TrustZone机密计算资源池化项目二期详细设计说明书

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 修订版本 | 修改描述 | 作者 |
| 2022-08-31 | V1.0 | 初始化 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 整体设计 3](#_Toc3548)

[1.1 总体思路 3](#_Toc26040)

[1.2 整体软件运行时序说明 3](#_Toc15222)

[1.3 软件第三方依赖说明 5](#_Toc7800)

[1.4 软件（方案）约束说明 5](#_Toc5168)

[1.4.1 硬件环境要求 5](#_Toc30206)

[1.4.2 操作系统要求 6](#_Toc5464)

[2 libteecc模块设计与实现 6](#_Toc27244)

[2.1 libteecc模块内运行时序 8](#_Toc30330)

[2.2 libteecc模块内接口与数据结构 9](#_Toc605)

[3 GP service设计与实现 19](#_Toc6335)

[3.1 基于异步多线程gRPC Service的gpproxy 20](#_Toc2283)

[3.2 gpproxy模块内运行时序 24](#_Toc3664)

[3.3 gpproxy模块内接口与数据结构 26](#_Toc21381)

[3.4 基于DBus的libdbuscgpw 36](#_Toc4855)

[3.5 基于DBus的gpworker 45](#_Toc27165)

[3.6 gpworker模块内运行时序 46](#_Toc7204)

[3.7 gpworker模块内接口与数据结构 47](#_Toc14296)

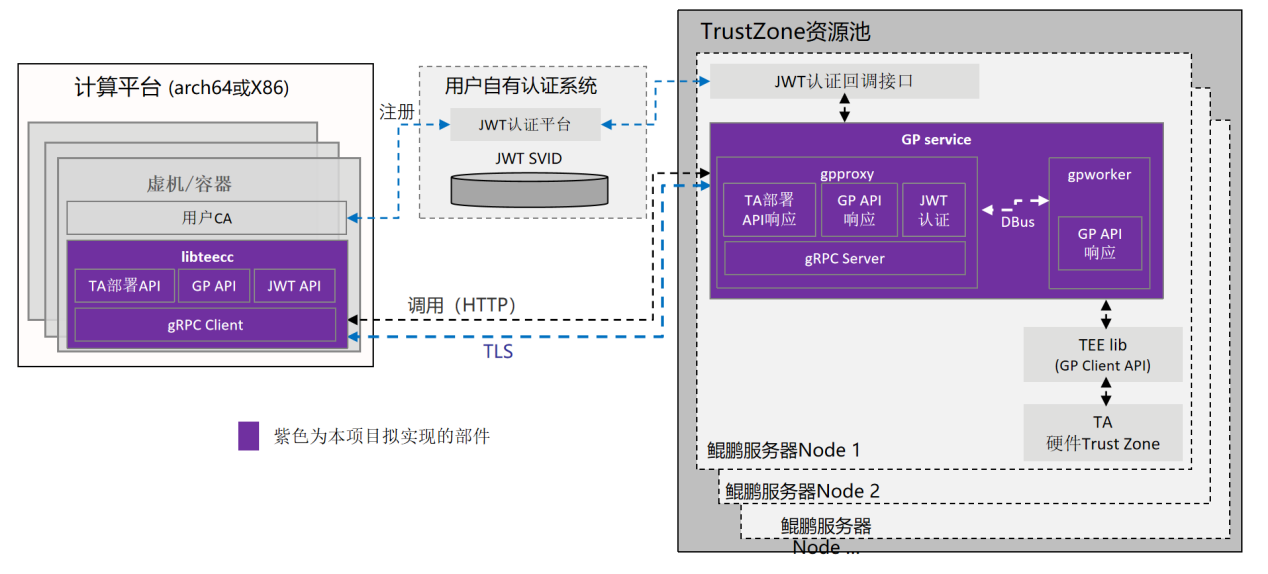
[4 GP Client API内存共享参数传递的透明实现 49](#_Toc1749)

[5 基于TLS的安全传输与基于JWT的鉴权 51](#_Toc19774)

1 整体设计

1.1 总体思路

本项目拟借助gRPC、充分利用gRPC提供的基础架构，构建云化可信执行环境Trustzone 资源池化，其整体架构如图 1所示。基于gRPC Client构建libteecc，提供标准GP接口，达到与业务无关，应用可自行调用相应接口开发需要的业务。GP service主要包括gpproxy和gpworkers。gpproxy基于gRPC Server实现，接收识别由libteecc转发而来的GP调用，通过DBus将它转发到本地相应的gpworker，gpworker再将调用转发给其所在Kunpeng服务器上物理TEE调用库libteec。调用结果由gpworker、gpproxy、libteecc返回给客户CA应用。从而实现异构服务器上的容器/虚机内部署客户CA应用，将异构平台上的CA的调用映射到真实的物理TEE中。本项目同时也利用JWT认证机制，解决TZ部署时，CA对于TA调用难以控制的难题。在JWT认证模块中，包含spire-server、spire-agent两个JWT分发与验证程序，同时加入了两个自研进程dbuss\_fetchjwt与dbuss\_validate用来与在本地运行的JWT认证平台spire-agent进行通信，spire-agent再向spire-server发送JWT获取、验证请求。TLS传输通过gRPC提供的grpc::CreateChannel()进行TLS通道的创建。



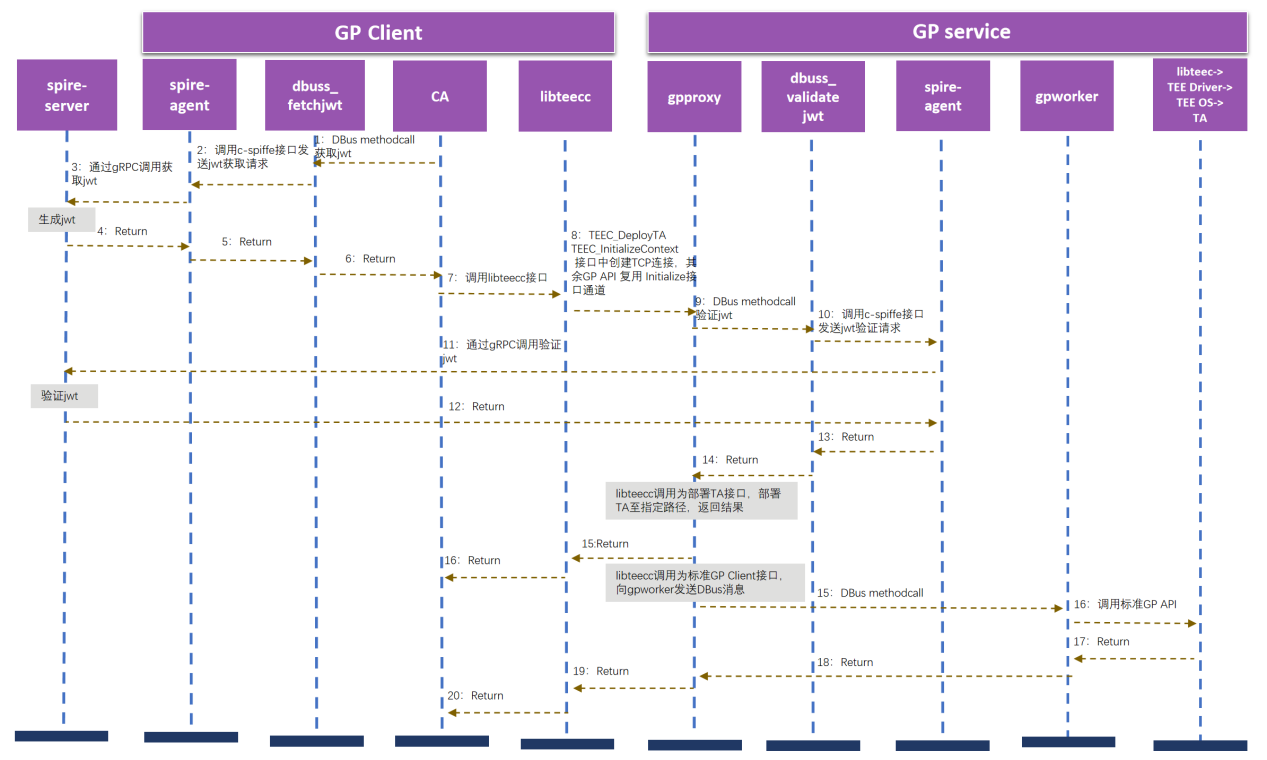
**图 1 TrustZone 机密计算资源池化项目总体架构**

1.2 整体软件运行时序说明

如图2中整体软件运行时序图所示，用户调用libteecc对远程TA进行调用时，默认为不需要JWT认证，若需要JWT认证则客户端需运行自研进程dbuss\_fetchjwt用来与在本地运行的JWT认证平台spire-agent进行通信，自研库dbusc\_jwt提供dbusmethodcall\_fetch\_jwt函数对进程dbuss\_fetchjwt发送DBus请求获取jwt，dbuss\_fetchjwt调用库spiffejwt中的spiffe\_fetch\_jwtsvid函数来向spire-agent发送请求获取jwt，获取到后将返回的jwt通过DBus传回给dbusmethodcall\_fetch\_jwt函数获取到token，再通过TEEC\_SetJwt将token的值传递到libteecc库中以供传递给gpproxy进行jwt认证。调用GP API与TA部署API时，通过gRPC框架传输经过protobuf序列化后的数据至gpproxy，其中也包含token的数据。

在libteecc与gpproxy通讯时，可设置TLS模式，若为TLS连接则会对证书进行检查，若检查无误则建立TLS连接，若需要进行JWT认证，则服务端需运行自研进程dbuss\_validatejwt用来与在本地运行的JWT认证平台spire-agent进行通信，自研库dbusc\_jwt提供dbusmethodcall\_validate\_jwt函数以供gpproxy对进程dbuss\_validatejwt发送DBus请求认证jwt，dbuss\_validatejwt调用库spiffejwt中的spiffe\_validate\_jwtsvid函数来向spire-agent发送请求验证JWT，获取到后将验证结果通过DBus传回给dbusmethodcall\_validate\_jwt函数，gpproxy通过该返回结果判断JWT是否认证通过，若未验证通过则返回JWT认证失败。

JWT认证通过后，TA部署API会将libteecc传输的TA文件直接存入指定路径，GP API中的TEEC\_InitializeContext则寻找空闲的gpworker，调用method\_call\_teec\_inicont函数通过DBus向gpworker发送method call，其余GP API中的函数会寻找Initialize时对应的gpworker进行本地GP API函数调用。gpworker读取传入的参数直接调用本地GP API获得硬件返回结果，通过DBus传输回gpproxy，gpproxy再通过gRPC传输protobuf序列化后的数据至libteecc，完成远程函数调用。



**图 2 整体软件运行时序图**

1.3 软件第三方依赖说明

编译、安装libteecc、gpproxy时所需第三方依赖：

* gRPC(3.19.4.0) //该依赖用于libteecc与gpproxy进行通信
* dbus(1.12.16-15.oe1) //该依赖用于支持dbus通信
* openssl(1.1.1f-1.oe1) //该依赖用于证书的生成与校验
* yaml-cpp-devel(0.6.3-1.oe1) //该依赖用于配置文件的读取

编译、安装c-spiffe所需第三方依赖：

* cjose-devel(0.6.1-3.oe1)
* subunit-devel(1.3.0-4.oe1)
* curl-devel(7.71.1-2.oe1)
* check-devel(0.12.0-4.oe1)
* uriparser-devel(0.9.3-2.oe1)
* jansson-devel(2.13.1-1.oe1)

1.4 软件（方案）约束说明

1.4.1 硬件环境要求

服务端硬件环境要求运行的泰山服务器已经预置了TrustZone特性，即预装iTrustee 安全OS以及配套的BMC、BIOS固件，UOS版本为20-SP1-server-arm64及以上。

客户端硬件具有网络通讯条件即可。

1.4.2 操作系统要求

服务端操作系统要求如表1所示。

**表 1 操作系统要求**

| 项目 | 版本 | 获取方式 |
| --- | --- | --- |
| openEuler OS | 20.03-LTS-SP1-everything-aarch64或更新版本 | [https://repo.openeuler.org/openEuler-20.03-LTS-SP1/ISO/aarch64/](https://repo.openeuler.org/openEuler-20.03-LTS-SP1/ISO/aarch64/" \o " ) |
| Euler OS | V2.0SP9-aarch64或更新版本 | [https://cmc-szver-artifactory.cmc.tools.huawei.com/artifactory/cmc-builtools-prod//0591A5MZ/toolVersion/0593A77N/20210222105258729/Software/EulerOS-V2.0SP9-aarch64-dvd.iso](https://cmc-szver-artifactory.cmc.tools.huawei.com/artifactory/cmc-builtools-prod//0591A5MZ/toolVersion/0593A77N/20210222105258729/Software/EulerOS-V2.0SP9-aarch64-dvd.iso" \o " ) |

2 libteecc模块设计与实现

libteecc的使用者为容器或VM中的用户TrustZone CA程序，替换掉原来CA程序只能在本地调用的libteec库，使得用户CA应用最大化感知不到替换。由于一般TrustZone的CA程序都是使用C语言编写，libteecc也应封装成C语言接口库。libteecc支持全部REE侧的GP Client API接口，为了方便部署和进行访问控制，横向扩展了三个接口TEEC\_DeployTa、TEEC\_SetJwt、TEEC\_Unsetjwt。表2中列出了具体接口。

**表 2 GP Client接口**

|  |  |
| --- | --- |
| 接口 | 功能 |
| TEEC\_Result TEEC\_InitializeContext(  const char\* name, TEEC\_Context\* context) | 初始化一个安全世界环境，建立CA与TEE的联系 |
| void TEEC\_FinalizeContext( TEEC\_Context \*context) | 释放一个安全世界环境，断开CA与TEE的联系 |
| TEEC\_Result TEEC\_OpenSession( TEEC\_Context \*context, TEEC\_Session \*session, const TEEC\_UUID \*destination, uint32\_t connectionMethod, const void\* connectionData, TEEC\_Operation \*operation, uint32\_t \* returnOrigin) | CA通过调用此函数打开会话 |
| void TEEC\_CloseSession(  TEEC\_Session \*session) | CA通过调用此函数关闭会话 |
| TEEC\_Result TEEC\_InvokeCommand( TEEC\_Session \*session, uint32\_t commandID, TEEC\_Operation \*operation, uint32\_t \* returnOrigin) | CA向TA发送命令 |
| TEEC\_Result TEEC\_RegisterSharedMemory(  TEEC\_Context\* context,  TEEC\_SharedMemory\* sharedMem) | 注册已由CA创建的内存块。 |
| TEEC\_Result TEEC\_AllocateSharedMemory(  TEEC\_Context\* context,  TEEC\_SharedMemory\* sharedMem) | 申请临时内存。 |
| void TEEC\_ReleaseSharedMemory(  TEEC\_SharedMemory\* sharedMem) | 释放共享内存空间 |
| TEEC\_Result  TEEC\_DeployTa(  char \*infile\_path,  char \*subdir,  char \*outfile\_name  ) | 部署TA至服务端 |
| TEEC\_Result  TEEC\_SetJwt(  char \*token  ) | 客户端设置为需要jwt认证 |
| TEEC\_Result  TEEC\_UnsetJwt(  ) | 客户端设置为不需要jwt认证 |

libteecc基于gRPC Client实现，定义一个GreeterClient类greeter和一个全局通道变量gpp\_channel。gpp\_channel用于提供一个libteecc与gpproxy之间的TCP连接，在连接通道的基础上封装一个stub\_，真正实现RPC调用。greeter中包含有构造函数、调用GP service端GP Client接口的方法以及一个Greeter stub指针stub\_。调用GP service端GP Client接口的方法有grpc\_teec\_initializecontext、grpc\_teec\_finalizecontext、grpc\_teec\_opensession、grpc\_teec\_invokecommand、grpc\_teec\_closesession。在这些方法里，用变量Inicont\_Request、Inicont\_Reply来传递参数和获取返回，还有一个变量ClientContext用以存储上下文。将上述三个变量作为相应的stub\_->TEECC\_XXX方法的参数，进行RPC调用。stub\_->TEECC\_XXX方法由gpproxy所定义和注册。

此外，libteecc定义实现表 2中的标准GP Client接口和三个横向扩展接口TEEC\_DeployTa、TEEC\_SetJwt，TEEC\_Unsetjwt。CA调用TEEC\_InitializeContext接口时对greeter和gpp\_channel进行初始化。在TEEC\_InitializeContext、TEEC\_FinalizeContext、TEEC\_OpenSession、TEEC\_CloseSession、TEEC\_InvokeCommand、TEEC\_DeployTa函数中，调用greeter中的相应方法grpc\_teec\_xxx与gpproxy进行通信，得到返回值。

libteecc中的TEEC\_RegisterSharedMemory、TEEC\_AllocateSharedMemory、TEEC\_ReleaseSharedMemory三个接口无需进行RPC调用，只需在libteecc所部署的异构平台虚拟机/容器里进行内存管理即可。原本部署在本地Kunpeng服务器上的CA与TrustZone安全世界里的TA之间通过共享内存来传递数据，本项目的关键技术之一是在Trustzone 资源池化里如何让Kunpeng服务器TA对异构平台虚拟机/容器里的CA透明，CA调用libteecc中的接口依然可以用内容共享方式来传递数据，与在Kunpeng服务器上调用本地libteec安全兼容。

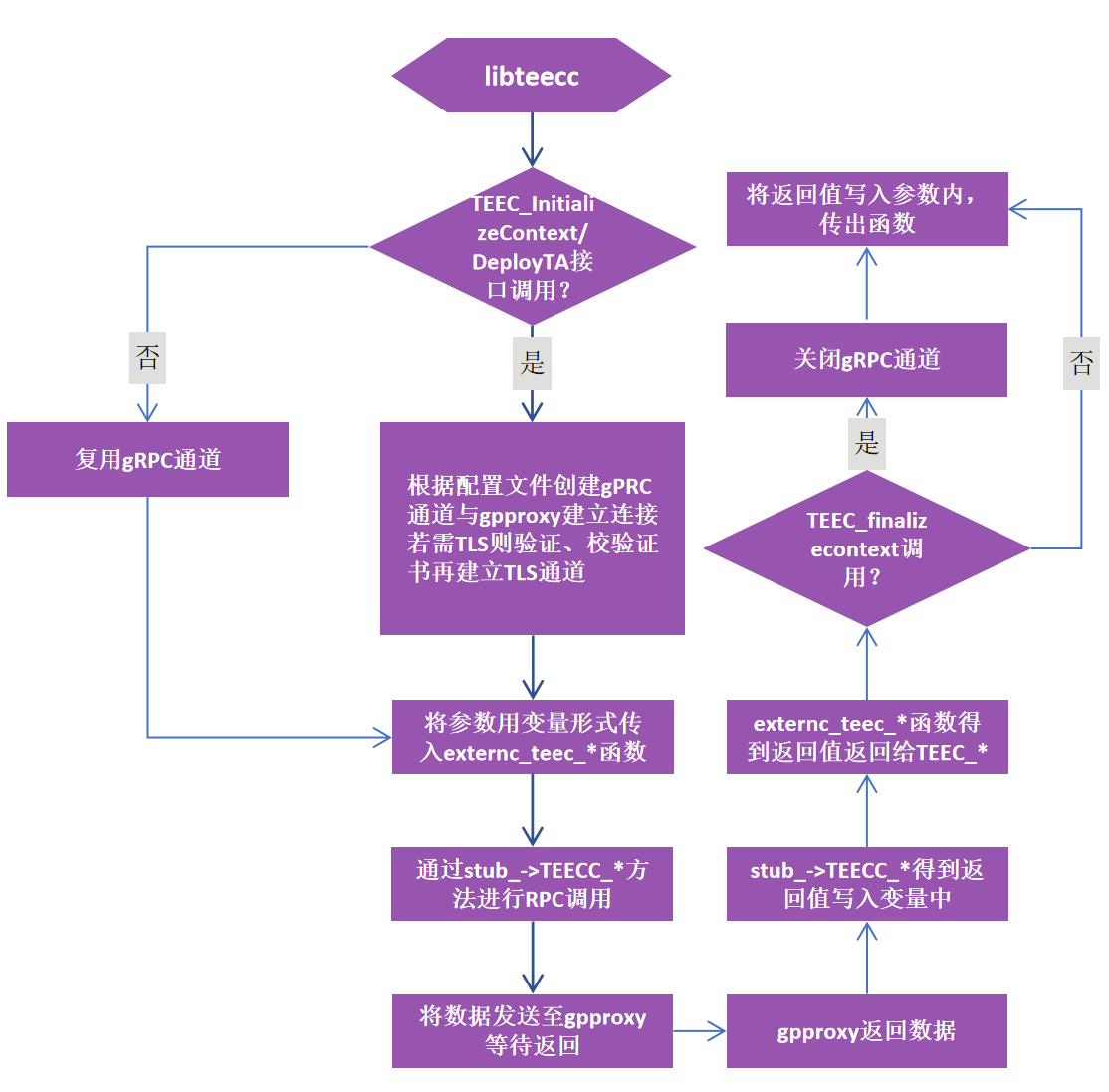
libteecc中的TEEC\_DeployTa接口用于用户将TA文件传输到GP service部署所在的Kunpeng服务器上，与其余端口传递参数方式相同，gRPC支持大文件传输，最大可支持1G文件。

libteecc中的TEEC\_Setswt用于接收用户CA传入的JWT，libteecc再将JWT作为一个附加参数加入到stub\_->TEECC\_XXX方法调用中。

libteecc中的TEEC\_Unsetjwt用于关闭客户端JWT认证，libteecc将JWT的值设为noToken，作为一个附加参数加入到stub\_->TEECC\_XXX方法调用中，传输给gpproxy标识客户端未开启JWT认证服务。

2.1 libteecc模块内运行时序

如图3 libteecc运行流程图所示，libteecc调用TEEC\_InitializeContext、TEEC\_FinalizeContext、TEEC\_OpenSession、TEEC\_InvokeCommand、TEEC\_CloseSession、TEEC\_DeployTa接口时，首先使用变量传递参数至函数内，其中包含token的数据，在TEEC\_InitializeContext，TEEC\_DeployTa接口中，若未创建gRPC通道，则根据配置文件选择对应的TLS通道进行创建，若无TLS则创建无TLS通道，若为仅服务端验证或双向验证时，读取证书文件并进行校验，创建对应TLS通道。创建通道后，在函数内将参数用变量传入至由GppClient提供的externc\_teec\_\*函数，在该函数中使用stub\_->TEECC\_XXX方法进行RPC调用，通过一个全局通道gpp\_channel将信息传输给gpproxy，经过gpproxy处理后返回给libteecc，再将获得的信息通过参数传递到函数外。



**图 3 libteecc运行流程图**

2.2 libteecc模块内接口与数据结构

**（1）libteecc模块内接口**

libteecc模块内部接口如表3 所示，接口均在GppClient类中实现，接口功能均为与gpproxy进行RPC通信，将数据发送至gpproxy并接收返回的数据。

**表 3 libteecc 模块内部接口**

|  |  |
| --- | --- |
| 接口 | 功能 |
| retstru\_teec\_inicont externc\_teec\_initializecontext(  std::uint8\_t \*name,  std::size\_t name\_size) | 将参数传递给gpproxy并接收相应的返回 |
| retstru\_teec\_fincont externc\_teec\_finalizecontext(  std::int32\_t in\_context\_fd,  std::uint8\_t \*in\_context\_tapath,  std::int32\_t in\_context\_tapath\_size,  std::uint64\_t in\_context\_sessionlist\_next,  std::uint64\_t in\_context\_sessionlist\_prev,  std::uint64\_t in\_context\_shrdmemlist\_next,  std::uint64\_t in\_context\_shrdmemlist\_prev,  std::uint64\_t in\_context\_sharebuffer\_buffer,  std::int64\_t in\_context\_sharebuffer\_bufferbarrier,  std::uint64\_t in\_context\_addr  ) | 将参数传递给gpproxy并接收相应的返回 |
| retstru\_teec\_opensession externc\_teec\_opensession(  std::int32\_t in\_context\_fd,  std::uint8\_t \*in\_context\_tapath,  std::int32\_t in\_context\_tapath\_size,  std::uint64\_t in\_context\_sessionlist\_next,  std::uint64\_t in\_context\_sessionlist\_prev,  std::uint64\_t in\_context\_shrdmemlist\_next,  std::uint64\_t in\_context\_shrdmemlist\_prev,  std::uint64\_t in\_context\_sharebuffer\_buffer,  std::int64\_t in\_context\_sharebuffer\_bufferbarrier,  std::uint32\_t in\_destination\_timelow,  std::uint32\_t in\_destination\_timemid,  std::uint32\_t in\_destination\_timehiandver,  std::uint8\_t \*in\_destination\_clockseqandnode,  std::int32\_t in\_destination\_clockseqandnode\_size,  std::uint32\_t in\_connectionmethod,  std::uint64\_t in\_connectiondata,  std::uint32\_t in\_operation\_started,  std::uint32\_t in\_operation\_paramtypes,  std::uint64\_t in\_operation\_param1\_tmpref\_buffer,  std::uint32\_t in\_operation\_param1\_tmpref\_size,  std::uint64\_t in\_operation\_param1\_memref\_parent,  std::uint32\_t in\_operation\_param1\_memref\_size,  std::uint32\_t in\_operation\_param1\_memref\_offset,  std::uint32\_t in\_operation\_param1\_value\_a,  std::uint32\_t in\_operation\_param1\_value\_b,  std::int32\_t in\_operation\_param1\_ionref\_ionsharefd,  std::uint32\_t in\_operation\_param1\_ionref\_ionsize,  std::uint64\_t in\_operation\_param2\_tmpref\_buffer,  std::uint32\_t in\_operation\_param2\_tmpref\_size,  std::uint64\_t in\_operation\_param2\_memref\_parent,  std::uint32\_t in\_operation\_param2\_memref\_size,  std::uint32\_t in\_operation\_param2\_memref\_offset,  std::uint32\_t in\_operation\_param2\_value\_a,  std::uint32\_t in\_operation\_param2\_value\_b,  std::int32\_t in\_operation\_param2\_ionref\_ionsharefd,  std::uint32\_t in\_operation\_param2\_ionref\_ionsize,  std::uint64\_t in\_operation\_param3\_tmpref\_buffer,  std::uint32\_t in\_operation\_param3\_tmpref\_size,  std::uint64\_t in\_operation\_param3\_memref\_parent,  std::uint32\_t in\_operation\_param3\_memref\_size,  std::uint32\_t in\_operation\_param3\_memref\_offset,  std::uint32\_t in\_operation\_param3\_value\_a,  std::uint32\_t in\_operation\_param3\_value\_b,  std::int32\_t in\_operation\_param3\_ionref\_ionsharefd,  std::uint32\_t in\_operation\_param3\_ionref\_ionsize,  std::uint64\_t in\_operation\_param4\_tmpref\_buffer,  std::uint32\_t in\_operation\_param4\_tmpref\_size,  std::uint64\_t in\_operation\_param4\_memref\_parent,  std::uint32\_t in\_operation\_param4\_memref\_size,  std::uint32\_t in\_operation\_param4\_memref\_offset,  std::uint32\_t in\_operation\_param4\_value\_a,  std::uint32\_t in\_operation\_param4\_value\_b,  std::int32\_t in\_operation\_param4\_ionref\_ionsharefd,  std::uint32\_t in\_operation\_param4\_ionref\_ionsize,  std::uint64\_t in\_operation\_session,  std::int32\_t in\_operation\_cancelflag,  std::uint32\_t in\_returnorigin,  std::uint64\_t in\_context\_addr) | 将参数传递给gpproxy并接收相应的返回 |
| retstru\_teec\_invokecommand externc\_teec\_invokecommand(  std::uint32\_t in\_session\_sessionid,  std::uint32\_t in\_session\_serviceid\_timelow,  std::uint32\_t in\_session\_serviceid\_timemid,  std::uint32\_t in\_session\_serviceid\_timehiandver,  std::uint8\_t \*in\_session\_serviceid\_clockseqandnode,  std::uintptr\_t in\_session\_serviceid\_clockseqandnode\_size,  std::uint32\_t in\_session\_opscnt,  std::uint64\_t in\_session\_head\_next,  std::uint64\_t in\_session\_head\_prev,  std::uint64\_t in\_session\_context,  std::uint32\_t in\_commandid,  std::uint32\_t in\_operation\_started,  std::uint32\_t in\_operation\_paramtypes,  std::uint64\_t in\_operation\_param1\_tmpref\_buffer,  std::uint32\_t in\_operation\_param1\_tmpref\_size,  std::uint64\_t in\_operation\_param1\_memref\_parent,  std::uint32\_t in\_operation\_param1\_memref\_parent\_flag,  std::uint32\_t in\_operation\_param1\_memref\_size,  std::uint32\_t in\_operation\_param1\_memref\_offset,  std::uint32\_t in\_operation\_param1\_value\_a,  std::uint32\_t in\_operation\_param1\_value\_b,  std::int32\_t in\_operation\_param1\_ionref\_ionsharefd,  std::uint32\_t in\_operation\_param1\_ionref\_ionsize,  std::uint64\_t in\_operation\_param2\_tmpref\_buffer,  std::uint32\_t in\_operation\_param2\_tmpref\_size,  std::uint64\_t in\_operation\_param2\_memref\_parent,  std::uint32\_t in\_operation\_param2\_memref\_parent\_flag,  std::uint32\_t in\_operation\_param2\_memref\_size,  std::uint32\_t in\_operation\_param2\_memref\_offset,  std::uint32\_t in\_operation\_param2\_value\_a,  std::uint32\_t in\_operation\_param2\_value\_b,  std::int32\_t in\_operation\_param2\_ionref\_ionsharefd,  std::uint32\_t in\_operation\_param2\_ionref\_ionsize,  std::uint64\_t in\_operation\_param3\_tmpref\_buffer,  std::uint32\_t in\_operation\_param3\_tmpref\_size,  std::uint64\_t in\_operation\_param3\_memref\_parent,  std::uint32\_t in\_operation\_param3\_memref\_parent\_flag,  std::uint32\_t in\_operation\_param3\_memref\_size,  std::uint32\_t in\_operation\_param3\_memref\_offset,  std::uint32\_t in\_operation\_param3\_value\_a,  std::uint32\_t in\_operation\_param3\_value\_b,  std::int32\_t in\_operation\_param3\_ionref\_ionsharefd,  std::uint32\_t in\_operation\_param3\_ionref\_ionsize,  std::uint64\_t in\_operation\_param4\_tmpref\_buffer,  std::uint32\_t in\_operation\_param4\_tmpref\_size,  std::uint64\_t in\_operation\_param4\_memref\_parent,  std::uint32\_t in\_operation\_param4\_memref\_parent\_flag,  std::uint32\_t in\_operation\_param4\_memref\_size,  std::uint32\_t in\_operation\_param4\_memref\_offset,  std::uint32\_t in\_operation\_param4\_value\_a,  std::uint32\_t in\_operation\_param4\_value\_b,  std::int32\_t in\_operation\_param4\_ionref\_ionsharefd,  std::uint32\_t in\_operation\_param4\_ionref\_ionsize,  std::uint64\_t in\_operation\_session,  std::int32\_t in\_operation\_cancelflag,  std::uint32\_t in\_returnorigin,  std::uint8\_t \*in\_buffer1,  std::uintptr\_t in\_buffer1\_size,  std::uint8\_t \*in\_buffer2,  std::uintptr\_t in\_buffer2\_size,  std::uint8\_t \*in\_buffer3,  std::uintptr\_t in\_buffer3\_size,  std::uint8\_t \*in\_buffer4,  std::uintptr\_t in\_buffer4\_size  ) | 将参数传递给gpproxy并接收相应的返回 |
| retstru\_teec\_closesession externc\_teec\_closesession(  std::uint32\_t in\_session\_sessionid,  std::uint32\_t in\_session\_serviceid\_timelow,  std::uint32\_t in\_session\_serviceid\_timemid,  std::uint32\_t in\_session\_serviceid\_timehiandver,  std::uint8\_t \*in\_session\_serviceid\_clockseqandnode,  std::uintptr\_t in\_session\_serviceid\_clockseqandnode\_size,  std::uint32\_t in\_session\_opscnt,  std::uint64\_t in\_session\_head\_next,  std::uint64\_t in\_session\_head\_prev,  std::uint64\_t in\_session\_context  ) | 将参数传递给gpproxy并接收相应的返回 |
| int Upload(  std::string infile\_path,  std::string subdir,  std::string outfile\_name  ) | 将参数传递给gpproxy，并接收相应的返回 |

（2）libteecc模块内数据结构

libteecc模块内数据结构表4 所示。

**表 4 libteecc 模块内数据结构**

|  |  |
| --- | --- |
| 数据结构 | 功能 |
| struct retstru\_teec\_inicont {  uint32\_t teecresult;  int32\_t context\_fd;  uint8\_t \*context\_tapath;  uintptr\_t context\_tapath\_outsize;  uint64\_t context\_sessionlist\_next;  uint64\_t context\_sessionlist\_prev;  uint64\_t context\_shrdmemlist\_next;  uint64\_t context\_shrdmemlist\_prev;  uint64\_t context\_sharebuffer\_buffer;  int64\_t context\_sharebuffer\_bufferbarrier;  uint64\_t context\_addr;  int32\_t flag;  } | 数据结构中包含了gpproxy的返回值，其中flag为jwt认证的标志位 |
| struct retstru\_teec\_fincont {  int32\_t context\_fd;  uint8\_t \*context\_tapath;  uintptr\_t context\_tapath\_outsize;  uint64\_t context\_sessionlist\_next;  uint64\_t context\_sessionlist\_prev;  uint64\_t context\_shrdmemlist\_next;  uint64\_t context\_shrdmemlist\_prev;  uint64\_t context\_sharebuffer\_buffer;  int64\_t context\_sharebuffer\_bufferbarrier;  int32\_t flag;  } | 数据结构中包含了gpproxy的返回值，其中flag为jwt认证的标志位 |
| struct retstru\_teec\_opensession {  uint32\_t teecresult;  int32\_t context\_fd;  uint8\_t \*context\_tapath;  uintptr\_t context\_tapath\_outsize;  uint64\_t context\_sessionlist\_next;  uint64\_t context\_sessionlist\_prev;  uint64\_t context\_shrdmemlist\_next;  uint64\_t context\_shrdmemlist\_prev;  uint64\_t context\_sharebuffer\_buffer;  int64\_t context\_sharebuffer\_bufferbarrier;  uint32\_t session\_sessionid;  uint32\_t session\_serviceid\_timelow;  uint32\_t session\_serviceid\_timemid;  uint32\_t session\_serviceid\_timehiandver;  uint8\_t \*session\_serviceid\_clockseqandnode;  uintptr\_t session\_serviceid\_clockseqandnode\_outsize;  uint32\_t session\_opscnt;  uint64\_t session\_head\_next;  uint64\_t session\_head\_prev;  uint64\_t session\_context;  uint32\_t operation\_started;  uint32\_t operation\_paramtypes;  uint64\_t operation\_param1\_tmpref\_buffer;  uint32\_t operation\_param1\_tmpref\_size;  uint64\_t operation\_param1\_memref\_parent;  uint32\_t operation\_param1\_memref\_size;  uint32\_t operation\_param1\_memref\_offset;  uint32\_t operation\_param1\_value\_a;  uint32\_t operation\_param1\_value\_b;  int32\_t operation\_param1\_ionref\_ionsharefd;  uint32\_t operation\_param1\_ionref\_ionsize;  uint64\_t operation\_param2\_tmpref\_buffer;  uint32\_t operation\_param2\_tmpref\_size;  uint64\_t operation\_param2\_memref\_parent;  uint32\_t operation\_param2\_memref\_size;  uint32\_t operation\_param2\_memref\_offset;  uint32\_t operation\_param2\_value\_a;  uint32\_t operation\_param2\_value\_b;  int32\_t operation\_param2\_ionref\_ionsharefd;  uint32\_t operation\_param2\_ionref\_ionsize;  uint64\_t operation\_param3\_tmpref\_buffer;  uint32\_t operation\_param3\_tmpref\_size;  uint64\_t operation\_param3\_memref\_parent;  uint32\_t operation\_param3\_memref\_size;  uint32\_t operation\_param3\_memref\_offset;  uint32\_t operation\_param3\_value\_a;  uint32\_t operation\_param3\_value\_b;  int32\_t operation\_param3\_ionref\_ionsharefd;  uint32\_t operation\_param3\_ionref\_ionsize;  uint64\_t operation\_param4\_tmpref\_buffer;  uint32\_t operation\_param4\_tmpref\_size;  uint64\_t operation\_param4\_memref\_parent;  uint32\_t operation\_param4\_memref\_size;  uint32\_t operation\_param4\_memref\_offset;  uint32\_t operation\_param4\_value\_a;  uint32\_t operation\_param4\_value\_b;  int32\_t operation\_param4\_ionref\_ionsharefd;  uint32\_t operation\_param4\_ionref\_ionsize;  uint64\_t operation\_session;  int32\_t operation\_cancelflag;  uint32\_t returnorigin;  int32\_t flag;  } | 数据结构中包含了gpproxy的返回值，其中flag为jwt认证的标志位 |
| struct retstru\_teec\_closesession {  uint32\_t session\_sessionid;  uint32\_t session\_serviceid\_timelow;  uint32\_t session\_serviceid\_timemid;  uint32\_t session\_serviceid\_timehiandver;  uint8\_t \*session\_serviceid\_clockseqandnode;  uintptr\_t session\_serviceid\_clockseqandnode\_outsize;  uint32\_t session\_opscnt;  uint64\_t session\_head\_next;  uint64\_t session\_head\_prev;  uint64\_t session\_context;  int32\_t flag;  } | 数据结构中包含了gpproxy的返回值，其中flag为jwt认证的标志位 |
| struct retstru\_teec\_invokecommand {  uint32\_t teecresult;  uint32\_t session\_sessionid;  uint32\_t session\_serviceid\_timelow;  uint32\_t session\_serviceid\_timemid;  uint32\_t session\_serviceid\_timehiandver;  uint8\_t \*session\_serviceid\_clockseqandnode;  uintptr\_t session\_serviceid\_clockseqandnode\_outsize;  uint32\_t session\_opscnt;  uint64\_t session\_head\_next;  uint64\_t session\_head\_prev;  uint64\_t session\_context;  uint32\_t operation\_started;  uint32\_t operation\_paramtypes;  uint64\_t operation\_param1\_tmpref\_buffer;  uint32\_t operation\_param1\_tmpref\_size;  uint64\_t operation\_param1\_memref\_parent;  uint32\_t operation\_param1\_memref\_parent\_flag;  uint32\_t operation\_param1\_memref\_size;  uint32\_t operation\_param1\_memref\_offset;  uint32\_t operation\_param1\_value\_a;  uint32\_t operation\_param1\_value\_b;  int32\_t operation\_param1\_ionref\_ionsharefd;  uint32\_t operation\_param1\_ionref\_ionsize;  uint64\_t operation\_param2\_tmpref\_buffer;  uint32\_t operation\_param2\_tmpref\_size;  uint64\_t operation\_param2\_memref\_parent;  uint32\_t operation\_param2\_memref\_parent\_flag;  uint32\_t operation\_param2\_memref\_size;  uint32\_t operation\_param2\_memref\_offset;  uint32\_t operation\_param2\_value\_a;  uint32\_t operation\_param2\_value\_b;  int32\_t operation\_param2\_ionref\_ionsharefd;  uint32\_t operation\_param2\_ionref\_ionsize;  uint64\_t operation\_param3\_tmpref\_buffer;  uint32\_t operation\_param3\_tmpref\_size;  uint64\_t operation\_param3\_memref\_parent;  uint32\_t operation\_param3\_memref\_parent\_flag;  uint32\_t operation\_param3\_memref\_size;  uint32\_t operation\_param3\_memref\_offset;  uint32\_t operation\_param3\_value\_a;  uint32\_t operation\_param3\_value\_b;  int32\_t operation\_param3\_ionref\_ionsharefd;  uint32\_t operation\_param3\_ionref\_ionsize;  uint64\_t operation\_param4\_tmpref\_buffer;  uint32\_t operation\_param4\_tmpref\_size;  uint64\_t operation\_param4\_memref\_parent;  uint32\_t operation\_param4\_memref\_parent\_flag;  uint32\_t operation\_param4\_memref\_size;  uint32\_t operation\_param4\_memref\_offset;  uint32\_t operation\_param4\_value\_a;  uint32\_t operation\_param4\_value\_b;  int32\_t operation\_param4\_ionref\_ionsharefd;  uint32\_t operation\_param4\_ionref\_ionsize;  uint64\_t operation\_session;  int32\_t operation\_cancelflag;  uint32\_t returnorigin;  uint8\_t \*buffer1;  uintptr\_t buffer1\_outsize;  uint8\_t \*buffer2;  uintptr\_t buffer2\_outsize;  uint8\_t \*buffer3;  uintptr\_t buffer3\_outsize;  uint8\_t \*buffer4;  uintptr\_t buffer4\_outsize;  int32\_t flag;  } | 数据结构中包含了gpproxy的返回值，其中flag为jwt认证的标志位 |

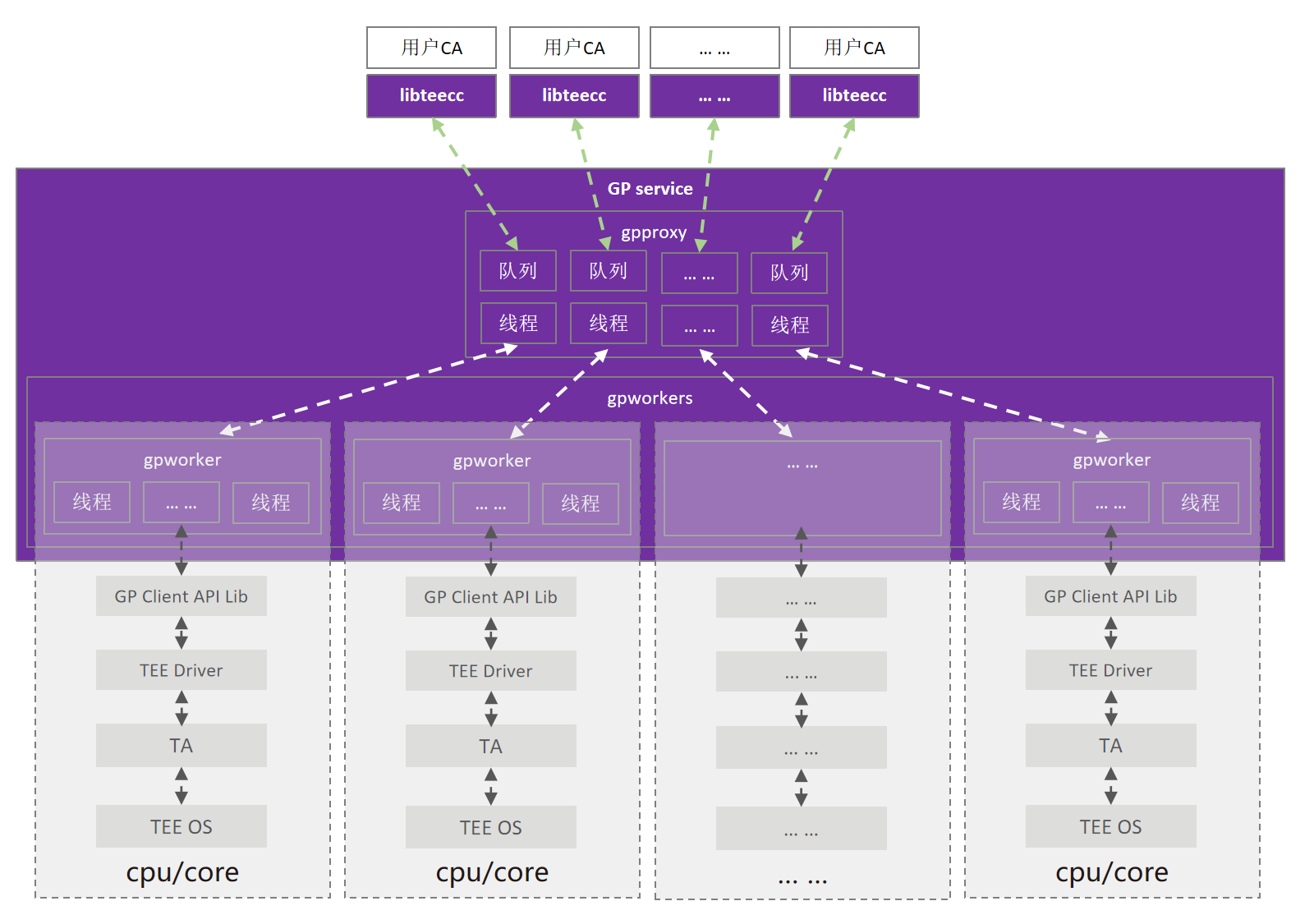
3 GP service设计与实现

本项目拟构建的Trustzone 资源池化试图将多个异构平台上容器/VM中的CA调用，通过Kunpeng服务器本地的代理统一调用，使用硬件TrustZone功能。Kunpeng服务器TrustZone的使用需要在REE侧部署一个CA，由CA发起TEE OS对相应TA的初始化。一台Kunpeng服务器上可能存在多个CA与多个TA一一对应，也可能多个CA对应同一个TA。TrustZone的工作原理是对一个CPU/CORE进行安全态和非安全态的分时复用，即CA与TA分享一个CPU/CORE，因此CA初始化在哪个CPU/CORE上，TA也将运行在哪个CPU/CORE上。

通常不同用户在异构平台上容器/VM中的CA，在Kunpeng服务器上会部署有不同的TA与之对应。若所有异构平台上的CA对Kunpeng服务器上相应TA的调用，都由Trustzone 资源池化在Kunpeng服务器上统一的代理GP service转发，GP service本身仅是一个进程，会导致所有TA都运行在一个CPU/CORE上，这将大大影并发性能。

为解决这个问题，本项目拟将GP service分为gpproxy和gpworkers两层，如图四所示。gpproxy基于gRPC Server实现，接收识别由libteecc转发而来异构平台上容器/VM中CA的GP调用。gpworkers是一个“进程池”，包含多个gpworker，一个gpworker对应一个进程，会初始化在不同的CPU/CORE上，这样不同的TA也会运行在不同的CPU/CORE上，从而优化多任务并发实现。gpproxy通过DBus将GP调用转发到本地相应的gpworker，gpworker再将调用转发给其所在Kunpeng服务器上物理TEE调用库libteec。调用结果由gpworker、gpproxy、libteecc返回给用户CA应用。

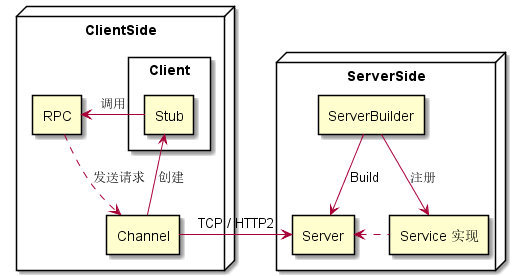
由于可能存在多个异构平台容器/虚拟机CA同时进行GP Client API调用的情况，gpproxy基于异步多线程gRPC Server实现，为每一个线程各自创建一个GP Client API调用队列，进行多线程的并发处理。每个线程再将各自队列里的调用转发给相应的gpworker。为保证性能，进程池gpworkers里的Worker进程预先创建好，而不是有调用时临时创建。考虑到gpproxy至少要占用一个CPU/CORE，作为业务入口，其计算资源应优先得到保障，故gpworker的数量默认为Kunpeng服务器上最大CPU/CORE数目减1。后续可根据服务器上的其他应用的使用情况，动态调整gpworker的数量，可以支持亲核性，只在指定的几个核上开展业务。



**图 4 GP service层次结构**

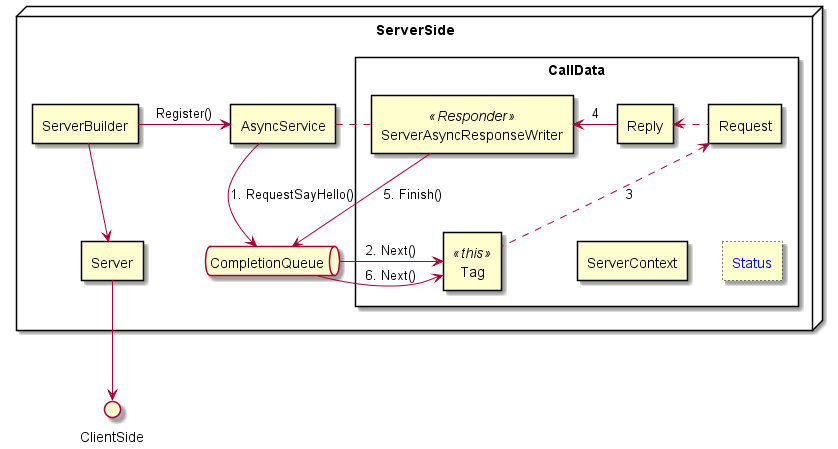
3.1 基于异步多线程gRPC Service的gpproxy

gRPC是RPC框架的一种，是一个高性能、开源和通用的RPC框架，基于ProtoBuf(Protocol Buffers) 序列化协议开发，且支持众多开发语言。基于HTTP/2设计，带来诸如双向流、流控、头部压缩、单TCP连接上的多复用请求等特。



**图 5 gRPC基础概念图**

如图5 所示，客户端创建一个stub进行RPC调用，Channel提供一个与 gRPC server 的主机和端口建立的连接。Server 端需要实现对应的 RPC，所有的 RPC 组成了 Service，Server 的创建需要一个 Builder，添加上监听的地址和端口，注册上该端口上绑定的服务，最后构建出Server并启动。在本项目中服务端使用了如图6所示的异步gRPC框架，并加入了多线程服务。



**图 6 异步gRPC Service**

gpproxy在开始运行时创建一个grpc ServerImpl类实例server。Server包含：构造函数、析构函数、异步处理相关的CallData类与HandleRpcs方法、开启服务的RunServer方法、两个用于监测与处理session与context超时的方法session\_timeout\_process与context\_timeout\_process以及一些私有变量。私有变量由队列向量vector cq\_、用于异步服务的AsyncService对象service\_、Server指针server\_、用于多线程存储的thread容器server\_threads\_、线程互斥变量mutex\_workerrec、线程等待条件变量cond\_notbusy、存储gpworkers信息的workerrec数组。

然后gpproxy执行server的RunServer方法开启服务。RunServer创建一个gRPC ServerBuilder类实例builder，为其添加上监听的地址和端口，注册异步服务service\_，使用builder.AddCompletionQueue()初始化队列向量cq\_，然后构建出server\_并启动监听服务。接着将session\_timeout\_process与context\_timeout\_process、多个RPC处理方法HandleRpcs加入到线程容器server\_threads\_中。每个RPC处理线程都拥有一个独立队列。

在每个RPC处理线程HandleRpcs方法里，为每一个远程调用接口创建一个CallData类实例，向CallData构造函数传入以下参数：AsyncService service\_；ServerCompletionQueue cq\_[i]，存储请求队列；SS\_TEECC\_InitializeContext，为CallData里的枚举类型数据，标记CallData实例属于哪一个远程调用接口；Inicont\_Request inicont\_request，接收Client端发送来的请求参数；pthread\_mutex\_t mutex\_workerrec，线程互斥控制；pthread\_cond\_t cond\_notbusy，线程等待控制；wr\_t workerrec，用于选择哪一个gpworker。CallData构造函数调用CREATE状态的Process方法。在CREATE状态的Process方法里，先将Process的状态置为PROCESS，根据枚举类型，登记远程调用：service\_->RequestTEECC\_InitializeContext、service\_->RequestTEECC\_FinalizeContext、service\_->RequestTEECC\_OpenSession、service\_->RequestTEECC\_InvokeCommand、service\_->RequestTEECC\_CloseSession。注册时包含以下几个参数：ServerContext ctx\_；xxx\_request，接收libteecc端发送来的数据；xxx\_response，gpproxy进行处理后返回的数据；ServerCompletionQueue \*cq\_，请求队列；指向自身的指针this。为每个远程调用创建CallData类实例完毕后，HandleRpcs方法用一个void类型的tag指针来获取一个具体CallData实例，用一个bool型的ok判断队列中是否还有请求，然后循环监听cq\_[i]队列中是否有任务，将请求通过tag参数传入CallData中的Process函数进行处理。

当有新的GP Client API调用请求经libteecc传入gpproxy时，与请求通道对应有一个处理线程，线程对应有队列，队列cq\_[i]->Next()的阻塞结束并返回，对应CallData类对象实例的地址存储在tag指针中，然后就是具体CallData类对象实例的PROCESS状态的Process方法来进行响应处理。在PROCESS状态的Process方法里，创建一个新的CallData类对象实例以接收新的调用请求，将Process状态设置为FINISH。根据前面注册的枚举类型进行对应的调用处理。处理完毕设置返回的xxx\_reply；调用xxx\_response.Finish(xxx\_reply,Status::OK,this)将返回发送给libteecc。发送完毕，调用tag对应CallData对象实例FINISH的Process，清除掉自己，一个调用请求处理完成。

CallData类对象的PROCESS态Process方法里，根据调用请求枚举类型数据的不同做相应处理。枚举类型数据包括SS\_TEECC\_InitializeContext、SS\_TEECC\_FinalizeContext、SS\_TEECC\_OpenSession、SS\_TEECC\_CloseSession SS\_TEECC\_InvokeCommand和SS\_TEECC\_DeployTa。在处理具体的调用请求时，先调用回调函数dbusmethodcall\_validate\_jwt判断JWT是否正确，若JWT错误，则直接进行错误返回。

对SS\_TEECC\_InitializeContext型调用请求进行处理时，根据workerrec数组里的记录，向第一个空闲的gpworker[i]发送DBus调用method\_call\_teec\_inicont，并将workerrec[i]的busy标志标记为1。若没有空闲的gpworker，则根据条件变量cond\_notbusy进行等待。若method\_call\_teec\_inicont的返回值为0即DBus调用成功，则将workerrec[i]的context\_fd设置为TA的fd，设置一个独特的context\_addr，context\_createtime设置为当前时间。若method\_call\_teec\_inicont的返回值不为0，则将workerrec[i]的busy标志标记为0，表示gpworker[i]已空闲；

对SS\_TEECC\_OpenSession型调用请求进行处理时，根据传入的context\_fd、context\_addr在workerrec数组里进行匹配，找到相应的gpworker[i]，向其发送DBus调用method\_call\_teec\_opensession。若method\_call\_teec\_opensession的返回值为0即DBus调用成功，则将workerrec[i]的session列表里加入一个元素，包括session\_id、session\_createtime，并更新workerrec[i].context\_createtime设置为当前时间。

对SS\_TEECC\_InvokeCommand型调用请求进行处理时，根据传入的context\_fd、context\_addr、session\_id在workerrec数组里进行匹配，找到相应的gpworker[i]，向其发送DBus调用method\_call\_teec\_invokecommand。并更新workerrec[i].context\_createtime及相应会话的session\_createtime为当前时间。

对SS\_TEECC\_CloseSession型调用请求进行处理时，根据传入的context\_fd、context\_addr、session\_id在workerrec数组里进行匹配，找到相应的gpworker[i]，向其发送DBus调用method\_call\_teec\_closesession。然后删除workerrec[i]的session列表里对应的元素，并更新workerrec[i].context\_createtime设置为当前时间。

对SS\_TEECC\_FinalizeContext型调用请求进行处理时，根据传入的context\_fd、context\_addr在workerrec数组里进行匹配，找到相应的gpworker[i]。若session列表不为空则先调用method\_call\_teec\_closesession关闭相应session。向gpworker[i]发送DBus调用method\_call\_teec\_ fincont。然后将workerrec[i]的busy标志标记为0，表示gpworker[i]已空闲，并使用pthread\_cond\_signal(cond\_notbusy\_)通知其他等待线程；重置workerrec[i]的context\_fd、context\_addr、sessionid\_count、first、last数据。

对SS\_TEECC\_DeployTa型调用请求进行处理时，通过gRPC接收数据块，使用ofstream.write拼接成文件，存储至服务器的指定位置。

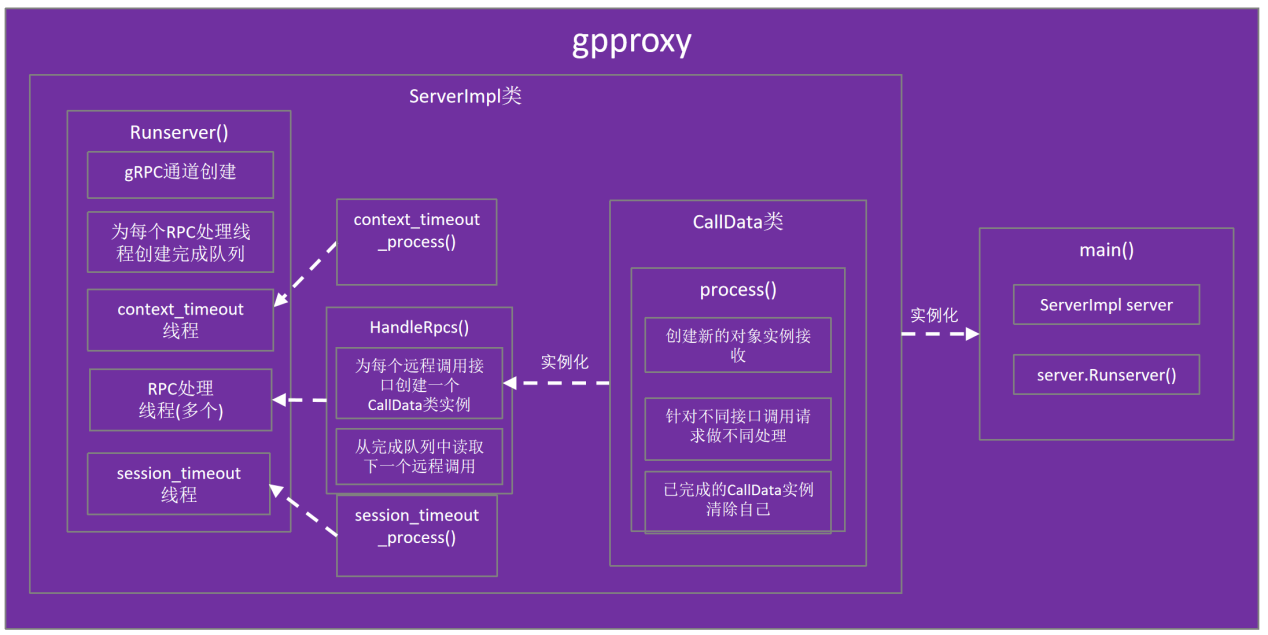
session\_timeout\_process线程判断各workerrec[i]会话列表里各元素的session\_createtime是否超时，若超时则会向相应的gpworker[i]发送DBus调用method\_call\_teec\_closesession，然后删除workerrec[i]的session列表里对应的元素，并更新workerrec[i].context\_createtime设置为当前时间。

context\_timeout\_process线程判断各workerrec[i]的context\_createtime是否超时，若超时，若session列表不为空则先调用method\_call\_teec\_closesession关闭相应session。向gpworker[i]发送DBus调用method\_call\_teec\_ fincont。然后将workerrec[i]的busy标志标记为0，表示gpworker[i]已空闲，并使用pthread\_cond\_signal(cond\_notbusy\_)通知其他等待线程；重置workerrec[i]的context\_fd、context\_addr、sessionid\_count、first、last数据。

这些线程共享workerrec数组，每个线程在对workerrec数组进行读、写操作时，先调用pthread\_mutex\_lock(mutex\_workerrec\_)对其他线程进行阻塞，保证正确性，操作完毕后调用pthread\_mutex\_unlock(mutex\_workerrec\_)进行解锁。

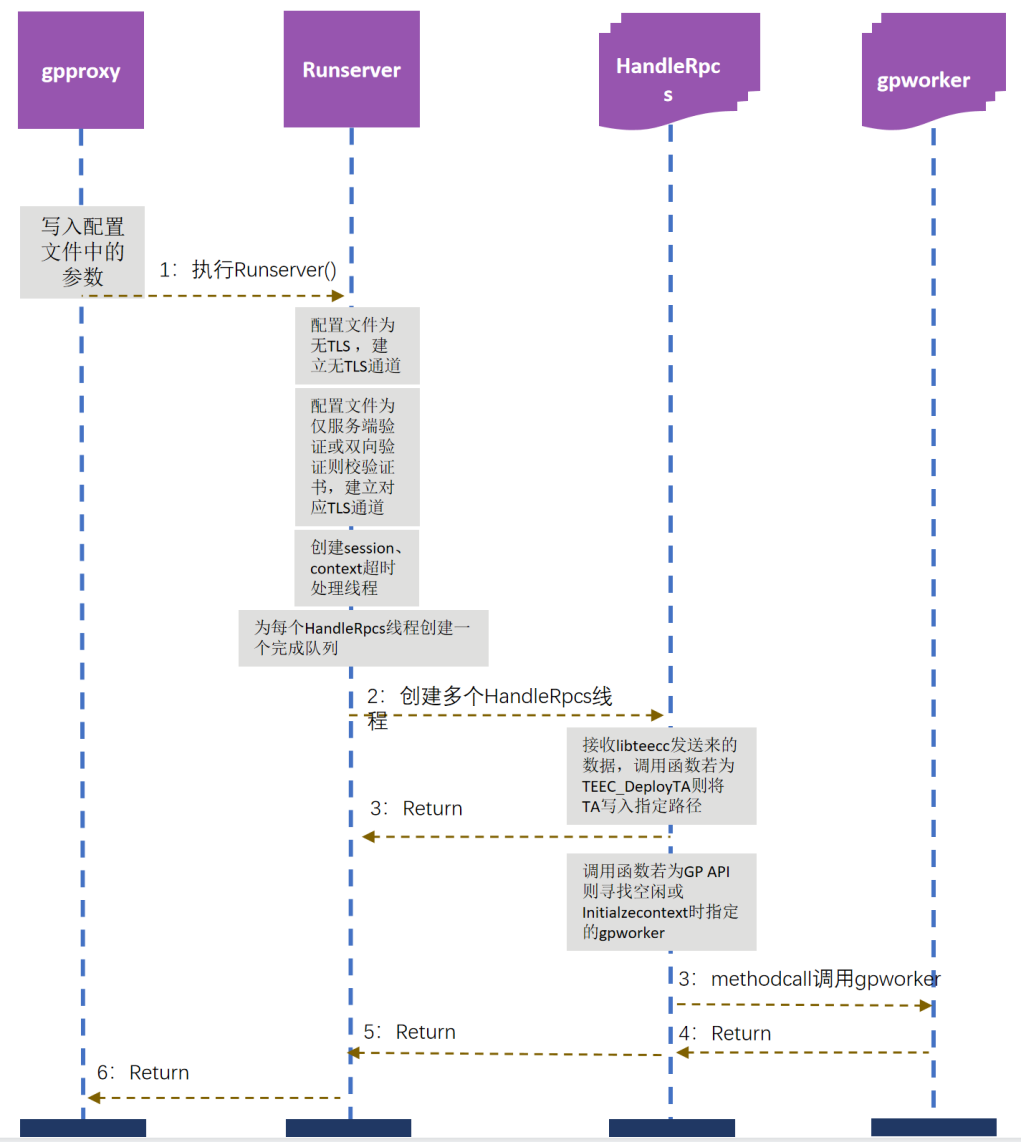
3.2 gpproxy模块内运行时序

如图7所示，gpproxy在启动时会创建多个具有相同功能的RPC处理线程，与两个超时处理线程，RPC处理线程中会为每个远程调用创建一个CallData类实例，进而调用类中的process()对远程调用进行对应处理，这些RPC处理现场会同时监听同一个端口，等待libteecc发来的远程调用。



**图7 gpproxy模块组织结构图**

如图8所示，gpproxy中包含libteecc发送来的RPC调用会以轮询的方式发送至某个线程，该线程内排队处理由同一个gpp\_channel发送来的RPC调用。接收到RPC调用后，读取发送来的数据，若需要jwt认证，则调用dbusmethodcall\_validate\_jwt向dbuss\_validate\_jwt发送method\_call进行jwt认证，认证通过后，在TA部署接口中，若存在文件名相同的TA文件则对比远程文件与本地文件的sha256值，若部署路 径不存在则创建该目录。在其余接口中，则寻找符合条件的gpworker，若找到则通过参数传递的方式将数据传输至method\_call\_teec\_\*函数，调用gpworker，由gpworker与硬件交互获得真实TEEC接口调用的返回值后通过DBus返回gpproxy。gpproxy则将返回的数据通过gpp\_channel返回给libteecc。



**图 8 gpproxy模块运行时序图**

3.3 gpproxy模块内接口与数据结构

（1）gpproxy模块内接口

gpproxy模块内部接口如表5 所示，接口功能为接收远程调用，并根据不同的远程调用进行不同的处理并返回数据至libteecc。

**表 5 gpproxy模块内部接口**

|  |  |
| --- | --- |
| 接口 | 功能 |
| void RunServer() | 在ServerImpl类中定义，用于gpproxy的初始化与启动 |
| void HandleRpcs(int i) | 在ServerImpl类中定义，用于RPC调用初始化与排队处理 |
| void session\_timeout\_process(  pthread\_mutex\_t \*mutex\_workerrec,  wr\_t \*workerrec  ) | 在ServerImpl类中定义，用于session的超时处理 |
| void context\_timeout\_process(  pthread\_mutex\_t \*mutex\_workerrec,  pthread\_cond\_t \*cond\_notbusy,  wr\_t \*workerrec  ) | 在ServerImpl类中定义，用于context的超时处理 |
| void Process() | 在CallData类中定义，用于RPC调用的处理 |

（2）gpproxy模块内数据结构

gpproxy模块内部数据结构如表6 所示。

**表 6 gpproxy模块内部数据结构**

|  |  |
| --- | --- |
| 数据结构 | 功能 |
| typedef struct sessionid\_node  {  uint32\_t session\_id;  struct timeval session\_createtime;  struct sessionid\_node \*next;  struct sessionid\_node \*prev;  } sin\_t | 记录session排队信息与在队中等待的时间 |
| typedef struct worker\_rec  {  uint8\_t busy;  int32\_t context\_fd;  uint64\_t context\_addr;  struct timeval context\_createtime;  int sessionid\_count;  sin\_t \*first;  sin\_t \*last;  } wr\_t | 记录gpworker的基本信息 |
| std::vector <std::unique\_ptr<ServerCompletionQueue>> cq\_ | 在ServerImpl类中定义，用于储存RPC调用的完成队列 |
| gpp::AsyncService service\_ | 在ServerImpl类中定义，用于定义gRPC的异步服务 |
| std::unique\_ptr <Server> server\_ | 在ServerImpl类中定义，用于记录gRPC builder的信息 |
| std::vector <std::thread> server\_threads\_ | 在ServerImpl类中定义，用于储存gpproxy中创建的线程 |
| pthread\_mutex\_t mutex\_workerrec | 在ServerImpl类中定义，用于在对workerrec数组操作时的线程阻塞 |
| pthread\_cond\_t cond\_notbusy | 在ServerImpl类中定义，用于标记线程是否忙碌 |
| wr\_t \*workerrec = new wr\_t[global\_max\_num\_worker] | 在ServerImpl类中定义，用于储存gpworker的基本信息，帮助gpproxy寻找对应的gpworker。 |

gpproxy与libteecc之间数据通信格式protobuf如表7所示。

**表 7 protobuf数据结构**

|  |  |
| --- | --- |
| 数据结构 | 功能 |
| service gpp {  rpc TEECC\_InitializeContext (Inicont\_Request) returns (Inicont\_Reply) {}  rpc TEECC\_FinalizeContext (Fincont\_Request) returns (Fincont\_Reply) {}  rpc TEECC\_OpenSession (Opes\_Request) returns (Opes\_Reply) {}  rpc TEECC\_InvokeCommand (Invo\_Request) returns (Invo\_Reply) {}  rpc TEECC\_CloseSession (Close\_Request) returns (Close\_Reply) {}  rpc TEECC\_TA(TA\_Chunk) returns (TA\_Reply) {}  } | 定义rpc传输接口与传输的数据结构 |
| message Inicont\_Request {  uint64 name\_size = 1;  string name = 2;  string token = 3;  string taname = 4;  } | 定义TEECC\_InitializeContext中的Request数据 |
| message Inicont\_Reply {  uint32 teecresult = 1;  int32 context\_fd = 2;  uint64 context\_tapath\_outsize = 3;  bytes context\_tapath = 4;  uint64 context\_sessionlist\_next = 5;  uint64 context\_sessionlist\_prev = 6;  uint64 context\_shrdmemlist\_next = 7;  uint64 context\_shrdmemlist\_prev = 8;  uint64 context\_sharebuffer\_buffer = 9;  int64 context\_sharebuffer\_bufferbarrier = 10;  uint64 context\_addr = 11;  int32 flag = 12;  } | 定义TEECC\_InitializeContext中的Reply数据 |
| message Fincont\_Request {  int32 in\_context\_fd = 1;  uint64 in\_context\_tapath\_size= 2;  bytes in\_context\_tapath = 3;  uint64 in\_context\_sessionlist\_next = 4;  uint64 in\_context\_sessionlist\_prev = 5;  uint64 in\_context\_shrdmemlist\_next = 6;  uint64 in\_context\_shrdmemlist\_prev = 7;  uint64 in\_context\_sharebuffer\_buffer = 8;  int64 in\_context\_sharebuffer\_bufferbarrier = 9;  uint64 in\_context\_addr = 10;  string token = 11;  string taname = 12;  } | 定义TEECC\_TEECC\_FinalizeContext中的Request数据 |
| message Fincont\_Reply {  int32 context\_fd = 1;  uint64 context\_tapath\_outsize = 2;  bytes context\_tapath = 3;  uint64 context\_sessionlist\_next = 4;  uint64 context\_sessionlist\_prev = 5;  uint64 context\_shrdmemlist\_next = 6;  uint64 context\_shrdmemlist\_prev = 7;  uint64 context\_sharebuffer\_buffer = 8;  int64 context\_sharebuffer\_bufferbarrier = 9;  uint64 context\_addr = 10;  int32 flag = 11;  } | 定义TEECC\_TEECC\_FinalizeContext中的Reply数据 |
| message Opes\_Request {  int32 in\_context\_fd = 1;  uint64 in\_context\_tapath\_size= 2;  bytes in\_context\_tapath = 3;  uint64 in\_context\_sessionlist\_next = 4;  uint64 in\_context\_sessionlist\_prev = 5;  uint64 in\_context\_shrdmemlist\_next = 6;  uint64 in\_context\_shrdmemlist\_prev = 7;  uint64 in\_context\_sharebuffer\_buffer = 8;  int64 in\_context\_sharebuffer\_bufferbarrier = 9;  uint32 in\_destination\_timelow = 10;  uint32 in\_destination\_timemid = 11;  uint32 in\_destination\_timehiandver = 12;  uint64 in\_destination\_cad\_size = 13;  repeated uint32 in\_destination\_clockseqandnode = 14;  uint32 in\_connectionmethod = 15;  uint64 in\_connectiondata = 16;  uint32 in\_operation\_started = 17;  uint32 in\_operation\_paramtypes = 18;  uint64 in\_operation\_param1\_tmpref\_buffer = 19;  uint32 in\_operation\_param1\_tmpref\_size = 20;  uint64 in\_operation\_param1\_memref\_parent = 21;  uint32 in\_operation\_param1\_memref\_size = 22;  uint32 in\_operation\_param1\_memref\_offset = 23;  uint32 in\_operation\_param1\_value\_a = 24;  uint32 in\_operation\_param1\_value\_b = 25;  int32 in\_operation\_param1\_ionref\_ionsharefd = 26;  uint32 in\_operation\_param1\_ionref\_ionsize = 27;  uint64 in\_operation\_param2\_tmpref\_buffer = 28;  uint32 in\_operation\_param2\_tmpref\_size = 29;  uint64 in\_operation\_param2\_memref\_parent = 30;  uint32 in\_operation\_param2\_memref\_size = 31;  uint32 in\_operation\_param2\_memref\_offset = 32;  uint32 in\_operation\_param2\_value\_a = 33;  uint32 in\_operation\_param2\_value\_b = 34;  int32 in\_operation\_param2\_ionref\_ionsharefd = 35;  uint32 in\_operation\_param2\_ionref\_ionsize = 36;  uint64 in\_operation\_param3\_tmpref\_buffer = 37;  uint32 in\_operation\_param3\_tmpref\_size = 38;  uint64 in\_operation\_param3\_memref\_parent = 39;  uint32 in\_operation\_param3\_memref\_size = 40;  uint32 in\_operation\_param3\_memref\_offset = 41;  uint32 in\_operation\_param3\_value\_a = 42;  uint32 in\_operation\_param3\_value\_b = 43;  int32 in\_operation\_param3\_ionref\_ionsharefd = 44;  uint32 in\_operation\_param3\_ionref\_ionsize = 45;  uint64 in\_operation\_param4\_tmpref\_buffer = 46;  uint32 in\_operation\_param4\_tmpref\_size = 47;  uint64 in\_operation\_param4\_memref\_parent = 48;  uint32 in\_operation\_param4\_memref\_size = 49;  uint32 in\_operation\_param4\_memref\_offset = 50;  uint32 in\_operation\_param4\_value\_a = 51;  uint32 in\_operation\_param4\_value\_b = 52;  int32 in\_operation\_param4\_ionref\_ionsharefd = 53;  uint32 in\_operation\_param4\_ionref\_ionsize = 54;  uint64 in\_operation\_session = 55;  int32 in\_operation\_cancelflag = 56;  uint32 in\_returnorigin = 57;  uint64 in\_context\_addr = 58;  string token = 59;  string taname = 60;  } | 定义TEECC\_OpenSession中的Request数据 |
| message Opes\_Reply {  uint32 teecresult = 1;  int32 context\_fd = 2;  bytes context\_tapath = 3 ;  uint64 context\_tapath\_outsize = 4;  uint64 context\_sessionlist\_next = 5;  uint64 context\_sessionlist\_prev = 6;  uint64 context\_shrdmemlist\_next = 7;  uint64 context\_shrdmemlist\_prev = 8;  uint64 context\_sharebuffer\_buffer = 9;  int64 context\_sharebuffer\_bufferbarrier = 10;  uint32 session\_sessionid = 11;  uint32 session\_serviceid\_timelow = 12;  uint32 session\_serviceid\_timemid = 13;  uint32 session\_serviceid\_timehiandver = 14;  uint64 session\_serviceid\_clockseqandnode\_outsize = 15;  repeated uint32 session\_serviceid\_clockseqandnode = 16;  uint32 session\_opscnt = 17;  uint64 session\_head\_next = 18;  uint64 session\_head\_prev = 19;  uint64 session\_context = 20;  uint32 operation\_started = 21;  uint32 operation\_paramtypes = 22;  uint64 operation\_param1\_tmpref\_buffer = 23;  uint32 operation\_param1\_tmpref\_size = 24;  uint64 operation\_param1\_memref\_parent = 25;  uint32 operation\_param1\_memref\_size = 26;  uint32 operation\_param1\_memref\_offset = 27;  uint32 operation\_param1\_value\_a = 28;  uint32 operation\_param1\_value\_b = 29;  int32 operation\_param1\_ionref\_ionsharefd = 30;  uint32 operation\_param1\_ionref\_ionsize = 31;  uint64 operation\_param2\_tmpref\_buffer = 32;  uint32 operation\_param2\_tmpref\_size = 33;  uint64 operation\_param2\_memref\_parent = 34;  uint32 operation\_param2\_memref\_size = 35;  uint32 operation\_param2\_memref\_offset = 36;  uint32 operation\_param2\_value\_a = 37;  uint32 operation\_param2\_value\_b = 38;  int32 operation\_param2\_ionref\_ionsharefd = 39;  uint32 operation\_param2\_ionref\_ionsize = 40;  uint64 operation\_param3\_tmpref\_buffer = 41;  uint32 operation\_param3\_tmpref\_size = 42;  uint64 operation\_param3\_memref\_parent = 43;  uint32 operation\_param3\_memref\_size = 44;  uint32 operation\_param3\_memref\_offset = 45;  uint32 operation\_param3\_value\_a = 46;  uint32 operation\_param3\_value\_b = 47;  int32 operation\_param3\_ionref\_ionsharefd = 48;  uint32 operation\_param3\_ionref\_ionsize = 49;  uint64 operation\_param4\_tmpref\_buffer = 50;  uint32 operation\_param4\_tmpref\_size = 51;  uint64 operation\_param4\_memref\_parent = 52;  uint32 operation\_param4\_memref\_size = 53;  uint32 operation\_param4\_memref\_offset = 54;  uint32 operation\_param4\_value\_a = 55;  uint32 operation\_param4\_value\_b = 56;  int32 operation\_param4\_ionref\_ionsharefd = 57;  uint32 operation\_param4\_ionref\_ionsize = 58;  uint64 operation\_session = 59;  int32 operation\_cancelflag = 60;  uint32 returnorigin = 61;  int32 flag = 62;  } | 定义TEECC\_OpenSession中的Reply数据 |
| message Invo\_Request {  uint32 in\_session\_sessionid = 1;  uint32 in\_session\_serviceid\_timelow = 2;  uint32 in\_session\_serviceid\_timemid = 3;  uint32 in\_session\_serviceid\_timehiandver = 4;  uint64 in\_session\_serviceid\_cad\_size = 5;  repeated uint32 in\_session\_serviceid\_clockseqandnode = 6;  uint32 in\_session\_opscnt = 7;  uint64 in\_session\_head\_next = 8;  uint64 in\_session\_head\_prev = 9;  uint64 in\_session\_context = 10;  uint32 in\_commandid = 11;  uint32 in\_operation\_started = 12;  uint32 in\_operation\_paramtypes = 13;  uint64 in\_operation\_param1\_tmpref\_buffer = 14;  uint32 in\_operation\_param1\_tmpref\_size = 15;  uint64 in\_operation\_param1\_memref\_parent = 16;  uint32 in\_operation\_param1\_memref\_parent\_flag = 17;  uint32 in\_operation\_param1\_memref\_size = 18;  uint32 in\_operation\_param1\_memref\_offset = 19;  uint32 in\_operation\_param1\_value\_a = 20;  uint32 in\_operation\_param1\_value\_b = 21;  int32 in\_operation\_param1\_ionref\_ionsharefd = 22;  uint32 in\_operation\_param1\_ionref\_ionsize = 23;  uint64 in\_operation\_param2\_tmpref\_buffer = 24;  uint32 in\_operation\_param2\_tmpref\_size = 25;  uint64 in\_operation\_param2\_memref\_parent = 26;  uint32 in\_operation\_param2\_memref\_parent\_flag = 27;  uint32 in\_operation\_param2\_memref\_size = 28;  uint32 in\_operation\_param2\_memref\_offset = 29;  uint32 in\_operation\_param2\_value\_a = 30;  uint32 in\_operation\_param2\_value\_b = 31;  int32 in\_operation\_param2\_ionref\_ionsharefd = 32;  uint32 in\_operation\_param2\_ionref\_ionsize = 33;  uint64 in\_operation\_param3\_tmpref\_buffer = 34;  uint32 in\_operation\_param3\_tmpref\_size = 35;  uint64 in\_operation\_param3\_memref\_parent = 36;  uint32 in\_operation\_param3\_memref\_parent\_flag = 37;  uint32 in\_operation\_param3\_memref\_size = 38;  uint32 in\_operation\_param3\_memref\_offset = 39;  uint32 in\_operation\_param3\_value\_a = 40;  uint32 in\_operation\_param3\_value\_b = 41;  int32 in\_operation\_param3\_ionref\_ionsharefd = 42;  uint32 in\_operation\_param3\_ionref\_ionsize = 43;  uint64 in\_operation\_param4\_tmpref\_buffer = 44;  uint32 in\_operation\_param4\_tmpref\_size = 45;  uint64 in\_operation\_param4\_memref\_parent = 46;  uint32 in\_operation\_param4\_memref\_parent\_flag = 47;  uint32 in\_operation\_param4\_memref\_size = 48;  uint32 in\_operation\_param4\_memref\_offset = 49;  uint32 in\_operation\_param4\_value\_a = 50;  uint32 in\_operation\_param4\_value\_b = 51;  int32 in\_operation\_param4\_ionref\_ionsharefd = 52;  uint32 in\_operation\_param4\_ionref\_ionsize = 53;  uint64 in\_operation\_session = 54;  int32 in\_operation\_cancelflag = 55;  uint32 in\_returnorigin = 56;  uint64 in\_bufer1\_size = 57;  repeated uint32 in\_buffer1 = 58;  uint64 in\_bufer2\_size = 59;  repeated uint32 in\_buffer2 = 60;  uint64 in\_bufer3\_size = 61;  repeated uint32 in\_buffer3 = 62;  uint64 in\_bufer4\_size = 63;  repeated uint32 in\_buffer4 = 64;  string token = 65;  string taname = 66;  } | 定义TEECC\_InvokeCommand中的Request数据 |
| message Invo\_Reply {  uint32 teecresult = 1;  uint32 session\_sessionid = 2;  uint32 session\_serviceid\_timelow = 3;  uint32 session\_serviceid\_timemid = 4;  uint32 session\_serviceid\_timehiandver = 5;  uint64 session\_serviceid\_clockseqandnode\_outsize = 6;  repeated uint32 session\_serviceid\_clockseqandnode = 7;  uint32 session\_opscnt = 8;  uint64 session\_head\_next = 9;  uint64 session\_head\_prev = 10;  uint64 session\_context = 11;  uint32 operation\_started = 12;  uint32 operation\_paramtypes = 13;  uint64 operation\_param1\_tmpref\_buffer = 14;  uint32 operation\_param1\_tmpref\_size = 15;  uint64 operation\_param1\_memref\_parent = 16;  uint32 operation\_param1\_memref\_parent\_flag = 17;  uint32 operation\_param1\_memref\_size = 18;  uint32 operation\_param1\_memref\_offset = 19;  uint32 operation\_param1\_value\_a = 20;  uint32 operation\_param1\_value\_b = 21;  int32 operation\_param1\_ionref\_ionsharefd = 22;  uint32 operation\_param1\_ionref\_ionsize = 23;  uint64 operation\_param2\_tmpref\_buffer = 24;  uint32 operation\_param2\_tmpref\_size = 25;  uint64 operation\_param2\_memref\_parent = 26;  uint32 operation\_param2\_memref\_parent\_flag = 27;  uint32 operation\_param2\_memref\_size = 28;  uint32 operation\_param2\_memref\_offset = 29;  uint32 operation\_param2\_value\_a = 30;  uint32 operation\_param2\_value\_b = 31;  int32 operation\_param2\_ionref\_ionsharefd = 32;  uint32 operation\_param2\_ionref\_ionsize = 33;  uint64 operation\_param3\_tmpref\_buffer = 34;  uint32 operation\_param3\_tmpref\_size = 35;  uint64 operation\_param3\_memref\_parent = 36;  uint32 operation\_param3\_memref\_parent\_flag = 37;  uint32 operation\_param3\_memref\_size = 38;  uint32 operation\_param3\_memref\_offset = 39;  uint32 operation\_param3\_value\_a = 40;  uint32 operation\_param3\_value\_b = 41;  int32 operation\_param3\_ionref\_ionsharefd = 42;  uint32 operation\_param3\_ionref\_ionsize = 43;  uint64 operation\_param4\_tmpref\_buffer = 44;  uint32 operation\_param4\_tmpref\_size = 45;  uint64 operation\_param4\_memref\_parent = 46;  uint32 operation\_param4\_memref\_parent\_flag = 47;  uint32 operation\_param4\_memref\_size = 48;  uint32 operation\_param4\_memref\_offset = 49;  uint32 operation\_param4\_value\_a = 50;  uint32 operation\_param4\_value\_b = 51;  int32 operation\_param4\_ionref\_ionsharefd = 52;  uint32 operation\_param4\_ionref\_ionsize = 53;  uint64 operation\_session = 54;  int32 operation\_cancelflag = 55;  uint32 returnorigin = 56;  uint64 buffer1\_outsize = 57;  uint64 buffer2\_outsize = 58;  uint64 buffer3\_outsize = 59;  uint64 buffer4\_outsize = 60;  repeated uint32 buffer1 = 61;  repeated uint32 buffer2 = 62;  repeated uint32 buffer3 = 63;  repeated uint32 buffer4 = 64;  int32 flag = 65;  } | 定义TEECC\_InvokeCommand中的Reply数据 |
| message Close\_Request {  uint32 in\_session\_sessionid = 1;  uint32 in\_session\_serviceid\_timelow = 2;  uint32 in\_session\_serviceid\_timemid = 3;  uint32 in\_session\_serviceid\_timehiandver = 4;  uint64 in\_session\_serviceid\_cad\_size = 5;  repeated uint32 in\_session\_serviceid\_clockseqandnode = 6;  uint32 in\_session\_opscnt = 7;  uint64 in\_session\_head\_next = 8;  uint64 in\_session\_head\_prev = 9;  uint64 in\_session\_context = 10;  string token = 11;  string taname = 12;  } | 定义TEECC\_CloseSession中的Request数据 |
| message Close\_Reply {  uint32 session\_sessionid = 1;  uint32 session\_serviceid\_timelow = 2;  uint32 session\_serviceid\_timemid = 3;  uint32 session\_serviceid\_timehiandver = 4;  uint64 session\_serviceid\_cad\_outsize = 5;  repeated uint32 session\_serviceid\_clockseqandnode = 6;  uint32 session\_opscnt = 7;  uint64 session\_head\_next = 8;  uint64 session\_head\_prev = 9;  uint64 session\_context = 10;  int32 flag = 11;  } | 定义TEECC\_CloseSession中的Reply数据 |
| message TA\_Chunk{  string name = 1;  bytes buffer = 2;  string token = 3;  string taname = 4;  bytes sha256 = 5;  string subdir = 6;  } | 定义TEECC\_TA中的Request数据 |
| message TA\_Reply{  int32 code = 1;  int32 flag = 2;  } | 定义TEECC\_TA中的Reply数据 |

3.4 基于DBus的libdbuscgpw

gpproxy通过DBus向gpworker转发GP Client API调用。DBus是一种进程间通信机制。当gpproxy进程发送调用信息给gpworker时，信息先发送到DBus后台进程，DBus后台进程再将信息转发到gpworker进程。DBus后台进程充当着一个路由器的角色。gpproxy发送的一个DBus method call消息包含gpworker进程的DBus name、方法名字、方法参数、gpworker进程中的object path、可选的接口名字等。gpworker进程将应答一个method reply消息或者error消息。

libdbuscgpw库提供如表8 所示的与标准GP API对应的methodcall函数接口，供给gpproxy调用来与gpworker进行通信，对libteecc发送来的参数进行转发，返回gpworker调用后所得数据。

**表 8 libdbuscgpw模块接口**

|  |  |
| --- | --- |
| 接口 | 功能 |
| int32\_t  method\_call\_teec\_inicont(  const char \*workername,  const uint8\_t \*name, size\_t name\_size,  int32\_t in\_context\_fd,  const uint8\_t \*in\_context\_tapath, size\_t in\_context\_tapath\_size,  uint64\_t in\_context\_sessionlist\_next,  uint64\_t in\_context\_sessionlist\_prev,  uint64\_t in\_context\_shrdmemlist\_next,  uint64\_t in\_context\_shrdmemlist\_prev,  uint64\_t in\_context\_sharebuffer\_buffer,  int64\_t in\_context\_sharebuffer\_bufferbarrier,  uint32\_t \*teecresult,  int32\_t \*context\_fd,  uint8\_t \*context\_tapath, size\_t context\_tapath\_insize,  uint64\_t \*context\_sessionlist\_next,  uint64\_t \*context\_sessionlist\_prev,  uint64\_t \*context\_shrdmemlist\_next,  uint64\_t \*context\_shrdmemlist\_prev,  uint64\_t \*context\_sharebuffer\_buffer,  int64\_t \*context\_sharebuffer\_bufferbarrier,  uint64\_t \*context\_addr,  uint32\_t \*context\_tapath\_outsize  ) | 向指定gpworker发送methodcall，获得返回值 |
| int32\_t  method\_call\_teec\_fincont(  const char \*workername,  int32\_t in\_context\_fd,  const uint8\_t \*in\_context\_tapath, size\_t in\_context\_tapath\_size,  uint64\_t in\_context\_sessionlist\_next,  uint64\_t in\_context\_sessionlist\_prev,  uint64\_t in\_context\_shrdmemlist\_next,  uint64\_t in\_context\_shrdmemlist\_prev,  uint64\_t in\_context\_sharebuffer\_buffer,  int64\_t in\_context\_sharebuffer\_bufferbarrier,  uint64\_t in\_context\_addr,  int32\_t \*context\_fd,  uint8\_t \*context\_tapath, size\_t context\_tapath\_insize,  uint64\_t \*context\_sessionlist\_next,  uint64\_t \*context\_sessionlist\_prev,  uint64\_t \*context\_shrdmemlist\_next,  uint64\_t \*context\_shrdmemlist\_prev,  uint64\_t \*context\_sharebuffer\_buffer,  int64\_t \*context\_sharebuffer\_bufferbarrier,  uint32\_t \*context\_tapath\_outsize  ) | 向指定gpworker发送methodcall，获得返回值 |
| int32\_t  method\_call\_teec\_opensession(  const char \*workername,  int32\_t in\_context\_fd,  const uint8\_t \*in\_context\_tapath,  size\_t in\_context\_tapath\_size,  uint64\_t in\_context\_sessionlist\_next,  uint64\_t in\_context\_sessionlist\_prev,  uint64\_t in\_context\_shrdmemlist\_next,  uint64\_t in\_context\_shrdmemlist\_prev,  uint64\_t in\_context\_sharebuffer\_buffer,  int64\_t in\_context\_sharebuffer\_bufferbarrier,  uint32\_t in\_destination\_timelow,  uint32\_t in\_destination\_timemid,  uint32\_t in\_destination\_timehiandver,  uint32\_t \*in\_destination\_clockseqandnode,  int32\_t in\_destination\_clockseqandnode\_size,  uint32\_t in\_connectionmethod,  uint64\_t in\_connectiondata,  uint32\_t in\_operation\_started,  uint32\_t in\_operation\_paramtypes,  uint64\_t in\_operation\_param1\_tmpref\_buffer,  uint32\_t in\_operation\_param1\_tmpref\_size,  uint64\_t in\_operation\_param1\_memref\_parent,  uint32\_t in\_operation\_param1\_memref\_size,  uint32\_t in\_operation\_param1\_memref\_offset,  uint32\_t in\_operation\_param1\_value\_a,  uint32\_t in\_operation\_param1\_value\_b,  int32\_t in\_operation\_param1\_ionref\_ionsharefd,  uint32\_t in\_operation\_param1\_ionref\_ionsize,  uint64\_t in\_operation\_param2\_tmpref\_buffer,  uint32\_t in\_operation\_param2\_tmpref\_size,  uint64\_t in\_operation\_param2\_memref\_parent,  uint32\_t in\_operation\_param2\_memref\_size,  uint32\_t in\_operation\_param2\_memref\_offset,  uint32\_t in\_operation\_param2\_value\_a,  uint32\_t in\_operation\_param2\_value\_b,  int32\_t in\_operation\_param2\_ionref\_ionsharefd,  uint32\_t in\_operation\_param2\_ionref\_ionsize,  uint64\_t in\_operation\_param3\_tmpref\_buffer,  uint32\_t in\_operation\_param3\_tmpref\_size,  uint64\_t in\_operation\_param3\_memref\_parent,  uint32\_t in\_operation\_param3\_memref\_size,  uint32\_t in\_operation\_param3\_memref\_offset,  uint32\_t in\_operation\_param3\_value\_a,  uint32\_t in\_operation\_param3\_value\_b,  int32\_t in\_operation\_param3\_ionref\_ionsharefd,  uint32\_t in\_operation\_param3\_ionref\_ionsize,  uint64\_t in\_operation\_param4\_tmpref\_buffer,  uint32\_t in\_operation\_param4\_tmpref\_size,  uint64\_t in\_operation\_param4\_memref\_parent,  uint32\_t in\_operation\_param4\_memref\_size,  uint32\_t in\_operation\_param4\_memref\_offset,  uint32\_t in\_operation\_param4\_value\_a,  uint32\_t in\_operation\_param4\_value\_b,  int32\_t in\_operation\_param4\_ionref\_ionsharefd,  uint32\_t in\_operation\_param4\_ionref\_ionsize,  uint64\_t in\_operation\_session,  int32\_t in\_operation\_cancelflag,  uint32\_t in\_returnorigin,  uint64\_t in\_context\_addr,  uint32\_t \*teecresult,  int32\_t \*context\_fd,  uint8\_t \*context\_tapath,  size\_t context\_tapath\_size,  uint32\_t \*context\_tapath\_outsize,  uint64\_t \*context\_sessionlist\_next,  uint64\_t \*context\_sessionlist\_prev,  uint64\_t \*context\_shrdmemlist\_next,  uint64\_t \*context\_shrdmemlist\_prev,  uint64\_t \*context\_sharebuffer\_buffer,  int64\_t \*context\_sharebuffer\_bufferbarrier,  uint32\_t \*session\_seesionid,  uint32\_t \*session\_serviceid\_timelow,  uint32\_t \*session\_serviceid\_timemid,  uint32\_t \*session\_serviceid\_timehiandver,  uint32\_t \*session\_serviceid\_clockseqandnode,  int32\_t session\_serviceid\_clockseqandnode\_size,  uint32\_t \*session\_serviceid\_clockseqandnode\_outsize,  uint32\_t \*session\_opscnt,  uint64\_t \*session\_head\_next,  uint64\_t \*session\_head\_prev,  uint64\_t \*session\_context,  uint32\_t \*operation\_started,  uint32\_t \*operation\_paramtypes,  uint64\_t \*operation\_param1\_tmpref\_buffer,  uint32\_t \*operation\_param1\_tmpref\_size,  uint64\_t \*operation\_param1\_memref\_parent,  uint32\_t \*operation\_param1\_memref\_size,  uint32\_t \*operation\_param1\_memref\_offset,  uint32\_t \*operation\_param1\_value\_a,  uint32\_t \*operation\_param1\_value\_b,  int32\_t \*operation\_param1\_ionref\_ionsharefd,  uint32\_t \*operation\_param1\_ionref\_ionsize,  uint64\_t \*operation\_param2\_tmpref\_buffer,  uint32\_t \*operation\_param2\_tmpref\_size,  uint64\_t \*operation\_param2\_memref\_parent,  uint32\_t \*operation\_param2\_memref\_size,  uint32\_t \*operation\_param2\_memref\_offset,  uint32\_t \*operation\_param2\_value\_a,  uint32\_t \*operation\_param2\_value\_b,  int32\_t \*operation\_param2\_ionref\_ionsharefd,  uint32\_t \*operation\_param2\_ionref\_ionsize,  uint64\_t \*operation\_param3\_tmpref\_buffer,  uint32\_t \*operation\_param3\_tmpref\_size,  uint64\_t \*operation\_param3\_memref\_parent,  uint32\_t \*operation\_param3\_memref\_size,  uint32\_t \*operation\_param3\_memref\_offset,  uint32\_t \*operation\_param3\_value\_a,  uint32\_t \*operation\_param3\_value\_b,  int32\_t \*operation\_param3\_ionref\_ionsharefd,  uint32\_t \*operation\_param3\_ionref\_ionsize,  uint64\_t \*operation\_param4\_tmpref\_buffer,  uint32\_t \*operation\_param4\_tmpref\_size,  uint64\_t \*operation\_param4\_memref\_parent,  uint32\_t \*operation\_param4\_memref\_size,  uint32\_t \*operation\_param4\_memref\_offset,  uint32\_t \*operation\_param4\_value\_a,  uint32\_t \*operation\_param4\_value\_b,  int32\_t \*operation\_param4\_ionref\_ionsharefd,  uint32\_t \*operation\_param4\_ionref\_ionsize,  uint64\_t \*operation\_session,  int32\_t \*operation\_cancelflag,  uint32\_t \*returnorigin  ) | 向指定gpworker发送methodcall，获得返回值 |
| int32\_t  method\_call\_teec\_closesession(  const char \*workername,  uint32\_t in\_session\_seesionid,  uint32\_t in\_session\_serviceid\_timelow,  uint32\_t in\_session\_serviceid\_timemid,  uint32\_t in\_session\_serviceid\_timehiandver,  uint32\_t \*in\_session\_serviceid\_clockseqandnode,  int32\_t in\_session\_serviceid\_clockseqandnode\_size,  uint32\_t in\_session\_opscnt,  uint64\_t in\_session\_head\_next,  uint64\_t in\_session\_head\_prev,  uint64\_t in\_session\_context,  uint32\_t \*session\_seesionid,  uint32\_t \*session\_serviceid\_timelow,  uint32\_t \*session\_serviceid\_timemid,  uint32\_t \*session\_serviceid\_timehiandver,  uint32\_t \*session\_serviceid\_clockseqandnode,  int32\_t session\_serviceid\_clockseqandnode\_size,  uint32\_t \*session\_serviceid\_clockseqandnode\_outsize,  uint32\_t \*session\_opscnt,  uint64\_t \*session\_head\_next,  uint64\_t \*session\_head\_prev,  uint64\_t \*session\_context  ) | 向指定gpworker发送methodcall，获得返回值 |
| int32\_t  method\_call\_teec\_invokecommand(  const char \*workername,  uint32\_t in\_session\_seesionid,  uint32\_t in\_session\_serviceid\_timelow,  uint32\_t in\_session\_serviceid\_timemid,  uint32\_t in\_session\_serviceid\_timehiandver,  uint32\_t \*in\_session\_serviceid\_clockseqandnode,  uint32\_t in\_session\_serviceid\_clockseqandnode\_size,  uint32\_t in\_session\_opscnt,  uint64\_t in\_session\_head\_next,  uint64\_t in\_session\_head\_prev,  uint64\_t in\_session\_context,  uint32\_t commandid,  uint32\_t in\_operation\_started,  uint32\_t in\_operation\_paramtypes,  uint64\_t in\_operation\_param1\_tmpref\_buffer,  uint32\_t in\_operation\_param1\_tmpref\_size,  uint64\_t in\_operation\_param1\_memref\_parent,  uint32\_t in\_operation\_param1\_memref\_parent\_flag,  uint32\_t in\_operation\_param1\_memref\_size,  uint32\_t in\_operation\_param1\_memref\_offset,  uint32\_t in\_operation\_param1\_value\_a,  uint32\_t in\_operation\_param1\_value\_b,  int32\_t in\_operation\_param1\_ionref\_ionsharefd,  uint32\_t in\_operation\_param1\_ionref\_ionsize,  uint64\_t in\_operation\_param2\_tmpref\_buffer,  uint32\_t in\_operation\_param2\_tmpref\_size,  uint64\_t in\_operation\_param2\_memref\_parent,  uint32\_t in\_operation\_param2\_memref\_parent\_flag,  uint32\_t in\_operation\_param2\_memref\_size,  uint32\_t in\_operation\_param2\_memref\_offset,  uint32\_t in\_operation\_param2\_value\_a,  uint32\_t in\_operation\_param2\_value\_b,  int32\_t in\_operation\_param2\_ionref\_ionsharefd,  uint32\_t in\_operation\_param2\_ionref\_ionsize,  uint64\_t in\_operation\_param3\_tmpref\_buffer,  uint32\_t in\_operation\_param3\_tmpref\_size,  uint64\_t in\_operation\_param3\_memref\_parent,  uint32\_t in\_operation\_param3\_memref\_parent\_flag,  uint32\_t in\_operation\_param3\_memref\_size,  uint32\_t in\_operation\_param3\_memref\_offset,  uint32\_t in\_operation\_param3\_value\_a,  uint32\_t in\_operation\_param3\_value\_b,  int32\_t in\_operation\_param3\_ionref\_ionsharefd,  uint32\_t in\_operation\_param3\_ionref\_ionsize,  uint64\_t in\_operation\_param4\_tmpref\_buffer,  uint32\_t in\_operation\_param4\_tmpref\_size,  uint64\_t in\_operation\_param4\_memref\_parent,  uint32\_t in\_operation\_param4\_memref\_parent\_flag,  uint32\_t in\_operation\_param4\_memref\_size,  uint32\_t in\_operation\_param4\_memref\_offset,  uint32\_t in\_operation\_param4\_value\_a,  uint32\_t in\_operation\_param4\_value\_b,  int32\_t in\_operation\_param4\_ionref\_ionsharefd,  uint32\_t in\_operation\_param4\_ionref\_ionsize,  uint64\_t in\_operation\_session,  int32\_t in\_operation\_cancelflag,  uint32\_t in\_returnorigin,  uint32\_t \*in\_buffer1,  uint32\_t in\_buffer1\_size,  uint32\_t \*in\_buffer2,  uint32\_t in\_buffer2\_size,  uint32\_t \*in\_buffer3,  uint32\_t in\_buffer3\_size,  uint32\_t \*in\_buffer4,  uint32\_t in\_buffer4\_size,  uint32\_t \*teecresult,  uint32\_t \*session\_seesionid,  uint32\_t \*session\_serviceid\_timelow,  uint32\_t \*session\_serviceid\_timemid,  uint32\_t \*session\_serviceid\_timehiandver,  uint32\_t \*session\_serviceid\_clockseqandnode,  int32\_t session\_serviceid\_clockseqandnode\_size,  uint32\_t \*session\_serviceid\_clockseqandnode\_outsize,  uint32\_t \*session\_opscnt,  uint64\_t \*session\_head\_next,  uint64\_t \*session\_head\_prev,  uint64\_t \*session\_context,  uint32\_t \*operation\_started,  uint32\_t \*operation\_paramtypes,  uint64\_t \*operation\_param1\_tmpref\_buffer,  uint32\_t \*operation\_param1\_tmpref\_size,  uint64\_t \*operation\_param1\_memref\_parent,  uint32\_t \*operation\_param1\_memref\_parent\_flag,  uint32\_t \*operation\_param1\_memref\_size,  uint32\_t \*operation\_param1\_memref\_offset,  uint32\_t \*operation\_param1\_value\_a,  uint32\_t \*operation\_param1\_value\_b,  int32\_t \*operation\_param1\_ionref\_ionsharefd,  uint32\_t \*operation\_param1\_ionref\_ionsize,  uint64\_t \*operation\_param2\_tmpref\_buffer,  uint32\_t \*operation\_param2\_tmpref\_size,  uint64\_t \*operation\_param2\_memref\_parent,  uint32\_t \*operation\_param2\_memref\_parent\_flag,  uint32\_t \*operation\_param2\_memref\_size,  uint32\_t \*operation\_param2\_memref\_offset,  uint32\_t \*operation\_param2\_value\_a,  uint32\_t \*operation\_param2\_value\_b,  int32\_t \*operation\_param2\_ionref\_ionsharefd,  uint32\_t \*operation\_param2\_ionref\_ionsize,  uint64\_t \*operation\_param3\_tmpref\_buffer,  uint32\_t \*operation\_param3\_tmpref\_size,  uint64\_t \*operation\_param3\_memref\_parent,  uint32\_t \*operation\_param3\_memref\_parent\_flag,  uint32\_t \*operation\_param3\_memref\_size,  uint32\_t \*operation\_param3\_memref\_offset,  uint32\_t \*operation\_param3\_value\_a,  uint32\_t \*operation\_param3\_value\_b,  int32\_t \*operation\_param3\_ionref\_ionsharefd,  uint32\_t \*operation\_param3\_ionref\_ionsize,  uint64\_t \*operation\_param4\_tmpref\_buffer,  uint32\_t \*operation\_param4\_tmpref\_size,  uint64\_t \*operation\_param4\_memref\_parent,  uint32\_t \*operation\_param4\_memref\_parent\_flag,  uint32\_t \*operation\_param4\_memref\_size,  uint32\_t \*operation\_param4\_memref\_offset,  uint32\_t \*operation\_param4\_value\_a,  uint32\_t \*operation\_param4\_value\_b,  int32\_t \*operation\_param4\_ionref\_ionsharefd,  uint32\_t \*operation\_param4\_ionref\_ionsize,  uint64\_t \*operation\_session,  int32\_t \*operation\_cancelflag,  uint32\_t \*returnorigin,  uint32\_t \*buffer1,  uint32\_t buffer1\_size,  uint32\_t \*buffer1\_outsize,  uint32\_t \*buffer2,  uint32\_t buffer2\_size,  uint32\_t \*buffer2\_outsize,  uint32\_t \*buffer3,  uint32\_t buffer3\_size,  uint32\_t \*buffer3\_outsize,  uint32\_t \*buffer4,  uint32\_t buffer4\_size,  uint32\_t \*buffer4\_outsize  ) | 向指定gpworker发送methodcall，获得返回值 |

3.5 基于DBus的gpworker

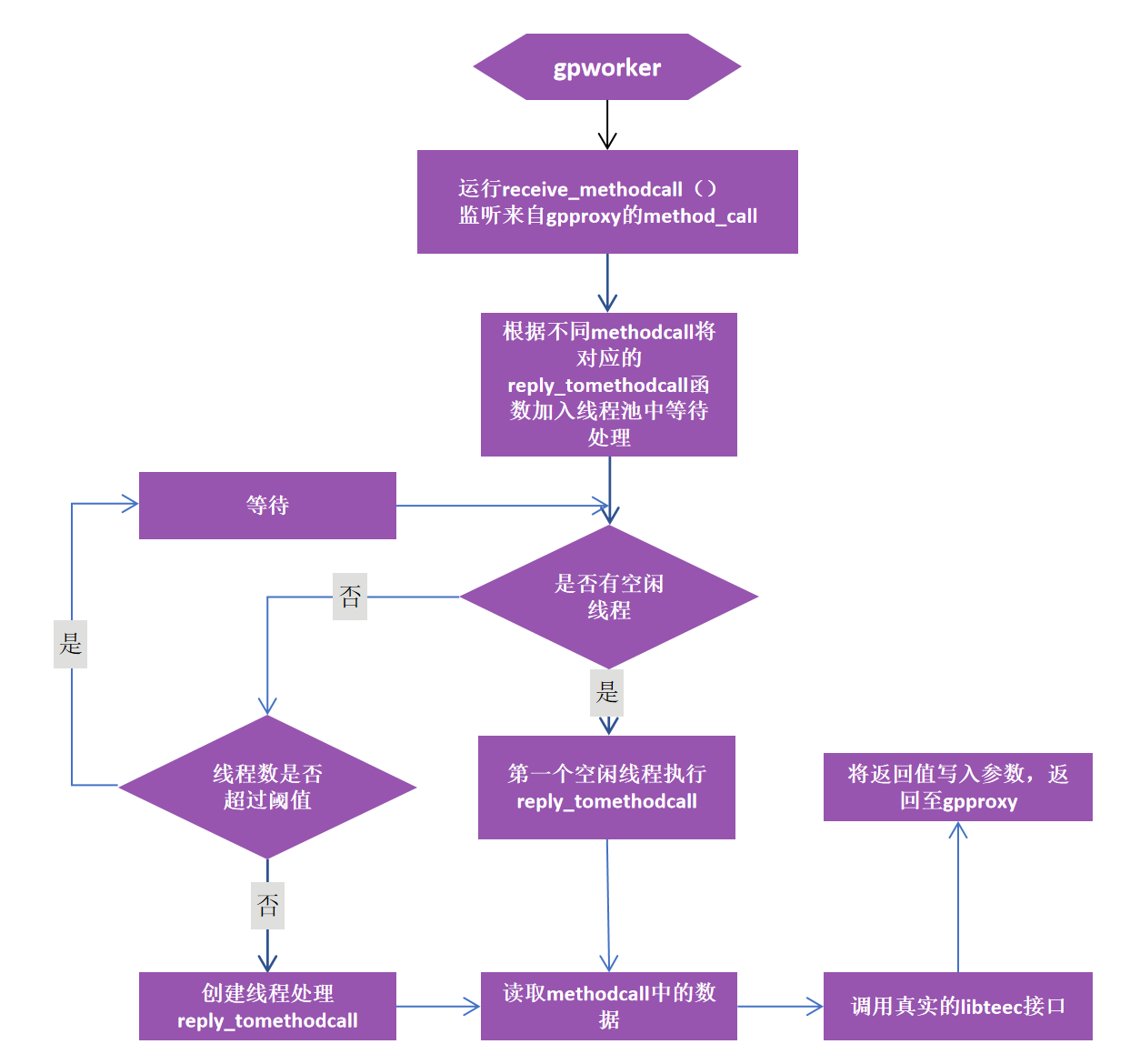
gpworker启动时，将线程池数据pool、互斥变量mutex\_tcl、条件变量mutex\_tsl、context链表tcl、session链表tsl作为参数传递至receive\_methodcall函数。在receive\_methodcall函数里，先初始化线程池数据，然后与DBus后台进程建立连接，并注册一个DBus name，然后开始接收来自gpproxy的methodcall。接收到methodcall后，根据调用的函数名将其对应的处理函数reply\_method\_call\_teec\_xxx加入到线程池任务处理队列中。线程池里如果没有空闲线程，且线程数量少于阈值，则新创建一个线程执行相应处理函数reply\_method\_call\_teec\_xxx；若有空闲线程，则让第一个空闲线程执行相应处理函数reply\_method\_call\_teec\_xxx；若没有空闲线程，且线程数量已达阈值，则需等待。

gpworker中各GP Client API的处理函数reply\_method\_call\_teec\_xxx会将参数传递给本地Kunpeng服务器上的libteec里相应接口，然后通过驱动调用TrustZone里的TA功能，将返回数据通过DBus返回给gpproxy，完成进程间函数调用。各线程处理函数reply\_method\_call\_teec\_xxx都会对context链表tcl、session链表tsl进行读、写操作，以记录上下文和所包含的会话信息，需要进行互斥操作。

gpproxy通过DBus调用一个method call时，先使用dbus\_bus\_get、 dbus\_bus\_request\_name与DBus后台进程进行注册连接，再使用dbus\_message\_new\_method\_call构造调用信息，然后使用dbus\_message\_iter\_init\_append初始化调用参数变量，再通过dbus\_message\_iter\_append\_basic向参数变量中中添加具体数据。接着使用dbus\_pending\_call\_steal\_reply接收返回信息将其写入对应的参数中。结束时，调用dbus\_connection\_close函数关闭连接，释放资源。

3.6 gpworker模块内运行时序

gpworker模块的运行流程如图9所示，启动gpworker时可同时创建多个具有相同功能的gpworker进程 ，收到gpproxy的methodcall后根据调用的函数名将其对应的处理函数reply\_method\_call\_teec\_xxx加入到线程池任务处理队列中。之后将参数传递给libteec中的真实TA接口，调用硬件获得返回值，再通过DBus将数据返回给gpproxy。



**图 9 gpworker模块运行流程图**

**3.7 gpworker模块内接口与数据结构**

（1）gpworker模块内接口

gpworker模块内部接口如表9 所示，接口功能均为调用真实libteec与硬件进行交互，将返回数据通过DBus返回给gpproxy。

**表 9 gpworker 模块内部接口**

|  |  |
| --- | --- |
| 接口 | 功能 |
| void  receive\_methodcall(  threadpool\_t \*pool,  pthread\_mutex\_t \*mutex\_tcl,  pthread\_mutex\_t \*mutex\_tsl,  tcl\_t \*tcl,  tsl\_t \*tsl,  char \*workername  ) | 开启methodcall监听程序 |
| void \*  reply\_to\_method\_call\_teec\_inicont(  void \*thdfargs  ) | 对gpproxy发送来的methodcall进行处理并返回对应数据 |
| void \*  reply\_to\_method\_call\_teec\_fincont(  void \*thdfargs  ) | 对gpproxy发送来的methodcall进行处理并返回对应数据 |
| void \*  reply\_to\_method\_call\_teec\_opensession(  void \*thdfargs  ) | 对gpproxy发送来的methodcall进行处理并返回对应数据 |
| void \*  reply\_to\_method\_call\_teec\_closesession(  void \*thdfargs  ) | 对gpproxy发送来的methodcall进行处理并返回对应数据 |
| void \*  reply\_to\_method\_call\_teec\_invokecommand(  void \*thdfargs  ) | 对gpproxy发送来的methodcall进行处理并返回对应数据 |

（2）gpworker模块内数据结构

**表 10 gpworker 模块内部数据结构**

|  |  |
| --- | --- |
| 接口 | 功能 |
| typedef struct threadpool  {  condition\_t ready;  task\_t \*first;  task\_t \*last;  int counter;  int idle;  int max\_threads;  int quit;  } threadpool\_t | 记录线程池中基本信息 |
| typedef struct teec\_session\_list  {  int count;  tsn\_t \*first;  tsn\_t \*last;  } tsl\_t | 记录session的排队信息 |
| typedef struct teec\_context\_list  {  int count;  tcn\_t \*first;  tcn\_t \*last;  } tcl\_t | 记录context的排队信息 |

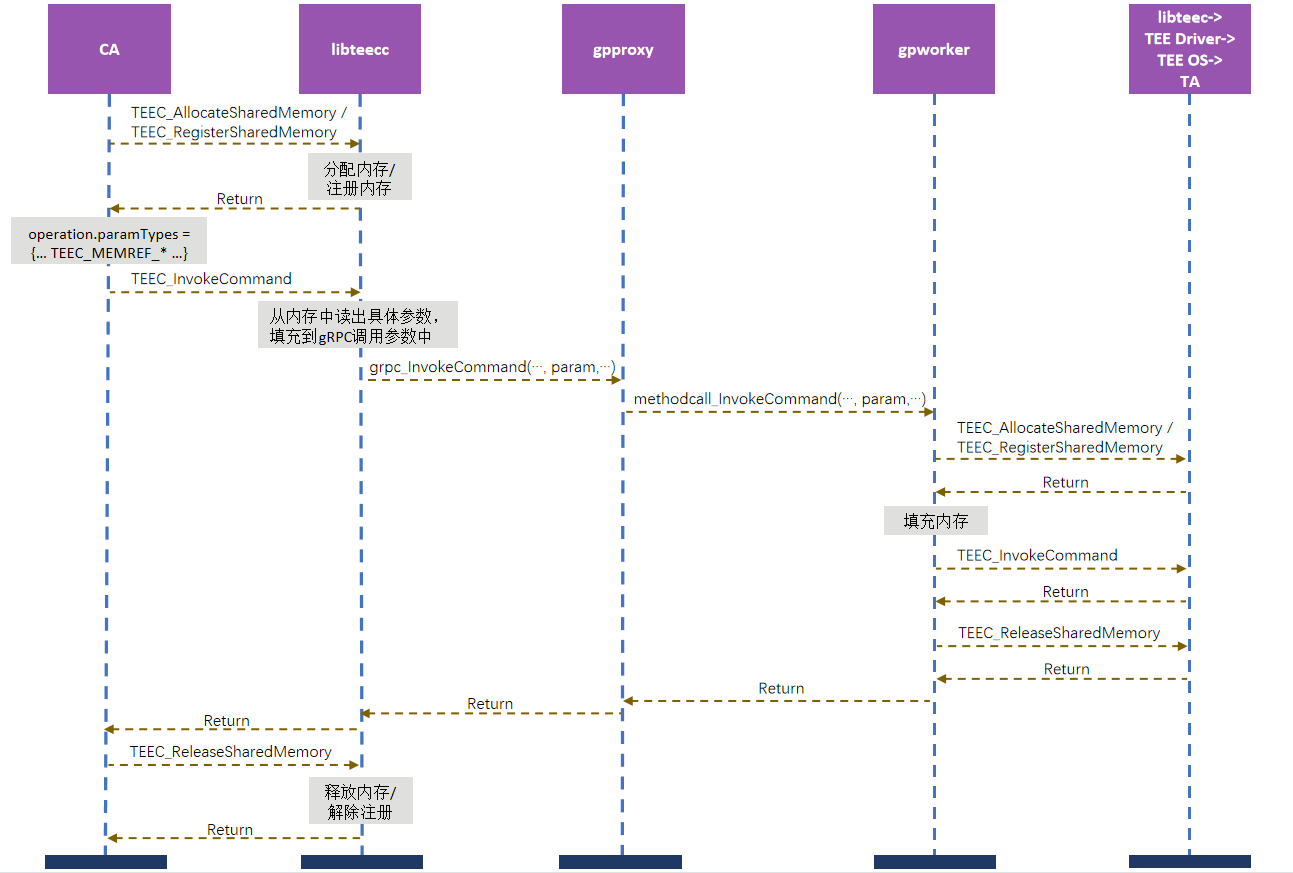
4 GP Client API内存共享参数传递的透明实现

在初始TrustZone应用场景里，用户应用CA都在处理器本地宿主机上的REE侧。GP Client API接口由基于C语言的链接库提供，接口参数使用了大量的内存指针。当REE侧的CA通过GP Client API接口向TEE侧的TA发起功能调用时，若参数值大于64比特，通常采用共享内存的方式来传递。

在Trustzone 资源池化场景里，用户应用CA都在异构平台虚拟机/容器里，无法和Kunpeng服务器里TEE侧的TA直接通过内存共享来传递参数。为了让用户开发CA更便捷，本项目拟开发的libteecc与原始libteec完全兼容，让在异构平台虚拟机/容器里的CA也可透明地使用共享内存的方式向Kunpeng服务器里TEE侧的TA传递参数。

初始TrustZone里，本地REE侧的CA与TEE侧TA基于内存共享进行参数传递的机制描述如下。CA调用初始化接口，连接到TEE，获得上下文context。然后CA在已创建的context连接上创建会话，指定TA，获得session。若CA计划使用共享内存来传递参数，先使用TEEC\_AllocateSharedMemory接口申请临时内存或者使用TEEC\_RegisterSharedMemory注册已分配好的内存，申请或注册需要在已有的Context下进行，可以供指定的Context里多个会话共享。调用申请或者注册内存接口时需要在TEEC\_SharedMemory结构中填入大小以及输入、输出类型。CA通过TEEC\_InvokeCommand接口向TA发起功能调用，参数传递方式由operation.paramTypes确定，参数信息在operation.params[0-3]里；若是通过共享内存传递参数，需要在operation.params[i].memref.parent、operation.params[i].memref.offset、operation.params[i].memref.size里填写相应的信息，真正的参数内容则在TEEC\_SharedMemory或TEEC\_TempMemoryReference结构对应的内存数据里。TA收到TEEC\_InvokeCommand接口调用后，根据operation.paramTypes和operation.params[0-3]，便可从相应内存里读取传入的参数。功能调用完毕后，CA调用TEEC\_ReleaseSharedMemory接口释放内存或者取消内存在Context里的注册。

在Trustzone 资源池化场景里，用户应用CA都在异构平台虚拟机/容器里，和TA所在Kunpeng服务器的内存空间不同。Trustzone 资源池化对GP Client API内存共享参数传递进行透明实现的机制如图10所示。CA调用TEEC\_AllocateSharedMemory / TEEC\_RegisterSharedMemory接口，libteecc收到后在异构平台上分配/注册内存，就直接返回给CA；CA用内存共享传递参数的方式调用TEEC\_InvokeCommand接口，libteecc收到后从内存中读出具体参数，将其填充到gRPC调用参数中，向gpproxy发起带有具体参数的调用grpc\_InvokeCommand；gpproxy收到后向gpworker转发DBus方法调用methodcall\_InvokeCommand；CTZWoker收到后，调用本地Kunpeng服务器上的libteec库接口TEEC\_AllocateSharedMemory / TEEC\_RegisterSharedMemory，然后将methodcall\_InvokeCommand中的参数填充到内存中，调用TEEC\_InvokeCommand接口使用TA的功能，完毕后将返回在共享内存里的响应数据填充到methodcall\_InvokeCommand的返回参数中，接着调用TEEC\_ReleaseSharedMemory接口释放/注销内存，再将methodcall\_InvokeCommand的结果返回给gpproxy；gpproxy收到后，将grpc\_InvokeCommand的结果返回给libteecc；libteecc将TEEC\_InvokeCommand的响应返回给CA；CA从libteecc分配/注册的内存里读取具体的返回参数，最后调用TEEC\_ReleaseSharedMemory接口让libteecc释放/注销在异构平台上的内存，完成共享参数传递的透明实现。



**图 10 Trustzone 资源池化对GP Client API内存共享参数传递的透明实现**

5 基于TLS的安全传输与基于JWT的鉴权

在libteecc与gpproxy之间的RPC调用通信过程中，为了保证数据传输的安全性，使用TLS对其传输通道进行安全保护。libteecc中定义的TEEC\_InitializeContext接口为用户CA进行RPC调用的第一个接口。在该函数中，根据配置文件中GRPC\_TLS的不同有三种TLS通道创建方式，分为无TLS通道、仅服务端认证通道、双向认证通道。

若无需TLS则直接使用grpc::CreateChannel(target\_str, grpc::InsecureChannelCredentials())创建Channel。

若为仅服务端认证则将服务端公钥证书通过get\_file\_contents函数写入对应的参数中，接着将该个参数传入一个grpc::SslCredentialsOptions类型的变量ssl\_opts中，再通过grpc::SslCredentials(ssl\_opts)将ssl\_opts赋值给creds，然后使用CreateChannel(targer\_address, creds)即可创建一个TLS Channel。

若为双向认证则将libteecc端私钥、公钥证书以及gpproxy端公钥证书通过get\_file\_contents函数写入对应的三个参数中，接着将三个参数传入一个grpc::SslCredentialsOptions类型的变量ssl\_opts中，再通过grpc::SslCredentials(ssl\_opts)将ssl\_opts赋值给creds，然后使用CreateChannel(targer\_address, creds)即可创建一个TLS Channel。gpproxy使用builder.AddListeningPort(server\_address, creds)启动TLS安全通道监听服务。

Trustzone 资源池化使用JWT来对异构平台虚拟机/容器里的用户CA使用Kunpeng服务器上的TrustZone功能进行访问控制。Trustzone 资源池化只提供相应的接口支持，用户自主提供JWT认证系统。用户CA需先在JWT认证系统里进行注册，获得JWT后，调用接口TEEC\_Setjwt将JWT传递给libteecc；用户CA再后续调用libteecc里的TEEC\_InitializeContext、TEEC\_FinalizeContext、TEEC\_OpenSession、TEEC\_CloseSession、TEEC\_InvokeCommand、TEEC\_DeployTa接口时，libteecc将JWT作为附加参数加入到GreeterClient类greeter里相应的方法grpc\_teec\_xxx，进行RPC调用。gpproxy端收到调用后，通过函数dbusmethodcall\_validate\_jwt来判断收到的用户CA JWT是否正确，从而确定是否允许用户CA使用Kunpeng服务器上相应的GP Cient接口。函数dbusmethodcall\_validate\_jwt通过调用用户自主提供的JWT认证系统链接库中的JWT认证函数获取JWT认证结果。