目录

[重头开始学Asp.net core 2](#_Toc28338561)

[迭代1 框架选择和类库管理 2](#_Toc28338562)

[迭代2 项目背景和领域模型 2](#_Toc28338563)

[迭代3 EFcore 和ORM 2](#_Toc28338564)

[迭代4 2](#_Toc28338565)

# 重头开始学Asp.net core

这么多年一直从事 Winform 开发，一直没有时间好好学学 web 开发。感觉自己就像从石器时代走来的古代类人猿。由于工作的调整，现在终于有时间学习一下 Web 开发。出于对技术和框架的熟悉和继承，决定还是学习微软的 Web 开发框架（虽然我一直认为java 是一种比C# 更优秀的语言，社区的活力远高于 C#，想想 eclipse 还是算了吧）。微软的 Web 开发框架从 ASP，ASP.NET，ASP.NET MVC 一直到现在的 ASP.NET Core一路走来，坏消息是没有一个熟悉的，好消息是 ASP.NET Core 是完全开源的，这对学习有很大的帮助。<https://github.com/aspnet/AspNetCore> 学习技术框架最好的方法是分析源码，但这种方法显然不适合我这种对 Web 开发一窍不通的人，对于我来说，最好的方式是通过项目开发尽快的熟悉 ASP.NET Core 。所以我决定尝试通过开发一个虚拟的项目来熟悉ASP.NET Core，同时因为要从 winform 转到 web 开发可以预料到各种问题会接踵而至。我决定重开博客，用一系列文章记录其中曲折的过程以及问题和思考，希望这一系列文章能够对需要重 winform 转型到 web 开发的朋友能够带来一些启发。关于这个虚拟项目的背景我会在下一篇介绍、首先我们要了解一下 ASP.NET Core。

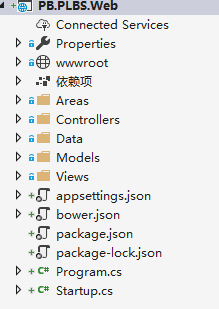
关于 ASP.NET Core 的介绍，网上资料铺天盖地，我就不啰嗦了。在这里主要谈谈从一个 winform 程序员的视角看 ASP.NET Core 的几点感受。

首先用 VS 创建一个 ASP.NET Core 项目，在项目模板选项中因为计算机环境的原因，暂时不选择启用 Docker 支持,Docker 支持在后期可以很方便的添加，当然前提是你搞定了 Docker 的安装和配置，Docker 是个好东西，在你的应用需要跨平台部署时。当然，如果你决定使用 IIS 托管，并且部署环境很固定的话，是不需要 Docker 的。关于 Docker 的安装和配置，网上资料很多，步骤也不是很繁琐，有兴趣的朋友可以百度一下。身份认证选择“个人用户账户”。



当然，在真实的项目中，你可能不太会直接用 新建 的方式创建一个项目，并把所有代码包含在一个项目中。更好的方式是创建一个解决方案，然后创建不同的项目对应程序不同的层。更常用的方式是在解决方案根目录下创建文件夹，把项目放在src文件夹，把测试项目放在 test 文件夹。在程序的分层架构方面 winform 或者说桌面程序和 Web 程序没有太大的区别。两者的区别在于表示层，桌面程序你需要自己设计 UI 程序 ，而 Web 程序你不需要自己设计 UI 程序（感谢各种浏览器）,你需要在表示层设计UI的内容，在运行时通过各种框架技术，把他们推送到客户的浏览器中。当然早桌面程序中你也可以这么做，先开发一个通过的窗体作为容器，能够运行用某种文件描述的可编辑的内容，用于工业控制的组态软件通常这么做，我以前做的一个生产线控制系统也是这么做的。

好了，废话少说，我们先看看创创建的项目的文件结构



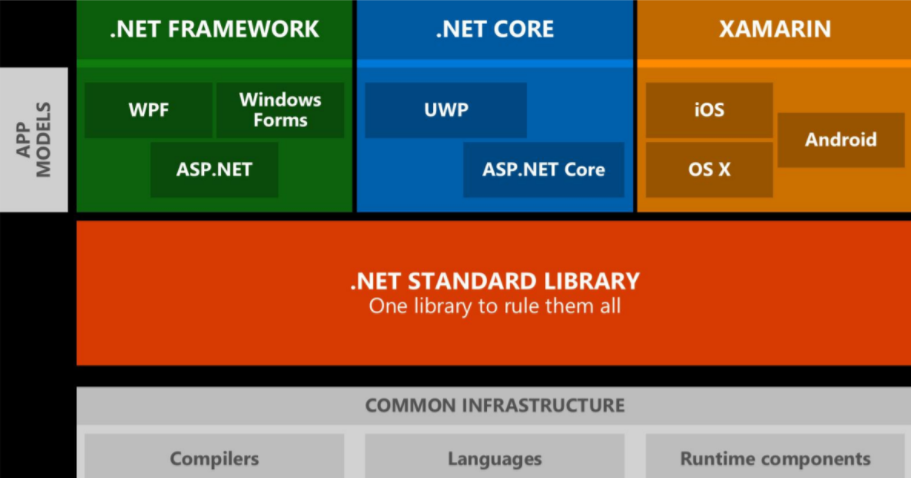
可以看出，程序的结果简单清晰，应该符合 Rails 约定，模型，视图和控制器的文件夹，到我们真正使用时再讨论，现在我们看看感兴趣的文件 – Program 和 Startup，看得出 ASP.NET Core 已经抛弃了 ASP.NET 把程序编译成 DLL ，运行时由 IIS 托管的方式。ASP.NET core 程序是由 Kestrel 托管的，在部署在 IIS 上时。IIS 仅起到 反向代理的作用,IIS 会把请求转发到 Kestrel ，为此IIS 必须添加 Asp Net core Module 的模块，以前在做一个用 ASP.Net Core API 宿主的 SIgnalR 程序时，因为对程序启动方式不了解，就碰到这个坑。让我们先看看 Program 的代码

# 迭代1 创建领域模型

在讨论完成领域DDD后，我们终于开始创建领域模型了。虽然这个项目的领域模型及其简单，但是作为虚拟项目，我们决定假装他复杂，这有我们就抛弃了使用行数据和表数据入口。而使用 O/R Mapping 作为数据访问层对数据的映射。因为使用 ASP.NT Core 数据映射框架就首选 EF Core,当然你要有兴趣，也可以使用 Hibernate。

一个项目的关键的能够设计一个好的领域模型，而设计好的的领域模型的一个必要条件就是和用户深入的接触，通过与用户的交流逐步了解和学习用于的领域模型。然后通过多次迭代，最终完成一个用户和设计人员都能充分理解和满意的领域模型(因为这是和虚拟项目，同时基于对混凝土行业的充分了解，完全可以模拟用户的灵魂大拷问，所以和用户交流的步骤就直接跳过)。在迭代的过程中，模型从简易到复杂，从臃肿到精炼，在这个过程中，我们需要对领域模型不断的进行场景走查式的测试（对于用户领域模型存储在用户的大脑和各种管理流程中，对于设计人员来说，领域模型可能是UML图或代码，它们都是设计人员对用于领域的理解和重构-类似解析主义，一万个读者有一万个哈姆雷特）。在迭代过程中，我们应该聚焦于对模型的创建，尽快的完成领域模型的原型产品。所以在构建领域模型时，我们应该尽量排除各种技术框架对领域模型的干扰，使用尽量简单和纯粹的技术创建，同时避免使用复杂的技术框架能方便我们对模型进行快速的测试。

在 ASP.NET Core 中，使用 .NET Standard 类库项目实现领域模型是个很好的选择。.NET Standard 是其他BCL必须准守的规范，而本身没有任何实现。这样用 .NET Standard 的类库可以在任何 .NET 平台上移植。



# 迭代2 项目背景和领域模型

让我们暂时告别一下 ASP.NET Core 先介绍一下这个虚拟项目。因为我的主要目的是通过一个项目，全面学习一下 ASP.NET Core，所以这个项目时一个很简单的，不具备实际应用价值的虚拟项目，但是项目会尽可能用到 Web 程序设计的常用技术。

熟悉商品混凝土生产的朋友都知道，在这个领域常用的软件有两种，一个是混凝土生产线的生产控制软件，主要负责监控生产线的运行和按照生产和工艺流程，自动执行生产任务。这个软件一般是厂家自己开发或委托外包开发的。另外一种是生产管理软件，就是混凝土行业中常说的 ERP（名字有点大）,一般是专业软件公司提供。ERP（原谅我使用这个名字）根据生产计划生成生产任务在通过接口（大部分是通过一个中间数据库）把任务推送（存储）到生产控制软件(中间数据库),生产任务软件接收（从中间数据库读取）到任务然后自动运行。

混凝土生产上有着多个存储仓，存储各种物料，而生产任务中包含着各种物料使用情况的配方。一个混凝土生产工厂，一般有多条生产线，而为了应对不同的生产工艺要求，不同的生产线有时会存储不同的物料。而ERP 在发送生产任务时，是靠人工来确定某个生产线是够存储需要的物料。这个虚拟项目就是在 ERP 的任务推送节后面增加一套管道机制，通过不同的管道过滤掉不适合的生产线，最后在通过各个生产线的等待生产的任务队列，自动负载平衡，选择最适合的生产线。

这套管道机制应该方便的支持增加和删除管道，达到任务调度的灵活配置。这样我们就可以把新的任务过滤逻辑通过类似中间件的技术添加到整个管道中。

这个虚拟项目的主要是为了学习 ASP.NET Core，索引领域模型设计的尽量简单，但是为了保证项目与实际的尽量贴近，也值得我们好好思考一下领域模型。

说到 领域模型就不得不提到分层架构，站在桌面程序员的视角和Web 方面，分层架构没有大的区别，尤其现在微软在 ASP.NET Core提供大量的技术框架支撑分层架构(MVC,EF Core,依赖注入等)。在这个项目里，我们也毫不例外的使用DDD的三层架构。在表示层，因为这个项目设想中要想管道一样可以快速的接入到搅拌站ERP和生产控制之间，所以表示层被设计为时髦的两个部分组成：管理界面和用于接入的服务API。在数据层，主要提供从另两个方面获取数据源：数据库和其他服务的API ，在部署时要根据ERP和生产控制软件的数据发布方式进行选择和配置，当然选择和配置服务层要用到依赖注入。最后是用于封装业务逻辑的领域模型,业务逻辑是个挺操蛋的词(Martin Fowler 语，至少意思差不多). 使用DDD方式时，领域模型和数据层的映射是个麻烦事，这个留在数据访问层在讨论。

尽管业务逻辑是个很操蛋的词，但是不可否认领域模型作为整个软件和核心价值和灵魂值得我们认真去思考。做个项目的朋友都有体会，如果数据映射（ORM）是个技术问题的话，领域模型的设计更像一个哲学问题。 我们经常会碰到下面的问题：

领域模型设计的不合理，这种软件很难真正嵌入到客户的管理流程或价值链中，用户在使用这种软件是就像骑着阿凡提的毛驴或者开着绝地求生里被打爆轮胎的蹦蹦车一样，虽然有载具了，但是它用于不会按你想要的方向行驶。

# 迭代3 EFcore 和ORM

# 迭代4