TenonStudio 2022 开发总结

许中兴

智能软件研究中心 RISE 团队
中国科学院软件研究所

2022 年

从原型验证到产品化
演进中的实践 (Evolutionary Practices)

- 总体开发原则
- 测试
- 界面
- 链接
- 命令模式

总体开发原则

- 逻辑检查要完备，工业软件处理大量数值情形，不能有遗漏未处理的情况存在。什么时候写 assert，什么时候抛异常要仔细考虑。
- 设计意图要文档化，为什么要做这个检查，为什么要处理这种情况，当时记得清楚的，过后一定会忘记。有些可以直接写在注释中，有些复杂的设计写在 internals 文档中。
- 功能的定义要写在用户手册里，主要的按钮，配置参数。
- 要写测试，测试分为单元测试和交互测试。有时候为了写单元测试，需要对代码进行重构，以减少被测代码的依赖。
- 要持续改进，上面这些要求，不可能在第一遍做的时候就全部达到。所以代码不是写一遍就完了，要持续地重构。
- 自己拿不准的地方，拿出来讨论，并作记录，不要埋在那里。

项目的主线方向

- 以界面和交互为核心：美观，顺手
- 高优先级：程序控制、几何处理、智能仿真
- 低优先级：运动控制、图形渲染、物理仿真

测试

- 测试不是一个可以附带着做的事情，是一件需要专门做的事情，因为测试需要对程序做大量的重构。
- 单元测试
- 交互测试

单元测试

- 需要考虑的主要问题是如何让被测代码能够被独立测试
- 在实现功能的时候各种代码都混合在一起
- 在编写单元测试的时候需要把要测试的代码独立出来

写代码为什么会出现 bug

- 大部分的 bug 出现的原因是因为“注意力缺乏”
- 我们在写代码的时候需要关注的方面非常多
  - 大的方面：功能逻辑，模块化，接口设计，测试
  - 小的方面：命名，局部逻辑，API 的含义
- 但是人类一次能够关注的事物数量是有限的，所以如果写代码的时候想的方面太多，就会有疏忽
- 这也解释了为什么 review 代码的时候总能发现别人代码的问题，因为只需要关注少量的方面
- 为了尽可能避免写出 bug，我们在写代码的时候只关注较少的方面
- 第一次实现只关注功能逻辑正确性，把代码写对，写成“面条状”，后续再通过不断重构来改进。每次重构只改进一个方面

自动化的交互测试

- 目标：通过编写程序的方式向目标程序（一个 Qt GUI 程序）发送一系列鼠标键盘事件，从而达到自动化模拟鼠标键盘对 GUI 程序进行交互测试的目的。
- 方式：开发了一套能够以编程的方式完成自动化交互测试的库。
- 开发人员通过调用 API 的方式将需要执行的测试步骤写成测试用例
  - clickItem(QString id)
  - clickItemWithSignal(QString id, QString signalSignature)
  - clickItemChild(QString id, QString childName)
  - setInputText(QString id, QString text)
  - dragAndDrop(QString dragAncester, QString dragItem, QString dropAncester, QString dropItem)
- 效果：可以将几乎所有的交互操作写成测试用例进行“重放”

自动化 GUI 交互测试技术基础

- Qt 的 hook 能力
  - qtHookData
  - QHooks::AddQObject
- Qt 的 introspection 能力
  - QQmlContext::nameForObject
  - QQuickItem::objectName
  - QTest::MouseAction
  - QMetaObject::invokeMethod
  - QObject::inherits

界面基础知识

- 1pt 是 1/72 英寸，也就是说如果屏幕的 ppi(pixel per inch) 是 72 的话，pt 和 px 是 1 比 1 对应的。
- 10.5pt 字体是一般情况下的标准字体大小 (3.7mm)。
- 但是，1920x1080 23 寸屏幕是 96ppi(Windows 系统的默认值)。ppi 变大了 1.333 倍，px 数也应该变大同样的倍数才能让物理尺寸一样。
- 所以在网上各个 ptpx 换算器里，10.5pt 字体对应的是 10.5*1.333=14px。

HiDpi 屏幕下的处理

- Qt 6 没有提供手动调整放大系数的机制，而是采取读取系统放大系数的路线。Qt5 提供的若干个环境变量都不再推荐使用了。
- 以 Ubuntu Wayland 为例，在“辅助功能”中放大字体，也就是设置 text-scaling-factor 不能影响 Qt 的界面尺寸。在“显示”设置中选择 200% 整体放大，是可以放大 studio 的界面的，但是会破坏 OpenGL 的 anti-alias 效果。
- 所以我们还是决定提供两套界面方案，即根据一个配置项决定使用标准尺寸还是放大尺寸。如果用户愿意使用系统的显示整体放大，那么他可以使用标准尺寸，只是 OpenGL 的渲染效果稍差一些，但不影响使用。如果用户仅放大系统的字体，那么他可以单独设置 studio 的 hidpi 配置项，让 studio 使用大尺寸界面。

HiDpi 屏幕下的处理：实现方式

- 字体：27 寸 4k 屏幕的 ppi 是 163，大致是 96 的 1.7 倍，所以大尺寸字体采用 14*1.7=24px。
- 控件的放大：在 qml 中设置了一个放大系数 ratio，根据配置项决定 ratio 是 1 还是 1.7，理论上在各个控件尺寸处乘上放大系数即可。这种做法还附带着暴露了一些排版上的实现问题。比如原先有些排版位置是凑上的，而不是根据尺寸正确设置的，会被设置 ratio 暴露出来。

链接

- 全部的链接产物是 2 个可执行程序，debug 版本分别是 429MB 和 117MB
- 主要的 ldd 依赖是 Qt，因为 Qt 的静态链接支持不太好
- 传统的链接方式是每个模块作为一个.so 库，第一方和第三方的.so 都生成在一起，最后的主程序依赖同目录下的几十个.so，以及系统 lib 目录中的几十个.so
- 现代程序采用整体化的静态链接方式，例如 chrome，只包含一个主程序

链接：几类形式

- 程序自己的第一方模块，特点是相互依赖关系复杂。以 cmake object library 存在，在最后链接的时候存在的形式是一堆.o 文件，好处是不需要考虑依赖顺序
- 第三方模块，以.a 形式存在，特点是依赖关系简单，仅存在第一方到第三方的依赖。在 cmake 文件中指定依赖关系
- Qt, 以.so 形式存在，在 cmake 文件中指定依赖关系
- Qml Module, 以 plugin.o 加上.a 静态库的形式存在，这个是 Qt 规定的。链接时通过.o 文件将真正的依赖引入可执行文件中。
- 最终程序的链接时间 (AMD 5900X, 32G DDR4)
  - Gnu ld: 16 秒
  - lld: 1.2 秒

打包

- 目前将可执行程序、Qt 库、资源文件一起打成一个 tar 包。
- 可执行程序指定"-Wl,-rpath,$ORIGIN"
- 所依赖的系统库以图形栈在 Qt 之下的部分为主：X11, Wayland, OpenGL.


libdw.so.1 => /lib/x86_64-linux-gnu/libdw.so.1 (0x00007f185a8ec000)
libffi.so.8 => /lib/x86_64-linux-gnu/libffi.so.8 (0x00007f185a8df000)
libGLX.so.0 => /lib/x86_64-linux-gnu/libGLX.so.0 (0x00007f185a7dc000)
libOpenGL.so.0 => /lib/x86_64-linux-gnu/libOpenGL.so.0 (0x00007f1856732000)
libnsl.so.2 => /lib/x86_64-linux-gnu/libnsl.so.2 (0x00007f1856718000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f18557a2000)
libfontconfig.so.1 => /lib/x86_64-linux-gnu/libfontconfig.so.1 (0x00007f18566ce000)
libfreetype.so.6 => /lib/x86_64-linux-gnu/libfreetype.so.6 (0x00007f1854f36000)
libstdc++.so.6 => /lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f1853a00000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f1855fe0000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1853600000)
/lib64/ld-linux-x86-64.so.2 (0x00007f185a9bc000)
libelf.so.1 => /lib/x86_64-linux-gnu/libelf.so.1 (0x00007f1855fc3000)
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f1855fa7000)
liblzma.so.5 => /lib/x86_64-linux-gnu/liblzma.so.5 (0x00007f1855777000)
libbz2.so.1.0 => /lib/x86_64-linux-gnu/libbz2.so.1.0 (0x00007f1855f94000)
libbrotlidec.so.1 => /lib/x86_64-linux-gnu/libbrotlidec.so.1 (0x00007f1855f86000)
libEGL.so.1 => /lib/x86_64-linux-gnu/libEGL.so.1 (0x00007f1854f23000)
libX11.so.6 => /lib/x86_64-linux-gnu/libX11.so.6 (0x00007f18544c2000)
libxkbcommon.so.0 => /lib/x86_64-linux-gnu/libxkbcommon.so.0 (0x00007f1854edd000)
libpng16.so.16 => /lib/x86_64-linux-gnu/libpng16.so.16 (0x00007f1853cfd000)
libicui18n.so.71 => /lib/x86_64-linux-gnu/libicui18n.so.71 (0x00007f1853200000)
libicuuc.so.71 => /lib/x86_64-linux-gnu/libicuuc.so.71 (0x00007f1853805000)
libGLdispatch.so.0 => /lib/x86_64-linux-gnu/libGLdispatch.so.0 (0x00007f1853c46000)
libtirpc.so.3 => /lib/x86_64-linux-gnu/libtirpc.so.3 (0x00007f18535d3000)
libexpat.so.1 => /lib/x86_64-linux-gnu/libexpat.so.1 (0x00007f18535a8000)
libuuid.so.1 => /lib/x86_64-linux-gnu/libuuid.so.1 (0x00007f185576e000)
libbrotlicommon.so.1 => /lib/x86_64-linux-gnu/libbrotlicommon.so.1 (0x00007f185449f000
libxcb.so.1 => /lib/x86_64-linux-gnu/libxcb.so.1 (0x00007f185357e000)
libicudata.so.71 => /lib/x86_64-linux-gnu/libicudata.so.71 (0x00007f1851400000)
libgssapi_krb5.so.2 => /lib/x86_64-linux-gnu/libgssapi_krb5.so.2 (0x00007f18531ac000)
libXau.so.6 => /lib/x86_64-linux-gnu/libXau.so.6 (0x00007f1854ed7000)
libXdmcp.so.6 => /lib/x86_64-linux-gnu/libXdmcp.so.6 (0x00007f1854ecf000)
libkrb5.so.3 => /lib/x86_64-linux-gnu/libkrb5.so.3 (0x00007f1851337000)
libk5crypto.so.3 => /lib/x86_64-linux-gnu/libk5crypto.so.3 (0x00007f1853552000)
libcom_err.so.2 => /lib/x86_64-linux-gnu/libcom_err.so.2 (0x00007f1853c40000)
libkrb5support.so.0 => /lib/x86_64-linux-gnu/libkrb5support.so.0 (0x00007f1853c33000)
libbsd.so.0 => /lib/x86_64-linux-gnu/libbsd.so.0 (0x00007f185353a000)
libkeyutils.so.1 => /lib/x86_64-linux-gnu/libkeyutils.so.1 (0x00007f1853c2c000)
libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007f1853199000)
libmd.so.0 => /lib/x86_64-linux-gnu/libmd.so.0 (0x00007f185318c000)

命令模式

- 命令模式就是将所有的动作组织成为一个命令栈，可以 undo,redo
- 所带来的本质区别是，一个动作要实现成“可逆”的，需要对整个状态进行完整的记录，并实现对记录状态的恢复。对程序的架构和组织提出更高的要求。让整个程序的架构变得更好。

Where is the Algorithms?

- xuzhongxing.github.io
- 用 NURBS 曲面逼近一般光滑曲面的技术路线
- OpenCascade 中的 Thin Plate Spline 插值
- B 样条曲线及其导数的高效计算