2024国赛决赛安全可信

Build赛题操作手册

2024年7月

目录

[1. 赛题概述 3](#_Toc32027)

[1.1. 赛题目标： 3](#_Toc240)

[1.2. 解题要求： 3](#_Toc27810)

[2. 赛题背景介绍 3](#_Toc2405)

[2.1. 任务背景介绍： 3](#_Toc837)

[2.2. 模拟方式： 4](#_Toc5663)

[2.3. 场景流程设计： 5](#_Toc18772)

[3. 赛题任务 6](#_Toc25818)

[3.1. 新手任务（25分）： 6](#_Toc18823)

[3.1.1. 背景应用了解（10分） 6](#_Toc12726)

[3.1.2. 身份认证安全测试（15分） 10](#_Toc17080)

[3.2. 攻击模拟任务（25分）： 13](#_Toc20083)

[3.2.1. 越权访问（6分）： 13](#_Toc23730)

[3.2.2. 恶意逻辑（8分）： 14](#_Toc5266)

[3.2.3. modbus数据监听与篡改（11分）： 14](#_Toc8267)

[3.3. 访问控制模拟任务（25分）： 15](#_Toc4485)

[3.3.1. 角色验证（7分）： 15](#_Toc13655)

[3.3.2. 角色权限限制（8分）： 15](#_Toc1362)

[3.3.3. 自动逻辑屏蔽：（10分） 15](#_Toc15677)

[3.4. 密码保护模拟任务（25分）： 16](#_Toc20309)

[3.4.1. 工程师签名上传：（12分） 16](#_Toc5943)

[3.4.2. modbus协议数据加密：（13分） 17](#_Toc514)

[附录 19](#_Toc896)

[附录1：PLC设备设定 19](#_Toc24027)

[附录2：数据结构设计列表 20](#_Toc1275)

[附录3：枚举值定义 21](#_Toc13559)

[附录4 数据类型定义表 22](#_Toc7455)

[4.1 工程师（PLC\_ENGINEER）相关类型定义 22](#_Toc965)

[4.2 操作员（PLC\_OPERATOR）相关类型定义 24](#_Toc7205)

[4.3 系统监视员（PLC\_MONITOR）相关类型定义 25](#_Toc12537)

[4.4 设备（PLC\_DEVICE）相关类型定义 26](#_Toc20433)

[4.5 MODBUS\_TCP 类型定义 27](#_Toc13759)

[4.6 MODBUS\_STATE 类型定义 27](#_Toc27975)

[附录5：操作流程详细介绍 29](#_Toc14504)

[5.1 用户登录流程 29](#_Toc14030)

[5.2 工程师程序上传流程 31](#_Toc14578)

[5.3 操作员操作流程 33](#_Toc9330)

[5.4 系统自动监视流程 37](#_Toc5400)

[5.5 管理中心远程控制流程 38](#_Toc20193)

# 赛题概述

## **赛题目标：**

让学生了解可信3.0中安全机制对应用透明的理念，体会工控系统中通过符合等保思路的访问控制、国密算法应用等防御技术保护业务流程的方式，熟悉安全可信赛题环境的操作，学习安全可信体系化设计的思路。

## **解题要求：**

时间：5天

目标：从体系化的角度体会安全可信

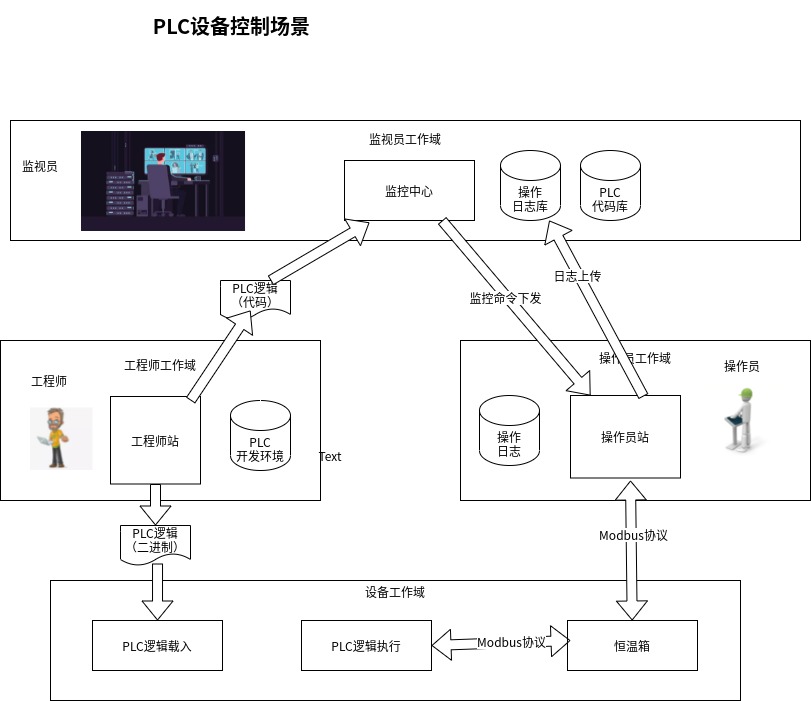
开赛后，在平台题目处获取到题目的docker镜像文件，并在本地自行部署，以root身份进入docker镜像中，根据题目要求，在docker镜像中完成任务中相关内容，在本地自测通过后，按题目文档要求，在平台上提交相应的zip压缩包，等待系统完成评分。

# **赛题**背景**介绍**

## **任务背景介绍：**

任务依托于对一个工控环境恒温控制系统的模拟，该系统由控制中心、工程师站、操作员站、PLC设备组成。其中工程师站由工程师进行操作，并向PLC设备上传恒温控制逻辑代码；操作员站由操作员进行操作，其决定恒温设备的启动、关闭、参数设置，并在突发状况下接管控制逻辑。控制中心则由监控人员操作，监控远程温度监测系统的运行状况，并在异常状况出现时输出报警信息，启动应急处置行为。

场景的逻辑框图如下图所示：



## 模拟方式：

系统使用通用可信基础软件cube-1.3模拟这一工控环境，模拟环境位于2024buildup目录下，工控环境中各节点用其中instance目录下的实例模拟，实例目录与对象模拟关系如下表所示：

表2 节点与拓扑结构模拟表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实例位置(instance目录下） | 模拟对象 | 模拟功能 | 连接状况 | 备注 |
| monitor/user | 安全监控员 | 模拟安全监控员的监控数据读取以及监控命令下发 | 客户端连入安全监控终端(端口号10011） |  |
| monitor/term | 安全监控终端 | 模拟安全监控员的监控数据读取以及监控命令下发 | 服务器端，端口号10011  客户端连入安全监控中心(端口号10001） |  |
| monitor/center | 安全监控中心 | 模拟监控数据定时获取以及监控命令下达过程 | 服务器端,端口号  10001 |  |
| engineer/user | 工程师 | 模拟工程师发送上传程序源码和上传二进制模块的过程 | 客户端连入工程师站  端口号10002 | 有PLC逻辑模块开发环境 |
| engineer/station | 工程师站 | 模拟工程师站上传二进制代码与上传程序源码的过程 | 服务器端 端口号10002  客户端连入安全监控中心，端口号  10001  客户端连入PLC代码载入器，端口号10101 |  |
| operator/user | 操作员 | 模拟操作员对设备的监控以及特殊情况下的控制命令 | 客户端连入操作员站  端口号10021 |  |
| operator/station | 操作员站 | 模拟操作员站的设备监测信息获取以及控制命令发放的过程 | 服务器端（端口号10021）  客户端连入安全监控中心（端口号10001）  客户端连入PLC设备（modbus协议，端口号1504） |  |
| plc/loader | PLC代码载入器 | 模拟plc代码的获取与载入，以及plc设备的启动过程 | 服务器端，端口号10101 | 可向PLC设备实例复制模块并启动PLC设备 |
| plc/device | PLC设备 | 模拟plc设备的命令控制 | 服务器端（modbus协议，端口号1504） |  |

此外，为模拟攻击行为，我们还在instance下设置了login\_hacker和modbus\_hacker两个实例分别作为登录认证过程和modbus命令传输过程的攻击模拟环境。

## 场景流程设计：

本场景总共模拟了6种操作流程，如下所列。

1. 身份认证流程：工程师、操作员和监控员在终端（工程师站，操作员站和监控终端）输入用户名和密码，在管理中心完成身份认证并返回。
2. 工程师代码上传流程： 工程师在工程师站完成代码编写后，将代码原文传送给安全监控中心。
3. 工程师逻辑上传流程：工程师编译代码后，将所编译的代码逻辑提交到plc代码载入器，由其将代码逻辑上传到PLC设备
4. 操作员操作流程：操作员发送操作命令，包括开机、观测与操作，命令发送给PLC设备，并据此对恒温箱进行操作。操作员操作命令及返回的数据将被记录在操作日志中。
5. 系统自动监控流程：PLC逻辑接收恒温箱设备温度信息，并根据恒温箱温度情况，调节电热丝档位，使恒温箱温度始终保持在目标温度附近。
6. 监控员操作流程：监控员获取设备当前温度信息，判断信息是否存在异常，在特殊情况下，可直接代替操作员进行操作。

流程的具体描述请参见附录，操作方法则参见赛题任务部分。

# 赛题任务

## **新手任务（25分）：**

### 背景应用了解（10分）

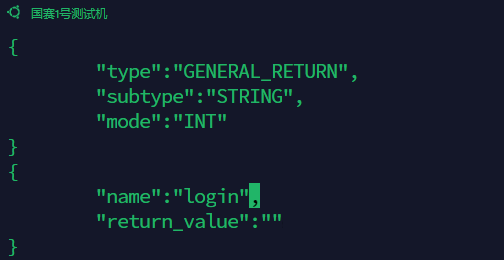
在提供安全保障方案时，一定注意要维护应用的可用性。主动免疫安全要求防护机制对应用透明。为此，我们首先要了解应用的业务流程。

请分别针对如下流程，正确配置相关文件，使相应的业务流程可以正常运行。填充后可使用

1. 模拟工程师登录流程

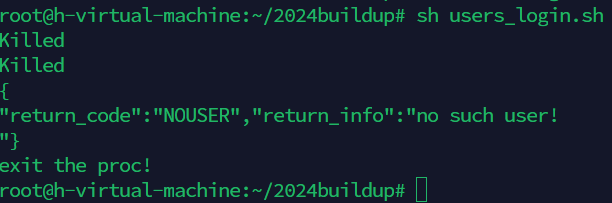
在工程师用户（instance/engineer/user)目录下，编写engineer\_login.msg， 填充工程师用户名与口令, 而后试运行users\_login.sh，确认用户登录成功。

engineer\_login.msg 内容如下所示：

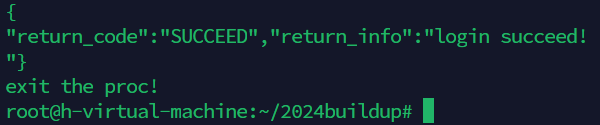


它用json格式定义了cube架构中一个格式为（GENERAL\_RETURN,STRING)的数据项，这个数据项是一个通用数据项，由两个字符串name和return\_value组成，name用来描述数据项的类型或用途，return\_value则填充该数据项具体内容。这里name取值为login，表示该数据项用于登录，return\_value的引号内则按照“用户名：口令”的方式填充内容。

直接执行users\_login命令测试（运行 sh users\_login.sh)，我们将看到如下结果：



如果内容填充正确，运行后则看到如下结果：



用户名和口令信息请在管理中心工作目录(instance/monitor/center)里寻找。

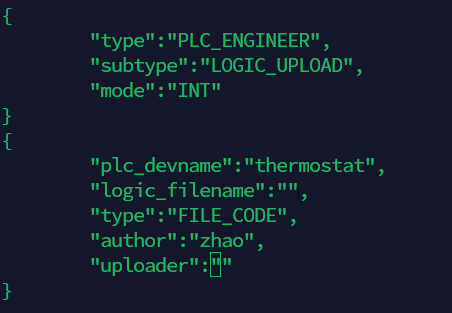
1. 模拟逻辑代码上传

本流程和后续的流程我们都使用task\_1\_1.sh脚本进行集中测试。为防止后续内容干扰前面测试结果，后续操作均已被注释掉，在完成前面内容测试后，我们可以打开注释项来测试后续内容。

在2024buildup/src/logic/thermostat\_logic目录下写了一个模拟PLC行为的代码，用其源码thermostat\_logic.c来模拟PLC源码，其编译后的动态库thermostat\_logic.so来模拟PLC逻辑。

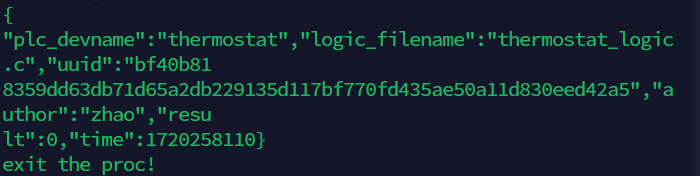
请在工程师用户目录编写code\_upload.msg，填写正确的上传代码信息，试运行并完成代码上传功能。

code\_upload.msg 内容如下：



它描述的是一个格式为（PLC\_ENGINEER,LOGIC\_UPLOAD)格式的数据项，数据项内容介绍参见附录中的数据格式描述部分。

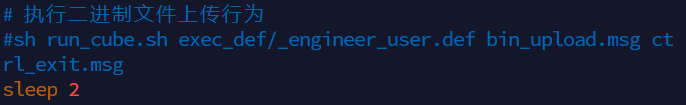
正确填充其中内容后，运行task\_1\_1.sh，可以看到在用户登录信息后，系统将输出如下信息：



这是格式为（PLC\_ENGINEER,PLC\_RETURN)的回复信息，result项为0，表示代码提交过程正确。

1. 模拟二进制逻辑上传

编辑task\_1\_1.sh，找到下面内容



删除sh前面的#号，打开注释行，并在工程师用户目录下编写bin\_upload.msg，填写正确的plc逻辑信息，再运行task\_1\_1.sh,以实现模拟的二进制逻辑上传。

完成二进制逻辑上传后，可看到反馈的（PLC\_ENGINEER,PLC\_RETURN）格式消息，在2024buildup的plugin目录下，可看到libthermostat\_logic.so文件。

1. 模拟PLC逻辑启动。

编辑task\_1\_1.sh，打开“启动plc设备模拟”提示后面的注释项，运行task\_1\_1.sh，可看到下面内容：



观察plc设备工作目录instance/plc/device/下的message\_log目录，可发现其增加了logic\_send.log文件，这里message\_log目录记录的是各用例模拟运行的过程，logic\_send则是在设备工作目录下配置的系统自动监控流程的模拟业务流。在设备工作目录下的router\_policy.cfg里如下定义：



这里用消息路由来将多个模块的功能串起来模拟业务流程。开头部分描述了消息路由的名称、类型和起始模块，中间部分表示当消息在起始模块位置时的格式和内容信息。当起始模块生成消息(模块中用message\_create函数生成，且最后激活消息参数设为NULL时）与其格式要求一致时（在这里就是消息类型为MODBUS\_CMD，用来模拟modbus协议命令)，会被识别为该路由的起始消息。最后部分则说明消息先后通过device和logic模块后结束传递。它其实模拟了plc逻辑（用logic模块模拟）发出命令给device,由device处理后返回logic模块的过程，可参考附录5.4中对logic\_send路由的描述。当系统模拟运行时，观察设备工作目录instance/plc/device中日志目录message\_log中的logic\_send.msg文件将显示该路由中的信息，由于该过程循环进行，因此可看到大量信息。

1. 模拟操作员登录行为

在task\_1\_1.sh脚本的“启动hacker”提示、“启动操作员站”提示和“执行操作员登录行为”提示后面打开被注释的命令，并编辑操作员用户工作目录下的operator\_login.msg，添加操作员用户的名称和口令信息，而后进行试登录。如登录成功，则表明通过该过程。

1. 模拟操作员的一组plc操作

最后我们模拟操作员进行三项plc操作行为。

打开task\_1\_1.sh最后几条注释掉的命令，然后，修改操作员用户工作目录下的plc\_start.msg , plc\_read\_t.msg和plc\_set\_t.msg，分别实现打开plc温度调节开关，观察plc当前环境温度以及将plc恒温温度设置为39度（3900）的目标。

plc数据格式和寄存器的定义参见附件，也可参考操作员站目录下的register.list文件。

**当上述功能均完成后，将过程中修改的各种\*.msg文件汇总，并打包为zip压缩包进行提交，系统裁决该部分的任务完成情况通过后即可获得该任务的分数。需汇总的msg文件可参见“任务接口文档列表.docx”（其他任务需要打包的文件不一定是msg文件）。**

### 身份认证安全测试（15分）

本过程让选手了解攻击机制模拟和安全机制插入的基本方法。

#### 中间人攻击（7分）

系统在工程师站engineer\_station和管理中心center之间插入了一个login\_hacker实例，用该实例来模拟攻击行为。当前实例中使用了一个echo\_plugin模块，该模块只是简单地转发收到的消息，不会产生攻击行为。

某攻击者开发了一个重放攻击模块login\_hacker,该模块可以记录登录用户的名称与所使用的认证口令，如果攻击者使用登录过的用户名进行登录，并且在口令处使用攻击者预设的口令，则模块将自动使用以前记录的口令进行登录。

请用login\_hacker模块替换echo\_plugin模块，并且正确填写工程师用户目录下的engineer\_login\_nopass.msg，使攻击者可以在没有工程师用户口令的情况下在工程师站仿冒工程师进行登录。

相关知识点：

1. cube架构的实例将准备使用的模块列在plugin\_config.cfg文件中，每个选用的模块都有两个名字，name表示实例中该模块的名称，在定义路由时使用这个名字，这个名字一般与模块在实例中的使用方式相关。libname则是该模块开发时使用的名字，这个名字一般表现了开发者对其的定位。这里echo\_login和login\_hacker都是libname。
2. login\_hacker模块的源码在2024buildup/src/security下面的某个同名目录中，以c语言编写，阅读其源码，可以找到所用的预设口令。此外，也可以顺便了解下cube架构中模块的逻辑。cube架构的模块其结构比较清晰，一般用一个模块名+init的函数（称为模块的init函数）进行模块载入时的初始化操作，用一个模块名+start的函数（称为模块的start函数）作为模块的主体，持续进行循环，每次循环检查模块接收到的消息并进行处理。start函数根据消息的格式分流消息，让每种消息调用自己的专用操作函数。如果模块的几个处理流程共用同一种消息格式，则一般在消息处理函数中获取消息后，再根据消息内容进行第二次分流，一般而言，每个流程都有自己专门的处理函数。

#### 挑战-应答机制（8分）

为了防范中间人攻击，我们准备用挑战-应答机制来替换掉当前使用的口令字符串哈希的认证机制。当前使用的用户端登录处理模块login\_user和服务端登录处理模块login\_server本身就具备了支持挑战-应答机制的能力。

请正确配置参数和路由，实现挑战-应答机制，以防范重放攻击行为。

知识点：

1. 在plugin\_config.cfg中定义模块时，可以使用init\_para参数来为模块提供一些配置信息。配置信息的定义一般在模块的定义文件cfg中。如login\_user.cfg中给出了init\_para\_desc的定义，定义中用mode值来表示模块采用哪一种登录机制。plugin模块定义时，可以用

"init\_para":{

"mode":1

}

来表示模块选择了模式1。

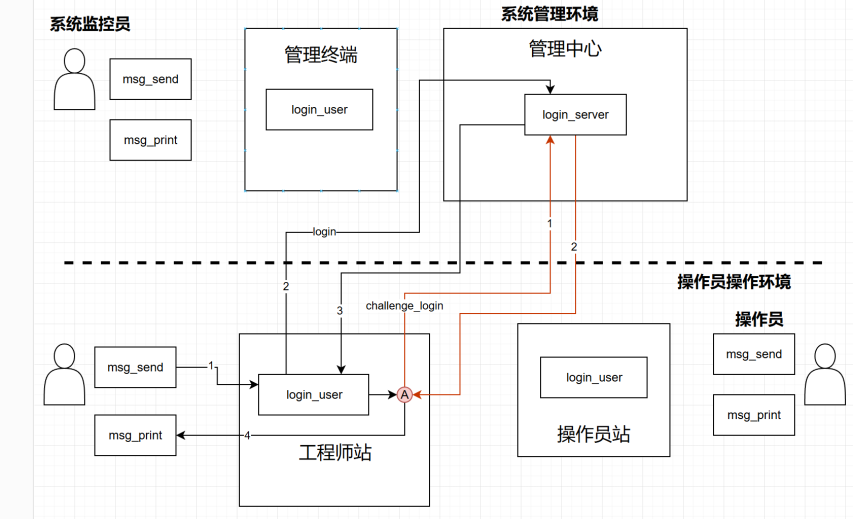
请通过阅读源码确定哪种模式是挑战应答模式，并在plugin\_config.cfg中正确配置。

**注意配置时要把配置内容写到plugin\_config.cfg.task\_1\_3中，本任务的测试脚本task\_1\_3.sh会自动将其复制成plugin\_config.cfg，后面的配置文件修改基本都是这种模式，这是为了区分选手们不同任务的提交作业，它直接影响选手的成绩，请务必注意！**

1. 挑战-应答机制中，用户和服务器之间会进行两次交互，而旧的消息路由只提供了一次交互流程，需要重新配置认证流程。

旧的身份认证流程是用login路由来认证的，它由engineer/user,engineer/station和monitor\_center中定义的login路由相互连接而形成，类型为QUERY，表示该路由为登录后返回的过程，可参考附录5.1中的说明。而在支持挑战-应答机制时，login\_user在收到返回消息后，会生成状态为CHALLENGE格式的登录消息，这个消息将再次发送到server端并完成登录行为。

我们把工程师站login\_user收到用户登录消息后，发送消息给管理中心login\_server，并获得挑战信息的过程看做是插入到原有登录过程中的新过程，就可以使用cube架构中的切面路由机制。这个机制定义的路由一般放在aspect\_policy.cfg中，选项可定为ASPECT,是指在原路由中插入了一段处理过程，处理过程完成后回到原本路由。 用路由图来描述则如下所示：



切面路由将有自己的新名称，这里我们将新的切面路由命名为challenge\_login，在管理中心的路由配置文件中定义，可以把它当作是由切入点开始的QUERY格式路由（询问模式，发出消息抵达目标后原路返回)，最终回到切入点，再重新执行原流程。

下面是本任务切面路由定义的示例，它在路由文件（一般为aspect\_policy.cfg)所在的实例位置，对指定路由进行拦截：



这里policy\_head表示其拦截位置，name表示被拦截路由，type有两种取值，复制（DUP)表示从原路由复制信息，不改变原路由传输方式，拦截（ASPECT)则是让原路由转去新路由进行处理，完成处理后无异常情况则返回原路由途径。

ljump表示是在本地路由的第几跳（生成/收到消息为第1跳，经过一个模块增加1跳，sender表示路由的发送者，这两个共同定义了路由的起始位置。rjump是一个准备被废弃或替代的参数，在这里的唯一作用是用负值表示拦截的是返回信息。

newname则表示复制或拦截路由所在的新路由名称，这里复制路由上的消息将沿着新路由传播，而拦截路由上的消息则在完成新路由并返回后，重新进入原路由，沿着拦截点继续传播。

“MATCH\_RULES”中的内容表示切面触发时消息要满足的条件，ROUTE\_RULES则指消息在新路由中传播的路径。这里的路由可以指向其它实例，在其它实例中与其对接的路由应当是与新路由名称一致的DELIVER格式路由（与DUP格式切面路由对接）或QUERY格式路由（与ASPECT格式路由对接）。

切面路由是不改变原有流程情况下添加安全机制的好方法，我们后续的安全机制基本都用这一方式添加。

**本题大家只要将上述内容填写到instance/engineer/station的aspect\_policy.cfg.task\_1\_3文件，task\_1\_3.sh脚本会自动将其复制为aspect\_policy.cfg,并使该切面路由生效，将挑战-应答过程添加进来。**

## **攻击模拟任务（25分）：**

### 越权访问（6分）：

本过程由用户测试系统中可以进行越权访问的位置。

本系统中，工程师、操作员和系统监视员除可执行规定允许的操作外，还有如下限制：

1. 工程师可以登录操作员站和管理终端，但只能观察数据，不能执行调节温度等控制行为。
2. 操作员可以登录管理终端，但只能观察数据，不能在监视终端执行调节温度等控制行为。
3. 监视员可以登录操作员站，但只能观察数据，不能在监视终端执行调节温度等控制行为。
4. 操作员和监视员均可登录工程师站，但不能执行代码上传和逻辑上传命令。

请在工程师站、操作员站和监视终端进行测试，找到可以违规操作的点。

控制行为用调节温度到39度代表，参考操作员用户目录下的plc\_set\_t.msg。

上传命令则用代码上传代表，参考工程师用户目录下的code\_upload.msg。

在工程师站、操作员站和管理终端各找到两个可违规操作点，按照下表创建违规登录与操作的组合消息。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 位置 | 编号 | 登录（使用非合规用户） | 操作（非合规用户执行违规操作） | 备注 |
| 工程师用户 | 违规点1 | attack\_login1.msg | attack\_cmd1.msg |  |
| 违规点2 | attack\_login2.msg | attack\_cmd2.msg |  |
| 操作员用户 | 违规点3 | attack\_login3.msg | attack\_cmd3.msg |  |
| 违规点4 | attack\_login4.msg | attack\_cmd4.msg |  |
| 监视员用户 | 违规点5 | attack\_login5.msg | attack\_cmd5.msg |  |
| 违规点6 | attack\_login6.msg | attack\_cmd6.msg |  |

组合填写完成后，可以使用测试脚本task\_2\_1.sh测试这些组合消息是否可以正确操作。

### 恶意逻辑（8分）：

本任务显示攻击者通过用恶意逻辑替代正常逻辑对系统进行攻击的行为。

在logic/hack\_logic中，设置了恶意的温度逻辑，在10秒后将始终保持设置温度档位为最高档。这里hack\_logic没有列在security的makefile中，需要单独编译，编译后，我们可以看到目录下会出现一个libthermostat\_logic.so文件，这个文件是用来冒充合法逻辑的恶意逻辑文件。

请为file\_replace模块填充文件替换代码，并通过修改模块配置文件和切面路由文件，拦截工程师站到plc的二进制逻辑文件传输过程(bin\_upload路由），使得file\_replace模块在合适位置发挥作用，将所传输的libthermostat\_logic.so文件内容替换为hack\_logic目录下的同名文件，从而导致系统运行一段时间后温度失控。

可以用task\_2\_2\_1.sh脚本测试二进制逻辑上传过程，用task\_2\_2\_2.sh脚本测试逻辑运行情况，该脚本启动后会5秒读一次温度状况，重复四次。

**需配置的路由文件为instance/engineer/station目录下的plugin\_config.cfg.task\_2\_2和aspect\_policy.cfg.task\_2\_2,脚本会自动用这两个文件替换当前的配置文件。**

### modbus数据监听与篡改（11分）：

本任务演示对modbus协议数据的攻击行为，选手需要对modbus协议有基础的了解。

在modbus\_hacker实例中，modbus\_monitor模块接入到modbus主设备和从设备的通信线路中，准备窃听当前温度。窃听温度后不必输出，当温度超过43度时，模块应篡改回传的温度数值，使操作员和监视员看到的温度始终低于43度。

可以用operator\_start.sh脚本观察modbus协议的输入输出，modbus\_hacker实例的modbus\_monitor模块会打印协议的输入输出信息。注意后续实验时，为测试访控和加密效果，modbus\_monitor模块建议改回原版模块。

## **访问控制模拟任务（25分）：**

### 角色验证（7分）：

本任务显示通过严格的身份认证机制防止越权访问的方法。

在login\_server身份认证机制前方放置一个角色检查模块role\_verify，检查用户角色身份与用户节点名称（登录信息中的proc\_name)是否一致，如一致则允许登录，否则禁止登录并返回登录结果不一致信息。

修改后，分别在管理终端、工程师站和操作员站尝试用非合规用户登录，非合规用户使用任务2-1中设置的用户，采用task\_3\_1.sh脚本进行测试，如恶意行为均被拦截则说明功能实现成功。

### 角色权限限制（8分）：

本任务取消角色验证功能 ，不限制用户的登录位置，但是不同角色用户在不同位置受到不同的功能限制。这里我们只在管理中心进行角色权限限制行为。

编写role\_access模块，在管理中心拦截用户命令，禁止执行不符合用户身份的命令，并且将该模块接入到管理中心的命令控制过程和代码上传业务中。

role\_access模块的框架已在src/security目录下，它需要正确处理管理中心收到的上传命令和plc调节命令，禁止权限不正确的两种命令执行。

### 自动逻辑屏蔽：（10分）

编写operator\_shield模块，在操作员发现异常并开始主动进行温度控制时，屏蔽自动逻辑的modbus命令，并将其正确接入到device的消息路由中（5分）

编写monitor\_shield模块，在系统监视员发现异常并开始主动进行温度控制时，屏蔽操作员的modbus命令，并将该模块正确接入到操作员站operator\_station的消息路由中（5分）

可以通过用message\_get\_sender函数获取操作命令消息发送者的方式确定设备端的modbus命令来源，函数具体定义参考/root/centoscloud/cube-1.3/include/cube.h。

在操作员站operator\_station用消息发送者来辨别就不是很方便了，这里可以直接用message\_get\_head函数获取格式为MSG\_HEAD的消息头，然后从消息头中获取消息路由字段来辨识plc调节命令是来自操作员还是监视员。

monitor\_shield模块需要用切面路由方式接入到operator\_cmd路由或者monitor\_ctrl路由过程中，在两种路由接入时，切面路由的名称(aspect\_policy.cfg定义中的newname)应当不同，以便于monitor\_shileld区分两种命令消息。在monitor\_shield模块源码中，我们分别将其设定为monitor\_aspect和operator\_aspect。

## **密码保护模拟任务（25分）：**

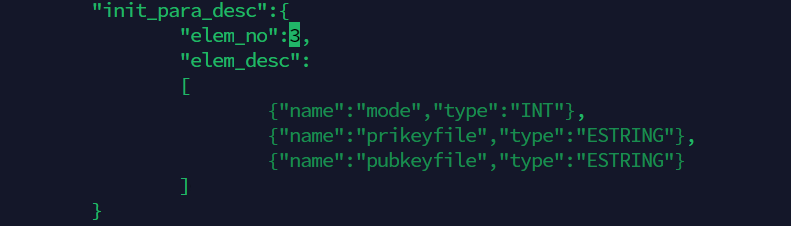
在完成密码保护模拟任务之前，需要做一项准备工作。

分别进入/root/centoscloud/gm\_sm2\_master和/root/centoscloud/zuc\_test目录下进行编译，完成编译后，将gm\_sm2\_master目录下的libsm2So.so和zuc\_test目录下的libzuc.so复制到2024buildup/locallib目录下。这两个库分别提供sm2算法函数和祖冲之算法函数，复制到locallib目录之后，我们的模块编译后就可以找到模块并调用这些函数了。

### 工程师签名上传：（12分）

工程师上传plc逻辑时，提供sm2签名，防止攻击者篡改plc逻辑。

请编写sm2sign模块来实现该功能。该模块设置了3个参数，如下所示：



这里mode=0表示其为签名者，mode=1表示其为验证者。

prikeyfile为私钥文件名，pubkeyfile为公钥文件名。这里固定设置为Priv.key和Pub.key。Sm2密钥相关函数使用开源库gm\_sm2\_master，在centoscloud/gm\_sm2\_master目录下，其头文件sm2.h在include目录下，其库文件则放在2024buildup/locallib目录下。

签名者运行时，如发现工作目录无两个公钥文件，则生成公私钥对，将生成的密钥值写入文件中。

模块如收到二进制PLC逻辑上传消息，则其为签名者时，找到对应二进制文件，计算摘要值，确认其与(PLC\_ENGINEER,LOGIC\_UPLOAD)消息中uuid一致，并对消息内容进行签名，将签名的二进制数据生成（GENERAL\_RETURN,BINDATA)格式数据，并作为(PLC\_ENGINEER,LOGIC\_UPLOAD)消息的扩展项发送，当其为验证者时，则从扩展项中获取(GENERAL\_RETURN,BINDATA)消息，验证收到的(PLC\_ENGINEER,LOGIC\_UPLOAD)消息的可靠性，并进而确认收到文件的可信性。

### modbus协议数据加密：（13分）

使用祖冲之密码算法，加密modbus协议中的返回数据，防止攻击者窃听温度信息。对发送的命令信息可不做任何处理。

operator/station端加密位置建议在modbus\_channel与modbus\_master端之间，plc/device端加密位置建议在modbus\_channel和modbus\_slave端之间。注意operator/station目录下有slave.list文件，是格式为（MODBUS\_STATE,SLAVE)的数据记录，在启动时载入到操作员站的内存数据库中，而plc/device下则有格式为（GENERAL\_RETURN,UUID)的记录项key.list，用以储存密钥。可看到slave.list中的slave\_key与key.list中的return\_value是一样的，这是主从设备双方的共享秘密，基于这一秘密，可以使用序列密码算法进行加密。

选择祖冲之序列密码算法是因为modbus协议有严格的格式要求，为保持兼容性，不能修改数据格式，只能对短且不规则的数据内容部分加密，但是即使如此，仍需考虑一些特殊情况，如是否可以让窃听者无法分辩长时间温度不变的情况，以及如果modbus数据传输过程中出现丢包现象，是否还能继续维持有效加密？

请在modbus\_zuc\_crypt模块中实现对应位置的加解密，并考虑对前述隐藏问题的处理要求。注意共享秘密的前16个字节作为祖冲之算法的密钥，后16个字节则作为祖冲之算法的初始向量。

实现数据加解密 5分

攻击者无法分辨重复的温度 4分

丢包情况下仍然可以正常加解密 4分

另外需要注意，在测试祖冲之加密时，请将modbus\_monitor模块还原为未被篡改状态，否则会影响测试。

# 附录

## 附录1：PLC设备设定

该场景的设备均由cube实例进行模拟。模拟的实例有一台PLC设备、一个工程师站、一个操作员站以及一个安全监控中心。

PLC设备模拟的是一个只具备加热功能的恒温箱，其基本功能为：打开/关闭设备，打开/关闭加热开关，调节加热档位（0-9档），设置恒温数值，以及观察恒温箱当前温度。对恒温箱的控制使用Modbus协议。

Modbus协议中采用寄存器的概念来描述设备中存放数据的位置。这里开关、观测数据、设置数据都可以用寄存器描述。Modbus协议定义了四类寄存器：线圈（Coil),用来描述PLC可控制的开关数据量，设备开关、加热器开关都是这一类型寄存器；离散输入（Discrete input)，描述外部设置的开关，PLC只能读取该数据量；保持寄存器（Holding registers)，表示PLC可设置数值的寄存器，在本例中，恒温温度设置和加热档位设置均为保持寄存器；输入寄存器（input registers),表示PLC只能读取的寄存器，恒温箱当前温度就存放在环境寄存器中。

本题目中，PLC设备共有五个寄存器R1-R5,其内容如下所列：

表1 PLC设备寄存器设计

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 寄存器名称 | 寄存器类型 | 寄存器地址 | 寄存器内容 | 备注 |
| device\_S | coil | 1 | 恒温箱开关 | 0为关闭，1为启动 |
| set\_T | Holding Registers | 40003 | 恒温数值要求 | 短整型 |
| curr\_T | Input Registers | 30004 | 恒温箱当前温度 | 短整型 |
| heating\_S | Coil | 2 | 电热丝开关 | 0为关，1为制热 |
| heating\_G | Holding Registers | 40005 | 电流档位 | 0-9档 |

系统运行时设定恒温数值要求（为set\_T寄存器设置值），并上传一个组态逻辑模块(由cube模块模拟），程序将根据恒温数值以及当前温度确定恒温器状态及档位，以保持恒温箱温度符合要求。

工程师站由工程师编写组态逻辑模块，并将模块传送给PLC设备。

操作员站由操作员执行读取PLC设备当前温度和状态，以及进行温度调节的操作。操作员站向PLC设备发送的命令采用modbus协议，其中操作员站为主设备，PLC为从设备，以模拟真实的操作员-PLC操作过程。

监控员通过监控终端在安全监控中心监控PLC操作行为，在PLC行为异常，监控员可以直接执行远程控制功能，在监控员站进行PLC设置。

## 附录2：数据结构设计列表

下表为针对题目的各种操作所定义的具体格式,这些格式为不同节点间的接口。这些数据格式的详细情况在附录4描述

|  |  |  |  |
| --- | --- | --- | --- |
| 类型定义 | 子类型定义 | 对应记录主要内容 | 备注 |
| PLC\_ENGINEER | LOGIC\_CODE | 工程师PLC代码数据的描述 |  |
| LOGIC\_BIN | 工程师PLC二进制逻辑的描述 |  |
| LOGIC\_UPDATE | 工程师代码的上传命令 |  |
| LOGIC\_RETURN | 代码上传后的返回数据 |  |
| PLC\_OPERATOR | PLC\_CMD | PLC发送的命令，包括启动命令，关闭命令，控制命令，监控命令，调节命令 |  |
| PLC\_RETURN | PLC返回结果，包括PLC命令的返回结果以及管理中心警告的返回结果。 |  |
| PLC\_MONITOR | PLC\_CMD | 管理中心监视命令，包括警告命令、审计命令、权限控制命令 | 数据监视命令、控制命令可以直接使用PLC\_OPERATOR命令的对应格式， |
| PLC\_AUDIT | 审计数据描述，包括工程师命令审计、操作员命令审计、系统监视员命令审计 | 给出审计数据的类型、范围、存储文件 |
| PLC\_RETURN | 管理中心命令返回结果。 |  |
| PLC\_DEVICE | REGISTER | PLC寄存器描述 | 描述寄存器类型、地址、对应的 |
| MODBU S\_TCP | MBAP | Modbus协议的包头 | 长度为6字节 |
| DATAGRAM | modbus数据报， | 最开始的data\_size不属于数据报内容，是用来描述最后一项数据长度的。 |
| MODBUS\_STATE | MASTER | Modbus Master设备的属性 | 包含modbus\_master所连接的从设备数目 |
| SLAVE | Modbus slave设备的属性 |  |
| SERVER | Modbus协议连接server端信息 | server端处于slave方向 |
| CHANNEL | Modbus协议连接channel端信息 | channel端处于slave方向 |
| CLIENT | Modbus协议连接Client端信息 | Client端处于master方向 |
| RELATE | Modbus发送命令与返回命令关联信息记录 | modbus传输标识与命令消息uuid的关系，在master端记录 |
| MODBUS\_CMD | XXX | Modbus协议发送的命令，每种协议一种定义，数量较多，此处忽略 |  |
| ... |  |  |
| MODBUS\_DATA | XXX | Modbus命令返回的数据，每种协议一种定义，数量较多，此处忽略 |  |
| ... |  |  |

## 附录3：枚举值定义

枚举值用来为不同场景下不同的选择给出对应的编号和定义，这些定义将用在数据类型定义中。

用户类型定义emu\_plc\_rule\_type

（位于检查用户登录过程的user\_define.json数据结构中）。

|  |  |  |  |
| --- | --- | --- | --- |
| 枚举项名称 | 枚举数值 | 内容 | 备注 |
| PLC\_ENGINEER | 0x01 | 工程师身份 |  |
| PLC\_OPERATOR | 0x02 | 操作员身份 |  |
| PLC\_MONITOR | 0x03 | 监控员身份 |  |
| PLC\_DEVICE | 0x10 | 设备（特指PLC逻辑执行的行为） |  |

文件类型定义 emu\_plc\_file\_type

（位于PLC模拟操作的plc\_emu.json）

|  |  |  |  |
| --- | --- | --- | --- |
| 枚举项名称 | 枚举数值 | 内容 | 备注 |
| FILE\_CODE | 0x01 | 代码文件 |  |
| FILE\_LOGIG | 0x02 | PLC逻辑文件 | 由对应代码文件生成 |
| FILE\_DESC | 0x10 | 描述文件 | 保留待以后选用 |
| FILE\_RULE | 0x20 | 制度文件 | 保留待以后选用 |

操作员操作类型定义 enum\_plc\_operator\_action

（位于plc\_emu.json）

|  |  |  |  |
| --- | --- | --- | --- |
| 枚举项名称 | 枚举数值 | 内容 | 备注 |
| ACTION\_ON | 0x01 | 开启操作 |  |
| ACTION\_OFF | 0x02 | 关闭操作 |  |
| ACTION\_MONITOR | 0x03 | 监视操作 | 读取寄存器信息 |
| ACTION\_ADJUST | 0x04 | 设置操作 | 调节恒温设置、档位设置 |

监视员操作类型定义 enum\_plc\_monitor\_action

（位于plc\_emu.json）

|  |  |  |  |
| --- | --- | --- | --- |
| 枚举项名称 | 枚举数值 | 内容 | 备注 |
| ACTION\_WARNING | 0x0100 | 警告操作 |  |
| ACTION\_AUDIT | 0x0101 | 审计操作 |  |
| ACTION\_CONTROL | 0x0102 | 控制操作 |  |

寄存器类型定义 enum\_modbus\_register\_type

（位于描述设备的plc\_dev.json）

|  |  |  |  |
| --- | --- | --- | --- |
| 枚举项名称 | 枚举数值 | 内容 | 备注 |
| COILS | 0x01 | 线圈状态 |  |
| DISCRETE\_INPUT | 0x02 | 离散输入状态 |  |
| INPUT\_REGISTER | 0x03 | 输入寄存器 |  |
| HOLDING\_REGISTER | 0x04 | 保持寄存器 |  |

modbus协议定义 enum\_modbus\_protocol\_id

（位于modbus\_tcp.json)

这里的线圈一般对应开关。

|  |  |  |  |
| --- | --- | --- | --- |
| 枚举项名称 | 枚举数值 | 内容 | 备注 |
| READ\_COILS | 0x01 | 读线圈状态 |  |
| READ\_DISCRETE\_INPUTS | 0x02 | 读离散输入状态 |  |
| READ\_HOLDING\_REGISTERS | 0x03 | 读保持寄存器状态 |  |
| READ\_INPUT\_REGISTERS | 0x04 | 读输入寄存器状态 |  |
| WRITE\_SINGLE\_COIL | 0x05 | 写线圈状态 |  |
| WRITE\_SINGLE\_REGISTER | 0x06 | 写单个寄存器 |  |
| READ\_EXCEPTION\_STATUS | 0x07 | 读特殊状态 |  |
| WRITE\_MULTIPLE\_COILS | 0x0F | 写多个线圈 |  |
| WRITE\_MULTIPLE\_REGISTERS | 0x10 | 写多个寄存器 |  |
| REPORT\_SLAVE\_ID | 0x11 | 报告从设备ID |  |

## 附录4 数据类型定义表

### 4.1 工程师（PLC\_ENGINEER）相关类型定义

（PLC\_ENGINEER,LOGIC\_CODE）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| logic\_filename | 0结尾字符串 | <=32字节 | 代码文件的名称 |  |
| code\_uuid | UUID | 32字节 | 代码的摘要值 |  |
| author | 0结尾字符串 | <=32字节 | 代码作者的名称 |  |
| time | INT | 4字节 | 代码完成时间 |  |

（PLC\_ENGINEER,LOGIC\_BIN）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| logic\_filename | 0结尾字符串 | <=32字节 | 代码文件的名称 |  |
| code\_uuid | UUID | 32字节 | 代码的摘要值 |  |
| bin\_uuid | UUID | 32字节 | 二进制逻辑的摘要值 |  |
| author | 0结尾字符串 | <=32字节 | 代码作者的名称 |  |
| time | INT | 4字节 | 代码完成时间 |  |

（PLC\_ENGINEER,LOGIC\_UPLOAD）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| logic\_filename | 0结尾字符串 | <=32字节 | 代码文件的名称 |  |
| type | ENUM | 4字节 | 文件类型 | **使用enum\_plc\_file\_type枚举** |
| uuid | UUID | 32字节 | 代码的摘要值 |  |
| author | 0结尾字符串 | <=32字节 | 代码作者的名称 |  |
| uploader | 0结尾字符串 | <=32字节 | 上传行为执行者名称 |  |
| time | INT | 4字节 | 上传行为执行时间 |  |

（PLC\_ENGINEER,LOGIC\_RETURN) 数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| logic\_filename | 0结尾字符串 | <=32字节 | 上传文件的名称 |  |
| uuid | UUID | 32字节 | 文件的摘要值 |  |
| author | 0结尾字符串 | <=32字节 | 代码作者的名称 |  |
| result | INT | 4字节 | 上传结果，0为成功 |  |
| time | INT | 4字节 | 上传行为执行时间 |  |

### 4.2 操作员（PLC\_OPERATOR）相关类型定义

（PLC\_OPERATOR,PLC\_CMD）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| action | ENUM | 4字节 | 操作员的行为 | 使用  enum\_plc\_operator\_action |
| action\_desc | 0结尾字符串 | <=32字节 | 操作员行为描述 | **使用寄存器名称进行描述** |
| value | INT | 4字节 | 寄存器设置的值 |  |
| plc\_operator | 0结尾字符串 | <=32字节 | 操作者的名称 |  |
| time | INT | 4字节 | 操作时间 |  |

（PLC\_OPERATOR,PLC\_RETURN）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| action | ENUM | 4字节 | 操作员的行为 | 使用  enum\_plc\_operator\_action |
| action\_desc | 0结尾字符串 | <=32字节 | 操作员行为描述 | **使用寄存器名称进行描述** |
| value | INT | 4字节 | 寄存器设置的值 |  |
| result | INT | 4字节 | 执行结果，0为成功 |  |
| time | INT | 4字节 | 操作时间 |  |

### 4.3 系统监视员（PLC\_MONITOR）相关类型定义

（PLC\_MONITOR,PLC\_CMD）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| action | ENUM | 4字节 | 操作员的行为 | 使用  enum\_plc\_monitor\_action |
| action\_desc | 0结尾字符串 | <=32字节 | 操作员行为描述 | **对行为的描述字符串** |
| value1 | INT | 4字节 | 行为的第一个参数 |  |
| value2 | INT | 4字节 | 行为的第二个参数 |  |
| plc\_monitor | 0结尾字符串 | <=32字节 | 监控者的名称 |  |
| time | INT | 4字节 | 监控时间 |  |

（PLC\_MONITOR,PLC\_AUDIT）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| user\_type | ENUM | 4字节 | 审计对象的类型的行为 | 使用  enum\_plc\_role\_type |
| action\_desc | 0结尾字符串 | <=32字节 | 操作员行为描述 | **对行为的描述字符串** |
| start\_time | INT | 4字节 | 审计起始时间 |  |
| value2 | INT | 4字节 | 需审计的数据项数目。 |  |
| end\_time | INT | 4字节 | 审计终止时间 |  |

（PLC\_OPERATOR,PLC\_RETURN）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| action | ENUM | 4字节 | 操作员的行为 | 使用  enum\_plc\_operator\_action |
| action\_desc | 0结尾字符串 | <=32字节 | 操作员行为描述 | **使用寄存器名称进行描述** |
| value | INT | 4字节 | 寄存器设置的值 |  |
| result | INT | 4字节 | 执行结果，0为成功 |  |
| time | INT | 4字节 | 操作时间 |  |

### 4.4 设备（PLC\_DEVICE）相关类型定义

（PLC\_OPERATOR,PLC\_RETURN）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| plc\_devname | 0结尾字符串 | <=32字节 | 设备的名称 |  |
| type | ENUM | 4字节 | 寄存器的类型 | 使用  enum\_modbus\_register\_type |
| addr | INT | 4字节 | 寄存器的地址 |  |
| register\_desc | 0结尾字符串 | <=32字节 | 寄存器的名字 |  |

### 4.5 MODBUS\_TCP 类型定义

本类型用来描述modbus\_tcp协议的数据内容，包括包头数据MBAP和数据报DATAGRAM。

(MODBUS\_TCP, MBAP）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| trans\_id | UINT16 | 2字节 | modbus传输标识 | 从0开始，每次发送命令时+1，返回时数据与命令一致 |
| protocol\_id | UIN16 | 2字节 | 协议标识 | 取值为0 |
| length | UIN16 | 2字节 | 数据长度 | 高位为0，低位为0-255间 |

(MODBUS\_TCP,DATAGRAM）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| datasize | UINT16 | 2字节 | 数据长度（字节） | 0-247之间 |
| unit\_id | UCHAR | 1字节 |  | 取值为0 |
| function | BENUM | 1字节 | 功能编号 | 对应枚举enum\_modbus\_protocol\_id |
| data | DEFINE | datasize | 协议传输的数据 |  |

### 4.6 MODBUS\_STATE 类型定义

本类型用来描述modbus\_tcp连接中主设备、从设备的设备信息、网络信息，以及主设备端命令与消息的关联关系。

(MODBUS\_STATE,MASTER）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| master\_name | 定长字符串 | 32字节 | 主设备名称 |  |
| unit\_num | UINT16 | 2字节 | 从设备数目 |  |
| slave\_uuid | UUID | 32字节 | 从设备唯一编号 | 与从设备共享的秘密 |

(MODBUS\_STATE,SLAVE）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| slave\_name | 定长字符串 | 32字节 | 从设备名称 |  |
| ip | 定长二进制数据 | 4字节 | 从设备ip |  |
| port | UINT16 | 2字节 | 端口号 |  |
| unit\_addr | UINT16 | 2字节 | 从设备地址 |  |
| state\_machine | SENUM | 2字节 | 从设备状态机 |  |
| slave\_key | BINDATA | 32字节 | 从设备密钥 |  |

(MODBUS\_STATE,SERVER）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| server\_name | 定长字符串 | 32字节 | server端名称 |  |
| unit\_addr | UINT16 | 2字节 | 从设备地址 |  |
| state\_machine | SENUM | 2字节 | 从设备状态机 |  |
| server\_key | BINDATA | 32字节 | 从设备密钥 |  |

(MODBUS\_STATE,CHANNEL）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| client\_name | 定长字符串 | 32字节 | 所连接客户端名称 |  |
| unit\_addr | UINT16 | 2字节 | 从设备地址 |  |
| client\_ip | 定长二进制数据 | 4字节 | 客户端ip |  |
| port | UINT16 | 2字节 | 端口号 |  |
| state\_machine | SENUM | 2字节 | 从设备状态机 |  |
| no | UINT16 | 2字节 | 从设备编号 |  |
| client\_key | BINDATA | 32字节 | 从设备密钥 |  |

(MODBUS\_STATE,CLIENT）数据结构的定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| client\_name | 定长字符串 | 32字节 | 客户端名称 |  |
| server\_name | 定长字符串 | 32字节 | 连接的服务端名称 |  |
| unit\_addr | UINT16 | 2字节 | 地址 |  |
| state\_machine | SENUM | 2字节 | 主设备端状态机 |  |
| no | UINT16 | 2字节 | 从设备编号 |  |
| client\_key | BINDATA | 32字节 | 客户端密钥 |  |

(MODBUS\_STATE,RELATE）数据结构的定义：

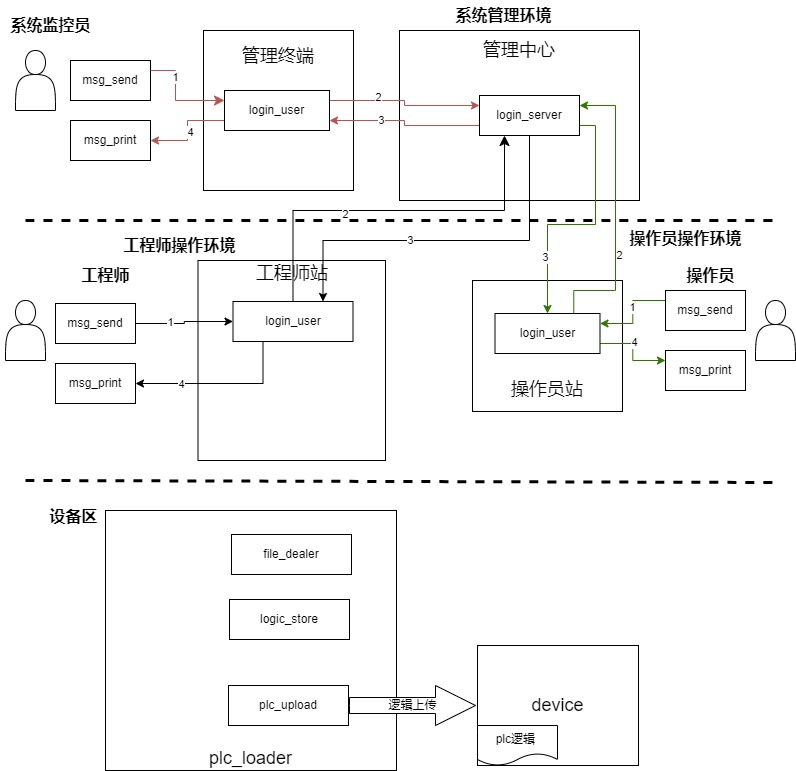
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 元素名称 | 元素类型 | 元素大小 | 信息内容 | 备注 |
| unit\_addr | UINT16 | 2字节 | 从设备地址 |  |
| no | UINT16 | 2字节 | 传输id编号 |  |
| msg\_uuid | UUID | 32字节 | 命令唯一编号 | 携带命令消息的编号 |

## 附录5：操作流程详细介绍

本架构模拟中，业务操作是通过消息路由将各种功能模块的功能衔接在一起实现的。因此，对操作流程的介绍实际就是给出实现这些操作的消息路由，以及列出消息路由中各模块的分工、输入和输出。

### 5.1 用户登录流程

用户登录的消息路由图设计如下，路由名称为login,三类用户采用三种不同的颜色加以区分。

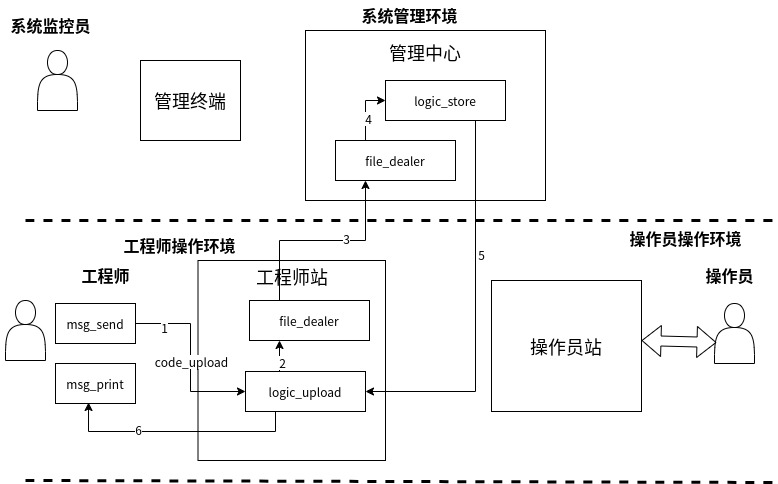


路由中各模块功能分解如下表所示：

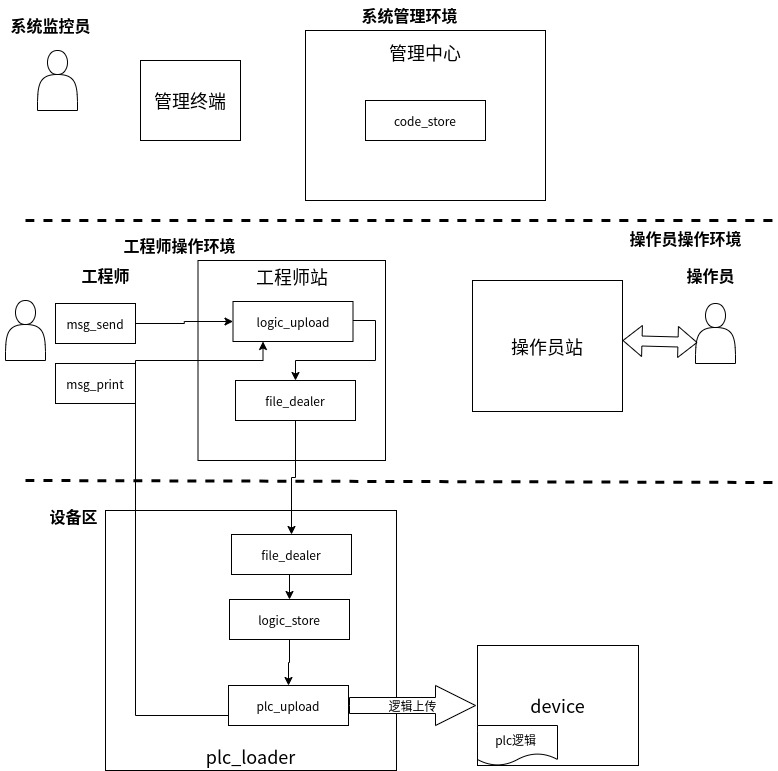
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实例名（按传输次序） | 模块名（按传输次序） | 模块功能 | 输入信息 | 输出信息 | 备注 |
| 用户（工程师，操作员，管理中心） | msg\_send  (旧模块） | 产生用户登录信息 | 无 | 用户登录信息  （GENERAL\_RETURN,  STRING) | 登录信息name取值“login”,return\_value按照“用户名:口令”方式 |
| 终端（  工程师站，操作员站，管理终端） | login\_user  （旧模块） | 将用户登录信息转化为标准的用户登录格式 | 用户登录信息  （GENERAL\_RETURN,  STRING) | 用户登录消息（标准格式）  （USER\_DEFINE,  LOGIN） | login\_user内部可以实现一些安全登录协议 |
| 管理中心 | login\_server  （旧模块） | 验证用户登录信息并返回验证结果 | 用户登录消息（标准格式）  （USER\_DEFINE,  LOGIN） | 用户登录返回结果  （USER\_DEFINE,  RETURN） | 依据user.list中的用户名、身份和属性进行 |
| 终端（  工程师站，操作员站，管理终端） | login\_user  （旧模块） | 处理返回消息 | 用户登录返回结果  （USER\_DEFINE,  RETURN） | 用户登录返回结果  （USER\_DEFINE,  RETURN） | 根据用户登录结果，设置终端上下文中的用户名和用户状态 |
| 用户（工程师，操作员，管理中心） | msg\_print  （旧模块） | 将返回消息输出到屏幕 | 用户登录返回结果  （USER\_DEFINE,  RETURN） | 用户登录返回结果  （USER\_DEFINE,  RETURN） |  |

### 5.2 工程师程序上传流程

工程师上传代码的消息路由为code\_upload，路由图如下所示：



工程师上传二进制数据的消息路由为bin\_upload，路由图如下所示：



工程师程序上传流程功能分解：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实例名（按传输次序） | 模块名（按传输次序） | 模块功能 | 输入信息 | 输出信息 | 备注 |
| 工程师 | msg\_send | 产生程序上传消息 | 程序上传消息文件 | 程序上传消息 | 已有模块 |
| 工程师站 | logic\_upload | 根据程序上传消息发出文件发送申请 | 程序上传消息 | 程序上传消息  文件发送申请（FILE\_TRANS,  REQUEST） |  |
| file\_dealer | 根据文件发送申请发送代码文件数据 | 文件发送申请 | 文件发送数据  (FILE\_TRANS,  FILE\_DATA) | 已有模块 |
| 管理中心 | file\_dealer | 接收文件数据，完成接收后输出文件名 | 文件接收数据 | 文件接收通知  （FILE\_TRANS,  FILE\_NOTICE) | 已有模块 |
| logic\_store | 存放文件信息，生成返回数据 | 程序上传消息  文件接收通知 | 文件接收返回数据 |  |
| 工程师站 | logic\_upload | 处理返回消息 | 程序上传返回消息 | 程序上传返回消息 |  |
| 工程师 | msg\_print | 将返回消息输出到屏幕 | 程序上传返回消息 | 屏幕输出返回消息内容 | 已有模块 |

工程师逻辑上传流程功能分解：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实例名（按传输次序） | 模块名（按传输次序） | 模块功能 | 输入信息 | 输出信息 | 备注 |
| 工程师 | msg\_send | 产生逻辑上传消息 | 逻辑上传消息文件 | 逻辑上传消息 | 已有模块 |
| 工程师站 | logic\_upload | 根据逻辑上传消息发出文件发送申请 | 逻辑上传消息 | 逻辑上传消息  文件发送申请（FILE\_TRANS,  REQUEST） |  |
| file\_dealer | 根据文件发送申请发送动态库文件数据 | 文件发送申请 | 文件发送数据  (FILE\_TRANS,  FILE\_DATA) | 已有模块 |
| plc\_loader | file\_dealer | 接收文件数据，完成接收后输出文件名 | 文件接收数据 | 文件接收通知  （FILE\_TRANS,  FILE\_NOTICE) | 已有模块 |
| logic\_store | 存放文件信息，生成返回数据 | 逻辑上传消息  文件接收通知 | 文件接收返回数据 |  |
| plc\_upload | 将上传的plc逻辑复制到plc device中 | 逻辑上传消息 | 逻辑上传返回数据 |  |
| 工程师站 | plc\_upload | 处理返回消息 | 逻辑上传返回消息 | 逻辑上传返回消息 |  |
| 工程师 | msg\_print | 将返回消息输出到屏幕 | 程序上传返回消息 | 屏幕输出返回消息内容 | 已有模块 |

### 5.3 操作员操作流程

操作员操作流程使用了一种特殊的消息路由方式来实现，我们称其为接力传输。

本应用的消息路由是由四个路由、五个阶段组成的。

阶段1: 操作员-操作员站的operator\_cmd 路由，明面上该路由是操作员发送命令消息，到操作员站的cmd\_process模块后返回，但实际上，操作员发送的命令消息到了cmd\_process模块后，就使用slot\_sock机制来暂存消息，同时根据命令生成携带消息uuid为扩展项的modbus命令消息。

阶段2： 操作员站内部的command\_send路由，这个路由传给modbus\_master模块，modbus\_master确定modbus对应的地址、寄存器、命令、trans\_id等信息，并且把这些信息和uuid一起存起来（使用MODBUS\_STATE,RELATE)数据结构。而后modbus\_master将modbus命令转化为modbus\_tcp包(由包数据(MODBUS\_TCP,DATAGRAM)和包头（MODBUS\_TCP,MBAP)组成，这个消息传给modbus\_channel模块后输出。

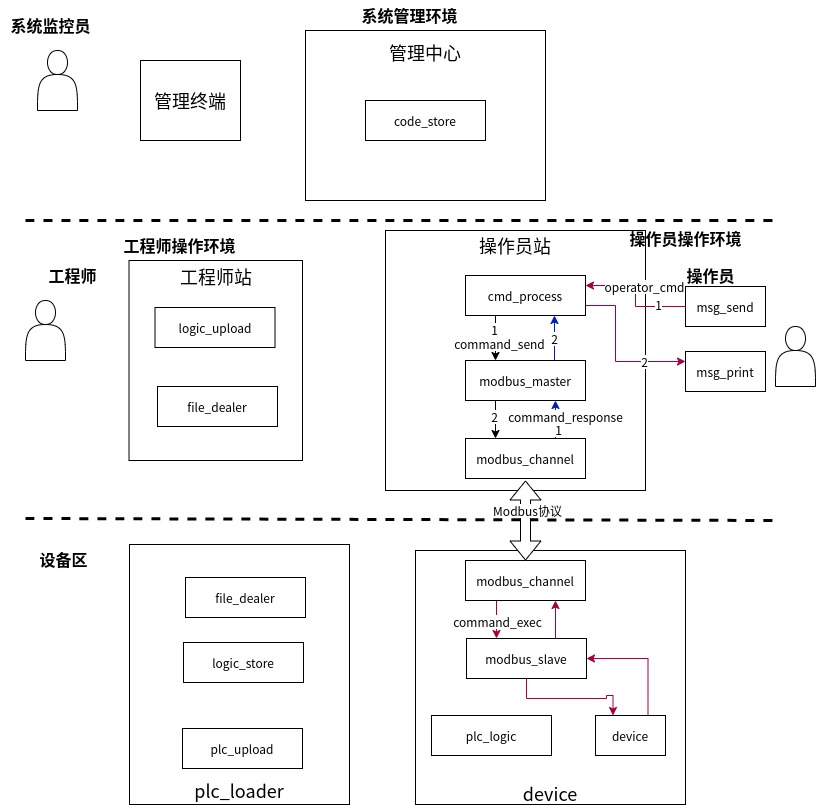
阶段3：PLC设备的modbus\_channel模块收到modbus协议数据后，执行command\_exec路由，路由获取modbus\_channel发送过来的modbus命令，转给设备，由设备执行后，返回数据到modbus\_channel数据通道，返回的数据其trans\_id和地址信息与输入命令一致。

阶段4： 操作员站收到modbus协议传来的数据后，转为MODBUS\_TCP两个数据结构的组合，发送给modbus\_master模块，modbus\_master模块获取设备地址和trans\_id后，查找数据库获得消息uuid,作为扩展项附在modbus\_master生成的modbus数据消息上，发送给cmd\_process。

阶段5：cmd\_process根据扩展项中的消息uuid，找到暂存的operator\_cmd路由消息，根据modbus数据内容生成（PLC\_OPERATOR,PLC\_CMD)格式消息，并延续operator\_cmd路由，即可返回操作员。

这一方式对系统监控员的操作命令也成立。

消息路由图如下图所示：

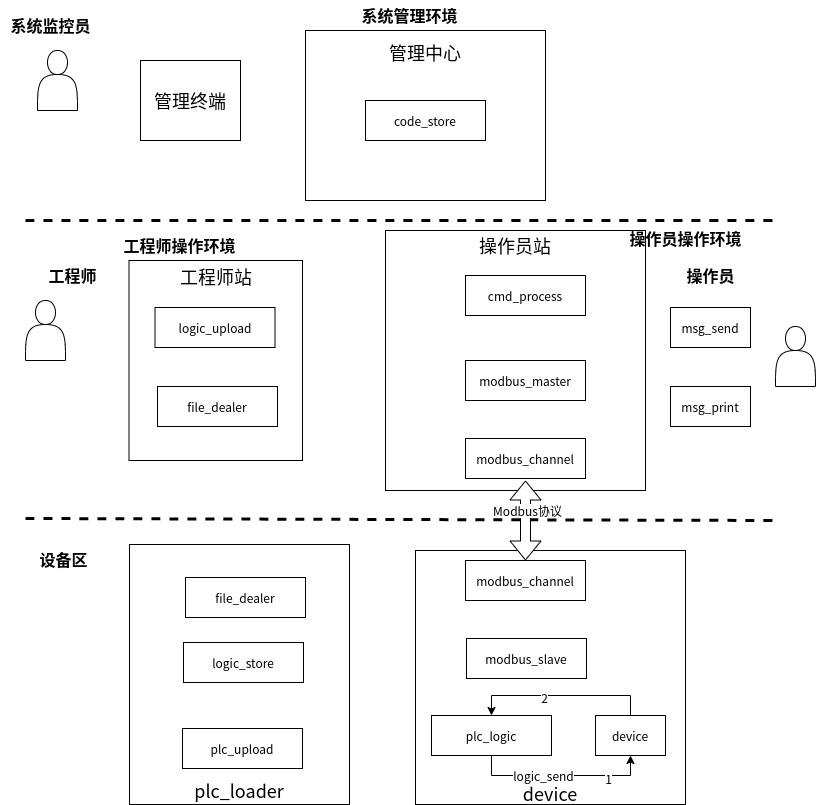


操作员操作流程功能分解：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实例名（按传输次序） | 模块名（按传输次序） | 模块功能 | 输入信息 | 输出信息 | 备注 |
| 操作员 | msg\_send  （旧模块） | 生成操作命令 | 操作员操作消息文件 | 操作员操作消息  （PLC\_OPERATOR,  PLC\_CMD) |  |
| 操作员站 | cmd\_process | 将操作员操作行为记录下来，并转化为Modbus协议命令 | 操作员操作消息  （PLC\_OPERATOR,  PLC\_CMD) | Modbus命令（MODBUS\_CMD,  XXXXXX）  扩展项  （GENERAL\_RETURN，UUID),其name为设备的名称，uuid为收到消息的uuid | 操作员操作消息被slot\_sock机制缓存 |
| modbus\_master | 对Modbus命令进TCP格式转化 | Modbus命令（MODBUS\_CMD,  XXXXXX） | Modbus\_TCP数据（MODBUS\_TCP,DATAGRAM)  扩展项（消息头）  （MODBUS\_TCP,MBAP) | 生成地址、trans\_id，消息UUID的关联信息并存储 |
| modbus\_channel  （master端） | 根据modbus数据输出modbus协议数据 | Modbus\_TCP数据（MODBUS\_TCP,DATAGRAM)  扩展项（消息头）  （MODBUS\_TCP,MBAP) | modbus\_tcp协议数据 | 已有模块 |
| plc设备 | modbus\_channel  （slave端） | 接收modbus协议数据，转化为modbus命令输出 | modbus\_tcp协议数据 | Modbus命令（MODBUS\_CMD,  XXXXXX) | 已有模块 |
| modbus\_slave | 对Modbus命令进行从设备端信息补充 | Modbus命令（MODBUS\_CMD,  XXXXXX) | Modbus命令（MODBUS\_CMD,  XXXXXX) | 已有模块 |
| plc\_device | 接收modbus命令，模拟设备执行并返回结果 | Modbus命令（MODBUS\_CMD,  XXXXXX) | Modbus数据（MODBUS\_DATA,  XXXXXX) |  |
| modbus\_slave | 对Modbus**数据**进行从设备端信息补充 | Modbus数据（MODBUS\_DATA,  XXXXXX) | Modbus数据（MODBUS\_DATA,  XXXXXX) | 已有模块 |
| modbus\_channel | 根据modbus数据输出modbus协议返回数据 | Modbus数据（MODBUS\_DATA,  XXXXXX) | modbus\_tcp数据 |  |
| 操作员站 | modbus\_channel | 接收modbus协议数据，转化为modbus命令输出 | modbus\_tcp协议数据 | Modbus\_TCP数据（MODBUS\_TCP,DATAGRAM)  扩展项（消息头）  （MODBUS\_TCP,MBAP) |  |
| modbus\_master | 对Modbus**数据**进行主设备端信息补充 | Modbus\_TCP数据（MODBUS\_TCP,DATAGRAM)  扩展项（消息头）  （MODBUS\_TCP,MBAP) | Modbus数据（MODBUS\_DATA,  XXXXXX)  扩展项  （GENERAL\_RETURN，UUID),其name为设备的名称，uuid为收到消息的uuid |  |
| cmd\_process | 将Modbus协议返回数据转化为操作员命令返回数据 | Modbus数据（MODBUS\_DATA,  XXXXXX)  扩展项  （GENERAL\_RETURN，UUID),其name为设备的名称，uuid为收到消息的uuid） | 操作员操作返回数据  （PLC\_OPERATOR,  PLC\_RETURN) | 从slot\_sock中获取消息，发送消息继续缓存消息路由 |
| 操作员 | msg\_print | 将返回消息输出到屏幕 | 操作员操作返回数据  （PLC\_OPERATOR,  PLC\_RETURN) | 屏幕输出返回消息内容 | 已有模块 |

### 5.4 系统自动监视流程

系统自动监视流程由plc逻辑向plc设备输出modbus命令，接收modbus信息并返回plc逻辑，由plc逻辑进行处理。该模型在设备启动后即开始自动执行，消息路由图如下图所示：



流程模块功能分解：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| plc设备 | plc\_logic | 生成内部的Modbus命令 | 无 | Modbus命令（MODBUS\_CMD,  XXXXXX) |  |
| device | 模拟device的行为，接收modbus命令，返回数据 | Modbus命令（MODBUS\_CMD,  XXXXXX) | Modbus数据（MODBUS\_DATA,  XXXXXX) | 模拟模块，根据modbus行为进行处理 |
| plc\_logic | 接收modbus返回数据，执行内部逻辑 | Modbus数据（MODBUS\_DATA,  XXXXXX) | 无 |  |

### 5.5 管理中心远程控制流程

管理中心远程控制流程与4.3 操作员操作流程原理类似，只是阶段1和阶段5是一个比较长的从管理员到工作站的路由monitor\_ctrl，消息路由图如下图所示（这里monitor\_manage和control\_center只是示意，实际中并无这两个模块，但这部分与解题无关）：

