采用技术

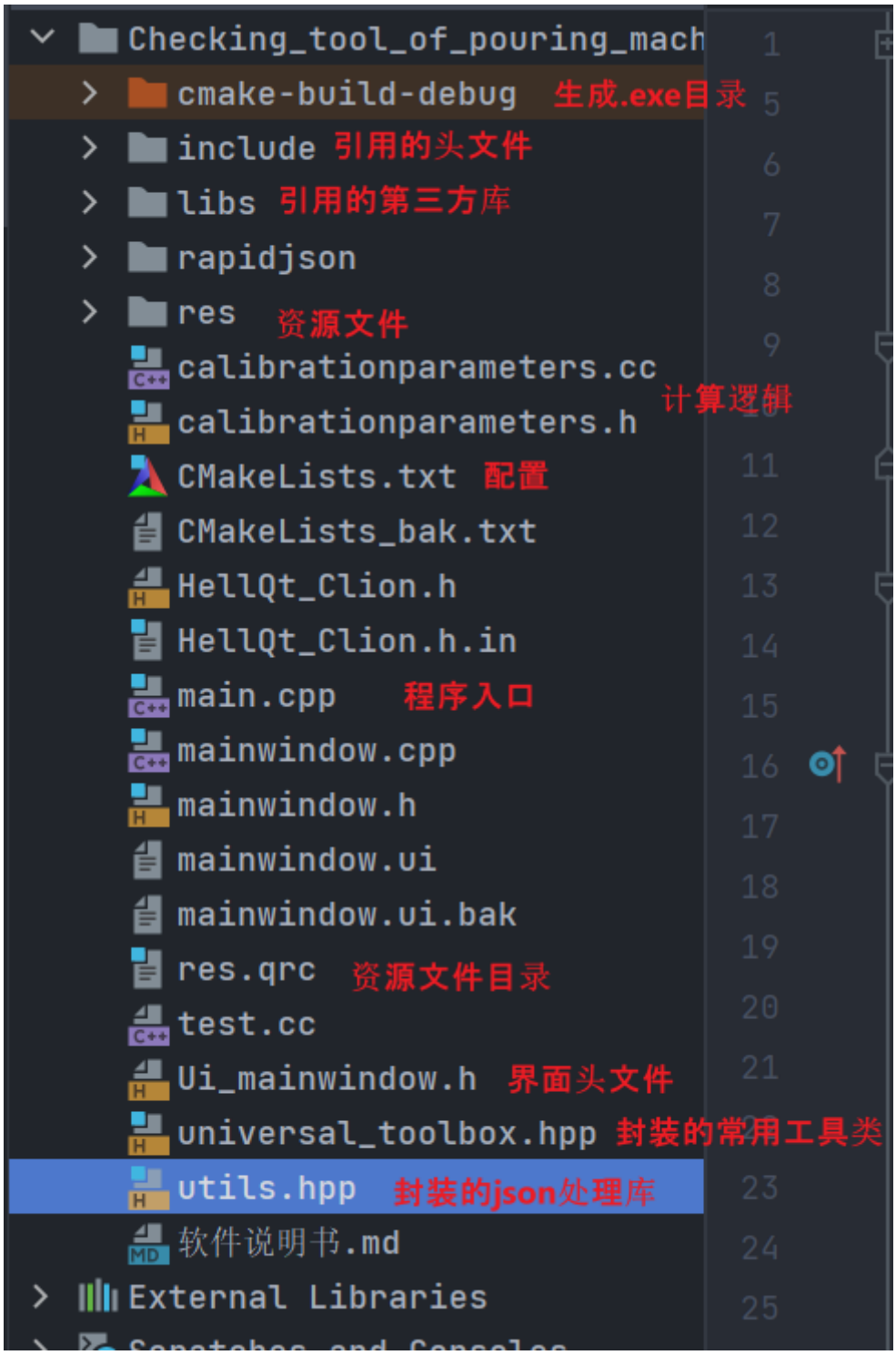
1. 后端 c++
2. 前端美化 qt + css
3. 通过本地文件进行持久化存储
4. 引用第三方库: jsoncpp , boost::asio (可以上网详情查询)
5. 采用 MVC 模式, 前后端分离, 业务代码, 与配置、代码分离.
6. 设计模式: 采用单例模式 + 抽象工厂 + 观察者模式.

总体逻辑

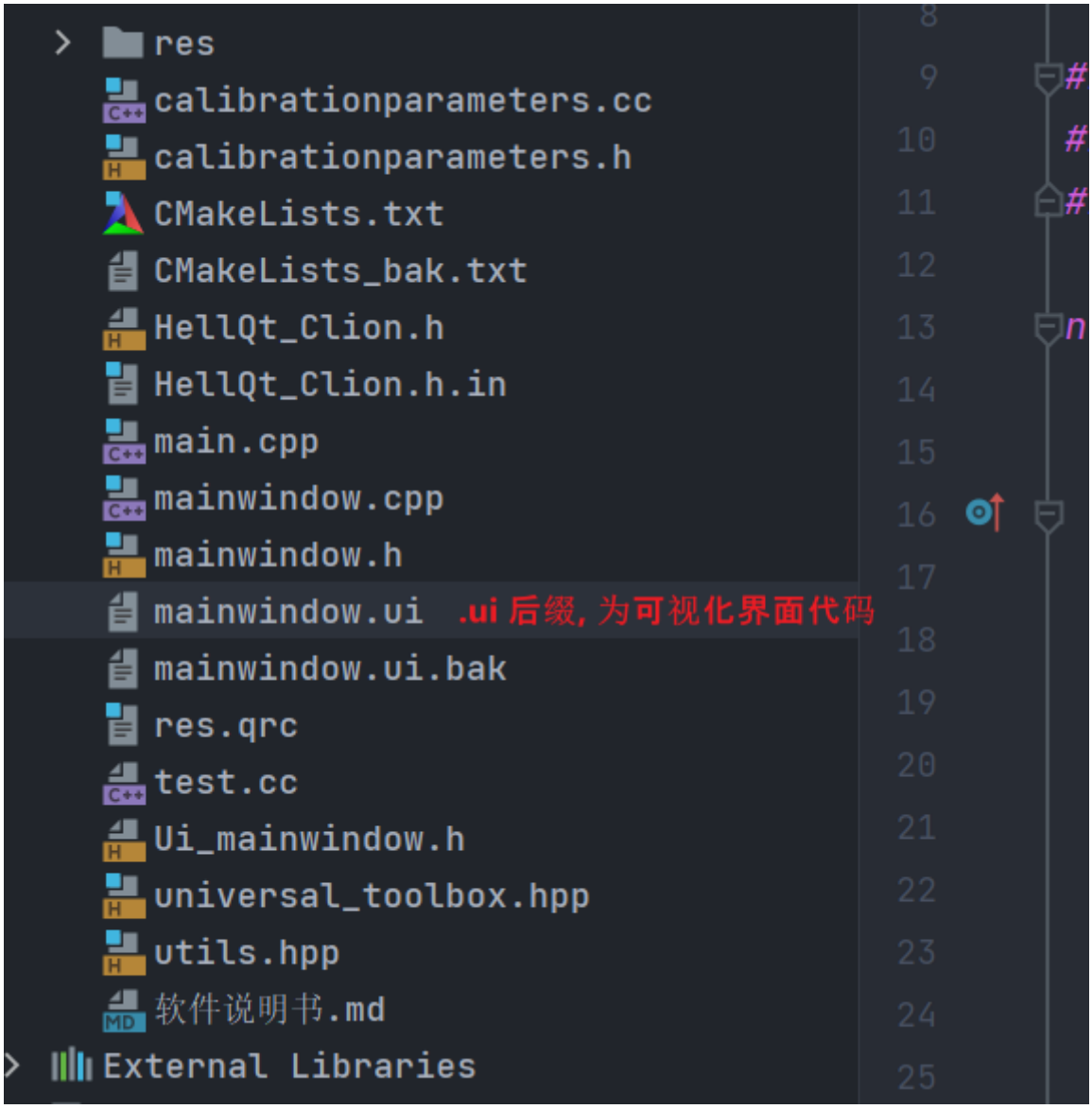
1. 构建界面 `xxx.ui`
2. 处理对应逻辑 `xxx.cc` `xxx.h`

3. 其中读取参数配置是通过json进行通讯. 此中采用 `jsoncpp` 第三方库进行使用

目录结构




res
area.jpg
d.jpg
f.jpg
icons8-robot-50.ico
inject_table.png
Injection_machine_meter.json
p0.jpg
p_mo.jpg
json动态读取配置文件
qs.jpg
time.jpg


res
calibrationparameters.cc
calibrationparameters.h
CMakeLists.txt
CMakeLists_bak.txt
HellQt_Clion.h
HellQt_Clion.h.in
main.cpp
mainwindow.cpp
mainwindow.h
mainwindow.ui
.ui 后缀, 为可视化界面代码
mainwindow.ui.bak
res.qrc
test.cc
Ui_mainwindow.h
universal_toolbox.hpp
utils.hpp
软件说明书.md
External Libraries

涉及计算

```
// 锁模力计算
connect( sender: ui->btn_suo_force , signal: &QPushButton::clicked , slot: [=]() -> void {

    ui->text_log->append( text: "<span style=color:'green'> 进行锁模力校核 </span>");
    double p_mo = ui->text_p_mo->text().toDouble();

    int f_suo = parser_->getModel( index: ui->comboBox_machine->currentIndex() + 1)["suo_force"].asInt();

    double act_suo = calc_->calc_F_zhang( p: p_mo);
    ui->text_log->append( text: "查表得锁模力: " + QString::number(f_suo) + "  实际锁模力: " + QString::number(act_suo));

    if (act_suo <= f_suo)
        ui->text_log->append( text: "<span style=color:'green'> 实际锁模力符合标准锁模力! </span>");
    else
        ui->text_log->append( text: "<span style=color:'red'> 实际锁模力 > 标准锁模力! </span>");

    ui->text_f_grow->setText(QString::number(act_suo));
```

- 所有的计算均在: `mainwindow.cpp` 中处理, 代码中均有注释.