南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论文题目	自动化测试系统远程调试子系统的设计与实现
作者姓名	蒋宇阳
学科、专业名称	工程硕士（软件工程方向）
研究方向	软件工程
指导教师	邵栋　副教授

2015 年 05 月 10 日

学　　号：**MF1332023**

论文答辩日期：　　年　月　日

指 导 教 师：　　（签字）

自动化测试系统远程调试子系统的设计与实现

作　　者：　　蒋宇阳

指导教师：　　邵栋　副教授

南京大学研究生毕业论文

（申请工程硕士学位）

南京大学软件学院

2015 年 05 月

The Design and Implementation of Remote Debug Subsystem in Intelligent Test System

Jiang, Yuyang

Submitted in partial fulfillment of the requirements for the degree of Master of Engineering

Supervised by

Associate Professor **Shao, Dong**

Software Institute

NANJING UNIVERSITY

Nanjing, China

May, 2015

摘 要

随着目前社会信息化水平逐步提高，软件产品的规模也在逐年扩大，因此，自动化软件测试也越来越重要。针对测试运行在主板上的程序的需求，Intel 公司设计并实现了一个自动化测试系统 ITS（Intelligent Test System），该系统可以运行用户预先编写好的脚本，对被测试的串口机器、SUT 机器、Switcher 机器进行自动化的测试。多个运行了 ITS 的 Agent 可以连接上一个服务器组成云平台，由云平台对所有的 Agent 进行统一管理，智能分配任务到空闲的 Agent 上，提高资源的利用率，也方便同组人员协调工作。

本文的远程调试子系统是自动化测试系统云平台的一个重要子系统，它提供了远程控制连接在 Agent 上的被测机器和 Switcher 机器以及远程控制运行在 Agent 上的 ITS 控制台的功能，以及在运行脚本遇到错误的时候记录错误发生的位置，在调试错误的时候运行到出错处暂停并进入调试模式的功能。这样用户可以在自己电脑上对运行在远程 Agent 上的脚本进行调试，并可以实时观察并控制各个硬件设备或者 ITS 控制台，方便找出错误并修正错误。

子系统分为控制和调试两大模块。控制模块又分为远程串口、远程 SUT、远程 Switcher 以及远程控制台几个模块，这些模块大多分为 Agent 和 Applet 两端，其中 Applet 端采用 Java Applet 技术实现通过浏览器进行远程控制，两端之间的沟通由采用 Netty 框架并使用 WebSocket 协议的通讯模块负责。调试模块又可以分为记录模块和重现模块，用以记录错误发生处以及在调试时重现错误现场。

子系统最终在整个云平台中运行良好，用户可以成功地在远端浏览器打开各种远程控制界面，再配合相关的调试功能，实现在远程调试运行在 Agent 上的脚本。

关键词：WebSocket 协议、Netty 框架、Java Applet 技术、远程控制

Abstract

With the rapid development of computer information technology, the scale of the software is increasing and the automated software testing is becoming more and more important. So Intel developed the Intelligent Test System to test the software on the motherboard. The ITS can run the script to test the serial machine, SUT machine and Switcher automatically. Multiple Agent which runs the ITS can connect to one server to build up the cloud platform which can manage all connected Agent. The cloud platform can assign the task to the idle agent to improve resource utilization and can make the cooperation among the team become more convenient.

Remote debug subsystem is an important subsystem in Intelligent Test System.It can remote control the machine which should be tested and the switcher which be connected to the Agent and can control the console of ITS which running on the Agent. When the script meets bug, it can record the fail point and pause in this point when you want to debug this error. So the user can debug the script which running in the Agent by remote computer, and user can control the machine and console to do with the bug.

This subsystem is composed by the remote control module and the debug module. The remote module includes remote serial module, remote SUT module, remote Switcher module and remote console module. These modules are divided into two sides: Agent side and Applet side. The Applet side use Java Applet to remote control by browser, and the Netty frame which supports WebSocket is responsible for the communication between the two sides. The debug module includes the bug record module which is used to record the location of the bug and the bug reappear module which reappear the scene when somebody wants to debug.

This subsystem works well in the cloud platform finally. User can open the remote control UI in browser to debug the script in remote Agent to find the bug by the relevant debugging features successfully.

Keywords：WebSocket, Netty, Java Applet, Remote Control

目 录

图目录

表目录

第一章 引言

1.1 项目背景

近年来，随着社会信息化程度的日益提高，在日常生产生活中对计算机软件的需求越来越大。这导致软件产品越来越多，同时单个软件产品的规模越来越大，其复杂度也越来越高。如何保证软件产品的质量，成为了近年来软件行业所面临的主要问题之一。[梁家安, 2011]

由于软件规模越来越大，以及频繁的软件需求变更导致的软件迭代越来越快，传统手工测试已经不能满足当前软件测试的需求，而自动化测试为此提供了成功的解决方案。[应杭, 2006]

因此，针对测试运行在主板上的程序的需求，开发了 ITS（Intelligent Test System），也就是自动化测试系统。该系统运行在单台电脑上的时候是一个自动化测试系统，它可以通过读取用户预先编写好的测试脚本，对连接上本机的被测机器，比如串口机器（如运行 BIOS 的主板）或者 SUT 机器（指有视频显示的机器，比如有 HDMI 输出的平板）进行操作，并可以对返回的输出流进行判断从而生成测试日志。

考虑到一方面运行 ITS 的机器（Agent）、运行待测试固件的串口机器、运行待测试桌面软件的 SUT 机器等都是昂贵的资源，另一方面同一个小组里的测试人员、开发人员以及项目经理等也缺少一个统一的平台。因此，进一步开发了统合多台 ITS 的云平台。

云平台会统一管理所有连接上来的 Agent、所有的测试脚本、所有需要测试的固件等等，当需要运行脚本的时候，云平台会智能地从所有空闲的 Agent 中寻找到一个符合该脚本运行条件（比如需要一台串口设备）的机器，将需要运行的脚本打包发送过去。还会根据最近几次脚本的运行情况，估计出某个脚本可能还会需要运行的时间，以便分配任务。当脚本运行失败出现 bug 时，云平台还会自动发送邮件通知负责人员，并可以重现出错场景以便调试。

一台云平台的服务器上连接的所有 Agent 可能分布在不同的实验室甚至不同的城市。当某个脚本运行出错时，仅仅凭借错误日志也许并不能够解决问题，

而开发人员和测试人员又是无法赶到出错的 Agent 上去当场查看运行脚本时刻的各个被测机器的状况，来进行脚本的调试工作。

因此，云平台需要支持用户在自己的电脑上通过浏览器将测试任务发送到 Agent 上，用户也能够在自己的电脑上看到远程 Agent 所连接的各种被测机器的状态并进行控制。本文设计并实现了远程调试子系统，以便可以在出 bug 的情况下进行远程的调试。

1.2 国内(外)相关系统的发展概况

1.2.1 国内(外)自动化测试系统的发展概况

Microsoft Application Center Test (ACT)是微软公司的一个基于软件的自动化测试工具，它主要对 Web 服务器进行测试，使用户能够获取执行流程，分析和判断流程中遇到的问题，并根据测试流程生成测试报告。测试人员通过 ACT 测试 Web 应用程序时，先录制或者编写一个测试脚本，模拟同一时间一个或者多个用户提出连接请求。[陆璐等, 2006]

ACT 支持动态测试，也就是支持用户以录制或者是手动编写的方式，创建一组同时连接到服务器的 Http 请求队列。同时用户可以动态的修改这些 Http 请求，比如可以修改其中的 Cookies 信息，或者是可以修改 Http 头部和提交者、用户代理相关的信息，同时修改主机信息、Http 请求的版本号以及其他一些存储在 Http 请求头部的信息。

ACT 还支持用户的概念，测试人员可以创建用户来模拟真实环境下的使用者，还可以给这些用户设置延迟时间以更贴近真实用户，还可以设置用户的账号密码等等来应对 Web 的验证。

ACT 支持多种通用的认证模式或者是加密算法。ACT 可以很方便的自动录制和执行匿名认证、Basic 认证和分类认证，对于 Windows 综合认证和微软.Net Passport 护照认证虽然没有办法自动记录生成测试脚本，但是可以通过手动编辑测试脚本达到测试的目的。ACT 完全支持执行 40 位或者 128 位 SSL 加密的测试脚本，当然，因为数据到达时是经过加密的，所以无法进行自动记录，但是可以手动修改脚本创建到 443 端口的使用 SSL 的连接达到测试目的。

LoadRunner 是惠普公司的一款著名的自动化性能测试工具。

LoadRunner 广泛支持业界标准协议，包括 EJB（RMI-Java 协议）、CS（MS SQL 协议）、Streaming（RealPlayer 协议）、Middleware（Tuxedo6 协议）、E-business（FTP 协议）等等。LoadRunner 也支持各种平台的开发脚本，支持各种开发语言比如 Java、.Net、C 等等，这不但降低了学习开发脚本的成本，同时可以应付各种情况下的问题。[陈绍英等, 2007]

LoadRunner 可以创建真实的系统负载。LoadRunner 通过创建一个测试场景进行测试，在其中选择运行脚本、设置虚拟用户数量以及其在测试时的行为、选择负载发生器、设置执行时间。LoadRunner 通过事务来测试服务器的性能，事务在业务上体现为使用者的一系列操作，代表一定的功能，而在脚本中则体现为一段测试脚本，可以用来衡量并发响应时间。

LoadRunner 还拥有强大的实时监控以及数据采集能力，实时显示的数据方便测试人员把握当前运行状况，并可以第一时间发现测试问题。在测试结束之后，LoadRunner 可以通过汇总并分析全部的测试结果，快速定位发生错误的地方，方便测试人员解决问题。

以上的测试软件都是测试服务器或者测试本机软件，无法测试某些运行在主板上的固件或者 BIOS，所以开发了 ITS，让它运行在 PC 上通过串口或者视频采集卡测试这些主板设备上的程序。

1.2.2 国内(外)远程控制系统的发展概况

VNC(Virtual Network Computing，虚拟网络计算)[Richardson, 1998]是一款基于 UNIX 和 Linux 平台的开源软件，由 AT&T 剑桥实验室所开发。VNC 的主要功能包括屏幕分享与远端操作，可以用于实现远程协作、远程技术支持等，远程控制能力强大，高效实用。[庄霄, 2009]

VNC 主要由 vncserver 和 vncviewer 两部分组成。其中 vncserver 是服务器，运行在需要被控制的机器上，主要作用是分享所在服务器的屏幕给客户端，并且接收客户端发过来的控制信息控制所在机器。Vncviewer 运行在客户端，使得用户可以在远程控制运行有 vncserver 的机器，当然在有需要的情况下，浏览器可以一定程度上替代 vncviewer 的作用。

VNC 最早的传输协议只会传送原始图像到客户端，也就是实际的点阵图信息，而客户端传递事件给服务器端。这样导致了运行 VNC 会占用大量的带宽，于是采取了各种各样的编码方案[Kaplinsky, 2001]来降低带宽的需要。比如可以预先扫描整个屏幕，找出和前一帧变化了的区间，只发送变化了的内容区间的信息给客户端，这样在服务器端屏幕没有变化或是只有很少变化（鼠标移动）的情况下能极大的减少带宽占用，但是在屏幕有大块区域连续发生变化的情况下（比如一次性打开多个窗口或是拖动窗口），会导致比原先更大的带宽需求。

TeamViewer 是一款用于远程操作、共享桌面、远程传输文件等操作的快捷软件[TeamViewer, 2015]。相比 Windows 远程桌面、QQ 的远程共享等等，TeamViewer 在功能、安全性、可操作性等方面具有优势。

TeamViewer 简单易用，无需配置即可穿越防火墙、路由器和各种代理服务器。在启动 TeamViewer 时，将自动创建一对 1024 位的使用 RSA 加密的非对称密码，TeamViewer 把这个密码发送到服务器以完成注册，并在与其他 TeamViewer 客户端交换信息时会使用该密码进行加密和解密。

每个 TeamViewer 在启动时会根据当前计算机的硬件信息生成一个 ID 号和一个随机密码，因此 ID 对于同一台机器每次都一样，而密码会每次随机生成（也可以设定成固定密码）。ID 会被注册到服务器，当另外一台 TeamViewer 想要对本机进行远程控制时，通过本机 ID 进行定位，并使用密码获取权限，然后就会在两机之间建立起 UDP 或者 TCP 的快速的点对点直接通讯。[冯光午, 2011]

以上的远程控制软件无法直接配合 ITS，因此开发了远程调试子系统。

1.3 本文主要的工作

本文所设计和实现的远程调试子系统，是 ITS 自动化系统云平台的一个子系统。其主要功能在于远程控制和错误调试。

远程控制对于用户来说，是一个可以在自己电脑上对远程 Agent 上的各种被测机器以及控制机器的一个控制系统，用户不需要关心运行了本地 ITS 系统的 Agent 的具体物理位置，也不需要知道 Agent 在云平台上的网络位置，只需要打开浏览器选中想要进行远程控制的 Agent 以及具体的被测机器，就可以在浏览器或者 Java Applet 的窗口上进行控制。远程控制对于整个系统来说，是一个本

地数据流的转发系统，需要将 Agent 上获取到的串口信息、被测机器视频流等输出信息转发到远端，远端的控制信息也要发送到 Agent 上。

错误调试使对于用户来说，是一个可以在修复 bug 的时候重现出错现场，并在现场进行单步调试或者查看变量状态等等的调试系统。对于系统本身来说，是一个错误记录和上传、断点自动插入、自动调用调试模式和虚拟机错误断点判断等等相关功能的集合系统。

结合上述远程控制需求，考虑到多数远程控制的窗口或者界面已经有本地的 Java 实现，所以采用浏览器弹出 Java Applet 窗口的方式实现远程控制界面，这样可以尽可能的复用已有代码，降低出错率。Applet 和 Agent 之间是一套 C/S 架构，Agent 作为服务器监听某个端口，等待 Applet 主动连接，它们之间通过支持 WebSocket 的 Netty 框架进行连接。考虑到要尽可能不影响已有的本地控制的功能，于是采用对输出消息拷贝后转发的方案，并为了实现多用户控制，对某些输出流进行压缩。

1.4 本文的组织结构

本文的组织结构如下：

第一章 引言部分。介绍了项目背景，国内外在该方向的研究现状。

第二章 技术综述。介绍了项目中使用的 WebSocket 协议、Netty 框架和 Java Applet 技术。

第三章 系统需求分析与概要设计。通过需求列表展示了项目的具体需求，通过用例图介绍了分析需求的结果，并以用例描述表的形式重点介绍了其中的几个用例。并对项目进行了概要设计，以组件图介绍整体架构，并介绍了功能上项目系统的几个子模块。

第四章 系统详细设计与实现。在概要设计的基础上，对远程通讯模块、远程串口控制模块、远程 SUT 控制模块、远程 Switcher 控制模块、远程控制台控制模块、错误重现子模块进行了具体描述，以类图展现类关系，并展示了关键代码。

第五章 总结与展望。总结论文期间所做的工作，并就远程调试子系统的未来方向作了进一步展望。

第二章 技术综述

2.1 WebSocket 协议

在本项目中，需要实时把 ITS 控制台的标准输出展现在浏览器的页面上，这样就需要实现浏览器和 Agent 的实时通讯。原有的方案有浏览器采用 AJAX 实现轮询，达到近似实时通讯的效果，但是这样一方面实时性比较差，另一方面也会有许多不必要的开销。也考虑过采用 Http 长连接，但是还是需要不断的发起长连接的请求，服务器也需要对大量的长连接进行管理，最后选择了 Html5 支持的 WebSocket。

WebSocket 是 HTML5 标准中的一个新协议，支持浏览器和服务器之间的全双工通讯，主要目的是在浏览器和服务器之间建立一个双向的有状态连接。此时浏览器和服务器处于实时连接状态，类似使用 TCP Socket 连接，可以随时向对方发送信息。[温照松, 2012]

WebSocket 协议分为两个部分：握手和数据交换。首先客户端和服务器端会通过 Http 请求进行握手，一旦双方的握手信息都发送而且相关的验证成功，便会进入数据交换阶段。[Fette, 2011]

```
GET /chat HTTP/1.4
Host: xxx.cn
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: sRd2IHNoYwdBZdBuuytjZC==
Origin: http://xx.com
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 12
```

图 2.1 浏览器 Http 请求头

WebSocket 连接的握手操作首先由浏览器端或者说是客户端发起，图 2.1 就是浏览器发起握手时首先需要发出的 Http 请求的头信息的一个例子。其中最主要的就是 Sec-WebSocket-Key，这个是浏览器随机生成的长度为 24 的字符序列，是用户验证的主要信息。另外可以看到这是一个 Get 请求，里面告知这是

一个升级连接，是为 WebSocket 做准备的，而且会带有浏览器自身支持的 WebSocket 协议的版本号。

```
HTTP/1.4 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s4pPLDdiTdwQ2k4MszeWfbK+xOo=
Sec-WebSocket-Protocol: chat
```

图 2.2 服务器 Http 请求头

如图 2.2，服务器在接收到客户端的握手请求后，也会返回一个 Http 请求给浏览器进行验证。服务器会将之前接收到的浏览器发送过来的 24 位的 key 进行 SHA-1 哈希编码，将编码后的结果放在头部的 Sec-WebSocket-Accept 中返回给浏览器。浏览器在接收到新的 key 以后，再将之前主动发送出去的 key 也进行一个 SHA-1 哈希编码，要是两者相同，则验证成功，浏览器和服务器间的连接也就建立成功了。[郑强, 2012]

握手之后会建立一个使用 TCP 协议的连接，传递信息的时候采用了 WebSocket 自定义数据格式。每帧进行传输的数据都会带有相关的控制信息，但也保证了带宽不会被浪费。连接在被建立以后会一直维持下去，这样浏览器和服务器之间可以随时相互传输数据，直到有一方主动发出了结束连接的命令或是传输过程中遇到了网络断开等异常。[易仁伟, 2013]

Html5 的 WebSocket 在应用层实现了类似 TCP 的 Socket 的功能，并且还具有支持穿透路由器和防火墙进行通讯、支持进行跨域数据传输、支持使用 cookie 进行认证、支持 Http 的负载均衡以及可以使用直接二进制数据等优点[李代立, 2010]。因此，像远程串口控制、远程 SUT 控制和远程 Switcher 控制这几个目前并不是使用浏览器而是使用 Java Applet，从而可以直接使用 TCP 连接的功能，也决定采用 WebSocket 实现通讯。这样一方面可以进行统一实现，降低错误发生率，降低开发成本，同时可以在 Agent 上只使用一个端口。另一方面，也方便将来将这些 Java Applet 界面移植到浏览器上，并采用 Html5 绘制界面，这样比 Java Applet 更加容易使用，也避免了 Java Applet 需要签名的问题，而且也可以降低对用户电脑资源的占用。

2.2 Netty 框架

项目采用 WebSocket 进行网络通讯，考虑到从头实现 WebSocket 相关功能的服务器和客户端将会花费大量的时间，而且其可靠性、可用性和可扩展性等都得不到保障，于是决定采用支持 WebSocket 协议的、与 ITS 原有部分一样基于 Java 实现的 Netty 框架。

Netty 是一个基于异步的事件驱动模型[Chun, 2013]的编程框架，内部 I/O 使用了 Reactor 设计模式[Schmidt, 1995]，分为一个主线程和若干个 Worker 线程。主线程负责处理远程网络连接相关的一系列工作，包括连接的建立、断开、维护管理等等，当获取到具体的数据需要进行处理时，则将数据分发给某一个 Worker 线程[蒋思佳, 2012]。这样将业务逻辑剥离出去，既方便开发人员使用，使开发人员可以专注于业务逻辑的处理，也提高了高并发下的响应速度。

Netty 是基于 JDK1.4 以后的 NIO 实现的[Hammerton, 2013]，Java 的 NIO 可以高效的处理高并发的情况，NIO 可以使用非阻塞的 Socket。在传统的 Java 的 Socket 处理方式下，一个 Socket 对应一个连接请求，独自占用一个线程，该线程会阻塞在从这个 Socket 获取信息的地方，直至 Socket 得到了从远程获取的信息。这样一方面一个 Socket 一个线程，会导致在高并发的时候产生大量的线程，而线程本身以及线程的创建和消除都需要占用不小的资源，这样会导致性能瓶颈；另一方面，为了能获取信息，Java 虚拟机需要不断在阻塞线程间进行切换，这又是一笔不小的开销。而 NIO 通过采用 Selector 的方法，由 Selector 挑选出完成了某个操作的连接（比如完成打开操作或者读操作），进行处理。这样一方面降低了开销，另一方面也防止在某个连接阻塞的时候干扰到其他连接。

Netty 编程框架由 3 个基本部分组成，分别是 Buffer、Channel 和 Event，其所有的上层特性都是基于这 3 个组件[郑建军, 2013]。

缓冲区(Buffer)是一个可以随机存取或者顺序存取的字节序列，存储整个 Netty 框架中最基本的数据。所有从网络读取到的数据都会先存入缓冲区，在缓冲区中等待被进一步处理的组件获取走；所有要通过网络传出的数据，也会先被放入缓冲区，然后再被发送出去。

通道(Channel)是所有负责读写功能的组件的核心，从功能角度可以划分为服务器监听通道和数据传输通道两类。服务器监听通道是服务器端独有的，会在

服务器启动时刻被创建而且只会被创建一个，并会一直维持工作状态。其主要的功能就是监听其绑定的端口，并在接收到客户端发来的建立连接的请求时进行响应，与客户端建立连接。在一个新的连接被建立之后，框架会新建一个数据传输通道。数据传输通道实质上对应一个远程主机，负责两端的信息传输。

Netty 本身就是一个事件驱动的框架，事件(Event)，或者表示通道中新接收到的数据请求，或者表示通道中已经完成的动作。从传递方向可以分为接收到远程主机传来数据而产生的入站事件，以及需要向远程主机发送数据而产生的出站事件。从类型上又可以分为表示一个通道（或者其子通道）的状态发生了变化的通道（子通道）状态事件、表示一个通道已经处于闲置状态的闲置状态事件、表示通道遇到了异常情况的异常事件、表示新接收或者发送完成了一个消息的消息事件[周乐钦, 2013]。

另外，Netty 还具有以下优点：提供接口统一管理阻塞性和非阻塞行的连接、拥有可以方便进行扩展的线程模型、拥有高效且灵活的事件驱动模型、支持多种网络协议（Http、TCP、UDP、FTP 以及子系统最终使用的 WebSocket）以及加密算法（完全支持 SSL/TLS），适用于软件的快速开发[金志国, 2014]。

2.3 Java Applet 技术

子系统的远程控制模块需要在用户运行浏览器的时候打开远程控制界面，一开始考虑使用 JavaScript 和 Html5 在浏览器上直接实现界面，但是发现开发难度较大、出错可能性较高。于是考虑采用 Java Applet 实现界面，这样可以直接复用已有的控制界面的 JFrame 代码、已有监听程序的线程代码以及已有的 Netty 的网络通讯代码。

Java Applet 是 Java 的小程序，而 Java 是由 SUN 公司开发的基于 C++的编程语言，具有平台可移植性、支持多线程、面向对象等等特点。Java Applet 可以用来创造动态交互页面，它由一段嵌入 Web 页面的 Html 标记或者一段 JavaScript 代码调用执行。

Java Applet 程序会被编译成字节码并打包成一个 JAR 包存储在服务器端，当用户的浏览器需要运行 Applet 时，会自动下载该 JAR 包到浏览器端，并使用浏览器中的 Java 虚拟机解释执行。该 JAR 包运行时依赖的是浏览器所在计算机

安装的 Java 运行时环境，这样逻辑等相关代码是由浏览器运行，降低了服务器压力。

Java Applet 是浏览器通过网络下载下来代码直接运行，因此有可能存在安全隐患，比如下载的 Applet 含有恶意代码，这样有可能会破坏计算机本地的文件或者在本地安装病毒或者木马，也有可能会去自动访问一些恶意或者会造成消费的 URL，给用户带来损失。所以，与其他 Java 程序不一样，浏览器运行的 Java Applet 有严格的强制安全检查，首先在编译的时候进行安全性检测，同时在具体解释运行的时候会对字节码进行严格的安全检查。这种安全机制可以阻止 Java Applet 对本地计算机资源的访问，也会限制其对外的网络通信，这样可以尽可能的降低运行 Java Applet 带来的风险[雷明剑, 2007]。

在本子系统的远程控制模块，JAR 包是被存放在服务器端。这样，用户在使用浏览器打开远程控制界面的时候，其实是从服务器的 IP 下载了 JAR 包，而运行的 Java Applet 又需要直接建立和另外一个 IP（Agent 地址）的 WebSocket 通讯。此时由于 Java Applet 的安全限制，这个行为会被阻止。因此，需要使用 keystore 工具来生成一个数字签名，并 jarsigner 将 Java Applet 的 JAR 包签名，这样才可以实行类似跨域访问的效果。

2.4 本章小结

这一章主要介绍了开发本系统所使用的相关技术和框架。首先介绍了 Agent 和 Applet 端或者浏览器端进行通讯所使用的 WebSocket 协议，然后介绍了实现 WebSocket 网络通讯所采用的 Netty 框架，最后介绍了实现远程界面能够展示出来采用的 Java Applet 技术。

第三章 系统需求分析与概要设计

3.1 远程调试子系统概述

ITS（Intelligent Test System 自动化测试系统）是一款运行于 windows 平台上的自动化测试工具。它的云平台是为了整合利用空闲的运行了 ITS 的 Agent 而开发出来的一套程序。

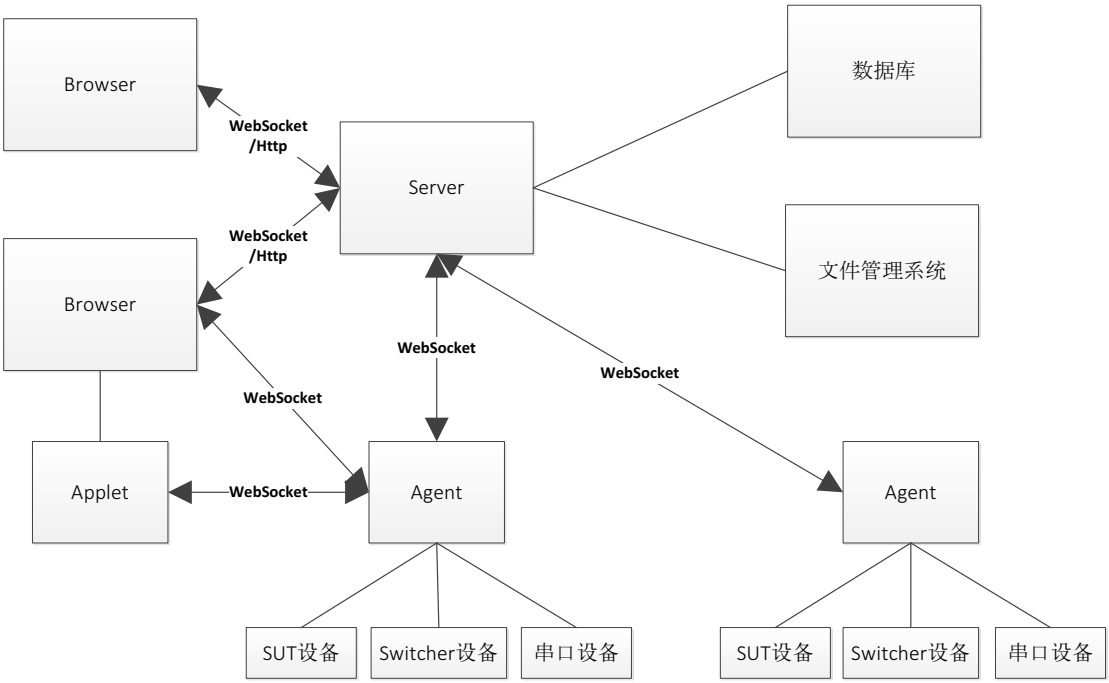

图 3.1 ITS 云平台系统总体结构

如图 3.1，一套 ITS 的云平台系统有一个进行集中管理的 Server 服务器。在该服务器上运行着整个云平台的数据库模块和文件管理系统，其中数据库组件记录并管理着所有的用户信息、任务信息、曾经连接上的 Agent 的信息以及任务运行以后留存下来的日志的信息，而文件管理系统是一个和数据库模块配合使用的系统，主要进行的是对所有用户上传上来的脚本的管理，以及运行脚本有所依赖的所有的二进制文件的存储和管理。

运行了 ITS 测试系统的 Agent 连接着各种测试所需的硬件设备（Switcher 设备）和被测机器（包括 SUT 设备和串口设备），然后 Agent 会在用户选择了进入云平台后，根据用户输入的 IP 地址寻找到 Server。连接上 Server 以后会将

自己的信息注册进 Server，包括 Agent 的 mac 地址（这个是一台 Agent 在 Server 上的唯一的不变的表示）和 IP 地址（方便 Applet 连接 Server）。并且会在自己状态有变化的发送消息通知 Server，变化的消息包括有 ITS 的状态（包括空闲、下载中、运行任务中等等）和所接的各种设备的状态，这里设备的状态是指比较笼统的设备的状态，包括串口设备的 TCP 打开和关闭、SUT 设备的打开关闭以及分辨率等等，这些信息会被远程调试子系统用来打开远程控制的各种窗口。

用户通过浏览器访问 Server，查看和管理用户信息，查看、上传或者下载脚本以及相关的二进制文件，新建、修改或者删除任务、查看任务的日志信息以及曾经连上和正在连上的 Agent 信息。当用户运行一个任务或者重现某个日志的错误的时候，Server 会将相关的脚本和二进制文件打包发送到 Agent 上去，Agent 在下载完相关的文件之后会添加本地任务并运行，运行的日志结果会发送回来并存储在 Server 上，如果出现错误，重现错误所需的信息也会一并被发送回 Server。

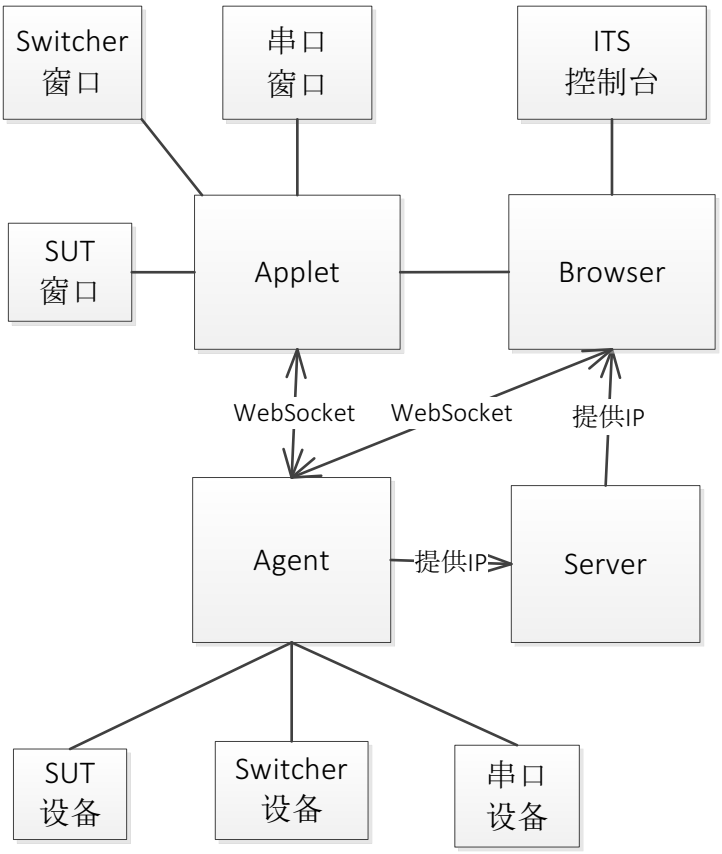

图 3.2 远程调试子系统总体结构

同时，用户可能希望可以在远程实现对 Agent 以及其上所连接的各种设备进行控制，从而远程调试脚本，因此开发了远程调试子系统。如图 3.2，考虑到这些实现远程控制的输入流和输出流会非常大而且频繁，要是都通过 Server 进行转发的话，会极大的占用 Server 的带宽、内存和 CPU 资源，很容易成为整个系统的瓶颈，甚至影响到其他功能的正常运作。因此，在实现远程控制的时候，子系统采用了浏览器以及其上的 Applet 和 Agent 直接通讯的方案。

为了实现这种直接通讯，浏览器需要先从 Server 获取到目标 Agent 的 IP 地址，然后从浏览器（远程控制 ITS 控制台）或者从 Applet（远程控制 SUT、Switcher 或者串口设备）建立直接到 Agent 的 WebSocket 连接。这样，远程控制所需的大量的数据就会绕过 Server，一方面降低的 Server 的负担，另一方面又提高了远程控制的效率。

3.2 远程调试子系统需求分析

3.2.1 远程调试子系统功能需求

在 ITS 测试系统里，每台运行了 ITS 测试系统客户端的 Agent 可以通过多个串口同时和多台串口设备相连。这些串口设备的输出信息以流的形式被 ITS 客户端逐字节地从串口读取出来，经过解析以窗口页面的形式展现在 Agent 上。然后用户可以在窗口直接进行键盘操作，这些操作会被转换为串口控制信息通过串口发送到相应的串口设备上去，达到对串口设备的操控。

同时，Agent 还可以通过 HDMI 视频采集设备和一台被测的有高清视屏输出的设备（ITS 系统里称呼该设备为 SUT 设备）相连，采集到一帧一帧的图片信息，这些信息直接显示在了一个窗口界面上。然后用户可以在该窗口进行一般的鼠标键盘操作，甚至更高级的 Win、Alt、Ctrl 等功能键及其组合的操作，这些操作的信息会发送到一个相连的鼠标键盘模拟器上，然后将模拟后的鼠标键盘信号发送到 SUT 设备上。

另外，Agent 会和一个使用 Usb 接口的 Switcher 设备相连，ITS 会实时监控这个设备的状态，并通过一个控制面板展现连接状态以及 Switcher 设备上的各个开关的状态，同时还可以直接通过这个面板控制 Switcher 设备上对外提供电

源的开关以及一些控制线的接通与否，从而达到间接控制被测机器电源状况和硬件开关的目的。

最后，运行在 Agent 上的 ITS 系统本身还有标准的控制台输出，这些输出可能是正常脚本运行时刻的运行日志信息、Agent 系统本身的错误日志信息、debug 模块的调试信息以及本地的标准输入流信息，这些输出会显示在 ITS 程序的控制台面板上。同时在该面板上方的输入框处用户可以输入内容，这些内容会进入系统的标准输入，被希望获取用户输入的组件得到，这些组件可能是脚本运行组件或者 debug 组件。

远程调试子系统的远程控制模块，就是要通过以上的 4 个控制模块实现远程控制，使得用户可以在自己的机器上，通过网络远程控制连接在 Agent 上的各种被测试机器，其主要的需求如表 3.1 所示。

表 3.1 远程控制模块需求列表

需求 ID	需求名称	需求描述
R1	远程串口控制	用户能够在远端浏览器通过打开的 Java Applet 窗口，获取到串口机器的流输出。这些输出信息流以等效 VT100 终端的方式显示成一个可视化窗口界面给用户。同时用户在这个 Applet 窗口上的键盘操作，也能够落实到连接在 Agent 上的串口机器。
R2	远程 TCP 转发标准串口信息	用户通过浏览 ITS 的服务器获取到目标 Agent 的 IP 地址和目标串口的 TCP 端口地址，直接在远程通过 VT100 标准终端的 TCP 连接访问 Agent 上的串口机器。
R3	远程 SUT 控制	用户能够在远端浏览器通过打开 Java Applet 窗口，查看从 Agent 上采集到的高清视频流。这些视频流的每一帧图片保持与 Agent 本地视频显示窗口的图片一致，并保持同步。同时用户在该窗口的鼠标键盘操作也能够实时控制在 Agent 上的被测机器。
R4	远程 Switcher 控制	用户通过远程 Java Applet 窗口实时查看 Agent 上的 Switcher 设备本身的连接状态和各个开关以及控制线的状态，并直接在窗口对 Switcher 设备的开关进行控制。

R5	远程控制 ITS 控制台	用户能够在远端浏览器直接查看到 Agent 上的 ITS 控制台的标准输出，这些输出信息会实时逐行显示。同时用户在浏览器内输入的信息，也会被转发到 Agent 上的 ITS 系统。

远程调试子系统除了以上的远程控制模块外，还有一个错误现场重现并进行 debug 的调试模块。

在 ITS 测试系统里，除了基本的运行测试脚本的 run 模式，还有通过预先设置好的断点对脚本进行单步调试的 debug 模式。在远程调试子系统里，借用并修改了这个 debug 模式。这样首先在运行 Server 推送过来的脚本的过程中，记录发生的错误。然后，当用户希望重现错误现场进行调试的时候，脚本会自动运行到出错的那一行，并进入可以单步调试的 debug 模式，该模式的输入输出就借助远程控制模块的 ITS 控制台远程控制子模块。同时，用户还可以打开远程的串口控制或者 SUT 控制，查看被测系统实时的运行情况，从而实现远程调试这个目的。其中主要的需求如表 3.2

表 3.2 调试模块需求列表

需求 ID	需求名称	需求描述
R6	错误记录	当用户的脚本在 Agent 运行过程中出现了错误的时候，系统能够自动记录错误发生的代码位置，并将这个信息存储到 Server 端。
R7	错误重现	当用户选择重现错误进行调试时，服务器能够自动匹配到合适的机器运行该脚本，并在运行到错误发生的那一行时停下，并进入可以进行单步调试的 debug 模式。

3.2.2 远程调试子系统非功能需求

表 3.3 系统非功能需求列表

时间特性	用户任何串口和控制台操作的响应时间不大于 0.5 second
	在 10M 局域网内用户任何 SUT 操作的响应时间不大于 1 second
负载	支持 2 位以上用户同时远程控制

3.2.3 远程调试子系统用例图

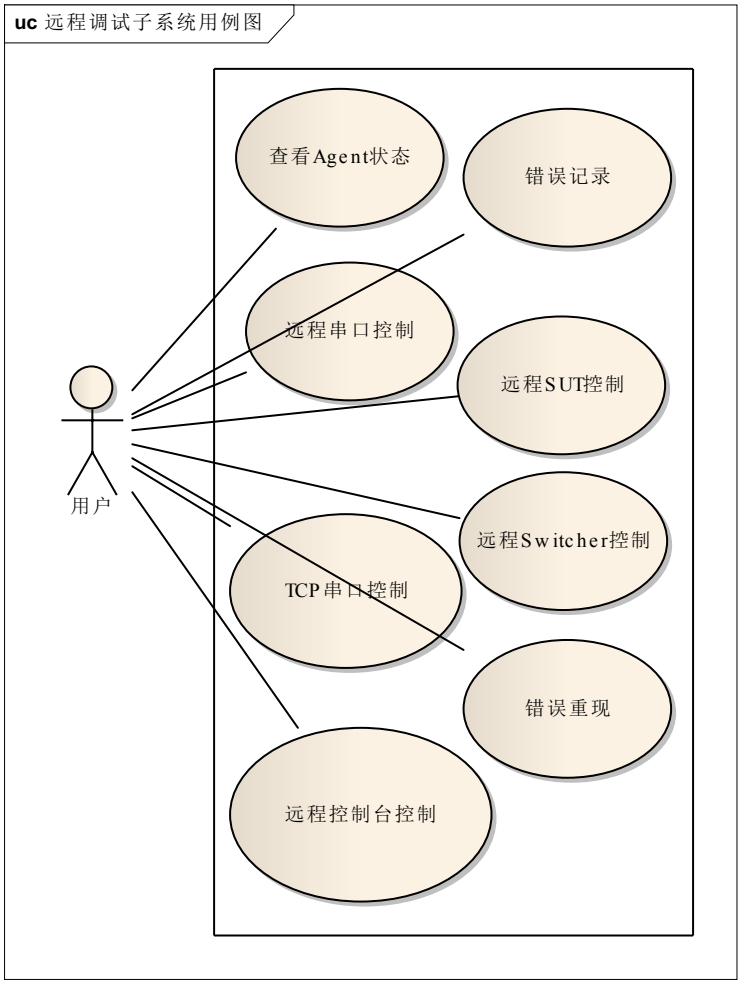

图 3.3 远程调试子系统用例图

根据相关功能性需求，可以分析得到如图 3.3 的 8 个用例，分别是远程控制相关的查看状态、远程串口控制、远程 SUT 控制、远程 Switcher 控制以及远程控制台控制 6 个用例，以及调试相关的错误重现和错误记录 2 个用例。

3.2.4 远程调试子系统用例描述

查看 Agent 状态，是远程控制模块的基本用例，描述的是用户在浏览器页面上查看 Agent 的状态，主要是远程窗口基本信息，具体的情况如表 3.4。

表 3.4 查看 Agent 状态用例描述表

ID	UC1
名称	查看 Agent 状态
用例属性	创建者：蒋宇阳
参与者	参与者：普通用户

	目的：查看某 Agent 上可以打开的远程窗口
描述	用户通过浏览器查看某 Agent 上的可用控制窗口信息。
优先级	高
触发条件	用户点击选择具体的 Agent。
前置条件	用户进入 Agent 列表。
后置条件	进入 Agent 页面，可以打开控制窗口。
正常流程	1.进入 Agent 列表 2.通过搜索筛选 Agent，选中其中一个进入状态页面 3.在该页面查看 Agent 的 IP 地址信息 4.在该页面查看 Agent 上打开的串口信息，可以查看到所有的串口信息，包括名字、串口号、对应的网络端口号以及此时该串口的状态。选中一个打开的串口点击即可打开串口远程控制窗口 5.在该页面查看 Agent 上打开的 SUT 信息，可以查看到 SUT 是否打开、获取到的视频信息的分辨率等等，点击 SUT 按钮即可打开 SUT 远程控制窗口
异常流程	4a.若是用户希望使用其他的标准的 VT100 终端进行串口控制，可以通过该页面提供的 Agent 的 IP 地址加上串口对应的端口号，直接在其他的终端新建一个 TCP 的连接，一样可以进行远程控制 4b.若串口设备在 Agent 上并没有被打开，那么用户也无法在 Agent 状态页面打开远程控制器

远程 SUT 控制，是远程控制模块的基本用例之一，描述的是用户在远程通过 Applet 操纵 Agent 上的 SUT 机器，具体的情况如表 3.5。远程串口控制和远程 Switcher 控制也是类似的基本用例，由于其基本操作被远程 SUT 控制所涵盖，因此不另建用例描述表。

表 3.5 远程 SUT 控制用例描述表

ID	UC2
名称	远程 SUT 控制
用例属性	创建者：蒋宇阳
参与者	主参与者：普通用户 目的：远程控制 SUT 机器
描述	用户通过浏览器在远端打开 SUT 控制界面对 SUT 机器进行操作。
优先级	高
触发条件	用户点击远程 SUT 控制打开按钮。
前置条件	用户进入 Agent 状态页面。

后置条件	无
正常流程	1.点击远程 SUT 控制按钮，Applet 窗口自动打开，获取到远程 SUT 的桌面并显示出来 2.在 Applet 窗口上进行的鼠标键盘操作会控制远程 SUT 设备，相应的界面的变化也会返回到窗口 3.点击窗口上的“窗口按钮”或者“Ctrl+Alt+Del”按钮等，将 Win 或者 Ctrl+Alt+Del 组合这样的会被操作系统捕获并响应的键盘事件发送到被测机器上 4.打开 Applet 控制台实时查看当前远程控制所占用的带宽，以及远程控制采集到的桌面的 FPS（每秒传输帧数） 5.点击 Applet 上的关闭按钮，主动断开与 Agent 的连接，关闭显示的界面。
异常流程	1.连接失败，弹出错误信息框告知用户

TCP 串口控制，是远程控制模块的一个附加用例，描述的是用户在远程通过标准的类似 VT100 的终端，以 TCP 的方式控制 Agent 上的 SUT 机器，具体的情况如表 3.6。

表 3.6 TCP 串口控制用例描述表

ID	UC3
名称	TCP 串口控制
用例属性	创建者：蒋宇阳
参与者	主参与者：普通用户 目的：远程控制串口机器
描述	用户通过标准的 VT100 终端在远端控制被测串口机器。
优先级	中
触发条件	用户进入 Agent 状态页面。
前置条件	用户进入 Agent 状态页面。
后置条件	用户可以使用 VT100 终端进行控制
正常流程	1.用户进入 Agent 状态页面 2.用户查看到 Agent 的 IP 地址 3.用户查看到希望远程控制的串口机器在 Agent 上的对外的端口号 4.用户打开一个 VT100 终端，以 TCP 形式新建连接，采取 Agent 的 IP 加上相应的端口号，连接上被测机器 5.用户控制被测机器，就如图采用打开串口的方式进行控制
异常流程	无

远程控制台控制，是远程控制模块的一个基本用例，描述的是用户在远程直接查看运行在 Agent 上的 ITS 系统的控制台输出，同时可以直接将控制信息输入到控制台，具体的情况如表 3.7。

表 3.7 远程控制台控制用例描述表

ID	UC4
名称	远程控制台控制
用例属性	创建者：蒋宇阳
参与者	主参与者：普通用户 目的：远程控制 ITS 控制台
描述	用户通过浏览器对 Agent 上的 ITS 的控制台进行控制。
优先级	高
触发条件	用户进入控制台页面。
前置条件	用户进入 Agent 状态页面或者任务状态页面
后置条件	无
正常流程	1.用户进入 Agent 状态页面点击控制台详情，或者进入任务状态页面 2.浏览器上实时从 Agent 的 ITS 系统一条一条获取到的控制台消息 3.用户在浏览器输入框输入内容，按下发送按钮，输入的内容会被发送到 ITS。如果有模块正在等待控制台输入，就会获取到这条信息。无论有没有模块在等待控制台输入，这条信息都会被打印到控制台输出框
异常流程	无

错误重现，是调试模块的最核心用例，描述的是系统如何将错误现场回复并停止在那里，具体的情况如表 3.8。

表 3.8 错误重现用例描述表

ID	UC5
名称	错误重现
用例属性	创建者：蒋宇阳
参与者	主参与者：ITS 系统 目的：重现出错现场以便用户调试
描述	Agent 上的 ITS 系统重现错误场景以便 debug。
优先级	高
触发条件	用户点击重现错误按钮。
前置条件	用户进入任务运行日志界面

后置条件	用户可以进行 debug
正常流程	1.用户在错误日志界面点击错误重现按钮 2.服务器寻找到出错的脚本，并从数据库里取出运行该脚本需要的环境，以及出错时刻记录下来的错误行数 3.服务器寻找目前符合脚本运行环境的空闲的 Agent，把脚本和错误行数发送过去，并以重新模式运行 4.Agent 上的 ITS 运行脚本，停留在出错的代码处，等待用户输入 debug 命令
异常流程	3.找不到符合运行环境需求的 Agent 的话，就通知用户需要等待，并将之放置到任务等待队列里去

3.3 系统总体设计与模块设计

3.3.1 模块设计

程序从功能上来说可以划分两个主要的模块：远程控制相关的模块和实现调试的模块。

调试模块是相对比较简单的一个模块，负责的是与调试错误本身相关的职责，分为错误记录子模块和错误重现子模块两个子模块。其中错误记录子模块主要负责在运行脚本遇到错误的时候，记录下此时出现的错误在脚本中的位置，并在脚本结束的时候，将所有记录下来的错误的位置统一发送到服务器。然后服务器会记录将这些错误位置记录再此次运行结果日志下，以便错误重现子模块使用这些记录下来的位置。

错误重现子模块则负责在用户想要调试错误的时候，根据运行日志选择一个与当时运行环境一致且处于空闲状态下的 Agent，将出错的脚本自动推送到该 Agent 上并运行。当出错脚本运行到之前记录的错误位置的时候，能暂停运行并进入 debug 模式，使得用户可以通过控制台进行单步调试并查看此时运行堆栈里的一些变量的信息。

远程控制模块主要负责支持用户从远程浏览器端以及通过浏览器打开的 Applet 界面，对连接到本地 Agent 上的被测机器、控制机以及运行在 Agent 上的 ITS 的控制台进行远程控制，从功能可以进一步细分为远程串口控制、远程 SUT 控制、远程 Switcher 控制、远程控制台控制以及远程通讯这 5 个模块，而

每个模块从所属位置又可以分为服务器端的 Agent 子模块和客户端的 Applet 子模块或者浏览器子模块。

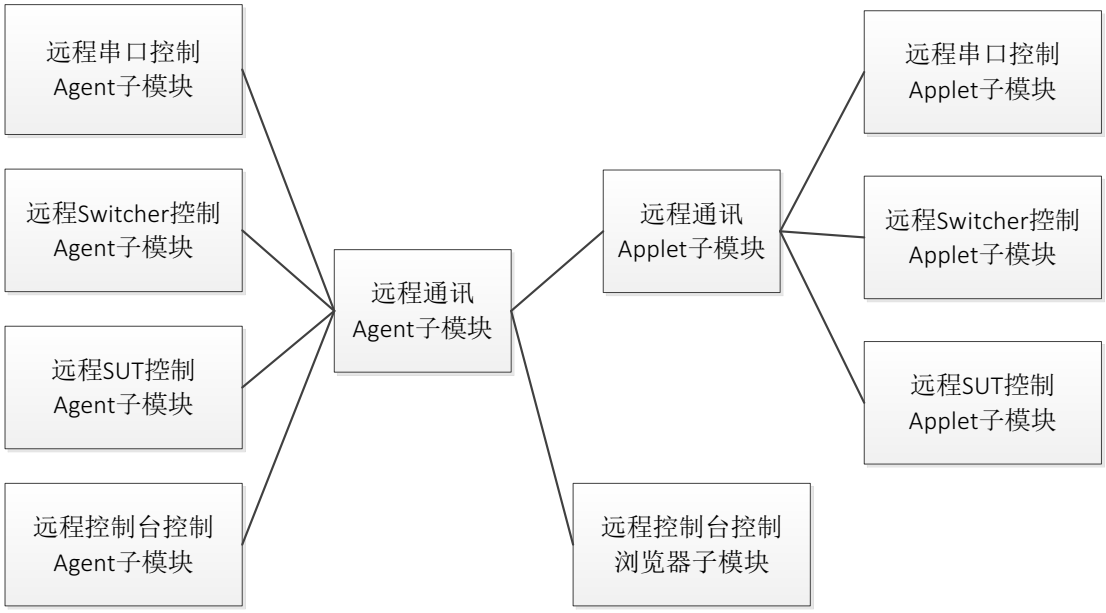

图 3.4 远程控制模块图

如图 3.4，4 个远程控制的 Agent 子模块都运作在 Agent 上的 ITS 系统里，主要负责从各自的本地控制组件里获取相应控制设备的输出流，将输出流的拷贝或者引用发送到远程通讯 Agent 子模块。远程通讯 Agent 子模块负责统一管理所有连接到 Agent 的进行远程控制的连接，无论是来自 Applet 或者浏览器的 WebSocket 连接或是其他标准 VT100 设备的 TCP 连接。并在接收到来自控制模块的输出信息时，选择相应的连接发送输出信息。同时通讯模块在接收到远程控制信息后会将之发送到相应的 Agent 控制子模块，再由这些子模块真正操作相应的设备。

串口、SUT 和 Switcher 远程控制的窗口都是使用 Java Applet 技术实现的，所以这 3 个远程控制有 Applet 子模块，他们会将获取到的转发过来的相关设备的输出信息流进行解析，解析的结果显示在相关的窗口界面上。同时，用户在这些界面上的操作也会被这些模块捕获并生成相应的控制信息，再发送到远程通讯 Applet 子模块。远程通讯 Applet 子模块是运行在 Applet 上的通讯模块，负责主动连接上 Agent，并转发使用 Applet 实现远程窗口的远程控制模块相关的数据。

考虑到控制台本身需要在好几个云平台的 Web 界面进行展现，因此远程控制台使用的是浏览器子模块，直接使用 JavaScript 相关的库函数实现与 Agent

的简单的 WebSocket 连接，实现控制台信息的实时显示与控制。

3.3.2 组件设计

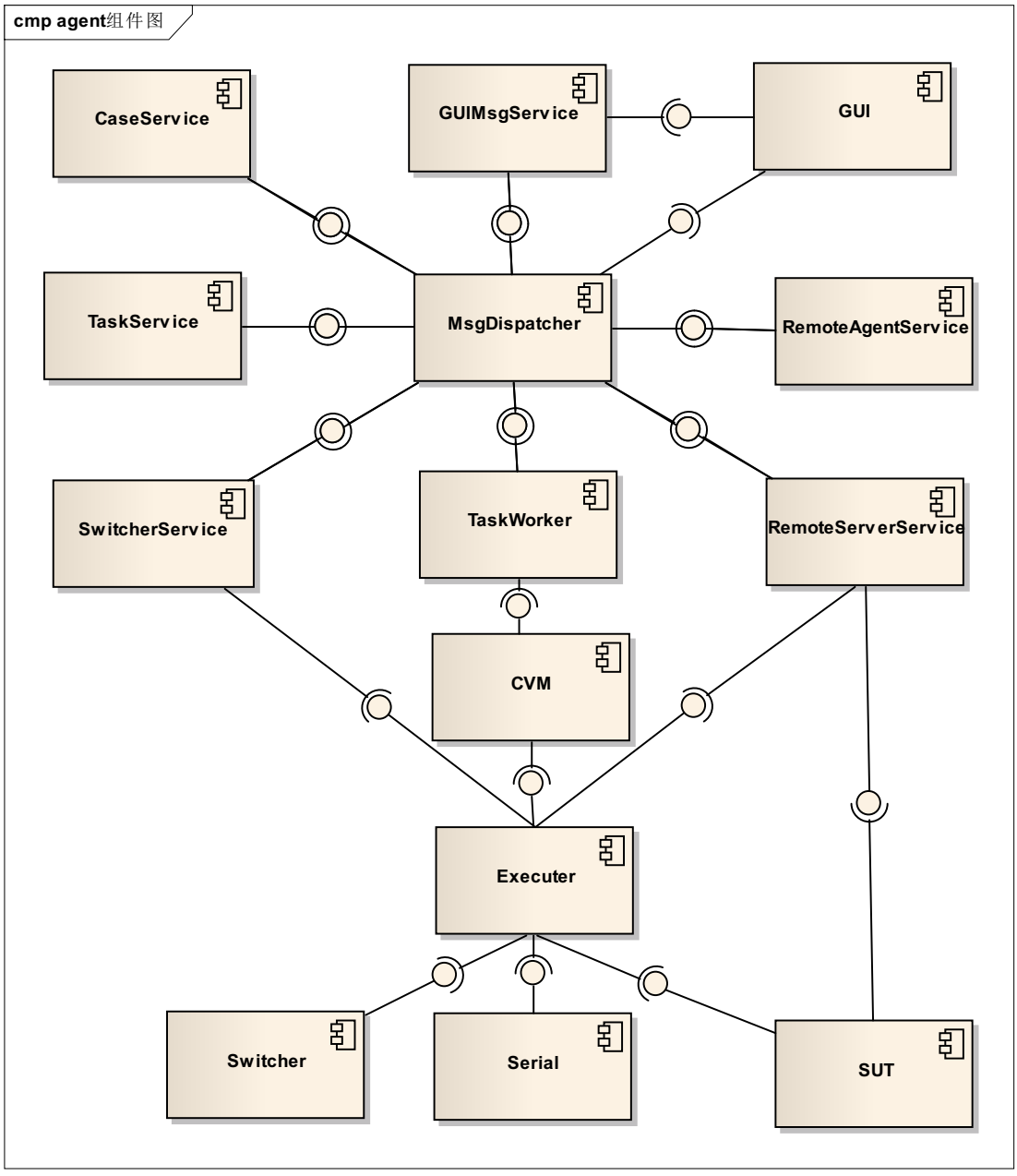

图 3.5 Agent 组件图

图 3.5 是运行在 Agent 上的 ITS 系统的组件图[李波,2014]，主要可以分为 Service 和 Executer 两块，然后还有一些零散的组件，比如负责各种页面绘制以及和用户进行直接交互的 GUI 组件、负责调用编译器编译脚本然后调用解释器运行编译结果的 CVM 模块。

Service 是负责各个具体业务的组件，其中：CaseService 负责脚本（比如重现错误需要的脚本）的导入、检查以及管理；TaskService 负责将相关的 Case 按照用户的设置（比如重现错误时从 Agent 得到的设置）组装成一个个任务，并在需要的时候对其进行修改或者删除；Switcher Service 可以通过 Executer 向连接到 Agent 的 Switcher 发送信息，这样一方面它可以实时监控 Switcher 的状态，另一方面也会在需要的时候控制 Switcher 上面的开关的状态，是实现本地以及远程 Switcher 控制的核心组件；TaskWorker 是运行任务的主要负责者，在重新错误时它会接收到需要运行任务的消息，会先做好任务运行的准备工作，然后调用 CVM 组件去运行脚本，并在 CVM 非正常退出时打印信息，最后会在任务运行结束之后通知有所需要的 Service，重新错误包括调试也是其相关的业务；GUIMsgService 则负责在接收到其他 Service 发送过来的信息（Switcher 状态信息）以后，对 GUI 进行相应的重新绘制与刷新；RemoteAgentService 负责的是 Agent 与 Server 之间的通讯，同时也负责在接收到 Server 的各种 Json 的消息之后进行解析并转发到相关的 Service（比如需要重现一个错误），同时在从其他 Service 获得相关的消息以后也会发送给 Server（比如记录的错误信息）；RemoteServerService 负责的是 Agent 和远程 WebSocket 的通讯，可能连接的是一个浏览器（远程控制 ITS 控制台时），也可能连接的是一个 Applet（远程 SUT、Switcher、串口控制），该组件的具体内容在论文之后部分会详细讲述。

这些 Service 都是以单例的线程的形式存在于系统中，并且都会在程序的 main 函数里被逐一启动。Service 都会有一个自己的消息队列，在整个运行过程中，Service 会不断的从队列中获取最新的消息，解析并且运行。当有其他的组件希望向某个 Service 发送消息时，绝大多数情况下都是要通过一个 MsgDispatcher 的 Service，它负责从各个组件获取各种各样的消息，然后根据消息的类型选择合适的一个或者多个接收者，将消息发送过去。这样可以对消息的发送有一个统一的管理，也方便在某些情况下对消息的流向进行控制。当然，可能会存在大量持续的从一个组件向某个 Service 发送消息，这种情况下也可以采用直接向目标 Service 的消息队里加消息的方式，当然这样会极大的增加组件直接的耦合度，不方便功能的扩展。

Executer 则是和远程调试相关的模块紧密相关的，它负责在对 Agent 上连接的串口设备（通过 Serial 组件）、Switcher 设备（通过 Switcher 组件）以及 SUT 设备（通过 SUT 组件）进行直接的交互，包括获取其输出值，并对其进行控制等等，其各个组件的具体内容会在第四章具体展开。

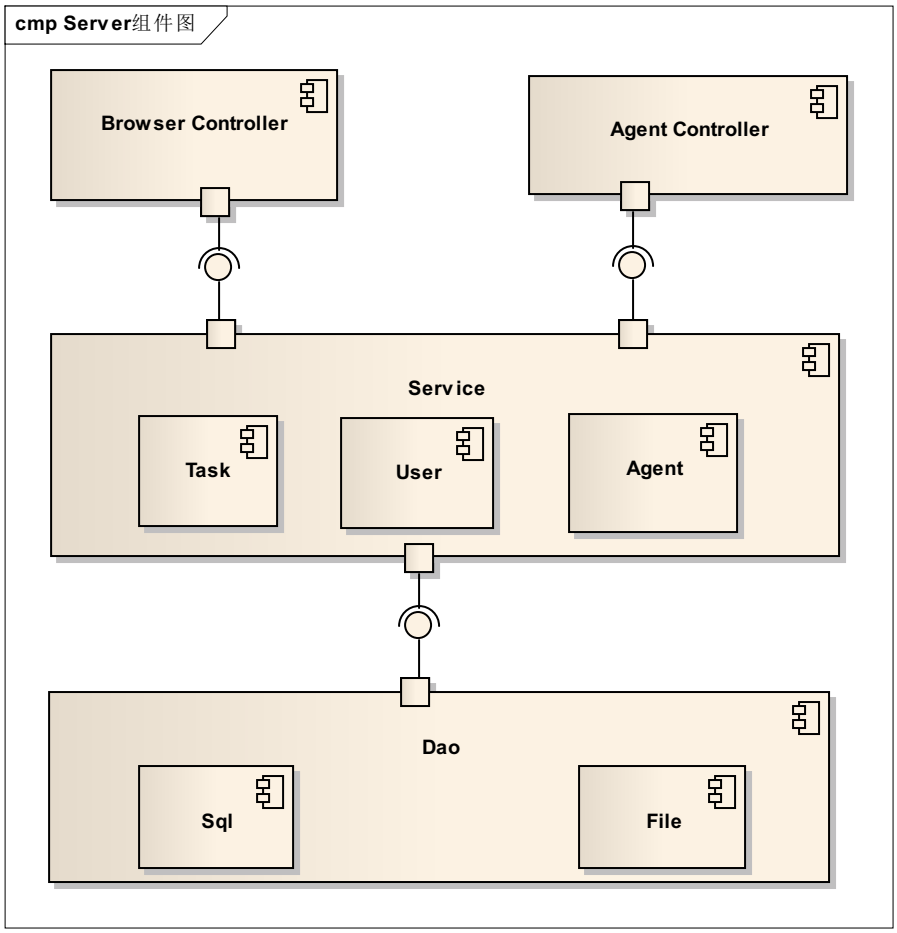

图 3.6 Server 组件图

如图3.6，Server在结构上按层可以划分为3层，也就是Controller层、Service层和 Dao 层。

Controller 层有两个完全不一样的组件，分别是对应浏览器和 Agent 两个方向上想需求。其中 Browser Controller 负责监听浏览器发过来的请求，是 B/S 架构的 Server 部分使用的是整合了对 WebSocket 支持的 Spring MVC 框架，浏览器发送过来无论是 Http 形式的基本请求（查询 Agent 的 IP 地址），还是一个对 WebSocket 长连接的请求（查看 Agent 的实时状态），都会有相关的 Controller 里的方法进行处理，而且这个 Controller 和浏览器展示的 jsp 页面是紧密相关的。

而 Agent Controller 主要是处理各个 Agent 发送过来的数据包，是 C/S 架构的 Server 部分。首先采用了 Netty 框架进行数据传输，而对获得到的 Json 格式的数据（比如 Agent 状态信息）初步解析之后，会知道是哪一功能方向的数据包，这样再由一个简单工厂的类来选择对应的 Controller 进行进一步的数据处理。

Service 层提供了各种业务逻辑的处理，同时也保存了一些不需要持久化存储公共数据（比如 Agent 上的各种设备的实时状态）。具体可以划分为处理用户信息的 User 组件、查看和处理 Agent 信息的 Agent 组件、管理和运行任务的 Task 组件。Controller 层通过注解配置了相应 Service 的实例，Browser Controller 和 Agent Controller 里的 Controller 是有可能配置到同一个实例的，这样就可以实现两个 Controller 的数据共享，也就实现了用户查看和控制 Agent 的一种手段。比如 Agent 上新开了串口窗口，那么相应的信息会以 Json 包的形式发送到 Server，然后某个 Service 实例里的 Agent 状态数据会发生变化，此时查看这个 Agent 详细信息的页面会刷新到最新的状态。

Dao 层管理的是各种持久化的数据，主要就是 SQL 组件和文件管理系统。任务的信息、日志的信息、Agent 的 IP 等等都存储在数据库中，包括远程调试子系统涉及到的重现错误所需要的信息，由 SQL 组件负责对数据库的增删改查。文件管理系统一方面是对文件本身的各种处理，包括获取文件的 MD5 值、监视文件更新变化、打包脚本文件、扫描脚本文件获得各个文件之间的依赖关系等等，另一方面是对文件存储位置的管理，包括本地存储地址和对外 URL 的映射、从远程共享目录下载最新的二进制文件等等。

3.4 本章小结

本章主要首先通过结构图对子系统所依赖的云平台以及子系统本身进行了说明，然后以需求列表的形式展现了子系统的主要功能性需求和非功能性需求，再以用例图[谭云杰, 2012]的形式对需求进行了归纳分析，得出了一系列用例，并对其中主要的用例采用了用例描述表的形式详细介绍。最后通过模块图和组件图对整个系统进行了概要设计，并初步介绍了按照功能划分出来的几个模块以及它们之间的关系，为下一章按模块进行详细设计和实现做准备。

第四章 系统详细设计与实现

4.1 模块综述

在功能上，远程调试子系统可以分为远程控制模块和调试模块两部分，远程控制模块具体又细化分为远程串口控制模块、远程 SUT 控制模块、远程控制台控制模块、远程 Switcher 控制模块和远程通讯模块，调试模块可以划分为错误记录子模块和错误重现子模块。本章节选取其中比较重要的 4 个远程控制模块和错误重现子模块，对其详细设计和实现展开论述。

4.2 远程通讯模块

4.2.1 远程通讯模块介绍

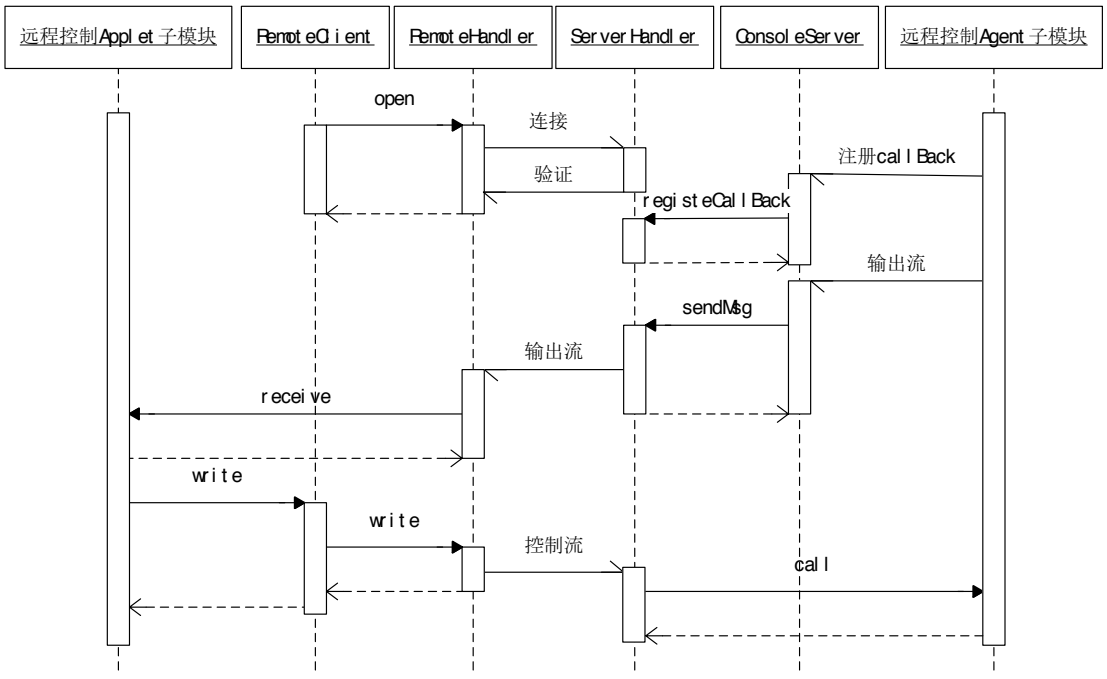

图 4.1 远程通讯模块时序图

远程通讯模块负责 Agent 和 Applet 端之间的通讯，分为运行在 Agent 端的 Server 和运行在 Applet 端的 Client 两部分。如图 4.1，各个远程控制模块的 Agent

端的子模块会在启动的时候将各自的回调实例注册进通讯模块，并在运行过程中不断的将输出流转发到通讯模块，通讯模块会将数据发送到 Applet 端相应的接收者。同样，Applet 端的操作也会变成控制流通过通讯模块最终发送到各个远程控制模块 Agent 端的回调对象中。

4.2.2 远程通讯 Agent 子模块详细设计

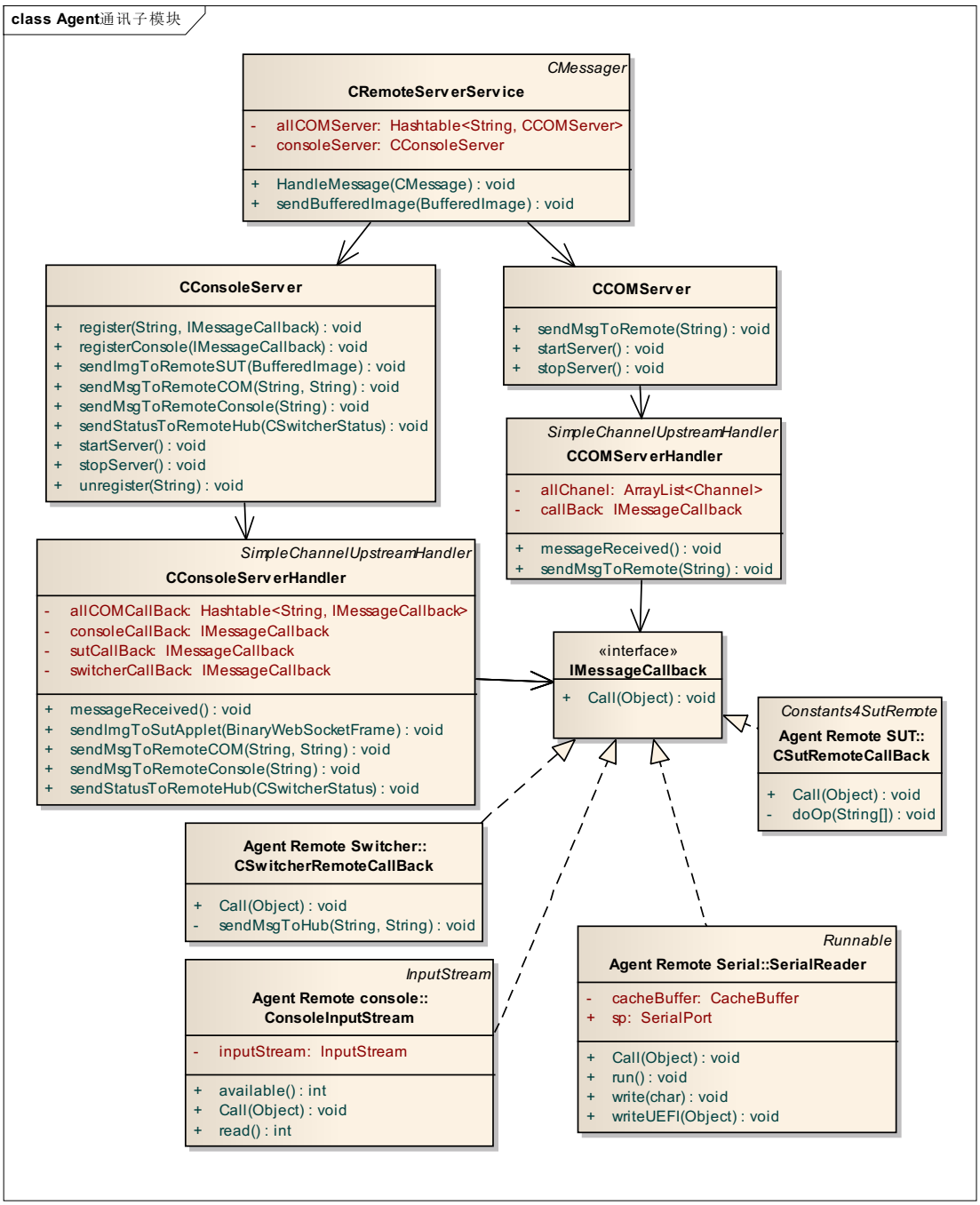

图 4.2 远程通讯 Agent 子模块类图

图 4.2 是远程通讯模块在 Agent 端的主要类图，可以看到该子模块对外的主要接口都是在类 CRemoteServerService 上，与之相关的几个类也就是组件图里介绍的 RemoteServerService 组件。CRemoteServerService 会从 Agent 部分的串口子模块、SUT 子模块、Switcher 子模块以及控制台子模块获取需要被转发出去的信息，解析然后调用相应的 Server 类的函数。

CConsoleServer 和 CCOMServer 就是模块内的负责与 Applet 或者浏览器通讯的 Server 功能的类，它们都是通过 Netty 框架实现了服务器的功能。区别在于 CConsoleServer 有且仅有一个实例，它所监听的端口就是 Agent 对外的唯一的 WebSocket 端口，它同时需要与远程的串口、SUT、Switcher 以及控制台进行通讯；而 CCOMServer 可能有多个或者 0 个实例，其数目与目前 Agent 上的 ITS 打开的串口窗口数目一致，每个实例会监视各自的端口 TCP 通讯，只会负责与标准的 VT100 串口终端通讯。

IMessageCallback 是一个回调的接口，它可以把自己实例的引用附着在消息中，从而实现了在 Service 之间的通讯，比如远程串口模块和远程 ITS 控制台模块。这些实现了该接口的类都在各自模块中负责对远程 Applet 发送过来的具体的信息进行解析并处理，从而对本地机器进行控制。

Handler 接口是 Netty 框架中自带的，是用来具体解析处理从网络上获取到的信息的类，实现了与具体的网络连接的开、关、异常处理等等功能（在 Server 中）的分离。CConsoleServerHandler 实现了 Handler，它需要发送信息到串口控制、SUT 控制等其他几个控制模块，所以内部需要维护各个模块的回调接口，而 CCOMServerHandler 只需要与 VT100 终端通讯，因此只需要维护串口控制模块的对应串口的回调接口。

4.2.3 远程通讯 Agent 子模块实现

```
private Hashtable<String, ArrayList<Channel>> allConnectedChannel;//url<--->list
private synchronized void addChannel(String uri, Channel channel) {
    allConnectedChannel.get(uri).add(channel);
}
//从端口向外发送消息
private synchronized void sendMessage(String uri, String data) {
    ArrayList<Channel> chanels = allConnectedChannel.get(uri);
    for (Channel channel : chanels) {
```

```
            channel.write(new TextWebSocketFrame(data));
        }
    }
    private void handleHttpRequest(ChannelHandlerContext ctx, HttpRequest req) {
        ......//调用 Netty 相关方法建立通道
        ChannelFuture cf = handshaker.handshake(ctx.getChannel(), req);
        ctx.getChannel().setAttachment(uri);//将 URL 信息附着在通道上
    }
    //从端口获取外界消息
    private void handleWebSocketFrame(ChannelHandlerContext ctx, WebSocketFrame
    frame) {
        String msg = ((TextWebSocketFrame) frame).getText();
        Channel channel = ctx.getChannel();
        Object arrachment = channel.getAttachment();
        if (((String) arrachment).equals(Constants4Cloud.CONSOLE_URI)) {
            consoleCallBack.Call(msg);
        } else if (((String) arrachment).equals(Constants4Cloud.HUB_URI)) {
            switcherCallBack.Call(msg);
        } else if (((String) arrachment).equals(Constants4Cloud.SUT_URI)) {
            sutCallBack.Call(msg);
        } else if (allCOMCallBack.get((String) arrachment) != null) {
            allCOMCallBack.get((String) arrachment).Call(msg);
        }
        return;
    }
```

图 4.3 类 CConsoleServerHandler 的相关方法

如图 4.3，可以看到在 Handler 每个远程连接都有一个 Channel 与之对应，无论是串口、SUT 的远程 Applet 控制器，还是充作远程 ITS 控制台的浏览器，每一个远程的 WebSocket 连接都会拥有一个唯一的与之对应的 Channel。每个 WebSocket 连接上来的时候会使用不一样的 URL，通过请求的 URL 的不一样，可以分辨出这个连接是希望进行远程串口控制还是远程 SUT 控制等等。将这些连接按照 URL 作为键存储在哈希表里，这样当向外重定向被测机器的输出流时，就可以准确的找出目标连接。同理，当有控制流从外部连接输入时，也可以按照附着在 Channel 上的 URL 判断出这是哪一种远程控制信号流，从而选择相应的回调实例进行处理。

4.2.4 远程通讯 Applet 子模块详细设计

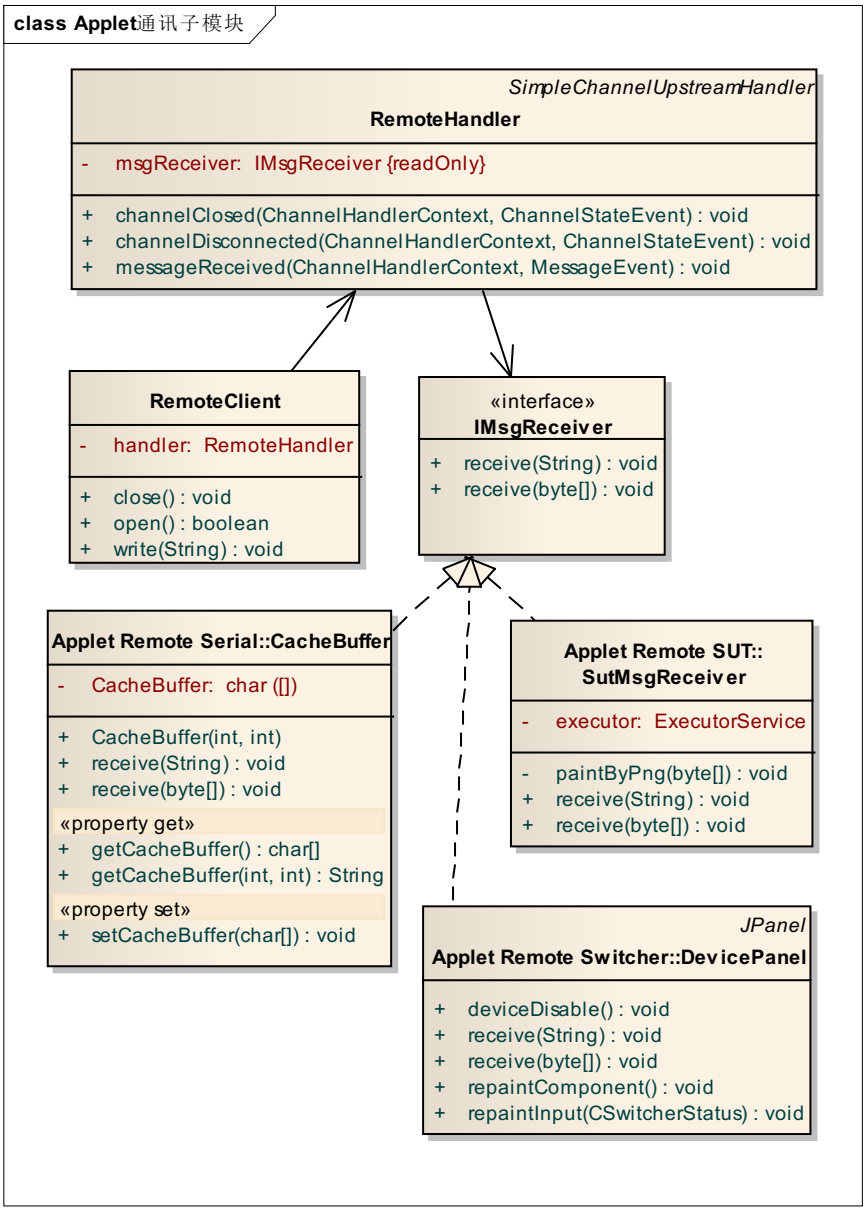

图 4.4 远程通讯 Applet 子模块类图

如图 4.4，在 Applet 端运行远程控制窗口的时候，一个应用只会执行对一个被测机器（串口、SUT 等等）的远程控制，所以一个应用里只有一个 RemoteClient 的实例，该对象负责根据创建 Applet 时刻选择的控制机器类型，以及从服务器获取到的 Agent 的 IP 和相应端口，创建一个 WebSocket 的连接与 Agent 进行远程通讯。

同样，RemoteClient 也只会有一个 Handler 的实例，这个 RemoteHandler 和 Agent 端的类似，也是负责初步处理接收到的消息。与 Agent 端的不同的是，

由于一个 Applet 只会有一个负责单一远程功能的需求，所以也只会有一个负责处理信息的实例。所以 Applet 只需初步解析远程连接发过来的内容，然后根据内容类型的不同调用 IMsgReceiver 实例的不同的函数即可。

IMsgReceiver 是该模块中负责真正处理接收到信息的接口，它在整个项目中有三个不同的实现，分别对应远程串口、远程 SUT 和远程 Switcher，但是单一 Applet 只会实例化其中之一并将之放入 Handler 之中。IMsgReceiver 可以接收两种不同类型的数据，一种是比较简短的字符串（串口信息和 Switcher 状态），一种是二进制流（SUT 的图像信息），由 Handler 按情况调用。

4.2.5 远程通讯 Applet 子模块实现

```
public void messageReceived(ChannelHandlerContext ctx, MessageEvent e) throws
Exception {
    Channel ch = ctx.getChannel();
    if (!handshaker.isHandshakeComplete()) {
        handshaker.finishHandshake(ch, (HttpResponse) e.getMessage());
        return;
    }
    Object frame = e.getMessage();
    if (frame instanceof TextWebSocketFrame) {
        String msg = ((TextWebSocketFrame) frame).getText();
        msgReceiver.receive(msg);
    } else if (frame instanceof BinaryWebSocketFrame) {
        byte[] msg = ((BinaryWebSocketFrame) frame).getBinaryData().array();
        msgReceiver.receive(msg);
    }
}
```

图 4.5 类 RemoteHandler 的相关方法

如图 4.5，RemoteHandler 主要负责对接收到的消息进行分类处理，主要包括一些进行连接的非功能性消息和两种类型的远程输出流。

一种是包装在 TextWebSocketFrame 对象中的字符串，当进行远程串口控制时，该字符串就是直接从串口机器导出的未经处理的串口数据流，当远程 Switcher 控制时，该字符串就是 Switcher 对象的字节形式的状态信息。还有一种是二进制数据流，是直接从 Java 的 BufferedImage 获取的 byte 数组，封装在 BinaryWebSocketFrame 对象中。

4.3 远程串口控制模块

4.3.1 远程串口控制模块介绍

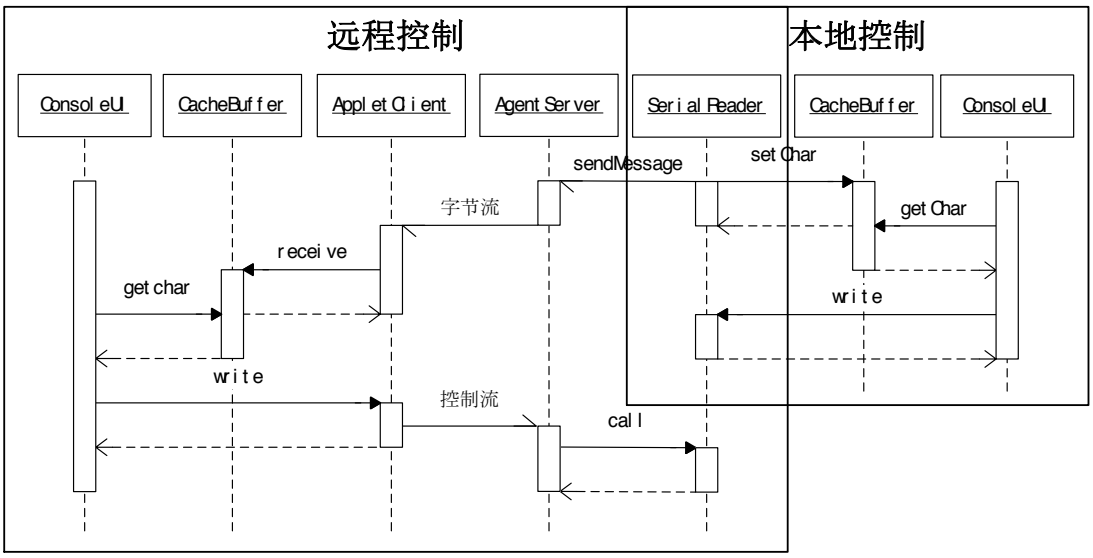

图 4.6 远程串口控制模块时序图

远程串口控制模块是远程控制模块的一个基本模块，主要负责实现远程的串口控制。ITS 本身会通过所在 Agent 的串口与一些设备（比如主板）相连，在本地 ITS 可以打开一个串口控制界面解析串口输出的信息并绘制出来(如同 VT100 终端)，同时用户的键盘操作会被捕捉并生成相应的串口控制命令发送过去。远程串口控制模块就是要在远程的 Applet 窗口实现一个控制窗口，可以和本地一样查看以及操作串口设备。

如图 4.6，右侧是本地控制的时序，SerialReader 获取到的数据会发送到本地缓存，再由本地 UI 获取到并进行绘制。本地 UI 获取到的用户操作也会变成字节数组形式的控制信号通过 SerialReader 发送到实际连接着的串口机器。

左侧是远程控制的时序，SerialReader 将本地输出流以异步信号的形式发送到通讯模块，再由通讯模块通过网络发送到 Applet 端，由 Applet 端通讯模块选择注册好的接收者（也就是 Applet 端的缓存）获取输出流。同时 Applet 端的 UI 会从缓存中得到输出流，绘制并展现给用户，而它捕捉到的用户键盘操作会通过通讯模块直接发送到一开始便注册到 Agent 端通讯模块的 SerialReader 中，实现了对本地串口机器的控制。

4.3.2 远程串口控制 Agent 子模块详细设计

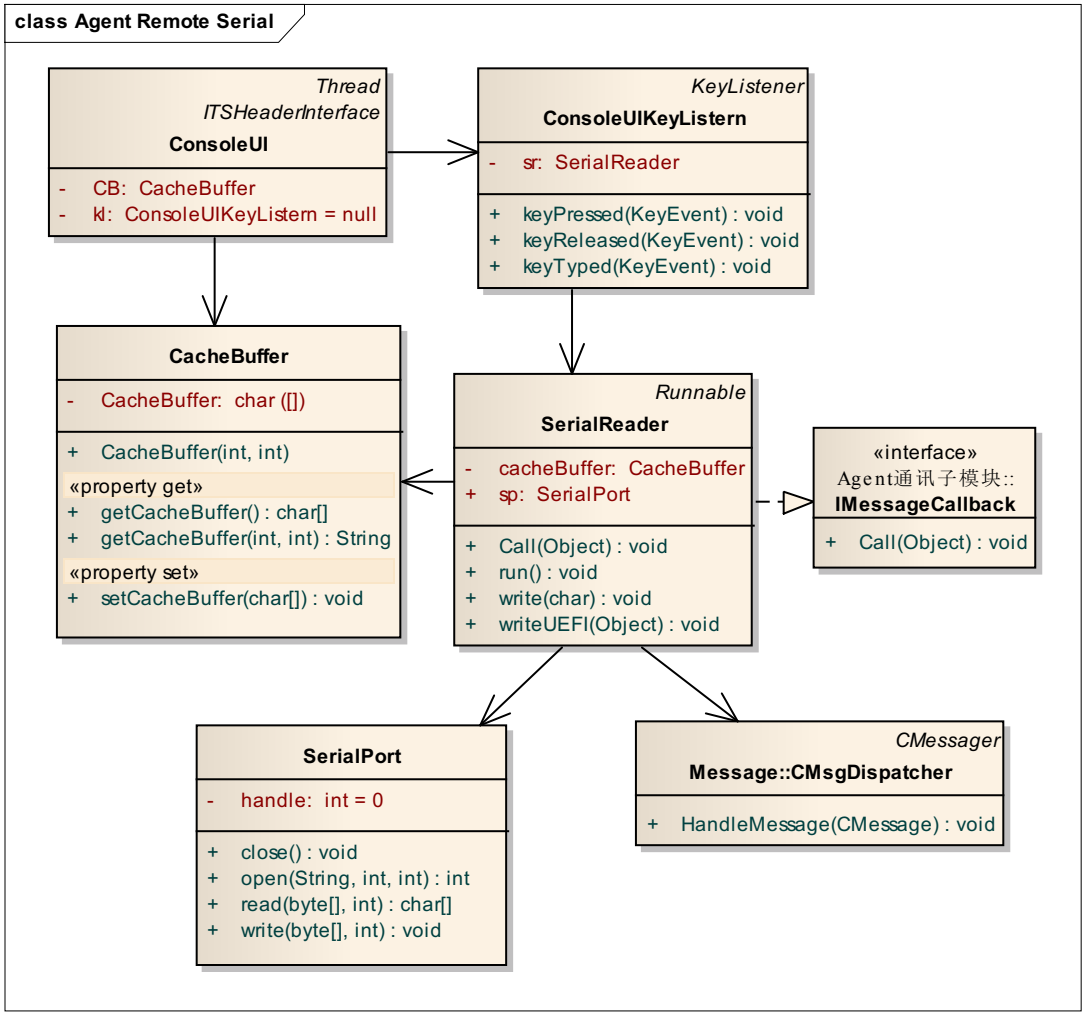

图 4.7 远程串口控制 Agent 子模块类图

图 4.7 是在 Agent 端远程串口控制子模块的类图。SerialPort 类提供了直接对串口进行读写的最简单的方法，包括打开关闭串口、按字节读写等等，都是 JNI 的 native 方法。而 SerialReader 类则是对 SerialPort 的一个封装，一方面 SerialReader 会对外提供写一个字母或者按下一个功能键的接口，自己内部解析成串口需要的字节再去调用 SerialPort 的方法；另一方面 SerialReader 本身是一个线程，它会不断读取串口信息，然后将内容不断写入 CacheBuffer 这个类的缓冲区中。

为了实现远程控制，SerialReader 拥有一个 CMsgDispatcher 的实例，当从串口读取到消息时，不仅仅写入 CacheBuffer，还会通过 Dispatcher 发送到通讯模块，从而实现将串口输出流重定向到远端。同时，SerialReader 本身作为一个

IMessageCallback 的实现会随着消息被发送到通讯模块，并被注册到 Handler 中，其 Call 方法也会在接收到远程消息的时候被调用，然后解析并调用相应的对串口的写入方法，从而实现了远程控制流的重定向。

而 ConsoleUI 一方面绘制展示界面，另一方面也是一个线程，会不断的从 CacheBuffer 的缓冲区里读取最新的字节信息，解析并且进行相应的绘制。UI 上的所有的鼠标事件都交给了 ConsoleUIKeyListern，它会在有键盘操作之后，异步调用 SerialReader 相应的方法，从而使得万一串口读写暂时卡住，UI 界面也不会卡死。

4.3.3 远程串口控制 Agent 子模块实现

```
private void sendRemoteData(String data) {
......//发送数据给通讯模块，同时将 StringBuilder 清零，lastSend 设置为当前时间
}
private long lastSend = System.currentTimeMillis();
private StringBuilder sb = new StringBuilder();
private void addRemoteCache(char data) {
    synchronized (objForRemoteData) {
        sb.append(data);
        if (sb.length() > 100) {
            sendRemoteData(sb.toString());
        } else if (System.currentTimeMillis() - lastSend > 1000) {
            sendRemoteData(sb.toString());
        }
    }
}
private final Thread sendRemoteThread = new Thread("sendRemoteThread") {
    public void run() {
        while (run) {
            Thread.sleep(200);
            synchronized (objForRemoteData) {
                if (System.currentTimeMillis() - lastSend > 500)
                    if (sb.length() > 0)
                        sendRemoteData(sb.toString());
            }
        }
    }
};
```

图 4.8 类 SerialReader 的相关方法

如图 4.8，如果 SerialReader 每次从 SerialPort 中读取到一个字节的信息，就直接通过网络发送到远程 Applet，肯定会过频繁的网络访问，一方面加大资源占用，另一方面也增加出错的可能性。所以在 SerialReader 内设置了一个缓冲，需要发送的字节先在这里缓存到一定数量再统一发送，同时还会有线程监视，要是超过一定时限没有新的数据获得到，也会将缓冲里的数据发往远端。

4.3.4 远程串口控制 Applet 子模块详细设计

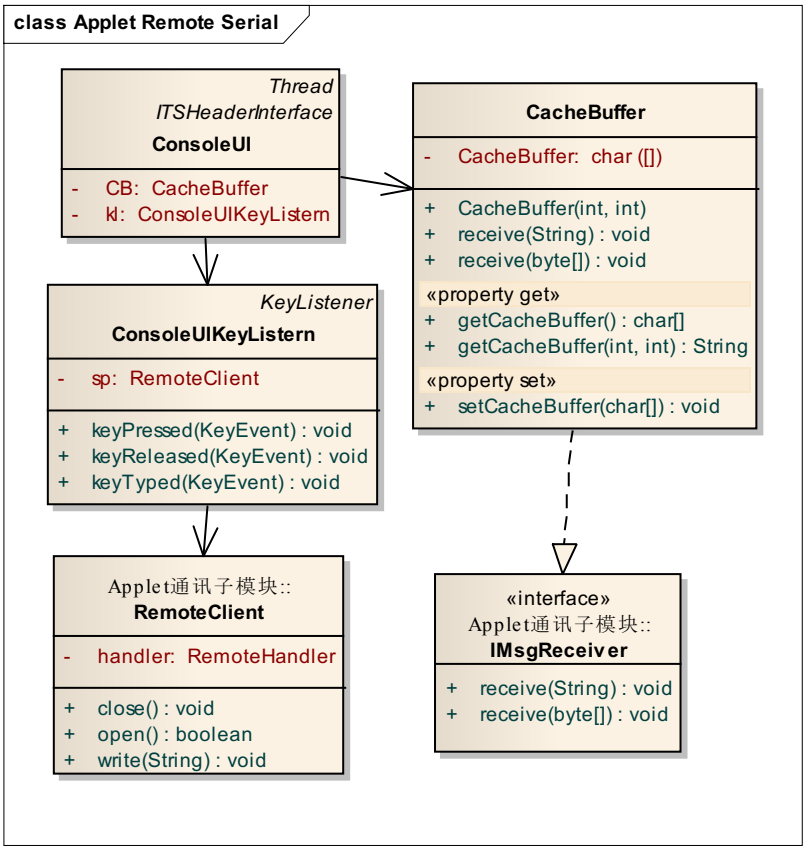

图 4.9 远程串口控制 Applet 子模块类图

如图 4.9，Applet 端的 ConsoleUI 类似 Agent 端的同名类，其所在线程一样会不断从 CacheBuffer 的缓存区中读取内容，绘制到界面，同时一样拥有一个 KeyListener 来监视键盘事件。

但是与在 Agent 上不一样，CacheBuffer 内的缓存区不再由 SerialReader 填充。因为其实现了 IMsgReceiver 并被注入通讯模块的 Handler，从而它的 receive 函数会在通讯模块接收到 Applet 消息（也即是从串口获得的输出流）时被调用，消息会被作为参数传入，然后直接被写入缓存区。

4.3.5 远程串口控制 Applet 子模块实现

```
public void keyTyped(KeyEvent e) {
    int code = e.getKeyCode();
    //inputSets 是一个 code 与控制码对应的哈希表
    if (inputSets.contains(code))
        writeThread.write(e.getKeyChar());
}
private class WriteThread extends Thread{
    private LinkedBlockingQueue<Character> mMsgQueue = new
LinkedBlockingQueue<Character>(5);
    public void run() {
        while (run) {
            char code = mMsgQueue.take();
            writeCodeToAgent(code);//向串口模块发送控制信息
        }
    }
}
```

图 4.10 类 ConsoleUIKeyListern 的相关方法

如图 4.10，ConsoleUIKeyListern 在触发了键盘事件的时候，会直接从内部早已记录好的哈希表中获取到相应的串口所能认识的二进制控制码，然后直接将这个控制码发送到 Applet 端。

另一方面，考虑到可能存在网络通讯延迟等卡住的现象，这些命令并不直接发送给通讯模块，而是存储在一个 LinkedBlockingQueue 中，由一个专门的线程负责调用通讯模块的发送方法。这样当出现通讯卡住的情况时，会记录最早 5 个控制命令等待恢复后再发送。而更多的控制命令会被直接抛弃，这样就不会因为一时的网络问题导致整个 UI 卡死。

4.4 远程 SUT 控制模块

4.4.1 远程 SUT 控制模块介绍

远程 SUT 控制模块是实现远程控制一个模块，主要负责对 SUT 设备的远程控制。本地 ITS 会需要测试一些有视频输出的、可以被鼠标键盘控制的设备（比如平板或者加载了操作系统的主板设备），ITS 会在本地打开一个显示窗口，再通过 HDMI 采集卡获取到被测机器的视频输出并显示在窗口上，同时用户在窗口

上的鼠标键盘操作都会被 ITS 捕捉并通过一个连接在 Agent 上的鼠标键盘控制器实现对被测机器的控制。远程 SUT 模块就是要将这个本地窗口能够通过 Applet 在远端打开并正常工作，同时用户在远程窗口的操作也一样能通过网络发送到 Agent 实现对本地被测机器的控制。

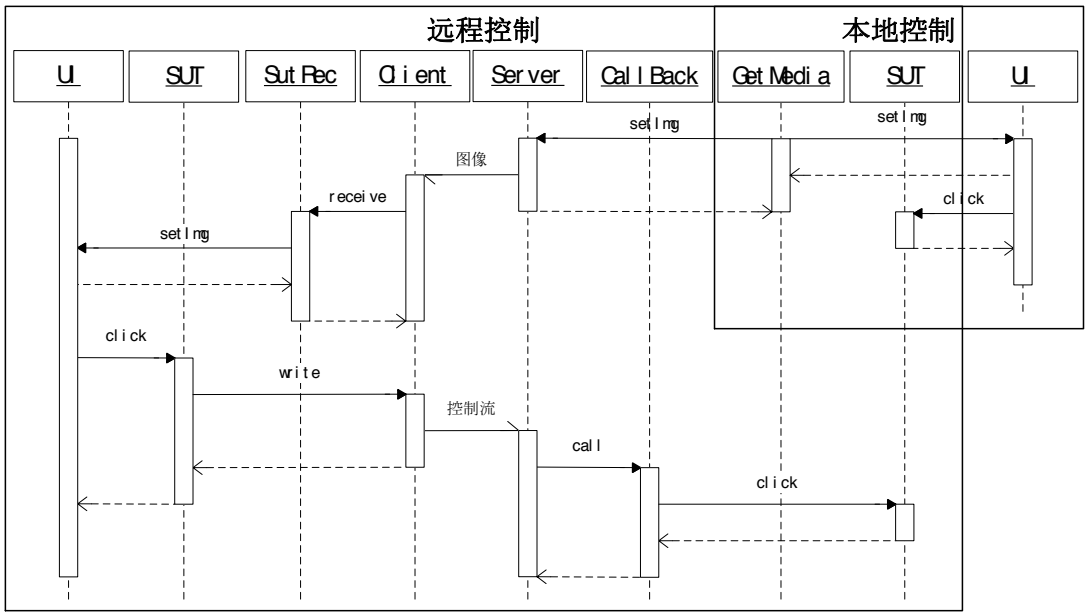

图 4.11 远程 SUT 控制模块时序图

如图 4.11，本地获取到的图像数据被同时发往了本地 UI 和通讯模块，本地 UI 会直接把图像绘制在界面，而通讯模块则将图像发送到 Applet 端。本地 UI 获得到用户的鼠标键盘操作以后会直接调用 SUT 的相关方法，实现向鼠标键盘控制器发送信号，从而控制被测机器的鼠标键盘事件。

发往通讯模块的图像经过处理以后，会选择相应的 Applet 传输过去。Applet 在获取到数据以后，会调用相应的接收者接收图片信息。此时这个接收者就类似本地控制里面的 GetMedia，会不断获得图像信息，然后传递给 Applet 端的 UI 进行绘制。Applet 端的 UI 的用户操作事件会交由 Applet 端的 SUT 处理，它会将操作事件编码并发送到 Agent 端，Agent 端的通讯模块将命令发送到相关回调对象里，由该对象解码得到用户操作及其参数，再来调用事实上进行鼠标键盘控制的 Agent 端 SUT 对象。

4.4.2 远程 SUT 控制 Agent 子模块详细设计

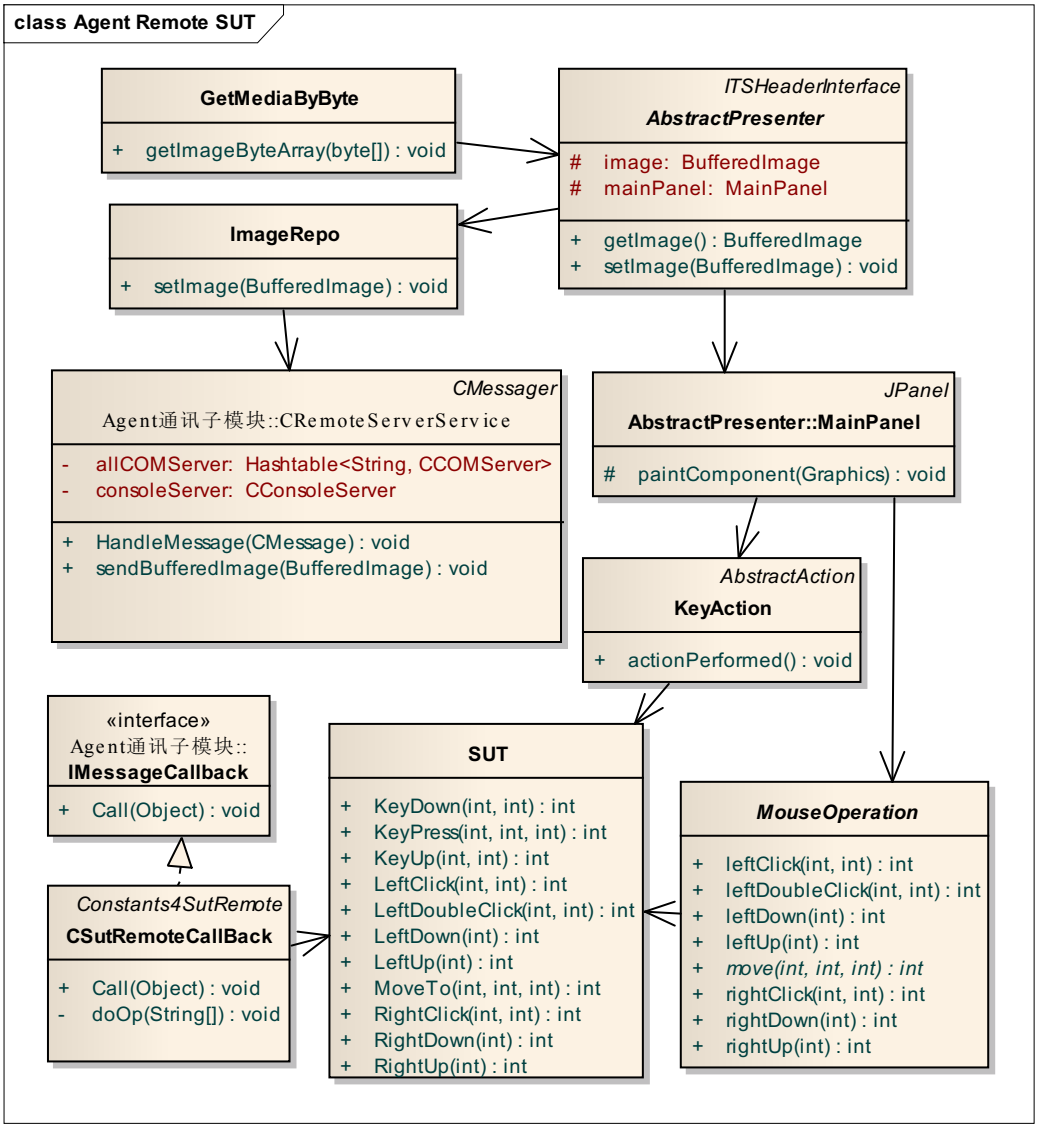

图 4.12 远程 SUT 控制 Agent 子模块类图

如图 4.12 所示，远程 SUT 控制模块和远程串口控制模块在底层实现上有所不同，读取设备视频输出流和进行鼠标键盘控制是由两个类分开进行处理的。

其中 GetMediaByByte 负责视频流的获取，它通过调用 native 方法启动底层的视频获取，然后 getImageByteArray 会被底层的线程以每秒 15 次的频率调用，传入的参数就是获取到的当前需要展示的图像位图的字节数组。

字节数组再被封装到 BufferedImage 对象中传递到 AbstractPresenter，再由它将图像对象的引用进行分发，一份放到 MainPanel 中，等待在重绘 UI 的时候使用，另一份引用被 ImageRepo 直接传到通讯模块，等待被压缩发送出去。这

里之所以不使用拷贝数据而是使用传递引用，是因为每秒 15 帧每帧都有几百甚至上千 KB，要是拷贝数据会导致内存严重不足，并大量占用 CPU 资源，而且无论是传送到 Applet 还是绘制到 UI，都只是对引用的只读操作，不会互相影响，所以采用了这个传递引用的方案。

另外，与其他三个控制子模块最大的不同在于 SUT 的视频输出流并不是通过发送消息的方式传递给通讯模块的，而是直接获取了通讯模块的实例，通过函数把图像传递过去。这是考虑到一方面图像的传递在打开控制的情况下每秒都需要十余次，另一方面对于图像的处理时间远远高于处理其余控制类型下传递的字符信息的时间。这会导致大量的消息在消息队列里积累，一方面使得其余控制的输出信息难以被即时转发，另一方面也会导致内存泄露或者是消息丢失。

SUT 这个类是负责对被测机器的鼠标键盘控制，它通过 Java 的 JNA 机制，直接把本身的 native 方法映射到加载的动态链接库相应的函数。因此，可以通过直接调用 SUT 实例里的方法实现对被测机器的控制。

SUT 的实例在本子模块中主要被两方掌握，一方是 UI 的鼠标键盘事件的监听者（KeyAction 和 MouseOperation），它们会在监听方法被调用时调用相应的 SUT 实例内部的方法；另一方是一个实现 IMessageCallback 接口的回调实例，它会在接收到远程控制信号后，解析并调用 SUT 实例的方法。

4.4.3 远程 SUT 控制 Agent 子模块实现

```
private ExecutorService executorService;
private ByteBuffer byteBuffer;
private ChannelBuffer cb;
public void sendImgToRemoteSUT(final BufferedImage img) {
    executorService.submit(new Callable<Boolean>() {
        public Boolean call(){
            ImageIO.write(newImg, "jpg", out);
            changeByteBuffer(out.toByteArray().length);
            handler.sendImgToSutApplet(bwsf);
            return true;
        }
        private void changeByteBuffer(int length) {.....}//修改缓冲区大小
    });
}
```

图 4.13 类 CConsoleServer 的相关方法

如图 4.13，CConsoleServer 在接收到发送过来的图像的缓存之后，并不是直接调用压缩和发送相关的方法，而是将相关的方法和图像信息一起放入一个任务，再放入线程池，由线程池调用。线程池特意设置成只有一个定长度为 1 的任务队列，这样当后一帧在前一帧的数据还未处理前就被传递过来时，直接抛弃后一帧的数据，以求在 CPU 或者带宽吃紧的情况下，以掉帧为代价尽量保证远程控制的实时响应速度。

另一方面，在压缩图像时选择压缩为 JPG，该格式经过测试在色彩比较单调的情况下比 PNG 要大，但是在色彩丰富的情况下比 PNG 要小，而且无论哪种情况下使用 Java 自带的导出方法，都是 JPG 更快。更小的 GIF 则画质太差。

压缩期间会使用一个缓冲区，因为 JPG 在不同情况下相同分辨率的图像可能有很大区别，因此该缓冲区的所需大小可能会有极大的变化，为了减少对资源的浪费，需要在合适时机下修改缓冲区的大小。

4.4.4 远程 SUT 控制 Applet 子模块详细设计

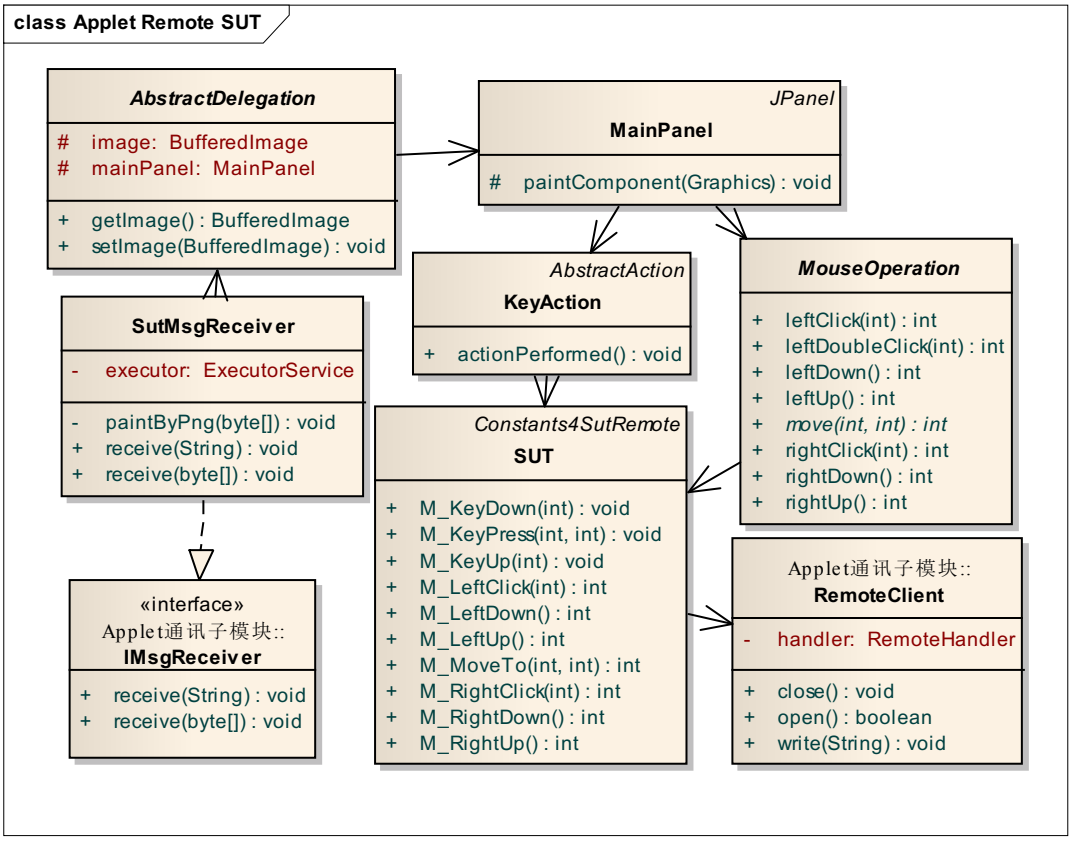

图 4.14 远程 SUT 控制 Applet 子模块类图

如图 4.14，远程 SUT 控制模块在 Applet 端是通过一个实现了 IMsgRecer 接口的 SUTMsgReceiver 来从通讯模块实时获取被测机器的图像信息的，类似于 Agent 端的 GetMediaByByte 获取视频信息的方式。它在接收到从 Agent 发送过来 JPG 格式的二进制图像后，会重新打包成用于 Java 绘制 UI 的 BufferedImage 并发送给 MainPanel。

MainPanel 的鼠标事件监听和键盘事件监听与 Agent 一样，交由 KeyAction 和 MouseOperation，它们和 Agent 端一样会在监听到相关事件之后去调用 SUT 实例的相关方法。

这里的 SUT 和 Agent 端的 SUT 一样担负着进行控制的任务，只是不再是通过调用相应的 native 方法，而是对操作进行编码之后通过通讯模块发送到 Agent。

4.4.5 远程 SUT 控制 Applet 子模块实现

```
private ExecutorService executorService;
private static long old = System.currentTimeMillis();
public void receive(final byte[] dataArray) {
    long now = System.currentTimeMillis();
    int size = dataArray.length / 1024;// 获取该帧图像的大小 kb
    long now = System.currentTimeMillis();
    long gap = (now - old);
    double fps = ((double) 1000) / gap;
    double sizePs = size * fps;
    ......//将计算出的大概的实时下载速率与帧数显示出来
    executorService.submit(new Callable<Boolean>() {
        public Boolean call() throws Exception {
            paint(dataArray);//绘制到 UI
            return true;
        }
    });
}
```

图 4.15 类 SutMsgReceiver 的相关方法

如图 4.15，SutMsgReceiver 在处理接收到的图像的时候，首先进行实时带宽占用情况和帧率的计算，并将之显示出来。

另一方面，考虑到将字节流重新转化成图像是一件比较耗时的操作，因此和 Agent 端类似，将字节数组转换成图片并绘制这个工作放入线程池，这样可以保

证接收数据不被干扰，提高的数据接收速度。同样该线程池也是只保留一个待处理任务，因为远程显示的时候为了及时显示，可以放弃个别帧不进行绘制。

4.5 远程 Switcher 控制模块

4.5.1 远程 Switcher 控制模块介绍

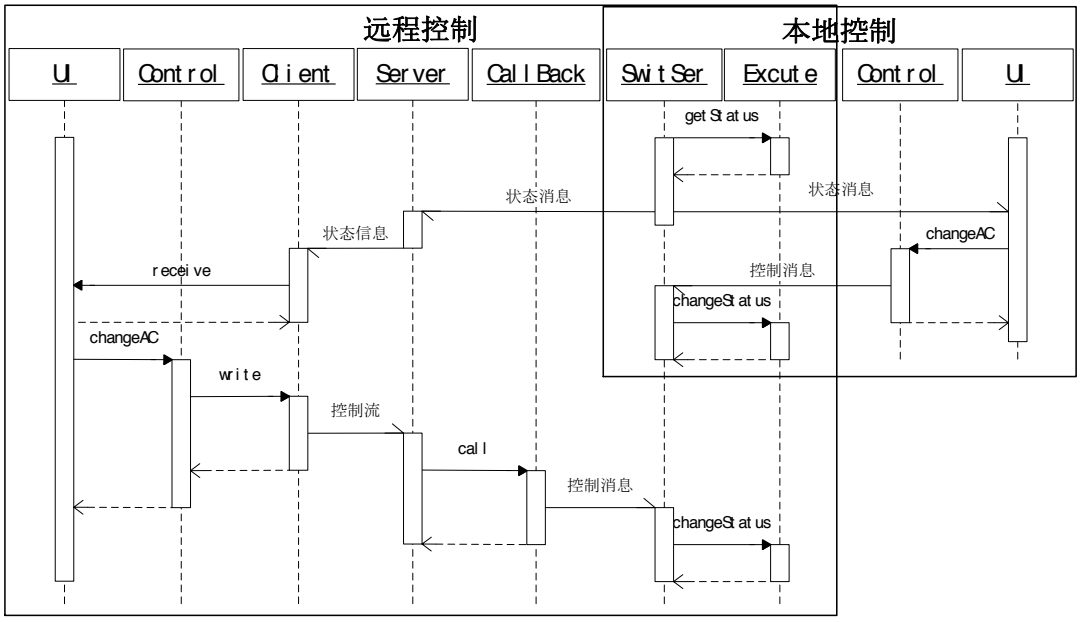

图 4.16 远程 Switcher 控制模块时序图

远程 Switcher 控制是实现远程控制的一个模块，主要负责对 Switcher 设备的控制。在本地 ITS 会通过 Usb 线与一个 Switcher 设备相连，这个 Switcher 设备有一些可以实现高低电位切换的接线，这些接线如果连接上主板的开关就可以实现控制开关的连通状态；还对外提供了交流和直流两种电力输出，可以实现对主板和硬盘通电状态的控制；还有一组硬盘接线口，通过对内部接线的切换实现主板所接硬盘的切换等等。本地 ITS 有一个状态页面显示 Switcher 上各个设备的状态，同时可以通过在状态页面的点击实现对 Switcher 控制。它相应的远程控制模块就是把这个界面移到 Applet 上并保持功能不变。

如图 4.16，本地的 Switcher 服务直接通过 Excuter 组件控制 Switcher，同时通过消息与其他组件进行通讯。为了实现远程控制，Switcher 状态的消息会在被发往本地 UI 服务的同时被发往通讯模块并发送到 Applet，由 Applet 端解析并且展现。同样，Applet 端的控制信息在通过通讯模块发送到回调对象以后，也会

被以控制消息的形式发送到 Switcher 服务，这个控制消息的形式与本地 UI 发送过来的控制消息一样，都会被该服务解析并最终控制 Switcher 设备。

4.5.2 远程 Switcher 控制 Agent 子模块详细设计

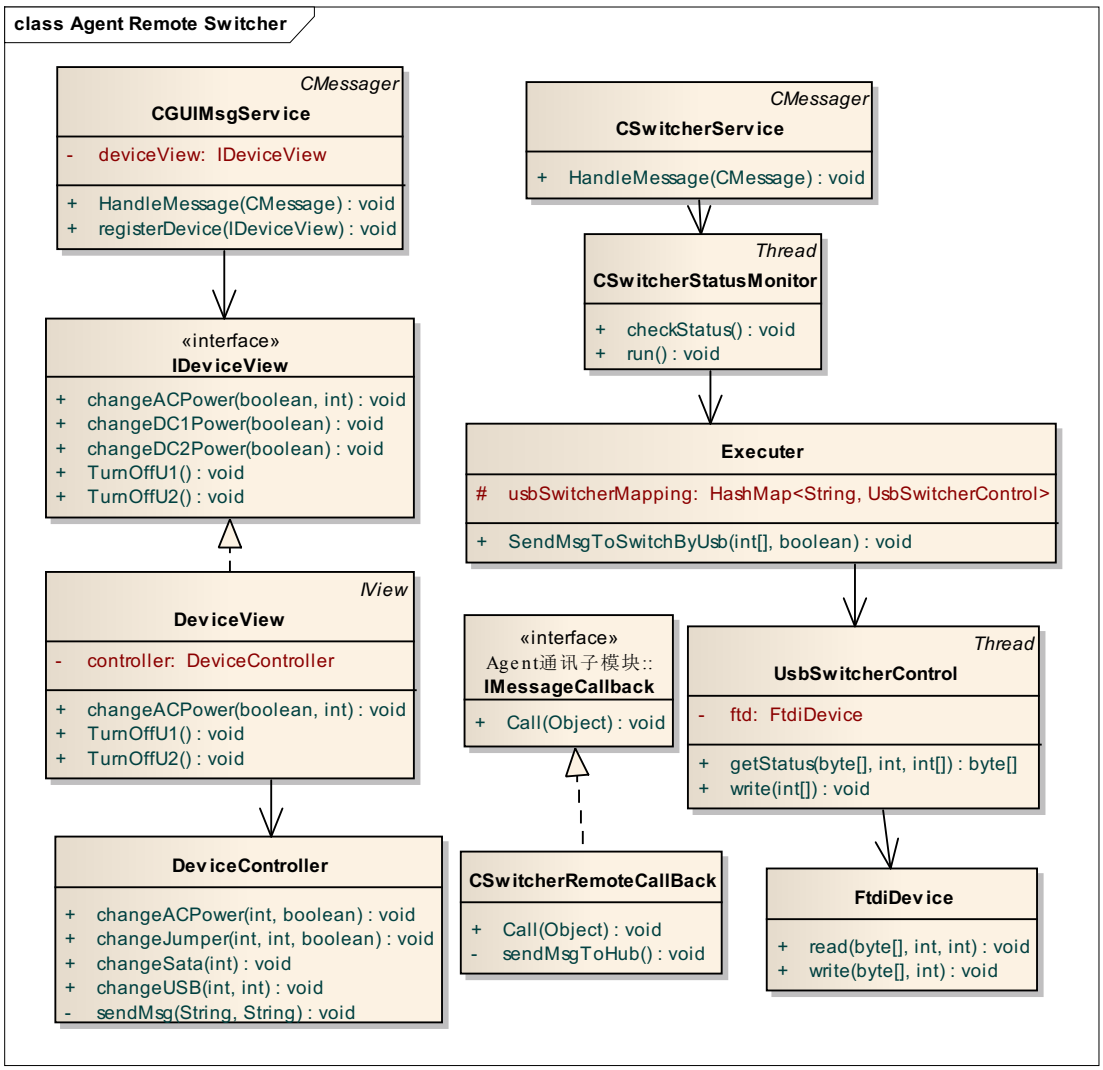

图 4.17 远程 Switcher 控制 Agent 子模块类图

如图 4.17，Switcher 控制模块和 SUT 以及串口控制模块的不同在于其本身拥有一个独立的可以接收消息的 Service。串口和 SUT 的 UI 并不是由 ITS 的 UI 直接创建的，而是由 Excuter 组件创建的，因此他们直接拥有了底层的实例。而 Switcher 的 UI 是 ITS 的一个标签页，因此，如果它直接拥有底层的实例会破坏原有的各个 Service 之间的独立性，所以新建了一个 Service 来解决这个问题。

CSwitcherService 就是这个服务，它会接收从消息转发组件发送过来的消息（可能来自本地 UI 或者通讯模块），获悉用户的操作（比如打开 AC，关闭 DC），

然后解析成为控制 Switcher 机器的一组字节命令，通过 CSwitcherStatusMonitor 发送到 Switcher 上。

CSwitcherStatusMonitor 是 Switcher 状态的监视器，它内部的线程会每个一段时间去获取一下 Switcher 的状态，当 Switcher 的状态发生变化的时候，该监视器会将最新的状态附着到消息内，交由消息转发组件进行发送，一份发送至本地 UI 的 Service，一份转发到通讯模块。

监视器与底层的交互是通过 Executer 调用 UsbSwitcherControl 里的相关方法实现的。与 SUT 和串口不一样，无论 Switcher 的状态是否发生变化，都不可以直接从底层获取到当前的状态，而是需要先写入一个获取状态的字节数组命令，然后才能通过相应的读方法获取到 16 进制的状态。这个通过一次写命令和一次读命令获取状态的函数就放置在 UsbSwitcherControl 里，同时它也负责监视底层的连接状态是否正常，以及在必需的时候（比如接线松动）重新调用底层的连接等等操作。FtdiDevice 就是负责进行底层的和 Switcher 设备进行逐个字节读写的工作。

Switcher 控制端在 Agent 本地的 UI 是一个类似 MVC[弗里曼,2007]的架构，当底层数据发生变化（也就是状态发生变化），GUI 的消息服务会收到一条消息，里面是最新的状态，这个服务会去修改 UI 上的数据并刷新 UI。而当 UI 监听到用户的操作需要去修改 Switcher 状态的时候，也是通过一个 Controller 的实例，将需要变化的信息发送到另外的服务。

CSwitcherRemoteCallBack 是一个实现了 IMessageCallback 的回调实例，它会解析从通讯模块得到的远程控制窗口发来的控制命令，然后新建一个服务之间的消息，通过消息转发的组件发送到 CSwitcherService 去，通知它需要对 Switcher 进行某些操作。而这样对于 CSwitcherService 来说，它完全不需要关心接收到的控制的信息是有谁发送过来的，也完全不需要关心它发送出去的状态的信息会被转发组件发送给哪个服务。这样在完全不修改 CSwitcherService 本身的情况下，实现了本地控制到远程控制的扩展。

4.5.3 远程 Switcher 控制 Agent 子模块实现

```
private final ChannelFutureListener handshakeListener = new ChannelFutureListener() {
    public void operationComplete(ChannelFuture channelFuture) throws Exception {
        ......//建立连接通道
        sendInitMsg((String) uri, channelFuture.getChannel());
    }
};
private void sendInitMsg(String uri, Channel channel) {…}//向新连入的 Switcher 写
private CSwitcherStatus oldStatus;
public void sendStatusToRemoteSwitcher(CSwitcherStatus status) {
    if (oldStatus == null || !oldStatus.equals(status)) {
        oldStatus = status;
        sendMessage(Constants4Cloud.SW_URI, status.getStatusStr());
    }
}
```

图 4.18 类 CConsoleServerHandler 的相关方法

如图 4.18，CConsoleServerHandler 在处理发送 Switcher 状态的时候遇到了一个问题：Switcher 的状态不会经常变化，甚至很有可能很长时间没有变化，因此相关监视 Switcher 的模块是不会重复发送相同的状态，这也是为了减少不必要的资源消耗。但是这就导致了通讯模块的向远程发送 Switcher 状态的函数可能很长直接不会被调用到，从而导致在这期间连接上 Agent 的远程控制器接收不到需要的 Switcher 状态。

因此，在 CConsoleServerHandler 内会保留一个最新的 Switcher 的状态的实例，当有新的连接通道建立之后，要是发现是 Switcher 的远程控制器，就将这个最新的 Switcher 的状态发送过去。另外，只要主动向远端发送 Switcher 状态的函数一被调用，这个状态就会被重新赋值。

4.5.4 远程 Switcher 控制 Applet 子模块详细设计

如图 4.19，在 Applet 上采用了和 Agent 上相似的类 MVC 方案，代表 View 的 DevicePanel 将自己的实例以 IMsgReceiver 接口的形式注入到通讯模块中去，这样当通讯模块接收到 Agent 发送过来的信息时（也就是数据发生变化时），就会自动调用 View 里的方法将信息传入。View 在解析获取到了需要变化到的状态后，相应的修改 UI 将之显示出来。

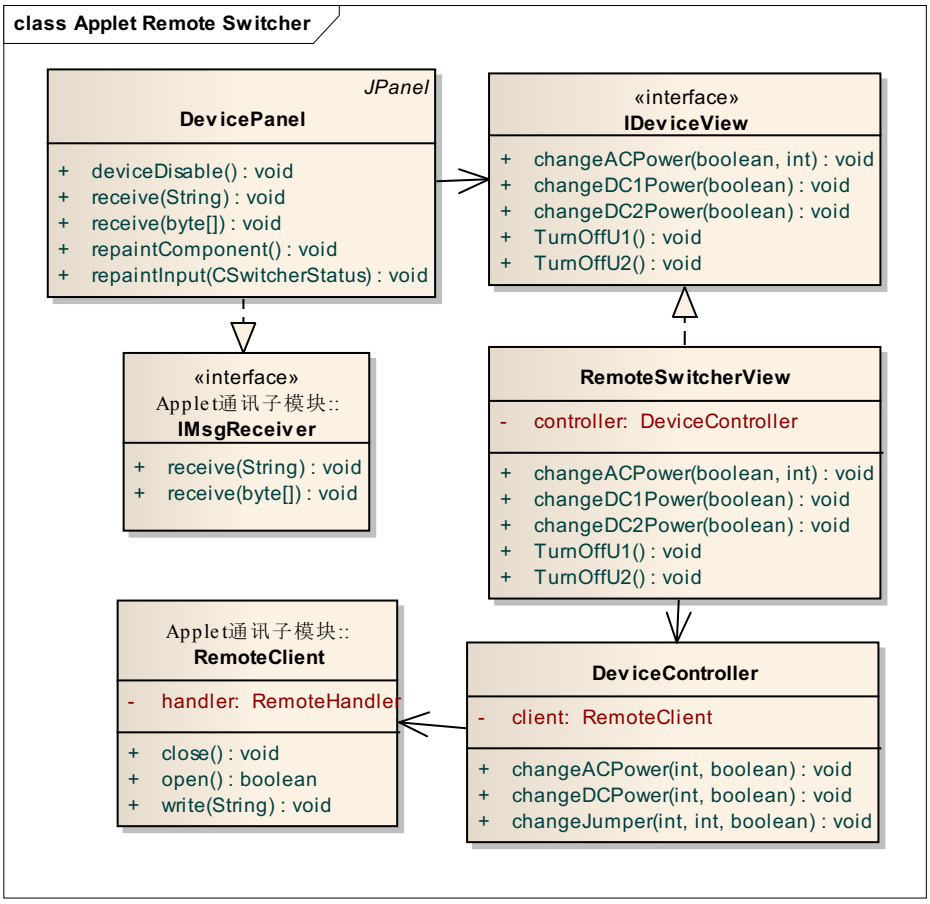

图 4.19 远程 Switcher 控制 Applet 子模块类图

当 View 监听到用户事件之后，直接调用 Controller 的相关方法去修改数据（也就是改变 Switcher 状态），修改方法与 Agent 类似是发出消息，只是这个消息需要在网络传输之后由 Agent 重新封装成 Switcher 服务识别的信息。

4.5.5 远程 Switcher 控制 Applet 子模块实现

```
public void changeDCPower(int DCNumber, boolean on) {
    String name = "DC";
    String control = "" + DCNumber + ",";
    if (on) {
        control = control + "On";
    } else {
        control = control + "Off";
    }
    sendMsg(name, control);
}
private void sendMsg(String name, String control) {…//发送给 Agent}
```

图 4.20 类 DeviceController 的相关方法

如图 4.20 就是 DeviceController 的一段典型的发送处理方案，简单的将一个用户的控制操作分解为 name 和 control 两个部分。Name 表示这个操作针对的是哪一个设备，比如 AC1，DC2，Jumper8 这样。Control 表示是对这个设备的操作，比如 On 和 Off。

4.6 远程控制台控制模块

4.6.1 远程控制台控制模块介绍

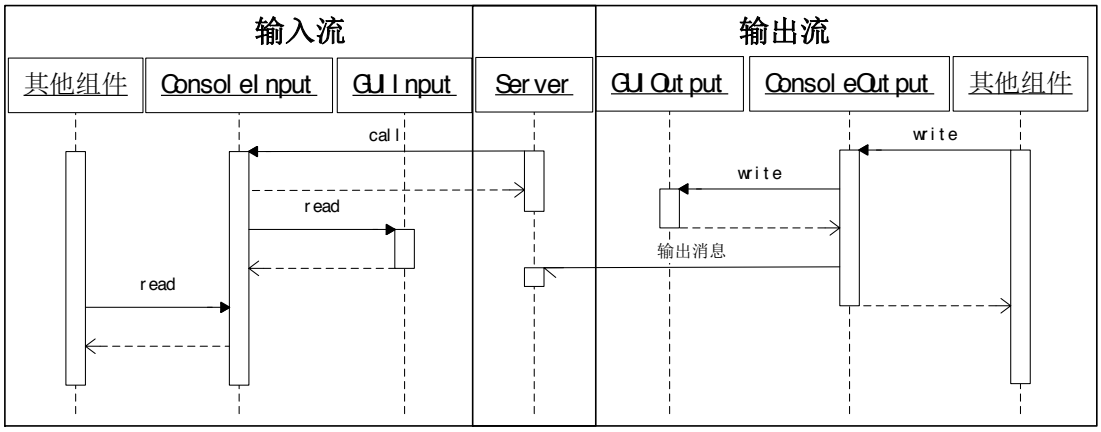

图 4.21 远程控制台控制模块时序图

远程控制台的控制模块是远程控制的最后一个模块。本地 ITS 有一个控制台，ITS 内部组件的输出信息（包括日志信息、调试信息和错误信息等）都会被发送到这个控制台界面并显示给用户，同时用户也可以通过这个控制台界面输入控制命令（比如调试命令、终止暂停的命令等），这些输入信息会被需要的 ITS 组件获得到。控制台的远程控制模块会将这些输出信息发送到浏览器上，由远程控制台的浏览器端子模块展现给用户，同时会把用户在浏览器上的输入信息通过网络发送到 Agent 上，再被需要的组件获取。浏览器端的子模块较为简单，便按输入输出两类来主要分析 Agent 端的构成。

如图 4.21，其他组件的输出信息会先发送到远程控制相关的类里，再由其发送到原本的 UI 输出类以及通讯模块里，这和其他几个远程控制组件是不一样的。而当组件想要获取输入流，也是要先访问远程控制相关的类，再由它从原本的输入类以及通讯模块两个方向获取输入信息。

4.6.2 远程控制台输出流子模块详细设计

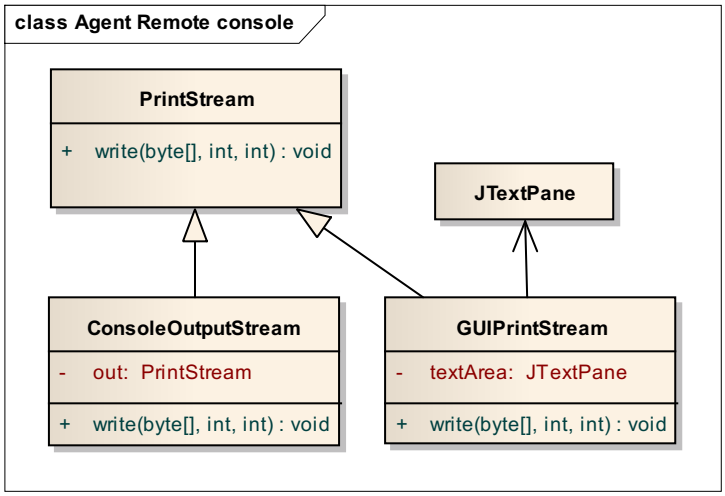

图 4.22 远程控制台输出流子模块类图

如图 4.22，GUIPrintStream 即是原有的继承 PrintStream 的标准输出类，它被重定向到 System.out，使得标准输出的信息会被发送到其中。它又有 UI 的依赖，当有输出信息到达时，便会绘制到 UI，从而显示给用户。

ConsoleOutputStream 是连接到云平台之后的标准输出类，同样继承了 PrintStream。当用户选择连接上服务器时，便会被通过 System.setOut 函数设置成标准输出。它本身会保存之前的标准输出类，这样可以使得原有的显示给本地用户的 UI 正常工作，也可以在程序断开连接时恢复到之前的状态。

4.6.3 远程控制台输出流子模块实现

```
private PrintStream out;
public void write(byte[] buf, int off, int len) {
    out.write(buf, off, len);
    final String message = new String(buf, off, len);
    ...//通过消息转发模块发送消息给通讯模块
}
```

图 4.23 类 ConsoleOutputStream 的相关方法

如图 4.23，ConsoleOutputStream 为了实现不影响原有 UI 输出情况下的远程转发，采用了类似装饰者模式[Erich Gamma,2007]的方法。它保留了原有的 UI 输出实现者，在其就收到标准输出流之后，会先调用原有的输出方法，再将输出信息发送到通讯模块由其转发到远程的浏览器上去。

4.6.4 远程控制台输入流子模块详细设计

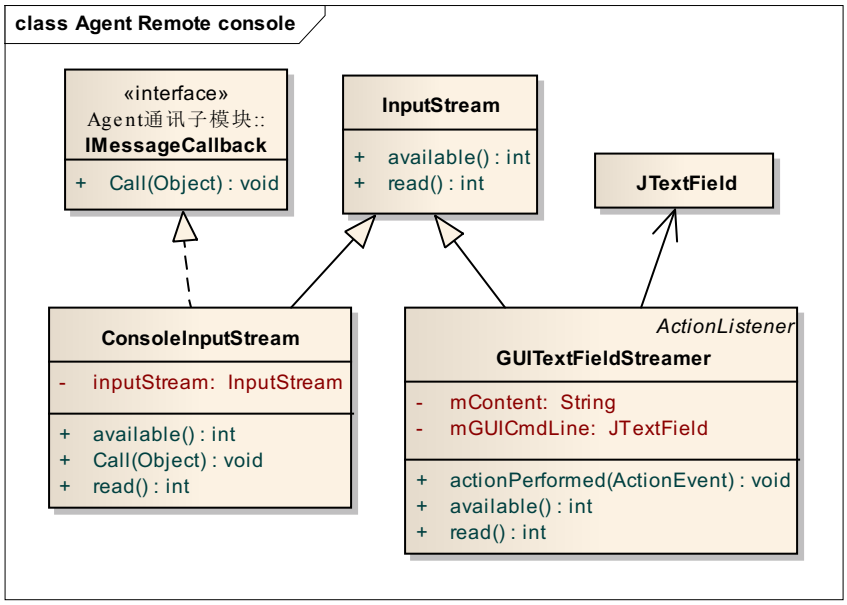

图 4.24 远程控制台输入流子模块类图

如图 4.24，GUIInputStream 继承自 InputStream，其对外提供的 available 和 read 方法，就是方便 ITS 的其他组件从 System.in 这个标准输入流中获取输入。GUIPrintStream 本身有一个 UI 类的依赖，同时又负责处理该 UI 类的鼠标监听，当有用户输入命令时，会去获取那些数据，存入缓冲中，等待被其他模块从流中取走。

ConsoleInputStream 是云平台模式下的标准输入流的实现类，同样继承自 InputStream。它会保留在进入云平台之前的标准输入流，从中获取信息从而使得原有的输入流照常工作，也用于在退出云模式时恢复。ConsoleInputStream 还是 IMessageCallback 的一个实例，这样可以接收远程信息。

4.6.5 远程控制台输入流子模块实现

```
private String mContent = null;
private int mPointer = 0;
private synchronized void resetContent(String content) {
    mContent = content;
    mPointer = 0;
    this.notifyAll(); //Content is available, so unblock the this.wait()
}
@Override
```

```
public synchronized void Call(Object Arg) {
    resetContent((String) Arg + '\n');
}
private Thread thread = new Thread() {
    public void run() {
        while (run) {
            if (inputStream.available() > 0) {
                …//从原有输入流获取数据
                resetContent(tmpContent);
            } else {
                Thread.sleep(100);
            }
        }
    }
};
```

图 4.25 类 ConsoleInputStream 的相关方法

如图 4.25，类 ConsoleInputStream 含有一个 String 作为缓冲，一个 int 记录当前读取到的位置。之所以直接使用 String 而不是 StringBuilder 或者 byte 数组，是考虑到无论是从之前的输入流还是从控制模块，获取到的数据都是以换行符结尾的一整行数据，同时当有新的数据获得到而老的数据又没有被取走时，老的数据就应该被直接抛弃了，以免其他模块获取到的信息不是所需要的最新的。

另外 ConsoleInputStream 含有一个线程，该线程会不断从保存下来的老的输入流获取信息。获取到的信息会用来重置缓冲与指针，重置的方法同时也会被接收网络端信息的函数调用。

4.7 错误重现子模块

4.7.1 错误重现子模块介绍

错误重现子模块是调试模块的最重要的一个模块，负责在用户希望调试错误的时候重现错误现场以方便用户在远程进行调试。该子模块分为 Server 端和 Agent 端两部分，Server 端负责找出需要运行的脚本以及出错时记录下来的需要调试的错误的位置，将相关信息打包并发送到 Agent 上。本小节主要讲述的是 Agent 端的详细设计和实现。

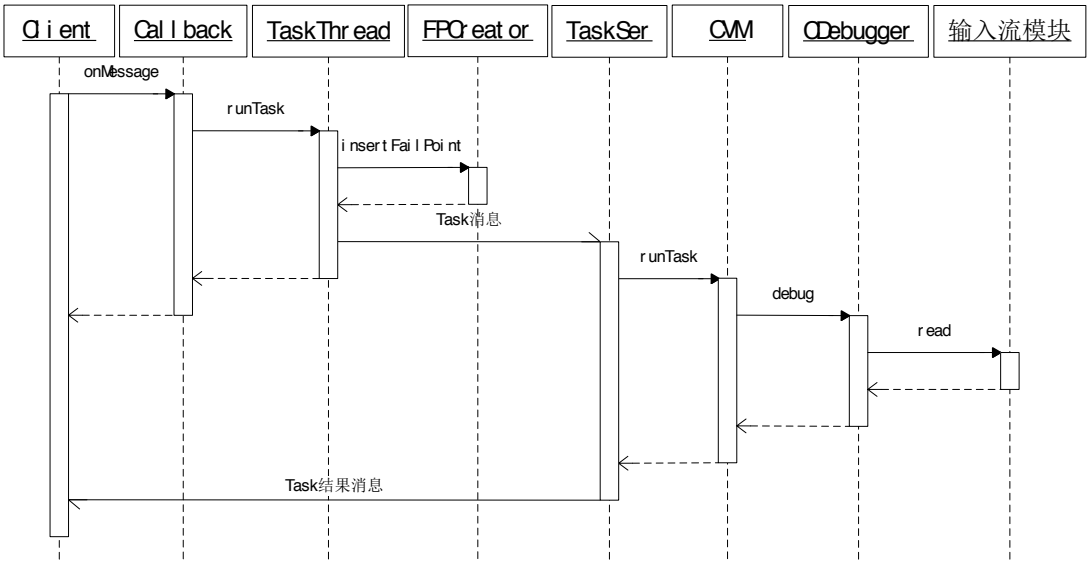

图 4.26 错误重现子模块时序图

如图 4.26 所示，Agent 端的错误重现子模块依赖于正常运行从 Server 发来的任务的相关的组件。该子模块主要负责在用户选择重现错误，并将需要重现的脚本发送到 Agent 以后，将错误断点插入脚本，并以调试模式来让虚拟机运行脚本。同时还通过对编译的修改，使得虚拟机能识别错误断点并像普通的调试断点一样在遇到时暂停下来等待用户的输入。此时的输入是从远程控制台控制 Agent 子模块获得到的，也就是用户在远程的输入命令。同时调试的信息也会打印到远程浏览器上，这样便实现了用户远程调试 Agent 上的脚本。

4.7.2 错误重现子模块详细设计

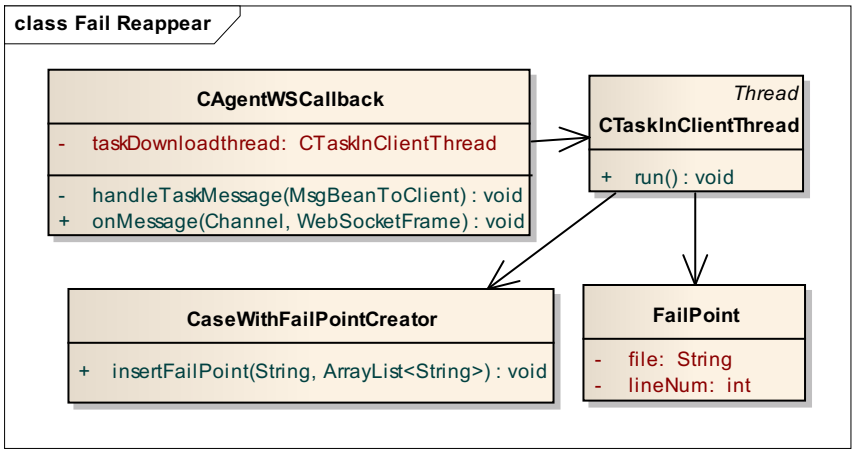

图 4.27 错误重现子模块类图

如图 4.27，CAgentWSCallback 负责处理从 Server 端接收到的消息，在本模块的需求中主要体现在处理运行任务的消息。错误重现其实和运行一个任务是几乎一样的 Json 消息，只不过错误重现会在运行的脚本中加入断点，而且会以“重现”模式运行任务。CTaskClientThread 是负责下载脚本并发送运行任务的消息给任务组件的，同时它还会从 FailPoint 对象中得知需要在哪些脚本文件的哪几行加入断点，CaseWithFailPointCreator 就是负责往脚本中加断点的对象，它会在脚本被下载完成之后由 CTaskClientThread 调用。

4.7.3 错误重现子模块实现

```
.....//其他字节码命令的处理
if (Instruction == INSTRUCTION_FAILPOINT) {
    if ((mRegFlag & VM_FLAG_DEBUG_FAIL) != 0) {//判断当前是否是重现模式
        mCurState = ESTATE.ST_DEBUG;
    }
}
```

图 4.28 类 CVM 的相关方法

如图 4.28，CVM 在执行解释器相关功能时，在解析到断点相关的字节码的时候，会区分出是为了错误重现而自动加入的断点，还是用户自己在编辑脚本时加入的调试断点，然后在判断一下 CVM 的标志位决定是否需要在断点处进行调试，这个标志位是 CVM 一开始初始化时根据接收到的运行消息来设定的，debug 模式和错误重现模式可以同时设置。当判断出需要进行调试以后，CVM 的状态就会被切换到 debug 状态，这里无论是调试断点还是错误重现断点都是一样，这样可以直接复用已有的 debug 模块，而且不会干扰原有脚本的 debug 功能。

4.8 本章小结

本章主要描述了远程通讯模块、远程串口控制模块、远程 SUT 控制模块、远程 Switcher 控制模块、远程控制台控制模块和错误重现子模块，通过时序图和类图对这些模块的详细设计进行了说明，然后通过其中的关键代码展示了这些模块的具体实现细节。

第五章 总结与展望

5.1 总结

为了满足随着信息化高速发展而越来越迫切的自动化测试的需求，设计和开发了自动化测试系统云平台，为了支持使用云平台用户在远程调试脚本，本文设计和实现了云平台的远程调试子系统。本文首先介绍了子系统的项目背景，然后介绍了和子系统相关的自动化测试和远程控制技术的研究现状。

然后，本文介绍了为了支持实时浏览器双向通讯而使用的 WebSocket 通讯协议，以及为了支持 WebSocket 以及 ITS 本地的原 Java 语言而选择的 Netty 通讯框架，以及在浏览器端为了复用原有 Java 而使用的 Java Applet 技术。

随后，对子系统以及子系统依赖的云平台的结构进行的介绍，并对子系统的需求进行了分析，分析得到的用例以用例图和用例描述表的形式展现。在用例的基础上进行了概要设计，通过组件图和模块图介绍了子系统的各个组件、模块以及它们之间的关系。

最后，对远程控制的串口、SUT、Switcher 和控制台的子模块以及调试的错误重现子模块通过时序图和类图介绍进行了详细设计，并以关键性代码展示的方法描述了实现的细节。

5.2 进一步工作展望

远程调试子系统还有进一步的发展空间，从整体结构来说，进一步工作可以考虑将目前通过 Java Applet 实现的远程控制界面全部移植到浏览器上，采用 Html5 和 JavaScript 等技术实现，这样一方面方便本机并没有 JVM 或者 JRE 的用户运行，另一方面也绕过了 Java Applet 的繁琐的安全机制。从细节上来说，远程 SUT 的图像传输可以从目前采用的单帧传输变成采用传输视频流的方案，这样可以降低对带宽等资源的占用，使得远程 SUT 控制更加流畅，显示的画面也更加清晰，响应速度也更加快。

参 考 文 献

[梁家安, 2011] 梁家安，*自动化软件测试技术研究*，硕士论文，江南大学，2011。

[应杭, 2006] 应杭，*软件自动化测试技术及应用研究*，硕士论文，浙江大学，2006。

[陆璐等, 2006] 陆璐，王柏勇，*软件自动化测试技术*，清华大学出版社 北京交通大学出版社，2006。

[陈绍英等, 2007] 陈绍英，刘建华，金成姬，*LoadRunner 性能测试实战*，电子工业出版社，2007。

[Richardson, 1998] Richardson T， Wood K R， Virtual network computing，*Internet Computing, IEEE*，1998，2(1): 33-38。

[庄霄, 2009] 庄霄，*VNC 穿越 NAT 相关技术的研究*，硕士论文，北京邮电大学，2009。

[Kaplinsky, 2001] Kaplinsky K V，VNC tight encoder-data compression for VNC，*Modern Techniques and Technology, 2001. MTT 2001. Proceedings of the 7th International Scientific and Practical Conference of Students, Post-graduates and Young Scientists(IEEE'2001)*，2001: 155-157。

[TeamViewer, 2015] https://www.teamviewer.com，TeamViewer GmbH，2015

[冯光午, 2013] 冯光午，赵庆明，肖庆，王军，TeamViewer 在远程系统管理中的应用，*时代教育(教育教学)*，2011，5:34-35。

[温照松, 2012] 温照松，易仁伟，姚寒冰，基于 WebSocket 的实时 Web 应用解决方案，*电脑知识与技术*，2012，16:3826-3828。

[Fette, 2011] Fette I，Melnikov A，*The websocket protocol*，2011。

[郑强, 2012] 郑强，徐国胜，Websocket 在服务器推送中的研究，*第九届中国通信学会学术年会*，2012，376-381。

[易仁伟, 2013] 易仁伟，*基于 WebSocket 的实时 Web 应用的研究*，硕

士论文，武汉理工大学，2013。

[李代立, 2010] 李代立，陈榕，WebSocket 在 Web 实时通信领域的研究，*电脑知识与技术*，2010，28:7923-7925。

[Chun, 2013] Chun B G，Condie T，Curino C，Reef: Retainable evaluator execution framework，*Proceedings of the VLDB Endowment*，2013，6(12): 1370-1373。

[Schmidt, 1995] Schmidt D C， Using design patterns to develop reusable object-oriented communication software，*Communications of the ACM*，1995, 38(10): 65-74。

[蒋思佳, 2012] 蒋思佳，杨志义，张兵，基于模式的通信服务器设计与实现，*科学技术与工程*，2012，3:578-585。

[Hammerton, 2013] Hammerton M，Trevathan J，Myers T，Optimising data transmission in heterogeneous sensor networks，*World Academy of Science, Engineering and Technology*，2013，81: 607-615。

[郑建军, 2013] 郑建军，Java 在高并发网络编程中的应用，*电脑编程技巧与维护*，2013，18:50-60。

[金志国, 2014] 金志国，李炜，基于 Netty 的 HTTP 客户端的设计与实现，*电信工程技术与标准化*，2014，1:84-88。

[雷明剑, 2007] 雷明剑，*Java Applet 技术在网络管理中的研究及应用*，硕士论文，重庆大学，2007。

[李波, 2014] 李波，杨弘平，吕海华，*UML2 基础建模与设计实战*，清华大学出版社，2014。

[谭云杰, 2012] 谭云杰，*大象：Thinking in UML*，中国水利水电出版社，2012。

[弗里曼, 2007] 弗里曼，*Head First 设计模式*，中国电力出版社，2007。

[Erich Gamma, 2007] Erich Gamma，Richard Helm，Ralph Johnson，刘建中等译，*设计模式：可复用面向对象软件的基础*，机械工业出版社，2007。

致　　谢

在本论文完成之际，我要向所有帮助过我的老师、同学还有同事表示衷心的感谢！

首先，我要感谢我的指导老师邵栋老师，毕业论文的完成离不开邵栋老师的谆谆教导。从论文的最初选题，到论文提纲的拟定，到论文各级目录的确定，到论文内容的编写直至最终的完成，都得到了邵栋老师的关心和教诲。邵栋老师严谨踏实的学术作风、勇于创新的科研精神以及亲切和蔼的待人方式，都给我留下了深刻的印象，受益匪浅。

然后，我要感谢在公司同一项目组一起实习的几位同学，以及组里的各位前辈。感谢他们在项目开发阶段带领我熟悉项目，并在我遇到难题时提供技术方面的帮助。非常怀念和大家一起奋斗、攻克难题、调试错误直至完全整个项目的时光，从中我学到了很多。

最后，我要感谢其他所有帮助过我的老师和同学们。同时，向评阅论文和论文答辩委员会的各位老师致敬，感谢你们在这期间的付出。

版权及论文原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期： 年 月 日