SHADOWSOCKS 设计分析文档

2017/12/27

目录

SHADOWSOCKS 设计分析文档

2017/12/27

项目背景和原理

Shadowsocks 是一款比较知名和应用比较广泛的翻墙软件。Shadowsocks 的代码质量很高，并且有 python 编程语言编写的版本，符合面向对象编程的要求。

Shadowsocks 的翻墙原理如下列示意图所示：

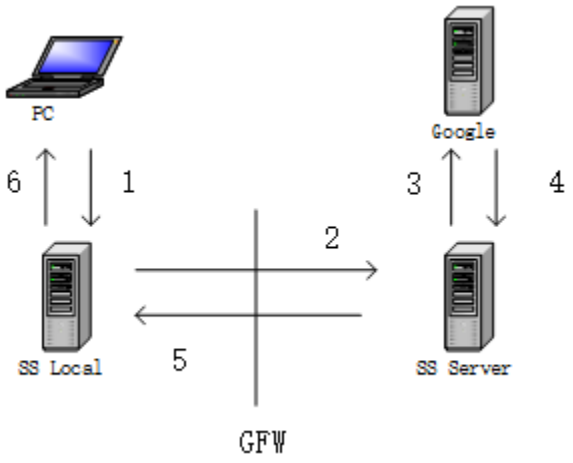


图 1. Shadowsocks 原理示意图

Shadowsocks 基于 Sock5 协议完成了一个墙内墙外的通信过程：

1. SS Local 把 PC 的数据包进行加密
2. SS Local 转发加密后的数据包到墙外的服务器 SS Server
3. SS Server 把解密后的数据包转发到目标地址
4. SS Server 把返回的数据包进行加密
5. SS Server 把加密后的数据包转发到墙内的 SS Local
6. SS Local 把解密后的数据包返回到 PC

上述就是 Shadowsocks 的基本原理。该软件实现过程涉及到网络通信、命令行交互、加密解密、IO 复用等方面，是一个有一定复杂度的软件。

项目方法

本次项目主要针对 shadoesocks 的 2.8.2 python 版本进行建模分析。在这个项目里面我们主要采用的建模方法是面向对象建模。原因主要有以下几点：

1. 本次分析的软件是用 python 语言进行编写的。而 python 是一个面向对象的编程语言，与面向对象建模及其吻合
2. 面向对象建模是目前比较流行的一种建模方法。掌握好这一方法有利于提高以后的编程能力。

因此在采用面向对象建模方法以后，主要流程包括 OOA 和 OOD 分析两个阶段。这两个阶段建立的模型分别为 OOA 模型和 OOD 模型。

OOA 模型

OOA 阶段主要发生在面向对象建模的分析阶段。OOA 模型框架的示意图如下：

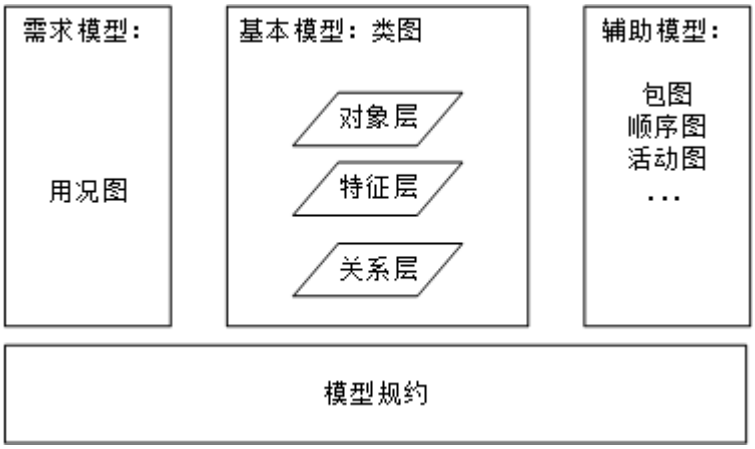


图 2. OOA 模型框架

本小节主要按照需求模型、基本模型和辅助模型的顺序一一阐述。

1. 需求模型

系统边界

在面向对象建模的方法中，第一步需要进行的就是需求分析建模。而需求模型主要是由用况图(use case diagram)和对应的文档构成。在构建需求模型之前，我们需要确定好系统边界。

在系统边界的表示上，我们首先把整个软件系统看成一个单独的系统，把软件之外的参与者先确定出来。

首先对整个系统进行边界确定。其示意图如图 3 所示。首先整个系统的参与者有以下几个：

1. 客户端：即墙内的 PC，利用本系统与墙外的某些服务器通信
2. 服务器端：即墙外的服务，利用本系统与墙内的 PC 通信
3. DNS 服务器：主要是客户端再必要的时候，通过 DNS 服务器查询服务器的 ip 地址。

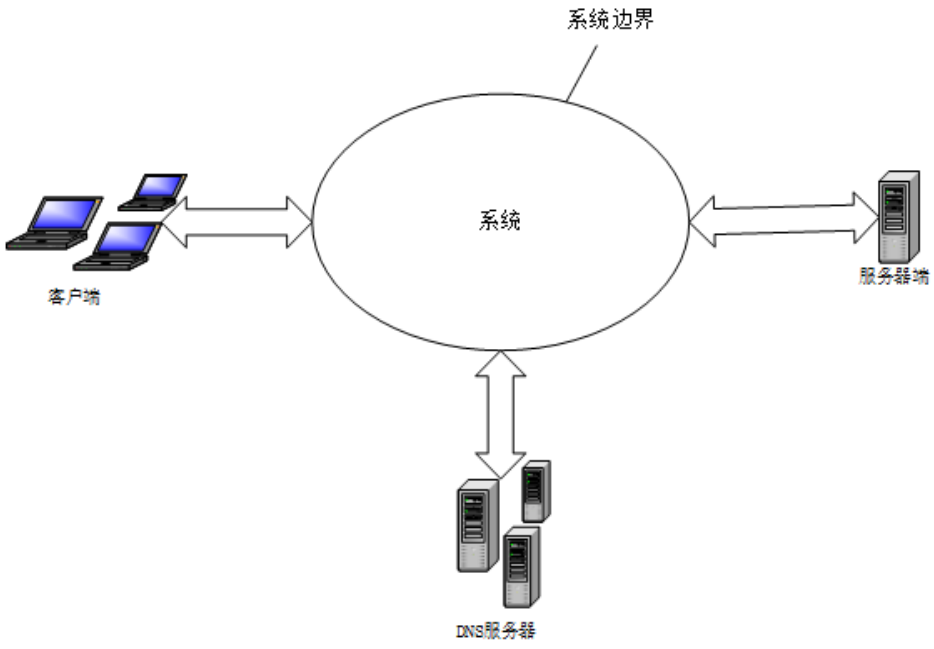


图 3. 整体系统的边界

根据以上信息建立的总体需求模型如图 4 所示下：

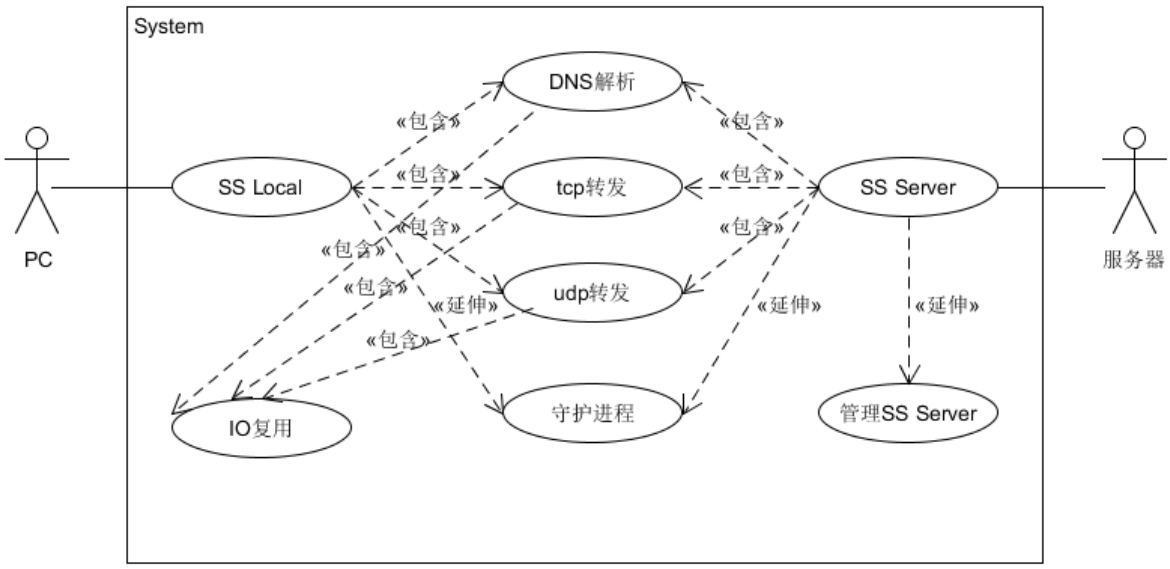


图 4 总体需求模型用况图

其中每个用况的规约如下：

用况名称	SS Local
用况编号	1

行为陈述	SS Local
	读取配置
	启动客户端进程
	while(1)
	　if DNS 解析请求 then
	　　call DNS 解析
	　end if
	　if tcp 转发 then
	　　call tcp 转发
	　endif
	　if udp 转发 then
	　　call udp 转发
	　endif
	　if 收到退出请求 then
	　　关闭 tcp、udp 连接

	end if
	if 收到终止请求 then
	关闭客户端进程
	break
	end if

用况名称	SS Server
用况编号	1

行为陈述	SS Server
	读取配置
	启动服务器端进程
	while(1)
	if DNS 解析请求 then
	call DNS 解析
	end if
	if tcp 转发 then
	call tcp 转发
	endif
	if udp 转发 then
	call udp 转发
	endif
	if 收到退出请求 then
	关闭 tcp、udp 连接
	end if
	if 收到终止请求 then
	关闭客户端进程
	break
	end if

用况名称	DNS 解析
用况编号	1

行为陈述	DNS 解析
	if 域名为空 then
	　退出并提示域名为空
	elif 域名已经是 ip 地址 then
	　直接返回结果
	elif 域名的解析结果已经存在操作系统了 then
	　直接返回结果
	elif 域名的解析结果已经缓存了 then
	　直接返回结果
	else
	　if 域名的拼写无效 then
	　　直接退出并提示域名的拼写错误
	　end if
	　向 DNS 服务器发送 DNS 解析请求
	　　等待 DNS 解析结果返回
	　返回 DNS 解析结果
	　关闭 DNS 解析请求
	end if

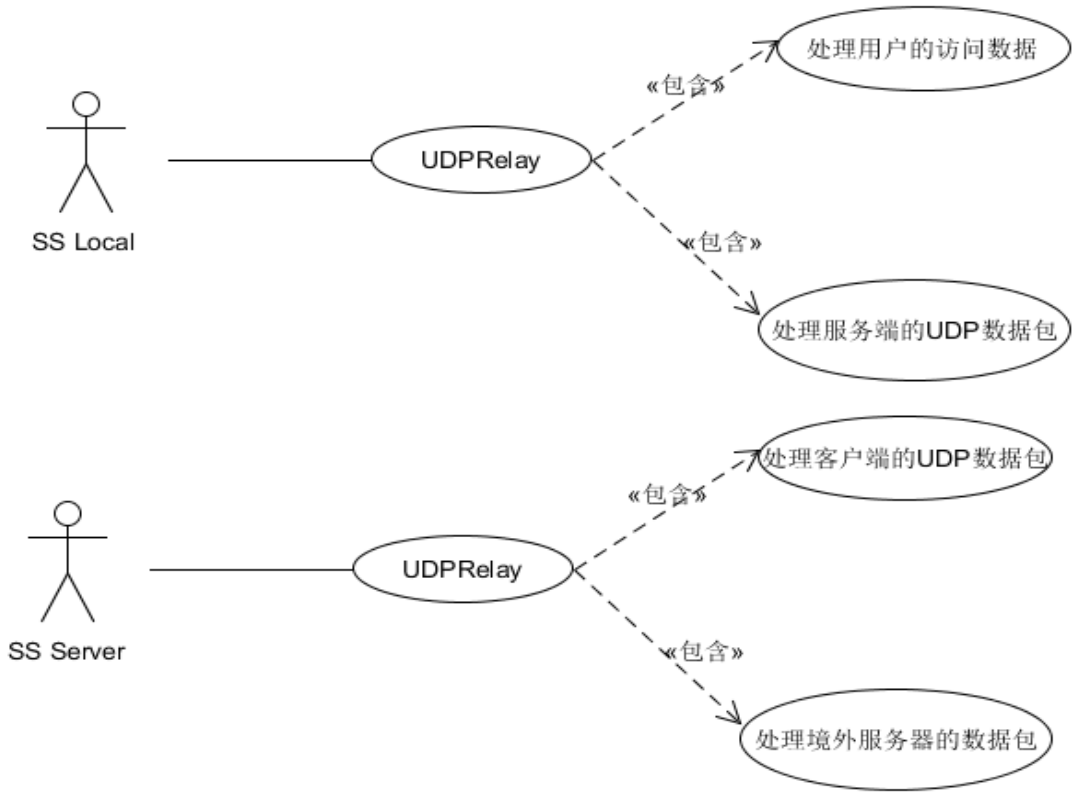
用况名称	守护进程
用况编号	1

行为陈述	守护进程
	获取配置文件
	while(1)
	　if 命令是开启守护进程 then
	　　开启守护进程
	　if 命令是终止守护进程 then
	　　终止守护进程
	　if 命令是重启守护进程 then
	　　终止守护进程
	　　开启守护进程
	　　break

用况名称	IO 复用
用况编号	1

行为陈述	IO 复用
	启动 IO 复用
	while(1)
	　if 有新的监听对象 then
	　　添加新的监听对象
	　end if
	　if 监听到新的活动 then
	　　提示活动的目标对象处理该事件
	　end if

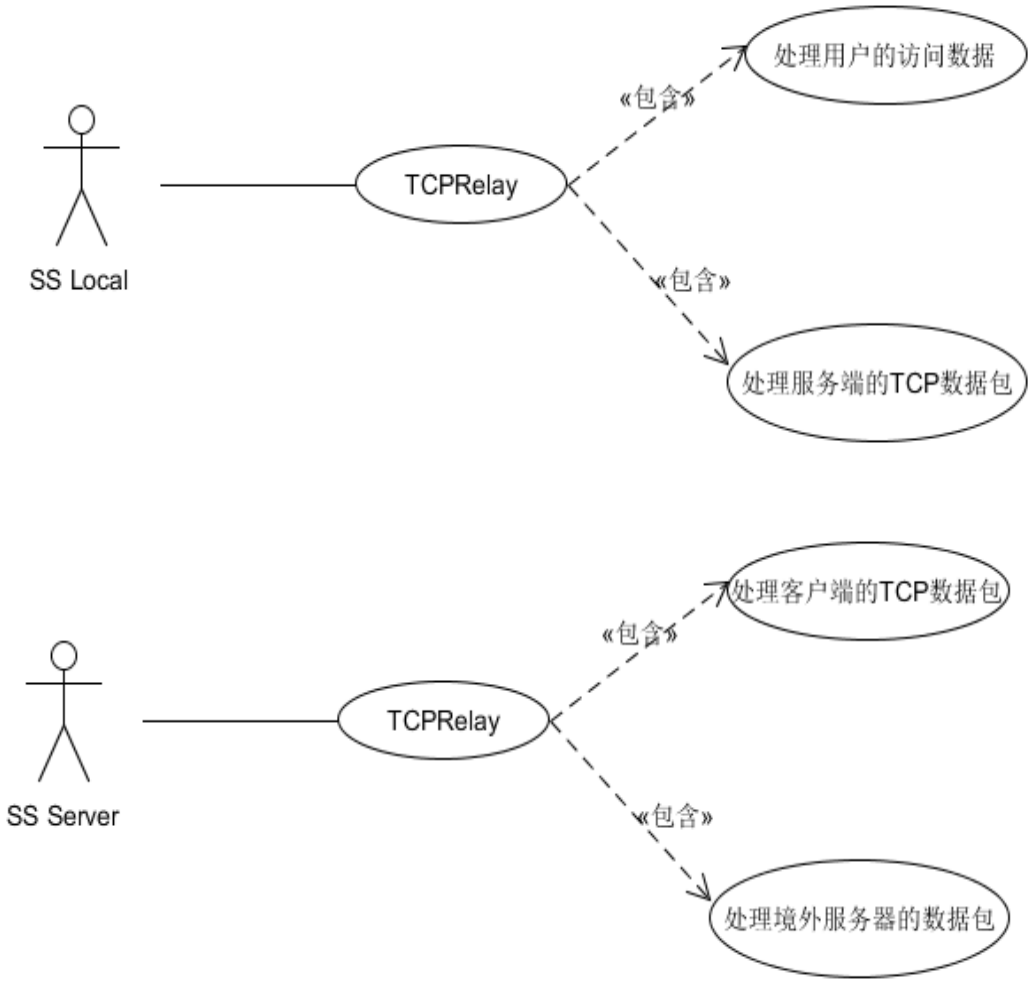
对于 tcp 和 udp 转发以及管理服务器模块, 我们将其视为三个子模块来进行分析，其用况图如下



udp 转发模块的用况分析图

用况名称	UDPRelay
用况编号	1

行为陈述	UDPRelay
	While（1）
	if 运行在服务器端 then
	if 收到来自客户端的 UDP 数据包 then
	解密包
	将数据发送到目标服务器
	end if
	if 收到来自境外服务器的数据包 then
	将数据加密
	将加密后的数据包使用 UDP 发送到目标客户端
	end if
	end if
	if 运行在客户端 then
	if 收到用户的访问数据 then
	加密数据
	将加密的数据使用 UDP 发送到服务端
	end if
	if 收到来自服务端的 UDP 数据包 then
	加密数据包
	将加密后的数据返回给用户
	end if
	end if

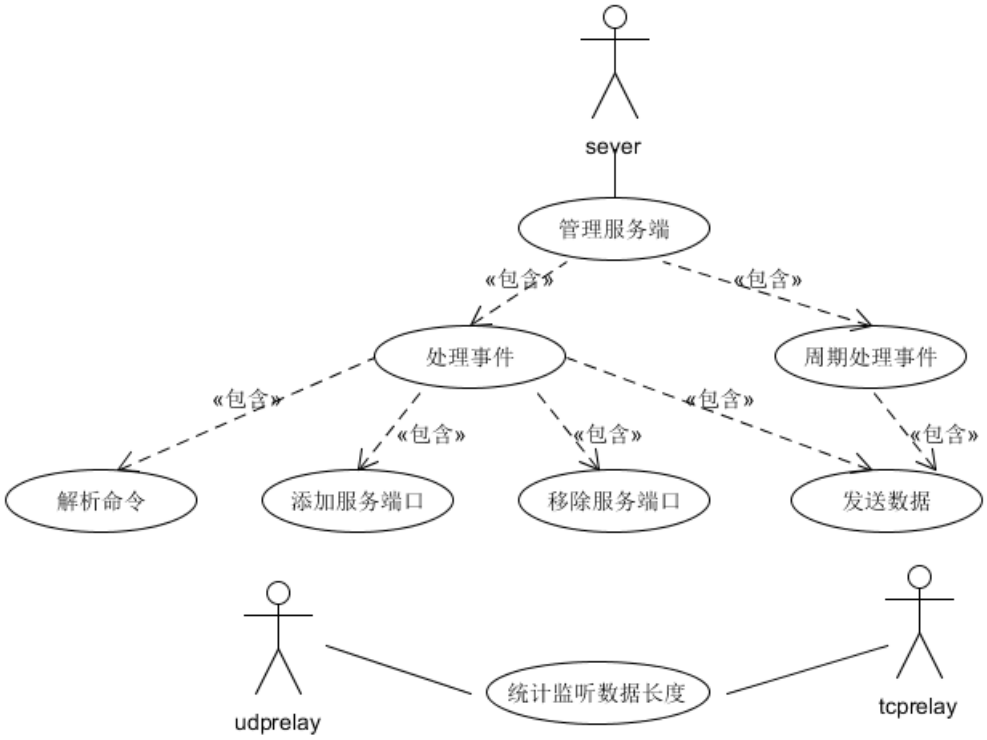


tcp 转发模块的用况分析图

用况名称	TCPRelay
用况编号	2

行为陈述	TCPRelay
	While（1）
	if 运行在服务器端 then
	　if 收到来自客户端的 TCP 数据包 then
	　　解密包
	　　将数据发送到目标服务器
	　end if
	　if 收到来自境外服务器的数据包 then
	　　将数据加密

	将加密后的数据包使用 TCP 发送到目标客户端
	end if
	end if
	if 运行在客户端 then
	if 收到用户的访问数据 then
	加密数据
	将加密的数据使用 TCP 发送到服务端
	end if
	if 收到来自服务端的 TCP 数据包 then
	加密数据包
	将加密后的数据返回给用户
	end if
	end if



管理服务器模块用况图

用况名称	添加服务端口
用况编号	1

行为陈述	添加服务端口
	从配置中获取端口号，根据端口号获得服务端；
	if 服务端已存在 then
	　　return
	end if;
	实例化 tcp 和 udp 管理，添加到事件监听的 IO 复用；

参与者	

用况名称	移除服务端口
用况编号	2

行为陈述	移除服务端口
	从配置中获取端口号，根据端口号获得服务端；
	if 服务端在管理器中 then
	　　停止监听该服务端 tcp 和 udp 数据包；
	　　删除该端口对应的 tcp 和 udp 的管理；
	else
	　　报错；
	end if;

参与者	

用况名称	解析命令
用况编号	3
行为陈述	解析命令 从数据中分出命令部分和配置部分
参与者	

1
1

用况名称	统计监听数据的长度
用况编号	4
行为陈述	统计监听数据的长度 增加对应端口的获得数据长度
参与者	udprelay, tcprelay

用况名称	发送控制数据
用况编号	5

行为陈述	发送控制数据
	if 存在客户端的地址 then
	　　发送数据
	end if;

参与者	

用况名称	周期处理事件
用况编号	6

行为陈述	周期性地向客户端发送接收到数据的总长度
	for 每个要发送的数据 then
	　　临时保存发送数据
	　　统计数量
	　　if 达到发送上限 then
	　　call 发送控制数据
	　　清除临时保存
	　　end if;
	end for;
	发送剩余的数据

1
2

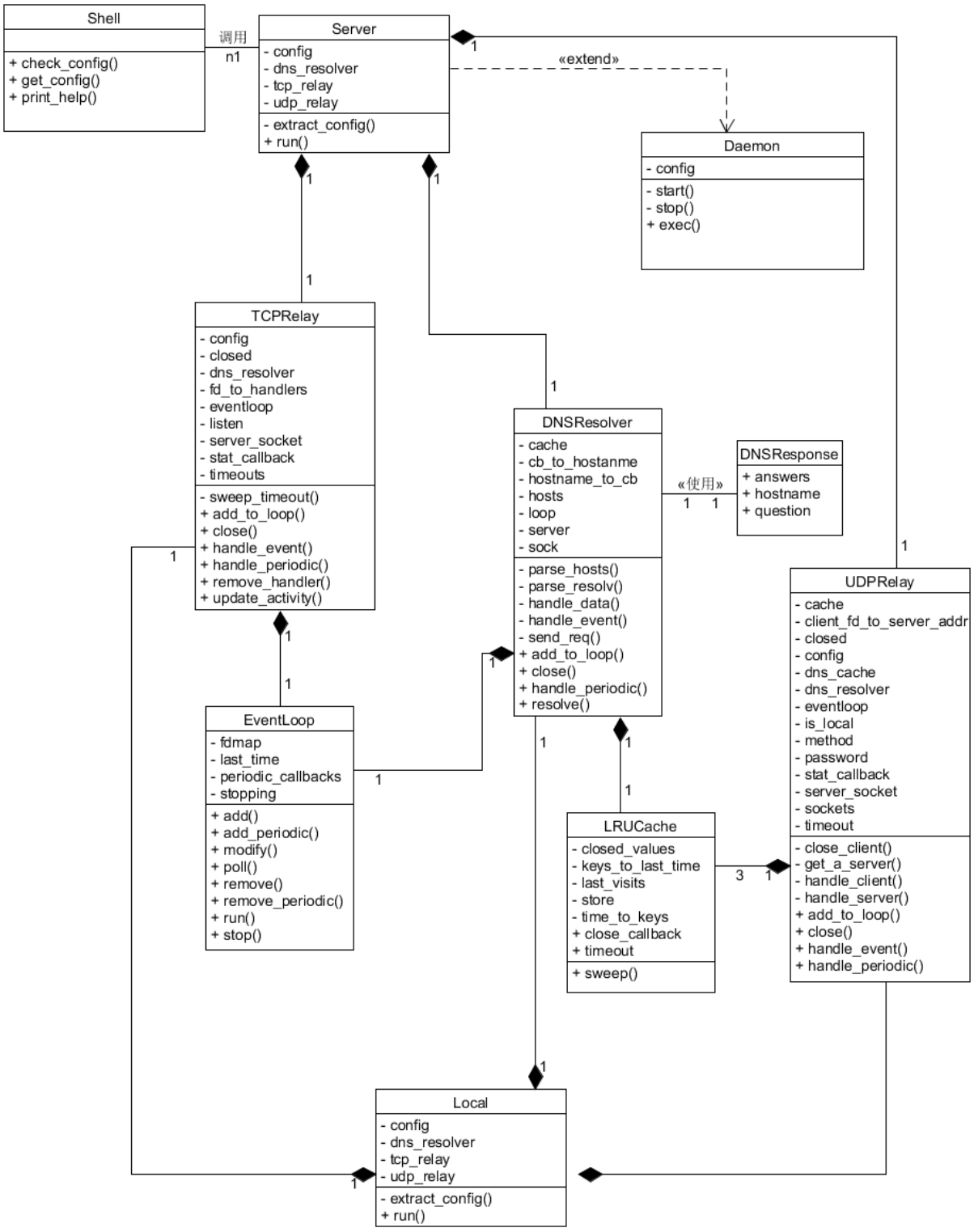
用况名称	管理服务端
用况编号	7
行为陈述	管理服务端 运行管理服务端程序

用况名称	处理事件
用况编号	8

行为陈述	处理事件
	if 有事件触发且 sock 存在 then
	获取数据和客户端地址
	call 解析命令
	if 获取到数据 then
	if 获取到配置信息 then
	更新配置信息
	end if;
	if 如果服务端口不在配置信息中 then
	报错
	else
	if 命令为‘add’ then
	call 添加服务端口
	call 发送控制数据
	elif 命令为‘remove’ then
	call 移除服务端口
	call 发送控制数据
	elif 命令为‘ping’ then
	call 发送控制数据
	else
	报错
	end if;
	end if;
	end if;
	end for;

2. 基本模型

基本模型是类图。根据总体需求用况图和三个子模块的用况图，我们也分成四个部分来展示类图



总体类图

类的总体说明		
	类名	<中文>守护进程;<英文>Daemon
	解释	这个类的功能是开启或关闭守护进程，目的是创建一个不受任何终端控制守护进程 来提供网络通信服务。
	一般类	None
	主动性	No
	持久性	No
	辅助模型	daemon_state_diagram
	其他	
属性说明		
	{	
	名称与数据类型	config :dict
	属性解释	这是一个 dict 类型的对象，包含创建守护进程的配置信息
	多态性	
	关联、聚合或组合	
	其它	
	}	
	{	
操作说明		
	{	
	特征标记	exec(config:dict)
	操作解释	该操作主要是解析 config 配置文件中的命令，选择开启守护进程，终止守护进程或 重启守护进程。
	主动性方法	被动
	多态性	
	消息发送	None
	操作流程	None
	其他	

	}	
	{	
	特征标记	set_user(username:string)
	操作解释	该操作主要是用于根据 username 设置用户，将用户设置为有效用户，非超级用户
	主动性方法	被动
	多态性	
	消息发送	None
	操作流程	None
	其他	
	}	
	{	
	特征标记	exec()
	操作解释	该操作主要是解析 config 配置文件中的命令，选择开启守护进程，终止守护进程或重启守护进程
	主动性方法	被动
	多态性	No
	消息发送	No
	操作流程	
	其他	
	}	
	{	
	特征标记	start()
	操作解释	该操作主要是开启守护进程
	主动性方法	被动
	多态性	No
	消息发送	No
	操作流程	
	其他	
	}	
	{	
	特征标记	stop()
	操作解释	该操作主要是终止守护进程
	主动性方法	被动
	多态性	No

	消息发送	
	操作流程	
	其他	
	}	

类的总体说明		
	类名	<中文>域名解析：<英文>DNSResolver
	解释	这个类的主要功能就是处理 DNS 解析请求来返回域名对应的 ip 地址。因为有可能 SS 服务器的地址是一个域名地址，所以需要相应的 DNS 解析请求模块来获取对应的 ip 地址。
	一般类	No
	主动性	No
	持久性	No
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	cache : LRUCache
	属性解释	cache 缓存了最近解析过的域名，可以减少访问域名服务器的次数并提高查找域名映射的速度。
	多态性	None
	组合	cache 是 DNSResolver 的一部分，必须得依靠 cache 才能利用缓存机制提高查找域名映射的速度。
	其它	
	}	
	{	
	名称与数据类型	cb_to_hostname : dict
	属性解释	该属性保留了每个回调操作应该对应的域名。
	多态性	None
	组合	None
	其它	
	}	

	{	
	名称与数据类型	hostname_to_cb : dict
	属性解释	该属性保留了每个域名对应的回调操作。
	多态性	
	组合	
	其它	
	}	
	{	
	名称与数据类型	hosts : dict
	属性解释	hosts 保存了操作系统中从域名到 ip 地址的映射。如果操作系统也没有缓存到从域名到 ip 地址的映射，则再调用其它方法获取映射。
	多态性	None
	关联、聚合或组合	None
	其它	
	}	
	{	
	名称与数据类型	loop : EventLoop
	属性解释	这个是一个 Eventloop 的实例，主要是为了利用 IO 复用机制来获得 DNS 解析请求返回的 结果。
	多态性	None
	关联、聚合或组合	DNSResolver 必须得依靠 Eventloop 才能利用 IO 复用机制。因此两者之前有一个紧密、牢 固的关系，适用于组合范畴。
	其它	
	}	
	{	
	名称与数据类型	server : list
	属性解释	server 里面保存了操作系统中 DNS 服务器的 ip 地址。如果操作系统没有保存有 DNS 服务器的 ip 地址，则默认为谷歌 DNS 服务器的 ip 地址。
	多态性	None
	关联、聚合或组合	
	其它	

	}	
	{	
	名称与数据类型	sock： Socket
	属性解释	sock 是 Socket 类的一个实例，负责和域名服务器通信来获取域名解析结果。
	多态性	None
	关联、聚合或组合	None
	其它	
	}	
操作说明		
	{	
	特征标记	parse_hosts()
	操作解释	该操作主要是解析本地操作系统的 hosts 文件配置，并将其中的映射信息提取到 hosts 属性 中。
	主动性方法	被动
	多态性	None
	消息发送	None
	操作流程	
	其他	
	}	
	{	
	特征标记	parse_resolv()
	操作解释	该操作主要是解析操作系统中的域名解析配置文件，并从中提取出域名服务器的 ip 地址。
	主动性方法	被动
	多态性	None
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_data(data : bytes)
	操作解释	该操作主要是对域名解析请求返回的包进行解析，并从中提取出 ip 地址。

	主动性方法	被动
	多态性	None
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_event(sock : Socket , fd : int , event : int)
	操作解释	该操作主要是响应 IO 复用的唤醒，并把域名解析请求得到数据进行处理来获得 ip 地址。
	主动性方法	被动
	多态性	None
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	send_req(hostname : bytes , qtype : int)
	操作解释	该操作主要是根据域名的类型来向域名服务器发送解析请求。
	主动性方法	被动
	多态性	
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	add_to_loop(loop : Eventloop)
	操作解释	该操作主要把该类的实例添加到 Eventloop 的实例中监听域名解析请求的返回与否。
	主动性方法	被动
	多态性	None
	消息发送	
	操作流程	
	其他	
	}	

	{	
	特征标记	close()
	操作解释	在实例结束的时候关闭域名解析请求的连结，同时注销在 IO 复用 Eventloop 中的监听。
	主动性方法	被动
	多态性	None
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_periodic()
	操作解释	该操作主要是定时清除域名解析的缓存。毕竟缓存的大小是有限的。
	主动性方法	被动
	多态性	None
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	resolve(hostname : bytes, callback : function)
	操作解释	该操作主要是检查域名是否合法以及是否已经可以从本地获取。若不可以从本地获取，则向域名服务器发送解析请求。同时该操作还会记录该解析请求返回以后要进行的操作。
	主动性方法	被动
	多态性	None
	消息发送	
	操作流程	
	其他	
	}	
对象实例说明	{	对每种需要该类对象的处理机说明如下 内容
	处理机	指出这种处理机的每一个实际的节点
	内存对象	指出用该类创建所有内存对象

	外存对象	指出为这个类保存的外存对象
	}	

类的总体说明		
	类名	DNSResponse
	解释	这个类保存了域名解析请求返回的结果中，解析出来的域名
	一般类	No
	主动性	No
	持久性	No
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	answers : list
	属性解释	该属性保留了应答报文的地址类型、查询类型和记录类型
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	hostname : bytes
	属性解释	该属性保留域名解析请求返回的 ip 地址
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	question : list
	属性解释	该属性保留了请求报文的地址类型、查询类型和记录类型
	多态性	
	关联、聚合或组合	
	其它	
	}	
对象实例说明	{	

	处理机	
	内存对象	
	外存对象	
	}	

类的总体说明		
	类名	Eventloop
	解释	这个一个侦听事件的类，用于将 socket 所侦听的事件注册进去，不断循环侦听，触发事件 将调用相应的 handler 处理。
	一般类	None
	主动性	Yes
	持久性	No
	辅助模型	None
	其他	None
属性说明		
	{	
	名称与数据类型	model: string
	属性解释	指明 Eventloop 目前是哪种 IO 复用接口。 有’epoll’、’kqueue’和’select’三种模式。
	多态性	No
	组合	None
	其它	None
	}	
	{	
	名称与数据类型	impl: Epoll/KqueueLoop/SelectLoop
	属性解释	IO 复用模式实例，可以为 Epoll、KqueueLoop 或者 SelectLoop。
	多态性	No
	组合	作为 Eventloop 必不可少的成员。
	其它	None
	}	None

操作说明		
	{	
	特征标记	add(socket, mode, handler)
	操作解释	将 socket 与对应的的处理 handler 加入字典中，并在相关 IO 复用接口中为 socket 注册 所指定的侦听事件。
	主动性方法	被动
	多态性	X
	消息发送	No
	操作流程	None
	其他	None
	}	
	{	
	特征标记	add_periodic(callback)
	操作解释	增加周期性回调函数，可添加的函数包括 TCPRelay、UDPRelay 或 DNSResolver 的 handle_periodic 函数处理超时或者清除缓存。
	主动性方法	被动
	多态性	x
	消息发送	No
	操作流程	None
	其他	None
	}	
	{	
	特征标记	modify(socket, mode)
	操作解释	修改指定 socket 所侦听的事件为 mode 事件。
	主动性方法	被动
	多态性	x
	消息发送	No
	操作流程	None
	其他	None
	}	
	{	
	特征标记	poll(timeout)

	操作解释	该操作主要是调用 IO 复用接口来等待事件触发，并返回触发的事件。
	主动性方法	被动
	多态性	x
	消息发送	No
	操作流程	None
	其他	None
	}	
	{	
	特征标记	remove(socket)
	操作解释	将 socket 从字典中移除，并移除注册的侦听事件。
	主动性方法	被动
	多态性	x
	消息发送	No
	操作流程	None
	其他	None
	}	
	{	
	特征标记	remove_periodic(callback)
	操作解释	移除周期性回调函数，移除的函数可能为 TCPRelay、UDPRelay 或者 DNSResolver 的 handle_periodic 函数处理超时或者清除缓存。
	主动性方法	被动
	多态性	x
	消息发送	No
	操作流程	None
	其他	None
	}	
	{	
	特征标记	stop()
	操作解释	暂停 IO 接口复用。
	主动性方法	被动
	多态性	x
	消息发送	No
	操作流程	None

	其他	None
	}	
	{	
	特征标记	run()
	操作解释	等待注册事件发生，然后通过事件对应的文件描述符找到 handler，并将事件交给 handler 处理。同时每隔一定时间调用 handle_periodic 函数处理超时或者清除缓存。
	主动性方法	被动
	多态性	x
	消息发送	No
	操作流程	活动图
	其他	None
	}	
对象实例说明	{	
	处理机	
	内存对象	
	外存对象	
	}	

类的总体说明		
	类名	<中文>客户端：<英文>Local
	解释	这个类的主要功能启动本地用户的客户端代理。
	一般类	No
	主动性	No
	持久性	No
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	config : dict
	属性解释	config 保留了客户端运行的必要配置信息，以一种键值对的形式保存信息。

	多态性	None
	关联、聚合或组合	No。
	其它	
	}	
	{	
	名称与数据类型	dns_resolver : DNSResolver
	属性解释	这个属性是为了运行域名解析的功能
	多态性	No
	组合	Local 必须得依靠 dns_resolver 才能执行域名解析的功能。
	其它	
	}	
	{	
	名称与数据类型	tcp_relay : TCPRelay
	属性解释	这个属性是为了转发用户和境外服务器之间的 TCP 包而设立的。
	多态性	No
	组合	Local 必须得依靠 tcp_relay 才能转发数据包。
	其它	
	}	
	{	
	名称与数据类型	udp_relay : UDPRelay
	属性解释	这个属性是为了用户和境外服务器之间的 UDP 包而设立的。
	多态性	
	组合	
	其它	
	}	
操作说明		
	{	
	特征标记	run()
	操作解释	该操作主要是启动域名解析、TCP 和 UDP 包转发的功能。
	主动性方法	被动
	多态性	None
	消息发送	None
	操作流程	
	其他	

	}	
	}	
对象实例说明	{	对每种需要该类对象的处理机说明如下 内容
	处理机	指出这种处理机的每一个实际的节点
	内存对象	指出用该类创建所有内存对象
	外存对象	指出为这个类保存的外存对象
	}	

类的总体说明		
	类名	<中文>LRU 缓存区：<英文>LRUCache
	解释	缓存，采用最近最久未使用缓存清除算法。
	一般类	None
	主动性	No
	持久性	Yes
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	closed_values: set
	属性解释	需要清除的缓存数据。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	
	{	
	名称与数据类型	keys_to_last_time: dict
	属性解释	每个缓存数据对应的键值的时间。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	

	{	
	名称与数据类型	last_visits: Deque
	属性解释	记录每次访问缓存数据的时间。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	
	{	
	名称与数据类型	store: dict
	属性解释	保存缓存数据的字典。
	多态性	x
	关联、聚合或组合	组合，一个缓存区里面必须要一个保存数据的区域。
	其它	
	}	
	{	
	名称与数据类型	Time_to_keys: dict
	属性解释	每次访问的时间所对应的缓存数据的键值。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	
	{	
	名称与数据类型	close_callback: Function
	属性解释	清除数据的回调函数，可以在清除缓存时，执行一些额外的功能。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	
	{	
	名称与数据类型	time_out: float
	属性解释	设置的时间间隔，当清除缓存数据时，就会清除从现在到 time_out 的时间间隔内的缓存数据。
	多态性	x
	关联、聚合或组合	None

	其它	
	}	
操作说明		
	{	
	特征标记	sweep()
	操作解释	清除最近最久未被使用的数据
	主动性方法	被动
	多态性	x
	消息发送	None
	操作流程	
	其他	
	}	
对象实例说明	{	
	处理机	
	内存对象	
	外存对象	
	}	

类的总体说明		
	类名	<中文>客户端：<英文>Local
	解释	这个类的主要功能启动本地用户的客户端代理。
	一般类	No
	主动性	No
	持久性	No
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	config : dict
	属性解释	config 保留了客户端运行的必要配置信息，以一种键值对的形式保存信息。
	多态性	None
	关联、聚合或组合	No。

	其它	
	}	
	{	
	名称与数据类型	dns_resolver : DNSResolver
	属性解释	这个属性是为了运行域名解析的功能
	多态性	No
	组合	Local 必须得依靠 dns_resolver 才能执行域名解析的功能。
	其它	
	}	
	{	
	名称与数据类型	tcp_relay : TCPRelay
	属性解释	这个属性是为了转发用户和境外服务器之间的 TCP 包而设立的。
	多态性	No
	组合	Local 必须得依靠 tcp_relay 才能转发数据包。
	其它	
	}	
	{	
	名称与数据类型	udp_relay : UDPRelay
	属性解释	这个属性是为了用户和境外服务器之间的 UDP 包而设立的。
	多态性	
	组合	
	其它	
	}	
操作说明		
	{	
	特征标记	run()
	操作解释	该操作主要是启动域名解析、TCP 和 UDP 包转发的功能。
	主动性方法	被动
	多态性	None
	消息发送	None
	操作流程	
	其他	
	}	
	}	

对象实例说明	{	对每种需要该类对象的处理机说明如下 内容
	处理机	指出这种处理机的每一个实际的节点
	内存对象	指出用该类创建所有内存对象
	外存对象	指出为这个类保存的外存对象
	}	

类的总体说明		
	类名	Shell
	解释	ShadowSocks 软件的内核，定义了配置文件的搜索、检查、提取内容的方法，以及打印 帮助文档的方法等。
	一般类	None
	主动性	No
	持久性	No
	辅助模型	
	其他	
操作说明		
	{	
	特征标记	check_config(config: list, is_local: boolean)
	操作解释	检查并安装配置。如果有异常（IP 地址异常或者加密算法不安全等）会发出警告， 出现错误（密码错误或未指定等）会退出程序。
	主动性方法	被动
	多态性	部分代码的执行与否取决于 is_local 的值（为 true 时仅需要密码，为 false 时需要本地密 码或端口密码）
	消息发送	None
	操作流程	
	其他	
	}	
	{	

	特征标记	get_config(is_local: boolean)
	操作解释	获取配置信息
	主动性方法	被动
	多态性	is_local 为 true 时获取客户端配置，为 false 获取服务器配置
	消息发送	None
	操作流程	
	其他	
	}	
	{	
	特征标记	print_help(is_local: boolean)
	操作解释	打印帮助信息
	主动性方法	被动
	多态性	is_local 为 true 时打印客户端帮助信息，否则打印服务器帮助信息
	消息发送	None
	操作流程	
	其他	
	}	

类的总体说明		
	类名	UDPRelay
	解释	这个类负责处理客户端跟服务端的所有通过 UDP 协议交换数据的事件
	一般类	No
	主动性	No
	持久性	No
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	Cache：LRUCache
	属性解释	缓存对象，用于缓存每一个 client 的信息，可以在该缓存找到具体的 client

	多态性	No
	关联、聚合或组合	Cache 使用了 LRUCache’ 类的方法存储数据，组合关系
	其它	
	}	
	{	
	名称与数据类型	closed：bool
	属性解释	标记 UDPRelay 是否关闭，如果标记为关闭，清空所有缓存，断开相关的所有 udp 会话
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	Client_fd_to_server_addr:LRUCache
	属性解释	这是存储 client socket 号，通过该 fd 号能够找到具体的 client
	多态性	No
	关联、聚合或组合	Cache 使用了 LRUCache’ 类的方法存储数据，组合关系
	其它	
	}	
	{	
	名称与数据类型	config：json
	属性解释	这是配置文件的信息
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	dns_resolver: DNSResolver
	属性解释	这是用于将域名转换成 IP 地址的实例，负责处理有关域名解析的工作，在 udp 连接时 有可能需要进行域名解析工作
	多态性	No
	关联、聚合或组合	这是专门设计的处理域名解析的类，属于组合关系
	其它	
	}	

3
4

	{	
	名称与数据类型	dns_cache: LRUCache
	属性解释	这是域名解析的缓存，用于将解析过的域名存储，下次需要域名解析时先查找该缓存
	多态性	No
	关联、聚合或组合	Cache 使用了 LRUCache’ 类的方法存储数据，组合关系
	其它	
	}	
	{	
	名称与数据类型	eventloop: EventLoop
	属性解释	事件循环，通过 IO 复用将事件传入该类进行处理
	多态性	No
	关联、聚合或组合	跟 Eventloop 是组合关系
	其它	
	}	
	{	
	名称与数据类型	listen：string
	属性解释	这个是 udp 监听的 ip 地址和端口
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	method: string
	属性解释	只是使用 SOCK5 的方法，通过方法和密码指定客户端与服务端之间信息交换的方法
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	Password: string
	属性解释	这是加密信息时使用的密码，这是用户自行设定的，在本地端和服务端使用密码加密后 再传输信息

	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	Remote: string
	属性解释	远端服务端的 ip 地址和端口号，用于连接
	多态性	
	关联、聚合或组合	
	其它	
	}	
	{	
	名称与数据类型	stat_callback：bool
	属性解释	回调标记位，如果有 udp 连接有新的信息，标记位变为真，可以通过检查该标记为确定 有没有新的信息
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	Sockets：socket[]
	属性解释	存储所有 client 的 socket 实例
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	
	名称与数据类型	Server_socket: socket
	属性解释	这是服务端的 socket 实例
	多态性	No
	关联、聚合或组合	No
	其它	
	}	
	{	

	名称与数据类型	Timeout：int
	属性解释	时间阈值，如果计时超过阈值还没有新的信息到来，将该 client 标记为断开
	多态性	
	关联、聚合或组合	
	其它	
	}	
操作说明		
	{	
	特征标记	Close_client()
	操作解释	该操作会关闭一个 client 的 udp 连接，并把对应的 cache 信息删除
	主动性方法	被动
	多态性	No
	消息发送	No
	操作流程	
	其他	
	}	
	{	
	特征标记	Get_a_server()
	操作解释	从配置信息中选取一个可用的服务端 ip 地址和端口并返回用于连接
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_client(sock: socket)
	操作解释	接收服务器回复给客户端的数据，并将其转发给客户端
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	

	其他	
	}	
	{	
	特征标记	handle_server()
	操作解释	服务端处理数据，如果是远端服务端则解密转发数据，如果是本地服务端则加密传送到远端服务端
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	add_to_loop(loop: Eventloop)
	操作解释	将 UDP 会话加入到事件循环中
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	close()
	操作解释	关闭 UDP 转接，将所有 udp 连接断开
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_event(sock: socket, fd: int, event: event)
	操作解释	判断新事件是来自哪个 socket，若是 server 的就新建一个 UDP 用户会话，若是 UDP 用户的，则转发
	主动性方法	被动

	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_periodic()
	操作解释	判断 UDPRealy 是否关闭，若是，关闭与之相关的 socket 连接，清除缓存
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
对象实例说明	{	
	处理机	
	内存对象	
	外存对象	
	}	

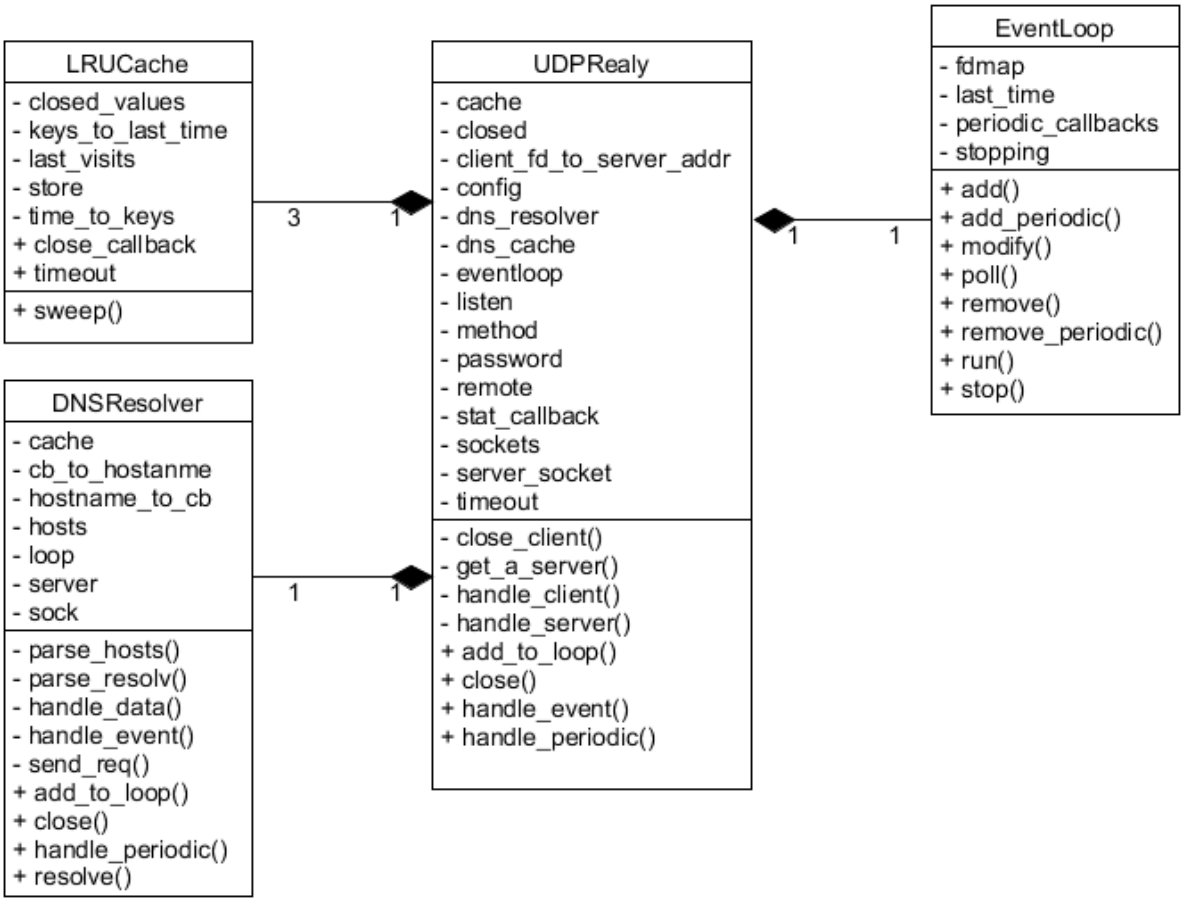
类的总体说明		
	类名	TCPRelay
	解释	对所有的 TCP 连接进行管理，然后根据类型来新建 TCP 连接或者分发任务事件
	一般类	None
	主动性	No
	持久性	No
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	Config: json

	属性解释	配置文件的信息
	多态性	No
	组合	No
	其它	
	}	
	{	
	名称与数据类型	Closed：bool
	属性解释	标记该 tcp 转接是否关闭了
	多态性	No
	聚合	No
	其它	
	}	
	{	
	名称与数据类型	Dns_resolver:: DNSResolver
	属性解释	用于 tcp 连接时将域名解析为 ip 地址
	多态性	No
	组合	使用专门定义的域名解析的 DNSResolver 类，是组合关系
	其它	
	}	
	{	
	名称与数据类型	Fd_to_handlers: int
	属性解释	这是每个 tcphandler 的标识号，根据标识号可以拿到具体的 tcphandler
	多态性	No
	聚合	No
	其它	
	}	
	{	
	名称与数据类型	eventloop: EventLoop
	属性解释	事件循环，通过 IO 复用将事件传入该类进行处理
	多态性	No
	组合	跟 Eventloop 是组合关系
	其它	
	}	
	{	

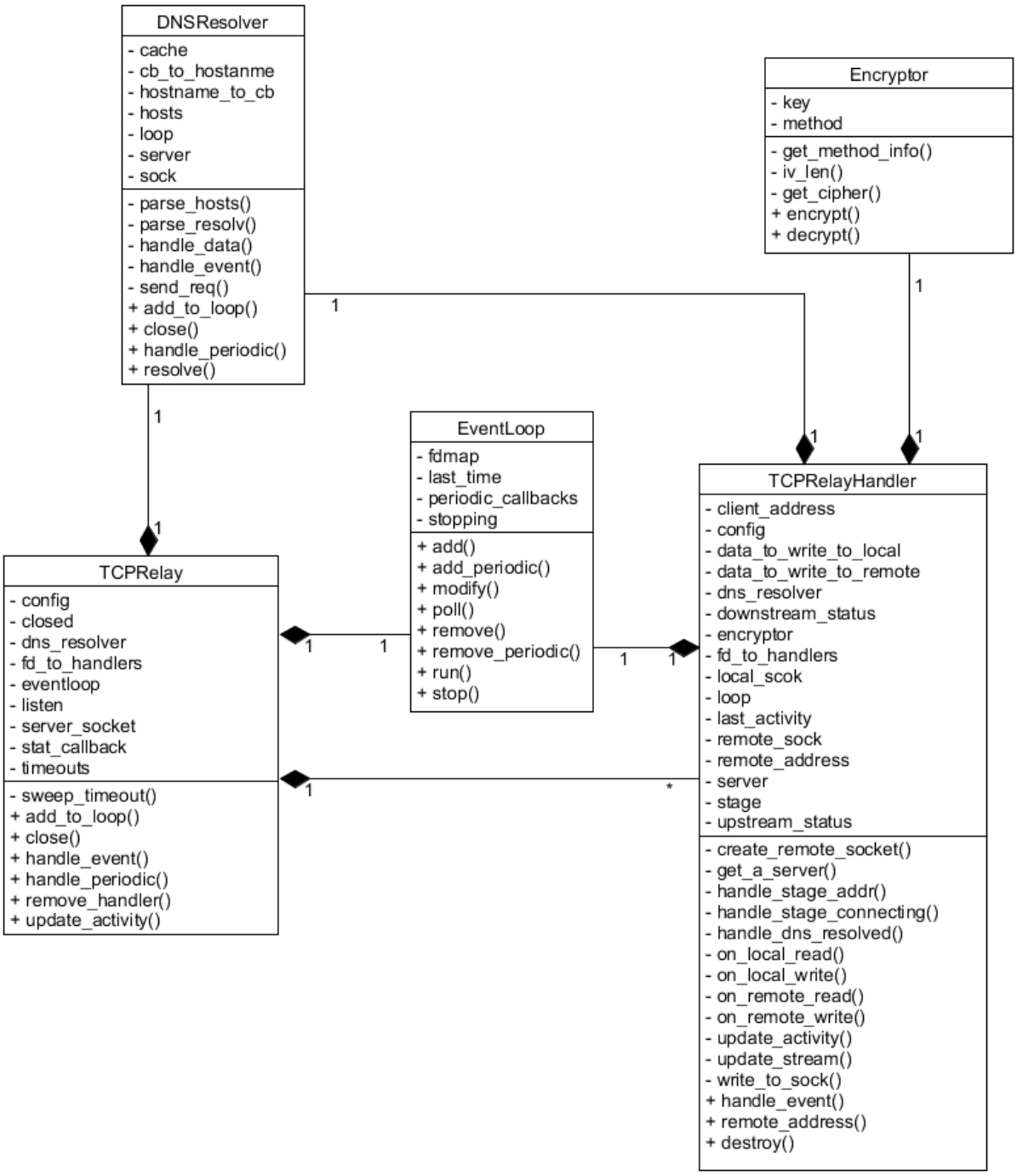
	名称与数据类型	Listen:string
	属性解释	这是 tcprelay 监听的 ip 地址和端口
	多态性	No
	聚合	No
	其它	
	}	
	{	
	名称与数据类型	Server_socket: socket
	属性解释	这是服务端的 socket 连接
	多态性	No
	组合	No
	其它	
	}	
	{	
	名称与数据类型	stat_callback: bool
	属性解释	回调标记位，标记有没有新的信息
	多态性	No
	聚合	No
	其它	
	}	
	名称与数据类型	Timeouts: int
	属性解释	时间阈值，如果 handler 超过时间阈值还没有新的消息，则删除该 handler
	多态性	No
	聚合	No
	其它	
	}	
操作说明		
	{	
	特征标记	sweep_timeout()
	操作解释	当 handler 关闭时，需要删除对应的计时
	主动性方法	被动
	多态性	No
	消息发送	No
	操作流程	

	其他	
	}	
	{	
	特征标记	add_to_loop(loop: Eventloop)
	操作解释	将 TCP 会话加入到事件循环中
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	Close()
	操作解释	关闭 tcp 转接 进程
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_event(sock: socket, fd: int, event: event)
	操作解释	判断事件是新建 handler 还是已有连接的事件并分发事件
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_periodic()
	操作解释	检查一个 TCPRealy 是否关闭，如果关闭了，就断开与其相关的 TCP 连接，并将其从事件循环中删除，调用 _sweep_timeout 定期清理一段时间内不活跃的 TCPRelayHandler

	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	remove_handler(handler: TCPRelayHandler)
	操作解释	删除一个 TCPRealy Handler
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	update_activity(handler: TCPRelayHandler, data_len: int)
	操作解释	更新一个 handler 的激活状态，将该 handler 的计时刷新，更新超时队列
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
对象实例说明	{	
	处理机	
	内存对象	
	外存对象	
	}	


LRUCache
- closed_values
- keys_to_last_time
- last_visits
- store
- time_to_keys
+ close_callback
+ timeout
+ sweep()
DNSResolver
- cache
- cb_to_hostanme
- hostname_to_cb
- hosts
- loop
- server
- sock
- parse_hosts()
- parse_resolv()
- handle_data()
- handle_event()
- send_req()
+ add_to_loop()
+ close()
+ handle_periodic()
+ resolve()
UDPRealy
- cache
- closed
- client_fd_to_server_addr
- config
- dns_resolver
- dns_cache
- eventloop
- listen
- method
- password
- remote
- stat_callback
- sockets
- server_socket
- timeout
- close_client()
- get_a_server()
- handle_client()
- handle_server()
+ add_to_loop()
+ close()
+ handle_event()
+ handle_periodic()
EventLoop
- fdmap
- last_time
- periodic_callbacks
- stopping
+ add()
+ add_periodic()
+ modify()
+ poll()
+ remove()
+ remove_periodic()
+ run()
+ stop()
3
1
1
1
1
1

udp 转发模块类图



类的总体说明		
	类名	<中文>加密器;<英文>Encryptor
	解释	对数据进行加密和解密
	一般类	None

	主动性	No
	持久性	No
	辅助模型	None
	其他	
属性说明		
	{	
	名称与数据类型	key:string
	属性解释	密钥，用于加密或解密数据
	多态性	
	关联、聚合或组合	
	其它	
	}	
	{	
	名称与数据类型	method:string
	属性解释	加密算法的名称，指明使用哪种加密算法进行数据加密
	多态性	
	关联、聚合或组合	
	其它	
	}	
操作说明		
	{	
	特征标记	encrypt(buf:string) : bytes
	操作解释	对数据 buf 进行加密
	主动性方法	被动
	多态性	
	消息发送	None
	操作流程	None
	其他	
	}	
	{	

	特征标记	decrypt(buf:string) : bytes
	操作解释	解密密文 buf
	主动性方法	被动
	多态性	
	消息发送	None
	操作流程	None
	其他	
	}	

类的总体说明		
	类名	TCPRelayHandler
	解释	这个是用来处理单个 TCP 连接的事件的 handler，每一个 handler 负责一个 tcp 连接的事件
	一般类	None
	主动性	No
	持久性	No
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	Client_address: string
	属性解释	TCP 连接中客户端的 ip 地址
	多态性	No
	组合	No
	其它	
	}	
	{	
	名称与数据类型	Config: json
	属性解释	配置文件的信息
	多态性	No
	聚合	No
	其它	

	}	
	{	
	名称与数据类型	data_to_write_to_local：bytes
	属性解释	需要发送给本地端的数据
	多态性	No
	组合	No
	其它	
	}	
	{	
	名称与数据类型	data_to_write_to_remote：bytes
	属性解释	需要发送给服务端的数据
	多态性	No
	聚合	No
	其它	
	}	
	{	
	名称与数据类型	Dns_resolver::DNSResolver
	属性解释	用于 tcp 连接时将域名解析为 ip 地址
	多态性	No
	组合	使用专门定义的域名解析的 DNSResolver 类，是组合关系
	其它	
	}	
	{	
	名称与数据类型	downstream_status：int
	属性解释	标记输入数据流是属于什么数据流
	多态性	No
	聚合	No
	其它	
	}	
	{	
	名称与数据类型	encryptor：Encryptor
	属性解释	加密类实例，用于传输数据时的加密
	多态性	No
	组合	与 Encryptor 类是组合关系

	其它	
	}	
	{	
	名称与数据类型	fd_to_handlers：int
	属性解释	属于该 handler 的 fd 标识号，专属于该 handler 的标识号
	多态性	
	聚合	
	其它	
	}	
	{	
	名称与数据类型	local_scok：socket
	属性解释	与本地 tco 连接的 socket 实例
	多态性	No
	组合	No
	其它	
	}	
	{	
	名称与数据类型	loop：Eventloop
	属性解释	事件循环，通过 IO 复用将事件传入该类进行处理
	多态性	No
	聚合	跟 Eventloop 是组合关系
	其它	
	}	
	{	
	名称与数据类型	last_activity：int
	属性解释	记录上次信息到来的时刻
	多态性	No
	组合	No
	其它	
	}	
	{	
	名称与数据类型	remote_sock：socket
	属性解释	这是跟服务端的 socket 实例
	多态性	No

	聚合	No
	其它	
	}	
	{	
	名称与数据类型	remote_address：string
	属性解释	这是跟服务端 tcp 连接时服务端的 ip 地址
	多态性	No
	组合	No
	其它	
	}	
	{	
	名称与数据类型	Server：socket
	属性解释	这是与服务端的连接 socket
	多态性	No
	聚合	No
	其它	
	}	
	{	
	名称与数据类型	Stage：int
	属性解释	标识当前 handler 的状态
	多态性	No
	组合	No
	其它	
	}	
	{	
	名称与数据类型	upstream_status：int
	属性解释	标识输出数据流属于什么数据流
	多态性	No
	聚合	No
	其它	
	}	
操作说明		
	{	
	特征标记	create_remote_socket（ip: string, port: int）

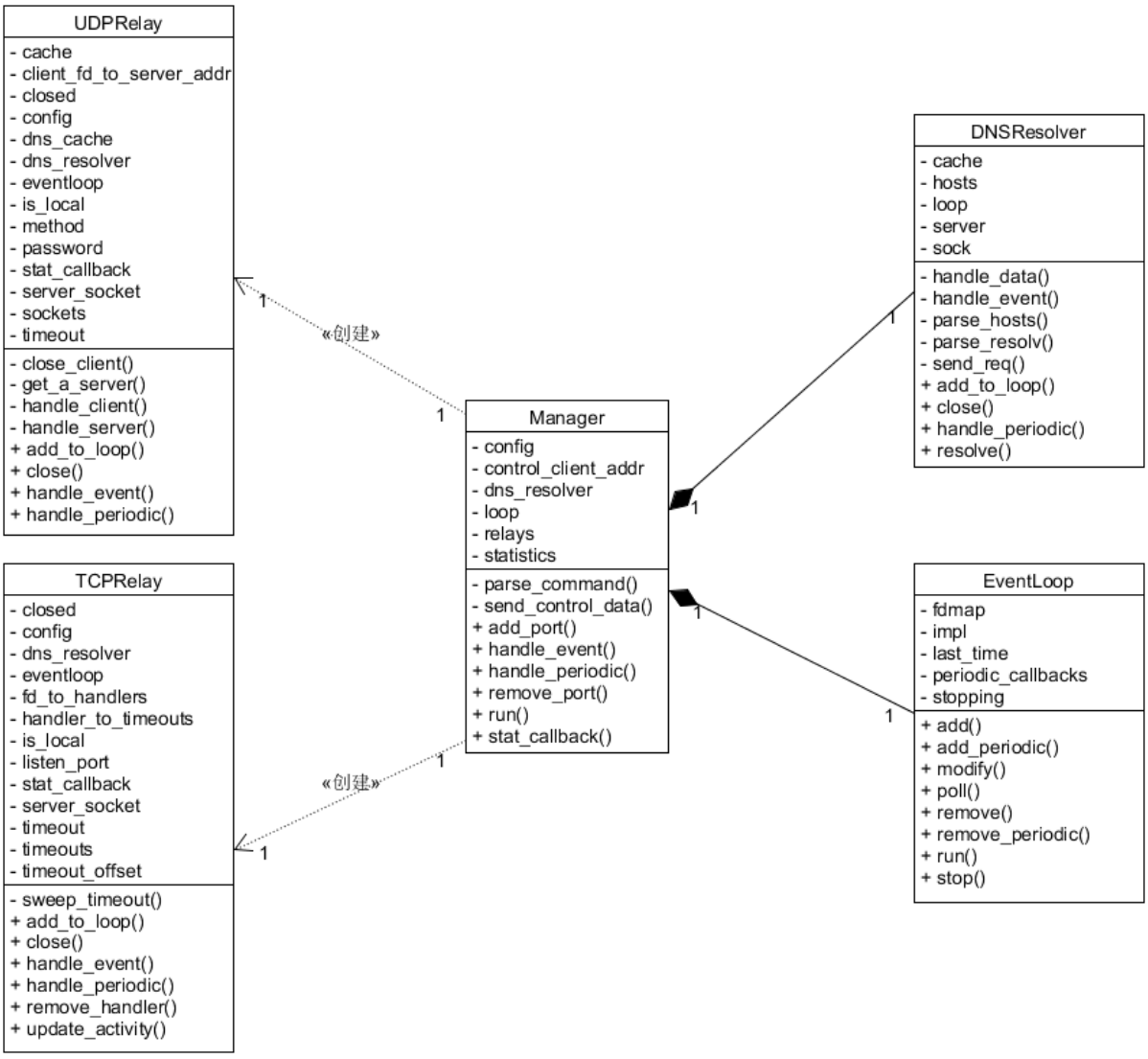
	操作解释	建立与远端服务器的 TCP 连接，并返回连接的 socket
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	get_a_server（）
	操作解释	从可选服务器中选出一个端口建立 TCP 连接
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_stage_addr(data: bytes)
	操作解释	建立 SOCK5 的连接，握手阶段
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_stage_connecting(data: bytes)
	操作解释	建立 SOCK5 的连接，连接阶段
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	

	{	
	特征标记	handle_dns_resolved(result: string, error:string)
	操作解释	在建立 SOCK5 时，通过域名解析将服务器转换成 IP 地址，然后根据 IP 地址和端口建立连接
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	on_local_read()
	操作解释	本地端向服务端读取信息
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	on_local_write()
	操作解释	本地端向服务端发送信息
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	on_remote_read()
	操作解释	服务端从本地端读取信息
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	

	其他	
	}	
	{	
	特征标记	on_remote_write()
	操作解释	服务端向本地端发送信息
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	update_activity(data_len: int)
	操作解释	更新 TCPhandler 的状态
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	update_stream(stream: int, status: int)
	操作解释	更新当前 socket 监听的事件，会在 status 发生变化时更新
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	write_to_sock(data:bytes, sock: socket)
	操作解释	向 socket 传输数据
	主动性方法	被动
	多态性	No

	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_event(sock: socket, event:event)
	操作解释	处理 TCPRelay 分发过来的事件
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	remote_address()
	操作解释	获取远端服务端地址
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
	{	
	特征标记	destroy()
	操作解释	删除 TCPRelayHandler
	主动性方法	被动
	多态性	No
	消息发送	
	操作流程	
	其他	
	}	
对象实例说明	{	
	处理机	
	内存对象	

	外存对象	
	}	



管理服务器模块类图

类的总体说明		
	类名	<中文>管理员：<英文>Manager
	解释	管理多个服务端的连接，断开或 ping，回应客户端是否成功连接或断开或 pong，负责统计服务端端口接收数据长度，周期性地向客服端发送从客户端接收数据的长度。
	一般类	None

	主动性	No
	持久性	No
	辅助模型	
	其他	
属性说明		
	{	
	名称与数据类型	config: dict
	属性解释	配置信息，里面包含了服务端口地址，端口号和密码和管理员地址等等的信息。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	
	{	
	名称与数据类型	control_client_addr: string
	属性解释	客户端的地址，通过这个地址向客户端发送数据。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	
	{	
	名称与数据类型	dns_resolver: DNSResolver
	属性解释	域名解析器，解析数据报中的域名地址。
	多态性	x
	关联、聚合或组合	组合，tcp 和 udp 数据包的管理需要域名解析才能监听数据报中的地址，从而分辨出数据的来源，获得来自客户端的数据报。
	其它	
	}	
	{	
	名称与数据类型	loop: EventLoop
	属性解释	利用 IO 复用机制管理事件，获取触发服务端的事件，从中获取数据流。
	多态性	x
	关联、聚合或组合	组合，管理服务端需要受到一定的触发事件来获得监听到的数据报，以及周期性地发送数据报。

	其它	
	}	
	{	
	名称与数据类型	relays: dict
	属性解释	服务端口：tcp 中继和 udp 中继；管理服务端对应端口号所监听到的 tcp 和 udp 数据包。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	
	{	
	名称与数据类型	statistics: dict
	属性解释	服务端口号接收到对应客户端发来数据的长度。
	多态性	x
	关联、聚合或组合	None
	其它	
	}	
操作说明		
	{	
	特征标记	parse_command()
	操作解释	从接收数据中获取控制指令命令和配置信息。
	主动性方法	被动
	多态性	x
	消息发送	None
	操作流程	
	其他	
	}	
	{	
	特征标记	send_control_data()
	操作解释	向客户端发送数据。
	主动性方法	被动
	多态性	x
	消息发送	socket.sendto()
	操作流程	
	其他	

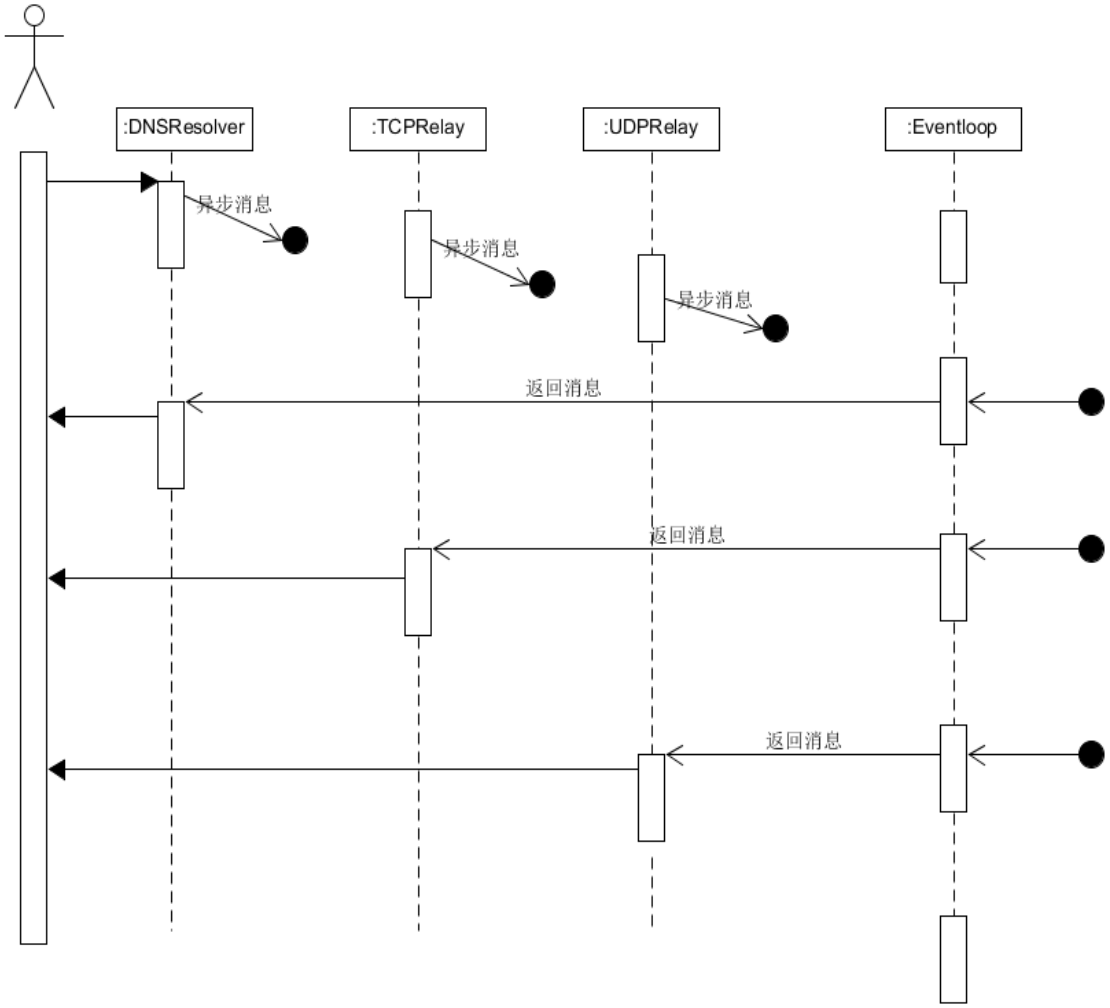
	}	
	{	
	特征标记	add_port()
	操作解释	添加服务端口，并保持只有一个服务端，添加该端口的 tcp 和 udp 中继器，并添加到事件循环中。
	主动性方法	被动
	多态性	x
	消息发送	TCPRelay.add_to_loop(), UDPRelay.add_to_loop()
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_event()
	操作解释	处理事件，根据端口号读取监听到的数据，从中读取控制指令，更新配置信息，处理相应的控制指令。
	主动性方法	被动
	多态性	x
	消息发送	None
	操作流程	
	其他	
	}	
	{	
	特征标记	handle_periodic()
	操作解释	周期性地向客户端发送接收到数据地总长度，并且根据发送上限，分段发送。
	主动性方法	被动
	多态性	x
	消息发送	None
	操作流程	
	其他	
	}	
	{	
	特征标记	remove_port()
	操作解释	删除服务端口，并结束对该端口的监听。
	主动性方法	被动

	多态性	x
	消息发送	TCPRelay.close(), UDPRelay.close()
	操作流程	
	其他	
	}	
	{	
	特征标记	run()
	操作解释	运行对服务端管理的 IO 复用模式。
	主动性方法	被动
	多态性	x
	消息发送	EventLoop.run()
	操作流程	
	其他	
	}	
	{	
	特征标记	stat_callback()
	操作解释	统计相应端口监听到数据的长度（或者数据量）。
	主动性方法	被动
	多态性	x
	消息发送	None
	操作流程	
	其他	
	}	
对象实例说明	{	
	处理机	
	内存对象	
	外存对象	
	}	

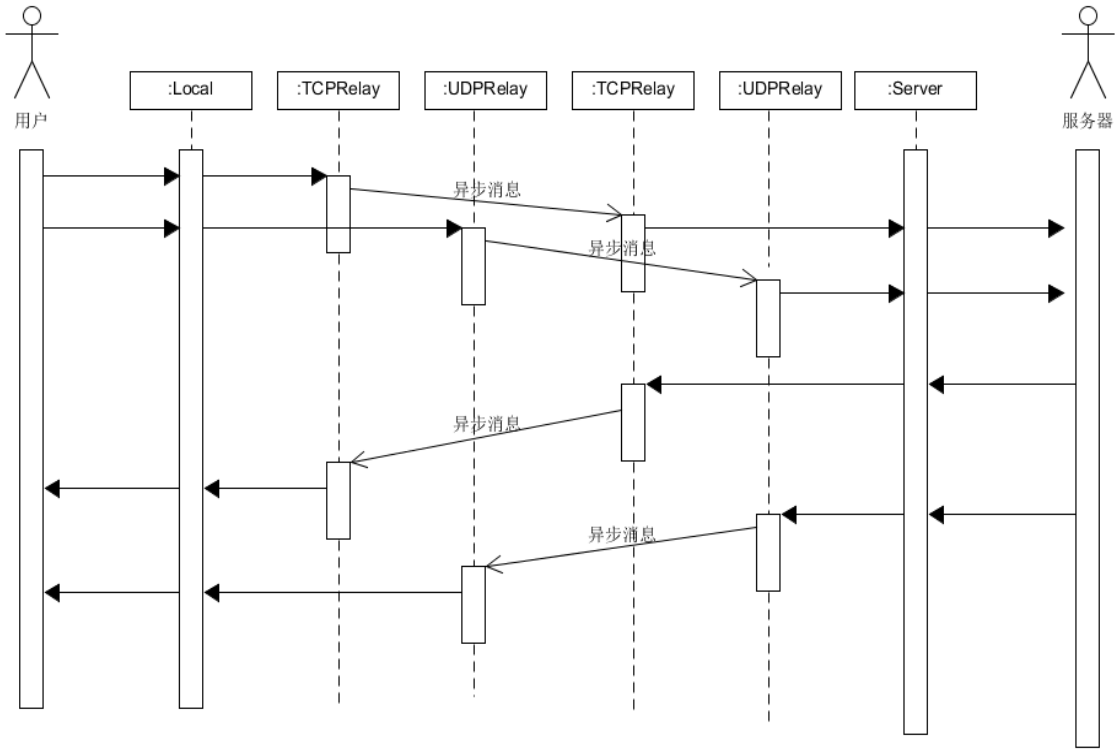
3. 辅助模型

本项目主要用到的辅助模型包括

顺序图：

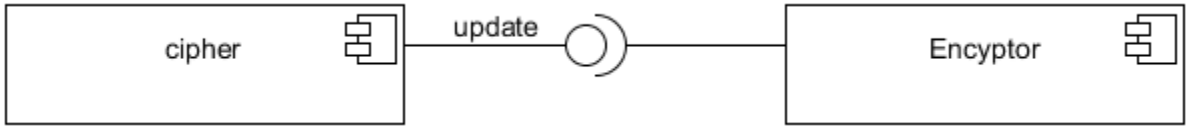

:DNSResolver
:TCPRelay
:UDPRelay
:Eventloop
异步消息
异步消息
异步消息
返回消息
返回消息
返回消息

IO 复用的顺序图

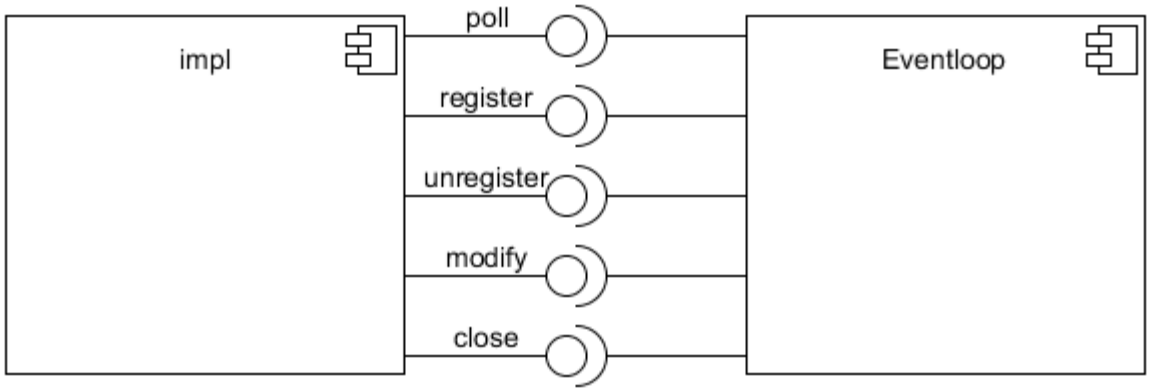


用户和服务器之间的包转发顺序图

构建图:

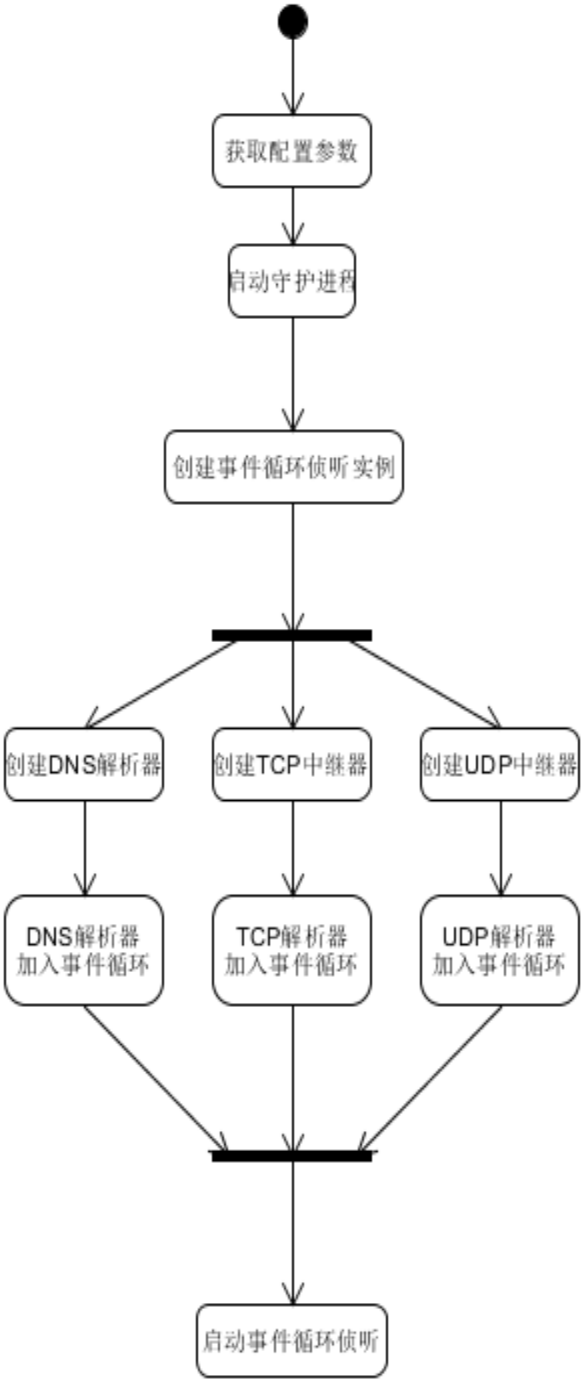


加密器的构建图

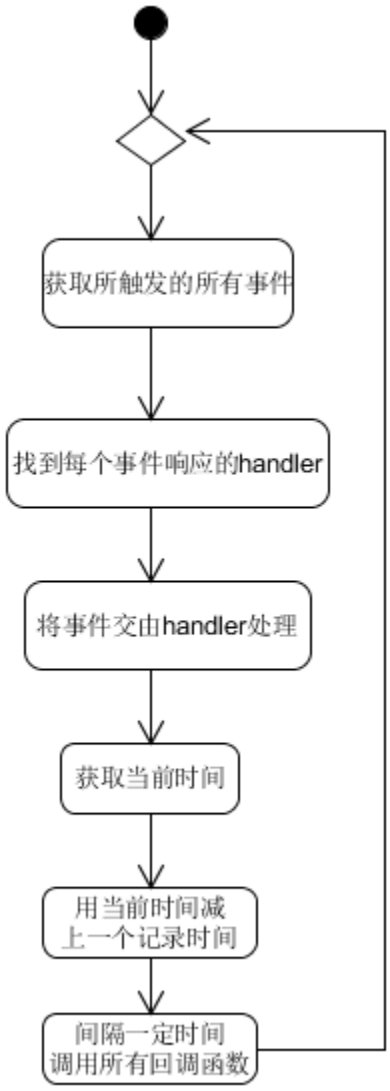


IO 复用的构建图

活动图



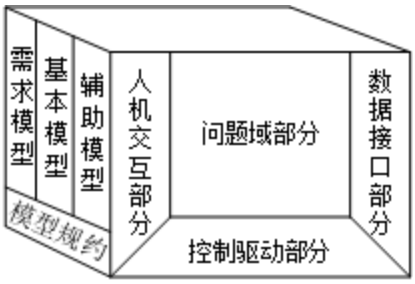
Local 的运行过程活动图



Eventloop.run()活动图

OOD 模型

OOD 阶段是建立在 OOA 阶段的基础上，对 OOA 模型作必要的修改和调整或补充某些细节。其模型框架如下图所示：



本小节主要是按照问题域部分、数据接口部分、控制驱动部分和人机交互部分去阐述。

1. 问题域部分

本应用所采用的编程语言是 Python。Python 语言是一款跨平台的编程语言，可以在 windows 以及 Linux 操作系统以命令行的方式运行程序。Python 也是一款能够支持多继承的编程语言，不论是单继承还是多继承都能很好地应对。而且本应用在设计当中并没有涉及到多继承的问题，自然就没有这方面的问题。

另外针对之前的 Encryptor 类的类规约做了补充，主要是说明在 Encryptor 类中只提供 4 种加密算法，Encryptor 类中 method 属性的注意事项，对 encrypt()操作返回数据的格式以及 decrypt()操作参数和返回数据做了说明。补充这些说明的主要原因是方便用户选择和使用 Encryptor 类提供的加密算法，对数据进行加密和解密。

类的总体说明		
	类名	<中文>加密器;<英文>Encryptor
	解释	对数据进行加密和解密
	一般类	None
	主动性	No
	持久性	No
	辅助模型	None
	其他	包含 4 种加密算法:rc4-md5，openssl，sodium 和 table
属性说明		
	{	
	名称与数据类型	key:string
	属性解释	密钥，用于加密或解密数据
	多态性	

	关联、聚合或组合	
	其它	
	}	
	{	
	名称与数据类型	method:string
	属性解释	加密算法的名称，指明使用哪种加密算法进行数据加密
	多态性	
	关联、聚合或组合	
	其它	method 的值必须是 rc4-md5，openssl，sodium 或 table，不分大小写
	}	
操作说明		

2. 数据接口部分

本应用需要在计算机上存储服务器和客户端的配置信息，因此需要设计一个持久化存储的方案。纵观本应用的配置信息，完全可以用一个文本的描述来存储这些信息。因此本应用直接应用文件系统来创建一个文本文件来存储配置信息。另外采用 json 序列化的方法可以很好地从文本文件中提取出信息。

3. 控制驱动部分

1. 计算机硬件：本应用是一款对内存、外存和 CPU 处理能力要求不高的应用。目前主流的 PC 都可以很好地运行本应用。

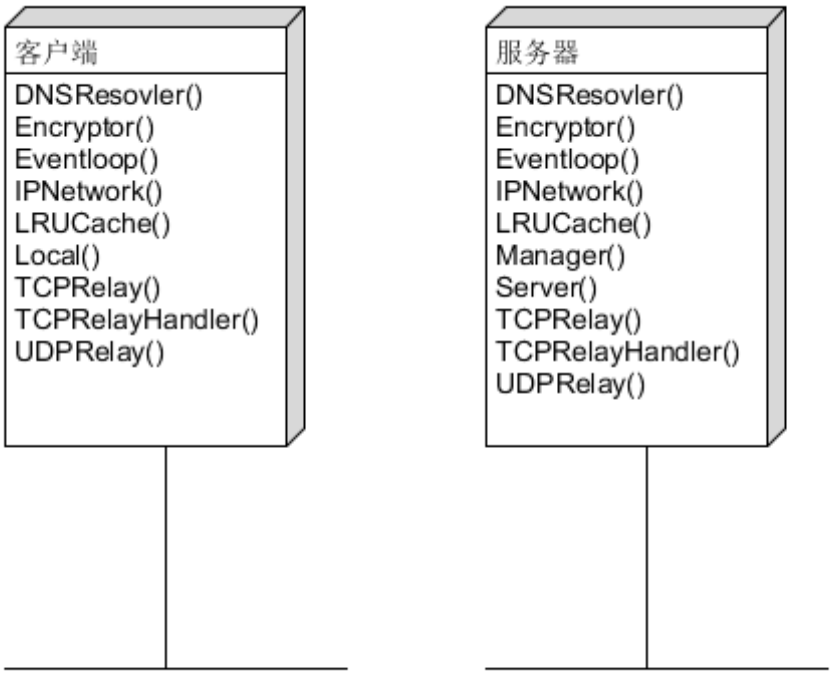
2. 操作系统：目前主流的两大 PC 操作系统：windows 和 Linux 都具有很好的多进程和多线程的支持，并且也支持进程之间的通信（IPC）以及远程过程调用（RPC）。本应用需要进行网络通信，因此无论是 IPC 还是 RPC 都是必需的一部分。不过目前这两个系统都有很好的支持了。

3. 网络方案：本应用所需的网络环境要能够支持经典的 TCP、UDP/IP 协议，也就是要支持 OSI 七层模型中的物理层、数据链路层、网络层、传输层。同时本应用并没有对网络带宽的要求，而通信的速度则取决于本地带宽和服务器带宽的较小者。

4. 软件体系结构：本应用的设计是一个很经典的客户-服务器体系结构。因为本应用的背景是是同墙外的服务器进行通信，很自然地采用了墙外墙内各自放置服务器和客户端的设置。这一种二层客户-服务器体系结构，客户端和服务器是两个界限分明的层次。

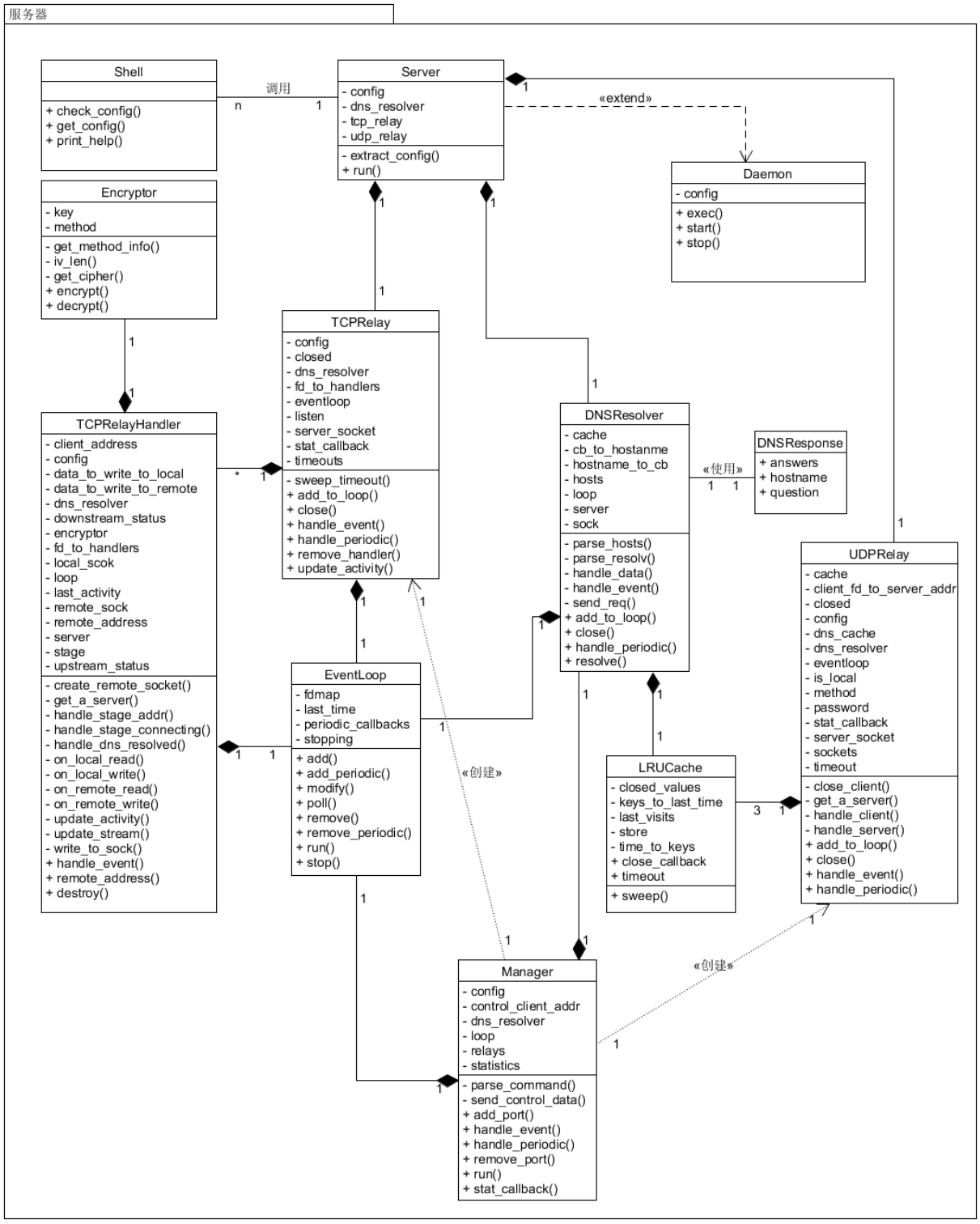
5. 根据软件体系结构锁定地系统分布方案分成两部分，一部分是对象分布；另外一部分是类分布。

   首先是对象分布图。

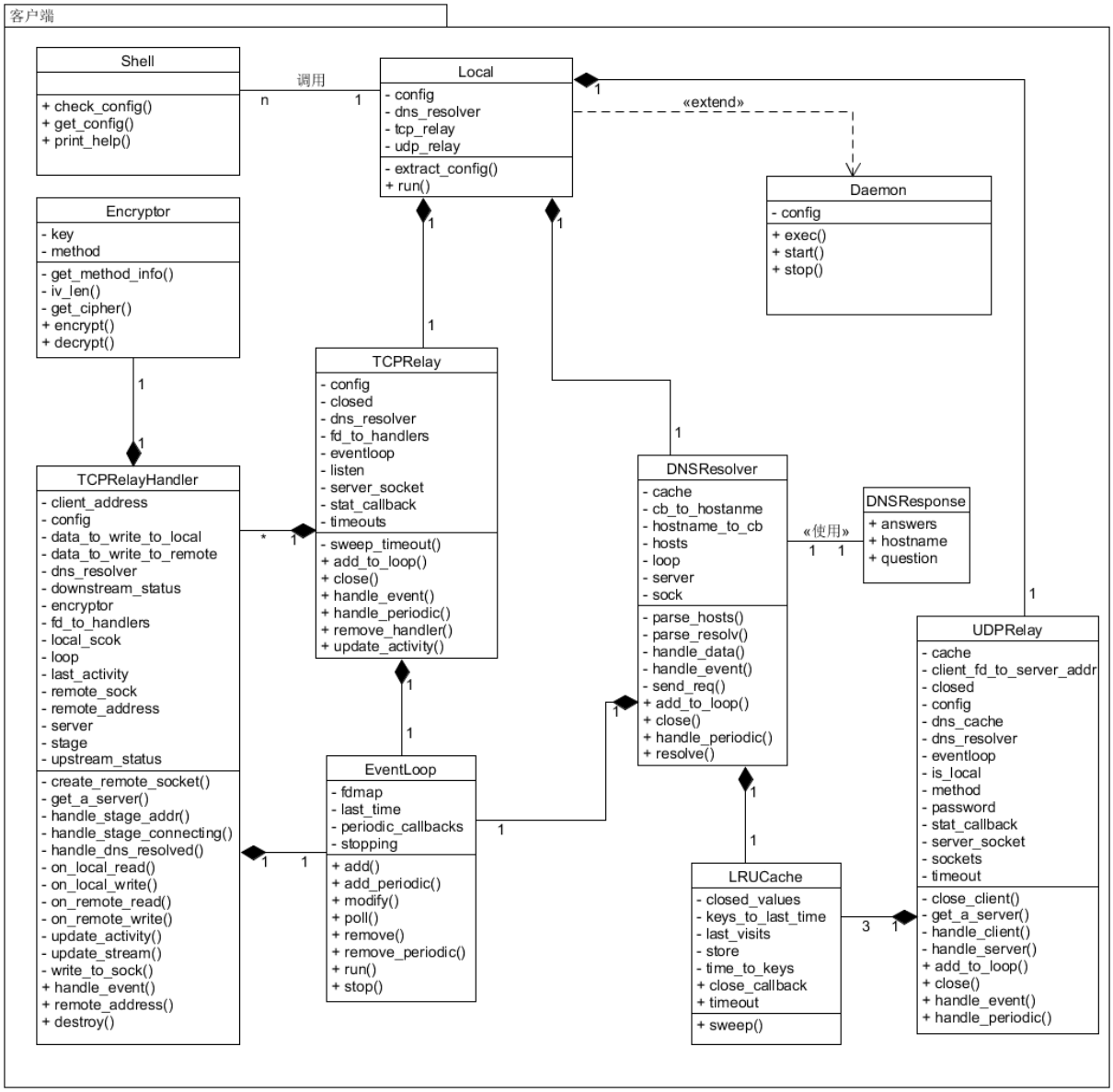


   从中我们可以看到对于服务器端和客户端而言，大部分的对象都是需要，不同的地方仅仅在于各自扮演的角色所需的启动类。如两者相比，客户端只有 Local()，而服务器只有 Manager()和 Server()。两者并没有交集。

   其次是类分布图。

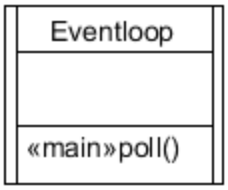

服务器
Shell
+ check_config()
+ get_config()
+ print_help()
调用
n
1
Server
- config
- dns_resolver
- tcp_relay
- udp_relay
- extract_config()
+ run()
«extend»
Daemon
- config
+ exec()
+ start()
+ stop()
Encryptor
- key
- method
- get_method_info()
- iv_len()
- get_cipher()
+ encrypt()
+ decrypt()
TCPRelay
- config
- closed
- dns_resolver
- fd_to_handlers
- eventloop
- listen
- server_socket
- stat_callback
- timeouts
- sweep_timeout()
+ add_to_loop()
+ close()
+ handle_event()
+ handle_periodic()
+ remove_handler()
+ update_activity()
TCPRelayHandler
- client_address
- config
- data_to_write_to_local
- data_to_write_to_remote
- dns_resolver
- downstream_status
- encryptor
- fd_to_handlers
- local_scok
- loop
- last_activity
- remote_sock
- remote_address
- server
- stage
- upstream_status
- create_remote_socket()
- get_a_server()
- handle_stage_addr()
- handle_stage_connecting()
- handle_dns_resolved()
- on_local_read()
- on_local_write()
- on_remote_read()
- on_remote_write()
- update_activity()
- update_stream()
- write_to_sock()
+ handle_event()
+ remote_address()
+ destroy()
DNSResolver
- cache
- cb_to_hostanme
- hostname_to_cb
- hosts
- loop
- server
- sock
- parse_hosts()
- parse_resolv()
- handle_data()
- handle_event()
- send_req()
+ add_to_loop()
+ close()
+ handle_periodic()
+ resolve()
«使用»
DNSResponse
+ answers
+ hostname
+ question
UDPRelay
- cache
- client_fd_to_server_addr
- closed
- config
- dns_cache
- dns_resolver
- eventloop
- is_local
- method
- password
- stat_callback
- server_socket
- sockets
- timeout
- close_client()
- get_a_server()
- handle_client()
- handle_server()
+ add_to_loop()
+ close()
+ handle_event()
+ handle_periodic()
EventLoop
- fdmap
- last_time
- periodic_callbacks
- stopping
+ add()
+ add_periodic()
+ modify()
+ poll()
+ remove()
+ remove_periodic()
+ run()
+ stop()
«创建»
LRUCache
- closed_values
- keys_to_last_time
- last_visits
- store
- time_to_keys
+ close_callback
+ timeout
+ sweep()
3
Manager
- config
- control_client_addr
- dns_resolver
- loop
- relays
- statistics
- parse_command()
- send_control_data()
+ add_port()
+ handle_event()
+ handle_periodic()
+ remove_port()
+ run()
+ stat_callback()
«创建»

服务器分布类图



客户端分布类图

6. 从用况当中可以看出我们需要设计一个控制流来进行不同计算机之间的网络通信。本应用利用 IO 复用的机制设计了一个主动类：Eventloop 来监听网络通信事件。因此该控制流可以被表示成下列类图：



这个是本应用实现网络通信的核心所在，也是实现进程同步的核心所在。

4. 人机交互部分

本应用的人机交互部分并没有涉及到。因为本应用还是一款基于命令行界面的 python 应用，并没有使用类似 windows 的 GUI 界面。因此在这一部分，交互的内容无从谈起。

总结

总结