LabTool设计文档

**目录**

[设计简介（项目背景、功能目的） 4](#_Toc491935929)

[项目背景 4](#_Toc491935930)

[主要功能 4](#_Toc491935931)

[设计目的 4](#_Toc491935932)

[设计原理 4](#_Toc491935933)

[系统框架结构图（原理图） 4](#_Toc491935934)

[主要技术应用 5](#_Toc491935935)

[系统描述 5](#_Toc491935936)

[界面展示 5](#_Toc491935937)

[功能模块 5](#_Toc491935938)

[后台配置 6](#_Toc491935939)

[日志系统 7](#_Toc491935940)

[文件管理系统 8](#_Toc491935941)

[软件架构 10](#_Toc491935942)

[数据结构 10](#_Toc491935943)

[设计模式 11](#_Toc491935944)

[优势分析 12](#_Toc491935945)

[Xml解析效率 12](#_Toc491935946)

[基于libssh2库的SSH实现 12](#_Toc491935947)

[智能指针 13](#_Toc491935948)

[GoogleTest编写测试代码 13](#_Toc491935949)

[完善的系统结构 13](#_Toc491935950)

[参考资料 14](#_Toc491935951)

[附录 14](#_Toc491935952)

|  |  |  |
| --- | --- | --- |
| 版本 | 修改时间 | 执行人 |
| V3.0.1.0 | 2017.08.30 | 余宗仁 |
|  |  |  |

#### 设计简介（项目背景、功能目的）

###### 项目背景

项目以实验室仿真环境建立为背景，设计并开发一款适合于实验室故障场景模拟的轻量化实验室工具。

通过工具来模拟各类基站和核心网的故障原因，使用工具一键式下发来修改参数或者下发命令，最终快速实现故障场景的构造。

本方案用于针对常规故障处理手段的演练，使用Nokia后台维护常用工具，包括SiteManager，emil，wireshark，site browser， remote，NetAct等，不包括Advanced平台工具，例如DO，FAST，ICES等。

###### 主要功能

LabTool主要包含两个部分，对基站的操作和核心网部分的操作。

对于基站部分主要是对BTS SiteManager的xml配置文件修改，获取xml文件、解析并修改参数、然后拼装成全新的xml文件并下发。对于下发的xml文件，可以有回退机制，也即是可以通过回退预设的xml文件来还原最初的配置。

对于核心网部分是通过实现windows到Linux的SSH通信连接，从而下发linux命令来完成参数的修改、查询等操作。

###### 设计目的

工具的设计目的旨在建立一个网络故障仿真实验室，为其提供高效有力的可快速构造故障场景的模拟软件。

#### 设计原理

###### 系统框架结构图（原理图）

LabTool Clinet

基站部分

核心网部分

SSH建立

XML配置文件获取

XML配置文件解析

对Linux命令下发

XML配置文件验证

XML配置文件下发

回退XML配置文件

###### 主要技术应用

LabTool是基于MFC开发的，开发语言是C++；代码中使用了boost标准库，使得代码简洁优美并且高效；封装和继承关系合理，并且整体贯穿设计模式思想使其更为高效。

使用了libSsh2库来完成从windows到Linux环境的SSH通信，利用库函数自主编写代码，简洁快速，并且还通过窗口显示来实时呈现出SSH连接建立后的命令执行结果。

#### 系统描述

###### 界面展示

作为一款实验室工具，以高效为目的，并未考虑太多相关的界面美化等因素，界面力求简洁、易用性强、可维护性强；工具力求稳定并且轻量化。整体界面展示如下图所示：

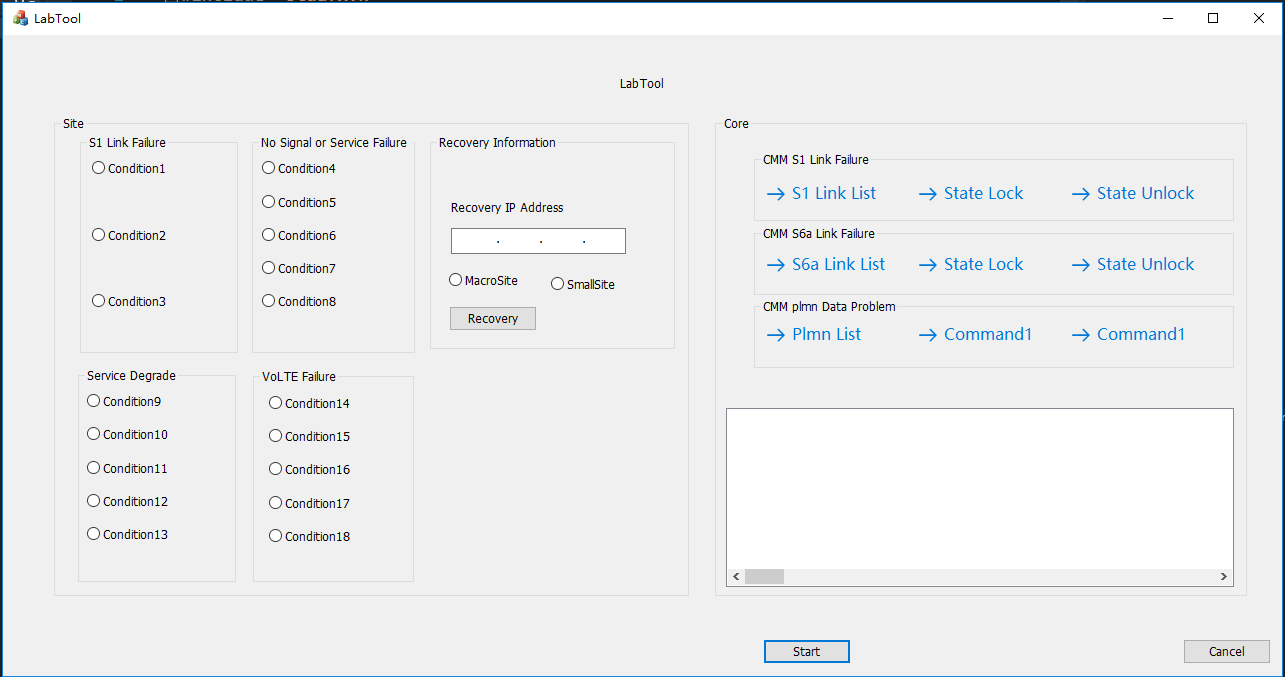


图1.整体界面截图

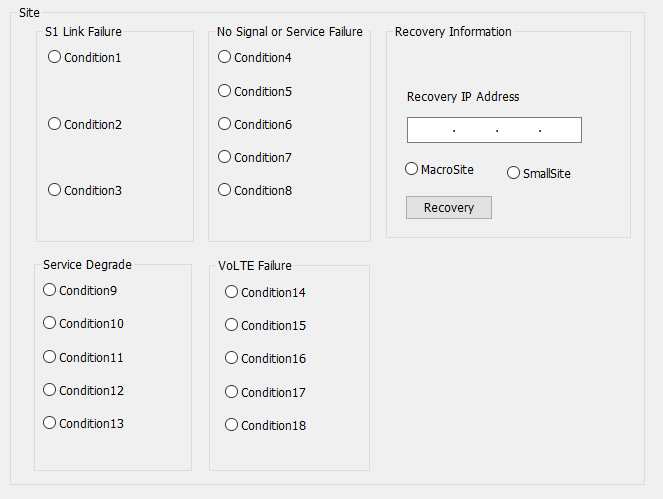
###### 功能模块

从界面来看，功能模块主要分为两大部分，分别是基站部分和核心网部分。软件通过解析后台配置的xml文件来获取需要进行操作的IP地址、用户名密码、调用CLI工具的路径。

界面通过失败场景分类和条件的选择，直接下发CLI命令。使用windows系统的CMD工具对CLI工具的调用；

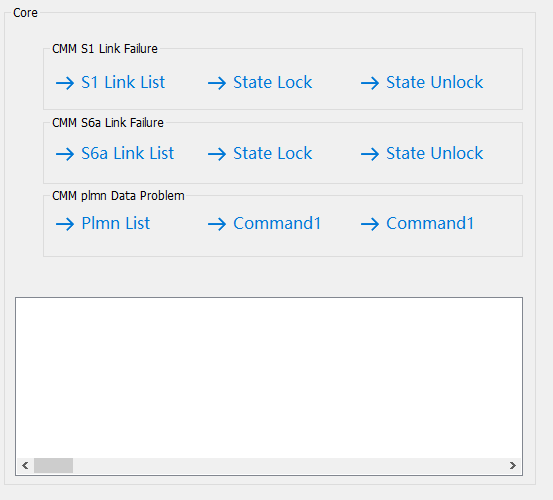
回退操作，回退采用和下发CLI命令相同的方式来进行调用，但是回退的配置文件为预置的配置文件，操作方式简单快捷。回退模块中只需要输入回退参数基站的IP地址，并且选择是小站还是宏站，点击恢复即可。

回退和下发都会有基站重启现象，所以中间过程需要等待。整个下发和回退时间为μs级别，主要耗时在对CLI的调用以及基站重启部分。



核心网部分的操作则是使用libSsh2库来完成从windows到Linux系统的SSH连接，并且建立连接后下发相应的Linux命令进行查询、链路锁定和解锁等操作。

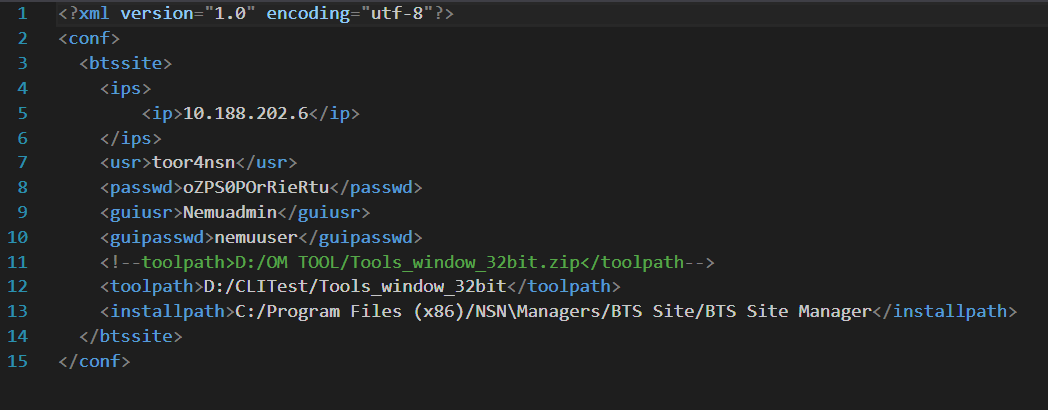
操作方式简洁明了，对相应模块的按钮进行点击即可快速建立SSH连接以及下发命令。在工具中开放了展示窗口，可以回显下发的执行结果，该功能类似一个小型的终端工具（如Secure CRT或者XShell）。

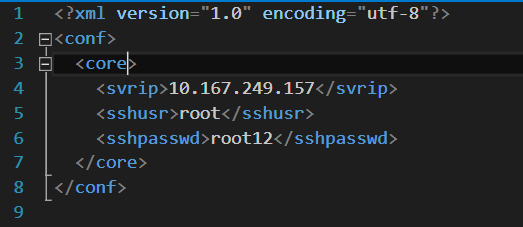


###### 后台配置

软件采用了后台配置xml的方式来存储相应的IP地址、用户名密码和路径；未在界面开放IP地址等信息是因为需要做到界面简洁，并且对于实验室环境来说，IP地址是相对稳定的，分配了之后几乎不用做任何修改，因此没有必要在界面进行配置。

对于两个功能模块，基站操作模块和核心网操作模块分别有两个xml文件，程序执行起来会对其进行解析，并获取相应的信息。

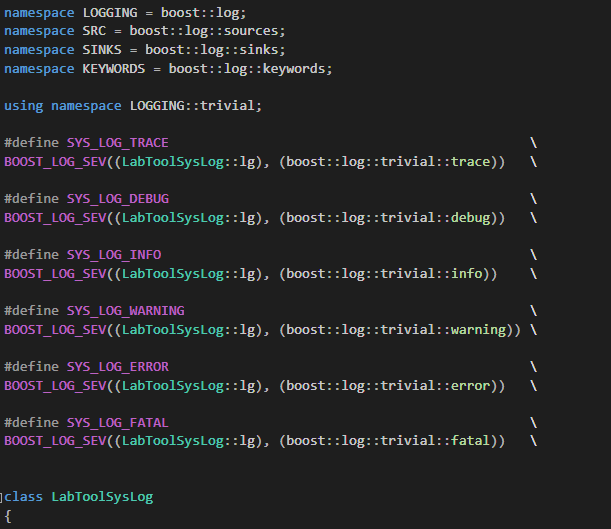




###### 日志系统

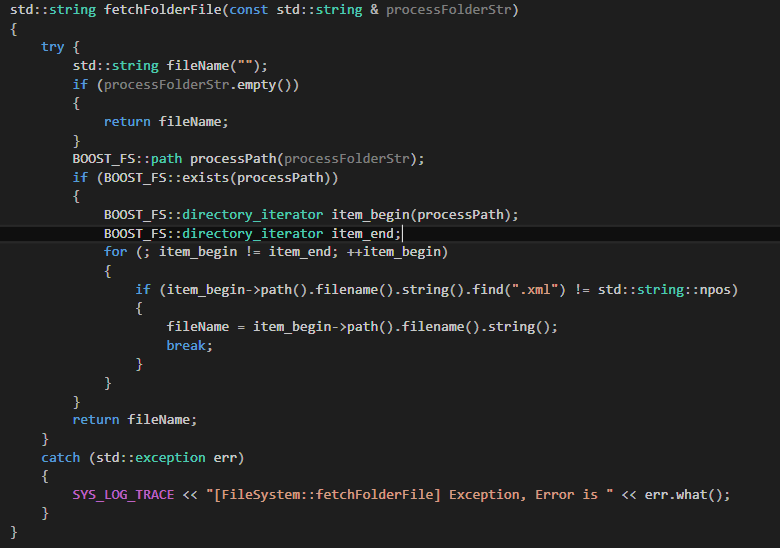
在软件中增加了日志系统以完成对每一步的记录，并将日志保存。

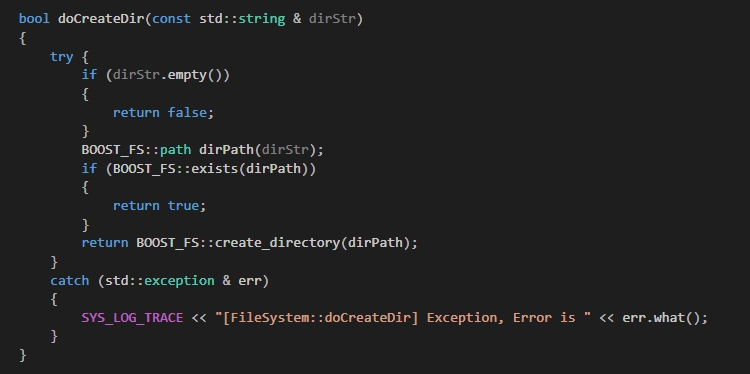
日志系统分类详细，可以通过设置来记录不同级别的日志。主要分为trace、debug、info、warning、error和fatal几个级别。

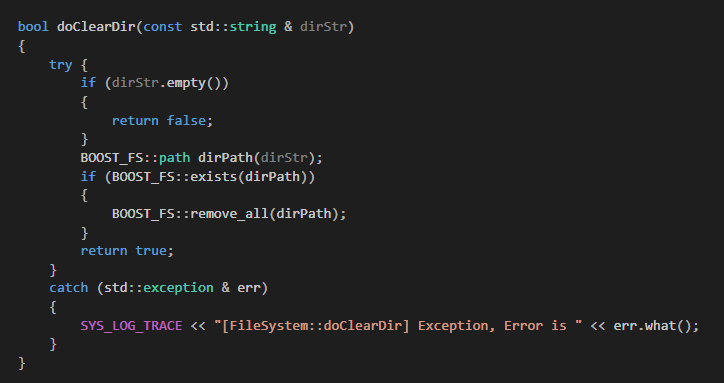


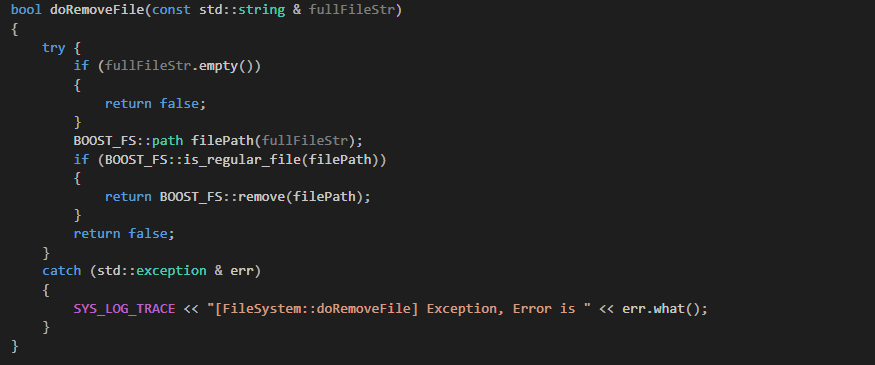
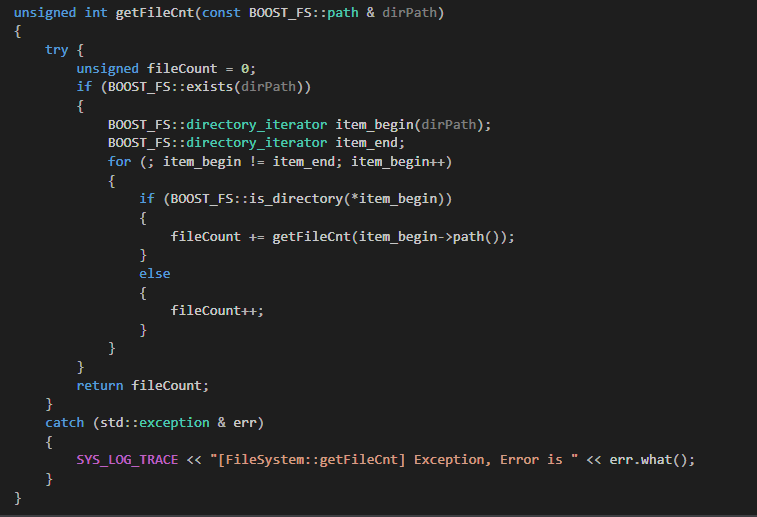
###### 文件管理系统

文件管理系统主要是软件中对获取到的xml文件进行管理，包括对文件的主要操作，对这些操作进行封装和提供相应的接口，使用更为便捷。主要提供了目录下特定文件获取、文件删除、文件拷贝、目录清除等一系列操作。





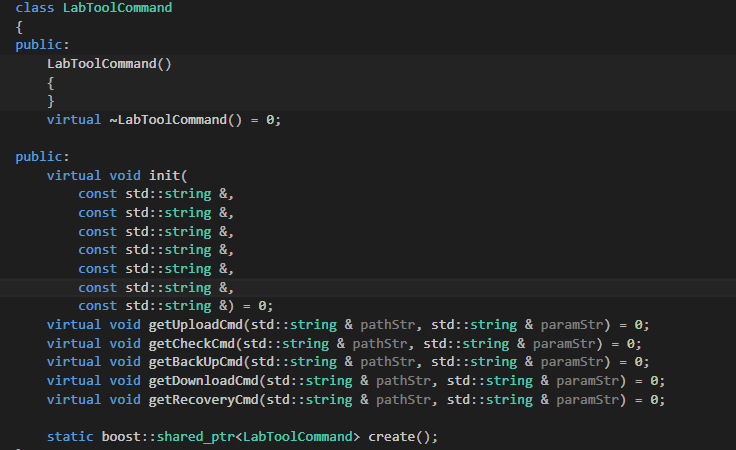


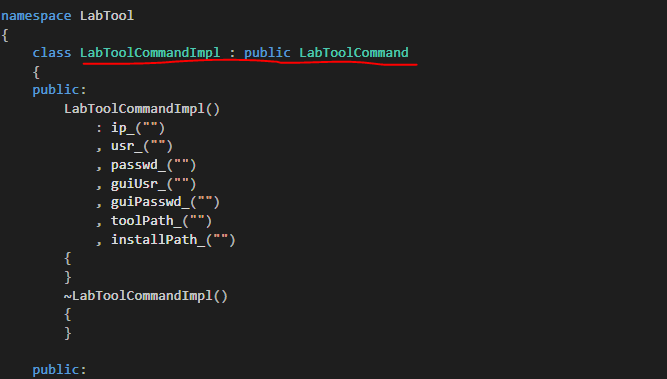


#### 软件架构

###### 数据结构

代码中的数据结构主要为两大块，一是数据状态（文件读取、下发、回退），另一类是下发命令（不同条件下发不同命令），均采用接口类（虚基类）的方式来完成；高内聚，低耦合的设计方式，结构设计合理并且所占用空间小。

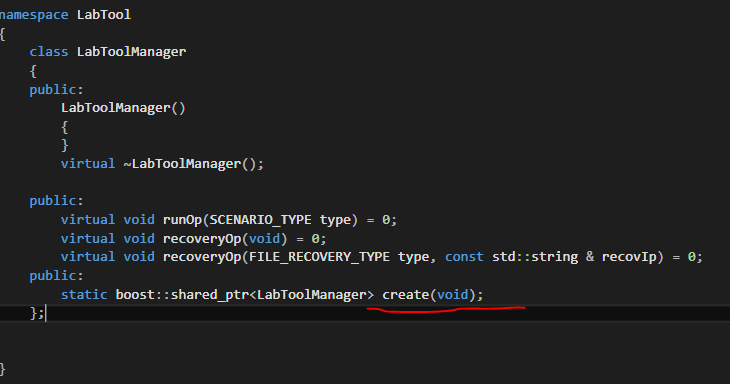


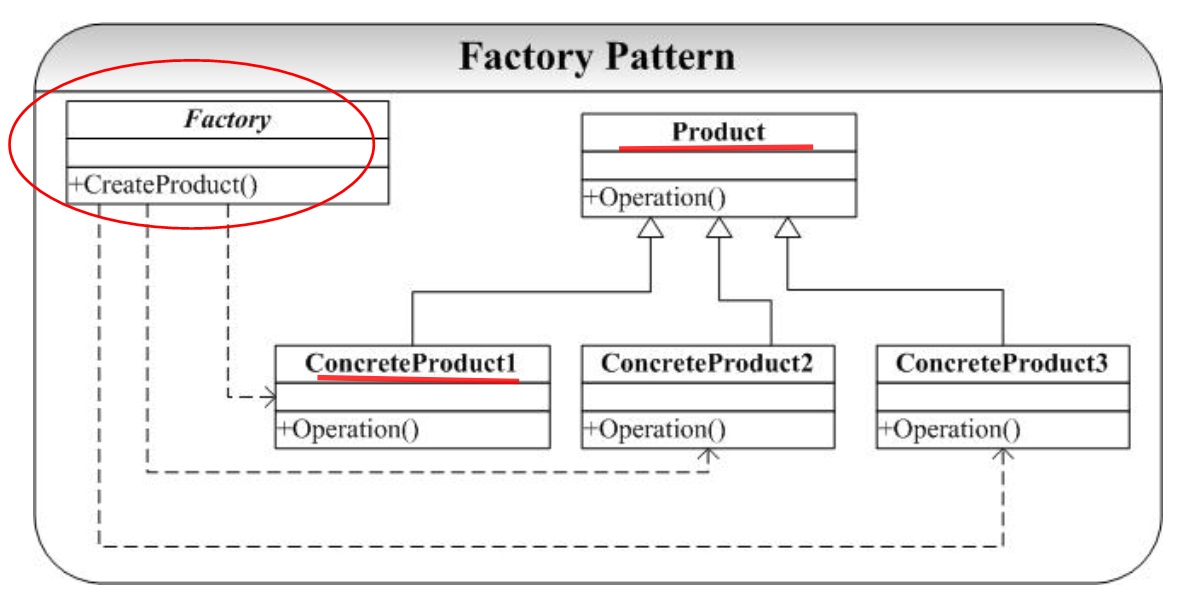


###### 设计模式

1.Factory模式

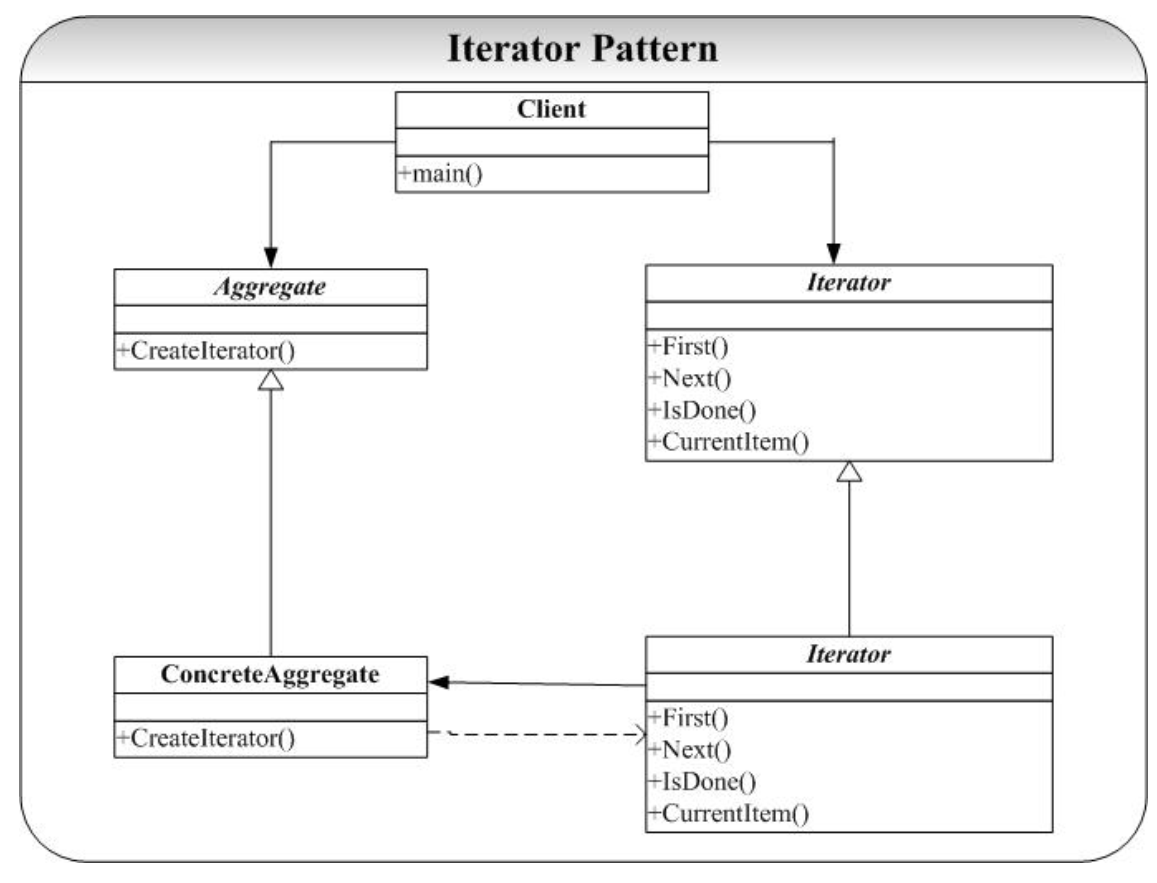
每一个功能模块的类采用的是Factory模式，每一个实例的产生都是由工厂来create的；





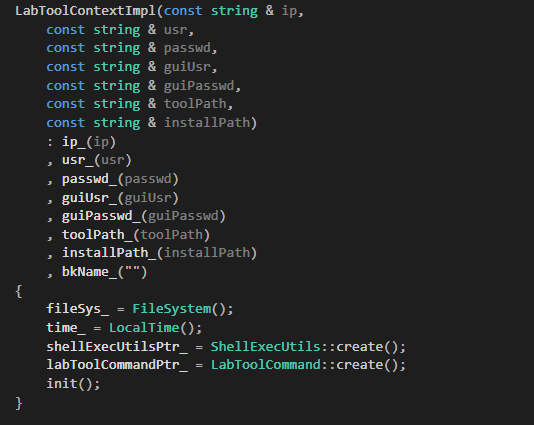
2.Iterator模式

使用STL的Iterator来遍历数据结构，减少存储压力并且改善内存管理。



3.Builder模式

软件中添加了context类，该类的实现方式就是Builder模式，在类中builder不同的操作来完成对所有操作的汇聚。在Context的构造函数中build出各个模块的智能指针。



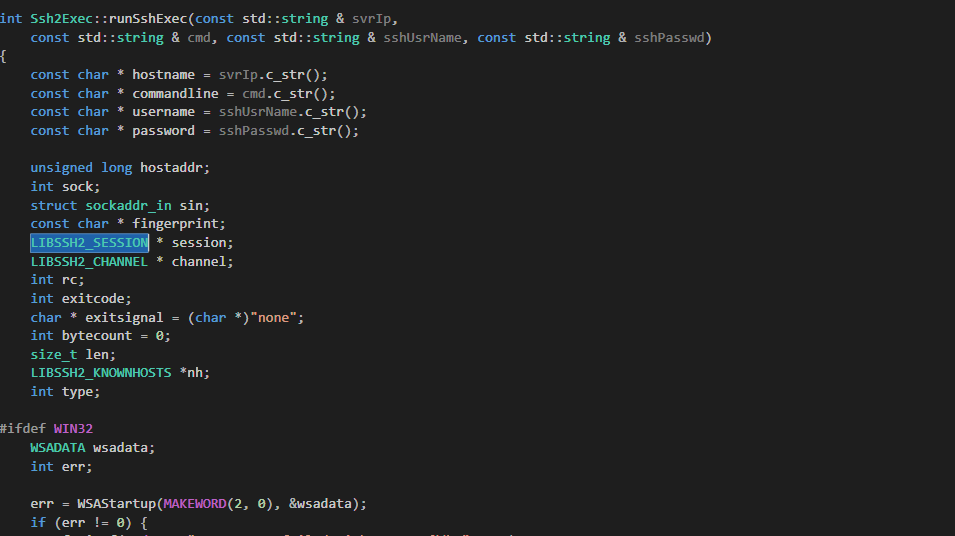
#### 优势分析

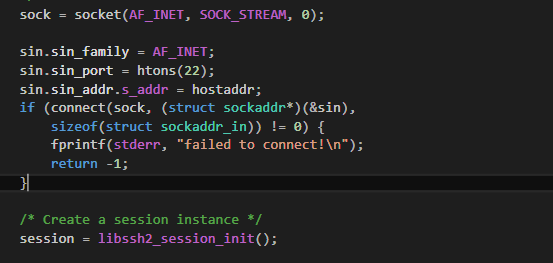
###### Xml解析效率

Xml文件在该工具软件中使用非常频繁，所以采用自己编写解析代码的方式来完成，否则调用第三方库会严重影响系统结构和执行效率。软件中通过boost标准库来编写xml的解析，执行效率占用内存合理并且快速。

###### 基于libssh2库的SSH实现

由于windows上不存在SSH的连接访问，所以从windows到Linux的SSH连接需要自己来完成代码编写，方案选择了使用libSsh2库，依赖该库类进行SSH的socket函数编写。



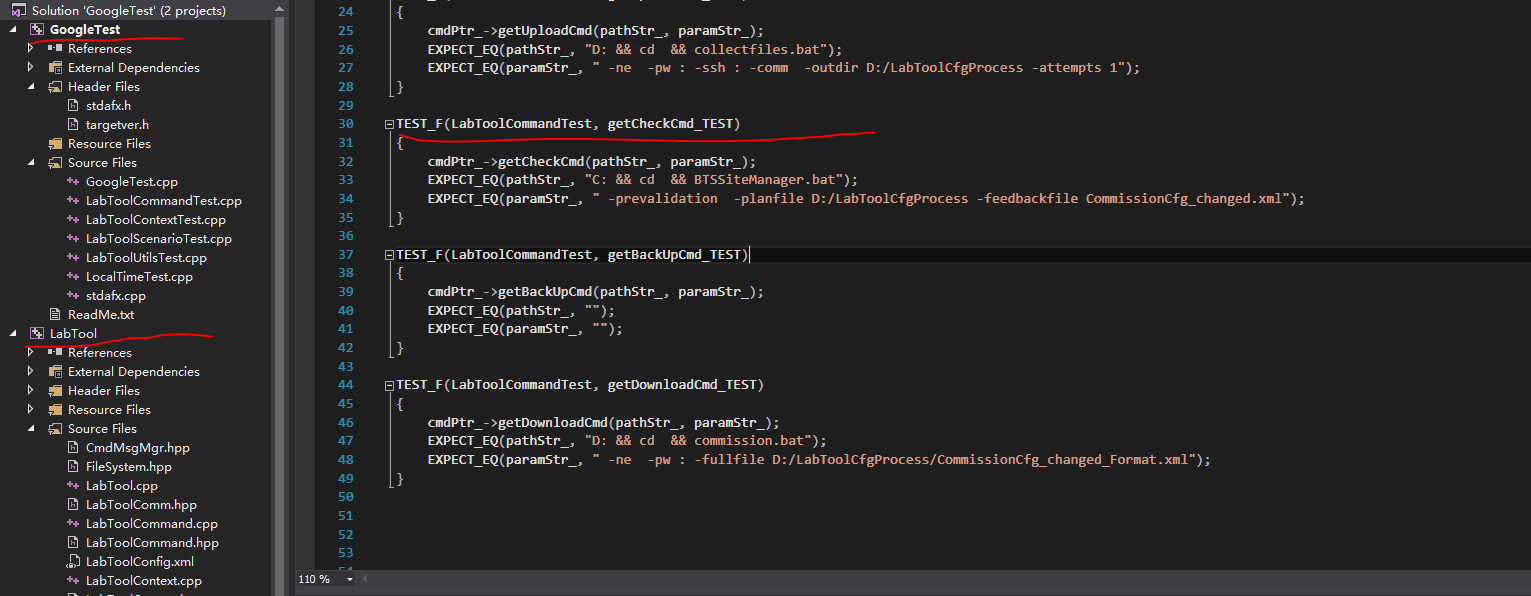


###### 智能指针

代码中很少使用对象实例，而是使用智能指针，使用指针的好处是占用执行空间少，并且易于维护，智能指针有效的实现了系统内存管理，合理释放内存，不需要担心普通指针被重复释放或者是悬挂指针等致命问题。

###### GoogleTest编写测试代码

除了功能代码之外，还编写了GoogleTest测试代码对关键逻辑步骤进行测试，确保每个关键步骤都有正确的执行结果。



###### 完善的系统结构

系统设计中采用了合理的数据结构和设计模式，并且在软件中添加了文件管理系统、日志系统和异常保护；

1. 文件管理系统

文件管理系统的封装编写可以使得代码中方便的调用各个文件的不同操作，避免重复代码并且避免了代码出错的概率；

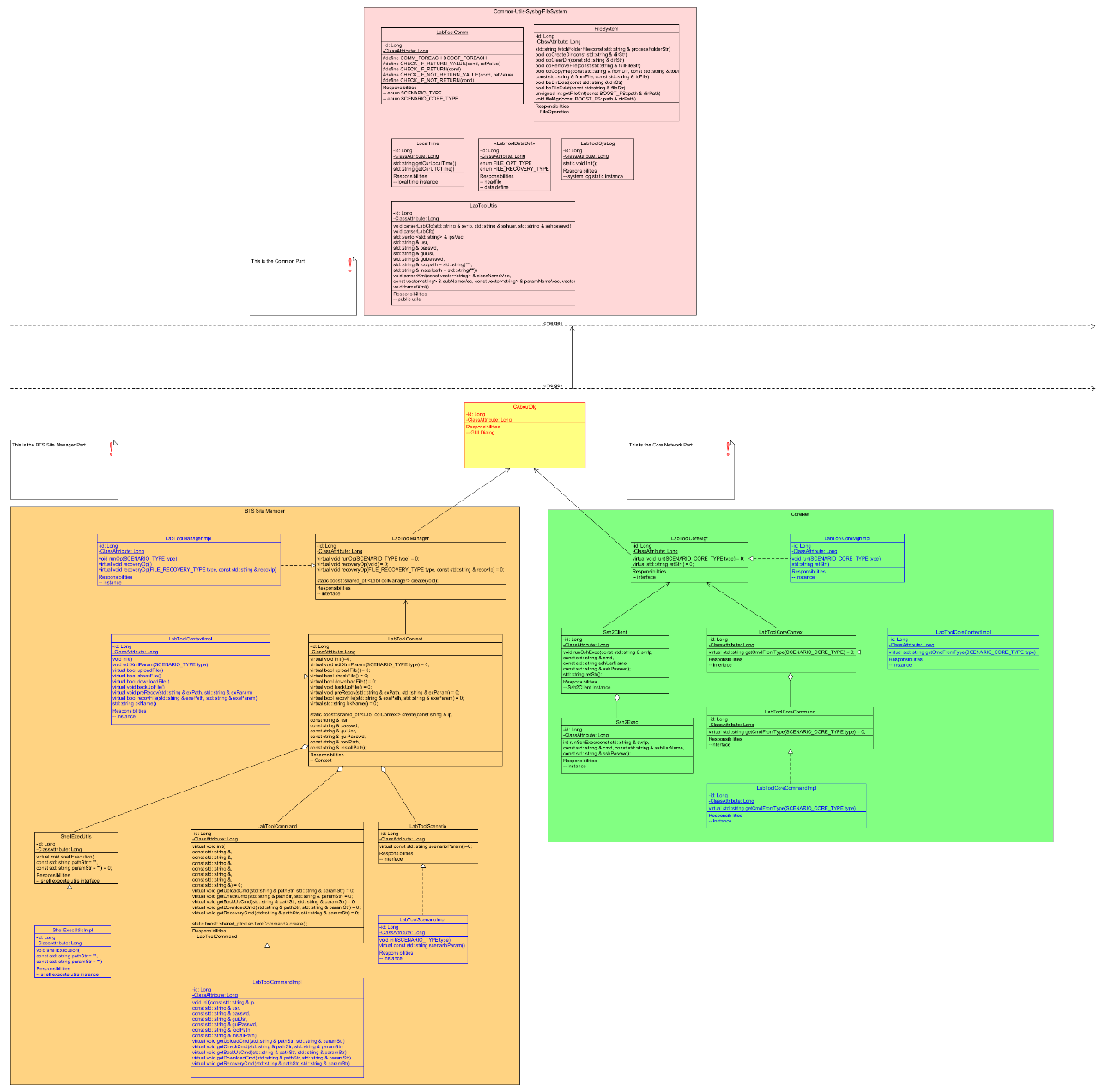
1. 日志系统

日志系统主要记录各个步骤操作的执行结果和异常错误结果，可以有效分析错误所在。

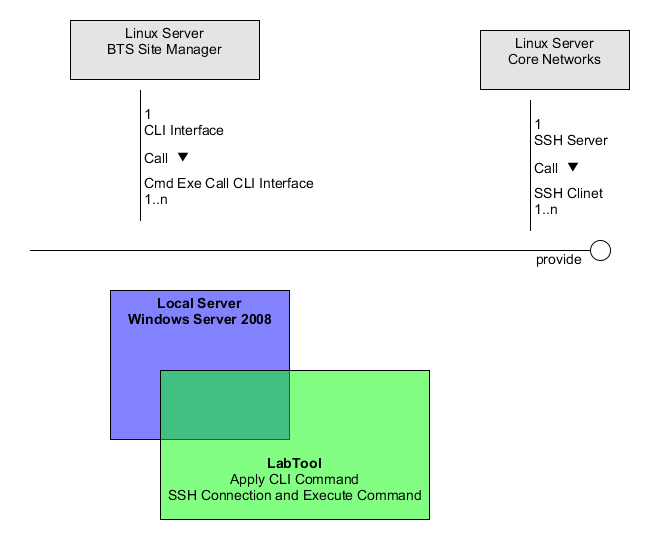
1. 异常保护

软件的每个关键步骤都有异常保护，只要是软件执行抛出异常都会被有效捕获，并且记录信息来进行保存，提高稳定性。

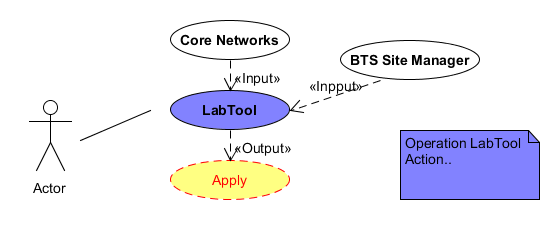
#### UML设计原理图



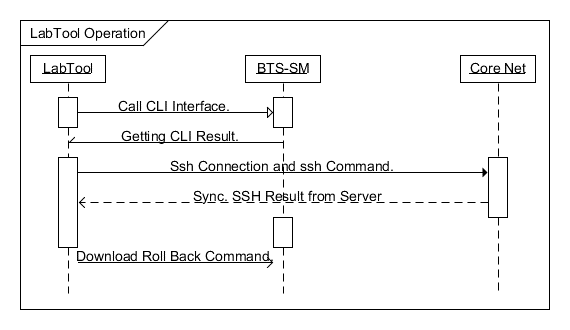
Class原理图



部署层次原理图



对象操作原理图



时序原理图

#### 参考资料

[www.boost.org](http://www.boost.org)；

Beyond the C++ Standard Library: An Introduction to Boost；By Björn Karlsson

#### 附录