AnywhereAnyway
软件概要设计说明

版本 1.0

撰写：中山大学计算机系 2001 级 AnywhereAnyway 项目组

王　萌 (Silentfish)

许洁舟 (SemiSleep)

索引

1.引言

1.1 编写目的

为指导项目 Anywhere Anyway 的开发与测试，保证项目开发不偏离原始需求，由 Anywhere Anyway 开发小组编写此概要设计说明书，并将在此说明书的基础上进行精细设计。

本概要设计说明预期读者为开发小组内部成员以及关注 Anywhere Anyway 软件功能或关注该项目开发的组外人员。

本概要设计说明书将作为 Anywhere Anyway 项目开发的永久开发文档保存。

1.2 项目背景

(1)项目名称：Anywhere Anyway。

(2)项目发起：

本软件项目由中山大学计算机科学系本科 2001 级 Anywhere Anyway 开发小组提出并由该开发小组负责设计开发。

(3)软件项目基本描述

本软件项目旨在提供一套跨平台的远程桌面访问控制系统(Platform-independent Remote Desktop Access & Control System)，整个系统包括三个子系统 Client、Server、Proxy，Client 和 Server 构成 C/S 体系结构。计算平台涉及普通 PC 工作站、大型计算机、PDA 和手机等移动信息设备。

本软件项目属一般应用软件范畴，为桌面系统提供多平台的远程访问。该项目各组件将运行于普通 PC 操作系统、大型计算机操作系统以及移动信息设备的嵌入式操作系统之上。其涉及网络计算、远程控制和多媒体编码(影像系统)等计算领域。

1.3 定义

(1)AWAW : Anywhere Anyway 软件系统

(2)Server：除特殊说明，本文档中所提到的 Server 均指 AWAW Server(AWAW 服务器),它是提供远程桌面服务的软硬件平台的总称。

(3)Client(Viewer)：除特殊说明，本文档中所提到的 Client 均指 AWAW Client(AWAW 客户端)，它是对 AWAW Server 提供桌面服务的远程访控的终端。

(4)会话(Session)：指从 Client 连接到 Server 并开始通信到通信结束这一过程。

2.SAP Implementation 概要设计

2.1 总体设计

2.1.1 需求规定

对本软件的需求规定见《需求分析》第 3 中的需求规定。

2.1.2 运行环境

对本软件的需求规定见《需求分析》第 3 中的运行环境设定。

2.1.3 基本设计概念和处理流程

考虑到本协议可能被应用到不同的网络底层协议之上，因此，我们提供了一个名称为 Connection 的接口，该接口规定了基本的数据流输入输出功能以及一些简单的数据流操作功能（比如打开、关闭等操作）。由于在应用过程总可能会有多个线程去调用同一个 Connection 的方法，我们最后为规定了两个方法已实现线程互斥，确保传输过程中某个特定内容的完整性，这两个方法分别是：void monopolize(boolean read, boolean write)（用于独占某一资源），void release(boolean read, boolean write)（释放对某一资源的占用）。在本协议实现的库中提供了一个基于 Socket 连接的 Connection 类，库的调用者可以自行为其他的网络连接方式实现 Connection 接口。

考虑到协议的将来可能会有的变动或扩充，我们希望本库能具有较强的可扩展性，以支持一些无法预料的变动。为此，我们提供了一个 Message 接口，所有实现了这一接口的类都可以作为协议消息进行传送。

最后，我们采用一个 Adapter 类对所有已知的 Message 和 Connection 进行管理。这三者的相互关系如图所示：

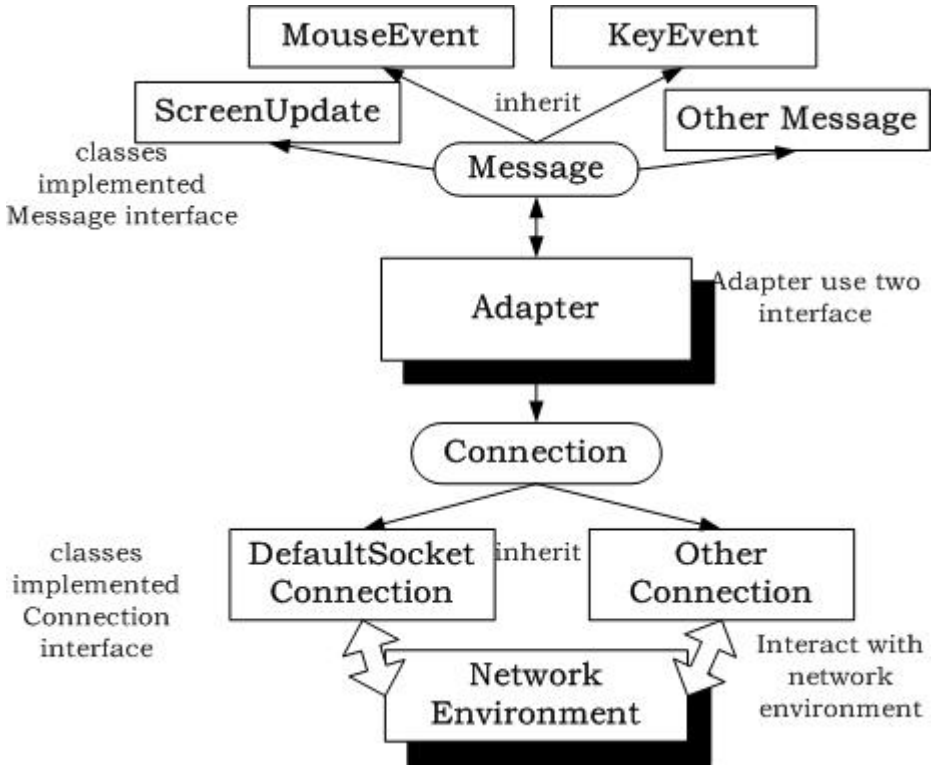

图 2-1

2.1.4 结构

下表表示本系统中系统元素的划分：

类/接口名称	功能	使用者	使用的类/借口
Connection	提供抽象的数据流传输接口，数据流操作接口，线程互斥接口	客户程序 Adpater	
Message	提供抽象的协议消息接口 public void transmit()	客户程序 Adapter	
Adapter	对多个实现了 Message 接口及 Connection 接口的	客户程序	Connection

	对象进行管理，为客户程序提供便捷的调用界面		Adapter

2.2 接口设计

以下是主要的接口列表：

模块名称	接口名称	接口说明
Adapter	void sendProtocolVersion(String)	发送协议版本号
Adapter	String readProtocolVersion()	读取协议版本号
Adapter	void sendUpdateRequest(Rectangle, Boolean)	发送屏幕更新请求
Adapter	Rectangle readUpdateRequest()	读取屏幕更新请求
Adapter	void sendUpdate(Rectangle[])	发送屏幕更新
Adapter	Image readUpdate()	读取屏幕更新
Adapter	void sendKeyEvent(int,int)	发送键盘事件
Adapter	int readKeyEvent()	读取键盘事件
Adapter	void sendPointerEvent(Point,int)	发送鼠标事件
Adapter	Point readPointerEvent()	读取鼠标事件

3.Server 概要设计

3.1 总体设计

3.1.1 需求规定

对本软件的需求规定见《需求分析》第 4 节的需求规定。

3.1.2 运行环境

对本软件的需求规定见《需求分析》第 4 节中的运行环境设定。

3.1.3 基本设计概念和处理流程

本软件的用户界面属于互交式界面，因此我们在软件整体架构上采用了经典的 MVC 设计模式。即在软件的架构上，功能模块（模型）与其用户界面（视图）是相互独立的，功能模块的状态从根本上决定了用户界面的表示，用户对功能模块的控制通过触发特定控制模块（控制器）完成。MVC 设计模式的图示如下：

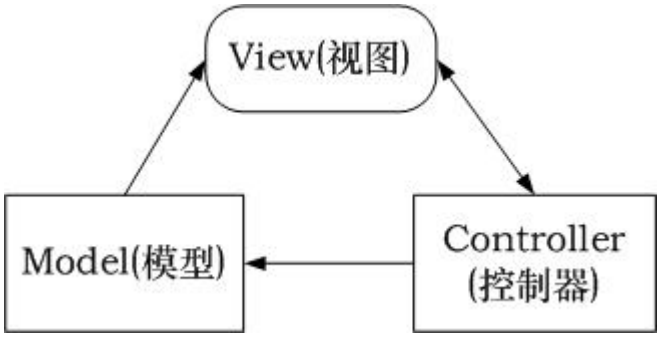

图 3-1

由于本软件的功能方面较为复杂，我们有必要对其功能进行一次分类和重新组织。我们将功能分为两大类：1.核心功能；2.扩展功能。对核心功能作如下树状划分：

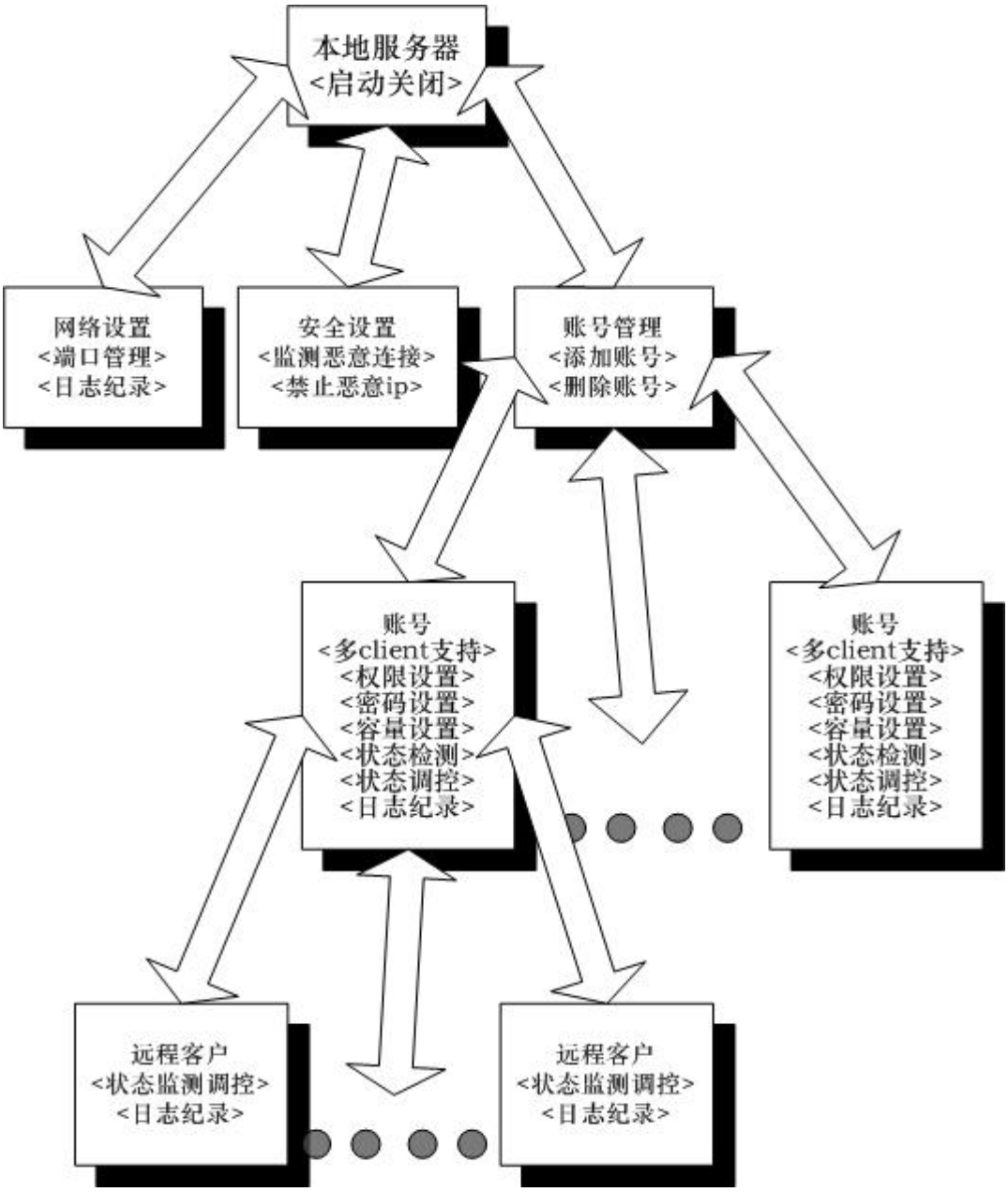

图 3-2

对于扩展功能有以下列表：

编号	功能名称	说明	备注
1	输入输出标准化	包括日志，软件设定等等的记录标准化	
2	本地语言支持	随时更变语言，能方便的添加新的语言支持	
3	跨平台支持	具有很强的可移植性，能够运行于各个不同的操作系统	
14	可扩展性	能够方便的添加新的功能而不影响原来的架构	

因此，我们将核心功能分为本地服务器，网络设置，安全设置，账号管理，账号，远程客户这六大模块，每一模块都是用 MVC 模式进行设计，模块之间的连接调用采用图中的所

示的树状结构。为了在用户界面上对各个核心功能模块进行组织，同时支持用户对扩展功能的互交操作，我们考虑使用一个界面框架模块。各模块之间的关系如图：

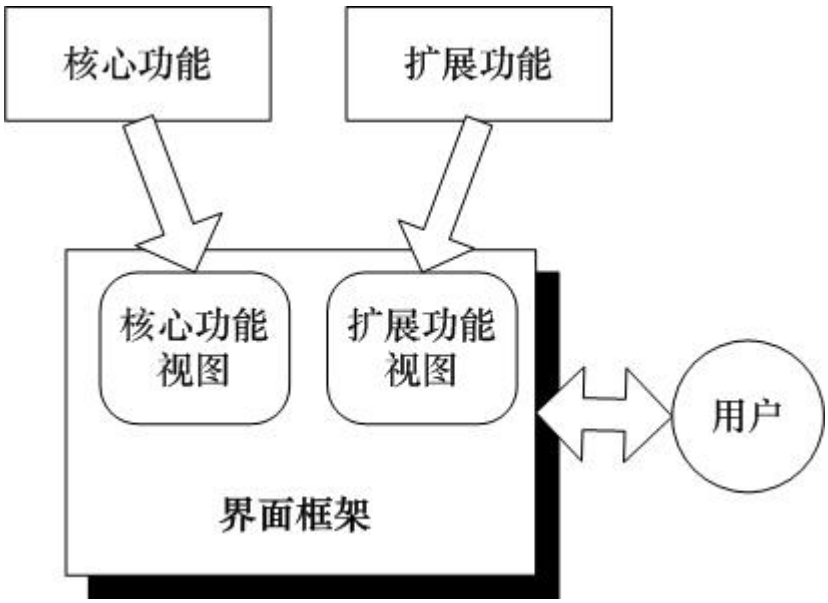

图 3-3

3.1.4 结构

下表表示本系统中系统元素的划分：

类名称	功能	调用者	直接使用的类
LocalServer	控制所有核心功能模块的启动和停止	UIFramework 主线程	Network Security AccountManager UI
Network	管理网络端口，获得网络连接	LocalServer	UI
Security	安全设置模块	LocalServer	UI
AccountManager	账号管理模块	LocalServer	Account UI
Account	账号模块	AccountManager	CurrentUser UI
CurrentUser	远程客户模块	Account	UI
UI	各个功能模块的界面	UIFramework UI	Controller
Controller	界面与功能的控制模块	UIFramework	UI
XML	负责标准化的输入输出模块	LocalServer	
I18N	负责语言本地化的模块	UI UIFramework	XML

3.1.5 功能需求与程序的关系

	Local Server	Network	Security	Account Manager	Account	Current User	UI	Action	XML	I18N
屏幕获取						O				
剪切板获取						O				
响应鼠标						√				
响应键盘						√				
剪切板更新						√				
使用端口		√								
禁止 ip			√							
多 client 支持				√	√					
账号管理				√	√					
权限设置					√	√				
密码设置					√					
容量设置					√					
状态检测					√	√				
状态调控					√	√				
日志纪录		√			√	√				
输入输出标准化									√	
本地语言支持										√

3.2 接口设计

3.2.1 用户接口

本软件采用的是互交式界面，用户通过使用鼠标键盘操纵本软件，详细的操作说明见本软件的使用说明书。

3.2.2 外部接口

本软件使用 SAP 协议(Simple Access Protocol)实现远程通讯，在操作系统层面上，本软件要求该操作系统已经安装 java 虚拟机。

3.2.3 内部接口

各个模块的主要接口见下表：

模块名称	接口名称	接口说明
LocalServer	setRunning(boolean)	开启或关闭服务器
LocalServer	isRunning()	查询服务器是否开启
LocalServer	setAutoRun(boolean)	设置服务器是否自动运行
LocalServer	isAutoRun()	查询服务器是否自动运行
LocalServer	setRunning(boolean)	开启或关闭服务器

Network	getAdapter()	获取一个远程连接
Network	getConnectionManager()	获取一个负责管理网络连接的对象
Network	getTimeout()	获取一个负责连接超时处理的对象
Network	getLogManager()	获取一个负责网络日志管理的对象
Sercurity	checkIn(RFBServerAdapter)	对新连接进行安全检测
Sercurity	getAttackCatcher()	获取一个负责捕获恶意网络连接的对象
Sercurity	getTempBlockList()	获取暂时的恶意 ip 列表
Sercurity	getBlockList()	获取用户自定的恶意 ip 列表
AccountManager	accept(RFBServerAdapter)	接受一个新的网络连接
AccountManager	addAccount(String)	添加新的帐号
AccountManager	removeAccount(Account)	删除已有的帐号
AccountManager	getAccount()	获取已有的帐号
Account	addNewUser(RFBServerAdapter)	接受远程连接
Account	getAccountSettings()	获得一个负责实现账号设定的对象
Account	getUserActivities()	获得一个负责实时管理远程客户的对象
Account	getUserQuantityChecker()	获得一个负责监测客户数量的对象
Account	getLogManager()	获取一个负责账号日志管理的对象
CurrentUser	start()	开始与远程客户进行交互
CurrentUser	freeze()	暂停与远程客户进行交互
CurrentUser	resume()	恢复与远程客户进行交互
CurrentUser	isFreeze()	查询远程客户进行交互是否暂停
CurrentUser	destroy()	强制中止与远程客户进行交互
CurrentUser	getInfo()	获取一个记录连接信息的对象
CurrentUser	setShare(boolean)	设置远程客户是否独占使用该帐号
CurrentUser	isShare()	查询远程客户是否独占使用该帐号
CurrentUser	getLogManager()	获取一个负责远程客户日志管理的对象
CurrentUser	getClipboard()	获取一个负责管理远程剪切板的对象
CurrentUser	getDesktop()	获取一个负责管理远程桌面的对象
CurrentUser	getClient()	获取一个负责管理远程客户的对象
UI	getView(Model)	获得对应模块的视图
Controller	getAction(String, UIFramework)	获得扩展功能模块的控制器
Controller	getOperations(Model, View)	获得核心功能模块的控制器
Controller	updateOperations(Model, View)	更新核心功能模块的控制器
I18N	getString(String)	查询获得对应项目的字符串
I18N	getIcon(String)	查询获得对应项目的图标
XML	loadSettings(LocalServer)	载入软件设置
XML	saveSettings(LocalServer)	保存软件设置

3.3 运行设计

以下列出几种比较重要的动态过程的实现方式。

(1)新建连接时对应的动态过程：

首先 LocalServer 从 Network 获取新连接，然后交由 Security 检测是否接受新连接，通过检测后进行密码验证，如果通过则由 Account 创建一个新的 CurrentUser，并由其负责与远程客户交户。

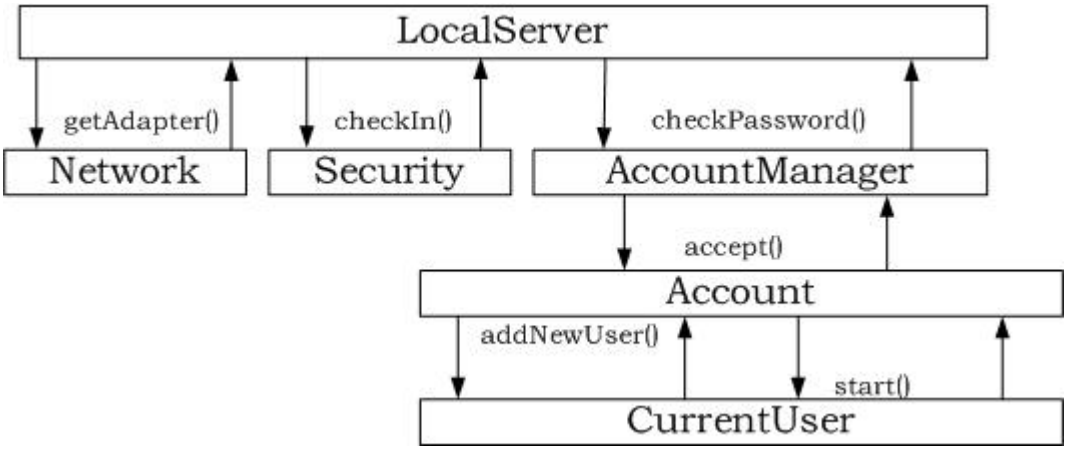

图 3-4

(2)启动软件时对应的动态过程：

首先由 LocalServer 按照上面所叙述的树状结构结构依次初始化各个核心功能模块，然后从固定存储器中读取设置文件，通过 XML 模块对各个核心功能模块的属性进行设置。

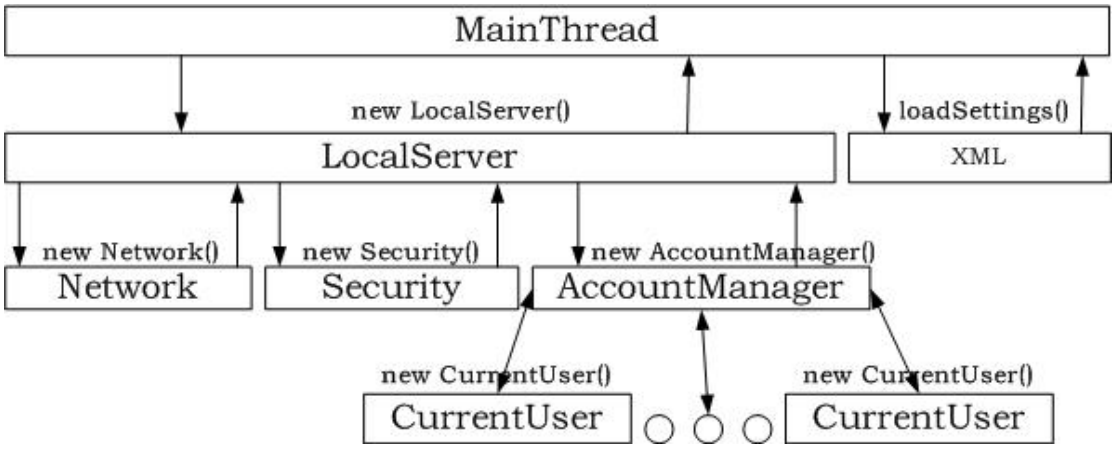

图 3-5

3.4 系统数据结构设计

系统中所有数据输出将以 XML 的形式输出。分别有三类文件：用户配置文件，语言设置文件，日志文件。相关的数据存储格式参见软件目录中的 settings.dtd、res.dtd、log.dtd。

3.5 系统出错处理设计

以下列出错类型与处理策略：

编号	出错类型	处理策略

1	网络断开	自动释放相关远程客户资源，并将此事件纪录
2	协议信息出错	自动断开开远程客户，释放相关资源，并将此事件纪录
3	用户配置错误	提示用户，并自动修改错误信息
4	系统接口出错	提示用户，自动退出程序运行

4.Client 概要设计(桌面部分)

4.1 总体设计

4.1.1 需求规定

AWAW 桌面 Viewer 的需求规定参看《需求分析》第 5 小节。

4.1.2 运行环境

(1)设备

AWAW Client 运行硬件环境范围很广，包括普通 PC 机、大中小型计算机，MAC 工作站等。下面列出基本硬件环境：

A.处理器要求：最小 166MHz 主频(Pentium 系列)。

B.内存储器要求：最小 32MB RAM

C.外储器要求：

D.外设要求：配置标准鼠标及键盘。

(2)支持软件

理论上应可运行所有操作系统平台（所有支持 Java 运行环境）的操作系统。目前此类操作系统包括：Microsoft® Windows 系列、Sun® Solaris 系列、Unix®系列、Linux 系列以及 MAC OS、BSD 系列等等。

所用到的编译软件及运行环境为 Sun® Java 运行环境 1.4.1 版本。

4.1.3 基本设计概念和处理流程

AWAW Viewer 的基本设计概念是控制中心——会话监视器(Control Center---Session Monitors)架构。这个架构的核心思想是由控制中心作为整个软件的总体管理模块，所有会话进程都由其创建并统一管理。这样，保证了不同会话间互不影响，一个或若干个运行会话进程不影响新会话的创建和运行（如图 4-1 所示）。

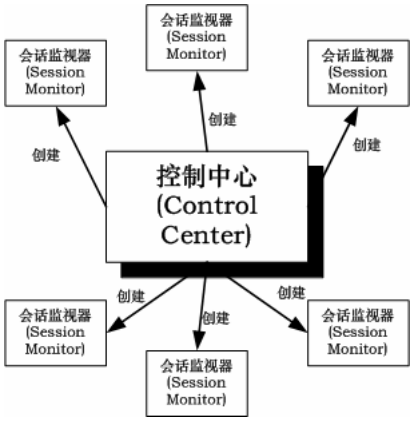

图 4-1 AWAW Viewer 基本架构：Control Center---Session Monitors 架构

由 Control Center 引发出会话库(Session Library)的概念。会话库是会话创建的基础和来源，它记录着用户曾经配置保存的会话设置（包括远程主机网络地址、端口、认证安全口令和相关性能参数的配置），控制中心根据从会话库读取出的配置创建会话。因此一个会话的创建过程如图 4-2 所示。

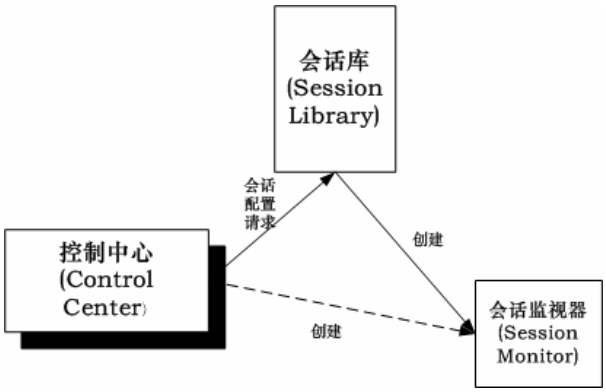

图 4-2 会话库与控制中心在创建会话中的作用

会话(Session)是 AWAW Viewer 中的核心概念，对会话的架构我们采用经典 MVC 设计模式。即在架构上，Session 的功能模块（模型）与其监视界面（视图）是相互独立的，功能模块的状态从根本上决定了监视界面的表示，用户对功能模块的控制通过触发特定控制模块（控制器）完成。MVC 设计模式的图示如图 4-3 所示。

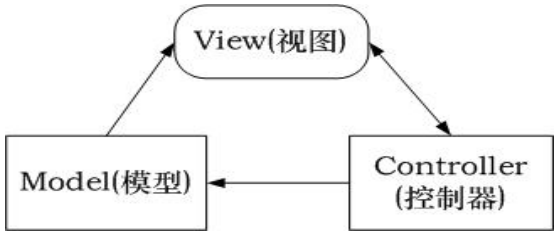

图 4-3 MVC 设计模式架构

基于 MVC 模式并结合 Session 的实际需求，功能模型命名为 SessionModel，视图命名为 SessionView，控制器命名为 SessionEventAdapter。于是，Session 的架构如图 4-4 所示。

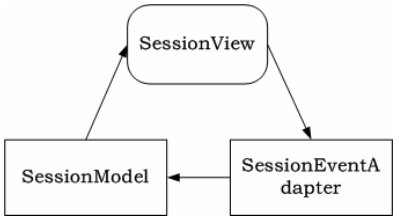

图 4-4 Session 的 MVC 架构

基于软件配置标准化的需求，设计软件配置和会话配置采用 XML 文档进行标准化数据存储。为此，分别设计 GlobalConfigurationManager 和 SessionLibraryConfigurationManager 两个模块负责软件全局和会话库配置的标准化，它们的功能就是从配置 XML 文档中读出配置和将当前配置或用户需要存储的配置写入 XML 文档。

基于软件国际化的需求，设计使用 XML 文档进行不同语言环境资源（语言）的存储，并设计模块 ResourceBurdle 用以进行环境资源的读取。

综合以上基本设计概念，我们得到 AWAW Viewer 概要设计架构图如图 4-5 所示。

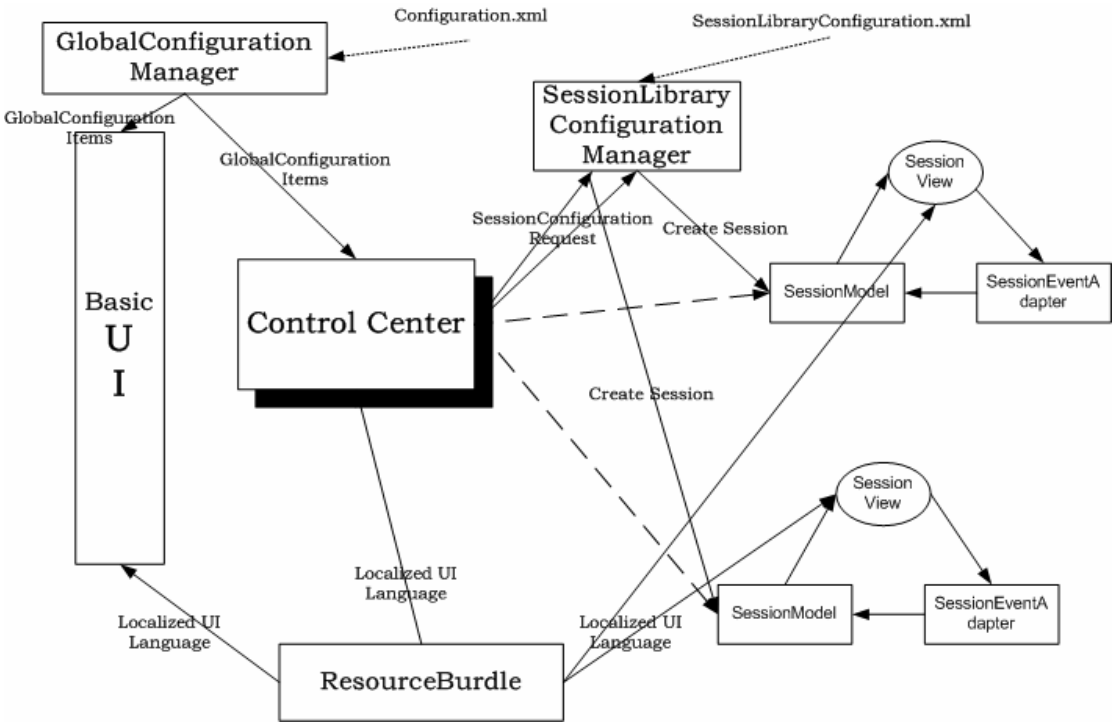

图 4-5 AWAW Viewer 整体架构

4.1.4 结构

以下用表格形式列 AWAW Viewer 系统主要模块的划分、功能、标识符以及相互调用关系。

编号	标识符	功能描述	调用者
1	ControlCenterUI	控制中心，除 2.3 中描述的功能外，还充当主用户界面	主程序进程
2	SessionLibraryPanel	会话库及其用户界面，负责会话的创建	ControlCenterUI
3	SessionLibraryItem	会话库中表征一个会话的配置	SessionLibraryPanel
4	SessionLibraryItemUI	SessionLibraryItem 的用户界面表示，用以接收用户命令并同志 SessionLibrary 创建会话	SessionLibraryPanel
5	SessionModel	会话的功能的封装抽象模块，封装了会话的全部过程，并对外提供会话启动、暂停、终止等外部服务接口。	SessionLibraryItem 、SessionLibraryPanel
6	SessionView	会话监视器(视图)的封装抽象模块，将远程屏幕信息呈现于本地系统	SessionModel
7	SessionEventAdapter	会话事件控制器，响应用户事件（包括会话控制级命令、本地鼠标事件、本地键盘事件等等）	SessionView
8	RemoteFramebuffer	远程信息本地绘制的画板，内嵌于 SessionView	SessionView

9	SessionProperties	用于保存一个运行中的会话的相关属性信息，如连接信息、性能配置信息等	SessionModel
10	ResourceBurdle	用于提取本地化语言资源信息	所有用户界面模块
11	Global Configuration Manager	全局配置管理器，用于全局配置的读取和写入(XML 文档操作)	所有需要全局配置信息的模块
12	SessionLibrary Configuration Mananger	会话库配置管理器，用于会话库配置的读取和写入(XML 文档操作)	SessionLibraryItem
13	SAPAdapter	SAP 协议适配器，封装 SAP 协议相关操作，包括相关协议报文的接收、发送和处理，向上为 SessionModel 提供协议相关服务	SessionModel
14	RemoteConnection	封装远程网络连接操作，包括从网络连接中获取输入输出流，从输入流读取信息和项输出流写入信息等操作，向上为 SAPAdapter 提供基础网络流传输服务	SAPAdapter

下面对其中若干模块的关系以图形方式澄清。

SessionModel、SAPAdapter 和 RemoteConnection 三者的关系如图 4-6 所示。

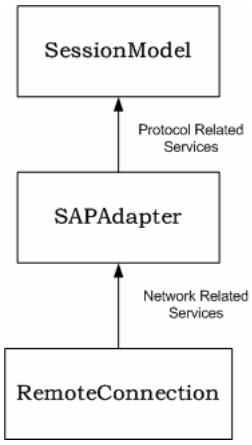

图 6 SessionModel-SAPAdapter-RemoteConnection 层次结构

SessionModel、SessionEventAdapter、RemoteFramebuffer、SessionView 之间的关系如图 4-7 所示。

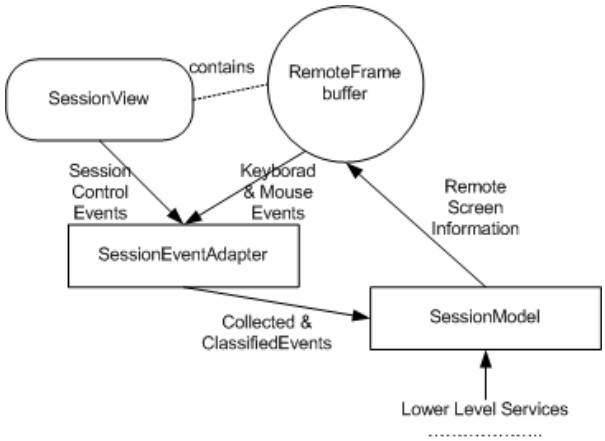

图 4-7 SessionModel-SessionEventAdapter-RemoteFramebuffer-SessionView 之间的关系

4.1.5 功能需求与程序的关系

结合需求，列出功能需求与主要程序模块之间的关系如下表所示：

	SAPAdapter	Remote Connection	Session View	Session Library	Session Event Adapter	Session Model	Control Center
协议处理功能	√						
网络通信处理功能		√					
本地绘制功能			√				
存储会话信息				√			
事件响应处理功能					√		
会话控制功能						√	√

4.2 接口设计

4.2.1 用户接口

由于 AWAW Viewer 使用图形用户界面，因此所有的用户命令均通过 GUI 组件监听，并

通过 GUI 事件监听机制以“事件命令”字符串的格式传输到软件模型内部进行处理。

4.2.2 外部接口

外部接口主要涉及网络通信的接口。首先，AWAW Viewer 需要 TCP/IP 网络协议的支持，其次还需要 SAP 协议库的支持(具体细节由 SAP 协议概要设计给出)。

4.2.3 内部接口

各模块的主要接口如下表所示：

编号	模块	接口	描述
1	SessionModel	start()	启动会话
2	SessionModel	pause()	暂停会话
3	SessionModel	terminate()	终止会话
4	SessionEventAdapter	handleEvent(Event e)	处理用户事件
5	SessionView	updateRemoteScreen()	更新本地监视器反映远程屏幕画面的变动
6	SessionLibrary	initialize()	读取会话库信息并在 GUI 上显示
7	SAPAdapter	handleSAPMessages(Message m)	处理与 SAP 库的交互
8	RemoteConnection	establish()	建立远程连接
9	SessionLibraryConfigurationManager	getSession()	从配置文件中读取会话信息
10	SessionLibraryConfigurationManager	putSession()	向配置文件写入会话信息
11	ResourceBurdle	getValue()	从语言资源文件中获取需要的本地化的语言资源

4.3 运行设计

下面列出典型的运行过程中的模块组合情况。

(1)会话创建过程

首先由 ControlCenter 向 SessionLibraryConfigurationManager 请求获取会话配置并籍此调用 SessionModel 的 start()接口启动一个会话。（如图 4-8 所示）

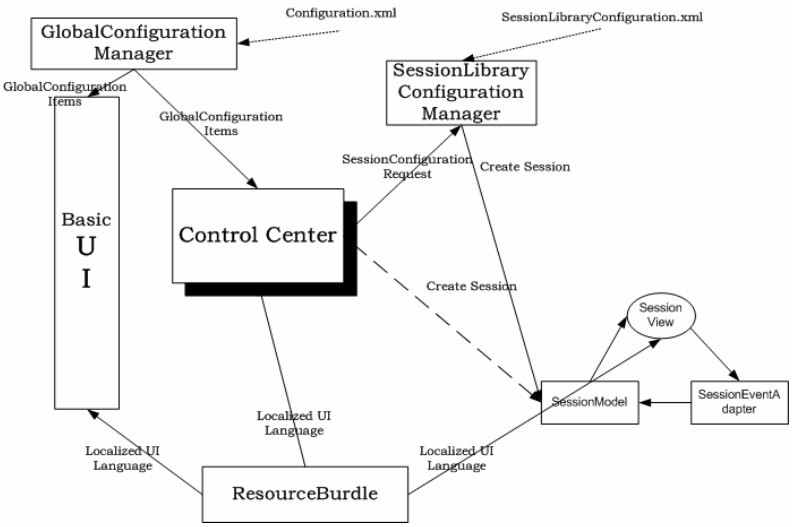

图 4-8 会话创建过程

(2)会话终止过程

首先由 SessionView 监听到用户终止命令，之后提交 SessionEventAdapter 进行分析，当获知该事件为终止命令事件时，调用 SessionModel 的 terminate()接口终止会话。(如图 4-9 所示)。

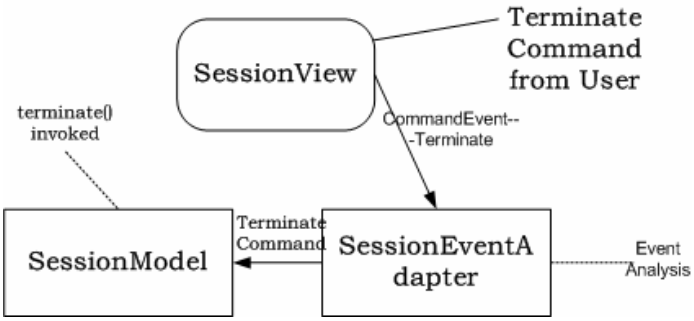

图 4-9 会话终止过程

4.4 系统数据结构设计

本部分主要涉及软件配置文件的格式及结构定义。

4.4.1 全局配置文件格式

全局配置文件采用 XML 格式存储，其 DTD 定义如下：

<?xml version="1.0" encoding="UTF-8" ?>

<!ELEMENT Configuration (UI) >

<!ELEMENT UI EMPTY >

<!ATTLIST UI locale NMTOKEN #REQUIRED >

<ATTLIST UI lookAndFeel NMTOKEN #REQUIRED >

下面是一个典型的配置文件：

```
<?xml version="1.0"?>
<!DOCTYPE Configuration SYSTEM "configuration.dtd">
<Configuration>
    UI locale="EN" lookAndFeel="default"/>
</Configuration>
```

4.4.2 会话库配置文件格式

会话库配置文件采用 XML 格式存储，其典型格式配置如下：

```
<?xml version="1.0" encoding="ISO8859_1"?>
<!DOCTYPE SessionLib SYSTEM "SessionLibrary.dtd">
<SessionLib>
    <DefaultSession name="Default Session" description="Default Session">
        <ConnectionSetting>
            <server_setting port="5900" password="1" host="localhost" />
        </ConnectionSetting>
        <EncodingList>
            <encode name="ZRLE" enabled="false" id="16" />
            <encode name="Hextile" enabled="false" id="5" />
            <encode name="CoRRE" enabled="false" id="4" />
            <encode name="RRE" enabled="true" id="2" />
            <encode name="CopyRect" enabled="false" id="1" />
            <encode name="Raw" enabled="true" id="0" />
        </EncodingList>
        <Misc view_only="false" update_frequency="0" shared="false" />
    </DefaultSession>
    <Library>
    </Library>
</SessionLib>
```

4.5 系统出错处理设计

4.5.1 出错信息设计

下面以一览表形式列出主要的出错情况对应的出错信息：

编号	出错情况	出错信息
1	网络连接失败	“网络连接失败，请检查网络连接”
2	文件读写失败	“配置文件读取/写入出错，请检查文件是否完整”
3	登陆认证失败	“未通过身份认证，请检查您输入的口令”
4	软件致命错误	“出现软件致命错误，请重新启动 AnywhereAnyway”

4.5.2 补救措施

针对文件读写失败的情况，设计配置文件的备份机制，每份配置文档均有对应的备份文件（只读属性），在配置文件意外遭到破坏的情况下可通过备份配置文件对配置进行恢复。

5.Client 概要设计（Mobile 部分）

5.1 总体设计

5.1.1 需求规定

对本软件的需求规定见《需求分析》第 6 小节中的需求规定。

5.1.2 运行环境

对本软件的需求规定见《需求分析》第 6 小节中的运行环境设定。

5.1.3 基本设计概念和处理流程

考虑到本软件是运行在各种资源（计算能力，存储空间，网络带宽等等）受限的移动设备上，为了更高效的对本地机器的资源进行利用，我们决定对其功能进行简化，精简功能数目，对核心功能进行优化，同时因应移动设备的特点提供若干的便捷功能，以方便用户使用。因此，我们将本软件的实现划分为界面模块和功能模块两部分。对于界面模块，我们将提供以下几个不同的界面：主界面，设置界面，使用界面，图像浏览器；对于功能模块，我们将其继续划分为两个不同的模块：远程通讯模块，图像解压模块。各个模块之间的关系图如图所示：

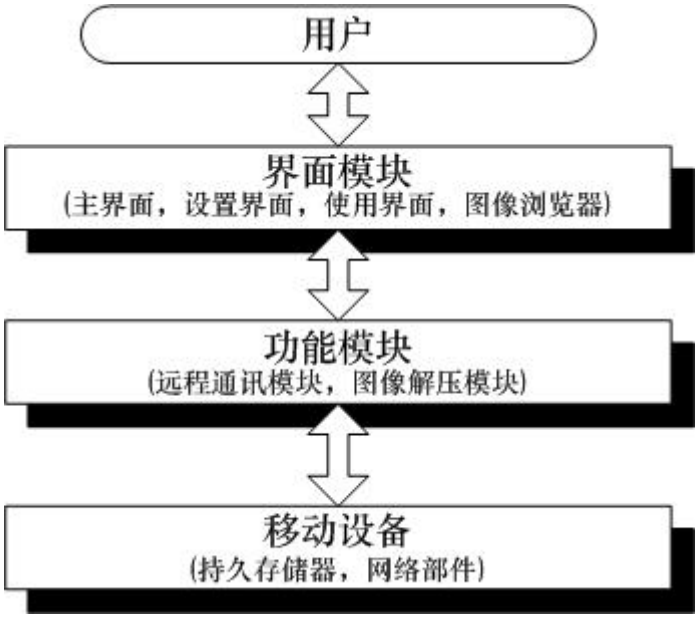

图 5-1

5.1.4 结构

下表表示本系统中系统元素的划分：

模块名称	功能	调用者	直接调用的模块
Menu	功能菜单	主线程	SettingForm RemoteFrame PicViewer
SettingForm	用户的设置界面	Menu	
RemoteFrame	与远程主机互交的界面	Menu	Adapter

			Decoder
PicViewer	用于浏览已截取的屏幕图片的浏览器	Menu	Decoder
Adapter	用于与远程主机实现协议信息的交互	RemoteFrame	
Decoder	用于将数据流解压成图片	RemoteFrame PicViewer	

5.1.5 功能需求与程序的关系

	Menu	SettingForm	RemoteFrame	PicViewer	Adapter	Decoder
屏幕获取					√	√
剪切板获取			√		√	
发送鼠标事件			√		√	
发送键盘事件			√		√	
发送剪切板更新			√		√	
连接信息自动存储		√				
屏幕截取			√	√		√

5.2 接口设计

5.2.1 用户接口

本软件采用的是互交式界面，用户通过使用移动设备提供的操作方式操纵本软件，详细的操作说明见本软件的使用说明书。

5.2.2 外部接口

本软件使用 RFB 协议(Remote Framebuffer)实现远程通讯，在操作系统层面上，本软件要求该移动设备已经安装 k-java 虚拟机。

5.2.3 内部接口

各个模块的主要接口见下表：

模块名称	接口名称	接口说明
Menu	commandAction(Command,Displayable)	响应用户命令，转入相应的界面
SettingForm	commandAction(Command,Displayable)	响应用户命令，进行对应的操作
SettingForm	getHostName()	获得主机名称
SettingForm	getPassword()	获得密码
SettingForm	getPort()	获得主机端口号
RemoteFrame	commandAction(Command,Displayable)	响应用户命令，对远程主机进行操控
RemoteFrame	run()	开始于远程主机交互

PicViewer	commandAction(Command,Displayable)	响应用户命令，浏览图片
Adapter	getServerProtocolVersion()	获取远程主机软件的版本号
Adapter	sendClientProtocolVersion()	发送本机软件的版本号
Adapter	getAuthenticationScheme()	获取远程主机的认证策略
Adapter	getText()	获取字符串
Adapter	proccessAuthentication(String)	发送认证密码
Adapter	sendShareFlag(boolean)	发送共享标志
Adapter	getFramebufferWidth()	获取主机屏幕宽度
Adapter	getFramebufferHeight()	获取主机屏幕高度
Adapter	getPixelFormat()	获取像素信息
Adapter	sendSetPixelFormat(PixelFormat)	发送像素信息
Adapter	sendSetEncodings(int[])	发送解码信息
Adapter	sendFramebufferUpdateRequest(int, int, int, int, boolean)	发送屏幕更新请求
Adapter	sendKeyEvent(Boolean, int)	发送键盘事件
Adapter	sendPointerEvent(int, int, int)	发送鼠标事件
Adapter	sendClientCutText(String)	发送剪切板文本
Adapter	getMsgType()	发送信息类型
Adapter	getFramebufferUpdate(Image, Decoder)	获取屏幕更新
Adapter	getColourMap()	获取调色板
Adapter	getServerCutText()	获取剪切板文本
Adapter	getDataInputStream()	获取该通讯模块对应输入流
Adapter	getDataOutputStream()	获取该通讯模块对应输出流
Decoder	decode(Graphics, DataInputStream)	从输入流解压出图像
Decoder	saveToByteArray(DataInputStream, ByteArrayOutputStream)	从输入流将图像存储至输出流
Decoder	setColourMap(int, int[][])	设置调色板
Decoder	setPixelFormat(PixelFormat)	设置像素信息

5.3 运行设计

以下列出几种比较重要的动态过程的实现方式。

(1)与主机进行通讯

首先 RemoteFrame 从 SettingForm 中获取连接信息，与远程主机建立连接，然后生成一个 Decoder，用于解压图像，最后生成一个 Adapter 并调用其与远程主机交互。

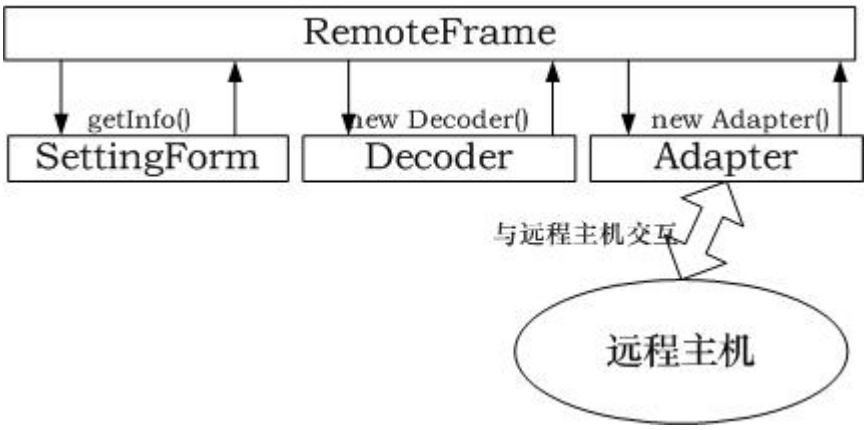

图 5-2

(2)浏览已截取的屏幕图像

首先，PicViewer 调用 java api 的 rms 库，从移动设备的持久存储器中获得输入流，然后调用 Decoder 将输入流还原成图片，最后将其显示在用户界面上。

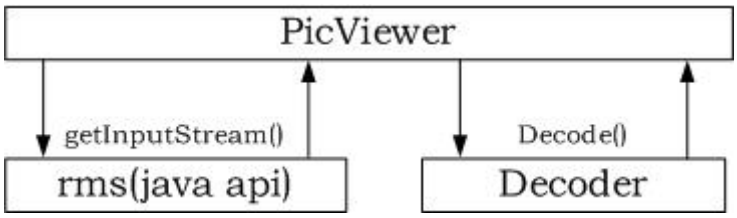

图 5-3