基于OCC的可交互式CAD建模工具设计与开发-初稿

卢鑫波

指导教师：王小刚

[第一章 绪论 3](#_Toc132573732)

[1.1 毕业设计背景和意义 3](#_Toc132573733)

[1.1.1 CAD软件本身的意义 3](#_Toc132573734)

[1.1.2 作为毕业设计的意义 3](#_Toc132573735)

[1.2 毕业设计任务和目标 4](#_Toc132573736)

[1.3 毕业设计方法和步骤 4](#_Toc132573737)

[1.3.1 设计方法的选择 4](#_Toc132573738)

[1.3.2 设计步骤。 5](#_Toc132573739)

[1.4 毕业设计论文结构和内容安排 5](#_Toc132573740)

[第二章 需求分析 5](#_Toc132573741)

[2.1 系统需求和功能分析 6](#_Toc132573742)

[2.1.1 功能需求 6](#_Toc132573743)

[2.2 用户需求和场景分析 6](#_Toc132573744)

[2.2.1 需求 6](#_Toc132573745)

[2.2.2 需求复核 7](#_Toc132573746)

[2.3 系统的非功能需求和性能指标 8](#_Toc132573747)

[第三章 系统设计 8](#_Toc132573748)

[3.1 系统架构和设计原则 8](#_Toc132573749)

[3.1.1 系统架构 8](#_Toc132573750)

[a. 功能模块层 9](#_Toc132573751)

[b. 业务逻辑层 10](#_Toc132573752)

[c. 应用层 10](#_Toc132573753)

[d. 数据层设计 11](#_Toc132573754)

[3.2 设计原则 11](#_Toc132573755)

[3.3 系统功能模块和接口设计 13](#_Toc132573756)

[3.3.1 功能模块 13](#_Toc132573757)

[3.3.2 模块接口与关系 14](#_Toc132573758)

[a. MainApplication 14](#_Toc132573759)

[b. RPObjectManager 14](#_Toc132573760)

[c. ProcessManager 15](#_Toc132573761)

[d. MainWindow 15](#_Toc132573762)

[e. docView 15](#_Toc132573763)

[f. DataLabel View 16](#_Toc132573764)

[g. DataLabel 16](#_Toc132573765)

[h. DataDriver 16](#_Toc132573766)

[3.4 数据结构设计 18](#_Toc132573767)

[3.4.1 DataLabel 18](#_Toc132573768)

[3.5 主要业务功能 19](#_Toc132573769)

[3.5.1 NewDocument 19](#_Toc132573770)

[第四章 系统实现 19](#_Toc132573771)

[4.1 技术选型和系统实现 19](#_Toc132573772)

[4.1.1 技术选型 19](#_Toc132573773)

[a. 主要编程语言：python 19](#_Toc132573774)

[b. 3d建模工具：opencascade 20](#_Toc132573775)

[c. 可视化工具：qt 20](#_Toc132573776)

[4.2 功能实现和代码实现 20](#_Toc132573777)

[4.2.1 DataLabel 实现 20](#_Toc132573778)

[4.2.2 LabelView 24](#_Toc132573779)

[4.2.3 MainWinodw 25](#_Toc132573780)

[4.3 系统的安全性和可靠性保证 26](#_Toc132573781)

[4.3.1 安全性 26](#_Toc132573782)

[4.3.2 可靠性保证 26](#_Toc132573783)

[a. 日志系统 26](#_Toc132573784)

[b. 错误处理和恢复： 26](#_Toc132573785)

[c. 容错设计： 26](#_Toc132573786)

[第五章 系统测试与优化 26](#_Toc132573787)

[5.1 系统测试方法和步骤 26](#_Toc132573788)

[5.2 功能测试 27](#_Toc132573789)

[a. 测试功能与范围 27](#_Toc132573790)

[第六章 参考文献 27](#_Toc132573791)

[a. Chat.Openai.com/chat 27](#_Toc132573792)

[第七章 附录 27](#_Toc132573793)

[7.1 系统使用说明和用户手册 27](#_Toc132573794)

[7.2 系统源代码和测试数据 27](#_Toc132573795)

* 1. 绪论
     1. 毕业设计背景和意义
        1. CAD软件本身的意义

CAD是计算机辅助设计（Computer-Aided Design）的缩写，是一种通过计算机软件辅助进行设计、绘图和模拟的技术。在现代设计和制造领域中扮演者至关重要的角色：

1. 提高设计效率和精度：

CAD软件可以帮助工程师和设计师更快地创建和修改设计图纸，并且可以实现精确的尺寸和几何约束。与手绘设计相比，CAD软件的设计效率和精度更高。

1. 支持3D建模和可视化

CAD软件可以进行3D建模和可视化，使设计人员可以更好地理解和展示设计方案。这有助于减少错误和提高设计效率。

1. 支持多种设计领域

CAD软件可以应用于各种设计领域，包括机械、建筑、电子、航空航天等领域。这为设计人员提供了更广泛的选择，使得CAD软件可以满足不同行业和领域的需求。

1. 支持数据共享和协作

CAD软件可以使设计团队之间共享设计数据，从而促进协作和沟通。这有助于提高设计质量和效率，减少错误和重复工作。

1. 降低生产成本和提供质量

CAD软件可以帮助制造商更快地制造和生产产品，从而减少生产成本。同时，CAD软件可以确保产品符合设计要求，从而提高产品质量

综上所述，CAD软件在现代设计和制造中具有不可替代的作用，可以提高设计效率、降低生产成本、提高产品质量，并且有助于促进协作和沟通。

* + - 1. 作为毕业设计的意义

掌握深入技术知识：设计并实现CAD软件需要本人深入理解CAD软件的底层技术原理，这对于我来说是一次深入学习和探究CAD技术的机会，可以更好地理解CAD软件的工作原理，提高对CAD软件的应用技能。

增强实践能力：设计并实现CAD软件需要本人付出大量的实践努力，通过这一过程，本人可以增强自己的实践能力和解决问题的能力，这对于未来从事相关领域的工作和面对实际问题都具有积极的影响。

培养创新能力：设计并实现CAD软件需要本人具备创新能力，本人需要通过不断地尝试、改进和创新，设计出更加完善和实用的CAD软件，这可以帮助本人培养自己的创新能力和实践能力，为未来的职业发展打下坚实的基础。

提升综合能力：设计并实现CAD软件需要本人具备较高的综合能力，包括空间想象能力、编程能力、沟通能力等。通过这一过程，本人可以不断提升自己的综合能力，从而更好地应对实际的设计任务和挑战。

提高竞争力：设计并实现CAD软件是一项非常具有挑战性和创造性的任务，这对于本人未来的职业发展和就业竞争都具有重要意义，设计并实现CAD软件的经历可以增加本人在就业市场上的竞争力，提高就业机会。

综上所述，设计并实现CAD软件对于本人具有重要意义，可以帮助本人掌握深入技术知识、增强实践能力、培养创新能力、提升综合能力和提高竞争力，这对于本人的职业发展和个人成长都具有重要的促进作用。

* + 1. 毕业设计任务和目标

任务：实现一个以参数建模为主要内容的cad软件。

目标：

1. 基于文档的用户编辑数据存储与读取。
2. 基于文档编辑的undo，redo功能。
3. 对于建模参数的建模与其进行可视化提示与显示。
4. 基于用户需求实现相应功能。
   * 1. 毕业设计方法和步骤
        1. 设计方法的选择

常用的软件设计方法包括：

1. 结构化设计方法：该方法采用模块化设计思想，将系统分为多个模块，每个模块的功能和接口都要明确定义，以便于实现、测试和维护。在设计过程中，使用流程图、数据流图等工具，通过自顶向下的设计思路进行系统分解和模块设计。
2. 面向对象设计方法：该方法基于面向对象的设计思想，将系统分解为对象，并建立对象之间的关系和交互。通过类的定义和对象的实例化，实现系统功能和实体之间的映射，采用继承、封装和多态等机制，提高软件的重用性、可维护性和可扩展性。
3. 服务导向设计方法：该方法基于服务导向架构（SOA）思想，将系统分解为服务，每个服务对应一个独立的功能单元，可以独立开发、部署和维护。在设计过程中，采用服务接口定义语言（WSDL）和服务描述语言（UDDI）等工具，实现服务的发现、注册、调用和管理。
4. 原型设计方法：该方法采用快速原型开发技术，通过迭代的方式，快速构建原型，进行功能验证和用户反馈，最终形成可靠的系统设计方案。在设计过程中，使用工具如Axure、Balsamiq等，进行交互设计、可视化设计和原型演示。
5. 面向服务设计方法：该方法是在面向对象设计基础上，进一步强调服务的重要性。将系统中的功能划分为不同的服务，通过服务的协同作用实现整体的系统功能。在设计过程中，使用工具如Service-Oriented Modeling Framework（SOMF）等，进行服务的分析和设计。

主要ICONIX的设计方法，以数据记录为中心（设立数据记录类，不同的业务记录被需要的信息，提取各自的所需信息）。

由于是第一次接触设计软件的开发，对于设计软件的文档编辑显示等功能都不是很熟悉，所以快速原型设计方法是毕业设计采用的主要开发方法。通过不断的尝试积累知识，对于软体主题或者部分功能采用不同的设计方法设计实现方法，并修正设计，在实践中接近目标。但由于快速原型是一个可以不断循环的设计过程，不适合在文档中描述。

显然，上述的各种设计方法都有其存在的意义，效率、易学习等特性。但是本人能力不足，且是独自开发软件，没有与其他开发人员协作的要求。所以，在毕业设计过程，我会结构化开发方法，将软体分为多个子模块。并根据子模块的功能特性，选择相应的开发方法。而各个模块之间，根据其功能特性抽象出接口，通过实现数据转换类，帮助各个模块间进行数据通信，从而实现各个模块之间的协同通信。

* + - 1. 设计步骤。

1. 需求分析
2. 根据需求功能，确定输入模块。
3. 根据需求功能，确定数据模块。（软体需要保存哪些数据）
4. 根据需求功能，补充信息显示对象。
5. 根据模块功能，确定模块对象及其数据输入、输出接口。
6. 根据需求功能，检查并完善模块对象功能完备性。
7. 建立模块之间的关系，设计数据转换类。
8. 以面向对象方法，分解模块对象的组成，建立子对象之间的关系。
9. 实现对象。

整体方向，从大的服务提供者分解为各种小的服务提供者，从基础服务到丰富服务，从主体功能框架到子功能实现的设计方向。不断分解任务，将复杂功能分解成能集体实现复杂功能的小功能模块。

* + 1. 毕业设计论文结构和内容安排

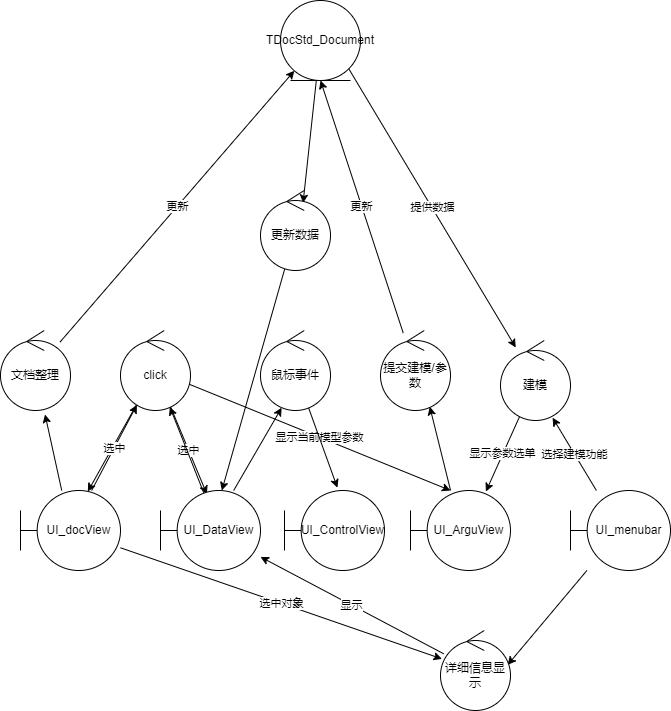
1. 绪论：介绍毕业设计的背景、意义和目的，概述研究内容和研究方法，以及论文的结构和内容安排。
2. 需求分析：介绍毕业设计的系统需求分析，包括功能需求、性能需求、可靠性需求、安全需求等方面的内容。
3. 系统设计：介绍毕业设计的系统设计，包括系统结构设计、模块设计、数据结构设计、算法设计等方面的内容。
4. 系统实现：介绍毕业设计的系统实现，包括系统开发环境、开发工具、开发语言、实现过程、测试过程等方面的内容。
5. 系统测试与评价：介绍毕业设计的系统测试与评价，包括测试方案、测试用例设计、测试过程、测试结果分析、评价指标等方面的内容。
6. 总结与展望：对毕业设计进行总结，评价研究成果，指出研究的不足和存在的问题，并对未来的研究方向和发展趋势进行展望。
   1. 需求分析
      1. 系统需求和功能分析
         1. 功能需求
7. 建模和设计功能：CAD软件需要具备建模和设计功能，可以让用户创建、编辑和修改3D模型或2D图形，支持各种类型的几何图形、曲面、体积等。
8. 数据管理和交互功能：CAD软件需要支持数据的管理和交互功能，包括与主流CAD文件格式的数据交换、版本控制等
9. 显示和渲染功能\
10. 界面和易用性功能：CAD软件需要具备良好的界面和易用性功能，使用户可以快速掌握软件操作，并能够自由地进行设计和编辑操作。
11. 安全和稳定性功能：CAD软件需要具备安全和稳定性功能，可以保护用户的数据安全，防止软件崩溃或数据丢失。
    * 1. 用户需求和场景分析
         1. 需求

3D模型建模过程的操作流程：

1. 创建新的工程文件：在CAD软件中创建一个新的工程文件，选择所需的单位和比例。
2. 绘制基本图形：使用CAD软件中的基本图形绘制工具（如线、圆、矩形等）创建模型的基本形状，例如创建一个立方体可以通过创建一个矩形，然后拉伸成立方体形状。
3. 对基本图形进行操作：使用CAD软件中的变换操作工具（如移动、旋转、缩放等）对基本图形进行操作，以实现所需的形状。
4. 创建辅助图形：根据需要，使用CAD软件中的辅助图形绘制工具（如线、圆、矩形等）创建辅助形状，以辅助完成所需的形状。
5. 组合基本图形和辅助图形：使用CAD软件中的组合操作工具（如合并、交集、差集等）将基本图形和辅助图形组合在一起，以创建所需的形状。
6. 添加细节：使用CAD软件中的绘制工具（如多边形、曲线、文字等）添加细节和标记，以完善模型的细节。
7. 进行分析和检查：使用CAD软件中的分析工具（如体积、重心、质量等）进行分析和检查，以确保模型符合要求。
8. 导出和保存：最后，将模型导出为所需的文件格式，并保存工程文件以备日后修改和使用。

总之，3D模型建模需要使用CAD软件的绘制、变换、组合、分析等多种工具和操作，以实现所需的模型形状和细节，最终导出和保存模型文件。

* + - 1. 需求复核



数据模块: TDocStd\_Document 与 dict\_DisplayedShape(保存可视化对象和对应在TDocStd\_Document中的位置)，TdataXtd\_table(按类型分类参数数据表)。

Document: 包含可视化操作工具(基本变换球， 坐标系)和TDocStd\_Document的显示和管理。

可视化界面: 可视化工具中TDocStd\_Document的Topods\_shape显示。

可视化信息: 显示当前可视化界面 对应的三维控件平面(可调整)，鼠标当前所在点坐标等。

参数建模: 显示建模所需参数，让用户从document中获取，或者从可视化信息中获取。

功能菜单: 点击相应建模函数，在参数建模界面进行可视化显示。

提示信息模块

* + 1. 系统的非功能需求和性能指标

1. 可靠性：CAD软件必须具有高度的可靠性，能够稳定运行，避免崩溃和数据损失。
2. 安全性：CAD软件必须具有良好的安全性，保护用户数据的机密性、完整性和可用性。
3. 易用性：CAD软件必须具有良好的易用性，操作简单，界面友好，用户能够快速上手并熟练使用。
4. 可维护性：CAD软件必须具有良好的可维护性，能够方便地进行升级、修复和扩展。
5. 性能：CAD软件的性能指标包括速度、响应时间、稳定性、并发处理能力等，要求高效、稳定、流畅。
6. 兼容性：CAD软件必须具有良好的兼容性，能够与其他软件和硬件设备无缝集成和协作。
7. 可扩展性：CAD软件必须具有良好的可扩展性，能够满足不同用户和行业的需求，支持定制化开发和个性化配置。

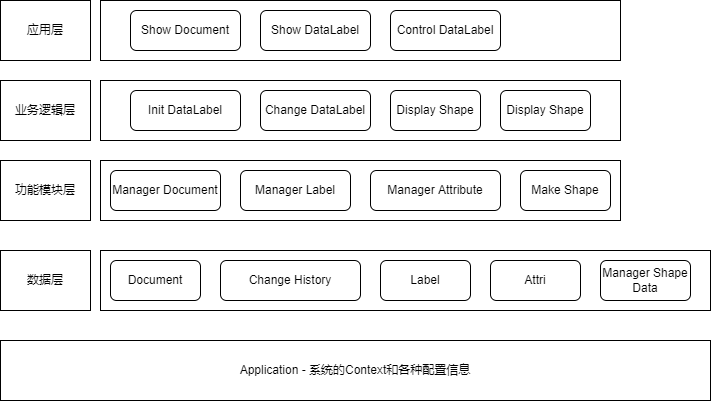
总之，CAD软件的系统的非功能需求和性能指标是确保软件质量和用户满意度的重要因素，需要综合考虑软件的功能、性能、易用性、可靠性、安全性、兼容性、可扩展性等多个方面，以满足用户的需求和期望。

* 1. 系统设计
     1. 系统架构和设计原则

我们将目标软件命名为[RedPanda]。

* + - 1. 系统架构

架构功能需求分析，我们首先从用户需求，确定系统中存在哪些主要业务。从业务需求确定所需的功能模块，确定业务与功能模块之间的交互，通过功能模块确定我们需要哪些数据模块。



* + - * 1. 功能模块层

从业务需求中，总结出我们需要的业务逻辑模块和业务。业务通过调用不同的业务模块以实现功能。

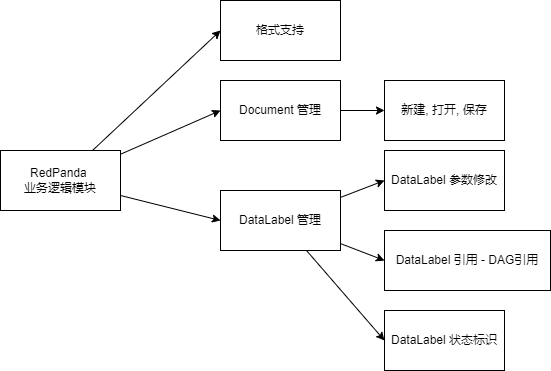
3D模型建模过程的操作流程：

1. 创建新的工程文件：  
   我们需要用对于[工程文件-Document]的管理
2. 绘制基本图形：  
   我们需要在[Document]存储图形节点[DataLabel]，提供对于图形节点的管理功能。  
   我们需要提供创建基本的图形的操作，根据此参数定义，创建不同的图形。  
   我们需要提供图形参数的定义。
3. 对基本图形进行操作，创建辅助图形，组合基本图形和辅助图形，添加细节：  
   我们提供对于图形的引用，从而可以对图形进行相应的进一步操作。
4. 进行分析和检查：

在[DataLabel]数据设置标识位，标识当前模型状态，根据对于模型参数等的，设置模型状态。

1. 导出和保存：最后，将模型导出为所需的文件格式，并保存工程文件以备日后修改和使用。

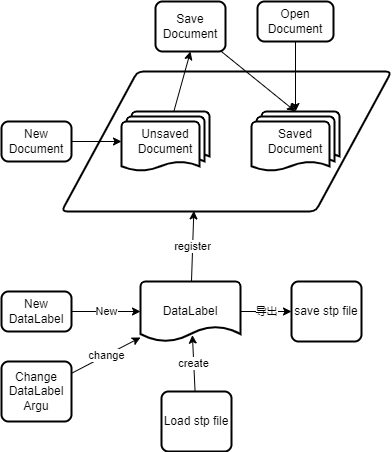
总之，我们主要对于用户的参数建模操作，提供可视化的输入ui，从而帮助快速实现建模，在此之外，我们还为用户提供文档视图等输入信息记录与显示功能。



* + - * 1. 业务逻辑层

从中，我们进一步归纳出，RedPanda中的主要业务：

1. New Document
2. Open Document
3. Save Document
4. New DataLabel
5. Change DataLabel argu
6. Undo，Redo
7. Load stp file
8. Save as stp



* + - * 1. 应用层

我们根据业务逻辑的模块来决定我们需要提供哪些ui控件，以帮助用户实现需求。通过判别哪些数据信息，有助于用户编辑【Document】，建立模型，我们就把那些数据提供给用户。

1. 首先，对于Document管理模块：  
   我们需要提供文档视图【DocView】，告诉用户当前有哪些以保存的图形数据。  
   我们需要提供【Document】的打开和保存，这些功能在[ui\_mainwindow]的[menubar]中实现即可。
2. DataLabel管理：  
   我们需要提供【ShapeView】，表现用户当前编辑的模型对象是何种形态。  
   我们需要提供【ArgumentView】，提供当前【DataLabel】的可编辑参数。
3. 数据导入导出: 在[ui\_mainwindow]的[menubar]中实现即可

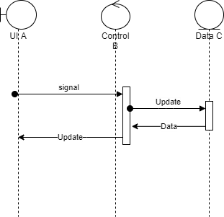
所以，我们有三个主要的ui页面，分别是【DocView】是【Document】的视图显示ui，【ShapeView】是【DataLabel】的显示UI，【Argument】是[DataLabel]的参数编辑UI。

* + - * 1. 数据层设计

从上，数据层需要【Document】，【DataLabel】两个数据结构两个与文档编辑相关的数据结构，【RDObject】用于保存程序执行过程中与【DataLabel】相关的数据信息。

* + 1. 设计原则

遵循的设计原则：

1. 跨模块的数据由主控模块,统一处理  
   当一个模块需要引起另外一个模块的数据更新,那么这个业务必须由主控模块A主导.原因:当子模块B、C可以自由通信,可以提高程序运行效率,但是这会让编程变得复杂:不好立即获取子模块B、C的产生的全部变化.  
   当A获取B信息更新信号,主动驱动B信息更新，我们就可以同时得知B的信息获得了更新，且立即可以获得B更新后的信息。
2. 用户输入是唯一的信息来源:  
   当且仅当用户输入时,系统的数据才会产生相应的变化.  
   系统内的信息流动是单向流动的, 用户user从UI输入数据,触发相应的信号signal，控制模块A启动相应的业务，更新内部数据，根据数据更新UI。  
   
3. 模块内部，动态对象与其衍生对象之间，需建立反查表

动态对象A常常需要更新数据，而其衍生对象B，常常是为了部分显示动态对象的某部分信息，需要我们在动态对象A更新数据，相应的更新衍生对象B。

例如，对于：

Class Shape {

}

Class DrawObject {

Public DrawObject(Shape)

}

当我们调用`B DrawObject(A)`时,我们需要进行如下插座, 向DrawObject的反查表中登记,DrawObject B:

A = Shape()

B = DrawObject(A)

DrawObject\_lookUP[A] = B

DrawObject\_lookUP[B] = A

1. 开放-封闭原则（OCP）

开放-封闭原则（OCP）是指软件实体（类、模块、函数等）应该对扩展开放，对修改封闭。也就是说，当需要对软件实体进行修改时，应该通过扩展它的功能来实现，而不是直接修改它的源代码。这样可以避免破坏原有的代码，提高代码的可维护性和可扩展性。

下面举一个例子来说明如何应用开放-封闭原则：

假设我们要设计一个图形库，其中包含三种图形：圆形、矩形和三角形。现在，我们需要为图形库添加一个新的图形：正方形。如果我们直接修改现有的代码，添加一个新的类来实现正方形，这就违反了开放-封闭原则，因为这样会破坏原有的代码，导致代码变得难以维护和扩展。

为了符合开放-封闭原则，我们可以通过扩展现有的代码来添加新的图形。具体来说，我们可以定义一个抽象的图形类，其中包含一个绘制方法，然后让每个具体的图形类继承这个抽象类并实现它的绘制方法。例如，我们可以定义一个Shape类：

abstract class Shape {

public abstract void draw();

}

然后定义具体的图形类来继承Shape类并实现它的绘制方法，例如：

class Circle extends Shape {

public void draw() {

// 绘制圆形的代码

}

}

class Rectangle extends Shape {

public void draw() {

// 绘制矩形的代码

}

}

class Triangle extends Shape {

public void draw() {

// 绘制三角形的代码

}

}

现在，当我们需要添加一个新的图形时，只需要定义一个新的具体的图形类来继承Shape类并实现它的绘制方法即可，例如：

class Square extends Shape {

public void draw() {

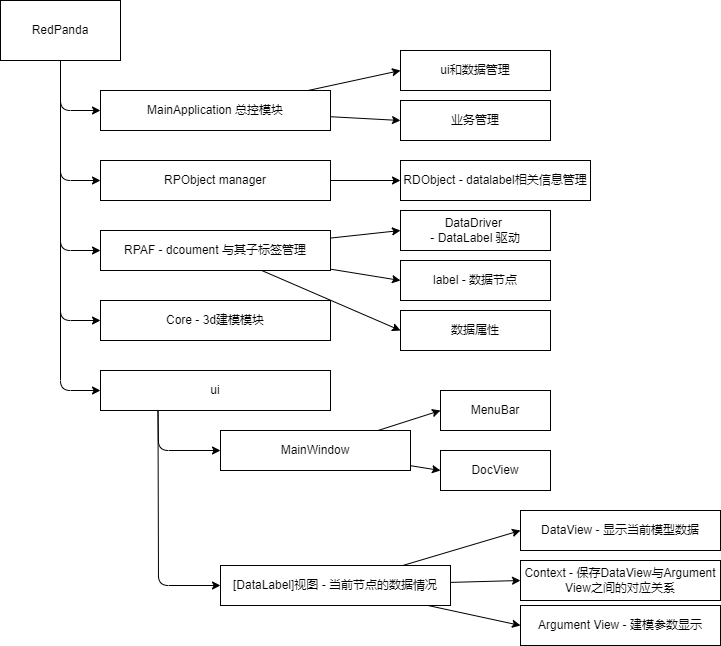
// 绘制正方形的代码

}

}

这样，我们就通过扩展现有的代码来添加新的图形，遵循了开放-封闭原则，避免了破坏原有的代码，提高了代码的可维护性和可扩展性。

1. 依赖倒置原则（DIP）：  
   高层模块不应该依赖于底层模块，而是应该依赖于抽象接口，而抽象接口不应该依赖于具体实现。
2. 接口隔离原则（ISP）：客户端不应该依赖于它不需要的接口，即一个类对另一个类的依赖应该建立在最小的接口上。
3. 最少知识原则（LKP）：一个对象应该只知道与之交互的对象的最少信息，即一个类不应该知道太多关于其他类的信息。
   * 1. 系统功能模块和接口设计
        1. 功能模块



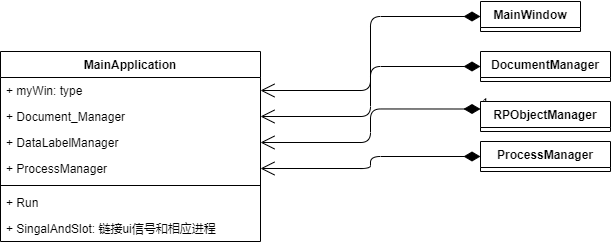
基于应用层的功能模块设计，设置不同的可视化模块，供用户使用使用不同的业务逻辑。

1. MainApplication：  
   管理程序中需要使用的所有资源，包括ui和数据管理模块（比如【RDObjectManager】，【DocumentApplication】）。  
   根据子ui的信号，启动相应业务；业务根据需要，调用程序资源，完成相应的操作。
2. RPObjectManger：  
   管理【RPObject】，而【RPObject】记录了程序运行与【DataLabel】相关的信息，比如【DataLabel】in 【DocView】，【DataLabel】的打开【LabelView】，【DataLabel】所表示的Shape.
3. RPAF – 文档表示框架：  
   管理【Document】数据，通过驱动器模式，对Label进行管理，负责表示节点的新增和更新。
4. Core – 3d建模模块  
   根据相应的命令和参数，构造3d图形数据。
5. Ui  
   两个主要UI窗体：【MainWindow】负责MenuBar与【DocView】的显示；【LabelView】负责显示参数信息和辅助的建模的3d、2d视图。
   * + 1. 模块接口与关系

首先,一切任务信息的最初来源是用户

* + - * 1. MainApplication

总控模块。



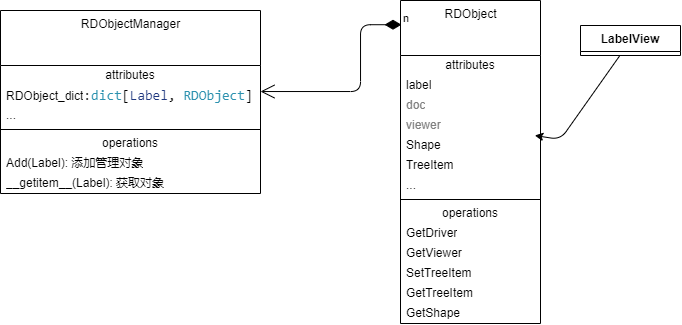
负责管理[RedPanda]中的所有资源,且作为总控模块,管理所有业务[Process]。

Run函数：启动程序。

SignalAndSlot：将用户输入数据信号与相应的业务【Process】相连接。

* + - * 1. RPObjectManager

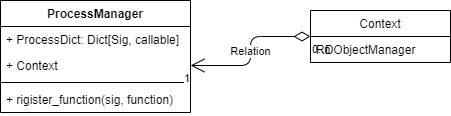
对象快捷索引，服务于【ProcessManager】。



RDObjectManager 管理【DataLabel】节点，记录与其相关的其他信息，比如在【DocView】中对应的树节点，打开的【LabelView】记录viewer中，所表示的Shape等。

函数操作，皆为获取与【DataLabel】相关之对象，比如TreeItem等。

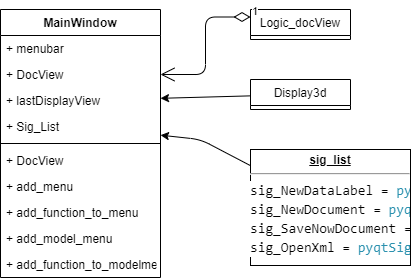
* + - * 1. ProcessManager



服务设计模式，根据需要记录相关业务操作。而业务操作只能由用户对于ui的操作产生的信号产生。

* + - * 1. MainWindow

主页面

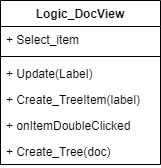


包含一个属性视图，展示当前打开的文档的数据标签，一个主视图，展示选中的对象的3d视图。

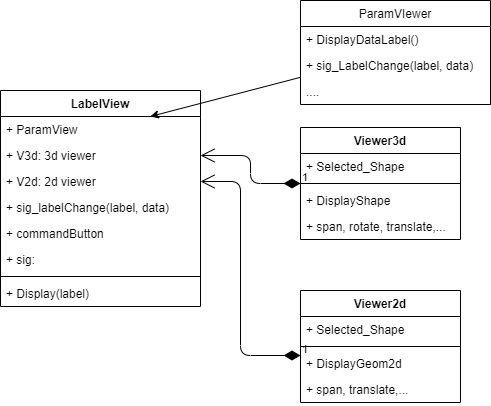
还有一些包含一些信号，对应用户在【DocView】上的操作的触发信号。

* + - * 1. docView

根据给定的标签，创造相应在树形视图下的TreeItem，标识当前选定的节点， Select\_item。



* + - * 1. DataLabel View



对【DataLabel】参数和建模结果的显示，根据用户对于参数的修改，即时更改建模结果。

* + - * 1. DataLabel

@startuml

class TDF\_Label {

\_\_hash\_\_

FindAttribute()

GetEntry()

GetFunctionID()

GetDriver()

GetAttribute()

GetAttrValue()

GetLabelName()

FindChild()

\_\_str\_\_()

\_\_repr\_\_()

}

[DataLabel]: 带有唯一数据Attribute(可能某些Attribute用来标识信息, 不直接标识数据)的Label, 或 sub Label中存在[DataLabel]的Label.

可编辑的最小对象是Attribute. 但是以Attribute的为基本单元进行管理是难度非常大, 因为其本身的意义不够明显. 所以我们选择[DataLabel] 作为管理的基本单元, 同时作为文档可视化的基本节点, 和construct& change的基本节点.

同时[DataLabel]确切表示一个数据.

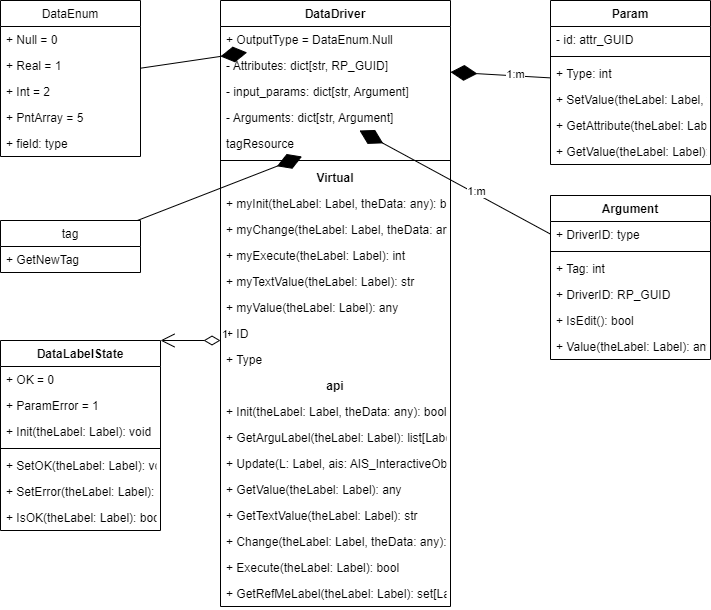
使用(func, param\_list)二元组来定义某个 [DataLabel].func: Execute函数, param\_list: Excute的确定类型参数列表.

* + - * 1. DataDriver

DataDriver是一个软件设计模式，用于将数据和处理逻辑分离，以便实现高度可重用性和可扩展性的数据处理。其基本思想是将数据的读取、处理和存储过程封装到一个独立的组件中，使得数据处理逻辑与应用程序的其它部分相互独立。

在软件开发中，DataDriver通常被用于处理与数据相关的操作，如从文件或数据库中读取数据、对数据进行转换、过滤、排序等操作，并将处理结果返回给应用程序。DataDriver可以用于处理各种类型的数据，包括数字、文本、图像、音频等。通过抽象数据处理过程，DataDriver可以在不同的应用程序中重复使用，从而减少代码量和开发时间，提高代码的可维护性和可重用性。

在CAD软件中，DataDriver被用来处理和管理对象的属性，包括对象的几何形状、位置、颜色、材质等。DataDriver可以在应用程序中实现对象属性的读取、修改和查询等操作，从而提高软件的灵活性和扩展性。例如，在CAD软件中，DataDriver可以用于实现对象的选择、过滤、排序和分组等操作，从而方便用户对复杂的图形场景进行编辑和管理。



1. DataEnum

DataEnum是一个枚举类，用于表示不同类型的数据。其中包括空数据、实数、整数和点数组等类型。

IsShape方法用于判断给定的类型是否为点数组类型。

1. DataLabelState

DataLabelState是一个枚举类，用于表示标签的状态。其中包括正常状态和参数错误状态。

Init方法用于初始化标签的状态。SetOK和SetError方法用于设置标签的状态为正常或参数错误。

IsOK方法用于判断标签的状态是否正常。

1. Tagsource

TagResource是一个标签资源类，用于管理标签的唯一标识符。GetNewTag方法用于获取一个新的标签标识符。

1. Param

Param是一个参数类，用于表示标签的参数。其中包括参数的唯一标识符、是否可编辑和参数的值等属性。

SetValue方法用于设置参数的值。GetAttribute和GetValue方法用于获取参数的属性和值。

1. Argument

Argument是一个参数类，用于表示标签的参数。其中包括参数的标签标识符、参数所属的数据驱动器的唯一标识符、

是否可编辑和参数的值类型等属性。Tag、DriverID和IsEdit方法用于获取参数的标签标识符、数据驱动器的唯一标识符和是否可编辑属性。

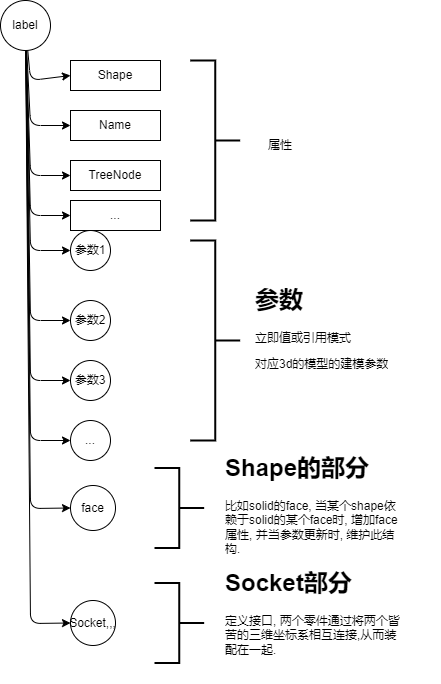
Value方法用于获取参数的值。

1. DataDriver

DataDriver是一个数据驱动器类，用于处理和管理数据。其中包括驱动器的输出类型、名称、唯一标识符、描述、

标签资源、执行结果、属性、参数等属性。DataDriver包括多个方法，用于初始化、获取、更新、执行和处理数据。

* + 1. 数据结构设计
       1. DataLabel



使用(func，param\_list)二元组来定义某个 [DataLabel]。func: Execute函数，param\_list: Excute的确定类型参数列表。

[DataLabel]的四种意义:

1. 作为数据节点，需要有函数(GetValue)来获取其表示的数据。
2. 作为一个函数节点，它的实际参数是可变的，我们拥有InitArgument函数对于参数进行初始化，和ChangeArgument函数对于参数值进行更改，从而更改表示的值。 当然还有Execute函数。
3. 作为一个可以被引用节点，或者作为参数节点(也相当于被引用)。 当self所表示的value改变，引用self的[DataLabel]的value也许改变。 那么，就需要能够获取所有引用self的[DataLabel]的GetRefMe函数。
4. 作为一个文档节点，需要能够获取Parent和sub Label这些由OCAF提供。

我们采用驱动器设计模式，在节点中添加一个GUID Attribute标识当前节点的驱动器/数据类型。 而驱动器包含了我们上述提到的所有函数。驱动Label作为函数被执行，产生数据。

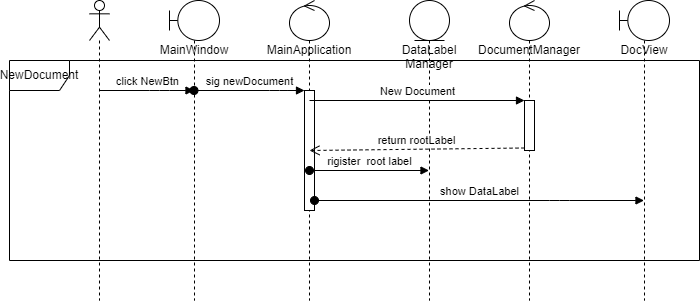
缺点

强类型的参数表达，限制了构造参数的种类，比如，构造一个Shape可能有多种参数方式。

当你想要拥有一个Cut and Transform的驱动，你需要自己重新定义。 非常的麻烦。

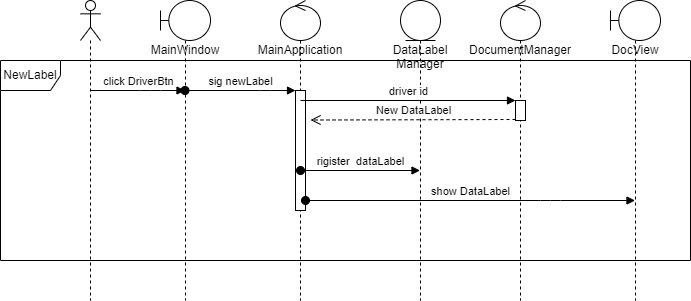
之前选择这个表示方式，主要设计集中于文档的节点的表示，且这是一个简单可行的方案，对于各可视化组件都相对简单，例如Construct & Change操作可视化参数几乎一样。

* + 1. 主要业务功能
       1. NewDocument



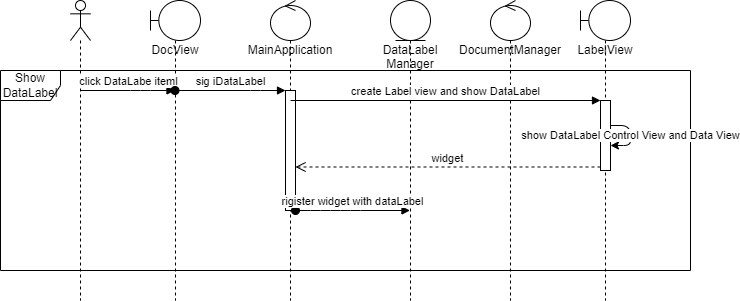
创建新的Document。

* + - 1. NewDataLabel

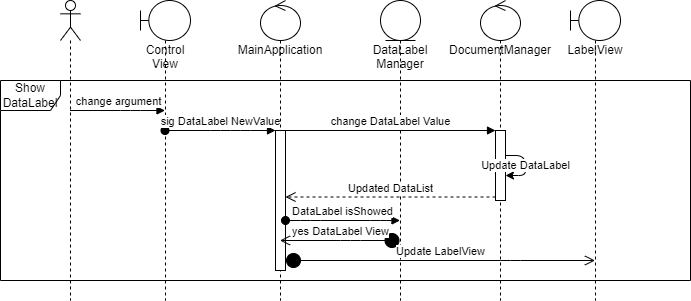


创建新的DataLabel

* + - 1. Show DataLabel



* + - 1. ChangeParam



更改【DataLabel】数据。

* 1. 系统实现
     1. 技术选型和系统实现
        1. 技术选型
           1. 主要编程语言：python

理由：数据结构丰富简单易用，加快开发效率。缺点：程序运行效率较慢。

* + - * 1. 3d建模工具：opencascade

OpenCASCADE是一种强大而灵活的开源软件开发平台，用于三维建模和数据交换。以下是OpenCASCADE和其他流行的开源三维建模软件开发平台之间的比较：

Blender：Blender是一种全功能的开源三维建模和动画软件，被广泛用于行业。虽然它与OpenCASCADE共享一些特性，如三维建模和可视化能力，但它主要专注于动画和渲染。另一方面，OpenCASCADE专门为工程和CAD应用程序设计，具有强大的参数化建模和数据交换支持。

FreeCAD：FreeCAD是另一种类似于OpenCASCADE的开源CAD软件平台。两者都提供强大的三维建模、参数化设计和数据交换工具。然而，FreeCAD具有更用户友好的界面，更适合初学者学习，而OpenCASCADE为经验丰富的用户提供更高级的功能和能力。

OpenJSCAD：OpenJSCAD是一种使用编程接口创建三维模型的开源平台。与OpenCASCADE和FreeCAD不同，它不是一个独立的CAD软件包，而是一个基于Web的开发环境。OpenJSCAD主要用于创建简单的模型和原型，而OpenCASCADE和FreeCAD更适合于更复杂的工程和CAD应用。

总的来说，OpenCASCADE是一种强大而灵活的软件开发平台，非常适用于各种工程和CAD应用。虽然它可能不像其他一些开源CAD软件平台那样用户友好，但它提供了高级的功能和能力，使它成为复杂的三维建模和分析工具开发人员的流行选择。

* + - * 1. 可视化工具：qt

1. 功能丰富：Qt提供了大量的GUI控件和功能，包括图形渲染、3D图形和动画等。相比之下，wxPython和Tkinter提供的功能相对较少。
2. 跨平台性：Qt是一个真正的跨平台框架，可以在多个操作系统和平台上使用，包括Windows、Linux、macOS、Android和iOS等。相比之下，wxPython和Tkinter的跨平台性能力有所限制。
3. 更好的可视化设计工具：Qt Creator是一个功能强大的IDE，提供了可视化的设计工具和图形界面编辑器，使得GUI应用程序的开发变得更加容易。
4. 更好的C++支持：由于Qt是用C++编写的，因此它提供了更好的C++支持。这使得Qt可以更好地与C++代码进行集成，并利用C++的高性能和内存管理能力。
5. 更好的性能：由于Qt的底层是C++，因此它可以提供更好的性能和更小的内存占用。相比之下，Python编写的wxPython和Tkinter性能较低，内存占用较高。

总的来说，Qt相较于wxPython和Tkinter具有更多的功能、更好的跨平台性能和更好的C++支持。如果需要创建复杂的GUI应用程序，并且需要高性能和跨平台性能，那么Qt是一个更好的选择。

* + 1. 功能实现和代码实现

主要代码:

* + - 1. DataLabel 实现

class DataEnum:

    Null = 0

    Real = 1

    Int = 2

    PntArray = 5

    Shape = 10

    Vertex = 13

    Edge = 14

    Wire = 15

    Face = 16

    Shell = 17

    Solid = 18

    @staticmethod

    def IsShape(enum):

        return 10 <= enum and enum <= 20

class DataLabelState:

    """

    状态标志, 表示DataLabel 数据是否可访问.

    """

    OK = 0

    ParamError = 1

    @staticmethod

    def Init(theLabel:Label):

        DataLabelState.SetError(theLabel)

    @staticmethod

    def \_SetState(theLabel:Label, state:DataLabelState):

        Attr\_State.Set(theLabel, state)

    @staticmethod

    def SetOK(theLabel:Label):

        Attr\_State.Set(theLabel, DataLabelState.OK)

    @staticmethod

    def SetError(theLabel:Label):

        Attr\_State.Set(theLabel, DataLabelState.ParamError)

    @staticmethod

    def IsOK(theLabel:Label):

        value = theLabel.GetAttrValue(Attr\_State.GetID())

        if value:

            value = DataLabelState.OK

            return True

        return False

class TagResource(object):

    def \_\_init\_\_(self) -> None:

        self.tag = 0

    def GetNewTag(self):

        self.tag += 1

        return self.tag

class Param(object):

    """ 属性对象声明

    """

    def \_\_init\_\_(self, theAttrID:RP\_GUID, value:str="", editAble=True) -> None:

        assert isinstance(theAttrID, RP\_GUID)

        self.id = theAttrID

        self.edit = editAble

        if isinstance(value, str):

            self.value = value

        else:

            self.value = str(value)

    @property

    def Type(self):

        return Lookup\_Attr[self.id]

    def SetValue(self, theLabel:Label, text:str):

        Lookup\_Attr[self.id].Set(theLabel, FromText(self.Type, text))

    def GetAttribute(self, theLabel:Label):

        return theLabel.GetAttribute(self.id)

    def GetValue(self, theLabel:Label):

        return theLabel.GetAttrValue(self.id)

    def \_\_str\_\_(self) -> str:

        return f"{Lookup\_Attr[self.id]}"

    def \_\_repr\_\_(self) -> str:

        return self.\_\_str\_\_()

class Argument(object):

    """ 子label 声明

    """

    def \_\_init\_\_(self, tag:TagResource, id:RP\_GUID,

                 editFlag:bool=True, valueType:int = DataEnum.Null) -> None:

        self.\_tag = tag.GetNewTag()

        self.\_driverID = id

        self.\_editFlag = editFlag

        self.valueType = valueType

    @property

    def Tag(self):

        return self.\_tag

    @property

    def DriverID(self):

        return self.\_driverID

    def IsEdit(self):

        return self.\_editFlag

    def Value(self, theLabel:Label)->any:

        from ..DriverTable import DataDriverTable

        aDriver = DataDriverTable.Get().GetDriver(self.DriverID)

        aLabel = theLabel.FindChild(self.Tag)

        return aDriver.GetValue(aLabel)

class DataDriver(object):

    """ base and manager

        通过函数接口访问数据, 方便更新.

    """

    OutputType = DataEnum.Null

    def \_\_init\_\_(self) -> None:

        self.name:str = ''

        self.id:RP\_GUID = None

        self.description:str = ''

        self.long\_description:str = ''

        self.socket\_0tag = 64

        # old

        self.tagResource = TagResource()

        self.Results: dict[str, dict] = dict()          # 结果

        self.Attributes: dict[str, RP\_GUID] = dict()    # 属性

        self.Arguments: dict[str, Argument] = dict()    # 参数

        # New

        self.input\_params = self.Arguments

    def myInit(self, theLabel:Label, theData: 实参)->bool:

        ''' 运行自己的参数检查和初始化就行

        返回初始化是否成功 <= 参数是否正确

        '''

        raise NotImplementedError()

    def myChange(self, theLabel:Label, theData:实参):

        # 只管更改

        raise NotImplementedError()

    def myTextValue(self, theLabel:Label):

        ''' 只复制取出值的text形式

        '''

        return str(self.myValue(theLabel))

    def myValue(self, theLabel:Label):

        ''' 只复制取出DataLabel 的value值

        '''

        raise NotImplementedError()

    def myExecute(self, theLabel:Label)->int:

        """ 仅负责获取参数和执行

        """

        return 0

    def Init(self, theLabel:Label, theData):

        """ 执行函数 - 初始化

        必要的初始化, 根据实话结果设置状态

        """

        self.\_base\_init(theLabel)

        if not self.myInit(theLabel, theData):

            DataLabelState.SetError(theLabel)

            return False

        if not self.Execute(theLabel):

            DataLabelState.SetError(theLabel)

            return False

        DataLabelState.SetOK(theLabel)

        return True

    def GetArguLabel(self, theLabel:Label)->list[Label]:

        label\_li = list()

        for argu in self.Arguments.values():

            argu:Argument

            sub = theLabel.FindChild(argu.Tag, False)

            label\_li.append(sub)

        return label\_li

    def Update(self, L: Label, ais: AIS\_InteractiveObject) -> bool:

        """ for TPrsStd\_AISPrsentaion,

        由于ocaf TPrsStd\_AISPrsentaion的原因,只能命名为Update

        """

        raise NotImplementedError()

    def GetValue(self, theLabel:Label):

        if not DataLabelState.IsOK(theLabel):

            return None

        return self.myValue()

    def GetTextValue(self, theLabel:Label):

        if not DataLabelState.IsOK(theLabel):

            return 'Label is error'

        return self.myTextValue(self.myTextValue)

    def Change(self, theLabel:Label, theData):

        """ 管理函数 更改

        """

        if not self.myChange(theLabel, theData):

            DataLabelState.SetError(theLabel)

            return False

        if not self.Execute(theLabel):

            DataLabelState.SetError(theLabel)

            return False

        DataLabelState.SetOK(theLabel)

        return True

    def Execute(self, theLabel:Label)->bool:

        """ 管理函数 - 执行

        """

        subLabel\_li = self.GetArguLabel(theLabel)

        for sub in subLabel\_li:

            if not DataLabelState.IsOK(sub):

                return False

        if self.myExecute(theLabel) != 0:

            return False

        return True

    def GetRefMeLabel(self, theLabel)->set[Label]:

        label\_set = set()

        if not theLabel.IsRoot():

            father = theLabel.Father()

            label\_set.add(father)

        shape\_Ref = Attr\_ShapeRef()

        if theLabel.FindAttribute(Attr\_ShapeRef.GetID(), shape\_Ref):

            for node in shape\_Ref:

                label\_set.add(node)

        return label\_set

    def \_base\_init(self, theLabel: Label):

        Logger().info(f'create Label:{theLabel.GetEntry()}, {None}, {self.ID}')

        TFunction\_Function.Set(theLabel, self.ID)

        DataLabelState.Init(theLabel)

    @classproperty

    def ID(self):

        """函数ID

        Raises:

            Exception: \_description\_

        """

        raise NotImplementedError('Must have ID')

    @classproperty

    def Type(self):

        """ 函数名

        """

        raise NotImplementedError('Must have Type')

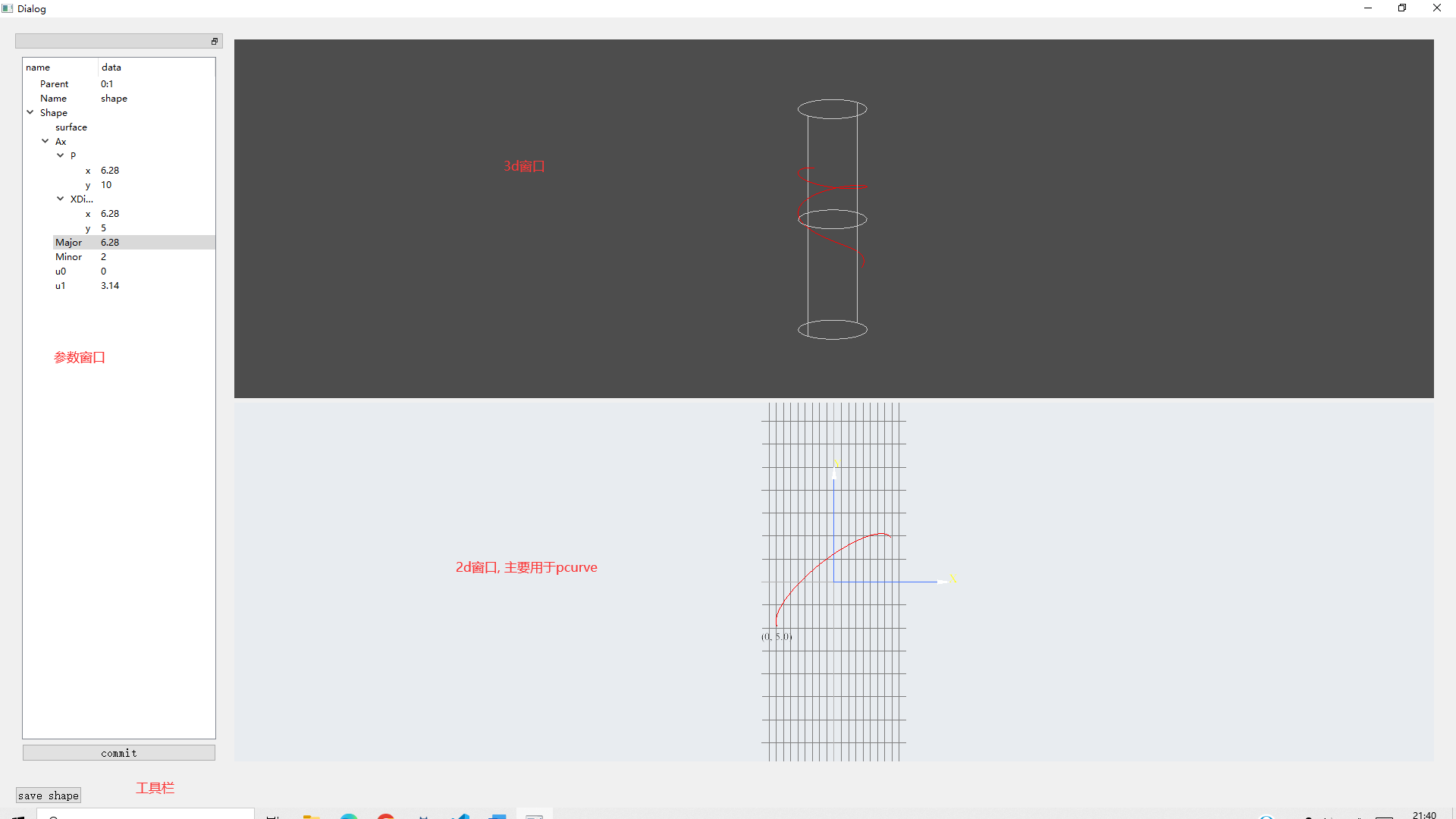
class SocketDriver(object):

    def \_\_init\_\_(self) -> None:

        self.\_\_socketType = ''

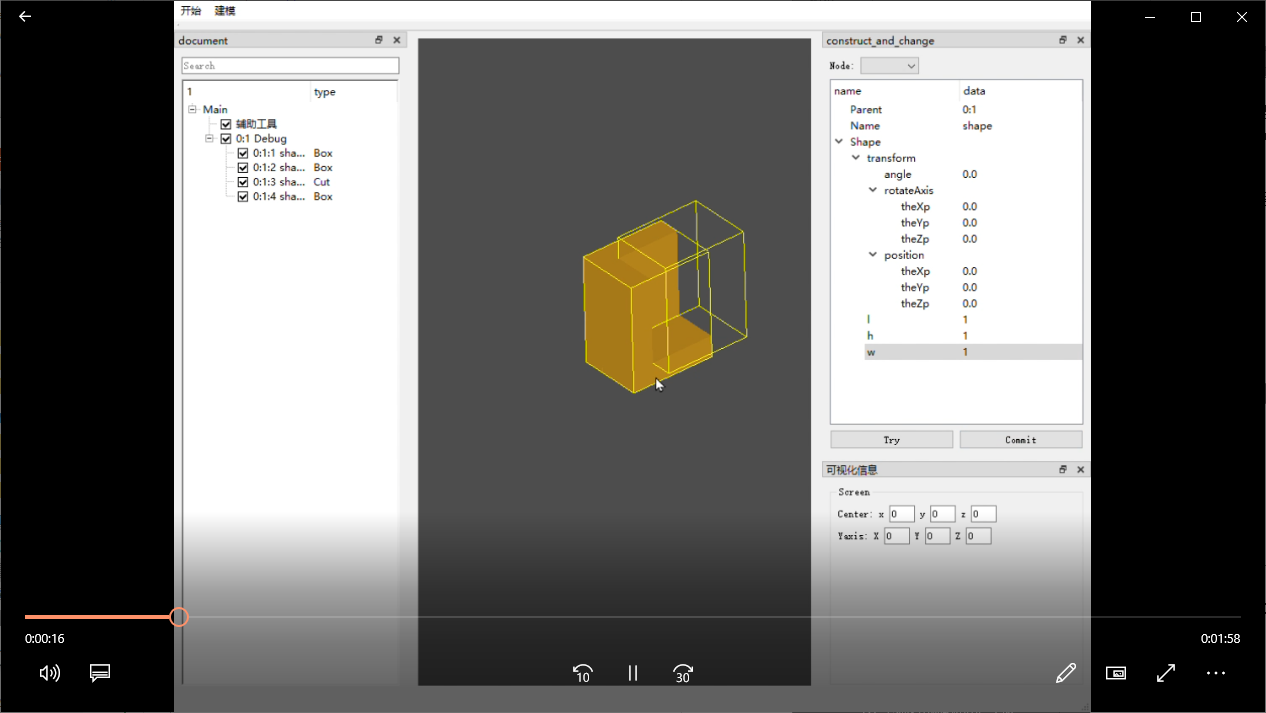
* + - 1. LabelView

1. Ui



* + - 1. MainWinodw

1. ui



* + 1. 系统的安全性和可靠性保证
       1. 安全性

开源本地操作软件， 无需安全性保证。

* + - 1. 可靠性保证
         1. 日志系统

在系统的核心内容部分，添加日志记录，以json格式记录当前输入参数和输出结果。

* + - * 1. 错误处理和恢复：

设计了错误处理和恢复机制，包括异常处理、数据备份和恢复，对于末端函数（实际读取数据，写入数据，不再次调用本系统内其他函数）严格检查返回值，对于错误返回数据加以报错。

对于用户文档，即时备份当前文档，防止用户未保存而造成大量数据丢失。以确保系统在出现故障和错误时能够及时恢复正常运行。

* + - * 1. 容错设计：

在系统设计和实现中采用了容错设计原则，包括硬件和软件的冗余、备份和自动切换等，以提高系统的可靠性和稳定性。

* 1. 系统测试与优化
     1. 系统测试方法和步骤

测试方法：功能测试，验证软件系统的功能是否符合需求和函数输入输出的要求。该测试方案可以通过手动测试、自动化测试、验收测试等方式进行。

测试步骤：

1. 确定测试目标和范围：在进行功能测试之前，需要确定测试的目标和范围，包括测试的功能点、测试的优先级、测试的时间和资源等信息。
2. 编写测试用例：根据函数需求，编写测试用例，包括测试场景、测试步骤、预期结果和实际结果等信息。测试用例应该覆盖所有的功能点，并且考虑到各种可能的输入和操作。
3. 设计测试数据：在进行测试之前，需要准备测试数据，包括各种输入和操作的数据和参数，以及测试环境的配置和条件。
4. 执行测试用例：按照测试计划和测试用例，执行测试，记录测试结果和测试日志。在测试过程中，需要注意测试环境的稳定性和数据的一致性。
5. 分析测试结果：对测试结果进行分析和比较，确定测试结果是否符合预期结果，以及测试用例的覆盖率和有效性。
6. 编写测试报告：根据测试结果，编写测试报告，包括测试概述、测试结果、问题列表、建议和改进等信息。测试报告应该清晰、简洁、准确地反映测试结果和测试过程。
   * 1. 功能测试

采用pytest，对于代码的主要功能进行测试，输出是否按照与输入的预期输出相匹配。

* + - * 1. 测试功能与范围
  1. 参考文献

1. [Introduction - Open CASCADE Technology Documentation](https://dev.opencascade.org/doc/overview/html/index.html)
2. Chat.Openai.com/chat
   1. 附录
      1. 系统使用说明和用户手册
      2. 系统源代码和测试数据

[nochat1205/RedPanda (github.com)](https://github.com/nochat1205/RedPanda)