

**信息与软件工程学院**

**企业实习总结报告**

学 号： 2021091202022

姓 名： 刘治学

专业方向： 软件工程（互联网+）

企业名称： 成都中科合迅科技有限公司

实习岗位名称： 实习生

企业指导教师： 刘鹏江

院内指导教师： 王伟东

摘要

本报告详细记录了本人在成都中科合迅科技有限公司的实习经历。本公司是一家专业从事自主安全可控军用软件研发和网络空间军事装备研制的高科技企业。本人实习岗位为软件开发实习生，主要职责是参与公司核心产品LarkSDK的研发。LarkSDK作为一款国产化的跨平台C++开发框架，旨在填补国内软件开发平台的空白，增强自主可控能力，希望对标业界领先的Qt框架。本报告深入讨论了实习目标、具体任务分配、工程问题解决方案以及个人在实习过程中的技术成长和职业素养提升。通过亲身参与项目实践，我深刻体会到软件开发的整个流程，对产品设计有了深层思考，同时见识到了跨平台技术的复杂性，以及团队合作的力量。

**关键词：**合迅科技，跨平台框架，LarkSDK，国产软件，技术自主可控

ABSTRACT

This report comprehensively documents my internship experience at Chengdu Zhongke HeXun Technology Co., Ltd., a high-tech enterprise specializing in the development of autonomous and secure military software and the research and development of cyberspace military equipment. As a software development intern, my main responsibility was to participate in the development of the company's core product, LarkSDK. LarkSDK, as a domesticated cross-platform C++ development framework, aims to fill the gap in domestic software development platforms and enhance the capability for independent control, aspiring to benchmark against the industry-leading Qt framework. This report delves into the internship objectives, specific task assignments, solutions to engineering problems, and personal technical growth and professional development during the internship process. Through hands-on project involvement, I have gained a profound understanding of the entire software development process, deep contemplation on product design, appreciation of the complexities in cross-platform technology, and the power of teamwork.

**Keywords:** Sinux Tech, Cross-Platform Framework, LarkSDK, Domestic Software, Technological Autonomy

目 录

[第一章 企业实习概况 1](#_Toc22386)

[1.1 实习公司情况和岗位职责 1](#_Toc20150)

[1.1.1 实习公司概况 1](#_Toc26797)

[1.1.2 岗位职责 2](#_Toc29168)

[1.2 项目的背景、意义以及国内外研究现状 3](#_Toc3426)

[1.3 实习目标和具体任务 6](#_Toc8015)

[1.3.1 实习目标 6](#_Toc24302)

[1.3.2 具体任务 6](#_Toc6475)

[第二章 复杂工程问题和解决方案 9](#_Toc12857)

[2.1 Qt Graphics View Framework 预研 9](#_Toc13394)

[2.1.1 整体流程 9](#_Toc6287)

[2.1.2 重要功能总结 10](#_Toc30893)

[2.1.3 2D BSP 树在 QGraphicsScene 中的应用 16](#_Toc3198)

[2.2 一些关于空间数据结构的简单研究与实现 18](#_Toc25723)

[2.2.1 前言 18](#_Toc21098)

[2.2.2 均匀网格的概念 20](#_Toc22109)

[2.2.3 BSP的概念 20](#_Toc4850)

[2.2.4 简单的研究与实现 21](#_Toc11992)

[2.3 在X11下使用Cairo引擎绘制图形 31](#_Toc9571)

[2.3.1 安装Cairo库 31](#_Toc25977)

[2.3.2 使用Cairo库绘制图形 46](#_Toc29380)

[2.4 使用Woboq CodeBrowser搭建源代码网站 58](#_Toc30766)

[2.4.1 背景 58](#_Toc24128)

[2.4.2 工作原理简介 58](#_Toc3715)

[2.4.3 安装Woboq CodeBrowser工具 58](#_Toc17924)

[2.4.4 如何使用 59](#_Toc14234)

[第三章 知识技能学习情况 64](#_Toc1980)

[3.1 开发环境和工具 64](#_Toc8476)

[3.1.1 开发环境 64](#_Toc3674)

[3.1.2 开发工具 64](#_Toc22132)

[3.2 预备知识 65](#_Toc23254)

[3.3 新知识点学习和掌握情况 65](#_Toc21262)

[3.3.1 CMake 65](#_Toc18424)

[3.3.2 Conan 66](#_Toc18296)

[3.3.3 对实用工具的深入理解 67](#_Toc2415)

[3.3.4 C++ Boost后备标准库的知识 67](#_Toc12432)

[3.3.5 Xml基础知识 67](#_Toc10445)

[3.3.6 对STL的进一步理解 68](#_Toc6440)

[第四章 结束语 70](#_Toc5908)

[4.1 实习工作完成情况总结 70](#_Toc25030)

[4.2 对于企业实习的收获及体会 70](#_Toc24571)

[参考文献 73](#_Toc1900)

[致谢 74](#_Toc13337)

### 第一章 企业实习概况

### 1.1 实习公司情况和岗位职责

### 1.1.1 实习公司概况

本人实习单位，成都中科合迅科技有限公司，成立于2013年，总部位于成都市高新区科园南路1号海特国际广场4号楼7-8层，注册资本2766万元，现有员工300余人。是专业从事自主安全可控军用软件研发和网络空间军事装备研制的高科技企业。公司在北京、上海、武汉、西安设有分支机构，全国有超过50家的军队、军工科研客户。

公司的口号是“铸造军工品质，磨砺国产匠心”。国家目前的军工行业现状分析主要有三点。第一，一带一路、走出国门，必须建设能打胜仗的全球化现代军队，国家军队体制改革孕育重大行业发展机会。第二，现有科研体系无法满足国家强国强军梦想“军民融合、民参军”，给民营科技企业提供历史发展机遇；西方主要发达国家军民通用技术已超过80%，军事专用技术已不到15%。第三，我国国防军费2017年突破万亿大关，武器装备建设有望持续加速，战略空军、远洋海军、国防信息化成为军队建设的重中之重。

公司就在这样的背景下成立与发展，至今已过十年。公司的发展阶段大致可分三个阶段。第一个阶段，2013年到2017年，是基础能力建设阶段，主要成绩有军工二级保密资格，GJB9001C-2017质量管理体系，国家级高新技术产业等。第二个阶段，2017年到2020年，是核心能力聚焦阶段，主要成绩有第二批国家级专精特新“小巨人”，四川省瞪羚企业，连续四年成都市新经济百强企业等。第三个阶段，2020年至今，是核心业务产品化阶段，主要成绩有四川省雁阵企业——省发改委定向培育，四川省首批30家“新经济示范”四川省计算机学会科学技术一等奖等8个省、部级奖项等。公司还获得了很多其他的资质与荣誉。

### 1.1.2 岗位职责

公司的核心业务可以简单概括为“一个核心产品，两个基本能力，两个数字化方向”。具体见图1-1：



图1-1 公司核心业务

我在实习期间就参与在合迅智灵产品的研发中。这也是公司的核心产品。合迅智灵产品完成了在国产化软硬件环境下多平台的全覆盖，支持桌面、移动、嵌入式等多种平台，满足了军工科研院所适用的更高要求，并与银河麒麟、元心OS、锐华Reworks等公司签订了战略合作协议，目前合迅智灵已进入国防科工局国产化工具软件推广应用目录。

公司合迅智灵产品的功能架构见图1-2。进入公司以后我所在项目组一直在进行基础平台的研发工作。这一部分的具体工作将在具体任务部分进行阐述。



图1-2 合迅智灵产品功能架构

### 1.2 项目的背景、意义以及国内外研究现状

LarkSDK是一款通用C++开发框架。框架的最终目的是简化应用程序的开发。为达此目的，LarkSDK也提供了一整套语义直观明晰、设计模块化的C++类库，为用户封装了操作系统平台与硬件差异、提供了开箱即用的基础编程工具。

和[STL](https://en.cppreference.com/w/cpp/standard_library)、[Boost](https://www.boost.org/)相比，LarkSDK可以用于构建图形界面。

和[MFC](https://learn.microsoft.com/en-us/cpp/mfc/mfc-desktop-applications)、[GTK](https://www.gtk.org/)相比，LarkSDK具备跨平台能力，同一套源码可以同时工作在Linux和Windows之下。

当然提到跨平台的C++图形开发框架，Qt在业界仍然是占据统治地位的产品。[1]Qt从上个世纪九十年代开始发展，直到现在都是业界的一面大旗，也为LarkSDK的研发提供了很多参考。Qt所提供的功能已经非常完备，绝大多数从事C++软件开发的工程师基本上都以Qt为第一选择。

但是可能由于战线过长或者历史遗留等问题，在研发LarkSDK的过程中，我们发现Qt中部分功能的设计也许并不是最优解，甚至部分功能的设计相当糟糕。同时加上国际形势和国产生态的需求，LarkSDK应运而生。相比Qt，LarkSDK更轻量、更简单、更年轻，同时更懂国产生态。LarkSDK并不奢望取代Qt，仅仅是为业界提供另一个选择。这条路并不容易，但我们坚信这是一条正确的路，难而正确的路。

随着自主可控IT行业重点部分建设日趋完善，国家现已正式步入产业链改革深水区。其中软件开发平台是IT行业的血液，它可以支持各种底层的芯片和操作系统，将下层硬件和操作系统细节封装起来，对上层应用软件提供统一、易用、简洁的开发接口与工具，是打通芯片、操作系统与应用软件的关键，更贯穿了应用软件的设计、开发、测试与维护全生命周期。

目前，国内在软件开发平台领域的缺失，导致国内软件行业在开发软件时不得不大量使用国外厂商的开发平台，从而遭遇信息安全存在隐患、国产操作系统环境适配性差、维护服务没有支持、版权使用存在风险等重要问题，没有国产软件开发平台支持，软件开发环节效率低、软件开发成果不稳定也成为国内软件行业普遍存在的问题。[2]国产C++软件开发平台的目的，是打造通用跨平台的国产化基础开发套件，提供对应用程序的关键共性运行逻辑的支持，进而构建完整的国产化软件集成开发环境，以构建完整的基于C++语言的国产应用程序开发生态。这也是我们需要努力做一个属于国人自己的LarkSDK产品的真正原因。

LarkSDK在设计之初就是致力于跨平台的，它封装了不同操作平台的特性和底层调用，隐藏了各个操作系统平台的不同之处，让用户可以专注于业务开发而无需过多关心平台差异；同时，LarkSDK也并非单纯的浅层UI库，它覆盖了从底层的操作系统事件监听与分发，到通用的跨平台用户组件的外观与行为定制；它既提供各种各样的底层代码工具，如常见元素容器和加解密算法，以提升C++的开发效率和体验，又能够通过自带的用户界面框架直接构建图形用户程序，未来还将提供界面编辑器及自动化测试工具等，从而完成应用软件开发的全生命周期闭环。

和Qt Framework一样，LarkSDK是一款单纯而完整的C++开发框架。万丈高楼平地起，除[必要的基础轮子](http://caiyi.tech/larksdk/concept/larksdk-3rdparty.html)外，LarkSDK并不依赖任何其他框架而存在。

由于一些原因，目前国内的软件行业语境下，对于诸如语言、框架、工具、平台等技术概念，存在一定程度上的混淆使用。虽然不太愿意这么表达，但是在国内LarkSDK确实是唯一的一款通用基础开发框架，技术上并没有真正意义的竞品。这也是我们所看到的绝大部分需要使用C++开发应用软件的场合都是Qt的原因。

而合迅智灵是一款全国产的、自主可控的、拥有完全自主知识产权的基础软件开发平台。LarkSDK则是平台为应用软件开发者提供的底层软件开发框架。LarkSDK致力于成为Qt之外的第二选择。

不过值得一提的是，站在市场层面解决实际需求的视角下，确实存在一些产品，和LarkSDK存在一定的功能覆盖，例如：

**1、**[**统信DTK**](https://docs.deepin.org/info/%E5%BC%80%E5%8F%91%E5%85%A5%E9%97%A8/%E5%9F%BA%E7%A1%80%E7%8E%AF%E5%A2%83/DTK/%E6%A6%82%E8%BF%B0/%E6%A6%82%E8%BF%B0/%E7%BB%9F%E4%BF%A1DTK)

DTK(Development ToolKit)是统信基于Qt开发的一整套简单且实用的通用开发框架，处于统信UOS操作系统的核心位置。其提供丰富的开发接口与支持工具，满足日常图形应用、业务应用、系统定制应用的开发需求，提供30余个预定义组件，如统信UOS浏览器、音乐、邮件等40余款UOS应用均使用DTK开发。

其本质上是一个在Qt的基础之上构建的扩展组件库。利用Qt框架与操作系统底层对接，借助Qt的能力实现各种具体的用户组件。用户本质上还是在使用Qt开发。其具备的跨平台能力本质上也是Qt本身的能力。

**2、**[**华为ArkUI**](https://developer.harmonyos.com/cn/develop/arkUI)

ArkUI是一套用于构建图形用户界面的声明式UI开发框架。它使用极简的UI信息语法，提供丰富的UI组件及包含实时界面预览工具在内的集成开发环境等。提供基于ArkTS开发语言的应用程序接口，支持各种HarmonyOS设备。

严格说ArkUI和LarkSDK在技术上并无关联，其本质是一套专门用于鸿蒙HarmonyOS生态的开发工具链的一部分，完整的鸿蒙HarmonyOS生态包含ArkTS(基于TypeScript的开发语言)、ArkUI (基于ArkTS的一套界面组件语言)、ArkCompiler (用于处理ArkTS的编译工具)，以及DevEco Studio(基于VSCode构建的集成开发环境)，构成完整的生态工具链。也即是说，ArkUI是生态专有生态的开发工具，和直接面向操作系统底层的通用开发工具LarkSDK并不处于一条技术路线上。

**3、**[**致远电子AWTK**](https://www.zlg.cn/index/pub/awtk.html)

AWTK全称为Toolkit AnyWhere，是ZLG倾心打造的一套基于C语言开发的GUI框架。旨在为用户提供一个功能强大、高效可靠、简单易用、可轻松做出炫酷效果的GUI引擎，支持跨平台同步开发，一次编程，到处编译，跨平台使用。

然而AWTK本质上也是一套基于[SDL(opens new window)](https://www.libsdl.org/)构建的GUI库。其主要能力，如图形渲染、跨平台与底层交互等，均由SDL框架提供。

总的来讲，LarkSDK产品的意义是重大的，我个人也很幸运能参与到项目的研发当中。

### 1.3 实习目标和具体任务

### 1.3.1 实习目标

对于实习目标，我将其分为公司目标和个人目标。对于公司目标，我希望严格按照公司的安排，全身心投入合迅智灵基础平台的研发。对于个人目标，我希望能在实习期间，能跟随前辈的脚步，保质保量完成好安排到的工作，同时能够在实习期间学到更多知识和技术，以学习为主，以功利为辅，提升培养自己的技术和能力。

### 1.3.2 具体任务

**1、LarkSDK**

前面提到，我目前参与在合迅智灵基础平台的产品研发当中，下面我对该基础平台的业务进行相关介绍。

合迅智灵基础平台，全称叫合迅智灵国产化基础开发套件，也就是前面提到的LarkSDK。LarkSDK是合迅智灵产品LarkStudio5的核心部件，是一套跨平台的C++基础开发库。

LarkSDK对标Qt。Qt是一款历史悠久，发展稳定的跨平台C++基础开发库。在全世界被广泛使用，但美中不足的是，该产品并不是国人开发。为顺应军工国产化的大趋势，公司在2020年起进入合迅智灵产品的研发，其中LarkSDK的部分作为基础平台，又是重中之重。

LarkSDK当前阶段分为以下三个主要子模块和三个扩展模块。

三个主要子模块：

(1) LarkSDK-Core (larkcore)：核心类库，包含对象模型、事件机制、线程管理和主程序框架等。

(2) LarkSDK-Util (larkutil)：实用跨平台工具类库，包含常用的数据结构和算法。

(3) LarkSDK-GUI (larkgui)：图形绘制类库，包含跨平台图形绘制接口、基础窗体和常用组件，以及多绘图引擎支持支持等。

三个扩展模块：

(1) LarkSDK-DB (larkdb)：数据库支持模块。

(2) LarkSDK-Network (larknetwork)：网络编程支持模块。

(3) LarkSDK-XML (larkxml)：XML支持模块。

公司与本校本院某实验室在2021年末达成了合作，校方开始了LarkSDK部分的初步编写。到23年末为止已经经过一期和二期的工作，已初步完成开发。但由于学校学生对实际工程的接触较少以及理论知识不牢固，编写的代码是存在很多问题的。在我进入公司以前，公司的前辈们就针对部分问题进行了优化重构，但是整体来看仍难以达到标准。

因此在进入公司以后，我的第一份任务就是进行代码走查。何为代码走查？简单来讲就是读别人的代码发现问题，但是这其中的工作量和难度也是不小的。领导对我们的要求是对于该部分涉及到的内容，包括涉及到的其他代码，都要阅读和理解，然后汇报。虽然枯燥甚至有些难，但我确实学到了很多知识和技巧。我的老总曾经说过：“提升技术的最好办法就是阅读代码。”我觉得说的有理，阅读别人的代码，首先是理解别人的思路，然后延申思考，思考他这里为什么写得好，写得不好，想想如果是自己该如何构思，如何下笔，这对一个问题从0到1的剖析是非常有帮助的，而我在学习的过程中，不仅要学习知识和技术，更重要的是培养工程上的思维和方法。

以下列出实习过程中的代码走查记录表1-1：

表1-1 代码走查记录表

|  |  |
| --- | --- |
| **迭代时间** | **走查代码** |
| 1.15 - 1.26 | LStack、LQueue、LByteArray |
| 1.29 - 2.8 | LObject、LApplication、LSignal |
| 2.19 - 3.1 | 线程管理、线程数据、互斥锁、读写锁部分 |
| 3.4 - 3.15 | LPen、LBrush、LLinearGradient、LMenu、LMenuItem、LMenuItemSeparator |
| 4.7 - 4.19 | LarkXML模块 |

既然做了代码走查，那么必然需要记录，后续优化重构。代码走查的目的是发现问题，记录问题，而最终的解决依赖于后续的代码优化重构。在实习期间，我完成了线程部分代码、LPen、LBrush等的优化修改，完成了栈LStack和队列LQueue容器、字符串列表（LStringList）、关联性容器（LHash、LMap、LSet）、日期时间数据类型（LDateTime、LDate、LTime）以及三元组容器（LTrio）等工具类代码的优化重构。同时我自主研究QDir和QFileInfo的设计，去掉设计不合理的地方，设计出了属于LarkSDK自己的LFileSystemPath和LFileSystemEntry，与上面的结构分别对应。目前，在组内成员的共同努力下，util部分目前已经基本全部处理完毕。经过测试，目前功能稳定可用，具体见图1-3。



图1-3 util模块优化重构工作记录表

第三，课题调研。绝大多数的功能都是在产品不断的发展和迭代中，因为需求而应运而生的。面对一个新的需求或者问题，需要进行大量的调研才能解决问题。这不仅是工作的过程，更是一个学习和进步的过程。在实习期间，我很幸运参与了一些课题的完整调研，并且从中学到了很多知识和技术，也为公司的产品贡献了自己的力量。具体见表1-2。

表1-2 课题调研记录表

|  |  |
| --- | --- |
| **迭代时间** | **调研课题** |
| 4.7 - 4.19 | 标准库string的sso优化对LVector插入影响的探究 |
| 4.22 - 4.30 | LDir和LFileInfo的语义和设计 |
| 5.13 - 5.24 | Qt Graphics View Framework 预研 |
| 5.27 - 6.7 | 一些关于空间数据结构的简单研究与实现 |
| 6.17 - 6.28 | 在X11下使用Cairo引擎绘制图形 |
| 7.1 - 7.12 | 使用Woboq CodeBrowser搭建源代码网站 |

**2、LarkTestKit**

LarkSDK作为一个跨平台的C++基础开发库。我们是采用googleTest开源框架进行测试的，为了做到国产化适配，故提出了自动化测试框架LarkTestKit的需求。该部分源代码交由校方某同学负责，提交了初版，我的任务是审核该部分代码，与校方同学和测试沟通，并设计编写测试用例。巧合的是，负责LarkTestKit的学长在四月中旬也来到公司也进行线下实习。因而在中期的工作中，我协助他一起完成LarkTestKit的工作，一起讨论LarkTestKit中的部分设计思路、实现方案等。目前LarkTestKit已经测试完毕，初步测试结果良好，功能稳定可用。

### 第二章 复杂工程问题和解决方案

本部分将针对上述具体任务当中的某个环节、某个步骤遇到的工程问题出发，分析问题的来龙去脉，并设计合理的解决方案应对，总结于此。有部分问题在早期的报告中已阐述过，这里不再赘述。

### 2.1 Qt Graphics View Framework 预研

在Qt中，有一个重要的功能模块是Qt Graphics View Framework，为批量管理2D图元，处理图元绘制和事件转发提供了很好的功能和实现。它允许开发者创建交互式的2D图形应用程序，特别适合于需要管理和交互大量定制的2D图形元素的应用程序，例如图形编辑器、游戏以及科学数据可视化等场景。最重要的是查询效率极高，经测试，能在几毫秒里面在几百万图元中找到目标。因此LarkSDK也提出了同样的需求，我开始进行这方面的预研。

该架构涉及到的最主要的三个类是QGraphicsScene、QGraphicsView和QGraphicsItem。

### 2.1.1 整体流程

**1、绘制流程（QGraphicsItem->QGraphicsScene->QGraphicsView）**

[3]QGraphicsItem当中保存了自身的“场景坐标”供QGraphicsScene进行管理。在绘制时，由QGraphicsView对象调用渲染方法，根据自身所设置的可视化相关属性，基于“视图坐标”确定将要绘制的QGraphicsScene当中有哪一部分“场景坐标”内的图元需要渲染，随后通过QGraphicsScene提供的方法，将属于这部分“场景坐标”内的图元(也就是QGraphicsItem)全部找出，并渲染这些图元到可视化viewport中。

**2、事件流转（QGraphicsView->QGraphicsScene->QGraphicsItem）**

由QGraphicsView绘制出的viewport是与用户直接交互的对象，用户发起的UI事件都由QGraphicsView首先接收，它在接收到事件以后对其中的部分参数进行适当的处理(如鼠标事件的坐标进行转换)，随后将事件转发给QGraphicsScene对象，由QGraphicsScene确定事件发送到哪个具体的图元(如鼠标事件发送到符合坐标位置的图元，键盘事件发送到当前焦点所在的图元)，图元在接收到事件以后作自行处理。

**3、QGraphicsScene的查询图元的BSP树相关算法**

在实际的工程场景中，极有可能会出现一个QGraphicsScene管理非常多个QGraphicsItem，例如几百万个，当用户与QGraphicsView交互的时候，需要经过QGraphicsScene将本事件传递到某个或者某些具体的图元，如果挨个遍历每个图元，并且判断是否触发事件，那么耗费的代价就太大了。而希望做到的效率是几百万条数据在几毫秒以内能够确定目标图元，因此需要实现对应的算法。

**4、图元实现优先级以及顺序和事件实现优先级以及顺序**

QGraphicsItem本质上是一个抽象父类，为子类规定了各种需要覆写的方法，子类通过重写这些方法能够很好的被QGraphicsScene所管理，最终呈现出来。因此不仅支持官方提供的矩形、椭圆、文本框等，还能处理自定义类型的图元。

在现阶段，先不考虑纯虚的抽象类，先把QGraphicsItem当成矩形框来做，先和QGraphicsScene把流程跑通，后续在扩展的时候也能非常快就完成工作。

事件处理是一个大课题。经讨论，事件处理的优先级是鼠标事件大于键盘事件大于拖拽事件。因此第一步实现的时候，先实现最基本的handleMousePressEvent，当这个流程通了以后，后续的所有操作都是一模一样的了。

**5、整体实现顺序**

先把QGraphicsView剔出来，QGraphicsView实际上只负责可视化的绘制，真正做处理的操作在QGraphicsScene这一层，因此管理类QGraphicsScene和图元类QGraphicsItem是第一步需要实现的。

目前的工作计划经商讨以后，决定先调研QGraphicsScene和QGraphicsItem的源码，明确如何以最小的代价实现我们目前需要的功能，包括事件转发、绘制逻辑、坐标系统等，这样明确需求和设计以后再实现初版，搭好整体架构。

### 2.1.2 重要功能总结

**1、QGraphicsScene**

QGraphicsScene类提供了一个管理大量QGraphicsItem的容器。

QGraphicsScene类只管理所有的图元，若想要可视化场景，需要配合QGraphicsView。

QGraphicsItem作为QGraphicsScene类的图元，自然有很多接口都是围绕着QGraphicsItem来的，例如：

(1) 添加图元：除了基本的addItem()，还有针对特殊类型图元的add\*()函数。

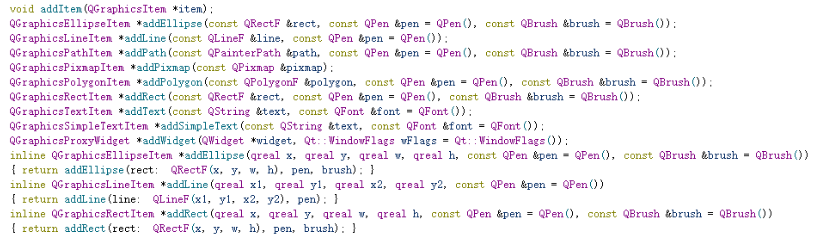


图2-1 addItem() 系列函数

(2) 查找图元：在大规模数据的情况下保证在极短的时间内查询到对应的图元，Qt使用的是索引算法，使用的是BSP（二进制空间分区）树。能够在极短的时间内在极大的数据规模中确定某个图元的位置，这也是QGraphicsScene的最大优势之一。

(3) 移除图元：removeItem()。

(4) 管理图元状态，处理图元选择和焦点处理等。

⚫ setSelectionArea()：可以传递一个形状范围来选择图元项目

⚫ clearSelection()：清除当前选择

⚫ selectedItems()：获取被选择的图元的列表

⚫ setFocusItem()、setFocus()：设置图元获取焦点

⚫ focusItem()：获取当前焦点

(5) 接受事件：用户通过QGraphicsView触发事件，并通过QGraphicsScene将事件转发到对应的图元：例如鼠标按下事件、移动事件、释放和双击事件，鼠标悬停事件、滚轮事件，键盘输入焦点和按键事件，拖拽事件等。



图2-2 事件转发系列函数

QGraphicsScene另一个重要的功能就是转发QGraphicsView的事件，通过一系列操作，例如通过鼠标点击的坐标计算出到底是选中了哪个图元，键盘事件对应的哪些图元具有焦点等，能够将这些事件转发给对应的图元，最后进入真正的事件循环进行处理。

**2、QGraphicsView**

QGraphicsView类真正提供可视化QGraphicsScene的内容的功能，它在一个可滚动的viewport之内将一个QGraphicsScene中的内容实现可视化。

QGraphicsView主要功能包括但不限于：

(1) 设置可视化操作的属性：QGraphicsView中提供了大量可设置的属性用以指示在实现可视化操作时的各种具体事项，如RenderHints提供参数初始化用于绘制的QPainter，Alignment提供当前视图中所绘制的场景的对齐方式。

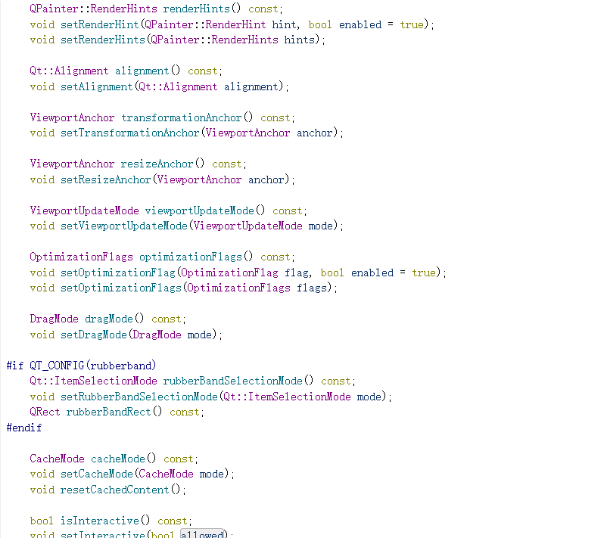


图2-3 设置可视化操作的接口

(2) 对场景（Scene）进行可视化与视觉效果调整：QGraphicsView对象的成员方法render对场景进行可视化的绘制呈现在viewport中，并提供了一系列方法对viewport整体的视觉效果进行调整，如centerOn()方法将滚动viewport中的内容以确保场景坐标pos在视图居中，fitInView()方法将缩放并滚动viewport中的内容使得场景内的矩形区域rect铺满当前viewport。

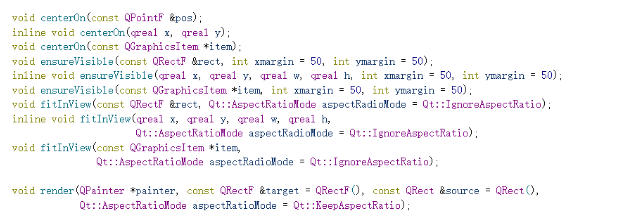


图2-4 进行可视化与视觉效果调整的接口

(3) 管理“场景（Scene）坐标”与“视图（View）坐标”之间的数学关系，并提供方法对视图内容施行各种坐标变换。QGraphicsScene对象当中的各个图元有其在QGraphicsScene中的坐标即“场景坐标”，它们代表了各个图元在QGraphicsScene中的位置信息；而QGraphicsView对各个图元进行绘制以及调整变换时则是通过由自身管理的“视图坐标”，它们代表了各个要绘制的图形在viewport中的位置信息。QGraphicsView可以由用户设置“场景坐标”向“视图坐标”变换的方式，对viewport实现旋转、伸缩等坐标变换，同时由于QGraphicsView的绘图使用“视图坐标”，因此这个过程不会干扰图元自身的“场景坐标”。此外还提供mapToScene/mapFromScene()方法供用户调用实现这两种坐标之间的数学换算。

(4) 接受鼠标和键盘的事件，并通过处理传递给QGraphicsScene对象，进而通过索引算法转发给对应的图元。

**3、QGraphicsItem**

QGraphicsItem是所有图元的基类，可以派生出各种典型的形状（例如矩形、椭圆、文本等）和自定义的形状。

QGraphicsItem主要功能包括但不限于：

(1) 接受QGraphicsScene传递的事件：进行事件处理，例如鼠标按下、移动、释放和双击事件，鼠标悬停、滚轮事件，键盘输入焦点和按键事件，拖拽事件等。



图2-5 QGraphicsScene传递的事件的方法

(2) 坐标系统

每个Item都有自己的本地坐标系，一般以自身的中心为(0, 0)，自身的方向作为基准方向建立，多个Item的情况如图所示。因此需要通过某些机制将不同Item的坐标联系在一起。

为了统一方便的管理，引入parent-child的关系链。每个对象的变换都依赖于其父对象的坐标。[4]子对象的pos()接口返回的是其在父对象坐标系统中的坐标。子对象的坐标处理是首先通过父对象不断向上传递，最终得到一个真实的坐标。同理，父对象的坐标变换也会同理影响到子对象的真实位置（批量处理），但是注意子对象存储的坐标没有变化，这样就非常好维护了。

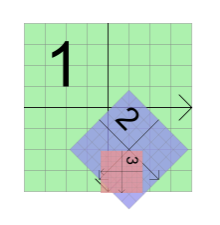


图2-6 坐标变换示意图

(3) 坐标变换

QGraphicsItem除了提供基本位置pos()以外，还支持坐标变换，例如setRotation()旋转，setScale()缩放等。

同理，父对象的变换会影响子对象，例如父对象顺时针转90度，子对象会跟着一起转90度。

(4) 提供分组功能（后续考虑）

通过前面的坐标系统可以知道，每个QGraphicsItem都有一个父对象，也可以有一系列子对象，这是类似于对象树机制的在构造的时候确定的关系。当然也可以手动分组。QGraphicsItemGroup是一个特殊的派生类，该类记录了一系列的图元为一个组，在该组的所有图元通过Group调用的时候移动、事件处理等都会进行批量处理。

(5) 提供编写自定义图元的接口（后续考虑）

创建一个QGraphicsItem的子类，然后覆写两个纯虚函数boundingRect()返回该图元项目所绘制区域的估计值，paint()实现实际绘图。

boundingRect()返回的是一个QRectF类型，将该图元的外部估计边界定义为矩形，这个方法也可以为不同Item的范围做大致的估算，可以被其他的方法所调用，省去一些暴力查找的过程。当然对于真正的矩形boundingRect()可以返回精确的范围，对于其他的曲线或者不规则的形状只能做大致的范围。

(6) 碰撞检测（后续考虑）

通过shape()函数和collidesWith()这两个虚拟函数，可以支持碰撞检测。shape()函数返回一个局部的坐标QPainterPath。目前没有细节调研QPainterPath，只简单知道QPainterPath记录了绘图的路径，比如2D图形的形状是由某些直线、曲线等构成的，通过这个能够确定图形的形状。

QGraphicsItem会根据默认的shape()函数自动处理碰撞检测，实现合理的效果，比如在碰撞区域应该如何进行绘制。如果用户想要定义自己的碰撞检测，可以通过collidesWith()实现。

(7) 图元顺序（后续考虑）

难免会发生两个图元的范围出现重叠的情况。合理处理顺序决定了鼠标点击的时候哪些场景会接受鼠标事件。一个比较合理的想法是子对象位于父对象的顶部，而同级对象之间按照定义的顺序进行堆叠。例如添加对象A、B，那么对象B位于A的顶部。这是比较符合自然逻辑的，Qt也是这样做的。

Qt提供了一些可以更改项目的排序方式的接口。例如可以在一个图元项目上调用setZValue()方法，以将其显式堆叠在其他同级项目之上或之下。项的默认Z值为0，具有相同Z值的项按插入顺序堆叠。还可以设置ItemStacksBehindParent标志以将子项堆叠在其父项之后。

### 2.1.3 2D BSP 树在 QGraphicsScene 中的应用

前面提到需要非常高的查询效率，而Qt中使用的是高度特化的2D BSP树进行处理的。[5]BSP树构造一个n维空间到凸子空间的分层细分（a BSP tree is a heirarchical subdivisions of n dimensional space into convex subspaces）。每个节点都有一个前叶子节点和后叶子节点。从根节点开始，所有后续插入都由当前节点的超平面划分。在二维空间中，超平面是一条线。在3维空间中，超平面是一个平面。BSP树的最终目标是让超平面的分布情况满足“每个叶节点都在父节点超平面的前面或后面”（The end goal of a BSP tree if for the hyperplanes of the leaf nodes to be trivially "behind" or "infront" of the parent hyperplane.）。

BSP树对于与静态图像的显示进行实时交互非常有用。在渲染静态场景之前，需要计算BSP树。可以非常快地（线性时间）遍历BSP树，以去除隐藏的表面和投射阴影。通过一些工作，可以修改BSP树以处理场景中的动态事件。

下面是在对象空间构建BSP树的过程：

(1) 首先，确定要划分的世界区域以及其中包含的所有多边形。为了方便讨论，这里将使用一个二维世界。

(2) 创建一个根节点L，该节点本身对应一个分区超平面(在二维世界中，分区超平面就是直线)；同时这个节点维护一个多边形列表，在节点刚刚创建时，多边形列表中保存着它对应的超平面所划分的目标区域(对 L 而言，它对应的直线所划分的目标区域就是整个二维世界)当中的所有多边形。

(3) 使用L对世界进行分区，假设分为了两个区域A和B。在根节点上创建两个叶子节点分别对应A和B，并将世界中的所有多边形移动到A或B的多边形列表中。遵循以下规则：

对于世界中的每个多边形：

如果该多边形完全位于A区域，请将该多边形移动到A区域的节点列表中。

如果该多边形完全位于B区域，请将该多边形移动到B区域的节点列表中。

如果该多边形与L相交，则将其拆分为两个多边形，并将它们移动到A和B的相应多边形列表中。这种情况下，算法必须找到多边形与分割线L的交点，以确定多边形的哪一部分位于哪个区域。

如果该多边形与L重合(也就是说，恰好是一条位于L上的线段)，保持其仍位于节点L处的多边形列表中。

在上述步骤完成后，节点A和B的多边形列表已经生成，但是A和B尚未完成创建分区超平面并划分的步骤，也就是说，对于当前的节点A，它的状态与步骤1中的“整个二维世界”一致(区域确定，包含的多边形确定)，只不过区域大小有差别，B同理。

(4) 在A和B上继续划分其所对应的二维区域，并将上述算法递归地应用于A和B的多边形列表。

最终的切分效果图如图2-7所示：

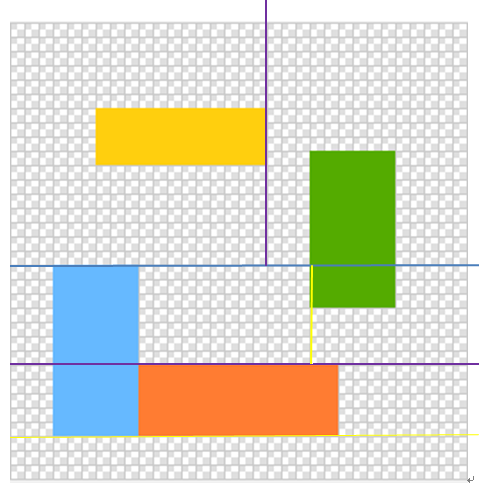


图2-7 切分效果图

### 2.2 一些关于空间数据结构的简单研究与实现

此部分紧接着上面的Qt Graphics View Framework，详细探讨查询算法的不同可能实现以及对应优劣情况。

### 2.2.1 前言

首先，需要来了解一下计算机中的“视图”“图形”等这些概念。在日常的语境当中，当提到“视图”、“照片”、“图形”等等这些和图像有关的概念时，一般都认为它单纯指代一个具有视觉效果的画面，比如高考卷子上立体几何的那个题，会把它对应到“图形”的概念上；相册里面某张具有回忆性质的照片，自然会把它对应到“照片”的概念上，等等等等。不过在计算机中，“视图”“图像”这些概念的涵义还包含了其他的方面。对于一个图像，计算机所要关注的不止是它呈现的视觉效果，还要使用合适的数字格式存储图像，并按照用户的要求对图像进行分析、处理和加工。[6]其实可以把计算机中的“视图场景”看作一个“虚拟的视图场景”：它在概念上更多地指代的是对图像的数字格式存储以及处理加工的相关操作。不妨回想一下前面叙述“空间数据结构”的时候所说的“保存图元信息”“定位查找元素”这些关键词，就对应的是这样的概念，很好理解，数字格式便是图像在计算机中的存在形式。这也是前面为什么要用“虚拟”一词，在计算机中说到“图像”“视图”等等时，不应直接以日常的语境将它理解成一幅纯粹的“视觉画面”，而是要关注到计算机中实际存储并操作的对象——“数字格式的图像”。因此，下面的内容当中也都基于上述概念，讨论的是“数字格式的图像”。

假如现在要设计一个视图场景(Scene)，在其中包含很多图元对象(Item)。就像这样:



图2-8 视图场景实例

上面就是一个场景，其中包含了大量的线条与色块等等，这些所有的图形要素都统称为"图元"。而场景就负责管理并绘制图元、接收并向对应的图元转发UI事件。假设现在场景是静态场景，也就是场景中的图元在初始化以后就不会有任何改变。

现在的需求是想让这个场景当中的图元响应用户的UI操作，比如说，用鼠标点击场景中的某个位置，场景就需要找到有哪根线条覆盖到了那个位置，然后让那个线条按照某种规则动起来。虽然在实现这个逻辑的时候是最好希望能有一个“点信息表”可以根据鼠标点下的位置直接检索出对应的点上有哪些图元覆盖到这样的相关信息，但场景保存的是“各个图元”相关的信息成一个“图元信息表”，而不太可能把“每个点”相关的信息保存成“点信息表”。如果想用“点”来衡量一整个视图场景的大小，或者说想要保存“每个点被哪些图元覆盖到”这样的信息，需要一个巨大规模的二维数组，那几乎是不可能的。所以当知道用户的鼠标落在哪个“点”上之后，要做的是去“图元信息表”寻找“哪些图元覆盖了这个点”，毕竟，场景保存的是“各个图元”的“图元信息表”，不会暴力到去把“每个点”的信息保存成一个“点信息表”。

如果使用最原始的、最暴力的算法，对图元信息表中的每个图元遍历一次，看看哪个图元覆盖了那个位置。这样是完全可以的，但是必须考虑的事情是，效和性能是非常重要的。宇宙是有限的，连一个葛立恒数大小的物理概念在现实的宇宙中都找不到，那就不能在讨论一个事情的时候抛开它的时空限制、可用资源限制这些事实层面的东西不谈。比如说，现在场景当中有一千万个图元(这种情况当然有可能出现，比如说卫星地图或者高品质游戏等一些高精绘图场景)，那用户每点一下就要暴力搜索数秒，如果这景象出现在一个游戏里那它早已被市场淘汰。但是前面又提到，“遍历图元”这样的操作确实是逃不掉的，那么有什么能提高效率的方法呢？

计算机图形学对此问题早有研究，并提出了“空间数据结构”(Spatical Data)的概念。很多时候需要能够方便地在空间中定位和查找元素的数据结构来处理物体，这称为空间数据结构。空间数据结构将空间划分为多个层次多个区域，并在保存图元信息时使用对应的数据结构记录每个划分出的区域中完全或部分包含的图元并保存，这样就更方便定位和查找空间中的元素，被广泛用在图形学场景中用来加快运算。例如，现在场景中有一千万个图元，原本在遍历的时候需要对这一千万个图元都遍历，但是现在有一种空间数据结构将整个场景不重不漏地划分成了1000个区域，那现在平均每个区域就只有一万个图元了，而用户点到的那个位置一定只落在某个区域当中，所以现在只需要遍历一万个左右的图元了。当然这只是一个比较理想化的模型，实际操作起来还会遇到其他的细节问题，比如说如果有很多图元它就是比你能划出来的区域更大该怎么办呢。而下面所要探讨的，就是均匀网格、BSP这两种空间数据结构在二维空间下的情形以及实现时的各种细节问题。

### 2.2.2 均匀网格的概念

均匀网格(Grid)是一种空间数据结构，它使用了一个最为简单朴素的做法，就是将一个空间均匀地划分为大小相等的网格。在二维空间中，均匀网格在一个平面区域内使用等距的直线将其划分为大小相等的网格子区域。把空间划分成均匀网格，使用数组记录每个网格中包含的图元，就形成了一个简单的空间数据结构。下图展示了一个二维均匀网格的示例：

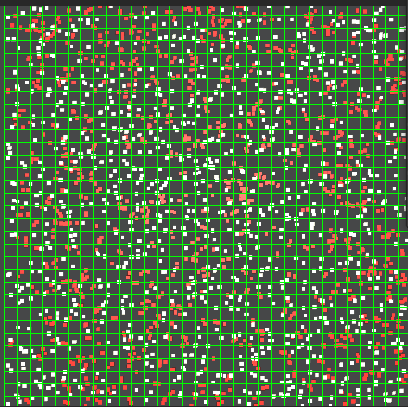


图2-9 二维均匀网格的实例

### 2.2.3 BSP的概念

BSP (Binary Space Partitioning，空间二叉划分)也是一种空间数据结构，它可以对一个二维或三维空间进行划分，这里探讨的是二维空间的情形。每次将空间划分为两个部分并对每个划分出的子空间递归重复这个过程，然后使用树结构将空间组合起来，最后得到的就是一棵二叉树，这棵二叉树的每个叶节点对应一块区域，每个非叶节点对应一次划分所用的分割线。这个道理其实很容易想到，因为在进行划分的过程中，所有非叶节点都经历了派生出子节点的过程，也就是说它们经历了“从叶节点变成了非叶节点”的过程，这个过程自然会让这个节点的角色发生改变：当一个节点还没有派生子节点时，它在这棵尚未完成的树上是一个叶节点，代表了一个区域；而当它派生出子节点后，它自己就变成了非叶节点，那么它现在就代表了一条分割线，其实很好理解，从一个原本的叶节点“派生子节点”对应的操作就是对原本节点代表的区域进行划分，在划分后这个节点就代表了分割线，而它派生的子节点现在就代表分出的两个子区域，当然，子节点也可以继续对自己进行上面这个操作。

一种简单的BSP划分方式是“轴向划分”，它每次划分都简单地在每个子空间的对称轴上进行划分，对于矩形场景而言，对称轴就是每个子空间水平或垂直的中线轴，而且这样划分出的子空间同样是矩形，对这些子空间也依然从对称轴上进行划分。通常来说，轴向划分最终形成一棵满二叉树：所有非叶节点的度都是2，所有叶节点都在同一层次上，也就是说每一个同级的子区域都会进行划分，直到达到所要求的树深度。我们在划分区域时一般都采取“横竖交替”的策略，如果一个节点是按照水平中线轴划分，那么它所派生的子节点再划分的时候就是按照垂直中线轴划分。

但是这样设计以后会发现，轴向划分最终得到的空间数据结构与网格无异。实际上，BSP划分时可以使用任意位置、任意方向的分割线，轴向划分是BSP最简单的划分方式，而在很多实际应用当中，都会基于图元的形状大小以及位置等信息而使用更加灵活的分割线，以使得尽可能多的图元都只落在一个子区域当中，能够提升运算的性能。

### 2.2.4 简单的研究与实现

**1、QT的BSPTree**

首先是看有名的图形界面框架Qt。Qt中的视图场景QGraphicsScene使用的是轴向划分BSP：

1. void QGraphicsSceneBspTree::initialize(const QRectF &rect, int depth, int index)
2. {
3. Node \*node = &nodes[index];
4. if (index == 0) {
5. node->type = Node::Horizontal;
6. node->offset = rect.center().y();
7. }
8. if (depth) {
9. Node::Type type;
10. QRectF rect1, rect2;
11. qreal offset1, offset2;
12. if (node->type == Node::Horizontal) {
13. type = Node::Vertical;
14. rect1.setRect(ax: rect.left(), ay: rect.top(), aaw: rect.width(), aah: rect.height() / 2);
15. rect2.setRect(ax: rect1.left(), ay: rect1.bottom(), aaw: rect1.width(), aah: rect.height() - rect1.height());
16. offset1 = rect1.center().x();
17. offset2 = rect2.center().x();
18. } else {
19. type = Node::Horizontal;
20. rect1.setRect(ax: rect.left(), ay: rect.top(), aaw: rect.width() / 2, aah: rect.height());
21. rect2.setRect(ax: rect1.right(), ay: rect1.top(), aaw: rect.width() - rect1.width(), aah: rect1.height());
22. offset1 = rect1.center().y();
23. offset2 = rect2.center().y();
24. }
25. int childIndex = firstChildIndex(index);
26. Node \*child = &nodes[childIndex];
27. child->offset = offset1;
28. child->type = type;
29. child = &nodes[childIndex + 1];
30. child->offset = offset2;
31. child->type = type;
32. initialize(rect: rect1, depth: depth - 1, index: childIndex);
33. initialize(rect: rect2, depth: depth - 1, index: childIndex + 1);
34. } else {
35. node->type = Node::Leaf;
36. node->leafIndex = leafCnt++;
37. }
38. }

nodes数组与index参数用于保存二叉树的信息，如何保存二叉树不是这里探讨的重点。看看这个二叉树是如何生成的：

由用户指定初始的rect和depth，rect参数是矩形对象用于指定BSP所要划分的区域，depth参数用于指定二叉树的深度。二叉树节点Node为如下的数据结构:

1. struct Node
2. {
3. enum Type { Horizontal, Vertical, Leaf };
4. union {
5. qreal offset;
6. int leafIndex;
7. };
8. Type type;
9. };

这里使用了一个非常巧妙的设计：非叶节点对应的是分割线，但是其实并不需要保存“一条线”下来，由于使用轴向划分的BSP，因此一条分割线只需要它的分割方向(水平/竖直)和它距离场景原点坐标轴的距离就可以确定位置，所以非叶节点保存的是type(Horizental/Vertical)以及当前分割线与type对应坐标轴之间的距离offset。叶节点的type是Leaf，它对应一个区域，区域的信息同样不保存在Node本身当中，而是将每个区域中所包含的图元的信息保存在另一个单独的数组leaves中，叶节点的Node保存一个leafIndex作为leaves数组的下标，这样叶节点对应的区域就是leaves[leafIndex]。由于一个节点只可能是叶节点或非叶节点之一，所以使用union联合offset和leafIndex。

生成二叉树的过程递归调用initialize()函数，递归调用的参数rect是本次调用后切出的子区域，depth每次递归都减1。在函数中判断depth，当depth为0时表示已经到达目标深度，则当前的节点为叶节点，type设为Leaf且记录leafIndex；否则当前的节点为非叶节点，并且根据当前节点的type和rect计算出下层节点(child)在递归调用时的参数：下层节点的type应与当前节点的相反，下层节点的rect是当前分割线所分出的两个子区域。

**2、BSP**

这里的BSP就是在吸收Qt的经验以后自己实现的BSP结构了。相关思路同上，不再赘述。

BSP的数据结构为Node所串联成的二叉树。Node的结构如下：

1. */\*\**
2. \* @enum SplitType
3. \* @brief 枚举区域分割的类型。
4. \*/
5. enum SplitType
6. {
7. Horizontal = 0, *///< 水平分割*
8. Vertical,       *///< 竖直分割*
9. Leaf            *///< 叶子节点*
10. };
11. */\*\**
12. \* @class Node
13. \* @brief 树的节点，对应分割形成的某块区域。
14. \*/
15. struct Node
16. {
17. */\*\**
18. \* @brief 默认构造。
19. \*/
20. Node() = default;
21. */\*\**
22. \* @brief 析构函数，处理本节点以及子节点的释放。
23. \*/
24. ~Node();
25. */\*\**
26. \* @brief 该节点的分割类型。
27. \*/
28. SplitType m\_splitType;
29. */\*\**
30. \* @details offset 和 leafIndex 分别对应非叶子节点和叶子节点的数据信息，对于同一个节点这两条数据不可能共存。因此为了节省内存，采用 union
31. \*/
32. union
33. {
34. */\*\**
35. \* @brief 分割线的横坐标或纵坐标的偏移量，是横坐标还是纵坐标取决于分割类型。
36. \*/
37. int m\_offset;
38. */\*\**
39. \* @brief 节点的下标，在外部的 leaves 数组中使用。
40. \*/
41. int m\_leafIndex;
42. };
43. */\*\**
44. \* @brief 左子节点指针。
45. \*/
46. struct Node \*m\_left = nullptr;
47. */\*\**
48. \* @brief 右子节点指针。
49. \*/
50. struct Node \*m\_right = nullptr;
51. };
52. Node::~Node()
53. {
54. if (!m\_left && !m\_right) return;
55. delete m\_left;
56. m\_left = nullptr;
57. delete m\_right;
58. m\_right = nullptr;
59. }

BSP包含的数据成员如下：

1. */\*\**
2. \* @brief 整棵 BSP 树的根节点。
3. \* @todo 后续考虑自己实现简单的对象树机制，不使用智能指针
4. \*/
5. Node \*m\_root = nullptr;
6. */\*\**
7. \* @brief 存储每个叶子节点中的 PicItem (图元) 列表。
8. \* @details 树构建成功以后，所有的 PicItem 都存储在叶子节点的区域中，为了方便获取，将数据提取到整棵树的数据结构中，叶子节点中存储下标方便访问。注意，每个"Vector<PicItem \*>"对应一个节点，因此m\_leaves实际上以一维的方式组织各个节点。
9. \*/
10. Vector<Vector<PicItem \*>> m\_leaves;
11. */\*\**
12. \* @brief 树的深度，对应分割的次数。
13. \*/
14. int m\_depth = 0;
15. */\*\**
16. \* @brief 整棵树作用的 2D 平面范围。
17. \*/
18. Rect m\_region;

由用户传入矩形区域region与树的深度depth，程序传入初始根节点并递归调用init函数创建各个子节点。

注：本代码中Rect对象的x1()和y1()返回矩形对象左上角的坐标，x2()和y2()返回矩形对象右下角的坐标，下同。

1. void init(Node \*node, const Rect &region, int depth)
2. {
3. *// depth > 0 ，继续向下分割*
4. if (depth > 0)
5. {
6. *// 为了统一命名，使用 left/right 对应逻辑上的 左/右 子节点*
7. *// 水平 Horizontal ： left 为上半边， right 为下半边*
8. *// 垂直 Vertical ： left 为左半边， right 为右半边*
9. int offsetLeft = 0, offsetRight = 0;
10. Rect leftRect, rightRect;
11. SplitType newSplit;
12. if (SplitType::Horizontal == node->m\_splitType)
13. {
14. *// 当前节点为水平分割 Horizontal ，则子节点为 Vertical ， left 为上半边， right 为下半边*
15. newSplit = SplitType::Vertical;
16. leftRect = Rect(region.x1(), region.y1(), region.width(), region.height() / 2);
17. rightRect = Rect(leftRect.x1(), leftRect.y2(), region.width(), region.height() / 2);
18. offsetLeft = leftRect.x1() + leftRect.width() / 2;
19. offsetRight = rightRect.x1() + rightRect.width() / 2;
20. }
21. else
22. {
23. *// 当前节点为垂直分割 Vertical ，则子节点为 Horizontal ， left 为左半边， right 为右半边*
24. newSplit = SplitType::Horizontal;
25. leftRect = Rect(region.x1(), region.y1(), region.width() / 2, region.height());
26. rightRect = Rect(leftRect.x2(), leftRect.y1(), region.width() / 2, region.height());
27. offsetLeft = leftRect.y1() + leftRect.height() / 2;
28. offsetRight = rightRect.y1() + rightRect.height() / 2;
29. }
30. node->m\_left = new Node;
31. node->m\_left->m\_splitType = newSplit;
32. node->m\_left->m\_offset = offsetLeft;
33. node->m\_right = new Node;
34. node->m\_right->m\_splitType = newSplit;
35. node->m\_right->m\_offset = offsetRight;
36. init(node->m\_left, leftRect, depth - 1);
37. init(node->m\_right, rightRect, depth - 1);
38. }
39. *// 遇到叶子节点*
40. else
41. {
42. node->m\_splitType = SplitType::Leaf;
43. node->m\_leafIndex = m\_leaves.size();
44. m\_leaves.append(Vector<PicItem \*>());
45. }
46. }

查询是空间数据结构应当提供的基本用途，根据用户传入的区域查询当前空间数据结构中该传入区域命中的子区域(在BSP中，是叶节点)；在查询的基础上，用户或程序可以对命中的区域或对应区域中的图元执行操作，这个操作可以是在区域中增删图元、对区域中指定特征的图元进行进一步查询，等等等等。

1. */\*\**
2. \* @brief 定义回调函数类型，用于对叶节点执行操作
3. \*/
4. using Visitor = std::function<void(LList<LCanvasItem \*> &)>;
5. *// 以addItem为例*
6. void addItem(LCanvasItem \*item)
7. {
8. auto func = [&item](LList<LCanvasItem \*> &items)
9. {
10. items.append(item);
11. };
12. update(func, m\_root, item->boundingRect());
13. }
14. */\*\**
15. \* @brief 根据所给的区域查询命中的叶子节点，并执行指定的操作。
16. \* @param visitor 函数对象。用于对查找到的叶子节点执行操作
17. \* @param node 根节点
18. \* @param rect 目标区域矩形
19. \*/
20. void update(const Visitor &visitor, Node \*node, const Rect &rect)
21. {
22. if (m\_leaves.isEmpty()) return;
23. switch (node->m\_splitType)
24. {
25. case SplitType::Leaf:
26. visitor(m\_leaves[node->m\_leafIndex]);
27. break;
28. case SplitType::Vertical:
29. {
30. if (rect.x1() < node->m\_offset)
31. {
32. update(visitor, node->m\_left, rect);
33. if (rect.x2() >= node->m\_offset) update(visitor, node->m\_right, rect);
34. }
35. else
36. {
37. update(visitor, node->m\_right, rect);
38. }
39. break;
40. }
41. case SplitType::Horizontal:
42. {
43. if (rect.y1() < node->m\_offset)
44. {
45. update(visitor, node->m\_left, rect);
46. if (rect.y2() >= node->m\_offset) update(visitor, node->m\_right, rect);
47. }
48. else
49. {
50. update(visitor, node->m\_right, rect);
51. }
52. break;
53. }
54. }
55. }

**3、Grid**

均匀网格的数据成员如下：

1. */\*\**
2. \* @brief 存储每个网格中的 PicItem 列表。
3. \* @details 树构建成功以后，所有的 PicItem 都存储在网格的区域中，为了方便获取，将数据提取到整个网格的数据结构中，通过数学计算得出下标方便访问。注意，每个"Vector<PicItem \*>"对应一个网格，因此m\_grids实际上以"一维数组排列二维网格"的方式组织各个网格。
4. \*/
5. Vector<Vector<PicItem \*>> m\_grids;
6. */\*\**
7. \* @brief 网格分割出的每边的区间个数。
8. \*/
9. int m\_sections = 0;
10. */\*\**
11. \* @brief 整个网格作用的 2D 平面范围。
12. \*/
13. Rect m\_region;

与BSP不同的是，BSP有代表节点Node的“单元结构”，而均匀网格当中没有设置“代表网格单元结构”Grid。这是基于两种空间数据结构的基本用途“查询”逻辑上的不同：BSP树本身的结构和内容就决定了在查询时必须与非叶节点的分割线比较才能找到最终位于叶节点的子区域，树结构的特性就是不提供直接访问任何特定叶节点的快速途径，访问树中的指定节点必须从根节点开始，无法避开各个Node之间的逻辑结构(即使用数组保存它也是如此)，也因此在BSP树中需要保存每条分割线的信息；而Grid本身的结构与数组对应，并且均匀划分使其在查询时能够通过数学计算的方式直接获知子区域的数组下标，因此就不需要专门去保存每个网格本身的信息，只需要将每个网格对应的图元列表保存为“图元列表的数组”作为查询时访问的目标。

均匀网格的成员函数通过用户传入的目标区域与分割线数量进行初始化：

1. */\*\**
2. \* @brief 带参构造。
3. \* @param region 需要作用的区域
4. \* @param splitNum 网格的分割线数量（经纬两个方向分割线数量相同）
5. \*/
6. Grid::Grid(const Rect &region, int splitNum) : m\_region(region), m\_sections(splitNum + 1), m\_grids(Vector<Vector<PicItem \*>>((splitNum + 1) \* (splitNum + 1))) {}

均匀网格的m\_grids在初始化时规定好大小也就是网格数量，不过此时并不存储任何内容，因为初始化时还没有图元。与BSP不同的是，BSP中init函数进行了划分区域、创建各个节点等操作，而均匀网格没有这一步骤。这也是因为Grid不需要保存每个网格的信息。

根据用户传入的区域查询当前空间数据结构中该传入区域命中的子区域(在均匀网格中，是数组下标)；在查询的基础上，用户或程序可以对命中的区域或对应区域中的图元执行操作，这个操作可以是在区域中增删图元、对区域中指定特征的图元进行进一步查询，等等等等。

1. void update(const Visitor &visitor, const Rect &rect)
2. {
3. int x1Index = rect.x1() < m\_region.x1() ? 0 : rect.x1() \* m\_sections / m\_region.width();
4. int y1Index = rect.y1() < m\_region.y1() ? 0 : rect.y1() \* m\_sections / m\_region.height();
5. int x2Index = rect.x2() >= m\_region.x2() ? (m\_sections - 1) : rect.x2() \* m\_sections / m\_region.width();
6. int y2Index = rect.y2() >= m\_region.y2() ? (m\_sections - 1) : rect.y2() \* m\_sections / m\_region.height();
7. for (int y = y1Index; y <= y2Index; ++y)
8. {
9. for (int x = x1Index; x <= x2Index; ++x)
10. {
11. visitor(m\_grids[y \* m\_sections + x]);
12. }
13. }
14. }

### 2.3 在X11下使用Cairo引擎绘制图形

LarkSDK目前的绘制引擎是最基础的暴力软渲染，因此在绘制数据规模较大和较复杂的图形的时候，会出现绘制瓶颈。这显然是不得不重视的问题。一个很好的解决办法是将第三方绘制引擎引入。而cairo是一个方便和高性能的第三方C库。它可以作为绘制引擎，帮助用户绘制各种图形，并且提供多种输出方式。因此这里将调研如何在Linux下结合X11图形显示协议绘制简单的图形。

效果图如下：

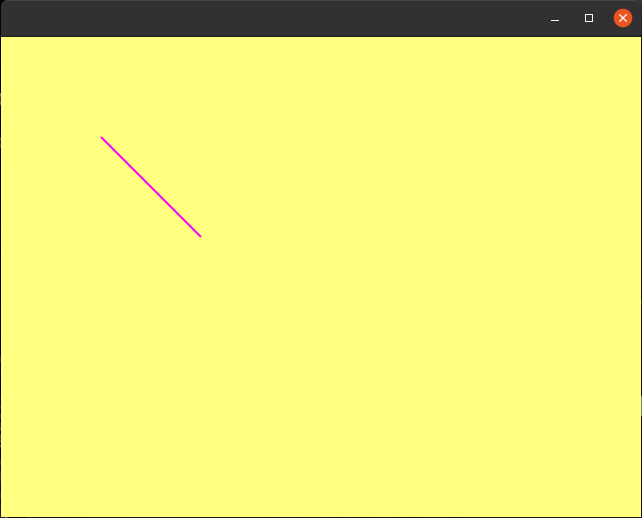


图2-10 cairo引擎绘制的示例图

### 2.3.1 安装Cairo库

cairo的官方网站是[https://cairographics.org](https://cairographics.org/)。其中对cairo图形库做了一个完整的介绍，包括如何下载、API接口、示例等等。

**1、通过包管理器安装**

通过官方文档知道，在Linux下可以直接通过包管理器进行下载，以Ubuntu为例。

1. sudo apt install libcairo2-dev

下载好以后头文件和动态库就安装好了。头文件安装在/usr/include/cairo/中，静态库和动态库分别位于/usr/lib/x86\_64-linux-gnu/libcairo.a和/usr/lib/x86\_64-linux-gnu/libcairo.so。因此能直接被系统识别，直接引入头文件，编译的时候链接cairo库即可。

**2、通过Conan安装**

对于个人用户而言，apt安装自然是非常友好的。[7]但是对于LarkSDK这样一个面向用户的基础框架而言，除非是最基本的系统库例如X窗口系统libx11-dev，Wayland窗口系统libwayland-dev和Wayland键盘处理libxkbcommon-dev等，其他的第三方库均最好不以apt包的方式引入。鉴于C++没有自己的包管理器，因此需要借助第三方的包管理器，例如conan，cpm等。本项目使用conan管理第三方包。

conan是一个使用python语言编写的C++的包管理器，官方网址是<https://conan.io/>。conan上整合了众多第三方的C++库，通过命令行操作就能方便的在自己的项目中引入conan包。需要通过conanfile.py或者conanfile.txt进行配置，具体见文档<https://docs.conan.io/2/>。

conan官方提供了自己的conan仓库：<https://conan.io/center>。在上面能找到很多市面上熟知的第三方库，例如gtest，qt，boost，fmt等。当然还有一些更基础的工具库，这里不赘述。幸运的是[cairo](https://conan.io/center/recipes/cairo)库也在其中。

现在尝试安装cairo库并尝试用CMake将其引入。在新开的项目中创建conanfile.txt，引入项目依赖cairo。使用conanfile.py主要用于生成和发布用户自己的conan包，conan提供了丰富的选项供用户操作。这里只是为了测试，因此使用最简单的conanfile.txt即可。

1. [requires]
2. cairo/1.18.0
3. [generators]
4. cmake

编写自己的CMakeLists.txt，配置项目的相关信息，引入conan的部分类似如下：

1. ...
2. include (${PROJECT\_BINARY\_DIR}/conanbuildinfo.cmake)
3. conan\_basic\_setup (NO\_OUTPUT\_DIRS)
4. ...
5. add\_executable (xxx
6. ...
7. )
8. target\_link\_libraries (xxx ${CONAN\_LIBS})

之后执行一般的构建流程即可。

1. conan install ..
2. cmake ..
3. make # windows 下默认没有 make 命令，使用 cmake --build ./ 代替

conan官方的包很多，很全，但是conan本身还有很多bug，单就cairo包的使用过程中就有很多问题。最典型的，执行conan install .. 可能会失败，并且遇到很多错误。因此首先需要了解conan包是个什么东西，才能明确问题是如何形成的。

现在对conan包做一个简述。对于C++库而言，为了让用户方便的使用，把.h和.cpp代码全部发布给用户是不合适的，这些代码不应该在用户的机器上再被编译一次，而应该在需要用的时候被直接使用。因此需要通过库的方式进行发布，也就是使用静态库和动态库。在发布的包中，最重要的文件就是头文件和库文件。当然可能会携带一些其他必要的文件，例如资源文件、版本说明文件等。

我们都知道，编译C/C++的代码需要依赖于C++和编译器的版本。[8]进一步的，由于C/C++是非常接近底层的代码，虽然标准库是跨平台的，但是如果需要写平台相关的程序，还需要注意操作系统的版本。因此，对于同一个版本的conan包，不同的系统，不同的编译环境，是静态库还是动态库，是Debug包还是Release包，甚至依赖包的版本，都可能对最后的编译造成一定影响。在本地的conan配置中会保存相关的这些信息，在conan拉包的时候会匹配本地的配置拉取合适的包。配置文件默认位于~/.conan/profile/default中。

在Linux下的配置类似于这样，看其参数很明显能知道对应的含义。

1. [settings]
2. os=Linux
3. os\_build=Linux
4. arch=x86\_64
5. arch\_build=x86\_64
6. compiler=gcc
7. compiler.version=9
8. compiler.libcxx=libstdc++11
9. build\_type=Release
10. [options]
11. [build\_requires]
12. [env]

在Windows下类似于这样：

1. [settings]
2. os=Windows
3. os\_build=Windows
4. arch=x86\_64
5. arch\_build=x86\_64
6. compiler=Visual聽Studio
7. compiler.version=16
8. build\_type=Release
9. [options]
10. [build\_requires]
11. [env]

现在以cairo/1.18.0@包为例，使用conan命令查看其远端的包的列表。

1. conan search cairo/1.18.0@ -r conancenter

得到的结果大致是这样：



图2-11 cairo的conan远端仓库的包列表

结果发现settings中的内容和profiles/default中的内容对应，这就是前面提到的匹配。例如图中是一个Mac下的apple-clang的13.0版本的静态库的Debug包。每一个包的options，settings和requires都会对最前面的Package\_ID产生影响，这是一个哈希计算值，具体如何影响和生成请参考<https://docs.conan.io/2/reference/binary_model/package_id.html>。当然这其中也有令人费解的地方，后面会提到。

前面提到，conan官方的包有问题，会导致在安装的时候出现错误。例如，我在安装的出现的错误如下：



图2-12 conan安装报错结果1

错误信息提示fontconfig的2.15.0的版本需要conan 1.60.4以上才能安装。由于公司使用的是conan 1.60.1，首先我想到的是conan版本不正确。深入研究后，发现的问题实际上出乎意料。

由于公司实际开发的conan版本不可能从1.60.1升到1.60.4，要升肯定一步到位到conan 2.0了，因此只能从conan包本身出发排查问题。首先查看conancenter中cairo/1.18.0@远端的包，检索符合当前系统和编译环境的。发现了有三个长的几乎一样的包：

1. ; package 1
2. Package\_ID: 703bcc640002869a53960c4449d3825ff8a103e6
3. [options]
4. fPIC: True
5. shared: False
6. tee: False
7. with\_fontconfig: True
8. with\_freetype: True
9. with\_glib: True
10. with\_lzo: True
11. with\_png: True
12. with\_symbol\_lookup: False
13. with\_xcb: True
14. with\_xlib: True
15. with\_xlib\_xrender: True
16. with\_zlib: True
17. [settings]
18. arch: x86\_64
19. build\_type: Release
20. compiler: gcc
21. compiler.version: 9
22. os: Linux
23. [requires]
24. brotli/1.1.0:b21556a366bf52552d3a00ce381b508d0563e081
25. bzip2/1.0.8:da606cf731e334010b0bf6e85a2a6f891b9f36b0
26. expat/2.6.0:c215f67ac7fc6a34d9d0fb90b0450016be569d86
27. fontconfig/2.15.0:b172ac37518ca059ccac0be9c3eb29e5179ecf1e
28. freetype/2.13.2:f1014dc4f9380132c471ceb778980949abf136d3
29. glib/2.78.3:06c63123a2bb8b6d3ea5dcae501525df32efb7b5
30. libelf/0.8.13:6af9cc7cb931c5ad942174fd7838eb655717c709
31. libffi/3.4.4:6af9cc7cb931c5ad942174fd7838eb655717c709
32. libmount/2.39:6af9cc7cb931c5ad942174fd7838eb655717c709
33. libpng/1.6.43:7929d8ecf29c60d74fd3c1f6cb78bbb3cb49c0c7
34. libselinux/3.5:6b0384e3aaa343ede5d2bd125e37a0198206de42
35. lzo/2.10:6af9cc7cb931c5ad942174fd7838eb655717c709
36. pcre2/10.42:647f8233073b10c84d51b1833c74f5a1cb8e8604
37. pixman/0.43.4:6af9cc7cb931c5ad942174fd7838eb655717c709
38. util-linux-libuuid/2.39.2:6af9cc7cb931c5ad942174fd7838eb655717c709
39. xorg/system:5ab84d6acfe1f23c4fae0ab88f26e3a396351ac9
40. zlib/1.3.1:6af9cc7cb931c5ad942174fd7838eb655717c709
41. Outdated from recipe: True
42. ; package 2
43. Package\_ID: 8098347825649d9fd3e21c49992446a2a2193ad4
44. [options]
45. fPIC: True
46. shared: False
47. tee: False
48. with\_fontconfig: True
49. with\_freetype: True
50. with\_glib: True
51. with\_lzo: True
52. with\_png: True
53. with\_symbol\_lookup: False
54. with\_xcb: True
55. with\_xlib: True
56. with\_xlib\_xrender: True
57. with\_zlib: True
58. [settings]
59. arch: x86\_64
60. build\_type: Release
61. compiler: gcc
62. compiler.version: 9
63. os: Linux
64. [requires]
65. brotli/1.1.0:b21556a366bf52552d3a00ce381b508d0563e081
66. bzip2/1.0.8:da606cf731e334010b0bf6e85a2a6f891b9f36b0
67. expat/2.5.0:c215f67ac7fc6a34d9d0fb90b0450016be569d86
68. fontconfig/2.14.2:b172ac37518ca059ccac0be9c3eb29e5179ecf1e
69. freetype/2.13.0:f1014dc4f9380132c471ceb778980949abf136d3
70. glib/2.78.0:06c63123a2bb8b6d3ea5dcae501525df32efb7b5
71. libelf/0.8.13:6af9cc7cb931c5ad942174fd7838eb655717c709
72. libffi/3.4.4:6af9cc7cb931c5ad942174fd7838eb655717c709
73. libmount/2.39:6af9cc7cb931c5ad942174fd7838eb655717c709
74. libpng/1.6.40:7929d8ecf29c60d74fd3c1f6cb78bbb3cb49c0c7
75. libselinux/3.5:6b0384e3aaa343ede5d2bd125e37a0198206de42
76. lzo/2.10:6af9cc7cb931c5ad942174fd7838eb655717c709
77. pcre2/10.42:647f8233073b10c84d51b1833c74f5a1cb8e8604
78. pixman/0.40.0:6af9cc7cb931c5ad942174fd7838eb655717c709
79. util-linux-libuuid/2.39:6af9cc7cb931c5ad942174fd7838eb655717c709
80. xorg/system:5ab84d6acfe1f23c4fae0ab88f26e3a396351ac9
81. zlib/1.3:6af9cc7cb931c5ad942174fd7838eb655717c709
82. Outdated from recipe: True
84. ; package 3
85. Package\_ID: a336bac291d8ec6a55c6257f3266f9a8760c7403
86. [options]
87. fPIC: True
88. shared: False
89. tee: False
90. with\_fontconfig: True
91. with\_freetype: True
92. with\_glib: True
93. with\_lzo: True
94. with\_png: True
95. with\_symbol\_lookup: False
96. with\_xcb: True
97. with\_xlib: True
98. with\_xlib\_xrender: True
99. with\_zlib: True
100. [settings]
101. arch: x86\_64
102. build\_type: Release
103. compiler: gcc
104. compiler.version: 9
105. os: Linux
106. [requires]
107. brotli/1.1.0:b21556a366bf52552d3a00ce381b508d0563e081
108. bzip2/1.0.8:da606cf731e334010b0bf6e85a2a6f891b9f36b0
109. expat/2.5.0:c215f67ac7fc6a34d9d0fb90b0450016be569d86
110. fontconfig/2.14.2:b172ac37518ca059ccac0be9c3eb29e5179ecf1e
111. freetype/2.13.2:f1014dc4f9380132c471ceb778980949abf136d3
112. glib/2.78.1:06c63123a2bb8b6d3ea5dcae501525df32efb7b5
113. libelf/0.8.13:6af9cc7cb931c5ad942174fd7838eb655717c709
114. libffi/3.4.4:6af9cc7cb931c5ad942174fd7838eb655717c709
115. libmount/2.39:6af9cc7cb931c5ad942174fd7838eb655717c709
116. libpng/1.6.40:7929d8ecf29c60d74fd3c1f6cb78bbb3cb49c0c7
117. libselinux/3.5:6b0384e3aaa343ede5d2bd125e37a0198206de42
118. lzo/2.10:6af9cc7cb931c5ad942174fd7838eb655717c709
119. pcre2/10.42:647f8233073b10c84d51b1833c74f5a1cb8e8604
120. pixman/0.42.2:6af9cc7cb931c5ad942174fd7838eb655717c709
121. util-linux-libuuid/2.39.2:6af9cc7cb931c5ad942174fd7838eb655717c709
122. xorg/system:5ab84d6acfe1f23c4fae0ab88f26e3a396351ac9
123. zlib/1.3:6af9cc7cb931c5ad942174fd7838eb655717c709
124. Outdated from recipe: True

这三个包的唯一区别在于requires不同，第一个包需要expat/2.6.0，第二个和第三个包需要expat/2.5.0。第二个包需要freetype/2.13.0，第三个包freetype/2.13.2。这就是唯一区别，然后生成了三个不同的哈希值，对应不同的package。

那么问题来了，conan install默认会读取本地的配置，但是本地配置不可能指定requires。不同的项目可能用的版本不同，这是很正常的事情，也不应该是由用户承担责任的地方。其次，conan install不能指定package\_id下载，因此下载的是哪一个包，完全就看conan官方怎么想了。非常不幸运的是，我需要第二个包，但是下载只能下载到第一个包。

那么如何解决这个问题呢？幸运的是，conan提供了conan download的方法，这个东西可以指定package\_id进行下载。那就简单了，先把cairo本包下载到本地，然后再指定这一套流程，由于本地有缓存，会跳过去远端拉取cairo包的这一步，这样所有本包和所有依赖就被顺利拉取下来了。

1. conan download cairo/1.18.0@:8098347825649d9fd3e21c49992446a2a2193ad4 -r conancenter

成功下载下来以后，再次执行conan install .. ，又出问题了。

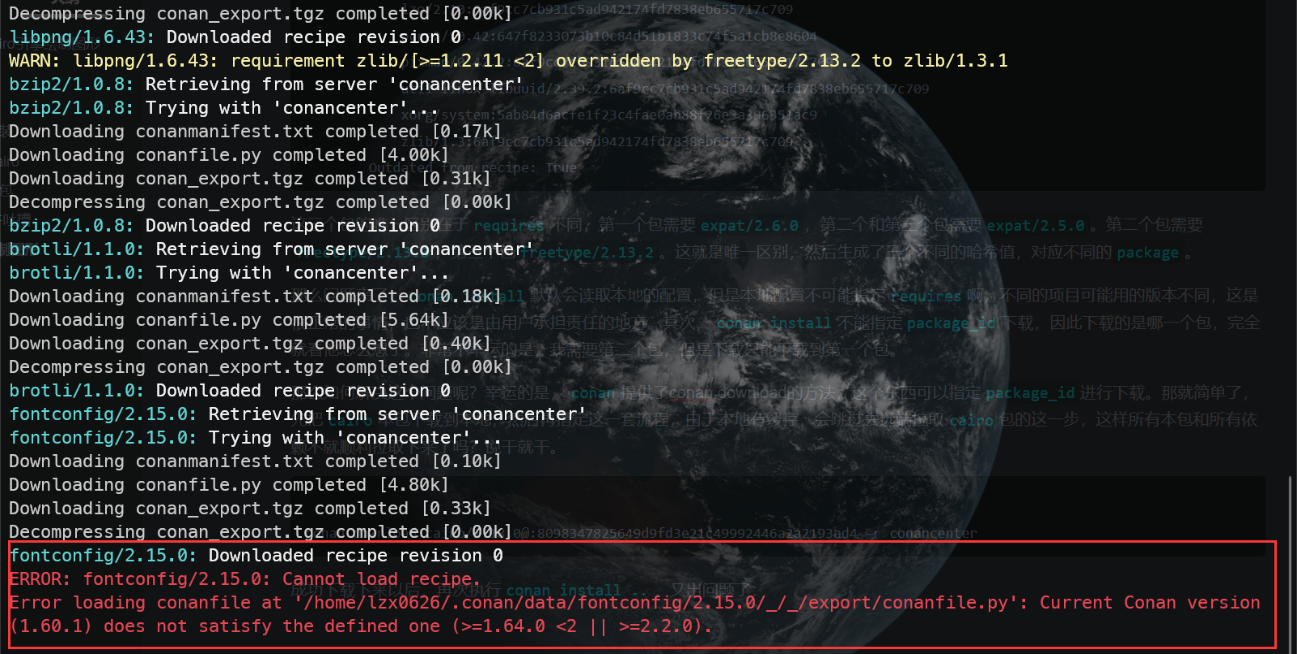


图2-13 conan安装报错结果2

这个包依赖的fontconfig的版本是2.14.2，但是这里还是尝试下载2.15.0。我百思不得其解。为了解决这个问题，我排查了对应的conanfile.py，阅读到requirements()函数的时候，豁然开朗。

1. def requirements(self):
2. self.requires("pixman/0.43.4")
3. if self.options.with\_zlib and self.options.with\_png:
4. self.requires("expat/[>=2.6.2 <3]")
5. if self.options.with\_lzo:
6. self.requires("lzo/2.10")
7. if self.options.with\_zlib:
8. self.requires("zlib/[>=1.2.11 <2]")
9. if self.options.with\_freetype:
10. self.requires("freetype/2.13.2", transitive\_headers=True, transitive\_libs=True)
11. if self.options.with\_fontconfig:
12. self.requires("fontconfig/2.15.0", transitive\_headers=True, transitive\_libs=True)
13. if self.options.with\_png:
14. self.requires("libpng/[>=1.6 <2]")
15. if self.options.with\_glib:
16. self.requires("glib/2.78.3")
17. if self.settings.os in ["Linux", "FreeBSD"]:
18. if self.options.with\_xlib or self.options.with\_xlib\_xrender or self.options.with\_xcb:
19. self.requires("xorg/system", transitive\_headers=True, transitive\_libs=True)
20. if self.options.get\_safe("with\_opengl") == "desktop":
21. self.requires("opengl/system", transitive\_headers=True, transitive\_libs=True)
22. if self.settings.os == "Windows":
23. self.requires("glext/cci.20210420")
24. self.requires("wglext/cci.20200813")
25. self.requires("khrplatform/cci.20200529")
26. if self.options.get\_safe("with\_opengl") and self.settings.os in ["Linux", "FreeBSD"]:
27. self.requires("egl/system", transitive\_headers=True, transitive\_libs=True)

问题就在出在这里，expat需要的是2.5.0，但是规定依赖的包是>=2.6.2 and <3。这不前后矛盾吗？对于freetype和fontconfig也是相同的问题。同时，再浏览一下cairo包拉下来以后的整体结构。原来同一个版本的所有包的conanfile.py都是同一个文件。这一点其实没有任何问题，这是conan的设计。但是不更新依赖版本的限制就很令人费解了，如果硬要不改的话，发布不同的版本也是可以的。这样一套流程下来，导致conan拉包就出现了问题。

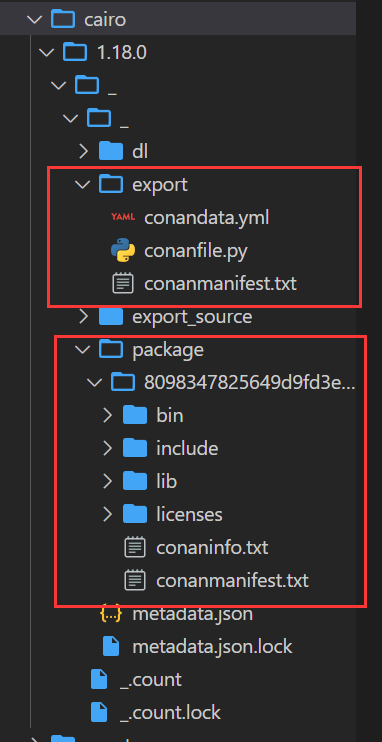


图2-14 conan包结构

那么如何解决呢？公司这边的解决方式是将该版本的cairo包上传到公司的conan服务器上，并手动修改conanfile.py使其版本匹配，并将所有的依赖以及穿透依赖全部拷贝到公司服务器上。至此，cairo终于成功通过conan安装。

Windows那边也是同样的问题，类似同样操作即可。

另外，在Linux下的Cairo包中，有一个依赖项值得看一下，即xorg/system。这个包的版本是system，意思是和系统相关的包，这里是Linux下X图形协议相关的依赖。众所周知，系统包是通过apt或者yum进行安装的，言下之意是conan能够调用这些包命令来帮我们自动安装对应的包吗？

答案是可以的，参见文档<https://docs.conan.io/2/reference/tools/system/package_manager.html>。留意这段代码：

1. def system\_requirements(self):
2. apt = package\_manager.Apt(self)
3. apt.install(["libx11-dev", "libx11-xcb-dev", "libfontenc-dev", "libice-dev", "libsm-dev", "libxau-dev", "libxaw7-dev",
4. "libxcomposite-dev", "libxcursor-dev", "libxdamage-dev", "libxdmcp-dev", "libxext-dev", "libxfixes-dev",
5. "libxi-dev", "libxinerama-dev", "libxkbfile-dev", "libxmu-dev", "libxmuu-dev",
6. "libxpm-dev", "libxrandr-dev", "libxrender-dev", "libxres-dev", "libxss-dev", "libxt-dev", "libxtst-dev",
7. "libxv-dev", "libxxf86vm-dev", "libxcb-glx0-dev", "libxcb-render0-dev",
8. "libxcb-render-util0-dev", "libxcb-xkb-dev", "libxcb-icccm4-dev", "libxcb-image0-dev",
9. "libxcb-keysyms1-dev", "libxcb-randr0-dev", "libxcb-shape0-dev", "libxcb-sync-dev", "libxcb-xfixes0-dev",
10. "libxcb-xinerama0-dev", "libxcb-dri3-dev", "uuid-dev", "libxcb-cursor-dev", "libxcb-dri2-0-dev",
11. "libxcb-dri3-dev", "libxcb-present-dev", "libxcb-composite0-dev", "libxcb-ewmh-dev",
12. "libxcb-res0-dev"], update=True, check=True)
13. apt.install\_substitutes(
14. ["libxcb-util-dev"], ["libxcb-util0-dev"], update=True, check=True)
15. yum = package\_manager.Yum(self)
16. yum.install(["libxcb-devel", "libfontenc-devel", "libXaw-devel", "libXcomposite-devel",
17. "libXcursor-devel", "libXdmcp-devel", "libXtst-devel", "libXinerama-devel",
18. "libxkbfile-devel", "libXrandr-devel", "libXres-devel", "libXScrnSaver-devel",
19. "xcb-util-wm-devel", "xcb-util-image-devel", "xcb-util-keysyms-devel",
20. "xcb-util-renderutil-devel", "libXdamage-devel", "libXxf86vm-devel", "libXv-devel",
21. "xcb-util-devel", "libuuid-devel", "xcb-util-cursor-devel"], update=True, check=True)
22. dnf = package\_manager.Dnf(self)
23. dnf.install(["libxcb-devel", "libfontenc-devel", "libXaw-devel", "libXcomposite-devel",
24. "libXcursor-devel", "libXdmcp-devel", "libXtst-devel", "libXinerama-devel",
25. "libxkbfile-devel", "libXrandr-devel", "libXres-devel", "libXScrnSaver-devel",
26. "xcb-util-wm-devel", "xcb-util-image-devel", "xcb-util-keysyms-devel",
27. "xcb-util-renderutil-devel", "libXdamage-devel", "libXxf86vm-devel", "libXv-devel",
28. "xcb-util-devel", "libuuid-devel", "xcb-util-cursor-devel"], update=True, check=True)
29. zypper = package\_manager.Zypper(self)
30. zypper.install(["libxcb-devel", "libfontenc-devel", "libXaw-devel", "libXcomposite-devel",
31. "libXcursor-devel", "libXdmcp-devel", "libXtst-devel", "libXinerama-devel",
32. "libxkbfile-devel", "libXrandr-devel", "libXres-devel", "libXss-devel",
33. "xcb-util-wm-devel", "xcb-util-image-devel", "xcb-util-keysyms-devel",
34. "xcb-util-renderutil-devel", "libXdamage-devel", "libXxf86vm-devel", "libXv-devel",
35. "xcb-util-devel", "libuuid-devel", "xcb-util-cursor-devel"], update=True, check=True)
36. pacman = package\_manager.PacMan(self)
37. pacman.install(["libxcb", "libfontenc", "libice", "libsm", "libxaw", "libxcomposite", "libxcursor",
38. "libxdamage", "libxdmcp", "libxtst", "libxinerama", "libxkbfile", "libxrandr", "libxres",
39. "libxss", "xcb-util-wm", "xcb-util-image", "xcb-util-keysyms", "xcb-util-renderutil",
40. "libxxf86vm", "libxv", "xcb-util", "util-linux-libs", "xcb-util-cursor"], update=True, check=True)
41. package\_manager.Pkg(self).install(["libX11", "libfontenc", "libice", "libsm", "libxaw", "libxcomposite", "libxcursor",
42. "libxdamage", "libxdmcp", "libxtst", "libxinerama", "libxkbfile", "libxrandr", "libxres",
43. "libXScrnSaver", "xcb-util-wm", "xcb-util-image", "xcb-util-keysyms", "xcb-util-renderutil",
44. "libxxf86vm", "libxv", "xkeyboard-config", "xcb-util", "xcb-util-cursor"], update=True, check=True)

对于Ubuntu来讲，conan能手动帮我们调用apt安装所需要的系统依赖。需要注意一点，需要在配置文件中加上两句，来指明开启这个功能和使用sudo。所以最终的配置文件类似如下：

1. [settings]
2. os=Linux
3. os\_build=Linux
4. arch=x86\_64
5. arch\_build=x86\_64
6. compiler=gcc
7. compiler.version=9
8. compiler.libcxx=libstdc++11
9. build\_type=Release
10. [options]
11. [build\_requires]
12. [env]
13. [conf]
14. tools.system.package\_manager:mode=install
15. tools.system.package\_manager:sudo=True

至此，我们成功通过conan安装下来了cairo。

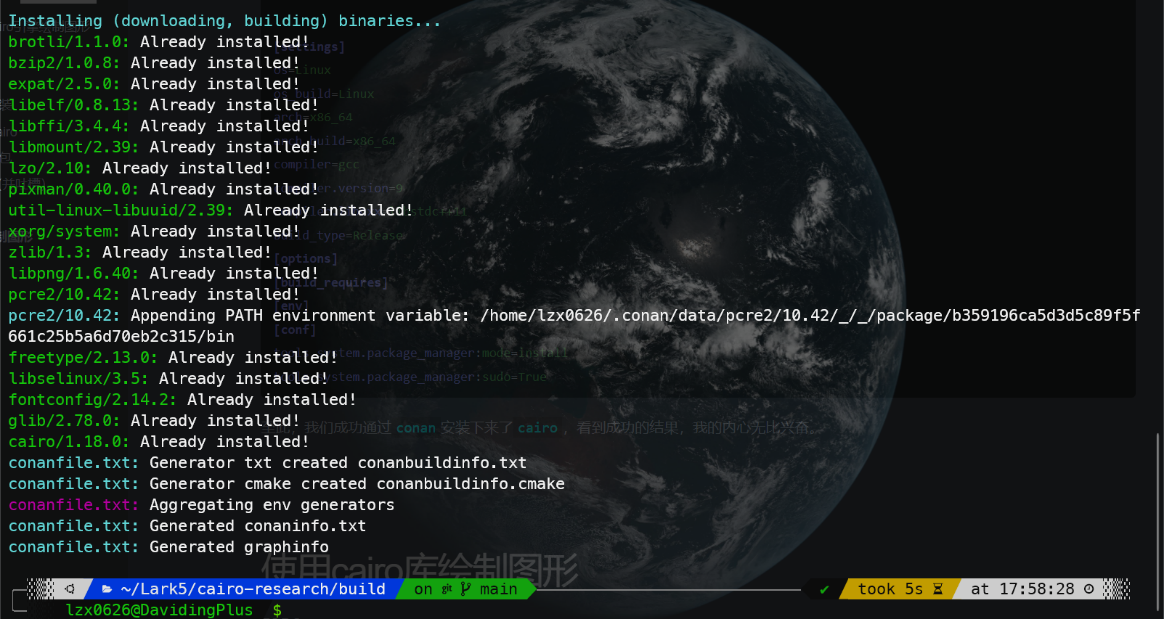


图2-15 安装成功的结果

### 2.3.2 使用Cairo库绘制图形

安装完cairo库，是时候使用它绘制一些简单的图形了。

**1、Linux GUI背景简述**

Linux本身是不带有图形界面的，真正原生的Linux系统只是一个基于命令行的操作系统。但是目前市面上所使用的Linux发行版，例如Ubuntu、Centos等，都是有图形界面的。[9]这是因为这些图形界面是Linux下的一个应用程序而已，是通过程序和协议模拟和实现出来的，或者说图形界面并不属于Linux内核的一部分。这一点和Windows系统完全不一样，Windows的命令行在我个人看来完全不如Linux这么好用，甚至有时候反而不好用，但是Windows的用户的依然最多。为什么呢？就是因为GUI好看。Windows的图形界面是操作系统的一部分，并且做的确实好看和丝滑。这样变相的降低了用户的学习和使用成本，对大多数的人而言是件好事。但是对于程序员特别是偏向底层的程序员来讲，却真不一定。

在Linux下，需要通过应用程序实现图形界面，那就需要设计一个合适的协议。[10]目前市面上比较流行的有两种协议，X协议和Wayland协议。这两种协议都是基于网络通信的思想，将图形显示分为了客户端（即用户的应用程序）和服务端通信的过程。输入设备和显示设备不是同一个设备，而且他们需要相互配合，进行画面显示，所以需要一个交互协议，建立他们直接的沟通桥梁。当然X协议和Wayland协议的细节有所区别，粗略的讲是server和compositor的设计不同，具体可见<https://www.secjuice.com/wayland-vs-xorg/>。

后续以X协议为例展开，并以XClient和XServer分别代指客户端和服务端。现在的需求是需要画一个圆，XClient需要告诉XServer在屏幕的什么地方，使用什么颜色，画多大的一个圆。至于这个圆如何生成，如何使用硬件真正绘制图形等等这些操作，都是由XServer完成的。当然更进一步的，XServer还可以捕捉鼠标和键盘的动作，会触发相应的事件。XClient可以接受相应的事件并且完成相应的逻辑。这就是整个X协议以及绘制逻辑的简要概括。

X协议有很多实现。目前用的最多的是XOrg，对应的XClient有Xlib和XCB的两种实现，提供了和XServer对接的API。（At the bottom level of the X client library stack are Xlib and XCB, two helper libraries (really sets of libraries) that provide API for talking to the X server.）本文的背景是X11，也是Xlib库的一个特殊版本。

**2、安装X11并编写GUI程序**

强烈建议在虚拟机下进行，因为Wsl需要内核更新到2.2.4以后才能使用最新功能的GUI，而且体验还不是很好。

以Ubuntu为例，系统默认是未安装X11库的（XServer肯定是有的，不然就看不到图形界面了。安装X11只是提供了窗口系统的开发支持）。使用如下命令手动安装。值得一提的是，由于X11是系统库，因此使用apt包安装即可。同理对于Wayland也是一样。

1. sudo apt install libx11-dev

安装完成以后，使用CMake就能很方便的引入X11支持了。

1. add\_executable (xxx
2. ...
3. )
4. target\_link\_libraries (xxx X11)

现在编写一个样例程序，用于在X11下绘制一个图形窗口。这里面涉及到很多的X11的API，这里只做简单介绍，具体请参考文档<https://www.x.org/releases/X11R7.7/doc/libX11/libX11/libX11.html>

1. #include <X11/Xlib.h>
2. #include <iostream>
3. int main()
4. {
5. *// 用于和 XServer 建立连接，dpy 指针全局只有一份*
6. Display \*dpy = XOpenDisplay(nullptr);
7. if (!dpy)
8. {
9. std::cerr << "Unable to open X display!" << std::endl;
10. throw;
11. }
12. int screen = DefaultScreen(dpy);
13. *// 创建一个顶层窗口*
14. Window w = XCreateSimpleWindow(
15. dpy,
16. RootWindow(dpy, DefaultScreen(dpy)),
17. 100, 100, 400, 300, 0,
18. BlackPixel(dpy, screen),
19. BlackPixel(dpy, screen)
20. );
21. *// 将需要检测的事件绑定在窗口上*
22. XSelectInput(dpy, w, ExposureMask);
23. *// 展示这个窗口*
24. XMapWindow(dpy, w);
25. std::cout << "Entering loop ..." << std::endl;
26. *// 进入事件循环*
27. XEvent e;
28. while (true)
29. {
30. XNextEvent(dpy, &e);
31. switch (e.type)
32. {
33. case Expose:
34. std::cout << "event: Expose" << std::endl;
35. break;
36. default:
37. std::cout << "event: " << e.type << std::endl;
38. break;
39. }
40. }
41. return 0;
42. }

接口的大致功能以在代码注释中体现。注意到整个程序最重要的架构是最后的这个事件循环，当XClient窗口成功创建出来以后，XServer需要不断监听XClient发送的事件并予以处理，这样才能实现GUI程序的功能。因此事件循环的存在就自然而然了。有点类似于IO多路复用中epoll技术的框架。在这里监听的是Expose事件，在窗口创建和窗口大小发生改变的时候会触发Expose事件，这里也是用作信息打印测试。

程序的运行结果如下，可以发现窗口创建和改变窗口大小的时候在不断打印Expose的信息。

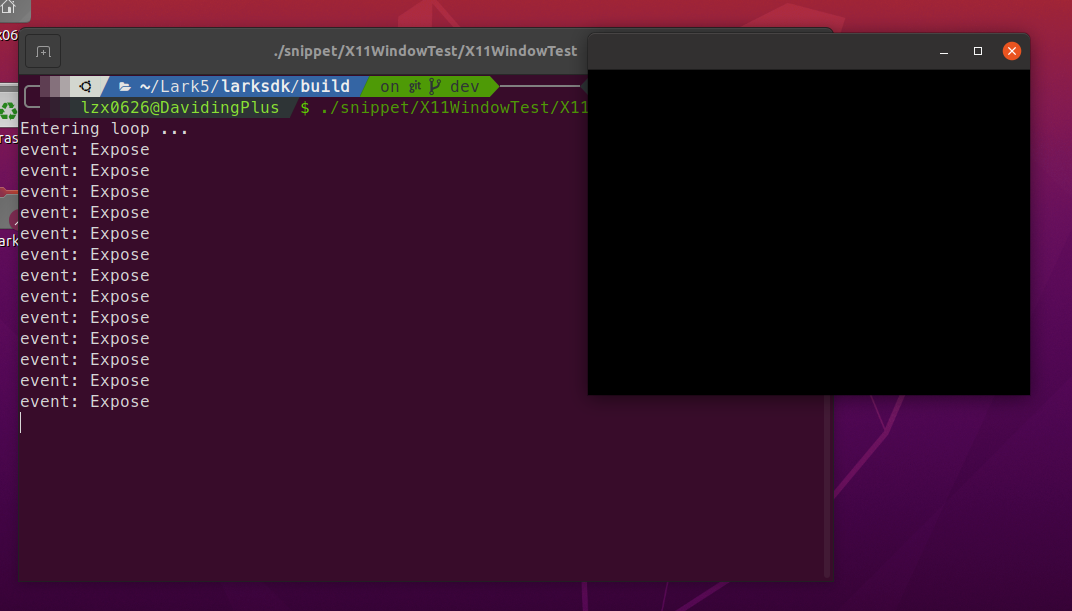


图2-16 样例程序运行结果

对于事件循环这个概念，不管是Qt，还是LarkSDK，还是对于一个跨平台的GUI框架，事件循环显然是必不可少的。问题在于跨平台需要统一不同平台下的窗体系统和事件处理等逻辑，最终抽象出跨平台的接口，这就是这些框架正在做最重要的一件事情。以LarkSDK为例，虽然最简单的跨平台的程序是四行就能搞定，但是其中涉及到的知识和背景是非常庞大的。

1. #include <lwindowapplication.h>
2. #include <lwindow.h>
3. int main()
4. {
5. LWindowApplication app; *// 创建窗体程序主框架实例*
6. LWindow w; *// 创建顶层窗体*
7. w.show(); *// 让窗体可见*
8. return app.exec(); *// 进入主事件循环*
9. }

**3、尝试引入Cairo**

cairo将输出和绘制的概念做了严格区分。cairo surface是一个抽象出来的概念，与其对接的是多种输出方式，例如PDF、PNG、SVG、Win32、XLib、XCB等，如图所示。

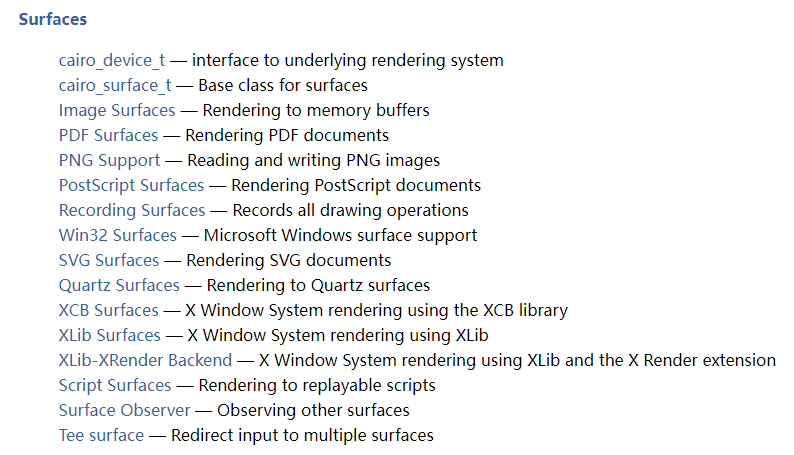
****

图2-17 cairo surface 概览

接着查看cairo官方提供的[samples](https://cairographics.org/samples/)，可以发现，官方提供的样例好像完全和surface没有关系，换句话说没有反映输出的方式。例如这段代码：

1. double xc = 128.0;
2. double yc = 128.0;
3. double radius = 100.0;
4. double angle1 = 45.0  \* (M\_PI/180.0);  /\* angles are specified \*/
5. double angle2 = 180.0 \* (M\_PI/180.0);  /\* in radians           \*/
6. cairo\_set\_line\_width (cr, 10.0);
7. cairo\_arc (cr, xc, yc, radius, angle1, angle2);
8. cairo\_stroke (cr);
9. /\* draw helping lines \*/
10. cairo\_set\_source\_rgba (cr, 1, 0.2, 0.2, 0.6);
11. cairo\_set\_line\_width (cr, 6.0);
12. cairo\_arc (cr, xc, yc, 10.0, 0, 2\*M\_PI);
13. cairo\_fill (cr);
14. cairo\_arc (cr, xc, yc, radius, angle1, angle1);
15. cairo\_line\_to (cr, xc, yc);
16. cairo\_arc (cr, xc, yc, radius, angle2, angle2);
17. cairo\_line\_to (cr, xc, yc);
18. cairo\_stroke (cr);

它的绘制结果是这样的：

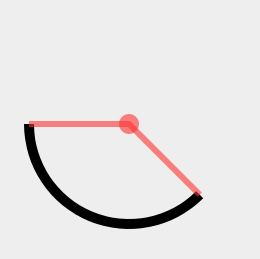


图2-18 样例绘制效果图

这个图样可以被输出到前面提到的任意一种surface中。同时这也是前面提到的，cairo将输出和绘制的概念做了完整区分，同样这也是容易想到和愿意看到的。所以，如果想要创建一个输出到PNG中的实例，代码应该类似如下：

1. *// 创建 surface*
2. cairo\_surface\_t \*surface = cairo\_image\_surface\_create(CAIRO\_FORMAT\_ARGB32, 640, 480);
3. *// 创建绘图上下文 context*
4. cairo\_t \*cr = cairo\_create(surface);
5. *// 绘制逻辑，只和 context 相关，与 surface 无关*
6. ...
7. *// 输出到 png 格式中*
8. cairo\_surface\_write\_to\_png(surface, "xxx.png");

至此，我习惯使用surface和context来代指输出和绘制的概念，也明白了cario是如何区分这两个概念的了。surface用作指明绘制的图形输出到哪里，context则用于进行绘制。

**4、Cairo与X11相结合**

使用cairo成功输出到图片文件中以后，试着想想如何与X11窗口系统结合了。提前声明，用到的surface只有XLib Surface和Image Surface，其他的API请自行查询[文档](https://cairographics.org/manual/)。

首先想到的肯定是XLib Surface。幸运的是，cairo帮用户做好了与X11平台的对接工作，用户只需要按部就班地使用cairo的API即可。所有的surface基本上只有在创建的时候会有区别，在context那一层的绘制几乎没有区别。例如下面就是XLib Surface的创建API，参数具体含义请参考[官方文档](https://cairographics.org/manual/cairo-XLib-Surfaces.html)。

1. cairo\_surface\_t \*
2. cairo\_xlib\_surface\_create (Display \*dpy,
3. Drawable drawable,
4. Visual \*visual,
5. int width,
6. int height);

查询该方法需要的接口如何获取以后，编写出如下的代码：

1. #include <iostream>
2. #include <exception>
3. #include <X11/Xlib.h>
4. #include "cairo.h"
5. #include "cairo/cairo-xlib.h"
6. int main()
7. {
8. Display \*dpy = XOpenDisplay(nullptr);
9. if (!dpy)
10. {
11. throw std::runtime\_error("Failed to open X display");
12. }
13. int screen = DefaultScreen(dpy);
14. Window w = XCreateSimpleWindow(
15. dpy,
16. RootWindow(dpy, DefaultScreen(dpy)),
17. 100, 100, 640, 480, 0,
18. BlackPixel(dpy, screen),
19. BlackPixel(dpy, screen));
20. XSelectInput(dpy, w, ExposureMask);
21. XMapWindow(dpy, w);
22. std::cout << "Entering loop ..." << std::endl;
23. *// 根据 window id 获取该窗口的信息*
24. XWindowAttributes attr;
25. XGetWindowAttributes(dpy, w, &attr);
26. *// 创建 XLib Surface*
27. cairo\_surface\_t \*surface = cairo\_xlib\_surface\_create(dpy, w, attr.visual, attr.width, attr.height);
28. cairo\_t \*cr = cairo\_create(surface);
29. XEvent e;
30. while (true)
31. {
32. XNextEvent(dpy, &e);
33. switch (e.type)
34. {
35. case Expose:
36. {
37. std::cout << "event: Expose" << std::endl;
38. *// 绘制操作*
39. cairo\_set\_source\_rgb(cr, 1.0, 1.0, 0.5);
40. cairo\_paint(cr);
41. cairo\_set\_source\_rgb(cr, 1.0, 0.0, 1.0);
42. cairo\_move\_to(cr, 100, 100);
43. cairo\_line\_to(cr, 200, 200);
44. cairo\_stroke(cr);
45. break;
46. }
47. default:
48. std::cout << "event: " << e.type << std::endl;
49. break;
50. }
51. }
52. cairo\_destroy(cr);
53. cairo\_surface\_destroy(surface);
54. XDestroyWindow(dpy, w);
55. XCloseDisplay(dpy);
56. return 0;
57. }

这样能在Expose事件触发的时候成功绘制出文章开始时候展示的图样。

XLib Surface的代码结构看着很像自动挡的感觉，创建surface，在事件循环中用context进行绘制，最终得到想要的图案。

深入思考一下，图形是如何被绘制到屏幕上的呢？前面举了个例子，画一个圆，告诉XServer在哪里，用什么颜色，画多大、多宽的圆。至于如何用硬件画不是XClient关心的事情，但是如何表示这些信息呢？显然需要用合适的data进行存储。进一步讲，context调用各种方法的实际过程，其实就是往数据缓冲区data中写数据的过程。当类似flush操作的被调用以后，这些数据才会真正反映在屏幕上，形成我们观看的效果。

在这样的语义下，进一步思考surface的概念，其实用可绘制表面的概念好像更加贴切（本概念借鉴于LarkSDK的LSurface）。[11]绘图的数据缓冲区记录了图形的数据，类型是unsigned char \*，这些数据和不同的输出方式对接就能达到不同的输出效果。至于为什么是unsigned char \*（猜测大概是字节流）以及如何对接。

知道这个过程以后，回到Image Surface本身，为什么要使用这个东西，是因为它为用户提供了获取数据的接口。也就是当用户使用context绘制以后，调用这个方法就能立刻拿到缓冲区的数据。

1. unsigned char \*
2. cairo\_image\_surface\_get\_data (cairo\_surface\_t \*surface);

最后思考一下绘制效率。不同引擎的效率的区别根本上就是在于如何快速的把这些数据计算出来，或者换句话讲，如何快速地让缓冲区的内存填充为指定的数据。比如对于最基本的暴力软渲染和cairo引擎，它们的效率差距显然是非常大的。这里有一个例子可以参考，是LarkSDK原生软渲染和cairo引擎同样绘制10000条斜线的效率差距，以下是结果，保守估计至少差了几百到一千倍。

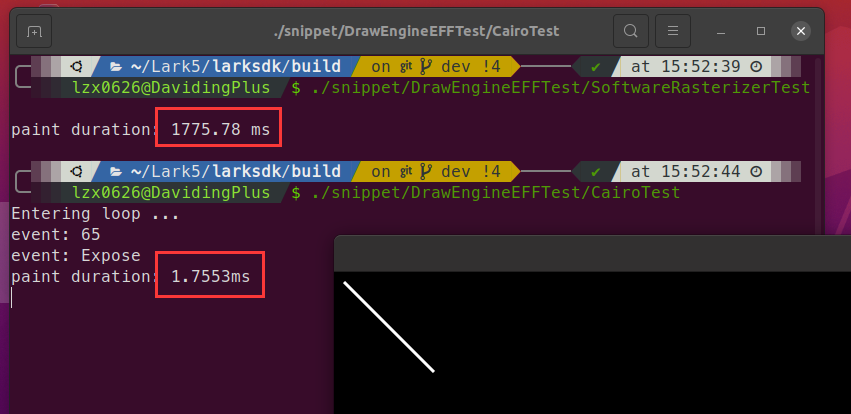


图2-19 绘制效率对比图

回到正题，再结合X11的API，可以给出使用Image Surface的代码：

1. #include <iostream>
2. #include <exception>
3. #include <X11/Xlib.h>
4. #include "cairo.h"
5. int main()
6. {
7. Display \*dpy = XOpenDisplay(nullptr);
8. if (!dpy)
9. {
10. throw std::runtime\_error("Failed to open X display");
11. }
12. int screen = DefaultScreen(dpy);
13. Window w = XCreateSimpleWindow(
14. dpy,
15. RootWindow(dpy, DefaultScreen(dpy)),
16. 100, 100, 640, 480, 0,
17. BlackPixel(dpy, screen),
18. BlackPixel(dpy, screen));
19. XSelectInput(dpy, w, ExposureMask);
20. unsigned long mask = 0;
21. XGCValues values;
22. GC gc = XCreateGC(dpy, w, mask, &values);
23. XMapWindow(dpy, w);
24. *// 创建 Image Surface*
25. cairo\_surface\_t \*surface = cairo\_image\_surface\_create(CAIRO\_FORMAT\_ARGB32, 640, 480);
26. cairo\_t \*cr = cairo\_create(surface);
27. *// 获得 Cairo 管理的绘图数据的指针*
28. unsigned char \*pData = cairo\_image\_surface\_get\_data(surface);
29. *// 创建 X11 下的 Image Buffer ，将其中的数据替换为 Cairo 的数据指针*
30. XImage \*pBackBuffer = XCreateImage(
31. dpy,
32. DefaultVisual(dpy, screen),
33. DefaultDepth(dpy, screen),
34. ZPixmap,
35. 0,
36. (char \*)pData,
37. 640, 480,
38. 8,
39. 0);
40. std::cout << "Entering loop ..." << std::endl;
41. XEvent e;
42. while (true)
43. {
44. XNextEvent(dpy, &e);
45. switch (e.type)
46. {
47. case Expose:
48. {
49. std::cout << "event: Expose" << std::endl;
50. cairo\_set\_source\_rgb(cr, 1.0, 1.0, 0.5);
51. cairo\_paint(cr);
52. cairo\_set\_source\_rgb(cr, 1.0, 0.0, 1.0);
53. cairo\_move\_to(cr, 100, 100);
54. cairo\_line\_to(cr, 200, 200);
55. cairo\_stroke(cr);
56. *// flush 操作，刷新缓冲区，更新数据*
57. cairo\_surface\_flush(surface);
59. *// X11 下真正绘制图形的方法，用到了外面定义的 X11 Image Buffer，而其内部的数据就是 Cairo 管理的缓冲区数据*
60. XPutImage(
61. dpy,
62. w,
63. gc,
64. pBackBuffer,
65. 0, 0,
66. 0, 0,
67. 640, 480);
68. break;
69. }
70. default:
71. std::cout << "event: " << e.type << std::endl;
72. break;
73. }
74. }
75. cairo\_destroy(cr);
76. cairo\_surface\_destroy(surface);
77. XDestroyWindow(dpy, w);
78. XCloseDisplay(dpy);
79. return 0;
80. }

至此，完成了在X11下使用Cairo引擎绘制图形的全部过程。Windows的程序架构和事件循环有所区别，但思路是相同的。当然，这仅仅是阐述了基本过程，还有更多的细节值得研究和探讨。

### 2.4 使用Woboq CodeBrowser搭建源代码网站

### 2.4.1 背景

在日常学习工作中，不免需要浏览一些库的源码。在本地浏览源代码，例如使用Source Insight，当然是可以的，但问题是一是不方便，二是很多库下载下来是以头文件配合静态库或动态库的形式存在的，看不到cpp代码，因此阅读会受限。现在Web技术高速发展，有没有办法用网页直接查看源代码，并且还有类似于Code的代码跳转功能呢？换句话说，如何把C/C++代码转化为前端页面，并且最好是静态的前端页面，就是一个难题了。

幸运的是，github上有人提前考虑到了这件事情，并且有了具体的[项目](https://github.com/KDAB/codebrowser)。并且原作者还基于这个框架弄出了一个在线的[源代码网站](https://codebrowser.dev/)，方便开发者查看各个C/C++库的源代码，例如Qt、GCC、Linux Kernel、GNU C Library等。

那么问题来了，都有在线的网站了为什么还要自己搭建一个呢？原因就是这个网站在国内被墙了无法访问。为了一劳永逸解决这个问题，因此决定自己搭建一个源代码网站。

### 2.4.2 工作原理简介

[12]Woboq CodeBrowser是基于LLVM/Clang实现的一个命令行工具。它通过深度解析C/C++源码生成最终我们需要的静态HTML文件。Woboq CodeBrowser包含codebrowser\_generator和codebrowser\_indexgenerator两个子命令。生成HTML文件到最终可以通过浏览器阅读代码，整体分三个步骤：

1. 先通过codebrowser\_generator解析.h和.cpp生成对应的.h.html和.cpp.html。

2. 然后通过codebrowser\_indexgenerator为所有目录生成 index.html。

3. 最后把这些HTML文件拷贝到某个Web服务器上，就可以在浏览器里愉快地浏览C/C++项目的源码了。

### 2.4.3 安装Woboq CodeBrowser工具

通过源码编译安装这个工具。

首先将该[项目](https://github.com/KDAB/codebrowser)克隆到本地，由于需要用LLVM/Clang工具链，强烈建议在Linux下操作。

1. git clone https:*//github.com/KDAB/codebrowser.git*

克隆下的项目中有三个目录比较重要，一是data目录，这是一些前端的资源文件，例如png、css、js等；二是generator目录，这是可执行文件codebrowser\_generator的源码目录；三是indexgenerator，这是可执行文件codebrowser\_indexgenerator的源码目录。

前面提到，编译这个工具需要用到LLVM/Clang工具链，结合github上的[issue#119](https://github.com/KDAB/codebrowser/issues/119)，使用llvm-14和clang-14的版本比较合适。以Ubuntu为例：

1. sudo apt install llvm-14 clang-14 libclang-14-dev

但是在Ubuntu20.04及以下的版本上会出现定位不到软件包的问题，官方提供了一个解决方案，参见博客<https://blog.csdn.net/weixin_50749380/article/details/128319851>

这样最重要的编译环境就安装好了，紧接着需要构建这个项目并且将编译后的文件安装到本地。由于项目受CMake管理，同时参考[官方文档](https://github.com/KDAB/codebrowser/blob/master/README.md)，构建流程就很简单了。

1. mkdir -p build
2. cd build
3. cmake -DCMAKE\_C\_COMPILER=clang-14 -DCMAKE\_CXX\_COMPILER=clang++-14 -DCMAKE\_BUILD\_TYPE=Release .. # 除了 clang 还可能安装了 gcc 等其他的编译器，所以指定一下；同时指定编译选项为 Release ，这个在 Linux 下没有影响
4. make
5. sudo make install # 必要加上 sudo

安装完毕以后data目录，两个可执行文件codebrowser\_generator和codebrowser\_indexgenerator就成功安装到本地了。

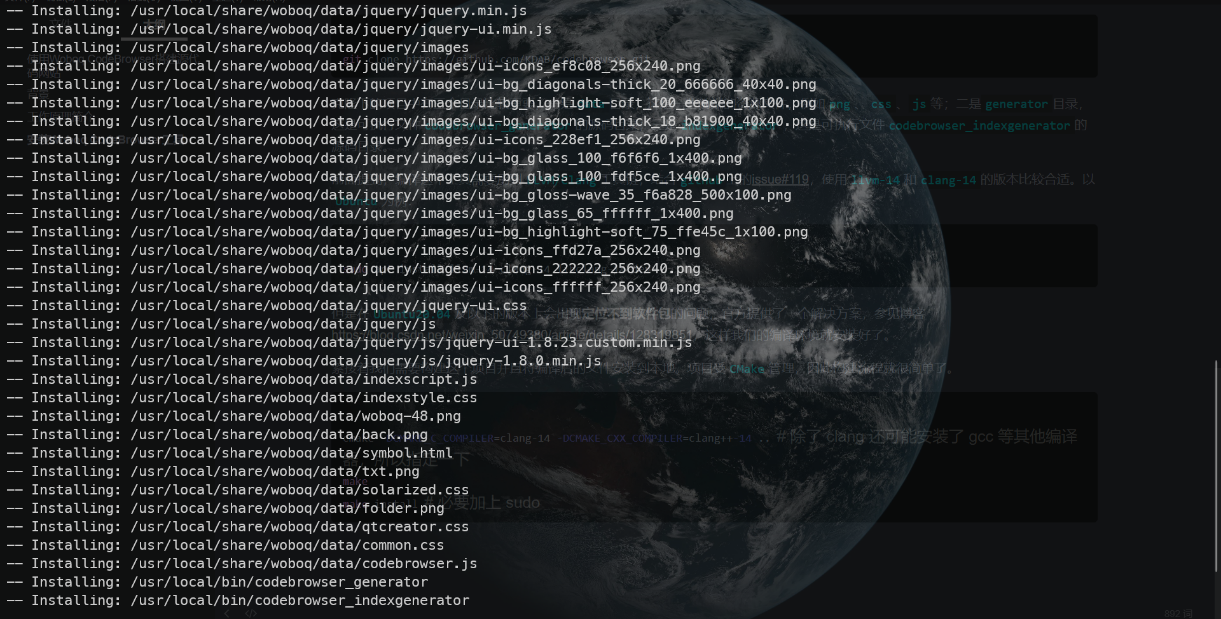


图2-20 woboq工具安装结果

### 2.4.4 如何使用

接下来以googletest-1.12.1为例，展示如何使用这个工具构建静态的源代码网站。

**1、生成compile\_commands.json**

[13]compile\_commands.json文件是一种特定格式的[compilation database](https://sarcasm.github.io/notes/dev/compilation-database.html)文件，而所谓compilation database其实很简单，它里面记录的是每一个源码文件在编译时详细的信息（包括文件路径，文件名，编译选项等等）。而compile\_commands.json文件是[LibTooling](https://clang.llvm.org/docs/LibTooling.html)需要的以json格式呈现的compilation database文件，以下截取的是compile\_commands.json中的一个entry：

1. [
2. ...
3. {
4. "arguments": [
5. "c++",
6. "-c",
7. "-g",
8. "-O2",
9. "-Werror",
10. "-std=c++0x",
11. "-Wall",
12. "-fPIC",
13. "-o",
14. "attrs.o",
15. "attrs.cc"
16. ],
17. "directory": "/home/astrol/libelfin/dwarf",
18. "file": "attrs.cc"
19. },
20. ...
21. ]

Woboq Codebrowser是基于LLVM/Clang实现的工具，也是基于compile\_commands.json来分析源码关系的。

换句话说，要想使用Woboq Codebrowser，必须首先生成compile\_commands.json文件。如果项目是由cmake构建的，那么恭喜，只需加上-DCMAKE\_EXPORT\_COMPILE\_COMMANDS=ON即可。如果是传统的make build system也无需担心，Bear和compdb工具可以帮助用户生成compile\_commands.json文件。

现在来生成googletest-1.12.1项目对应的compile\_commands.json。首先拉取[googletest](https://github.com/google/googletest)的源码到本地。源码采用CMake工具的管理，这样正好很方便的帮助用户生成compile\_commands.json。之所以这么推荐CMake是因为使用其他的工具可能需要编译整个工程才能生成该文件，费事费力。

因为我是C++11标准的环境，因此需要使用的版本是1.12.1，切换到对应的tag。当然直接下载对应的release源码也行。

1. git checkout release-1.12.1 # target tag

接着使用CMake工具生成compile\_commands.json文件，个人建议使用gcc/g++编译器。

1. mkdir -p build
2. cd build
3. cmake -DCMAKE\_BUILD\_TYPE=Release -DCMAKE\_C\_COMPILER="gcc" -DCMAKE\_CXX\_COMPILER="g++" -DCMAKE\_EXPORT\_COMPILE\_COMMANDS=ON ..

在build目录下看到compile\_commands.json文件正确生成即为成功。

**2、使用codebrowser\_generator生成html**

codebrowser\_generator是一个可执行文件，有各种执行参数，说明如下：

-a：处理compile\_commands.json中的所有文件。如果没有传递这个参数，那么就需要传递要处理的文件列表

-o：指定文件输出目录

-b：是指包含compile\_commands.json的构建目录。如果没有传递这个参数，编译参数可以在--后通过命令行传递

-p：（一个或多个）用于项目规范。即项目的名称、源代码的绝对路径和用冒号分隔的版本信息。示例：-p projectname:/path/to/source/code:0.3beta

-d：指定包含所有JavaScript和CSS文件的数据URL。默认为相对于输出目录的../data。示例：-d https://codebrowser.dev/data/

-e：是对外部项目的引用。示例： -e clang/include/clang:/opt/llvm/include/clang/:https://codebrowser.dev/llvm

例如，对于当前googletest-1.12.1项目，一条合适的命令可能是这样的：

1. codebrowser\_generator -b ./compile\_commands.json -a -p googletest-1.12.1:"${PWD}/..":1.12.1 -o ./docs -d ../data

执行完的结果是这样的：

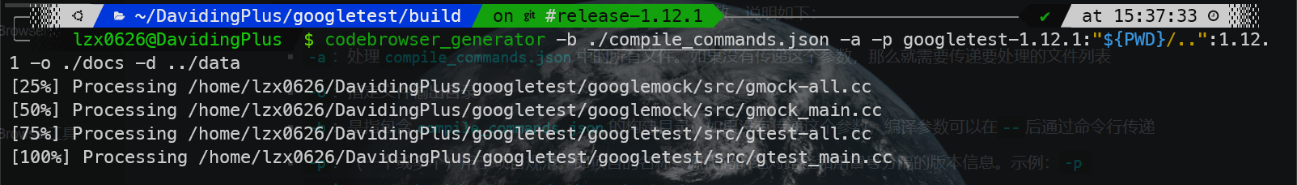
****

图2-21 codebrowser\_generator命令执行结果

需要注意的是，上面的操作都是在根目录的build构建目录执行的，这也是CMake用户的构建习惯。

同时，-o ./docs代表文件输出在build/docs中，-d ../data是指定资源文件的相对路径，代表资源文件位于生成目录docs的父级目录。至于为什么是../data，后面会详细解释。安装好Woboq CodeBrowser工具以后data目录会被安装在/usr/local/share/woboq/data，将其拷贝为build/data即可。

**3、使用codebrowser\_indexgenerator为每个目录生成index.html**

codebrowser\_indexgenerator同样有很多执行参数，说明如下：

-p：（一个或多个）用于项目规范。即项目的名称、源代码的绝对路径和用冒号分隔的版本信息。示例：-p projectname:/path/to/source/code:0.3beta

-d：指定包含所有JavaScript和CSS文件的数据URL。默认为相对于输出目录的 ../data。示例：-d https://codebrowser.dev/data/

同样执行类似命令：

1. codebrowser\_indexgenerator ./docs -d ../data

类似执行结果如下，可以看到每个目录对应都有了index.html。

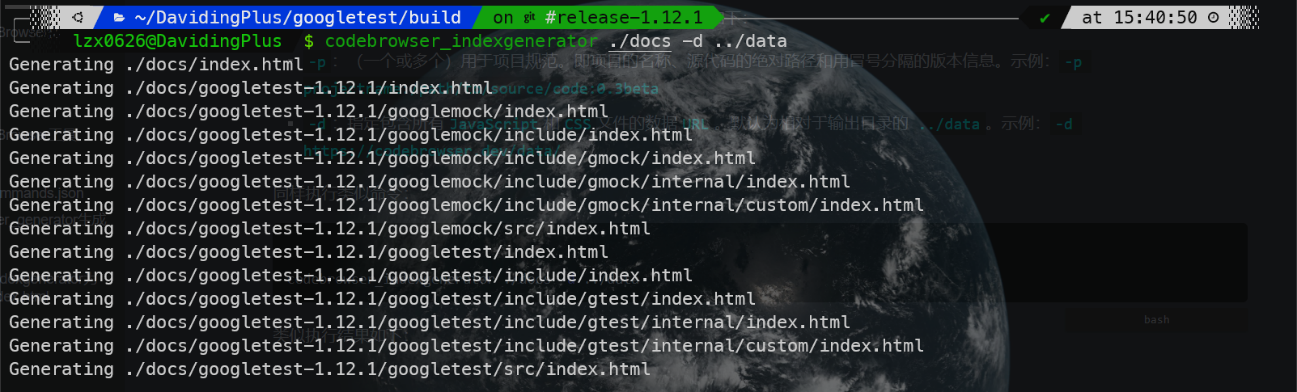


图2-22 codebrowser\_indexgenerator命令执行结果

**4、关于-d参数以及资源文件**

上面提到，为什么我自己使用的是-d ../data呢？因此需要先弄清楚data目录是干嘛的。

data目录存放着前端代码需要的资源文件，例如CSS、JS和图片资源等。因此需要指明这个目录的路径，绝对路径或者相对路径。由于我的[code-browser](https://github.com/DavidingPlus/code-browser/)项目需要存放很多库的源代码，因此我将data目录统一在根目录中，每个库的源代码对应一个自己的子目录，因此最终的结果是这样的：

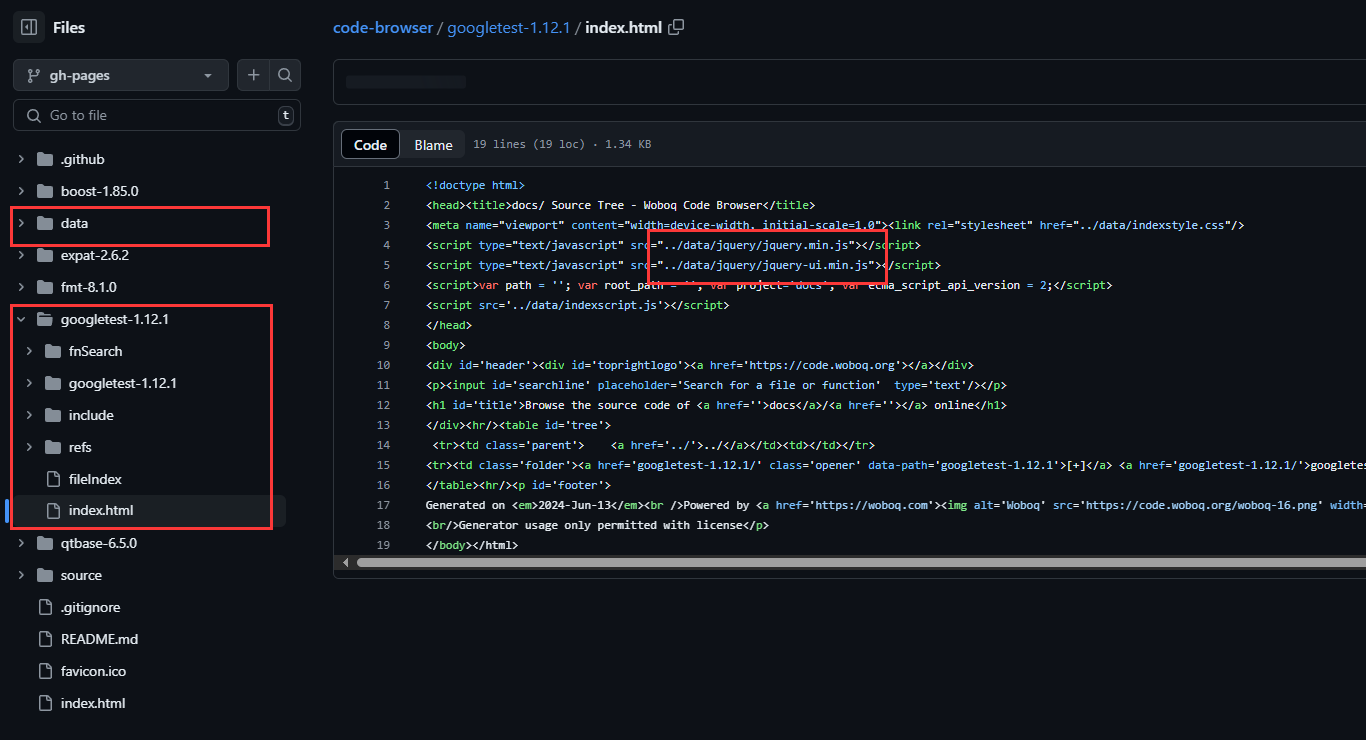


图2-23 code-browser项目结构图

这样就能正确定位到资源文件的路径了，codebrowser\_generator生成的html文件同理。用户使用自己的习惯和方式对应处理即可。

### 第三章 知识技能学习情况

本部分将从知识技能学习的主题出发，从开发环境和工具、预备知识、新知识点的学习出发，阐述实际工作过程中所涉及到的技术栈、开发环境、工程思想等，做一个总结性收获的记录。

### 3.1 开发环境和工具

### 3.1.1 开发环境

LarkSDK是跨平台的，在很多操作系统上都能构建，主要是windows和Linux。在实习的初期阶段，主要在Linux操作系统上进行研发，使用的Linux系统发行版是Ubuntu 20.04，由于项目的语言是c++，语言标准规定为c++11，同时需要使用编译器gcc，版本是gcc 9.4.0，同时构建需要的CMake版本不小于3.12。但是在中期以后的工作中，由于部分功能涉及到跨平台的校验，例如文件系统filesystem的相关处理，需要依赖windows的环境，从而兼容跨平台了。目前编译LarkSDK框架的windows系统推荐使用win10，使用的编译器是微软官方提供的MSVC14+，对应VS的版本是VS2019。

由于LarkSDK是一个跨平台的底层c++框架，因此兼容多个操作系统，例如linux和windows，是必然需要考虑的问题，在LarkSDK中体现的最突出的就是不同平台下的窗口逻辑处理的相关机制，包括但不限于事件循环，绘图机制等。目前LarkSDK中考虑了win32、x11和wayland的平台差异并分别处理，同时采用抽象父类指针多态的模式进行统一管理。同时，在其他模块的工作中或多或少都涉及到了跨平台的相关需求。

### 3.1.2 开发工具

关于代码IDE，windows下推荐使用VS，Linux下使用VS Code即可。VS完整的工具链为windows环境下的构建、测试等提供了非常便捷的操作，缺点则是工具太大，且功能过于复杂，甚至有些繁琐；而VS Code，由于具有众多插件，并且IDE非常轻量，因此非常适合在Linux下进行使用。

关于代码托管工具，毫无疑问是git。至于平台，我们是在公司服务器上部署gitlab。gitlab开源，并且可以通过脚本或者docker轻松的部署在内网当中，对于任何企业，特别是像我们这样的军工企业，就非常方便了。同时同前面提到，gitlab的CI等功能，可以方便实现自动化打包、扫描等需求，开发流程一目了然。

### 3.2 预备知识

由于项目中使用的是c++语言，因此一定的c++语言基础是必不可少的。这其中包括但不限于基础语法、关键字、命名空间、输入和输出、函数重载、引用等等。另外c++作为一门面向对象的语言，理解类和对象也是必不可少的，其中包括但不限于面向对象和面向过程的区别、类的定义、类的封装、类的作用域、初始化列表、内部类等等知识的熟悉和掌握。另外由于LarkSDK是一个偏底层的基础平台库，其中很多的基础类例如LVector、LString等都和std的标准容器有着很大的联系，因此对c++的STL有一定的了解也是很重要的，每种容器的内存模型和管理、设计思路等，都会在项目中无时无刻不被运用。

其次，由于LarkSDK主要在Linux上进行研发，因此Linux基本的命令是非常必要的，包括但不限于ls、rm、cd、mkdir等。同时需要理解Linux系统和windows系统的区别，包括但不限于文件系统、命令系统等。

同时由于项目托管使用了git，因此需要会使用git的基本命令，例如git pull、git merge、git push等等。这些都会在实际工作当中时时刻刻被运用。

### 3.3 新知识点学习和掌握情况

### 3.3.1 CMake

[14]在使用IDE开发软件的过程中，代码的编译和构建一般是使用IDE自带的编译工具和环境进行编译，例如VS，JetBrains全家桶等，开发者参与的并不算多。但是如果想要控制构建的细节，则需要开发者自己定义构建的过程。而CMake工具是一个开源、跨平台的编译、测试和打包工具，它使用比较简单的语言描述编译、安装的过程，输出Makefile或者project文件，再去执行构建。

CMake工具具有很多优点。第一，CMake 是一个跨平台的构建工具，可以在各种操作系统上使用，包括 Windows、Linux、macOS 等。第二，CMake 允许开发者使用简单的语法编写项目的构建规则，然后根据这些规则自动生成特定平台下的构建系统文件，如 Makefile、Visual Studio 解决方案、Xcode 项目等。第三，CMake 使用 CMakeLists.txt 文件来描述项目的构建规则和依赖关系。这些文件可以分成多个模块，使得项目的组织结构更加清晰，并且易于维护和管理。第四，CMake 支持各种主流的编译器和构建工具，包括 GCC、Clang、Visual Studio、Xcode 等，以及各种构建系统，如 Make、Ninja 等。第五，CMake 支持生成多种不同配置的构建系统，例如 Debug、Release、MinSizeRel 等，这使得开发者可以针对不同的需求生成不同的构建版本。第六，CMake 允许开发者方便地管理项目所依赖的外部库和模块，可以通过 find\_package() 函数来查找已安装的库，并自动配置相关的构建规则。最后，CMake 拥有庞大的用户社区和活跃的开发团队，提供了丰富的文档、示例和支持资源，使得开发者能够快速上手并解决遇到的问题。

关于多个c/cpp文件的编译，用MakeFile自然是没有问题的，但是一旦文件架构复杂起来，MakeFile文件的编写就比较困难。CMake就是在MakeFile的基础上，提供了全自动化的构建方案，只需要做好配置，就能自定义构建的过程，这也是CMake最方便的地方。

### 3.3.2 Conan

在任何项目的当中，除非是从头造轮子，引入一些第三方库是必不可少的。在前后端当中有比较完善的包管理器例如前端的node，后端的maven；在python当中有pip/pip3；在Ubuntu当中有apt，在centos中有yum等等。但是对于c++，我以前从来没有听过任何的第三方库或者包管理器，这也和c++是偏底层的语言，习惯造轮子比较相关。但是实际项目里面确实需要依赖一些第三方开源库，例如LarkSDK的libpng库，这个时候Conan包管理器就发挥它的作用了。

[15]Conan 是一个开源的 C/C++ 软件包管理器，用于帮助开发人员管理项目中的依赖项。它的主要目标是简化 C/C++ 项目中的依赖项管理过程，提供了一种方便的方式来引入、构建和共享依赖项，同时解决了版本冲突和平台兼容性等问题。Conan包的安装也非常简单，只需要通过pip即可。

Conan有很多优点。第一，Conan 可以在各种操作系统上运行，包括 Windows、Linux 和 macOS 等。第二，Conan 允许用户从中央仓库或自定义仓库中获取预编译的软件包，并将它们安装到本地环境中。第三，Conan 能够解析项目依赖项的依赖关系，并自动下载和安装所需的依赖项，包括库文件、头文件和其他构建工具等。第四，Conan 允许用户指定所需依赖项的版本，并在需要时进行版本切换或升级。第五，Conan 允许用户根据项目需求自定义构建选项，并将这些选项传递给依赖项的构建过程。第六，Conan 可以与各种常用的构建系统集成，包括 CMake、Makefile、Visual Studio 等，以便将依赖项集成到项目的构建过程中。最后，Conan 支持与其他包管理系统（如 CMake、vcpkg 等）集成，以便在不同的项目和环境中共享和重用依赖项。

在LarkSDK中需要引入很多第三方开源库，在LarkTestKit中则需要引入LarkSDK库。将LarkSDK和LarkTestKit的源文件编译成为对应的库文件，再和头文件打包在一起，通过Conan包的相关命令，就能很方便的将结果打成Conan包，这不仅对于后续测试人员做测试、版本发布等，都起着非常重要的作用。至于Conan具体的使用细节，官方文档当中都写得非常清晰明了，同时前面第二部分已经阐述过了，这里不再赘述。

### 3.3.3 对实用工具的深入理解

在以前，关于STL，也就是c++的标准模板库，我只是会用，稍微了解内部原理的状态。但是由于代码走查的工作，我接触到了不同的底层设计，也借机会对比了我们的实用工具和STL的设计思路。

以std::string和我们的LString为例对比。std::string对应c++的标准字符串，使用的单字节编码，并且只支持ASCII或者ISO-8859-1编码；LString使用的是双字节编码，因此可以支持unicode中的字符，应用空间更广。其次，std::string的内存结构是栈区存放堆区的指针，而堆区存放C风格的字符串（当然std::string还有其他的优化，这里仅声明绝大多数情况）；而LString继承LVector<unsigned short>，字符串中的每个元素都是一个unicode字符，这样大大复用LVector的功能，对于底层的内存管理也交给LVector处理。同时LString提供了更多的功能，例如split、trim、format等等。

以上的例子还有很多，通过代码走查的工作，我对底层实用工具的理解更深了，虽然可能达不到自己动手实现的水平，但是能够大致说出不同的设计思路，实现方案，也算是达到了目的。

### 3.3.4 C++ Boost后备标准库的知识

[16]Boost是为C++语言标准库提供扩展的一些C++程序库的总称。Boost库是一个可移植、提供源代码的C++库。作为标准库的后备，是C++标准化进程的开发引擎之一，是为C++语言标准库提供扩展的一些C++程序库的总称。

Boost社区建立的初衷之一就是为C++的标准化工作提供可供参考的实现，Boost社区的发起人Dawes本人就是C++标准委员会的成员之一。在Boost库的开发中，Boost社区也在这个方向上取得了丰硕的成果。在送审的C++标准库 TR1中，有十个Boost库成为标准库的候选方案。在更新的TR2中，有更多的Boost库被加入到其中。从某种意义上来讲，Boost库成为具有实践意义的准标准库。

具体到本项目的任务上，本项目原则上不应直接引用Boost库，然而，由于Boost库中包含了很多已经达到标准库水平但标准库尚未实现的功能模块，因此可以将其作为参考，挖掘原理以后采取自行编写的方式。例如util模块中的LUuid、LMd5，xml模块的LXmlStream类等都在Boost库中都有相应的体现。

### 3.3.5 Xml基础知识

[17]可扩展标记语言(Extensible Markup Language, XML)，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。XML 是一种独立于软件和硬件的工具，用于存储和传输数据。在电子计算机中，标记指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种的信息比如文章等。它可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。它非常适合万维网传输，提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。是Internet环境中跨平台的、依赖于内容的技术，也是当今处理分布式结构信息的有效工具。

具体到本项目的任务上，我在进行Xml模块的代码走查的时候，首先需要了解Xml的一些基础知识，比如语法定义、语句含义等，这样才能为我们自己的解析、读取和写入Xml流打下基础。虽然没有深入挖掘Xml的更多特性或者高级用法，但是Xml确实是一种高度自定义化的存储和传输数据的语言，非常方便好用。

### 3.3.6 对STL的进一步理解

在重构关联性容器LMap、LHash、LSet的时候，初版的代码是采用从头手写造轮子的方式，虽然功能可用，但是难以达到商用的需求。因此经过代码走查以后，决定采用封装STL标准库容器的方式实现。我负责这部分的工作，由此进一步加深了对底层红黑树、哈希表的理解。

以标准库的map和unordered\_map为例，map的底层是红黑树，unordered\_map的底层是哈希表。二者提供的功能类似，都是通过键来快速查找值，但是实现方式却完全不同，导致使用的时候需要注意的细节也不一样。

红黑树（Red-Black Tree）是一种自平衡二叉搜索树，通过一组严格的规则来保持树的平衡，确保插入、删除和查找操作的时间复杂度为O(log n)。其主要特性包括：每个节点是红色或黑色，根节点是黑色，所有叶子节点（NIL节点）是黑色，红色节点的子节点必须是黑色，从任一节点到其每个叶子的所有路径包含相同数量的黑色节点。为了维护这些特性，红黑树通过左旋和右旋操作调整节点的位置。红黑树的这种高度平衡特性，使其在插入和删除操作后，树的高度始终保持在较低水平，从而保证操作的效率。红黑树广泛应用于计算机系统和标准库容器中，如C++的std::map和std::set，在实际应用中，它提供了高效的动态数据集管理，使得各种操作在大多数情况下都能在对数时间内完成。这种数据结构的稳定性和高效性，使其成为许多算法和系统实现中不可或缺的一部分。

哈希表（Hash Table）是一种高效的数据结构，用于实现平均情况下常数时间复杂度的插入、删除和查找操作。其基本原理是通过哈希函数将数据的关键码映射到表中的一个位置（即哈希值），然后将数据存储在这个位置。哈希表主要由两个部分组成：哈希函数和存储数组。哈希函数的设计至关重要，决定了数据在表中的分布情况，理想的哈希函数能将数据均匀地分布在整个数组中，从而减少冲突（多个关键码映射到同一位置的情况）。为了处理冲突，常用的方法包括链地址法和开放地址法。链地址法在每个数组元素中维护一个链表来存储所有哈希值相同的元素，而开放地址法则在冲突位置寻找下一个可用位置。在C++标准库中，std::unordered\_map就是一个基于哈希表实现的容器，通过哈希表提供快速的键值对存储和查找功能。虽然哈希表在最坏情况下的时间复杂度为O(n)，但通过设计良好的哈希函数和适当的负载因子，哈希表在大多数实际应用中的性能非常优越。

即便是封装标准库已有的容器，但在实际处理的时候，我仍然遇到了一些问题。其中一个不小的问题就是关于迭代器。为了兼容Qt用户的使用习惯，迭代器的\*运算符重载需要返回元素值data，因此不能直接简单的直接使用标准库对应的迭代器，而需要再做一层封装。这个时候问题来了，如果在使用迭代器的时候越界了怎么办？LHash是单向迭代器，LMap是双向迭代器，而标准库对于越界的处理是很模糊的，尤其这两个东西都涉及到指针，因此稍不注意就会发生内存泄漏的问题，紧接着就是一系列的段错误core dump，结果可以称之为灾难性的打击。因此我不得不深挖标准库的源码，找出迭代器++或者--到底发生了什么操作，这样才能知道原因，并且给出合理的解决方案。经过调研，目前在迭代器中引入容器的size以及当前访问的下标index辅助判断迭代器的操作是否越界，方便抛出异常，这才是一个完整的问题的解决过程。

### 第四章 结束语

### 4.1 实习工作完成情况总结

结合前面的内容，我的实习工作总结为以下几点：

(1) 负责LarkSDK的部分代码走查工作。审核走查现有的老版本的代码，总结归纳相关问题，形成文档，并于会议讨论，形成解决方案。

(2) 负责LarkSDK的部分代码优化重构工作。针对代码走查中出现的问题，根据解决方案实行修改。

(3) 负责LarkSDK的部分新功能的开发。根据实际应用过程的产生的新需求，进行调研工作，总结文档，形成设计方案，并予以执行。

(4) 负责LarkTestKit的协助工作。审核该部分源代码，与校方同学和测试沟通，并设计编写测试用例。同时协助校方同学完成该部分的工作，讨论部分设计思路、实现方案等。

各个部分在实习期间具体完成的内容已经以图表的形式罗列在前面了。总体而言，我对自己的实习目标完成的还是相当不错的。对于公司目标，我严格按照公司的安排，全身心投入合迅智灵基础平台的研发，完成了很多有意义的工作。对于个人目标，我在实习期间跟随前辈的脚步，保质保量完成好安排到的工作，同时在实习期间学到了很多知识和技术，对以后的深造和就业有着非常重要的帮助。这是一个双向互利、共赢的过程。

### 4.2 对于企业实习的收获及体会

在实习期间，我们部门的工作基本分为两个板块，交付和研发。交付就是使用Qt，根据客户的需求，很多都是一些军工企业和研究所，去做一些项目。研发就是领导带领我们一起做Lark5产品，主要是[LarkSDK](https://larksdk.davidingplus.cn/)的相关工作。而我参与的就是研发工作，也是非常有意思和有意义的工作。

在进入公司之前，我就一直就有疑惑，为什么我们需要做一个自己的国产的C++通用开发框架呢？当时对于本科学生的我而言，我只把它当作是一门提升C/C++代码水平的途径。所以，当我的同学问我，公司费时费力做这个的意义是什么呢？我答不上来。现实情况也的确如此，业界对于跨平台的C++图形开发框架，业界第一反应都是Qt。是啊，我在进入实验室以前都一直以为实验室是在做Qt外包开发，甚至我还专门写了一个[扫雷](https://github.com/DavidingPlus/mine-sweeper)来准备面试。Qt在这个领域似乎是垄断的存在。并且公司的交付项目都是使用Qt框架开发的，包括另个组在进行的[合迅智灵插件框架](https://www.sinux.com.cn/product1.html)的开发，也是基于Qt框架做的二次加工。说白了，我们是拿着别人写好的东西去做开发。至于Qt内部是怎么实现的，我们作为用户不用关心，只需要知道这样能解决问题就行。

和Qt Framework一样，LarkSDK是一款跨平台C++开发框架。它使得用户无需花费精力在操作系统的特性和底层调用细节，从而更专注于业务需求开发。LarkSDK并非是任何其他框架的套壳产品，这意味着，除了引用一些不得不使用的基础轮子之外，LarkSDK的所有代码都是完全自研的。这就是我们这个产品LarkSDK的概述。更多细节请参见[合迅智灵LarkSDK用户手册](https://larksdk.davidingplus.cn/)。好，还是那个问题，既然Qt都有成熟的框架了，为什么我们还要费尽心思做一款国产的LarkSDK呢？

为了国产生态。目前，国内在软件开发平台领域的缺失，导致国内软件行业在开发软件时不得不大量使用国外厂商的开发平台，从而遭遇信息安全存在隐患、国产操作系统环境适配性差、维护服务没有支持、版权使用存在风险等重要问题，没有国产软件开发平台支持，软件开发环节效率低、软件开发成果不稳定也成为国内软件行业普遍存在的问题。对标到我们的LarkSDK产品上，虽然Qt是开源的，它的源代码就摆在那，但是它毕竟是老外写的，不受国人完全掌控，当然也没有哪个神仙去一行一行扒Qt的源码看有没有埋雷。在这样的背景下，我们无法预测未来的某一天会不会出现类似被卡脖子的问题。像华为芯片事件和微软蓝屏事件就是最好的印证。更不用说Qt还有社区版和商用版的区别了。万一某些东西就是只能通过商用版得到，那后续的一切都受制于别人。尤其我们还是一个军工企业，这样的问题就更应该得到重视。还是那句话，技术只有掌握在我们自己手中，国家发展的命脉才不会被别人牵着走。我可以不用，但我不能没有。这一点国家已经开始意识到了，例如国产操作系统银河麒麟、华为鸿蒙等等。而我们致力于在基础软件开发平台这一赛道，制作出一款属于我们国人自己的，更懂国产生态的LarkSDK。这条路并不容易，但我们坚信这是一条正确的路，难而正确的路。

同时，在公司实习了半年，我学到了太多有用的知识和技术。除了学习到技术层面的知识以外，更多是程序员职业素养的一些思考。这里我总结了三点。

首先是重视设计和思考。由于LarkSDK产品是完全从头开始，独立自研的，我们的目标是对标Qt，甚至是替代乃至超越Qt。不可否认我们参照了行业巨头Qt的很多思路，但是人无完人，强如Qt的设计也存在很多不合理，有缺漏的地方。当然Qt自己也在后续的版本中自己在努力修改，但是Qt毕竟是一个完整的产业链产品，有些东西久了是改不掉的。而LarkSDK作为一个新鲜血液的产品，目前非常轻量。我们有必要，并且是必须在搭建底层地基的时候就考虑好整体的架构和设计，例如如何设计窗口和组件的关系，如何设计跨平台的统一管理调度等。这些功能不仅需要稳定可用，还需要为后续研发的上层功能留出口子。当然这个过程是一个非常复杂、繁琐并且困难的过程。在这个过程中，设计花费的时间远大于真正写代码的时间。但事实证明，这样的策略是行得通的。

其次是多阅读，多记录。我的老板曾经讲过，提升代码水平的最好办法就是阅读代码。在我个人看来，不仅是多阅读代码，更需要多阅读文档，同时多做好记录。单打独斗是成不了气候的，每个人都存在知识的盲区，因此需要不断学习，不断提升自己，当然在当今社会也是为了不被淘汰。同时，最好做好知识的记录，就像初高中在课本上做笔记一样，知识太多，难免会忘记，因此需要做好记录，后续方便复习回顾。我个人一直有写博客的习惯，后续回顾以前知识的时候能很快回想起来，同时这样也能增加心中的成就感，为自己的前进提供动力。在我调研标准库std::string的sso优化的时候，查询了很多博客和技术文档，最终才把具体sso优化的细节，它具体的内存模型是怎样的等问题研究明白。这样清楚以后应用到我们的LarkSDK中，问题自然就迎刃而解了。

最后，也是我觉得最重要的一点，端正态度，脚踏实地。没有解不出来的问题，有的只是半途而废的人。我的领导钟老师完美诠释了这一点。他对组员很有信心，愿意给我们安排各种各样的需求工作，培养我们的技术和能力，当然，他自己也以身作则，带头完成项目中最困难的部分。我觉得有这样的领导和前辈，是我的幸运，甚至我认为以后真正工作以后都很难再遇到这样的领导了。同时，从他身上我也看出了，问题并不是不可战胜的，只要不断探索，肯下功夫，并且团队协作，一起努力，问题就一定能迎刃而解。在这个过程中，技术和能力也就自然而然的提升了。

### 参考文献

1. Eberhart, M.. "Programming With Qt Writing Portable Gui Applications On Unix And Win32." (2016).
2. 郭涛,李震宁.正版化推动国产软件快速发展[J].电子知识产权,2011,240(09):59-60.
3. Ahmet, , Hadrovic, and A. Hadrovi훶. "Graphic Design Cover Books by Professor Ahmet Hadrovic." (2022).
4. Prasad, A., et al. "Interfacing Architecture between Telemetry and On-Board Computer for a Nanosatellite." 2020 IEEE Aerospace Conference (2020).
5. Davenport, Dennis E., et al. "The Boundary of Ordered Trees." J. Integer Seq. (2015).
6. 李文绩.计算机领域中图形与图像的区别与联系[J].黑龙江科技信息,2012(07):122.
7. Harder, J.. "Understanding CC Libraries panel." (2020).
8. Stroustrup, B.. "The C++ Programming Language, 4th Edition." (2013).
9. Awan, Muhammad Talha. "Linux vs. Windows: A Comparison of Two Widely Used Platforms." Journal of Computer Science and Technology Studies (2022).
10. 杨荣尊.Linux系统下的虚拟桌面显示协议实现[J].电子世界,2018,551(17):128-129+132.DOI:10.19353/j.cnki.dzsj.2018.17.067.
11. Schmidmeier, M.. "Hammocks to visualize the support of finitely presented functors." Journal of Algebra (2021).
12. Hoddie, P., and Lizzie Prader. "JavaScript for Embedded C and C++ Programmers." (2020).
13. Andrews, Henry, and A. Wright. "JSON Schema: A Media Type for Describing JSON Documents." (2019).
14. Butler, Branden A., and Ryan M. Richard. "CMinx: A CMake Documentation Generator." J. Open Source Softw. (2022).
15. Kiss, Áron. "An explorative analysis of managed CI/CD usage among open-source C/C++ projects." Production Systems and Information Engineering (2022).
16. Németh, Boldizsár, et al. "Defining C Preprocessor Macro Libraries with Functional Programs." Comput. Informatics (2016).
17. Klaib, Alhadi A., A. Milad, and Mustafa Almahdi Algaet. "A New Approach for Labelling XML Data." 2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT) (2021).

# 致谢

在这篇报告的最后，我想对实习过程中给予我帮助的家人、老师、领导、同事等，致以最诚挚的感谢。

首先我要感谢我的家人，尤其是我的父母，他们为我实习过程提供了重要的后勤保障。同时，没有他们也没有我的今天。其次我想感谢我的院内导师王伟东老师。虽然王老师没有直接参与项目的统筹管理，但是给予了我很多关心和指导，包括但不限于如何适应公司的环境，学校报告的提交流程等，为人耐心和蔼，是个非常好的导师。紧接着我想感谢我在公司的领导钟采奕老师和导师刘鹏江老师。无论是作为技术还是社会履历上的前辈，对我提供了莫大的引导和关怀，让我能快速适应公司工作的环境和节奏，快速融入整个项目组的研发进程。同时遇到问题也会一起讨论，分享技术和解决方案，共同进步。另外，我要感谢在我实习期间所有帮助我的人，这注定是我人生履历上浓墨重彩的一笔。

最后，祝学院实习政策越来越完善，同学们的实习做的越来越好。也祝我的公司合迅科技越来越好，祝LarkSDK越来越好。