TSMaster RPC 编程指导 V0.1

修改时间	记录	修改人员	备注
2024.06.11	初始创建	Seven	

目录

TSMaster RPC 编程指导 V0.1 1
1. 什么情况下需要此文档？ 3
2.RPC 说明 3
2.1 RPC 的基本概念 3
5、执行方法并生成响应：本地方法执行完毕后，生成响应结果。 3
3.TSMaster RPC 应用 4
4.TSMaster RPC 使用说明 4
4.1 激活 server 5
4.2 激活 client 5
4.3 修改 server 端数据 5
4.3.1 启动停止 server 工程 5
4.3.2 读写系统变量 5
4.3.3 读写 CAN 信号 6
4.3.4 读写 LIN 信号 6
4.3.5 读写 FlexRay 信号 6
4.3.6 rpc 使用 TSMaster 系统函数 7
4.3.7 rpc 使用小程序库函数 7
5.TSMaster RPC 函数说明 8
1. rpc_tsmaster_create_client 8
2. rpc_tsmaster_activate_client 8
3. rpc_tsmaster_is_simulation_running 9
4. rpc_tsmaster_cmd_set_mode_realtime 9
5. rpc_tsmaster_cmd_set_mode_sim 9
6. rpc_tsmaster_cmd_start_simulation 10
7. rpc_tsmaster_cmd_set_can_signal 10
8. rpc_tsmaster_cmd_get_can_signal 10
9. rpc_tsmaster_cmd_set_lin_signal 11
10. rpc_tsmaster_cmd_get_lin_signal 11
11. rpc_tsmaster_cmd_set_flexray_signal 11
12. rpc_tsmaster_cmd_get_flexray_signal 12
13. rpc_tsmaster_cmd_write_system_var 12
14. rpc_tsmaster_cmd_read_system_var 13
15. rpc_tsmaster_cmd_write_signal 13
16. rpc_tsmaster_cmd_read_signal 14
17. rpc_tsmaster_delete_client 14
18. rpc_tsmaster_cmd_stop_simulation 14
19. rpc_tsmaster_call_system_api 15
20. rpc_tsmaster_call_library_api 15

1. 什么情况下需要此文档?

用户基于 TSMaster 软件开发好了对应的 TSMaster 工程，但是无法实现自动化控制 TSMaster 工程时，可以查阅本文档。

本文档适用程控模式:TSMaster1 控制 TSMaster2,或者其他进程控制 TSMaster 进程(使用 TSMaster.dll）适用于语言:C++ Python C#等语言。

2. RPC 说明

远程过程调用（RPC, Remote Procedure Call）是一种网络通信协议，使得程序可以调用另一台计算机上的程序或服务，就像调用本地的程序一样。RPC 的主要目的是简化分布式计算，使得开发者无需关注底层的网络通信细节。远程过程调用（RPC, Remote Procedure Call）是一种网络通信协议，使得程序可以调用另一台计算机上的程序或服务，就像调用本地的程序一样。RPC 的主要目的是简化分布式计算，使得开发者无需关注底层的网络通信细节。

2.1 RPC 的基本概念

客户端和服务器:

客户端：发起 RPC 请求的程序。

服务器：接收 RPC 请求并执行相应过程的程序。

代理:

客户端代理：封装请求并将其发送到服务器。

服务器代理：接收请求，解包并调用本地过程，之后将结果返回给客户端代理。

通信机制:

传输协议：底层使用的协议，比如 TCP、UDP。

数据序列化：将数据结构或对象转换成可以传输的格式，比如 JSON、XML、Protocol Buffers。

RPC 工作流程:

1、客户端调用本地代理方法：客户端调用一个看似本地的方法，但实际上这个方法由客户端代理负责处理。

2、客户端代理序列化请求：将方法名、参数等信息打包成消息。

3、消息传输：客户端代理将消息通过网络传输到服务器。

4、服务器代理解包请求：接收到消息后，服务器代理解包消息并调用实际的本地方法。

5、执行方法并生成响应：本地方法执行完毕后，生成响应结果。

6、服务器代理打包响应：服务器代理将结果打包成消息并通过网络发送回客户端。

7、客户端代理解包响应：客户端代理接收到响应消息后，解包并将结果返回给客户端。

3. TSMaster RPC 应用

3.1 RPC 功能

基于 TSMaster 的 RPC 机制，用户可以在 TSMaster 上位机环境中搭建完整的工程，涵盖从测试用例开发到程控设备管理、总线通讯配置以及控制板卡操作等一系列流程。通过这一机制，用户能够高效地在客户端远程控制 TSMaster 服务器，实现对**系统变量、CAN 信号、LIN 信号、FlexRay 信号以及以太网（ETH）信号等的读写操作**。此外，用户还可以**调用 TSMaster 服务器上定义的各种函数**，进一步扩展和定制系统功能。

这种集成化的解决方案使得工程管理和自动化测试更加便捷和高效。用户无需在多个平台之间切换，即可完成包括硬件配置、信号监控、数据采集和测试执行等复杂任务。通过 TSMaster 的 RPC 机制，用户在客户端即可实现对服务器的远程程控，简化了操作流程，提高了测试和开发效率，确保了系统的稳定性和可靠性。

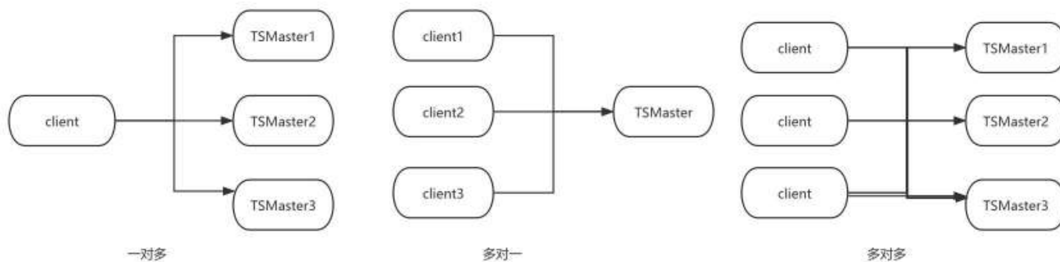
TSMaster 提供了强大的接口和丰富的功能模块，用户可以根据需求灵活组合使用，实现对各种信号和设备的精细控制和管理。这种架构不仅适用于研发测试阶段，也适用于生产环境中的实时监控和故障诊断，极大提升了工程项目的整体质量和效率。

不仅如此，TSMaster 的 RPC 机制支持多种拓扑结构，不仅可以实现客户端与服务器之间的**一对一**通信，还能够实现以下复杂的通信拓扑：

1：**一对多 (One-to-Many)**：单个客户端可以同时控制多个 TSMaster 服务器，适用于需要同时管理多个测试环境或设备的情况。

2：**多对一 (Many-to-One)**：多个客户端可以同时连接到一个 TSMaster 服务器，这样不同的用户或测试系统可以共享同一个服务器资源，实现协同工作和资源共享。

3：**多对多 (Many-to-Many)**：多个客户端和多个服务器之间可以进行灵活的通信和控制，构建复杂的分布式测试和控制系统，适用于大型工程项目和分布式测试环境。



这种灵活的拓扑结构使得 TSMaster 能够适应各种复杂的工程需求，无论是在单一项目中还是在跨项目、跨地域的分布式测试中，都能提供高效、可靠的解决方案。通过这种多样化的通信模式，用户可以最大限度地利用硬件和软件资源，提高系统的扩展性和灵活性，满足不同规模和复杂度的工程项目需求。

4. TSMaster RPC 使用说明

基于RPC本身机制，TSMaster提供了相应接口，在需要被程控的工程中，激活RPC server端，为程控脚本提供相应资源。

4.1 激活 server

开启server端操作如下：

新建一个C脚本，在启动事件中输入下面代码，即表示激活了当前工程的rpc server

```
rpc_tsmaster_activate_server(true);
```

实际上在，TSMaster v2024.06.05.1124 版本之后，所有TSMaster工程已经默认激活了RPC功能。

4.2 激活 client

```
native_int h; //client 句柄
//参数 1 为提供 rpc 服务的 TSMaster 应用程序名
com.rpc_tsmaster_create_client(“TSMaster”,&h);
//激活 client 端
Com.rpc_tsmaster_activate_client(h,true);
```

4.3 修改 server 端数据

4.3.1 启动停止 server 工程

启动工程：

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_start_simulation(h);
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_start_simulation(h);
```

4.3.2 读写系统变量

设置系统变量：

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_write_system_var(h, "Var1", "1.2345");
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_write_system_var(h, "Var1", "1.2345");
```

获取系统变量：

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_read_system_var(h, "Var1", "1.2345");
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_read_system_var(h, "Var1", "1.2345");
```

4.3.3 读写 CAN 信号

设置 CAN 信号：

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_set_can_signal(h, "chnidx/net/node/msg/signal", 1234)
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_set_can_signal(h, "chnidx/net/node/msg/signal", 1234)
```

获取 CAN 信号：

double d = 0;

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_get_can_signal(h, "chnidx/net/node/msg/signal", %d)
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_get_can_signal(h, "chnidx/net/node/msg/signal", %d)
```

4.3.4 读写 LIN 信号

设置 LIN 信号：

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_set_lin_signal(h, "chnidx/net/node/msg/signal", 1234);
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_set_lin_signal(h, "chnidx/net/node/msg/signal", 1234);
```

获取 LIN 信号：

double d = 0;

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_get_lin_signal(h, "chnidx/net/node/msg/signal", %d);
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_get_lin_signal(h, "chnidx/net/node/msg/signal", %d);
```

4.3.5 读写 FlexRay 信号

设置 FR 信号：

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_set_flexray_signal(h, "chnidx/net/node/msg/signal", 1234)
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_set_flexray_signal(h, "chnidx/net/node/msg/signal", 1234)
```

获取 FR 信号：

double d = 0;

TSMaster 小程序：

```
com.rpc_tsmaster_cmd_get_can_signal(h, "chnidx/net/node/msg/signal", %d)
```

API(C\C++\C#\Python):

```
rpc_tsmaster_cmd_get_can_signal(h, "chnidx/net/node/msg/signal", %d)
```

4.3.6 rpc 使用 TSMaster 系统函数

```
// 第一步：准备调用函数的输入参数
#define STR_BUFFER_SIZE 1024
char args[4][STR_BUFFER_SIZE];
char* pArgs[4] = {&args[0][0], &args[1][0], &args[2][0], &args[3][0]};
sprintf_s(pArgs[0], STR_BUFFER_SIZE, "%s", "var1");
sprintf_s(pArgs[1], STR_BUFFER_SIZE, "%d", svtString);
sprintf_s(pArgs[2], STR_BUFFER_SIZE, "%s", "string default value");
sprintf_s(pArgs[3], STR_BUFFER_SIZE, "%s", "this is a comment");

// 步骤 2：调用任意 API
s32 ret;
ret = com.rpc_tsmaster_call_system_api(h, "app.create_system_var", 4, STR_BUFFER_SIZE,
&pArgs[0]);

// 步骤 3：处理参数中的返回值（如果可用）
s32 i;
log("API call result = %d", ret);
for (i=0; i<4; i++){
  log("Argument %d: %s", i+1, pArgs[i]);
}
```

上述代码等价与在 TSMaster 进程中使用 app.create_system_var 来创建系统变量，即：app.create_system_var(var1,svtString,”string default value”,”this is a comment”);

需要注意的是，使用该方式调用 TSMaster 内的系统函数，无法使用参数为指针类型(报文类型除外)的函数。

4.3.7 rpc 使用小程序库函数

```
// 第一步：准备调用函数的输入参数
#define STR_BUFFER_SIZE 1024
char args[4][STR_BUFFER_SIZE];
char* pArgs[4] = {&args[0][0], &args[1][0], &args[2][0], &args[3][0]};
sprintf_s(pArgs[0], STR_BUFFER_SIZE, "%s", "var1");
sprintf_s(pArgs[1], STR_BUFFER_SIZE, "%d", svtString);
sprintf_s(pArgs[2], STR_BUFFER_SIZE, "%s", "string default value");
sprintf_s(pArgs[3], STR_BUFFER_SIZE, "%s", "this is a comment");

//步骤 2：调用任意 API
s32 ret;
```

```
ret = com.rpc_tsmaster_call_library_api(h, "mylib.create_system_var", 4, STR_BUFFER_SIZE, &pArgs[0]);

// 步骤 3：处理参数中的返回值（如果可用）
s32 i;
log("API call result = %d", ret);
for (i=0; i<4; i++){
   log("Argument %d: %s", i+1, pArgs[i]);
}
```

5. TSMaster RPC 函数说明

1. rpc_tsmaster_create_client

函数名称	UInt32 rpc_tsmaster_create_client(const char* ATSMasterAppName, const psize_t AHandle);
功能介绍	创建一个 TSMaster Rpc 客户端
调用位置	初始化 tsmaster lib 库之后
输入参数	ATSMasterAppName: TSMaster server 端的应用程序名称; AHandle : TSMaster Rpc 客户端句柄
返回值	==0: 函数执行成功 其他值：函数执行失败
示例	s32 h; rpc_tsmaster_create_client("TSMaster1", &h);

2. rpc_tsmaster_activate_client

函数名称	UInt32 rpc_tsmaster_activate_client(const size_t AHandle, const bool AActivate)
功能介绍	激活或者停用一个 TSMaster Rpc 客户端
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 AActivate: true=激活，false=停用
返回值	==0: 函数执行成功 其他值：函数执行失败
示例	rpc_tsmaster_activate_client(h, true);

3. rpc_tsmaster_is_simulation_running

函数名称	UInt32 rpc_tsmaster_is_simulation_running(const size_t AHandle,const pbool AIsRunning)
功能介绍	获取远程 TSMaster 仿真运行的状态
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 AIsRunning：远程 TSMaster 仿真运行的状态的数据指针 True=正在运行，false=未运行
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	bool b;
	if (0 == com.rpc_tsmaster_is_simulation_running(h, &b)){
	  if (b){
	    // current simulation is running
	  }
	}

4. rpc_tsmaster_cmd_set_mode_realtime

函数名称	UInt32 rpc_tsmaster_cmd_set_mode_realtime(const size_t AHandle)
功能介绍	将 TSMaster Rpc server 配置为实时模式
调用位置	创建 TSMaster Rpc 客户端之后，仿真运行之前
输入参数	AHandle : TSMaster Rpc 客户端句柄
返回值	==0: 函数执行成功 其他值：函数执行失败
示例	rpc_tsmaster_cmd_set_mode_realtime(h);

5. rpc_tsmaster_cmd_set_mode_sim

函数名称	UInt32 rpc_tsmaster_cmd_set_mode_sim(const size_t AHandle)
功能介绍	将 TSMaster Rpc server 配置为仿真模式
调用位置	创建 TSMaster Rpc 客户端之后，仿真运行之前
输入参数	AHandle : TSMaster Rpc 客户端句柄
返回值	==0: 函数执行成功 其他值：函数执行失败
示例	rpc_tsmaster_cmd_set_mode_sim(h);

6. rpc_tsmaster_cmd_start_simulation

函数名称	UInt32 rpc_tsmaster_cmd_start_simulation(const size_t AHandle)
功能介绍	启动 TSMaster Rpc server 仿真
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄
返回值	==0: 函数执行成功 其他值：函数执行失败
示例	rpc_tsmaster_cmd_start_simulation(h);

7. rpc_tsmaster_cmd_set_can_signal

函数名称	UInt32 rpc_tsmaster_cmd_set_can_signal(const size_t AHandle, const char* ASgnAddress, const double AValue)
功能介绍	在远程 TSMaster 上修改数据库中的 CAN 信号值
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ASgnAddress：数据库中信号的路径 AValue: 信号值
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	if(0==rpc_tsmaster_cmd_set_can_signal(h,
	"0/CAN_FD_Powertrain/Engine/EngineData/EngSpeed", 1234)){
	  // signal written
	}

8. rpc_tsmaster_cmd_get_can_signal

函数名称	UInt32 rpc_tsmaster_cmd_get_can_signal(const size_t AHandle, const char* ASgnAddress, const pdouble AValue)
功能介绍	在远程 TSMaster 上获取数据库中的 CAN 信号值
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ASgnAddress：数据库中信号的路径 AValue: 信号值指针
返回值	==0: 函数执行成功 其他值：函数执行失败
示例	double d; if(0==rpc_tsmaster_cmd_get_can_signal(h,

	"0/CAN_FD_Powertrain/Engine/EngineData/EngSpeed", &d)){
	  // signal is retrieved
	}

9. rpc_tsmaster_cmd_set_lin_signal

函数名称	UInt32 rpc_tsmaster_cmd_set_lin_signal(const size_t AHandle, const char* ASgnAddress, const double AValue)
功能介绍	在远程 TSMaster 上修改数据库中的 LIN 信号值
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ASgnAddress：数据库中信号的路径 AValue: 信号值
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	if(0==rpc_tsmaster_cmd_set_lin_signal(h, "chnidx/net/node/msg/signal", 1234)){
	  // signal written
	}

10. rpc_tsmaster_cmd_get_lin_signal

函数名称	UInt32 rpc_tsmaster_cmd_get_lin_signal(const size_t AHandle, const char* ASgnAddress, const pdouble AValue)
功能介绍	在远程 TSMaster 上获取数据库中的 LIN 信号值
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ASgnAddress：数据库中信号的路径 AValue: 信号值指针
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	double d;
	if(0==rpc_tsmaster_cmd_get_lin_signal(h, "chnidx/net/node/msg/signal", &d)){
	  // signal is retrieved
	}

11. rpc_tsmaster_cmd_set_flexray_signal

函数名称	UInt32 rpc_tsmaster_cmd_set_flexray_signal(const size_t

	AHandle, const char* ASgnAddress, const double AValue)
功能介绍	在远程 TSMaster 上修改数据库中的 flexray 信号值
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ASgnAddress：数据库中信号的路径 AValue: 信号值
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	if(0==rpc_tsmaster_cmd_set_flexray_signal(h, "chnidx/net/node/msg/signal", 1234)){
	// signal written
	}

12. rpc_tsmaster_cmd_get_flexray_signal

函数名称	UInt32 rpc_tsmaster_cmd_get_flexray_signal(const size_t AHandle, const char* ASgnAddress, const pdouble AValue)
功能介绍	在远程 TSMaster 上获取数据库中的 flexray 信号值
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ASgnAddress：数据库中信号的路径 AValue: 信号值指针
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	double d;
	if(0==rpc_tsmaster_cmd_get_flexray_signal(h, "chnidx/net/node/msg/signal", &d)){
	// signal is retrieved
	}

13. rpc_tsmaster_cmd_write_system_var

函数名称	UInt32 rpc_tsmaster_cmd_write_system_var(const size_t AHandle, const char* ACompleteName, const char* AValue)
功能介绍	从远程 TSMaster 按名称写入系统变量
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ACompleteName：系统变量名称 AValue: 数据值
返回值	==0: 函数执行成功

	其他值：函数执行失败
示例	rpc_tsmaster_cmd_write_system_var(h, ″v1″, ″1.2345″);

14. rpc_tsmaster_cmd_read_system_var

函数名称	UInt32 rpc_tsmaster_cmd_read_system_var(const size_t AHandle,const char* ASysVarName,const pdouble AValue)
功能介绍	从远程 TSMaster 按名称读取系统变量
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ASysVarName：系统变量名称 AValue: 数据值指针
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	double d;
	if (0 == rpc_tsmaster_cmd_read_system_var(h, ″v1″, &d)){
	  log(″value = %f″, d);
	}

15. rpc_tsmaster_cmd_write_signal

函数名称	UInt32 rpc_tsmaster_cmd_write_signal(const size_t AHandle,const TLIBApplicationChannelType ABusType,const char* AAddr,const double AValue)
功能介绍	从远程 TSMaster 按名称写入信号值
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ABusType：总线类型 AAddr：数据库中信号的路径 AValue: 数据值
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	if(0==rpc_tsmaster_cmd_write_signal(h,APP_CAN, ″0/Powertrain/Engine/EngSpeed″, 1234)){
	  // value written
	}

16. rpc_tsmaster_cmd_read_signal

函数名称	UInt32 rpc_tsmaster_cmd_read_signal(const size_t AHandle,const TLIBApplicationChannelType ABusType,const char* AAddr,const pdouble AValue)
功能介绍	从远程 TSMaster 按名称读取信号值
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 ABusType：总线类型 AAddr：数据库中信号的路径 AValue: 数据值指针
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	double d;
	if(0==rpc_tsmaster_cmd_read_signal(h,APP_CAN,"0/Powertrain/Engine/EngSpeed", &d)){
	  log("signal value = %f", d);
	}

17. rpc_tsmaster_delete_client

函数名称	UInt32 rpc_tsmaster_delete_client(const size_t AHandle)
功能介绍	删除 rpc client
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄
返回值	==0: 函数执行成功 其他值：函数执行失败
示例	rpc_tsmaster_delete_client(h);

18. rpc_tsmaster_cmd_stop_simulation

函数名称	UInt32 rpc_tsmaster_cmd_stop_simulation(const size_t AHandle)
功能介绍	停止远程 TSMaster 仿真
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄
返回值	==0: 函数执行成功 其他值：函数执行失败
示例	rpc_tsmaster_cmd_stop_simulation(h);

19. rpc_tsmaster_call_system_api

函数名称	UInt32 rpc_tsmaster_call_system_api(const size_t AHandle,const char* AAPIName,const s32 AArgCount,const s32 AArgCapacity,const char** AArgs)
功能介绍	**Client 调用 server 端使用系统函数**
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 AAPIName：系统函数名称，"app.create_system_var" AArgCount: 函数参数数量 AArgCapacity：参数字符串长度 AArgs：参数字符串数组
返回值	==0: 函数执行成功 其他值：函数执行失败

示例	// 第一步：准备调用函数的输入参数
	#define STR_BUFFER_SIZE 1024
	char args[4][STR_BUFFER_SIZE];
	char* pArgs[4] = {&args[0][0], &args[1][0], &args[2][0], &args[3][0]};
	sprintf_s(pArgs[0], STR_BUFFER_SIZE, "%s", "var1");
	sprintf_s(pArgs[1], STR_BUFFER_SIZE, "%d", svtString);
	sprintf_s(pArgs[2], STR_BUFFER_SIZE, "%s", "string default value");
	sprintf_s(pArgs[3], STR_BUFFER_SIZE, "%s", "this is a comment");
	
	// 步骤 2：调用任意 API
	s32 ret;
	ret = com.rpc_tsmaster_call_system_api(h, "app.create_system_var", 4, STR_BUFFER_SIZE, &pArgs[0]);
	
	// 步骤 3：处理参数中的返回值（如果可用）
	s32 i;
	log("API call result = %d", ret);
	for (i=0; i<4; i++){
	   log("Argument %d: %s", i+1, pArgs[i]);
	}

20. rpc_tsmaster_call_library_api

函数名称	UInt32 rpc_tsmaster_call_library_api(const size_t AHandle,const char* AAPIName,const s32 AArgCount,const s32 AArgCapacity,const

	char** AArgs)
功能介绍	**Client 调用 server 端使用系统函数**
调用位置	创建 TSMaster Rpc 客户端之后
输入参数	AHandle : TSMaster Rpc 客户端句柄 AAPIName：小程序库函数名称，"mylib.create_system_var" AArgCount：函数参数数量 AArgCapacity：参数字符串长度 AArgs：参数字符串数组
返回值	==0：函数执行成功 其他值：函数执行失败

示例	// 第一步：准备调用函数的输入参数
	#define STR_BUFFER_SIZE 1024
	char args[4][STR_BUFFER_SIZE];
	char* pArgs[4] = {&args[0][0], &args[1][0], &args[2][0], &args[3][0]};
	sprintf_s(pArgs[0], STR_BUFFER_SIZE, "%s", "var1");
	sprintf_s(pArgs[1], STR_BUFFER_SIZE, "%d", svtString);
	sprintf_s(pArgs[2], STR_BUFFER_SIZE, "%s", "string default value");
	sprintf_s(pArgs[3], STR_BUFFER_SIZE, "%s", "this is a comment");
	
	//步骤 2：调用任意 API
	s32 ret;
	ret = com.rpc_tsmaster_call_library_api(h, "mylib.create_system_var", 4, STR_BUFFER_SIZE, &pArgs[0]);
	// 步骤 3：处理参数中的返回值（如果可用）
	s32 i;
	log("API call result = %d", ret);
	for (i=0; i<4; i++){
	    log("Argument %d: %s", i+1, pArgs[i]);
	}