1 简介

1.1 概述

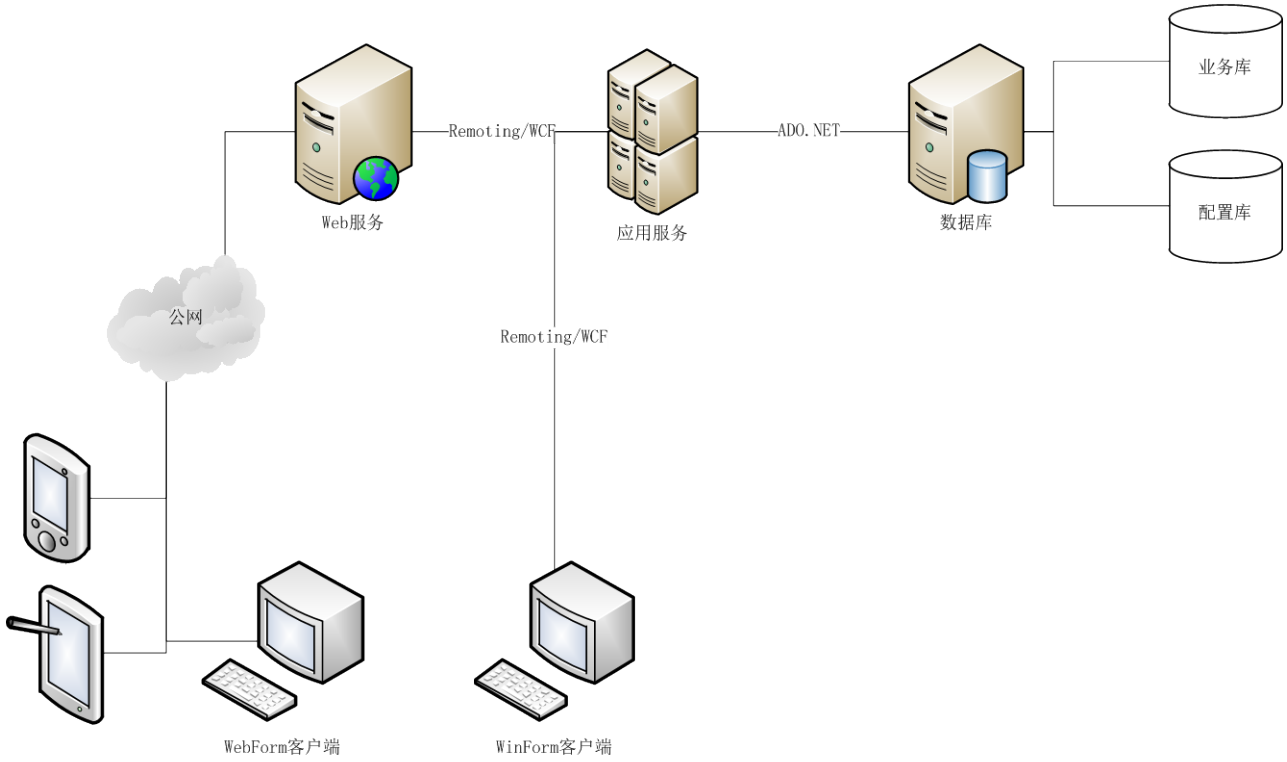
Phenix丶是基于 LAN/WAN、针对企业应用信息系统开发所提供的一套分布式软件快速开发工具和引擎。它采用多层、组件构架模式，在 CSLA 框架基础之上，对应用系统中普遍、通用的技术实现，做最大可能的封装。它为应用系统的开发，从分析、设计、编码一直到实施，提供一条快速、高效、简捷的实现途径。尽最大的努力让技术人员少操心技术实现，留出更多的精力去投入到业务逻辑的分析和设计当中，从而大大缩短应用系统的开发周期、进而降低开发成本，并有效地提高应用系统的稳定性、可维护性和兼容性。

1.2 特点

- 持久层：基本上透明化，无需编写代码（比如 SQL），减少编写代码工作量，特别是降低代码差错率。
- 业务层：封装 CSLA 开源框架，提供 Addin 嵌入到 IDE 的自有代码生成工具，可所见即所得生成和编辑业务类、查询类、业务规则类等，减少编写代码工作量，特别是降低代码差错率。
- 服务层：全透明化、无需编写代码（默认 remorting TCP 8086/8087 端口），可通过配置将系统运行在不同的服务协议（WCF/remorting/WebAPI）和通讯协议（TCP/HTTP）上。
- 表现层+UI 逻辑层：提供 WinForm 界面框架，统一界面风格（增删改查等标准功能按钮的 UI 逻辑等），界面功能组件操作权限的可配置化等。
- 提供标准的版本升级、用户验证、用户权限等功能和配置工具，无需在系统中编写权限验证代码。
- 提供标准的业务逻辑功能实现，比如分片区管理（通过配置工具定义数据的切片过滤器并与用户绑定，无需在系统中编写代码）、业务对象流水号的自动生成和填充、基本业务逻辑规则的验证，等等。
- 模块开发插件化。
- 支持跨平台应用软件的开发，WebAPI 数据服务可响应任何异构平台 Http 客户端的数据持久化请求，并提供了基于 Xamarin 的 Entity 框架。

1.3 系统架构

1.3.1 物理架构



Phenix丶继承 PACKER 开发平台的设计理念，为应用系统提供了 C/S/S、B/S/S 两种物理架构。

C/S/S 架构区别于普通的 C/S 架构，在于 WinForm 客户端与数据库（业务库与配置库在一个数据库内）之间增加了一个无状态的应用服务，使得负载均衡、应用发布、自动升级等等系统特性得以能够方便地实现，从而破解了传统观念对 C/S 物理架构下桌面应用的偏见。当然，如果希望应用系统仍采用 C/S 架构的话，系统登录组件 Phenix.Services.Client.Security.LogOn 提供了一键切换功能。这样，在应用系统的设计阶段，开发人员无需关心系统部署时到底要采取什么物理架构。这个一键切换功能，还常常被用于应用系统的调试阶段，不必为客户端和服务端的业务组件版本是否一致问题而揪心。

‘Web 服务’作为‘应用服务’特殊的调用方（客户端），有两点优势：

- 可重用性：“Web 服务”和“WinForm 客户端”共享‘应用服务’所提供的业务逻辑服务；
- 高安全性：业务逻辑、数据库连接串等敏感数据可以被部署到企业内网里；

但也带来缺点：

- 性能相对 B/S 架构要差点，因为多了‘Web 服务’与‘应用服务’之间的远程调用和数据转换传递的过程。

如果要兼顾上述这些优缺点的话，可考虑将‘Web 服务’与‘应用服务’部署在一台物理服务器上。当然，Phenix丶并不排斥应用系统直接采用 B/S 架构。

1.3.2 逻辑架构

应用系统的逻辑架构必须遵照面向对象、五层逻辑架构进行搭建。

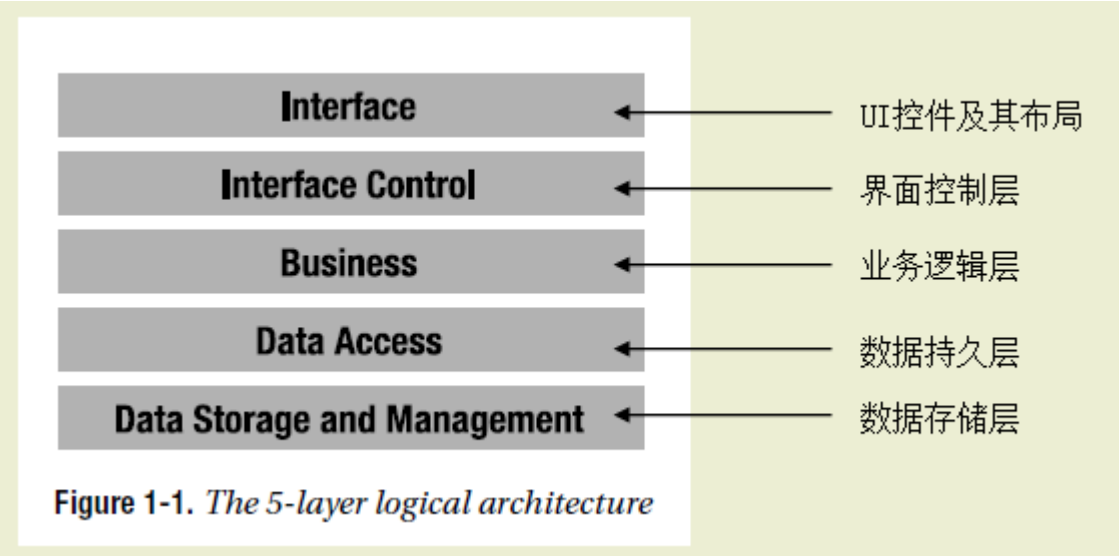


Figure 1-1. *The 5-layer logical architecture*

上图摘自 CSLA 作者的《Expert C# 2008 Business Objects》。

各个逻辑层之间是从上到下的依赖关系，下一层为上一层提供服务，上一层是服务的消费方。消费方必须遵照服务方所提供的接口规范（指广义的接口，形式可多种多样）来调用服务方所提供的服务。

逻辑架构并不和物理架构一一对应的。应用系统如何搭建物理架构，主要受其性能、环境等应用系统功能外的需求方面考虑的结果；而如何搭建逻辑架构，则主要是为了设计、团队、维护等软件工程角度考虑的结果。Phenix丶支持应用系统一键切换为纯粹的 C/S（物理）架构，但并没有改变这个应用系统的内里仍然是多层（逻辑）架构。

非面向对象的设计理念，造成的结果往往是 UI 层与业务逻辑层揉在了一起；业务逻辑层（所谓 BLL）成为了持久层（所谓 DAL）操作数据 CRUD 的再封装；而为了跨物理域，甚而多加两层对 BLL 再次封装（一个在物理域的服务端、一个在物理域的客户端，虽说有代码生成工具可以自动生成这些无聊的代码，但改变不了其混搅物理架构与逻辑架构、逻辑架构各层职责的事实），甚而将应用系统绑定到了一个具体的技术细节上，造成的后果是这些庞大繁复的代码难以维护，其生命周期受到技术升级的威胁。

1.3.3 系统工程划分

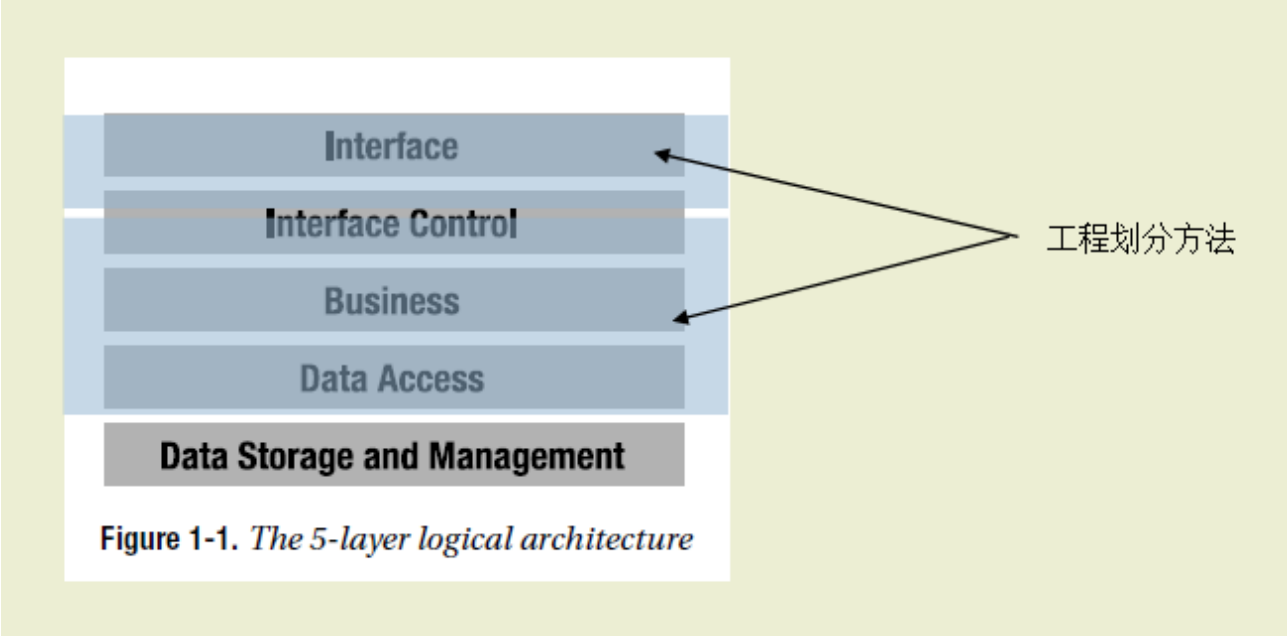


Figure 1-1. *The 5-layer logical architecture*

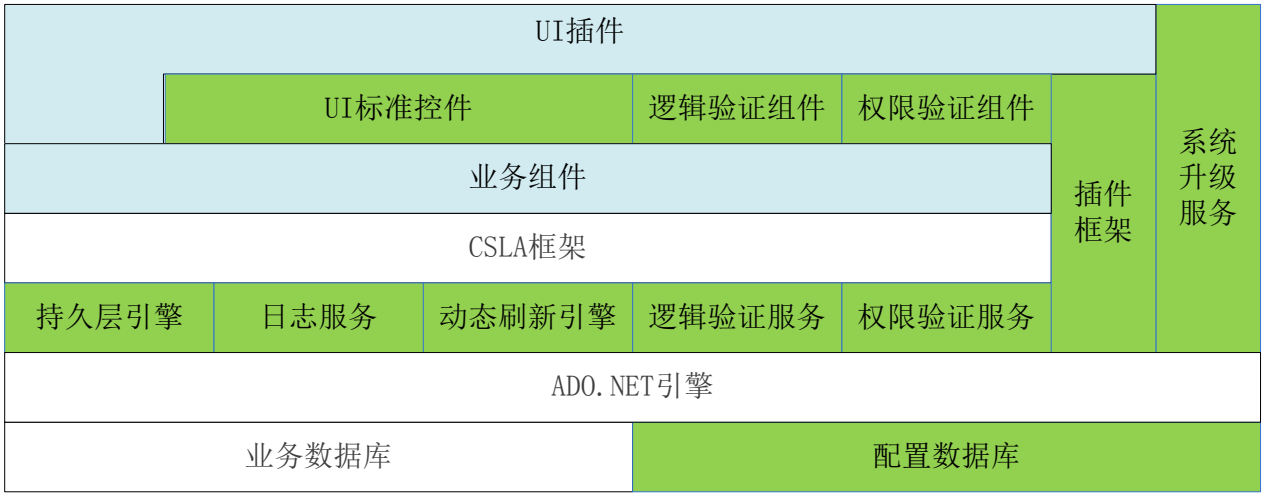
将逻辑架构与系统工程划分等同起来，也是种机械的认知行为。诸如在一个解决方案里纵向搭建 UI、BLL、DAL 工程（甚至更加细分）的行为，其实是叠床架屋，实乃迂腐古板，也无形中限制了开发效率的提升机会。

在 Phenix丶平台上搭建工程，仅需设计两种物理架构上的组件：界面组件和业务组件。界面组件包含了 UI 层和一部分界面控制层（直接操作界面控件）代码，业务组件包含了业务逻辑层和一部分界面控制层（提供界面操作、验证的逻辑控制）、持久层（因 Phenix丶持久层引擎内嵌在业务类中，只有在特殊应用场景下才需编写）代码。

总之，逻辑架构是一种设计规范，并通过 Phenix丶平台限定了这样一种开发模式，最低限度地减少繁复的、技术性的代码，引导开发人员将各层、各类代码编写在它正确的位置上，从而增强了代码的可读性、可维护性。

1.3.4 系统层次结构

Phenix丶为各个逻辑层提供了用于快速开发的接口、工具和引擎。



上图说明了应用系统是如何架构在 Phenix丶所提供的框架上的。绿色背景的模块为 Phenix丶实现的功能，蓝色背景的模块是需要应用系统自身实现的功能。

以下为 Phenix丶实现功能的清单：

内容	项目
多国语言	Phenix丶自身提供简繁英三种语言资源文件，可根据应用系统环境需要添加相关资源文件来做扩展，应用系统只要在主程序代码里指定系统的语言环境即可。
持久层引擎	为了将面向对象业务设计与关系数据库设计相融合，通常在应用系统逻辑分层上分出一个持久层，专门处理对象化存储，也就是将数据库中的数据在对象中实例化，当增删改这个对象时，数据库中的相应表的记录及字段也会随之改变，另外，还需实现事务管理、锁机制、缓存机制。 Phenix丶将持久层的功能做了完整的封装，应用系统的开发无需编写代码即可实现大部分场景下的对象持久化功能，这仅需在业务类上打上相关的 Mapping 标签，以实现持久层引擎的接口即可。
日志服务	对应用系统中敏感数据的更新，必须记录由何人、何时、何地（IP）、新旧数据值，以留下痕迹便于跟踪。 应用系统中，由数据库、中间层、客户端所产生的陷阱信息也归在日志管理中。 日志按照轻重程度、缓急程度进行分级。 按照日志类型和分级，可设置日志发送的目的地：指定终端、指定数据库表、监控中心、指定手机等等。（待实现）
动态刷新引擎	将应用系统中发生的数据变化，实时通知需要获知变更的组件中，使得数据在需要同步的组件中保持一致性。
逻辑验证服务	应用系统的开发工作需分清逻辑验证和权限验证。比如一个对象是否允许增删改，除了受权限规则限制外，还可能受到这个对象内某个（/些）属性值（往往是枚举值）限制，这后者的验证就是逻辑验证。 这些规则按照Phenix丶代码编写规范写在业务类中，而验证的效果表现在界面层。
逻辑验证组件	对应于逻辑验证服务的界面层组件。 在运行期自动控制界面层的逻辑规则。 在设计期，可通过 IDE 对逻辑规则进行配置。
权限验证服务	系统登录验证及系统功能的权限控制。 采用基于角色（权限-角色-用户）的系统安全控制模型。 提供对分公司、分片区等集团企业管理模型的支持。 应用系统无需编写验证代码，运行期的验证控制完全由 Phenix丶实现。

	Phenix.Security 组件包提供了权限配置功能，可在运行期供应用系统实施人员和用户使用。
权限验证组件	对应于权限验证服务的界面层组件。 在运行期自动控制界面层的权限规则。
插件框架	主要用于界面层、客户端的开发。 规范主程序与窗体组件间的接口。 窗体组件（DLL）以插件（Phenix.Core.Plugin.IPlugin）形式进行封装，主程序通过插件容器（Phenix.Core.Plugin.PluginHost）来驱动窗体组件。 服务端插件，可挂到 Phenix.Services.Host.exe 上，作为服务端的任务被执行、暂停或中止。
系统升级服务	用户登录系统时，将比对服务端上的客户端文件版本是否和本地一致，否则下载更新。 客户端如需升级，仅需将最新的客户端文件放到 Phenix.Services.Host 所在目录的 ClientLibrary 子目录即可。 服务端升级，由 Phenix.Services.Host 中的一键升级功能实现，最大限度减少对在线用户的影响。
负载均衡	在多个应用服务器提供服务的情况下，客户端应均衡连接服务器。而一旦某台服务器当机时，客户端可以自动尝试连接其他服务器。 基于 Phenix丶实现的应用系统是自动实现负载均衡的。
用户界面框架	总结 WinForm 界面层常用的界面模式，归整为基于 Phenix丶统一的控件、组件或模板，即方便开发也统一了应用系统的界面风格。
通用报表	用户可以自定义报表，自定义的报表模板可按功能（树状层次）保存供以后或其他用户重复使用。 报表不仅要输出到打印机，而且应该能输出到 Text、RTF、PDF、Excel、Word、HTML 等等文档格式，并能够通过 Email 发送出去。 报表可以人为增加、删除列表项，人为将列表项重新排列、进一步搜索列表项等等。
工作流	工作流是针对工作中具有固定程序的常规活动而提出的一个概念。 通过将工作活动分解成定义良好的任务、角色、规则和过程来进行执行和监控，达到提高生产组织水平和工作效率的目的。 工作流技术为企业更好地实现经营目标提供了先进的手段。
协同作业	作业的协同性在企业管理上非常重要，很多作业需要多角色的配合来完成，因此在应用系统的实现上，需要考虑协同性问题，以方便用户工作。 各部门、各人员，以及企业和外部资源之间可以共享信息、实现即时沟通和协同运作，大大提升运作效率，减少企业管理和运营的成本。

1.3.5 应用系统部署结构

1.3.5.1 第三方框架文件部署位置

文件名	客户端	服务端	客户端升级方式（注 4）
Csla.dll（v4.3.14）	√	√	拷贝到 ClientLibrary 目录/用 ClickOnce
ChnCharInfo.dll	√	√	拷贝到 ClientLibrary 目录/用 ClickOnce
System.Data.OracleClient.dll（注 1）		√	
Newtonsoft.Json.dll		√	
Microsoft.AspNet.SignalR（注 3）		√	
Microsoft.AspNet.WebApi（注 3）		√	

注 1：适用于 Oracle 数据库。

注 3：需运行在.NET Framework 4.6 以上环境。

注 4：指 Host 所在目录下的 ClientLibrary 子目录，当客户端通过 LogOnDialog 登录服务端时，会自动将该目录里的文件下载到本地，并覆盖客户端程序所在目录下的旧文件来升级终端。

1.3.5.2 Phenix丶框架文件部署位置

文件名	客户端	服务端	客户端升级方式（注 1）	服务端升级方式（注 2）
Phenix.Core.dll	√	√	用 ClickOnce	拷贝到 ServiceLibrary 目录
Phenix.Services.Contract.dll	√	√	用 ClickOnce	拷贝到 ServiceLibrary 目录
Phenix.Business.dll	√	√	拷贝到 ClientLibrary 目录/用 ClickOnce	拷贝到 ServiceLibrary 目录
Phenix.Services.Host.x86/x64.exe		√		拷贝到 ServiceLibrary 目录
Phenix.Services.Library.dll		√		拷贝到 ServiceLibrary 目录
Phenix.Services.Client.dll	√		用 ClickOnce	
Phenix.Windows.dll	√		拷贝到 ClientLibrary 目录	

注 1：指 Host 所在目录下的 ClientLibrary 子目录，当客户端通过 LogOnDialog 登录服务端时，会自动将该目录里的文件下载到本地，并覆盖客户端程序所在目录下的旧文件来升级终端。

注 2：指 Host 所在目录下的 ServiceLibrary 子目录，用于 Host 里自带的一键升级服务功能。也可以自行关闭 Host 服务程序，直接将新文件拷贝到 host 所在目录下覆盖旧文件来升级服务。

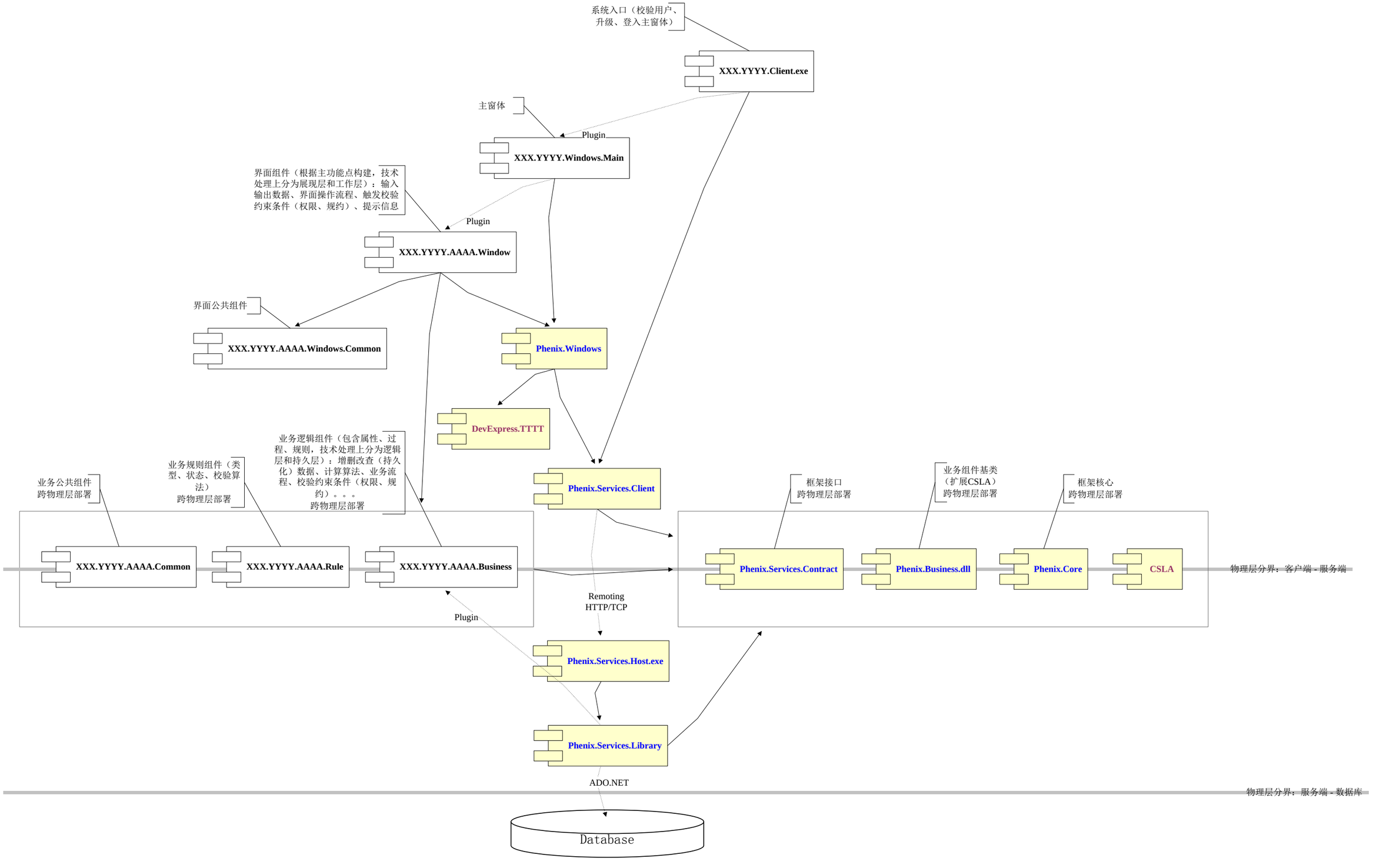
1.3.5.3 应用系统文件部署位置

文件名	客户端	服务端	客户端升级方式（注 1）	服务端升级方式（注 2）
客户端登录程序（*XXX.YYYY*.Client.exe）	√		用 ClickOnce	
客户端主界面（*XXX.YYYY*.Windows.Main.dll）	√		拷贝到 ClientLibrary 目录	
界面公用控件、第三方控件	√		拷贝到 ClientLibrary 目录	
界面组件（*XXX.YYYY.AAAA*.Window.dll）	√		拷贝到 ClientLibrary 目录	
界面公共组件（*XXX.YYYY.AAAA*.Windows.Common.dll）	√		拷贝到 ClientLibrary 目录	
业务逻辑组件（*XXX.YYYY.AAAA*.Business.dll）	√	√	拷贝到 ClientLibrary 目录	拷贝到 ServiceLibrary 目录
业务规则组件（*XXX.YYYY.AAAA*.Rule.dll）	√	√	拷贝到 ClientLibrary 目录	拷贝到 ServiceLibrary 目录
业务公共组件（*XXX.YYYY.AAAA*.Common.dll）	√	√	拷贝到 ClientLibrary 目录	拷贝到 ServiceLibrary 目录

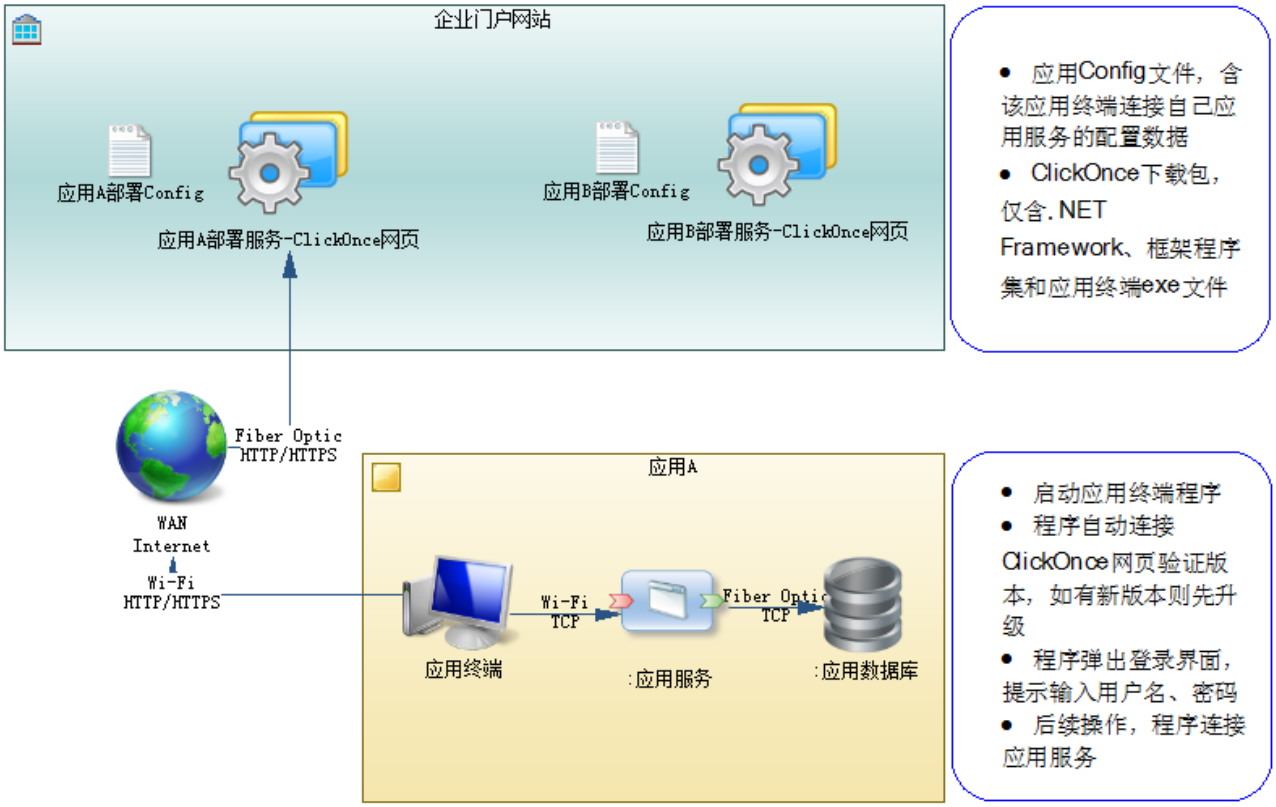
注 1：指 Host 所在目录下的 ClientLibrary 子目录，当客户端通过 LogOnDialog 登录服务端时，会自动将该目录里的文件下载到本地，并覆盖客户端程序所在目录下的旧文件来升级终端。

注 2：指 Host 所在目录下的 ServiceLibrary 子目录，用于 Host 里自带的一键升级服务功能。也可以自行关闭 Host 服务程序，直接将新文件拷贝到 host 所在目录下覆盖旧文件来升级服务。

1.3.5.4 部署结构

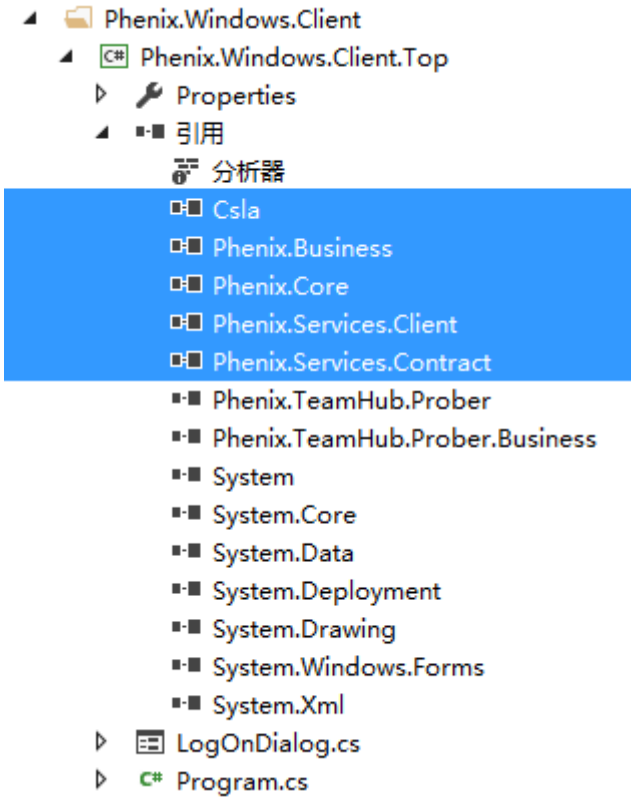

系统入口（校验用户、
升级、登入主窗体）
XXX.YYYY.Client.exe
主窗体
Plugin
XXX.YYYY.Windows.Main
界面组件（根据主功能点构建，技术处理上分为展现层和工作层）：输入输出数据、界面操作流程、触发校验约束条件（权限、规约）、提示信息
Plugin
XXX.YYYY.AAAA.Window
界面公共组件
XXX.YYYY.AAAA.Windows.Common
Phenix.Windows
DevExpress.TTTT
业务逻辑组件（包含属性、过程、规则，技术处理上分为逻辑层和持久层）：增删改查（持久化）数据、计算算法、业务流程、校验约束条件（权限、规约）。。。
跨物理层部署
业务规则组件（类型、状态、校验算法）
跨物理层部署
业务公共组件
跨物理层部署
Phenix.Services.Client
框架接口
跨物理层部署
业务组件基类
（扩展CSLA）
跨物理层部署
框架核心
跨物理层部署
XXX.YYYY.AAAA.Common
XXX.YYYY.AAAA.Rule
XXX.YYYY.AAAA.Business
Phenix.Services.Contract
Phenix.Business.dll
Phenix.Core
CSLA
物理层分界：客户端 - 服务端
Remoting
HTTP/TCP
Plugin
Phenix.Services.Host.exe
Phenix.Services.Library
ADO.NET
物理层分界：服务端 - 数据库
Database

1.3.5.5 发布方案、升级方案

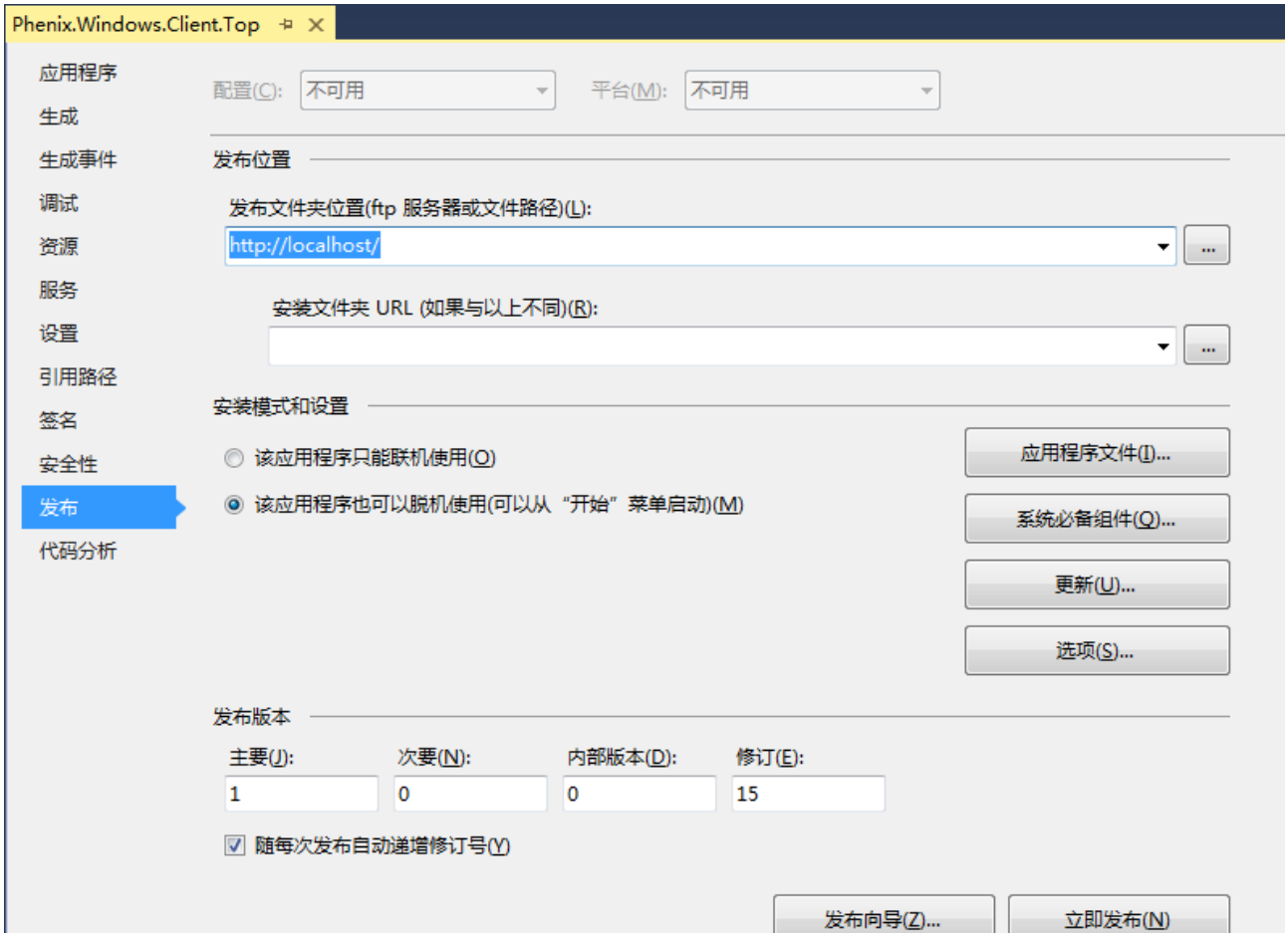


具体可参考 Phenix.Windows.Client 和 Phenix.Windows.Main 工程的设计方法，是如何实现系统发布的。

Phenix.Windows.Client 引用了（也是被 ClickOnce 打包发布）的最精简的程序集：

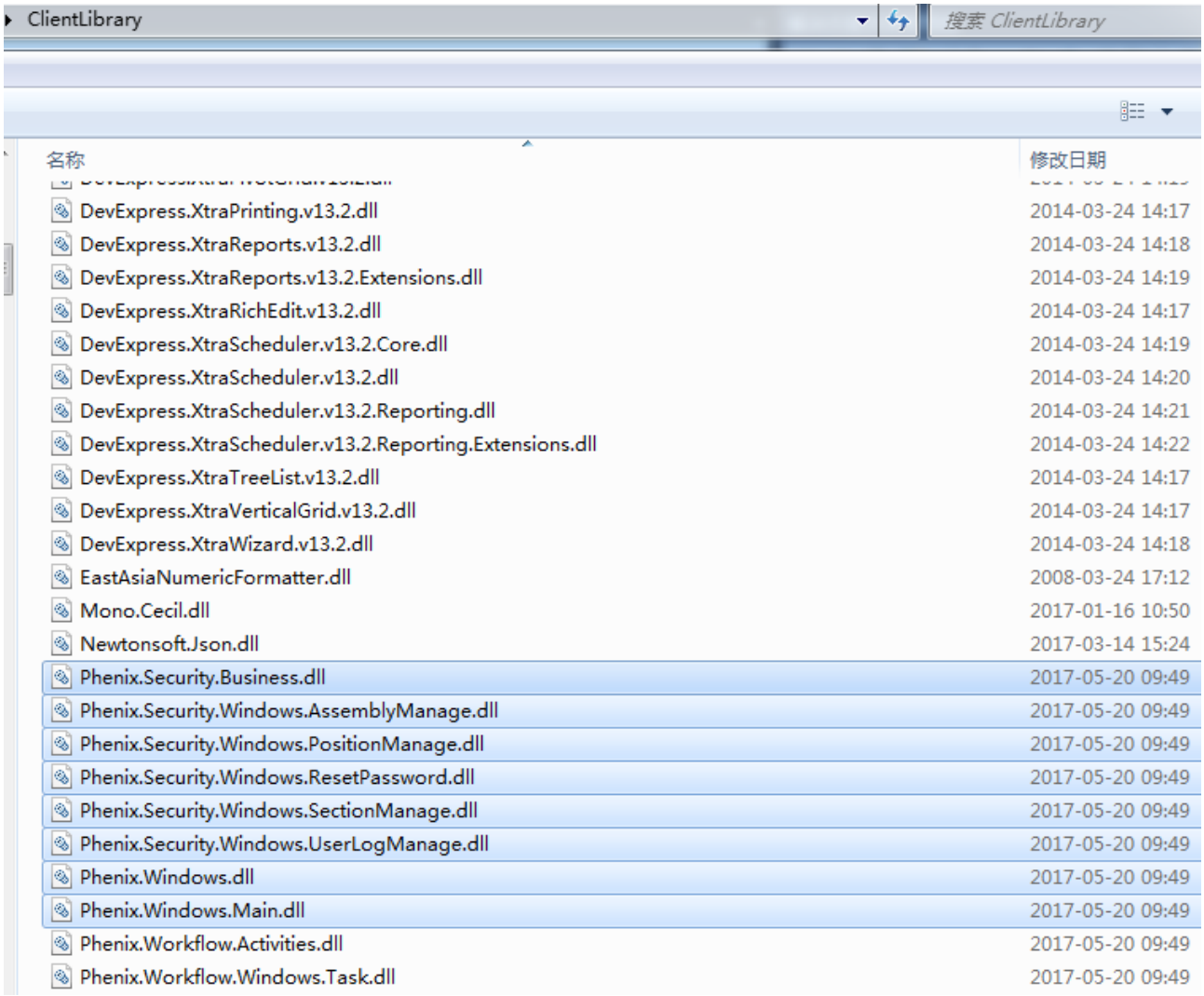


Phenix.Windows.Client 的发布设置界面：

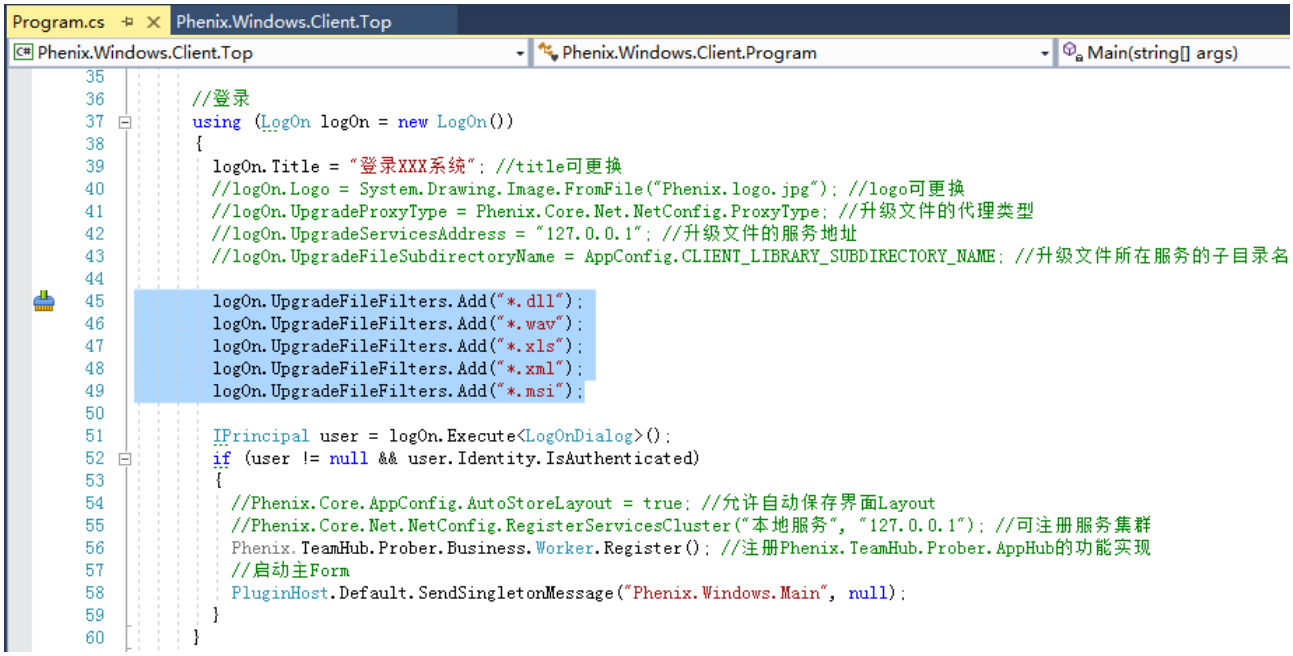


只要能通过 ClickOnce（这些程序集如有版本升级需重新打包发布）下载安装到本地，启动并弹出 LogOnDialog 界面。

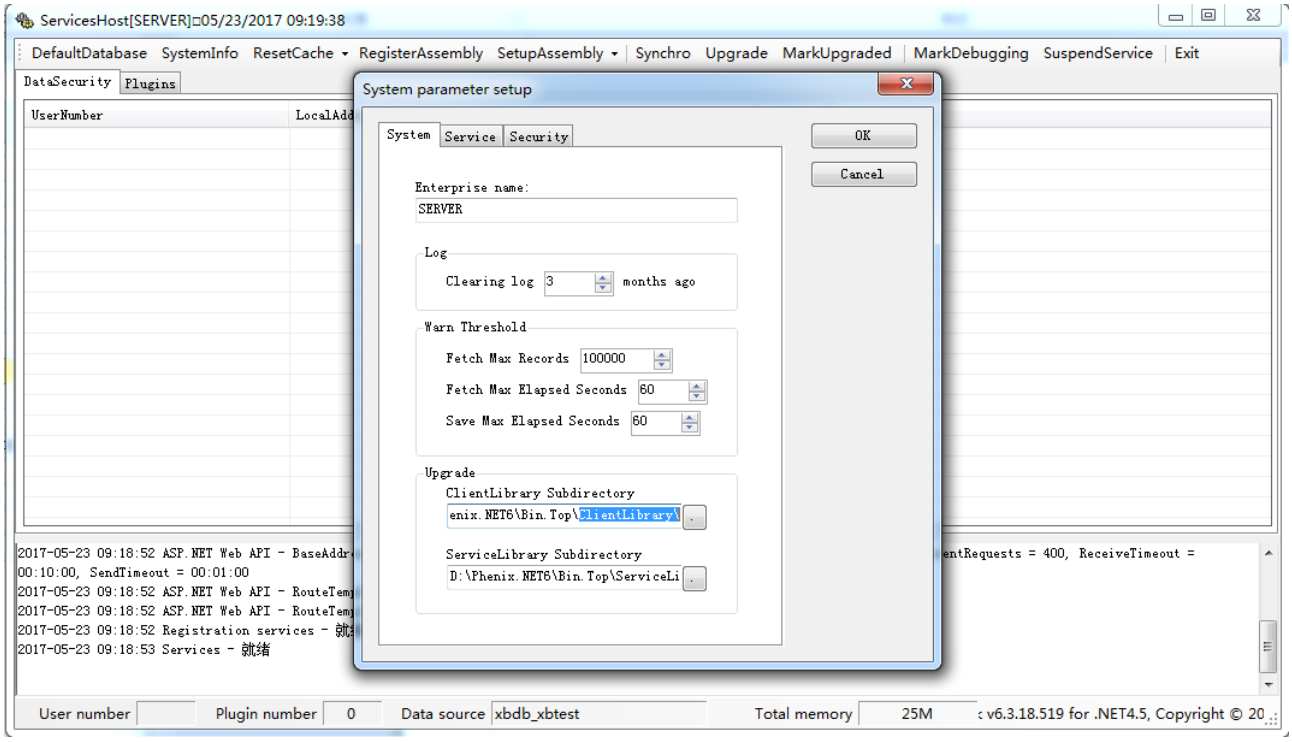
剩下需部署到本地的程序集，拷贝到 Host 所在目录下的 ClientLibrary 子目录下（当登录界面验证用户后会被自动（判断如有文件大小不同或文件日期更加新的话）下载到本地）：



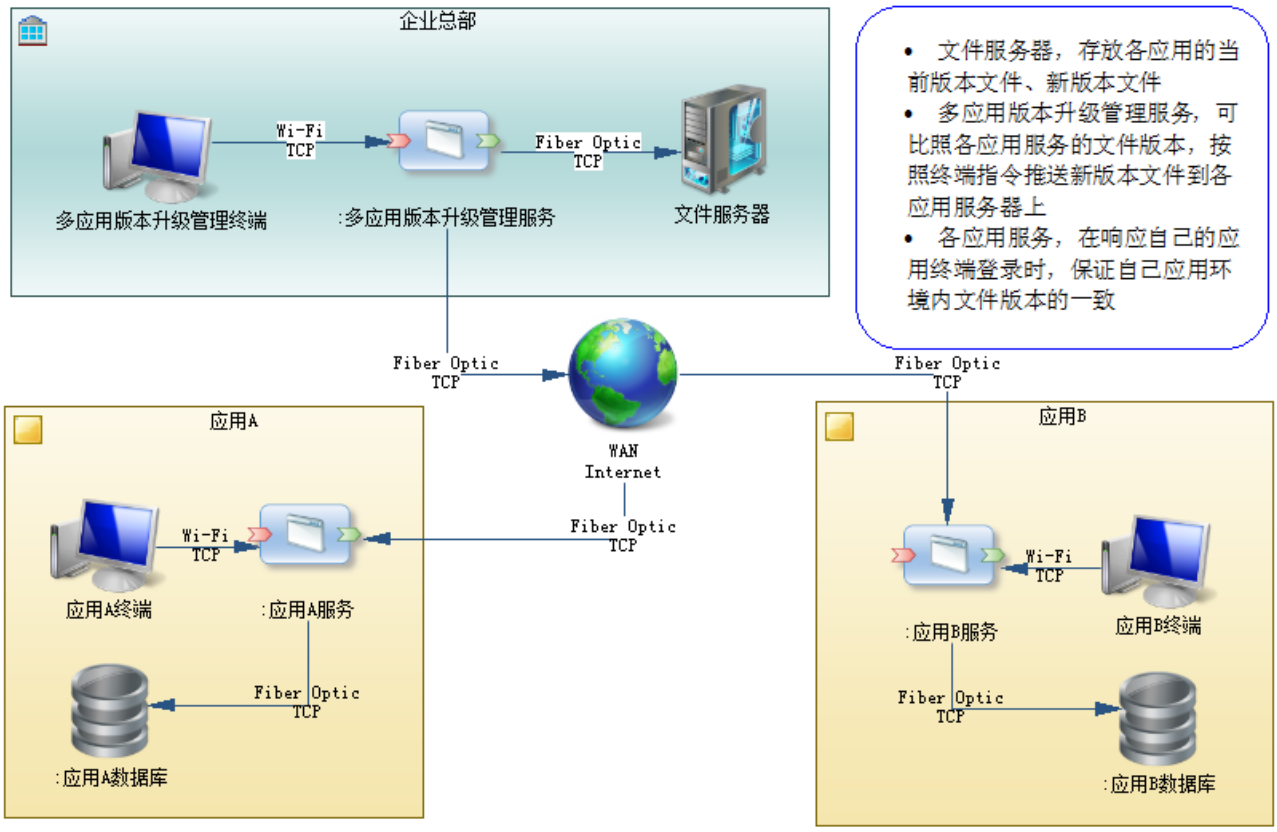
也可指定下载其他类型的文件到本地：



在 Host 的 System parameter setup 界面上，我们可以设置 ClientLibrary 的路径，比如升级文件获取自统一指定的网络文件路径：



此外，也可以自行设计一套多应用版本升级管理服务，以下是可供参考的某方案：



1.4 设计目标

1.4.1 降低项目的研发成本

项目研发成本的控制有很多方面，和业务层面一样，在技术层面上也可以提炼出许多通用的技术和方法，封装在框架中、或者整理到案例文档中，提供给每个项目组使用。在这样一个较高层次上来进行项目的研发，可以加快开发进度、缩短开发周期，从而减低研发成本。

1.4.2 规范研发团队的开发过程，提升项目质量，降低项目维护成本

由于本框架封装了大部分的技术实现，项目研发人员基本上仅需要编写业务逻辑部分的代码，而且只有通过框架所提供的一系列编程接口才能将应用系统运转起来，从而在代码层面上规范了研发人员的行为。同样，Phenix丶所提供的一系列设计方案，间接对系统分析、系统设计等软件研发过程中的各式文档编写和交流方式进行了引导，可逐步培养起研发人员一致的开发习惯和语境，进而使得研发团队内部的交流变得更加通畅，最终提升项目的整体质量。

1.4.3 保持技术的先进性

如果您仔细研究过 CSLA 的话，就会对此很有感慨，CSLA 在设计之初，微软的 WCF、Silverlight 等等技术都还没面世，但是，CLSA 一方面时刻紧跟技术潮流，一方面对研发人员的接口基本不做任何变动，能做到将基于其上的应用无缝升级到最新的.NET4 版本。这深层的道理，就是框架提供给研发人员的开发接口，一定是不包含任何特定的技术，而那些新技术、新方法都是要被框架封装起来的，这样，就能让基于其上开发的应用系统具备长久的生命力，不会被新技术抛在后面（要知道微软的数据引擎可是 3 年一换，而所谓的 WCF 技术，到底什么时候会被新推出的技术所替代，这是谁都说不清楚的事情）。

1.5 预备知识

如上所述，Phenix丶采用了许多技术和设计理念。要利用好这个工具，真正能为应用系统的项目服务，带来快速的开发效率、规范的开发过程和稳定的代码质量，设计、开发人员必须具备如下素质：

预备知识	推荐学习材料
领域驱动设计	《领域驱动设计.软件核心复杂性应对之道》
面向对象分析与设计	《面向对象分析与设计》、《UML 和模式应用》
对象-关系库映射技术(ORM)	了解基本原理、会用
CSLA 框架	《C#企业应用开发艺术.CSLA.NET 框架开发实战》
智能客户端(SmartClient)	《Windows Forms 2.0 数据绑定：.NET 智能客户端数据应用程序设计》
分布式架构（WCF、Remoting、WebAPI）	了解基本原理
MVC 架构、AJAX 技术	了解基本原理、会用

1.6 成功案例

Phenix丶与 PACKER for Delphi（2002 年至今一直作为核心组件运行在各个集装箱码头等客户的大中型营运管理系统上）及更久远的 Fordize for Delphi（2000 年-2002 年）一样，都是作者基于相同目的研发的企业级快速开发平台。

Phenix丶自 07 年正式推出第一版以来，已经历了六个大版本的升级，并成功应用、历练于多个项目和产品，主要有：

项目或产品	代码行级别	工程数级别	是否分片区（集团化总部+分部架构）
XX 滚装船码头营运管理系统		300	是
XX 整车物流链营运管理系统		200	是
XX 整车分拨库营运管理系统		100	是
XX 堆场仓储营运管理系统产品化软件系列		300	是
XX 服装 ERP 管理系统	>20w		

1.7 免责声明

为快速提升 Phenix丶的普适性、强壮性，得到更多行业软件的历练，特意放出供社会使用。为此，还请务必遵守 Phenix.license.txt 版权声明。

对使用到 Phenix丶的应用系统，因其存在的缺陷而造成的任何问题，承担相应的技术责任，随时整改、升级发布。

对使用到 Phenix丶的应用系统，自身在其系统架构设计（数据建模设计、负载和压力处理等）、代码编写、第三方控件的使用等各方面，自身设计、使用不当或考虑不周，而造成的系统效能低下等等问题，并由此而造成的后果，本 Phenix丶不承担任何责任或连带责任。