DDE 自启动管理插件 项目文档

队员：复旦大学 朱元依、沈扬、朱俊杰
指导老师：张亮、陈辰
企业导师：王子冲

一、目标描述

我们小组的选题为proj223-control-center-startup-management-plugin

DDE 深度桌面环境的控制中心提供了插件功能，以便第三方开发者可以扩展其功能并额外的功能组件添加到控制中心之中。我们将在本项目中为 DDE 桌面环境的控制中心编写一个自启动管理插件。

二、比赛题目分析和相关资料调研

为了明确题目设置的原因与需求，我们请教了了企业导师，得知：目前用户对自启动权限的管理目前只能通过 dde-launcher （启动器/“开始菜单”）的右键菜单进行管理；考虑到此功能本身也并不复杂，所以设置了此课题，旨在解决这样的问题。

而对于此开发任务的具体实施，主要需要对`dde-dock`插件运行的原理和`deepin`开机自启动的设置方式进行调研，以便进行开发。调研的具体内容包括：

了解了Qt 插件标准（https://wiki.qt.io/Plugins）

阅读整理了`dde-dock`官方仓库中的插件工作原理（https://github.com/linuxdeepin/dde-dock/blob/master/plugins/plugin-guide/plugins-developer-guide.md）

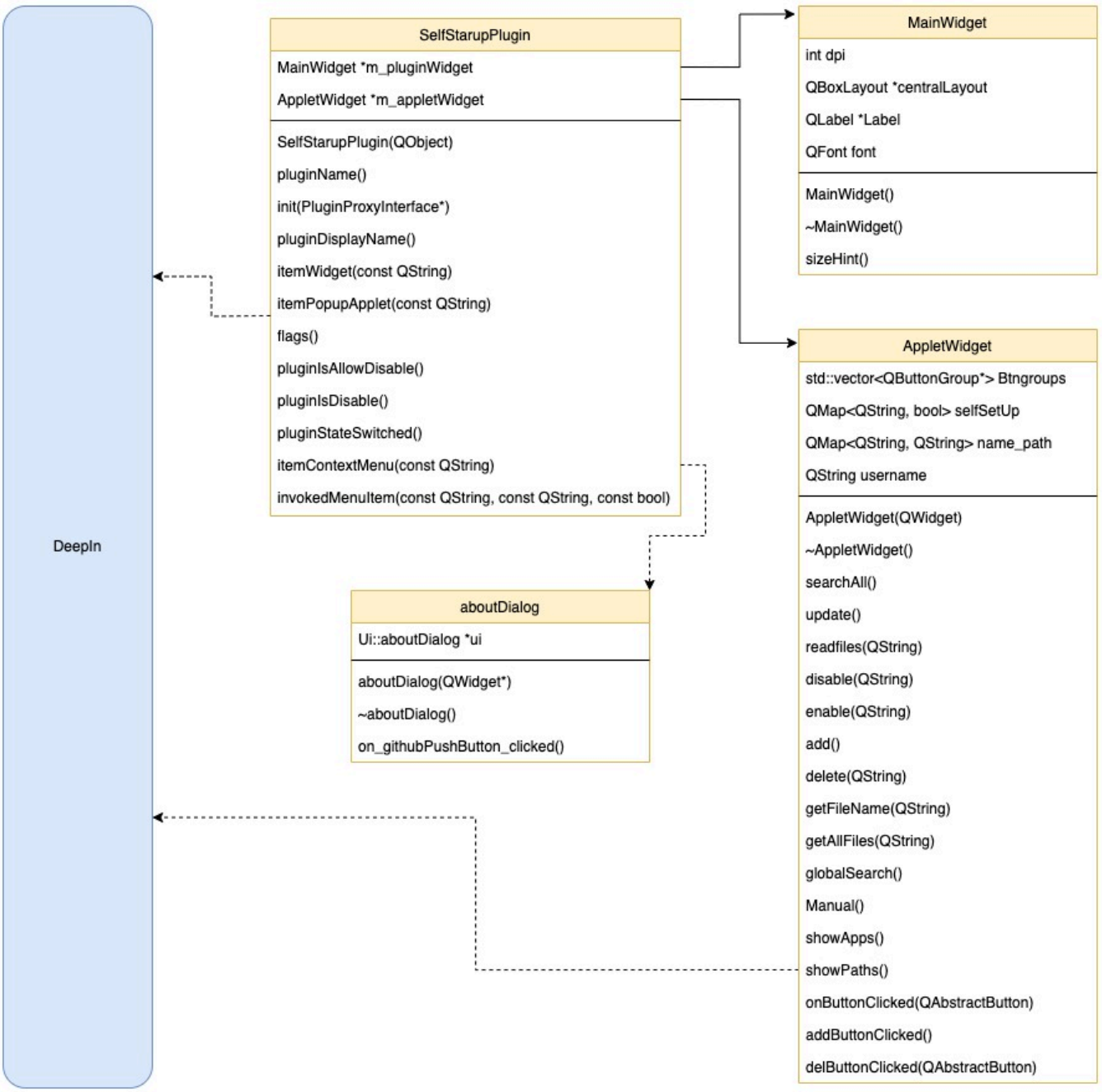
学习其他开发者的插件项目（https://github.com/sonichy/CMDU_DDE_DOCK）

阅读整理了各个论坛中关于`deepin`开机自启动项的讨论（https://bbs.deepin.org/zh/post/169824、https://blog.csdn.net/qq_21137441/article/details/124825726）

三、系统框架设计

1、项目整体结构设计

符合`dde-dock`提供的接口与插件开发规范，本项目将分为插件类（`SelfStartPlugin`）、部件类（`MainWidget`）和自启动管理窗口（`AppletWidget`）分别进行功能的实现。其中，项目的结构和各对象所包含的数据结构与方法如下图所示：



2、类功能说明

插件类`SelfStartPlugin`负责实现插件与`dde-dock`交互所必须的接口；部件类`MainWidget`负责在`dde-dock`中展示该插件的图标；自启动管理窗口`AppletWidget`负责实现核心功能：读取`Deepin`系统的开机自启动项、同时对各个软件的开机自启动进行管理（包括添加、删除、启用、禁用）。

3、实现描述

（1）插件类

在插件类中，主要实现了`dde-dock`中`PluginItemInterface`相关的接口，便于系统加载并实现插件的功能。接口包括以下内容：

名称	功能
SelfStarupPlugin	类的初始化函数
pluginName	返回插件名称，用于在 dde-dock 内部管理插件时使用
init	插件初始化入口函数，参数 proxyInter 可认为是主程序的进程
pluginDisplayName	返回插件名称，用于在界面上显示
itemWidget	返回插件主控件，用于显示在 dde-dock 面板上
itemPopupApplet	返回鼠标左键点击插件主控件后弹出的控件
flags	用于返回插件的属性，例如插件显示的位置，插件占几列，插件是否支持拖动等
pluginIsAllowDisable	返回插件是否允许被禁用（默认不允许被禁用）
pluginIsDisable	返回插件当前是否处于被禁用状态
pluginStateSwitched	当插件的禁用状态被用户改变时此接口被调用
itemContextMenu	返回鼠标左键点击插件主控件后要执行的命令数据
invokedMenuItem	返回鼠标右键点击插件主控件后要显示的菜单数据

（2）部件类

名称	功能
MainWidget	`MainWidget`部件的初始化函数，设置`dde-dock`图标的基础样式与文字内容
~MainWidget	`MainWidget`部件的析构函数
sizeHint	设置图标大小的函数

插件图标

最左侧的`SELF_STARTUP`图标为该插件图标。

（3）自启动管理窗口

名称	功能
AppletWidget	`AppletWidget`的初始化函数，负责绘制与用户交互的开机自启动项展示表格与修改设置选项
~AppletWidget	`AppletWidget`部件的析构函数
searchAll	搜索所有可设置开机自启动的软件
update	更新被自启动的软件
readfiles	工具函数，用于读取文件中的内容
disable	禁用开机自启动设置的后端接口
enable	启用开机自启动设置的后端接口
add	添加开机自启动设置的后端接口
delete	删除开机自启动设置的后端接口
getFileName	通过绝对路径找到文件的名字
getAllFiles	递归获取到某目录中所有文件
globalSearch	在所有文件中找到.desktop类型文件
Manual	寻找拟添加软件的函数，给用户在系统中寻找想要添加自启动项的软件
showApps	Debug工具，在Debug信息中打印软件名称
showPaths	Debug工具，在Debug信息中打印路径名称
onButtonClicked	自启动管理窗口启用/禁用按钮的处理函数
addButtonClicked	自启动管理窗口添加按钮的处理函数
delButtonClicked	自启动管理窗口删除按钮的处理函数

自启动管理窗口的前端页面

在`AppletWidget`的构造函数中，我们实现了便于用户查看系统开机自启动设置信息列表、并便于管理的前端展示页面：

开机自启动管理的后端接口

该部件中，主要设置了四个函数接口实现与操作系统的交互，分别是searchAll、update、disable、enable

首先要在MainWidget启动的时候在`/home`中读到用户名username

searchAll()

在`opt/apps`中找到所有用户下载程序的文件夹，读入文件夹名称`subdir`，再到`opts/appp/subdir/entries/application`中找到`.desktop`启动文件，解析文件内容，读入name字段并把`name - path`存到MainWidge类中，并且返回所有找到的app名称

update()

根据启动时读到的username在`/data/home/username/.config/autostart`里面找到里面所有的`.desktop`文件，分别读取并获取状态Hidden是否为false，把所有Hidden字段为False的插件（被设置成了开机自启动）在MainWidget的selfSetUp成员中设置为`pair<name, true>`

disable()

功能设计为禁用自启动。通过`name_path`中对name的索引找到路径path，对设置为自启动的应用path会在`/data/home/username/.config/autostart/*.desktop`中。读取该`.desktop`文件修改Hidden字段为True即可

enable()

功能设计为启用自启动。对于之前添加过的应用，`name_path`中得到的路径是在`/data/home/username/.config/autostart`中，此时与`disable()`过程相同，把Hidden字段设置成false即可；对于之前没有添加过的应用，`name_path`中的路径会在`opts/appp/subdir/entries/application/appname/entries/application/*.desktop`中，并且该文件是只读的。需要读取该文件并写入到`/data/home/username/.config/autostart/*.desktop`中，并且在文件的第一个分区里面写入一行"Hidden=false"

Add()

用户点击后打开文件资源管理器对话框，从中选择希望添加的可执行文件并返回文件路径。如果该文件是一个`.desktop`类型文件则添加到autostart中，如果不是则会在autostart中创建一个新的`.desktop`文件，并且把`Exec=`行设置为该可执行文件的路径

Delete()

从管理窗口中删除某应用的管理。传入参数是应用名称，找到该应用的`.desktop`文件路径并从autostart中删除，并且在数据结构`name_path`和`selfSetUp`中删去该部分信息

四、开发计划

第一步（4/9～4/18）

- [x] 调研`Deepin`、`dde-dock`、`QT`框架等相关内容
- [x] 设计项目方案
- [x] 分工

第二步（4/19～5/2）

- [x] 搭建主体插件类的框架
- [x] 设计启动项管理窗口的前端展示页面

第三步（5/3～5/13）

- [x] 开发部件类接口
- [x] 完善插件类功能

第四步（5/14～5/21）

- [x] 插件类右键功能开发
- [x] 完成配置文件

第五步（5/22～5/31）

- [x] Debug
- [x] 撰写文档

五、比赛过程中的重要进展

日期	进展
4/17	完成开发环境配置
4/27	完成插件框架设计
5/3	完成自启动管理界面的实现
5/8	完成自启动管理接口实现的讨论
5/12	完成自启动操作项的接口设计与实现
5/21	完成配置文件，并通过cmake编译
5/22	完成图标的Debug，前端界面展示正常
5/29	完成自启动操作接口Debug

六、系统测试情况

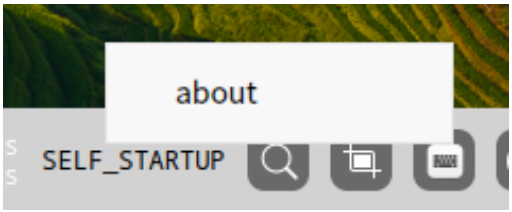
1、前端测试

在项目总路径下运行`install.sh`脚本，可完成插件的编译与安装：

```
sh install.sh
```

该脚本使用`cmake`编译代码，并安装到`dde-dock`插件文件夹中。运行后，可以正常展示自启动管理页面。点击加号后，用户可以在弹窗中选择想要添加自启动项的软件。

右键功能展示正常：



2、自启动管理功能测试

以浏览器为例。根据自启动管理窗口的设置，我们尝试添加并启用浏览器的自启动项。关机重启后，浏览器完成自启动。同理，删除、禁用功能均通过测试。

七、遇到的主要问题和解决方法

1、开发环境配置

（1）、配置 Deepin 操作系统

开发环境：Deepin 20Beta版

系统架构：x86

镜像下载链接：http://uni.mirrors.163.com/deepin-cd/20/deepin-desktop-community-1003-amd64.iso

虚拟机平台：WMware Workstation 16Pro

操作系统环境搭建参考博客：https://blog.csdn.net/qq_44133136/article/details/105887560

（2）、配置 Deepin 插件开发环境

安装基本开发环境：

安装包`build-essential`、`git`、`g++`、`cmake`、`dde`、`dtk`

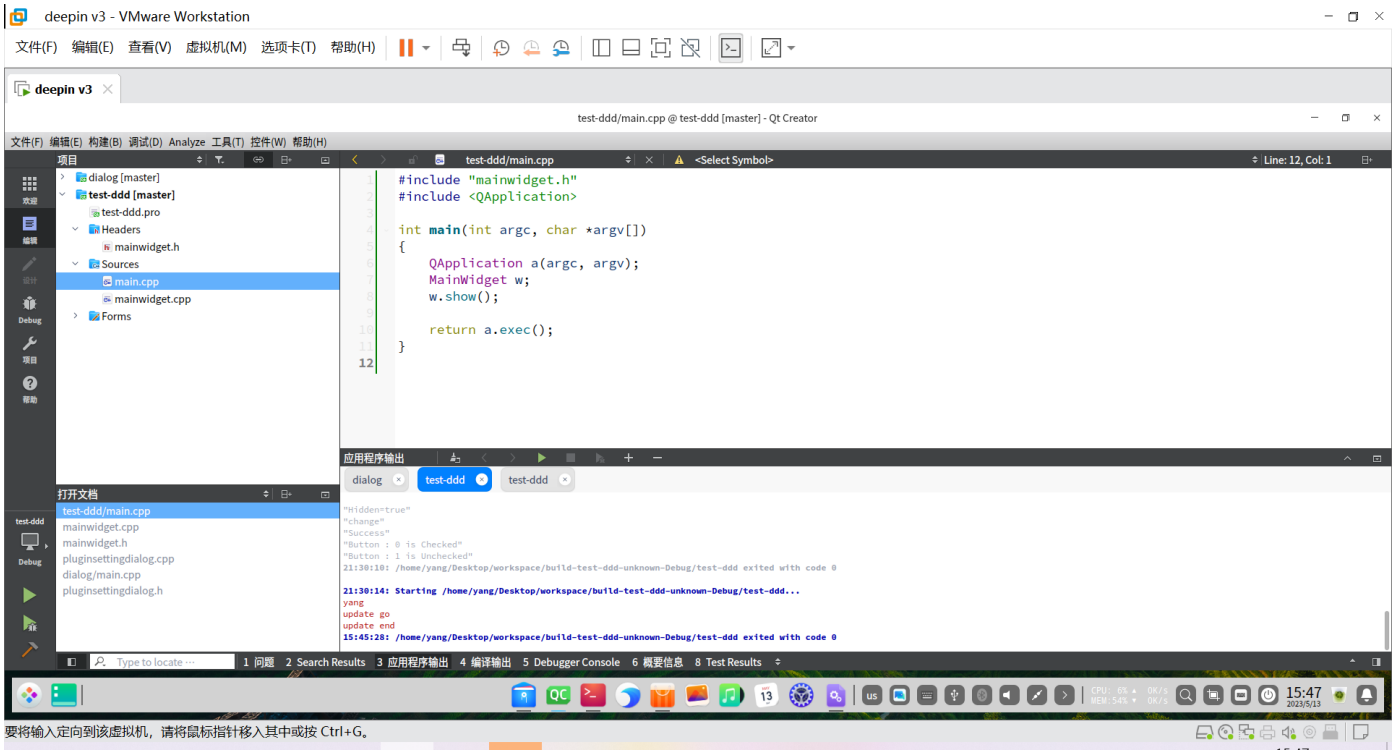
```
sudo apt install build-essential git g++ cmake
sudo apt install dde-dock-dev libdtkwidget-dev
```

安装 QT 开发环境：

安装`qt5-default`、`qt5-doc`、`qtcreator`

```
sudo apt install qt5-default qt5-doc qtcreator
```

依照上述方法，可在虚拟机中运行`qtcreator`，并在`qtcreator`中对插件进行测试



（3）、插件安装测试

为了测试所配置的虚拟机环境可用于 DDE 插件的开发，在环境配置中，本小组选取了`Github`仓库中的插件`dde-sys-monitor-plugin`（项目地址：https://github.com/q77190858/dde-sys-monitor-plugin）进行试运行。

根据上述方法配置插件开发环境后，可按照`dde-sys-monitor-plugin`中的提示信息顺利运行该插件。这表明开发环境配置已完成。

（4）环境配置中遇到的问题

安装sudo apt install libdtkwidget-dev libdtkgui-dev libdtkcore-dev出错

正试图覆盖 /usr/lib/x86_64-linux-gnu/qt5/mkspecs/features/dtk_install_dconfig.prf，它同时被包含于软件包 libdtkcommon-dev 5.5.23-1

在处理时有错误发生：

```
/var/cache/apt/archives/libdtkcore-dev_5.6.4-1+rb1_amd64.deb
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

解决方法：

使用sudo apttitude install libdtkwidget-dev libdtkgui-dev libdtkcore-dev
选择第二种解决方式（先卸载nY再重新安装sudo apt install libdtkwidget-dev libdtkgui-dev libdtkcore-dev）

2、开机启动项处理思路

为了实现插件可视化管理软件的开机自启动，了解`Deepin`系统开机自启动的功能实现是至关重要的。

通过查阅资料与实践，我们了解到`Deepin`系统包含自启动文件夹`~/.config/autostart`，该文件夹类似于 Windows 下的启动文件夹，系统开机时会执行该文件夹下的每个 desktop 文件 Exec 参数指向的脚本或可执行文件。

为了确认可行性，小组进行了该方法的验证。首先，通过`Deepin`系统自带的修改开机自启动设置的方法，修改开机启动项（图中修改`终端`的自启动项）：

随后检查自启动文件夹`~/.config/autostart`：

```
yang@yang-PC:~/.config/autostart$ ls
deepin-terminal.desktop org.deepin.browser.desktop
```

发现`deepin-terminal.desktop`文件被添加入了该自启动文件夹中，并且会在开机时自启动。

而取消终端的自启动时，`~/.config/autostart`中已有的`deepin-terminal.desktop`文件并不会被移除，而是其中的`Hidden`的字段会被修改为`false`，表示取消开机自启动设置。

由此，可以通过在插件中检查所有`~/.config/autostart`文件夹中`.desktop`文件的`Hidden`字段来搜索系统所有的自启动软件；也可以通过添加`.desktop`文件、修改`Hidden`字段的方式进行开机自启动设置的修改。

3、与 deepin 接口设计

（1）、关于开机自启动项

在查找deepin如何实现开机自启动时，看到了在deepin文件夹中有多个名为autostart的文件夹，其中有一部分包含了系统文件（例如终端、输入法等），但是在我们通过“开始”菜单修改其是否自启动性质时发现并为出现变化，并且我们自行在“应用商店”里面下载的软件并没有在设置为开机自启动后进入到该文件夹。经过在网络上搜索了关于deepin开发的一些讨论资料以及学习了一些关于启动文件`.desktop`文件功能并通过vscode对是否开机自启动设置后的文件进行比较后，找到了文件路径位于`/data/home/username/.config/autostart`中，并且在文件中有Hidden字段使得不用每次取消自启动是都要删除文件并且下次设置又重新加入

（2）、关于查找应用程序

一开始的想法是在系统目录下整体查找，找到系统中所有的`.desktop`文件，但显然这样过于繁琐且效率低下。后来在通过打开每个引用的`.desktop`文件后发现exec项均在`/opt/apps`中，打开该文件夹可以发现里面有所有用户下载的应用程序及其依赖等文件的文件夹，并在打开每个文件夹后可以找到启动文件均在`entries/application`中，并且可能有多个`.desktop`文件，所以把自启动的工作变为了对启动器文件用文本方式打开后的解析与修改

（3）、关于函数接口设计

对QT库中的数据类型和class的使用不太熟悉，需要根据官方文档对提供的api和C++模式的代码进行替换。但熟悉使用之后理解到了QT数据类型的多样性以及提供的方法会更加完备，在做开发时避免了很多自己重写方法的复杂流程。

4、cmake中遇到的问题

（1）、cmake 出现 dbus、core 找不到

解决方法：find_package()的REQUIRED后面新增找不到的对应的包

（2）、cmake 出现 FOUND=FALSE

```
/usr/lib/x86_64-linux-gnu/cmake/DtkWidget/DtkWidgetConfig.cmake


but it set DtkWidget_FOUND to FALSE so package "DtkWidget" is considered to

be NOT FOUND.
```

解决方法：使用.pro文件设置cmake需要的相关参数。

八、分工和协作

朱元依：插件类框架开发、部件类前端开发

沈扬：自启动管理功能逻辑设计、插件类右键功能开发

朱俊杰：自启动管理窗口后端接口开发（添加、删除、启用、禁用）

九、提交仓库目录和文件描述

```
. \
├── CMakeLists.txt \
├── README.md \
├── aboutdialog.cpp         #关于窗口的实现文件 \
├── aboutdialog.h           #关于窗口的头文件 \
├── aboutdialog.ui          #关于窗口的UI文件 \
├── appletwidget.cpp        #自启动管理窗口的实现文件 \
├── appletwidget.h          #自启动管理窗口的头文件 \
├── images                  #图片 \
|   ├── QT_IDE.png \
|   ├── QT_前端.png \
|   ├── deepin自启动修改.png \
|   ├── 类图.jpg \
|   ├── 右键.png \
|   ├── 图标.png \
|   ├── 结果.png \
```

```
|   ├── 中期类图.jpg \
|   ├── 中期测试.png \
|   └── 前端界面.png \
├── install.sh                    #插件安装脚本 \
├── main_aboutdialog_test.cpp   #关于窗口的测试文件 \
├── main_test.cpp                #测试文件 \
├── mainwidget.cpp               #插件类的实现文件 \
├── mainwidget.h                 #插件类的头文件 \
├── self_startup.json            #插件的元数据文件，指明了当前插件所使用的 dde-dock 的接口版本 \
├── self_startup.pro             #辅助 cmake 的配置文件 \
├── self_startup.qrc             #用于展示插件图片 \
├── selfstartupplugin.cpp         #部件类的实现文件 \
├── selfstartupplugin.h           #部件类的头文件 \
├── uninstall.sh                 #插件卸载脚本 \
├── 初赛报告.md \
└── 过程文档.md
```

十、比赛收获

借由为`deepin`的`dde-dock`编写插件的机会，我们小组了解了`deepin`系统相关的接口、`dde-dock`插件加载原理、开发逻辑等等操作系统相关的知识；同时我们在合作开发的过程中熟悉了软件工程的开发规范。小组同学在比赛中均受益匪浅。

十一、与企业导师的沟通情况

我们已于企业导师（王子冲）通过电子邮件进行联系。王导师向我们推荐了 deepin 开源社区各种公开渠道（如实时聊天渠道 Matrix、开发者社区讨论板等），鼓励我们在开发过程中将所遇到的问题在社区研发话题板块中进行公开探索。

此外，王导师还耐心的向我们介绍了该课题的设置原因，这对与我们在项目设计的过程中了解用户需求起到了很大的作用。