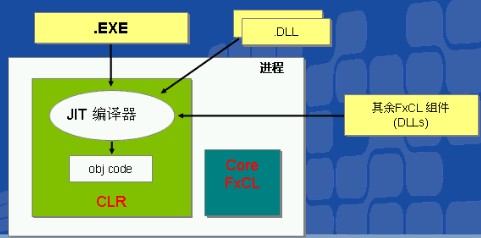
**[.NET 查找程序集路径（CLR关于Assembly的搜索路径的过程)](http://blog.csdn.net/wangjunhe/article/details/6692194)**

最近在回顾.Net应用程序的执行环境，这里做一个很小的总结，方面以后需要的时候进行查找：



CLR必须可以找到正确的Assembly，Net提供了Assembly搜索算法，可以根据.config文件(类似于.ini)自定义assembly搜索。

算法分析过程如下：

1、 在GAC(Global Assembly Cache)中搜索相应版本的DLL.

2、 配置文件(web.config或app.config)中

<codebase version=”Assembly Version" href=”URL of assembly” />

3、 应用程序(.exe)当前目录下

4、 配置文件(web.config或app.config)

<probing privatePath=”Paths” />

 OK,CLR就是根据上面的顺序从1到4进行搜索Assembly的。如果没有搜索到指定版本的DLL，则程序会抛出异常，提示：DLL文件无法找到。

下面通过程序DEMO来演示这个搜索算法：

我们新建立一个ClassLibary，名称为：NetLibraryTest，并且在一个类Charles.cs中建立一个方法 GetVersion()，为了测试CLR 查找DLL的顺序，我们针对上面的4个顺序实现4个不同的NetLibraryTest.dll 即：

一、GAC(Global Assembly Cache)

public class Charles2008  
    {  
        public string GetVersion()  
        {  
            return "GAC";  
        }  
}

 二、CodeBase元素

public class Charles2008  
    {  
        public string GetVersion()  
        {  
            return "CodeBase";  
        }  
}

三、应用程序根目录：

public class Charles2008  
    {  
        public string GetVersion()  
        {  
            return "CurrentRoot";  
        }  
}

四、Privatepath路径

public class Charles2008  
  {  
        public string GetVersion()  
        {  
            return "Privatepath";  
        }  
   }

 编译上面的4中不同的方法，即得到4个不同的NetLibraryTest.dll文件,另外为了放到GAC中，需要为该dll生成publicKeyToken的值。

OK,下面我另外建立一个项目:Console应用程序NetVersionTest.exe：(代码很简单)

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

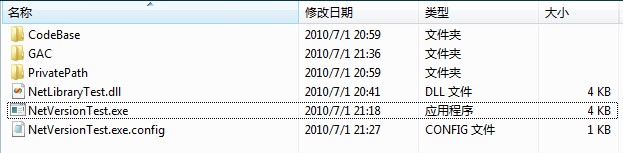
 class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Begin Process");  
            try  
            {  
                NetLibraryTest.Charles2008 Charles = new NetLibraryTest.Charles2008();  
                Console.WriteLine(Charles.GetVersion());  
            }  
            catch (Exception ex)  
            {  
                Console.WriteLine(ex.Message);  
            }  
            Console.WriteLine("End Process");  
            Console.ReadLine();  
        }  
}

配置文件NetVersionTest.exe.config的内容如下：

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <runtime>  
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">  
      <dependentAssembly>  
        <probing privatePath="PrivatePath"/>  
        <assemblyIdentity name="NetLibraryTest"  
                              publicKeyToken="a5015e7d589eac1d"  
                              culture="neutral" />  
        <codeBase version="1.0.242.0" href="CodeBase\NetLibraryTest.dll" />  
      </dependentAssembly>  
    </assemblyBinding>  
  </runtime>  
</configuration>

好了，代码部分就完成了，最后我们生成一个exe文件NetVersionTest.Exe，另外也去掉一些额外的文件，我们这样定义如下的结构：



备注：CodeBase文件夹下存放对应的NetLibaryTest.dll文件。即返回值为”CodeBase”。

同理：PrivatePath文件夹下也存放的是对应的NetLibaryTest.dll文件，即返回值：”PrivatePath”。

GAC文件夹也存放对应的NetLibaryTest.dll文件，即返回值：”GAC”。

OK,下面讨论几种情况：

1.    如果GAC里面的NetLibaryTest.dll文件放到GAC中去了，则最后NetVersionTest.exe首先会从GAC中里面去寻找,即最后输出:”GAC”

2.    如果NetLibaryTest.dll没有GAC中注册，则根据前面的顺序，NetVersionTest.exe则会调用CodeBase文件夹下的DLL.即最后输出：”CodeBase”

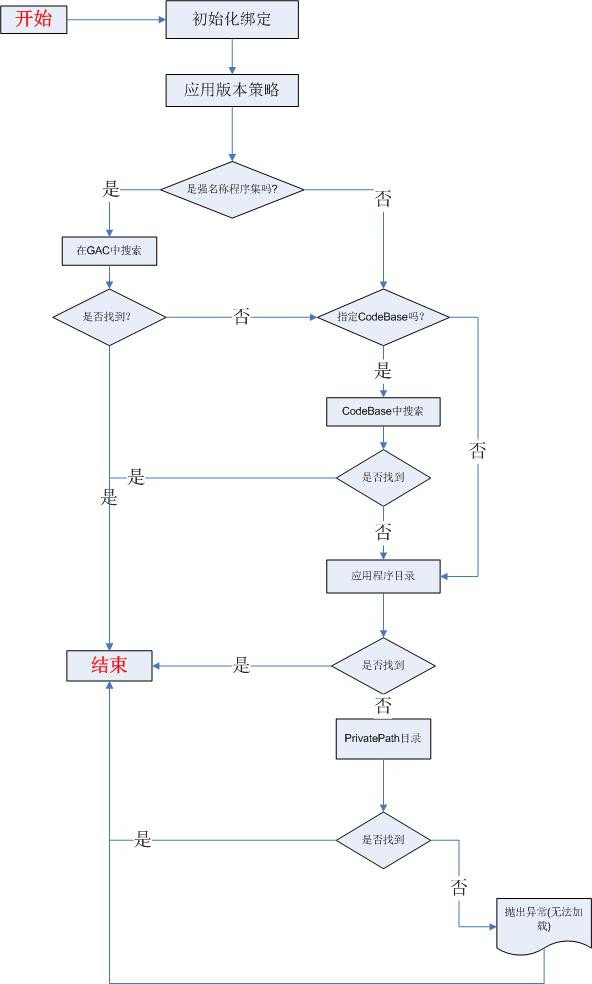
3.    如果在NetVersionTest.exe.config文件中没有CodeBase元素的配置的话，根据顺序则会调用到应用程序根目录下的NetLibaryTest.dll，即最后输出：”CurrentRoot”.

4.    最后一种情况：没有在GAC中注册，没有配置CodeBase,而且应用程序目录下也没有NetLibaryTest.dll文件，则CLR会从Probing的PrivatePath路径下去寻找指定的dll

5.    如果在1.2.3.4情况下都没有找到的话，则会发生异常，提示不能加载指定版本的dll文件。

**说明一下**：上面我测试用的Demo的那NetLibaryTest.dll是强名称的程序集，如果是弱名称的程序集则不会再GAC中去寻找（也就是跳过了第一步）。

为了更好的说明CLR搜索程序集的顺序，下面用一个流程图来表示：



附：

(1) 初始化绑定。基本上，这意味着从元数据中取出相关的AssemblyRef记录，并查看其中包括什么内容--它的外部程序集名称，它是否经过强命名，是否指定了文化等。

(2) 应用版本策略。这是一些由应用程序、被引用的共享程序集发布者或管理员生成的语句。这些语句包含在XML配置文件中，并且只是将程序集的特定版本（或一组版本）重定向到不同的版本。

**[配置程序集的版本策略](http://blog.csdn.net/wangjunhe/article/details/6692401)**

  本文转载于:<http://www.itgrass.com/a/dev/rjgc/200812/18-13060.html> 供朋友们学习和分享！

     在实际工作中，版本策略被使用得比较多，本节将借助分析这个面试题来简要介绍程序集的版本策略和配置方法。   
所涉及到的知识点   
• 版本策略的基本原理   
• 程序集版本策略的配置   
分析问题   
当一个程序集通过名字、版本、文化和公钥进行加载时，CLR允许程序员指定该程序集的哪些版本可以代替目前版本进行加载。这些都是通过版本策略来实现的。 所谓的版本策略，就是一个程序集版本的重定向，把加载当前这个版本定向到加载可替代的版本。版本策略可以在以下三个级别上进行配置：   
• 应用程序策略   
• 发行者策略   
• 计算机策略   
这三个级别的版本策略都可以通过XML文件来进行配置。现在笔者来分别介绍。   
**1．应用程序策略**   
应用程序策略可以在应用程序配置文件中进行配置，应用程序配置文件位于应用程序目录下。对于.EXE应用程序，其配置文件由exe文件名加 上.config后缀名构成，例如一个test.exe的应用程序，其配置文件就是test.exe.config。而对于任何一个Web应用程序来说， 其配置文件的文件名都是web.config。   
版本策略都记录在配置文件的assemblyBinding节点下。代码2-7是一个应用程序的配置文件例子，这里只选取了本节所关心的版本策略部分内容。   
代码2-7 VersionStrategy：Web.Config

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

<?xml version="1.0"?>   
<configuration>   
<runtime>   
<assemblyBinding>   
<!--对这个程序集进行版本重定向-->   
<dependentAssembly>   
<assemblyIdentity   
name="NET.MST.Second.Compile"   
publicKeyToken="60c29e5f0af3e9bb">   
</assemblyIdentity>   
</dependentAssembly>   
<!--重定向的策略-->   
<bindingRedirect oldversion="0.0.0.0-12.2.2.2"   
newversion="12.3.0.0">   
</bindingRedirect>   
</assemblyBinding>   
</runtime>   
</configuration>

在代码2-7这个配置文件中，指定了NET.MST.Second.Compile，60c29e5f0af3e9bb这个组件的版本策略，这个策略将0.0.0.0到12.2.2.2之间的所有版本重定向到12.3.0.0版本上。

**2．发行者策略**   
发行者策略是针对那些被放入全局程序集缓存（GAC）中的程序集，发行者策略以一个和代码2-6同样格式的配置文件形式绑定到程序集上，被一同放入GAC 中。发行者策略配置文件的文件名非常古怪，它是这样的一个字符串：主版本号.次版本号.程序集名.dll。正因为如此，一个程序集的每个主版本/次版本号 只能有一个发行者策略。   
**3．计算机策略**   
同样地，计算机策略同样由一个配置文件表示，它的格式也和代码2-7基本类似。计算机级版本策略配置文件的文件名为：machine.config，它被 存储在%SystemRoot%\ Microsoft.NET\ Framework\v\*\*\*\*\CONFIG\目录下。   
读到这里，读者可能会有这样的疑问：版本策略可以在3个级别进行配置，那这些策略是如何协作的呢？按照.NET的机制，3个级别的版本策略将会按照顺序依次执行，而上一级别的执行结果将会被作为下一级别的执行输入，图2.6展示了这个执行过程。   
如图2.6所示，3个级别版本策略被依照：应用程序、发行者、计算机的顺序依次执行。而其中，发行者策略是可选的，在以下两种情况下发行者策略将不会被执行。   
• 程序集没有被加入到GAC中   
• 应用程序策略制定忽略发行者策略   
在第一种情况下，根本就不存在发行者策略配置文件，当然CLR也就不会执行发行者策略。而第二种情况，是程序员在应用程序策略中指定忽略发行者策略，具体 做法是在应用程序配置文件中加入publisherPolicy节点，并且把apply属性值设置为no。代码2-8就是在代码2-7的基础上，指定了忽 略发行者策略。   
代码2-8 VersionStrategy-NoPolisherPolicy：Web.Config

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

<?xml version="1.0"?>   
<configuration>   
<runtime>   
<assemblyBinding>   
<!--对这个程序集进行版本重定向-->   
<dependentAssembly>   
<assemblyIdentity   
name="NET.MST.Second.Compile"   
publicKeyToken="60c29e5f0af3e9bb">   
</assemblyIdentity>   
</dependentAssembly>   
<!--重定向的策略-->   
<bindingRedirect oldversion="0.0.0.0-12.2.2.2"   
newversion="12.3.0.0">   
</bindingRedirect>   
<!--指定忽略发行者策略-->   
<publisherPolicy apply="no">   
</publisherPolicy>   
</assemblyBinding>   
</runtime>   
</configuration>

答案   
CLR支持在3个级别上设定版本策略，依次是：应用程序策略、发行者策略和计算机策略。所有策略的设置都是通过修改配置文件来实现。3个级别的策略依次会 被CLR执行，而上一个策略的执行结果将被作为下一个策略的输入。发行者策略仅仅针对那些放入GAC的程序集，并且可以在应用程序策略中被指定忽略。