**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,

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

客户端：

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