0、前言

概念：

DBus通信是IPC通信机制的一种方式，它有两种模式，分别为：session(会话模式)、system(总线模式)。

总线模式：采用总线模式时，系统需要维护一个DBus Daemon，每个DBus的请求通过DBus Daemon转发。这种模式Server只需要维护一个与DBus Daemon的链接，结构清晰，可以通过广播的方式发送消息到各个Client。

会话模式：这种模式一般称之为点对点的星型结构，Client与Server之间是直接的Peer2Peer的连接，少了DBus Daemon的中转，因此性能较好。

特点：

DBus实现进程间通信，最底层是靠socket的方式，DBus是socket通信方式的封装，简化代码的编写。

1、编写cn.com.itep.echo.xml文件

GDBus使用Introspection XML来描述RPC的接口。在这个xml文件中首先要描述要使用的接口。接口的名字与域名类似，采用“.”分隔，如cn.com.itep.echo。然后是属于这个接口的方法。需要描述方法的名字，该方法参数的名字、类型和方向。

```
<node>
    <interface name="cn.com.itep.echo">
        <method name="echo">
            <arg name="content" type="s" direction="in"/>
        </method>
    </interface>
</node>
```

2、用已有的cn.com.itep.echo.xml文件，通过gdbus-codegen命令生成C语言代码。

```
gdbus-codegen --generate-c-code generated-code \
                --c-namespace Itep    \
                --interface-prefix cn.com.itep.  \
                cn.com.itep.echo.xml
```

生成generated-code.c 与 generated-code.h

3、编写服务端程序，将生成的generated-code.h包含进来。

```
#include "generated-code.h"
```

1)编写服务器中供客户端调用的函数。

```
static gboolean on_echo(ItepEcho *echo,
                        GDBusMethodInvocation *invocation,
                        gchar *content,
                        gpointer user_data)
{
    g_printf("%s\n",content);
    itep_echo_complete_echo(echo, invocation); //声明函数执行完毕
    return TRUE;
}
```

2)编写GBusAcquiredCallback实现函数(连接到DBus总线的时候调用);

```
static void on_bus_acquired(GDBusConnection *connection,
                            const gchar *name,
                            gpointer user_data)
{
    ItepEcho *echo = NULL;
    echo = itep_echo_skeleton_new();//在DBus接口中，创建一个skeleton对象
    g_signal_connect(echo, "handle-echo", G_CALLBACK(on_echo),NULL);
    g_dbus_interface_skeleton_export(G_DBUS_INTERFACE_SKELETON(echo), connection,
"/cn/com/itep/echo",NULL);//导出该接口
```

```
    }
```

3)编写GBusNameAcquiredCallback实现函数(获得名字的时候调用);

```
    static void on_name_acquired(GDBusConnection *connection,
                                 const gchar *name,
                                 gpointer user_data)
    {
        g_printf("Acquired the name %s\n",name);
    }
```

4)编写GBusNameLostCallback实现函数(连接关闭或获取名字失败的时候调用);

```
    static void  on_name_lost(GDBusConnection *connection,
                              const gchar *name,
                              gpointer user_data)
    {
        g_printf("Lost the name %s\n",name);
    }
```

4.服务器端的主函数。

```
gint main(gint argc, gchar *argv[])
{
    GMainLoop *loop;
    guint id;
    loop = g_main_loop_new(NULL,FALSE);

    id = g_bus_own_name(G_BUS_TYPE_SESSION,
                        "cn.com.itep.echo",
                        G_BUS_NAME_OWNER_FLAGS_ALLOW_REPLACEMENT |
G_BUS_NAME_OWNER_FLAGS_REPLACE,
                        on_bus_acquired,
                        on_name_acquired,
                        on_name_lost,
                        loop, //传递参数
                        NULL);

    g_main_loop_run(loop);
    g_bus_unown_name(id);
    g_main_loop_unref(loop);
    return 0;
}
```

5、编写客户端程序，将生成的generated-code.h包含进来。

```
#include "generated-code.h"

gint main(gint argc, gchar *argv[])
{
    GError *error = NULL;
    ItepEcho *echo = itep_echo_proxy_new_for_bus_sync(
                        G_BUS_TYPE_SESSION,
                        G_DBUS_PROXY_FLAGS_NONE,
                        "cn.com.itep.echo",
                        "/cn/com/itep/echo",
                        NULL,
                        &error);//获取远程接口的代理对象

    if (echo == NULL)
    {
```

```
        g_printf("Error getting proxy: %s\n", error->message);
        g_error_free(error);
        return 0;
    }
    itep_echo_call_echo_sync(echo,"abc",NULL,NULL);//调用远程方法
    g_object_unref(echo);
    return 0;
}
```

6、编译

服务器端：

```
gcc server.c -o server `pkg-config --libs --cflags gtk+-2.0`
```

客户端：

```
gcc client.c -o client `pkg-config --libs --cflags gtk+-2.0`
```