SD473362

Revit 开发在 BIM 项目中的演进及与人工智能的结合

梁裕卿
同济大学建筑设计研究院（集团）有限公司
上海建筑数字建造工程技术研究中心

学习目标

- 如何搭建 Revit 插件产品架构
- 开发中常用组件的选择方案
- 在 BIM 项目中如何与设计师相互沟通配合，迭代产品
- Revit 二次开发与人工智能算法的结合

描述

本课程将分享上海建筑数字中心在搭建 Revit 插件产品的经验，包括完整的需求、软件架构、开发、第三方程序集的使用、测试及打包过程。然后介绍 Revit 开发在 BIM 项目中是如何逐步迭代：包括与设计师的沟通、业务逻辑与软件性能的权衡、用户体验、使用反馈，产品迭代的全过程。最后，将介绍 Revit 开发与人工智能算法相互结合来实现在 Revit 中的合规检查：包括需求的提出、算法的分析、与 Google TensorFlow 的对接。通过软件合理的架构及高效的沟通方式并运用人工智能算法，Revit 插件产品在 BIM 项目中帮助设计师节省了时间，提升了工作效率。

讲师

梁裕卿，同济大学建筑设计研究院（集团）有限公司上海建筑数字建造工程技术研究中心开发负责人，Autodesk Expert Elite 成员。2013 年本科毕业于华北电力大学（保定）通信工程专业，2016 年硕士毕业于天津理工大学信息与通信工程专业（光子信息处理方向）。同年 4 月加入上海比程信息技术有限公司担任软件开发工程师。2018 年 5 月加入同济大学建筑设计研究院（集团）有限公司上海建筑数字建造工程技术研究中心 BIM 技术研究所建筑组担任开发工程师，主管软件开发工作。曾参与多项 Revit 二次开发、BIM 管理平台、移动端的开发工作并承担多项部门科研课题的相关开发工作。目前开发的基于 Revit 平台的同济设计 BIM 三维设计工具集已在数字中心部门全面推广使用，其它业务产品正在紧锣密鼓的测试之中。日常管理若干 Revit 二次开发学习 QQ 群，拥有 4000 多 BIM 技术开发爱好者的参与并运营个人微信公众号“BIMCoder 梁老师”。

如何搭建 Revit 插件产品架构

对于日常的一般需求或者是初入 Revit 二次开发的爱好者，我们会在 Visual Studio 中创建一个工程进行开发，编译之后在 ProgramData 路径下创建 Addin 文件来完成一个简单的插件。对于企业产品来说，上述的方式距离真正的构建产品还有很多的路要走，本章内容会详细介绍 Revit 插件产品是如何搭建的，包括产品打包的全过程、软件该如何架构、对于多产品又该如何做到高效使用已积累的函数库。希望能带给大家更多的思考与启发 :)

打包的全过程

一个插件产品的打包过程不仅仅只包含编译部分。其流程可以概括为以下步骤：将 cs 文件编译成 dll，然后对 dll 进行混淆，接着对 Addin 对应的 dll 进行数字签名，之后对 dll 进行打包，最后对打包出来的 msi 安装文件再次执行数字签名。我们直接跳过编译的部分，先说说代码混淆。

代码混淆

代码混淆是保护代码的一种重要方式，防止自己的代码被他人反编译。由于 C#的一些先天机制，我们很难说一定能做到百分之百的保护。如果需要保护代码，我们该如何选择合适的混淆工具呢？这里我不对任何安全产品进行商业广告，但可以给大家提供一个验证方法，检查自己代码是否能抵御一些反编译工具。在 Github 上有一款开源的反编译工具，名称为 de4dot，该工具能反编译绝大多数 C#代码。作者也很贴心的列举了可反编译的保护壳清单。如果您使用了清单中的混淆工具，那我十分推荐您下载 de4dot 去反编译您混淆后的程序集，检查下是否真正得到了保护。

目前已支持反编译的工具清单。

市面上还有一款开源混淆工具，名称为 confuserEx。该工具不在上述清单内，因为该工具得到了 de4dot 作者的技术指导。但是非常遗憾地是该工具也有对应的反编译工具能进行去壳，被称为 DeconfuserEx。所以您想更好的保护自己的成果，那么我还是建议您去购买一些商业混淆加密软件。可以参考一些市面上已有的 Revit 插件产品来选择合适的混淆工具，一般来说安全商都提供试用版，可以自己测试一下。需要提示的是混淆强度如果过高，插件可能无法载入或者直接报错，所以一定要实际测试。

Dll 数字签名

众所周知，Autodesk Revit 自 2017 版开始推行数字签名验证，没有经过数字签名的插件会在打开 Revit 的开始对话框提醒用户是载入该插件还是取消载入。数字签名证书需要向软件安全商进行购买，价格从 2000 到 5000 元人民币不等。当购买完数字签名后就需要对 dll 进行数字签名了。并不是所有编译后的 dll 都需要签名。只有启动 dll 需要，也就是 Addin 对应的 dll 文件。有了数字签名证书，我们就需要一款签名工具将签名打在 dll 上。微软给我们提供了一个便捷的签名工具：signtool。需要注意的是 signtool.exe 有很多依赖项，我们需要把 signtool.exe 及其依赖项放在一起才能正常执行。对于 Windows 10 系统来说，signtool 及其依赖项一般存放于 Windows Kits\10\bin***[version]***\x86，大家可以在自己的电脑中寻找。以下是 signtool 的完整清单：

- appxpackaging.dll
- appxsip.dll
- Microsoft.ComparePackage.Lib.dll
- Microsoft.Windows.Build.Appx.AppxPackaging.dll.manifest
- Microsoft.Windows.Build.Appx.AppxSip.dll.manifest
- Microsoft.Windows.Build.Appx.OpcServices.dll.manifest
- Microsoft.Windows.Build.Signing.mssign32.dll.manifest
- Microsoft.Windows.Build.Signing.wintrust.dll.manifest
- mssign32.dll
- opcservices.dll
- signtool.exe
- signtool.exe.manifest
- wintrust.dll
- wintrust.dll.ini

Signtool 签名工具清单。

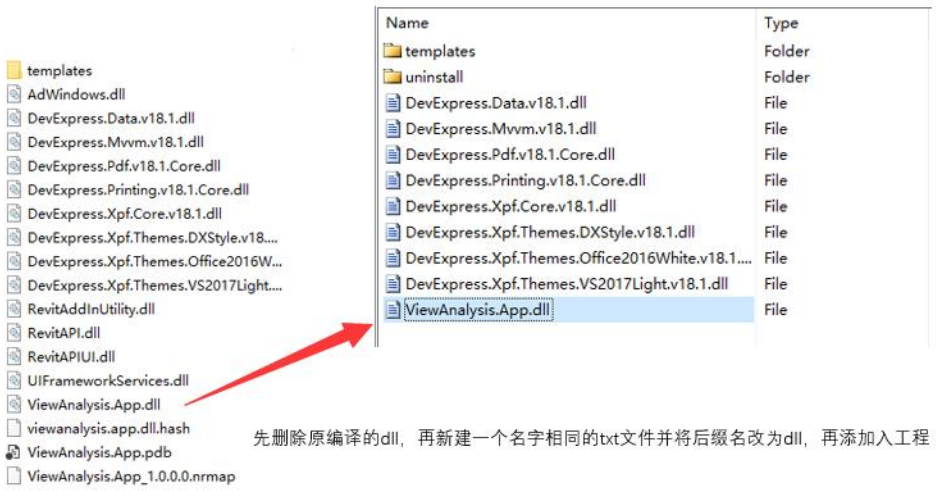
当我们准备好签名工具后，可以使用脚本对 dll 进行签名。在这里建议大家对 dll 进行双签名，即 SHA1 与 SHA256 签名。假定我们的数字签名证书称为 xxx.pfx，所需要签名的 dll 称为 App.dll。详细的操作脚本如下：

```
signtool.exe sign /f xxx.pfx /p password /t
http://timestamp.verisign.com/scripts/timestamp.dll /v app.dll

signtool.exe sign /f xxx.pfx /p password /fd sha256 /tr sign256url /v /as app.dll
```

打包产品

数字签名之后我们就可以制作安装包，打包产品了。对于打包工具，我建议两种：Microsoft Visual Studio Installer Project 与 Wix Toolset。前者可以在工具->扩展和更新中找到并安装，后者可以直接进入官网并下载。前者的优点在于有一个可视化界面，可以清晰地知道自己的安装包里有哪些内容，在何处设置注册表，在启动时需要执行什么脚本。而 Wix 没有可视化界面，完全依靠类似 XML 格式的方式手动操作需要打包的内容，准入门槛较高，刚使用 Wix 的朋友比较痛苦。但是 Wix 的通用便利性较强，如果有其它插件产品需要打包的话则可以迅速复制原来的 wxs 文件进行少量修改即可。而 VS Install 由于采用可视化界面，需要在属性栏中逐一修改，占位文件也必须重新添加（占位文件就是将原来的 dll 以 txt 文件修改后缀名添加入 VS Installer），过程要比 Wix 更加繁琐。所以，我个人还是非常推荐使用 Wix 进行打包。



VS Installer 添加占位文件示意图。

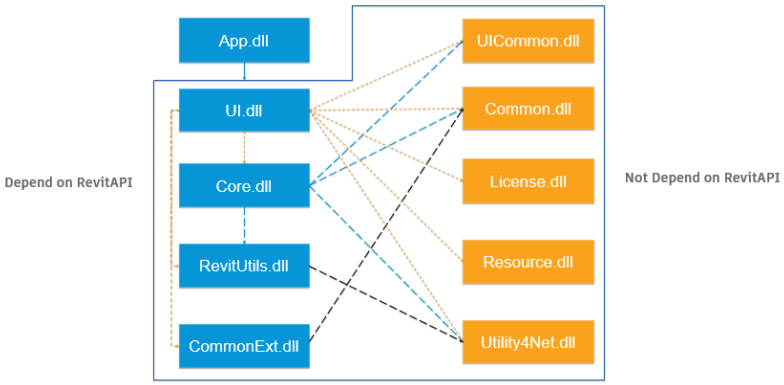
软件架构

一个好的产品就像一幢优秀的建筑一样，需要扎实的地基和合理的布局。对于代码来说，这就是软件架构。较好的软件架构不仅仅是便于作者自己总结经验，提升软件运行效率，其更能使团队的小伙伴能快速定位代码，提升团队协作效率。作为程序员来说，大家肯定不希望在一堆代码山里深挖寻找，而是希望像衣柜一样，能快速找到各季的衣服。从我个人的使用角度出发，提供一个参考方案。

1) App.csproj
2) UI.csproj
3) Core.csproj
4) RevitUtils.csproj
5) CommonExt.csproj
6) UICommon.csproj
7) Common.csproj
8) License.csproj
9) Resource.csproj
10) Utility4Net.csproj

首先说说 App 工程，该工程可以包含 Application 与 Commands 等等用于初始化 Revit 选项卡的代码。UI 工程顾名思义就是插件中各功能的界面，各功能的模态或者非模态 xaml 文件，Model 以及 ViewModel 文件都可以写在 UI 工程中。一些界面通用的内容，比如进度条或者提示弹出框、Converter 可以写在 UICommon 工程中。Common 工程主要是一些数据结构的定义，该工程是不依赖于 Revit 的 dll，如果有些数据定义需要引用 Revit 的 dll，那么我建议可以放在 CommonExt 工程中。License 工程可以放一些关于授权的代码，post 或者 get 方式发送/接收服务器信息进行用户验证都可以写在这个工程中。Resource 工程不再赘述，主要用于多语言版本的扩展或者放置一些使用频率较高地图片文件。RevitUtils 工程很重要，它是对 Revitapi 的进一步封

装，比如一些 Filter，Solid，Face，Point 的运算，该工程为核心业务 Core 工程提供必要的方法。Utility4Net 工程是对 Windows api 封装或者常用 C#操作，比如获取界面的句柄，创建文件，读取文件等等。下图所示为整个插件代码包含的工程，蓝色部分需要依赖 Revit 的 dll，橙色部分则不需要。

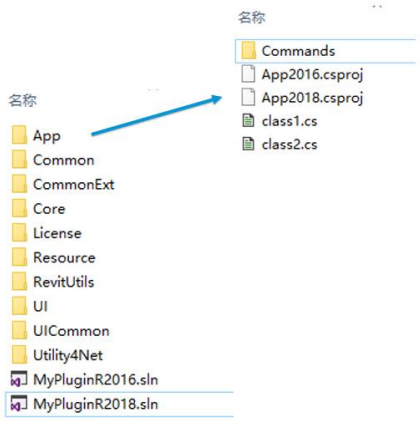


软件架构工程及依赖示意图。

说完软件架构部分，我们再聊聊软件文件的架构，也就是怎么组织自己的产品文件夹。我觉得可以分为以下几个部分：

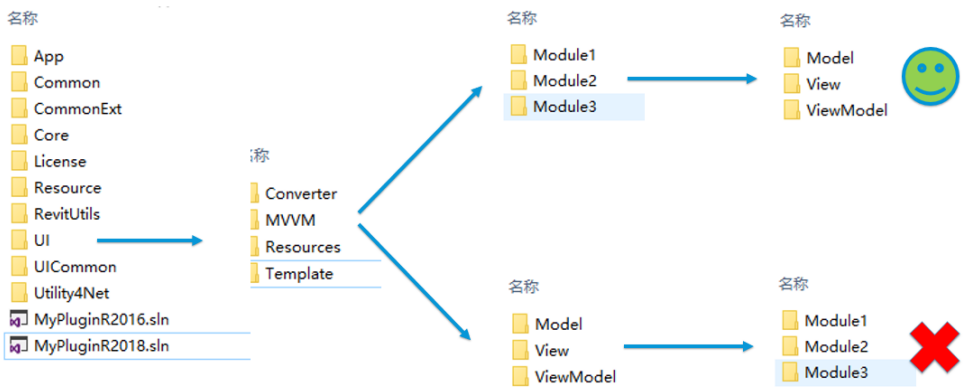
1) bin 文件夹，所有代码编译输出的位置，方便进行打包。
2) data 文件夹，主要放置一些族文件，config 文件。
3) doc 文件夹，主要是一些需求文档或者会议记录的文件。
4) install 文件夹，不仅包括安装包工程文件，还包括签名工具，开启动画的应用程序，卸载程序，混淆工具。所有为安装包工程服务的内容都可以放置在此。
5) source 文件夹，代码工程。
6) thirdparty 文件夹，专门放置第三方依赖程序集。

一般来说，一个插件产品需要支持多个版本的 Revit，那么该如何做多版本呢。首先有一个原则，那就是能复用的尽量复用，在代码中可以使用编译条件做多版本处理。其次在 source 文件夹架构上可以参考以下方式：



多版本文件架构示意图。

如图所示，可以在最外层创建对应版本的解决方案文件，如果某工程依赖于 Revit 就创建对应的项目文件。再举一个例子，关于 UI 工程该如何合理安排文件夹，如下图所示：

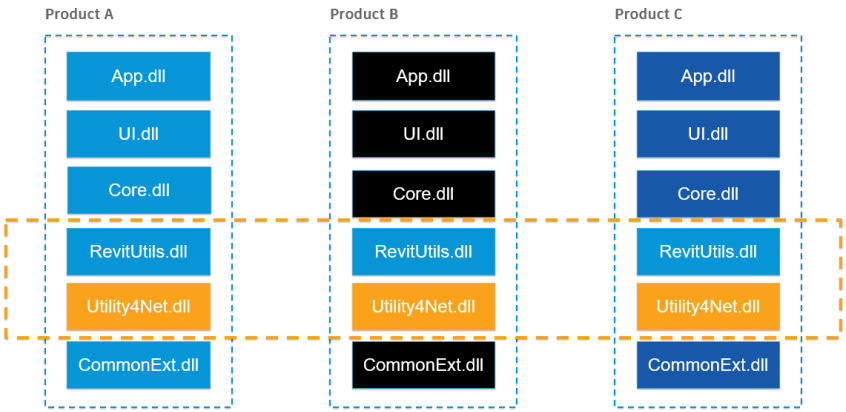


UI 工程文件夹示意图。

从上图我们可以看到两种方案，第一种在 MVVM 文件中创建各功能模块文件夹，然后在模块文件夹内创建三个对应文件夹。第二种方案，创建三个文件夹，在每个文件夹中再创建各功能模块文件夹。两者的区别在哪儿呢？其实一个插件产品可能有很多功能按钮，如果按第一种的方式能方便程序员修改对应模块的所有 MVVM 文件。而第二种在多模块的情况下非常不易于程序员快速寻找，大家可以试试如果 MVVM 文件夹下各创建几十个模块文件是否方便自己修改。

多产品架构

在上述内容中有一个很重要的工程称为 RevitUtils，我们可以将平时积累对于 RevitAPI 封装代码放入其中。如果你正在开发多个产品而不想每次都创建差不多的 RevitAPI 封装代码，那么复用这个工程是理想的解决办法。



多产品架构示意图。

如果你在 A 产品使用了 RevitUtils.dll 与 Utility4Net.dll，而之后的 B,C 产品由于需要一些调整，所以你修改了 RevitUtils 或者 Utility4Net 的代码。那么在加载入 Revit 时，两个产品就会发生加载冲突。这时候我们需要合并模块。合并模块可以使用上述的 VS Studio 或者 Wix 制作，它的作用就是能让若干产品能共享某些依赖。对于合并模块， 以下有几个建议供大家参考：

1) 指定 Merge Module 的安装位置为公共位置，类似于 C:\ProgramData
2) 使用 IExternalDBApplication 的实现对 MergeModule 中的 dll 进行加载
3) Merge Module 中的 dll 所对应的.addin 文件的 Addin Type 也应该是 DBApplication，而不是 Application
4) 保证 Merge Module 中的 dll 所对应的.addin 文件要先于产品插件首先载入到 Revit 中
5) 对于 Merge Module 中的 dll 中的方法只能添加，但不能删减或修改，不然旧版本插件会出错。

```
[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)]
public class MyMSM : IExternalDBApplication
{
    public ExternalDBApplicationResult OnStartup(ControlledApplication application)
    {
        var currentDirectory = Path.GetDirectoryName(typeof(MyMSM).Assembly.Location);
        var directoryInfo = new DirectoryInfo(currentDirectory);
        foreach (FileInfo fileInfo in directoryInfo.GetFiles("*.dll"))
        {
            var loadFilePath = fileInfo.FullName.ToLower();
            if (loadFilePath.EndsWith("revitapi.dll") || loadFilePath.EndsWith("revitapiui.dll"))
                continue;

            try
            {
                Assembly assembly = Assembly.LoadFrom(loadFilePath);
                application.WriteJournalComment("Loaded " + assembly.Location, true);
            }
            catch (Exception ex)
            {
                application.WriteJournalComment("Cannot load " + loadFilePath, true);
                application.WriteJournalComment(ex.ToString(), true);
            }
        }
        return ExternalDBApplicationResult.Succeeded;
    }
    public ExternalDBApplicationResult OnShutdown(ControlledApplication application)
    {
        return ExternalDBApplicationResult.Succeeded;
    }
}
```

合并模块预加载代码。

开发中常用组件的选择方案

本节将介绍一些在开发中常用的第三方组件。包括界面部分，数据库连接，日志，Office 文件操作。

UI

1. DevExpress(控件+MVVM)，我一直用 DevExpress，比较好用而且 UI 控件比较多。
2. Telerik，用的不是太多，类似于 DevExpress。
3. Prism，MVVM 是其强项。
4. MVVMLight，轻量级的 MVVM 框架，做一些小插件比较适合。
5. HandyControl，国人开发的优秀 UI 控件库，在 WPF 界知名度很高，风格偏互联网。

Log
这块毋庸置疑，绝对使用 Log4Net。

数据库
对于插件端我比较推荐 SQLite，基本满足日常需求。

ORM
Dapper，其实它不算完全的 ORM，也是要写点 SQL 语句的，但并不妨碍它的便捷。

Hook
我比较推荐 Github 上开源的一个库，名称为 Gma.System.MouseKeyHook.dll，Github 地址是：https://github.com/gmamaladze/globalmousekeyhook。

Office
我推荐 NPOI，不需要安装 Office 即可用，比微软提供的 Office 组件好用。

在 BIM 项目中如何与设计师相互沟通配合，迭代产品

一个优秀的 BIM 插件产品离不开团队的配合。仅靠开发人员或设计人员是无法实现的。那么在实际 BIM 项目中开发人员该如何根据设计人员的要求来开发呢？当需求如潮水般扑过来时，开发人员是否每个功能都要接受？遇到某些特殊设计情况该如何在代码中处理？

BIM 项目中的开发
本章将介绍在 BIM 项目中的软件开发流程，开发建议，并配以实例加以说明。

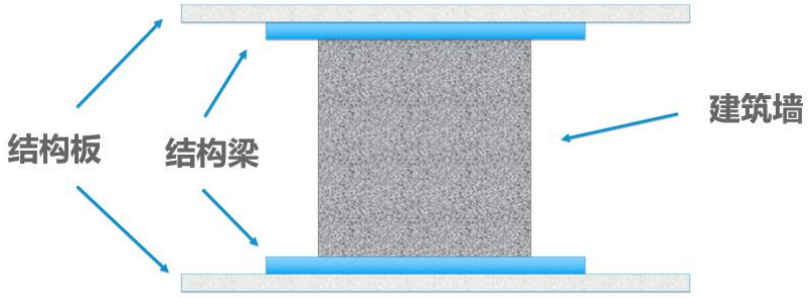
软件开发流程
1）由设计团队提出需求，产品经理统一收集并评估该需求是否为有效需求。
2）产品经理对需求梳理初步逻辑与界面，与产品顾问交流得出结论
3）由产品顾问将需求告知开发人员
4）开发人员评估该功能 RevitAPI 是否支持并报告给产品顾问
5）若支持，产品顾问向开发人员下发任务
6）开发人员进行开发，并发布产品给设计团队使用，由设计团队进行意见反馈给开发人员，逐步迭代。

BIM 项目开发建议
1）产品的人机交互界面美观、简单；切勿追求炫酷，操作花里胡哨。
2）遇到批量操作的需求时建议服从大局，切勿为某些特殊情况在循环体内做特殊处理，避免插件运行时造成不必要的错误，拉长执行时间。

3）在对应的 Revit 版本中加载对应版本的族，不建议在高版本中载入低版本族文件，因为有些族在自动升级时可能带有错误。
4）当有链接文件参与的功能时一定要弄清楚该功能需要各链接文件中的哪一类构件。
5）当需要存储一些数据在 Revit 中时，可以按以下顺序优先考虑：config 文件、Revit 自带外部存储、Excel、SQLite 数据库。
6）程序不是万能的，不建议一开始追求极致智能化，反而欲速不达，可以步步为营。

实例

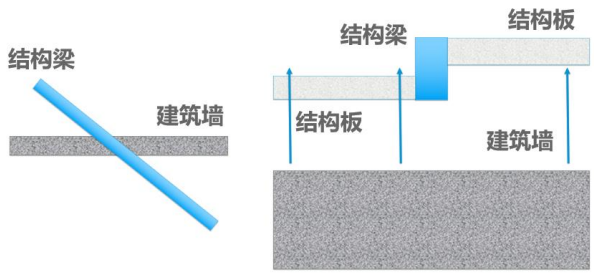
这里我们来介绍一个在 BIM 项目中应用的功能，（建筑）墙贴（结构）梁板。在建筑模型中链接结构模型，然后调整建筑墙的高度来紧贴上下链接结构梁板。大致示意图如下所示：



（建筑）墙贴（结构）梁板示意图

本功能的主要关键点是修改墙的上下偏移属性。我们来逐一分析这个需求：
1. 应该提供界面给用户让用户选择是哪个链接模型，因为一个项目可能链接多个模型。
2. 然后遍历建筑墙，通过 BBX 来筛选上下结构梁板。
3. 先找板，通过墙的 LocationCurve 各点使用 Project 映射到上下方的板。
4. 然后再找找有没有结构梁，方法同找板。
5. 计算上下差值，调整建筑墙的两个属性 BuiltInParameter.WALL_TOP_OFFSET 与 BuiltInParameter.WALL_BASE_OFFSET 最终实现功能。

在这个功能中可能会碰到以下情况：



特殊情况

图片左侧部分结构梁的 LocationCurve 与建筑墙的 LocationCurve 方向并不一致，只是部分有交叠。图片右侧的两块板高度并不一致，而下面只有一堵建筑墙。对于这种特殊情况，大家可能想做的完美一点，把建筑墙切割，然后分别调整。但我的建议是不考虑这些，还是按照之前的判断逻辑执行。我们可以单独做个小功能去处理这些特殊情况或者让设计人员手动处理。在一些大型模型中如果为了处理这些特殊情况而增加几何、事务、循环的操作会严重拖慢运行时间，甚至造成 Revit 长时间无效应。如果在没有保存 rvt 文件的情况下遇到 Crash，我相信你的设计同事或者客户是要和你拼命的。 其实在日常很多需求中都会遇到一些特殊情况，希望大家在编写时斟酌一下是否要增加代码来处理这些特殊情况。

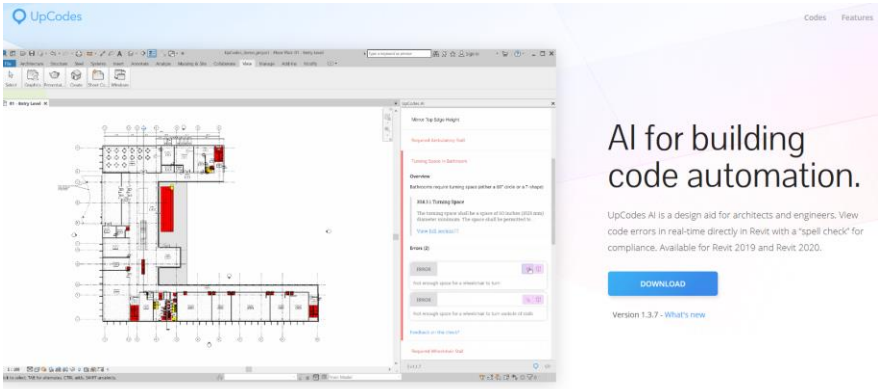
Revit 二次开发与人工智能算法的结合

目前人工智能技术已深入到生活，生产的方方面面。最新发布的 Revit2021 已集成了衍生式设计功能。本章将介绍 Revit 二次开发与人工智能算法的结合。

已发布的 Revit+AI 产品

UpCodes

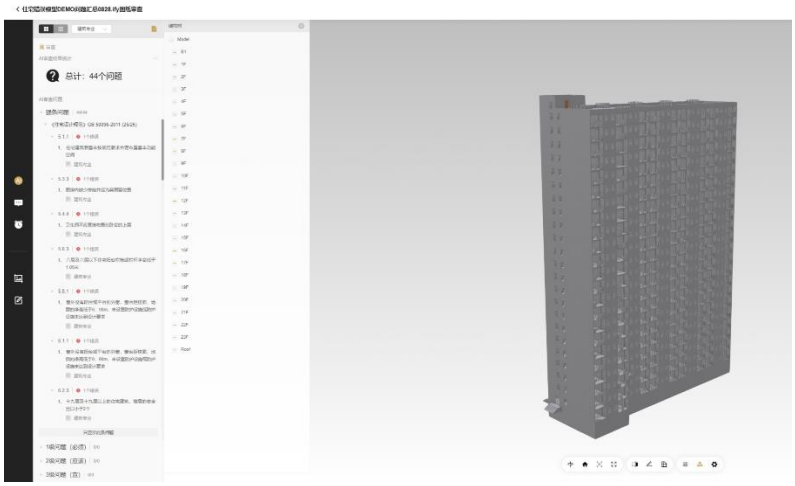
UpCodes.AI 是一款适用于美国建筑标准的合规检测软件。通过人工智能技术将规范转义成数据库，通过数据库提供的接口来检查在 Revit 设计中的问题。



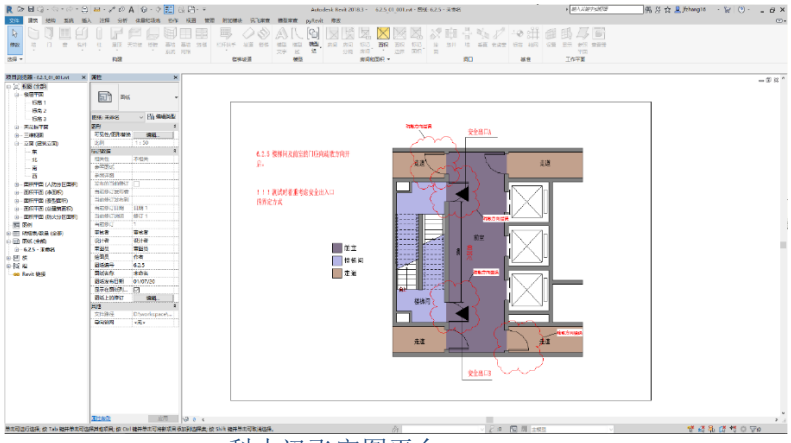
UpCodes。

科大讯飞审图平台

科大讯飞是我国建筑产业结合人工智能的先锋，目前科大讯飞正式对外发布了智慧审图平台，在 Revit 及 Web 端都可以使用该产品。



科大讯飞审图平台。



科大讯飞审图平台。

TJAD 自动规则检查（ARC） 解决方案

同济大学建筑设计研究院（集团）有限公司上海建筑数字建造工程技术研究中心在自动合规方向也做了若干研究，包括智能疏散，智能成图，智能校审，智能算量等多个方面。本章节主要介绍我们部门最新研发的基于深度强化学习的消防疏散路径绘制方案。

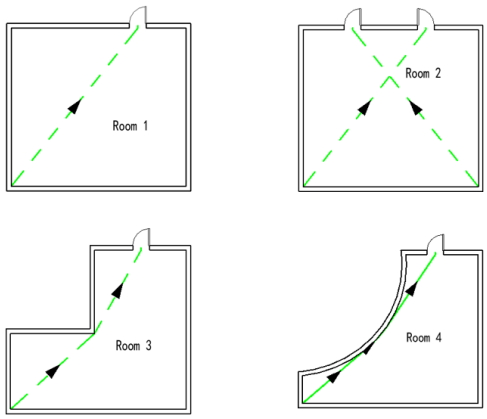
消防疏散

消防疏散路径绘制是 BIM 验证设计是否符合建筑消防规范的重要环节。BIM 建模人员需要花费大量时间在 Revit 平台中勾勒出房间和楼层的疏散路径及疏散方向。Autodesk 已于 Revit 2019 版发布了行进路径功能，但其有以下四点局限性：1）无法批量对房间进行路径绘制。2）只能在 Revit 2019 及以上版本使用。3）绘制效果不符合设计院日常习惯。

4）无法应对较复杂房间构型。为了突破上述的局限性并提高设计效率，我们开发了基于深度强化学习的消防疏散路径绘制方案。

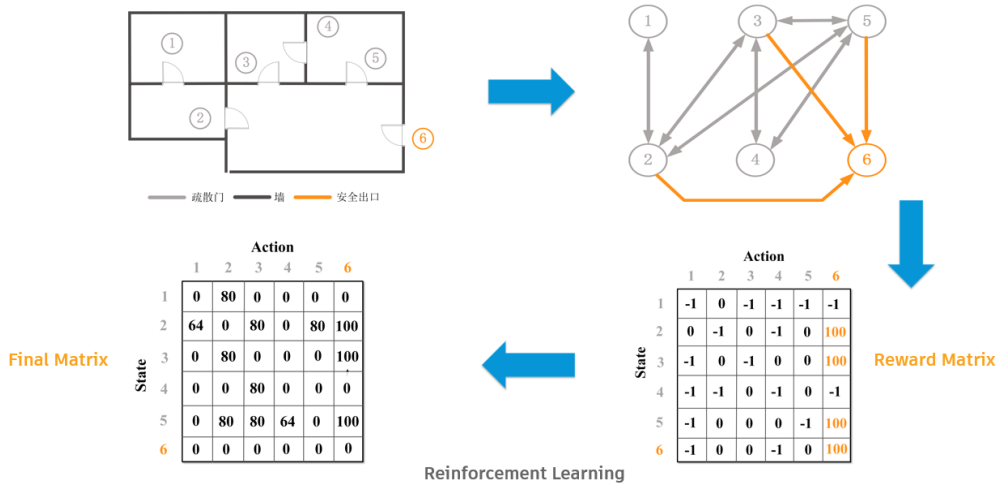
算法概述

对于各个房间来说，我们可以使用 A*算法计算房间内离疏散门最远点到疏散门的路径。对于弧形边界可以将其切分成若干直线段进行拟合。对于公共区域，比如走廊来说，也可以将其设为房间来做同类处理。



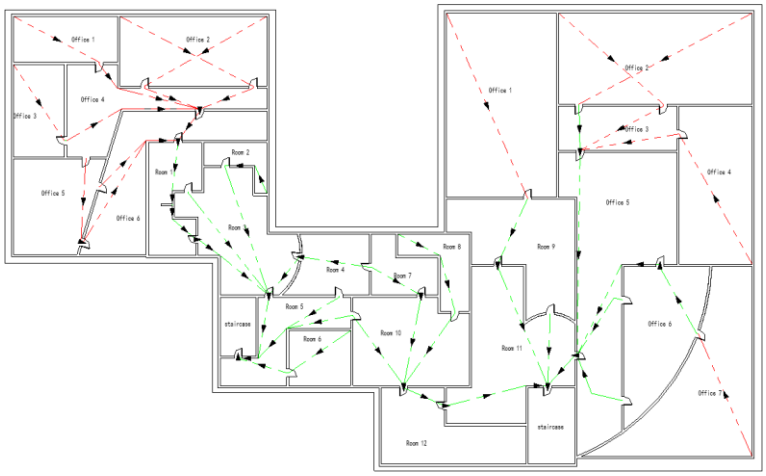
A* 算法处理房间疏散路径。

楼层具有一个或多个防火分区，每个防火分区拥有多个房间。对于每个防火分区来说，如何得到该区域内各门到安全出口的疏散路线呢？这个问题其实类似于走迷宫，那么该问题就转换成了门到门的路线选择问题。我们可以将模型中的门都进行排序编号，并构成节点关系图形成矩阵数据。如果两个疏散门之间能相互到达则赋值为 1，不能到达则记为 0，疏散门可以到达安全出口的赋值为 100，疏散门到自己本身设为 0，安全出口到达自己本身设为 100。我们构建完矩阵后就可以根据 Q-Learning 算法进行计算，当矩阵收敛时得到路径编号。有了路径编号再获取对应的门，通过 A*算法就能得到路径了。



强化学习处理疏散路径。

Q-learning 及其变种算法存在一个问题，就是不能处理大维度矩阵，一旦矩阵维度较大时则会计算较慢，这在 Revit 程序中是不能容忍的。所以，我们需要引入神经网络去帮助我们测算矩阵值， Deepmind 的深度 Q 学习是一个不错的选择。我们通过改进 DQN 算法，实现了快速得到平面楼层疏散路径，并且标注完行径箭头。对于超过疏散规范长度的路线显示红色，合规路线显示绿色。整个过程计算耗时是人工手绘的 1/3 到 1/2 时间，大大节省了时间，提高了效率。



DQN 完成疏散路径。

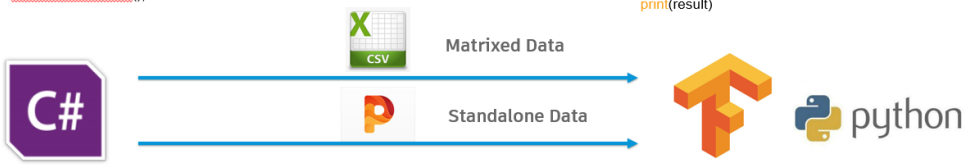
C#与 Tensorflow 对接

对于矩阵数据来说，可以先将 Revit 中的数据保存为 csv 格式。然后让 python 去读取 csv 文件路径。其余单个参数，可以通过 Process 直接传参。其实两种语言之间的对接还有其它方式，比如使用 IronPython，或者将 python 版的 Tensorflow 代码改写成.NET 版本，即 TensorflowSharp。但我个人觉得如果写习惯了 py 版本是完全没动力改成.NET 版本，即使语法没有差太多。

```
var result = string.Empty;
var pypath = Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location), "My.py");
var p = new ProcessStartInfo();
var sArguments = pypath;
sArguments += " "+param1+" "+ param2;
p.FileName = @"your_pythonexe_path\python.exe";
p.Arguments = sArguments;
p.UseShellExecute = false;
p.RedirectStandardError = true;
p.RedirectStandardOutput = true;
p.CreateNoWindow = true;
using (Process process = Process.Start(p))
{
    using (StreamReader reader = process.StandardOutput)
    {
        string stderr = process.StandardError.ReadToEnd();
        result = reader.ReadToEnd();
    }
}
```

```
# -*- coding:utf-8  -*-
...
...
if __name__ == "__main__":

    dir = sys.argv[1]
    path = os.path.join(dir,'Mydata.csv')
    r = np.genfromtxt(open(path,"rb"),delimiter=",")[:,:-1]
    final = int(sys.argv[2])
    g_network = DeepNet(dir,r,final)
    result = g_network.get(r,final)
    print(result)
```



C# and Tensorflow。