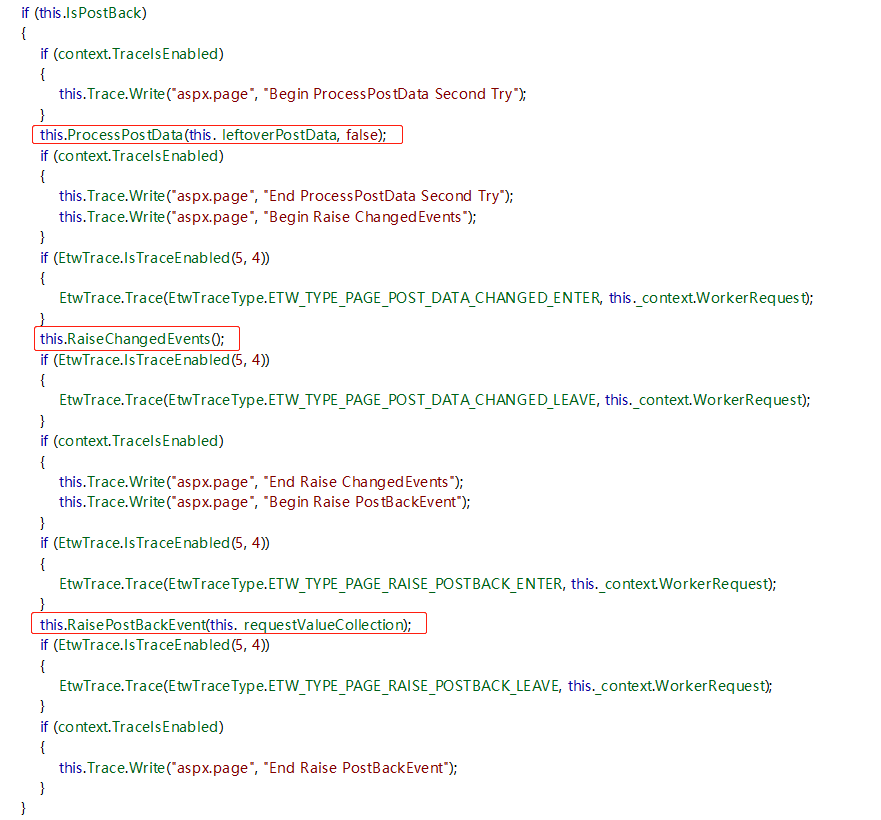


1. 紧接着，又经过一个【IsPostBack】的判断，那么在这个IF的上下文中又干了以下几件事

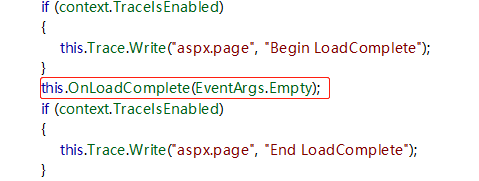
-> 二次调用【ProcessPostData】函数进行回传数据的处理，二次调用的目的用于防止在Page的【Page\_Load】Event Handler中动态创建了新的控件导致页面控件树改变而相应的状态还未更新的情况

-> 【[RaiseChangedEvents](http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://System.Web:4.0.0.0:b03f5f7f11d50a3a/System.Web.UI.Page/RaiseChangedEvents()" \o )】函数的内部会去调用实现了【IPostBackDataHandler】接口控件的【OnValueChanged】Event Handler，例如TextBox的【OnValueChanged】Event Handler那么ASP.NET是怎么知道具体是哪个控件所触发的【OnValueChanged】Event Handler呢？答案是通过所提交上来的【\_\_EVENTTARGET】和【\_\_EVENTARGUMENT】这两个隐藏域的值去判断的，具体可以查看[上一个文档的【](ASP.NET WebForm.docx)**[AutoPostBack](ASP.NET WebForm.docx)**[属性的详解】](ASP.NET WebForm.docx)

-> 【RaisePostBackEvent】函数会去调用实现了【IPostBackEventHandler】接口控件的【OnPostBack】Event Handler，例如Button控件的【OnClick】Event Handler



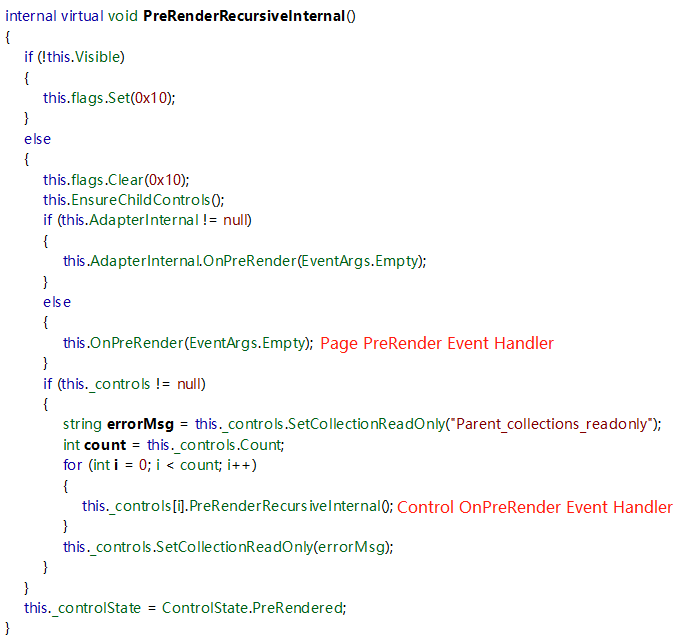
1. 调用【OnLoadComplete】函数，在该函数内部又调用了【Page\_LoadComplete】Event Handler



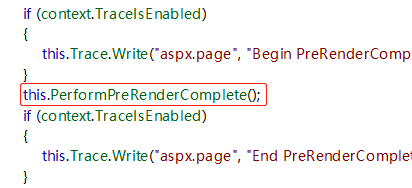
1. 调用【PreRenderRecursiveInternal】函数，那么在其内部干了以下几件事

-> 调用Page的【Page\_PreRender】Event Handler

-> 通过递归方法调用Page页上所有控件的【Control\_OnPreRender】Event Handler



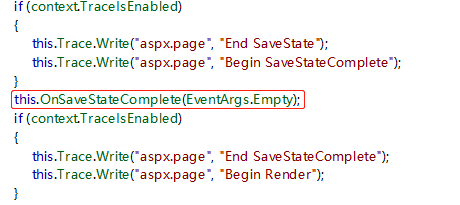
1. 调用【PerformPreRenderComplete】函数，在该函数内部又调用了【Page\_PreRenderComplete】Event Handlerdler



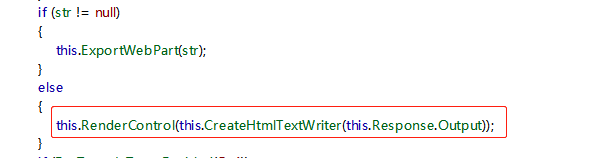
1. 调用【SaveAllState】函数，在该函数内部又通过【SavePageStateToPersistenceMedium】（【SavePageStateToPersistenceMedium】函数是一个虚函数，如果我们对该函数进行了重写，那么相应的进行页面控件树反序列化的函数【LoadPageStateFromPersistenceMedium】也要进行重写）为当前页面控件树的内容进行base64编码，并把编码后的内容存放至 \_\_VIEWSTATE 隐藏域当中



1. 调用【OnSaveStateComplete】函数，在该函数内部又调用了【Page\_SaveStateComplete】Event Handlerdler



1. 调用【RenderControl】函数，该函数会把所使用的ASP.NET控件转换成相应的HTML标签：前面在第二步的时候说到，通过【[FrameworkInitialize](http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://System.Web:4.0.0.0:b03f5f7f11d50a3a/System.Web.UI.TemplateControl/FrameworkInitialize()" \o )】函数内部调用【\_Render\_Control1】去构造ASP.NET页面控件树Control的时候并不是直接调用，而是会把【\_Render\_Control1】函数封装进一个委托里面，而这个委托就是在【RenderControl】函数内部进行调用的



1. 调用【UnloadRecursive】函数，在该函数内部又调用了【Page\_UnLoad】Event Handlerdler

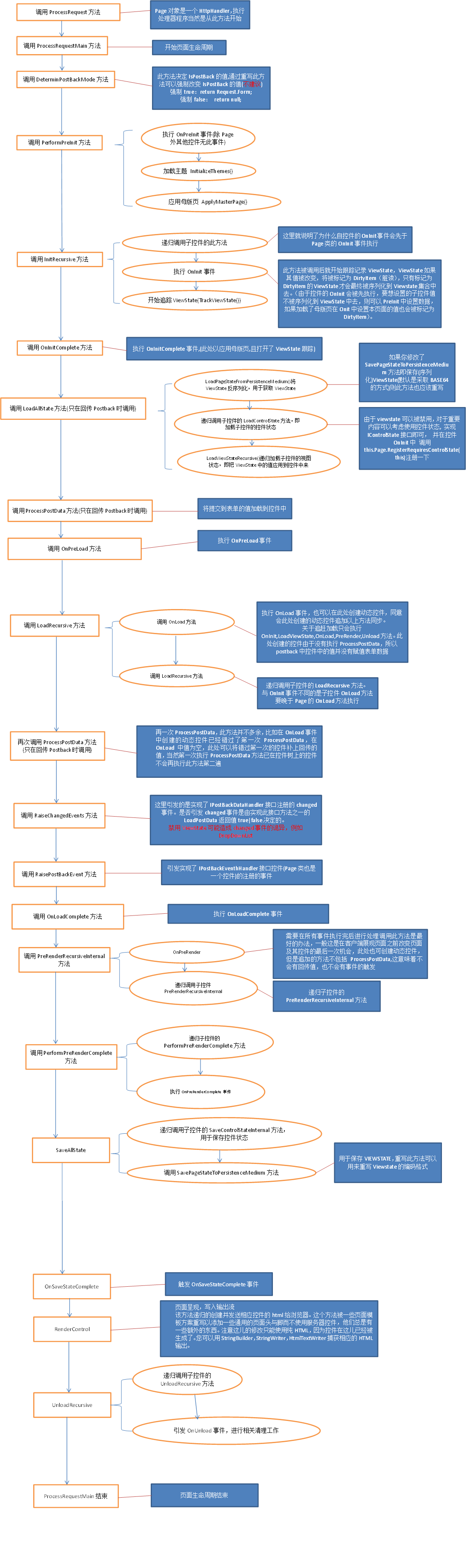


1. 至此，WebForm的生命周期结束
2. 总结

-> WebForm生命周期具体有哪些事件？



-> WebForm生命周期一个图纲流程



-> 参考网址

1. <https://www.w3cschool.cn/aspnet/1wiv1j2g.html>
2. <https://www.cnblogs.com/xhwy/archive/2012/05/20/2510178.html>
3. <https://www.cnblogs.com/edisonchou/p/4216337.html>