

linux嵌入式驱动软件开发

android底层开发和移植

目录视图

摘要视图

RSS 订阅

个人资料



林伟

访问：829246次
积分：10988
等级：

BLOG 7

排名：第981名

原创：249篇 转载：97篇
译文：0篇 评论：118条

文章搜索

文章分类

- android bug及解决方法 (5)
- android framework (87)
- android NDK 开发 (5)
- android-java (6)
- arm体系架构 (25)
- bluetooth (1)
- c/c++数据结构和常用算法及其分析 (0)
- chrome (8)
- dalvik的研究与分析 (3)
- gnu 编译器 (3)
- lcd/led/oled相关知识 (6)
- linux graphics study (3)
- linux kernel 的分析 (8)
- linux 多线程编程 (1)
- linux 电源管理 (2)
- linux嵌入式驱动开发 (37)
- linux音视频编解码 (12)
- matlab 数学建模与仿真 (0)
- media framework (17)
- mips 体系架构设计 (1)

【CSDN技术主题月】深度学习框架的重构与思考 【观点】有了深度学习，你还学传统机器学习算法么？ 【知识库】深度学习知识图谱上线啦

D-Bus 体系

标签： object session signal interface server socket

2009-12-24 11:27 4329人阅读 评论(0) 收藏 举报

分类： openbinder和dbus (5)

版权声明：本文为博主原创文章，未经博主允许不得转载。

有很多种IPC或者网络通信系统，如：CORBA, DCE, DCOM, DCOP, XML-RPC, SOAP, MBUS, Internet Communications Engine (ICE)等等，可能会有数百种，dbus的目的主要是下面两点：

- 1.在同一个桌面会话中，进行桌面应用程序之间的通讯
- 2.桌面程序与内核或者守护进程的通信。

Dbus是一套进程通信体系，它有以下几层：

- 1.libdbus库，提供给各个应用程序调用，使应用程序具有通信和数据交换的能力，两个应用程序可以直接进行通信，就像是一条socket通道，两个程序之间建立通道之后，就可以通讯了。
- 2.消息守护进程，在libdbus的基础上创建，可以管理多个应用程序之间的通信。每个应用程序都和消息守护进程建立dbus的链接，然后由消息守护进程进行消息的分派。
- 3.各种包装库，有libdbus-glib, libdbus-qt等等，目的是将dbus的底层api进行一下封装。

- [openbinder和dbus \(6\)](#)
- [RTOS----RTlinux/culinux/ucos-ll/ecos \(2\)](#)
- [u-boot 的源码分析 \(3\)](#)
- [unix编程--c/c++ \(16\)](#)
- [unix网络编程 \(26\)](#)
- [wifi \(3\)](#)
- [开源硬件设计 \(1\)](#)
- [心情日志 \(4\)](#)
- [数字图像处理 \(2\)](#)
- [数字视频编码格式 \(12\)](#)
- [文件系统的分析 \(9\)](#)
- [相关开源网站 \(2\)](#)
- [硬件选材 \(2\)](#)
- [网卡驱动 \(2\)](#)
- [面试 \(1\)](#)

文章存档

- [2011年05月 \(1\)](#)
- [2011年03月 \(3\)](#)
- [2011年02月 \(2\)](#)
- [2011年01月 \(3\)](#)
- [2010年12月 \(1\)](#)

展开

阅读排行

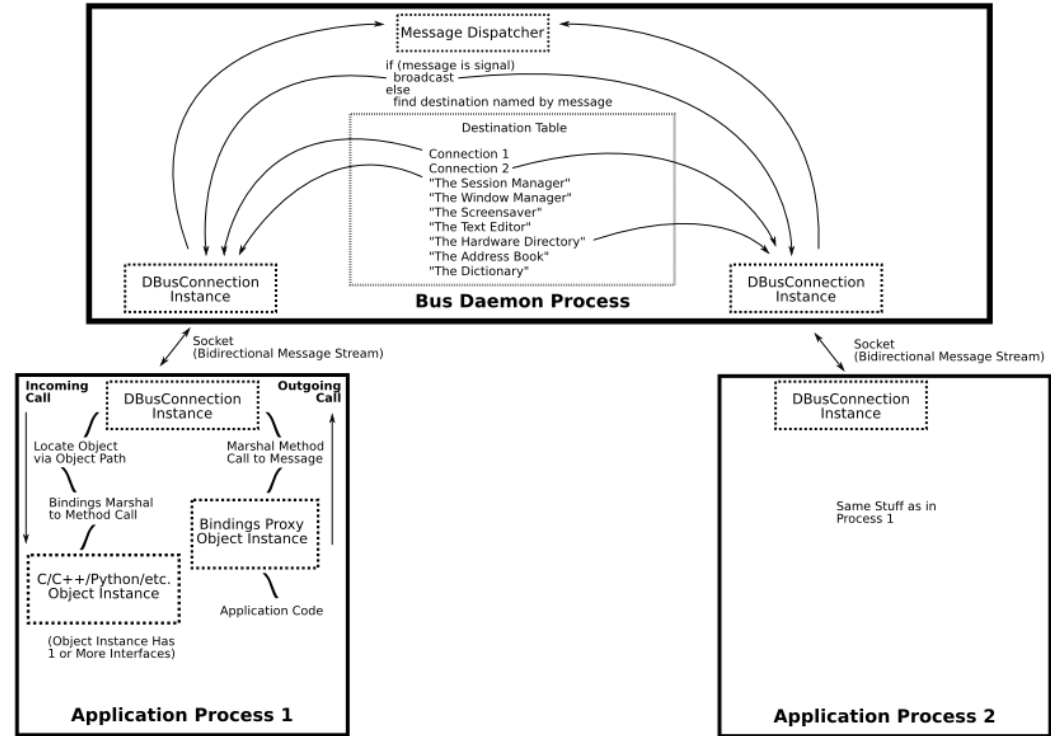
- [error: insufficient permis \(14040\)](#)
- [Android PMEM驱动研究 \(11559\)](#)
- [Dbus组成和原理 \(11269\)](#)
- [alsa 音频库的移植 \(10958\)](#)
- [Android权限获取机制与 \(9258\)](#)
- [移植rp-pppoe到s3c244C \(9194\)](#)
- [Ubuntu上架设PPPoE Se \(8943\)](#)
- [HDMI之EDID \(8757\)](#)
- [android 改变线程优先级 \(8514\)](#)
- [Linux的cpufreq（动态变 \(8447\)](#)

评论排行

- [中国做技术没前途 \(9\)](#)
- [android lk机制介绍 \(6\)](#)
- [ADROID 2.1 架构解析 语 \(6\)](#)
- [Android内核和驱动篇-Ar \(6\)](#)
- [Android中通过按键旋转 \(4\)](#)
- [Android移植之-dropbear \(4\)](#)
- [android下的开源库 \(4\)](#)
- [移植rp-pppoe到s3c244C \(3\)](#)
- [error: insufficient permis \(3\)](#)
- [内存调试技巧 \(3\)](#)

推荐文章

下面有一张图可以很方便说明dbus的体系结构。



dbus中的消息由一个消息头（标识是哪一种消息）和消息数据组成，比socket的流式数据更方便一些。bus daemon 就像是一个路由器，与各个应用程序进行连接，分派这些消息。bus daemon 在一台机器上有多个实例，第一个实例是全局的实例，类似于sendmail和或者apache，这个实例有很严格的安全限制，只接受一些特定的系统消息，用于系统通信。其他bus daemon是一些会话，用于用户登录之后，在当前会话(session)中进行的通讯。系统的bus daemon 和会话的bus daemon 是分开的，彼此不会互相影响，会话bus daemon 不会去调用系统的bus daemon 。

Native Objects and Object Paths

在不同的编程语言中，都定义了一些“对象”，如Java中的java.lang.Object，GLIB中的GObject，QT中的QObject等等。D-BUS的底层接口，和libdbus API相关，是没有这些对象的概念的，它提供的是一种叫对象路径（object path），用于让高层接口绑定到各个对象中去，允许远端应用程序指向它们。object path就像是一个文件路径，可以叫做/org/kde/kspread/sheets/3/cells/4/5等。

Methods and Signals

每个对象都有一些成员，两种成员：方法(methods)和信号(signals)，在对象中，方法可以被调用。信号会被广播，感兴趣的对象可以处理这个信号，同时信号中也可以带有相关的数据。每一个方法或者信号都可以用一个名字来命名，如“Frobate”或者“OnClicked”。

Interfaces

每个对象都有一个或者多个接口，一个接口就是多个方法和信号的集合。dbus使用简单的命名空间字符串来表示接口，如org.freedesktop.Introspectable。可以说dbus接口相当于C++中的纯虚类。

Proxies

代理对象用于模拟在另外的进程中的远端对象，代理对象像是一个正常的普通对象。d-bus的底层接口必须手动创建方法调用的消息，然后发送，同时必须手动接受和处理返回的消息。高层接口可以使用代理来替换这些，当调用代理对象的方法时，代理内部会转换成dbus的方法调用，等待消息返回，对返回结果解包，返回给相应的方法。可以看看下面的例子，使用dbus底层接口编写的代码：

```
Message message = new Message("/remote/object/path", "MethodName", arg1, arg2);
Connection connection = getBusConnection();
connection.send(message);
```

* 2016 年最受欢迎的编程语言是什么？

* Chromium扩展（Extension）的页面（Page）加载过程分析

* Android Studio 2.2 来啦

* 手把手教你做音乐播放器（二）技术原理与框架设计

* JVM 性能调优实战之：使用阿里开源工具 TProfiler 在海量业务代码中精确定位性能代码

最新评论

Linux Platform Device and Drive
测试一下: 手机上的sensor用了这个没?

Chrome源码剖析—Chrome的UI
快乐的骑士: it's interesting

android lk机制介绍
mylove2693: thanks very much.

中国做技术没前途
FOS_Jim: 对待钱的态度取决于你物质需求, 做技术的, 努力一点, 5年达到年薪25万应该问题不大。确实25w对于很多...

error: insufficient permissions fo
kangear: Gooooood.

ARM MMU工作原理剖析
sprindy: 虚拟地址8192的图片不对吧(图片上page index是“1000”, 图片下面的解释是“0010”...

Linux的cpufreq（动态变频）技
haichunzhao:

@maomaochong1989:设备驱动级应该指的是runtime这种形式吧。系统平台级睡眠指的是...

Android 图形系统剖析
newpb80: 博主 你好对Android研究 那么早就开始了, 应该很有造诣了, 为啥继续写博客文章了

BMP格式结构详解
shanshanlin: 学习了, 谢谢楼主

ARM MMU工作原理剖析
macoo_ma: 例4不论CPU处于何种模式下, 读写都会引起permission fault吧?

```
Message reply = connection.waitForReply(message);
```

```
if (reply.isError()) {
```

```
} else {
```

```
Object returnValue = reply.getReturnValue();
```

```
}
```

使用代理对象编写的代码:

```
Proxy proxy = new Proxy(getBusConnection(), "/remote/object/path");
```

```
Object returnValue = proxy.MethodName(arg1, arg2);
```

客户端代码减少很多。

Bus Names

当一个应用程序连接上bus daemon时, daemon会分配一个唯一的名字给它。以冒号(:)开始, 这些名字在daemon的生命周期中是不会改变的, 可以认为这些名字就是一个IP地址。当这个名字映射到应用程序的连接上时, 应用程序可以说拥有这个名字。同时应用可以声明额外的容易理解的名字, 比如可以取一个名字com.mycompany.TextEditor, 可以认为这些名字就是一个域名。其他应用程序可以往这个名字发送消息, 执行各种方法。

名字还有第二个重要的用途, 可以用于跟踪应用程序的生命周期。当应用退出(或者崩溃)时, 与bus的连接将被OS内核关掉, bus将会发送通知, 告诉剩余的应用程序, 该程序已经丢失了它的名字。名字还可以检测应用是否已经启动, 这往往用于只能启动一个实例的应用。

Addresses

使用d-bus的应用程序既可以是server也可以是client, server监听到来的连接, client连接到server, 一旦连接建立, 消息就可以流转。如果使用dbus daemon, 所有的应用程序都是client, daemon监听所有的连接, 应用程序初始化连接到daemon。

dbus地址指明server将要监听的地方, client将要连接的地方, 例如, 地址: unix:path=/tmp/abcdef表明server将在/tmp/abcdef路径下监听unix域的socket, client也将连接到这个socket。一个地址也可以指明是TCP/IP的socket, 或者是其他的。

当使用bus daemon时, libdbus会从环境变量中(DBUS_SESSION_BUS_ADDRESS)自动认识“会话daemon”的地址。如果是系统daemon, 它会检查指定的socket路径获得地址, 也可以使用环境变量(DBUS_SESSION_BUS_ADDRESS)进行设定。

当dbus中不使用daemon时, 需要定义哪一个应用是server, 哪一个应用是client, 同时要指明server的地址, 这不是很通常的做法。

Big Conceptual Picture

要在指定的对象中调用指定的方法, 需要知道的参数如下:

Address -> [Bus Name] -> Path -> Interface -> Method

bus name是可选的, 除非是希望把消息送到特定的应用中才需要。interface也是可选的, 有一些历史原因, DCOP不需要指定接口, 因为DCOP在同一个对象中禁止同名的方法。

Messages - Behind the Scenes

如果使用dbus的高层接口, 就可以不用直接操作这些消息。DBUS有四种类型的消息:

- 1.方法调用(method call) 在对象上执行一个方法
- 2.方法返回(method return) 返回方法执行的结果
- 3.错误(error) 调用方法产生的异常
- 4.信号(signal) 通知指定的信号发生了, 可以想象成“事件”。

要执行 D-BUS 对象的方法，需要向对象发送一个方法调用消息。它将完成一些处理并返回一个方法返回消息或者错误消息。信号的不同之处在于它们不返回任何内容：既没有“信号返回”消息，也没有任何类型的错误消息。

D-Bus basic type	Type	Free function	Notes
每个消息都有一个消息头，包含多个字段，有一个消息体，包含多个参数。可以认为消息头是消息的路由信息，消息体作为一个载体。消息头里面的字段包含发送的bus name，目标bus name，方法或者信号名字等，同时消息头里面定义的字段类型规定了消息体里面的数据格式。例如：字符“i”代表了“32-bit integer”，“ii”就代表了消息体里面有两个“32-bit integer”。			

Calling a Method - Behind the Scenes

在dbus中调用一个方法包含了两条消息，进程A向进程B发送方法调用消息，进程B向进程A发送应答消息。所有的消息都由daemon进行分派，每个调用的消息都有一个不同的序列号，返回消息包含这个序列号，以方便调用者匹配调用消息与应答消息。调用消息包含一些参数，应答消息可能包含错误标识，或者包含方法的返回数据。

方法调用的一般流程：

- 1.使用不同语言绑定的dbus高层接口，都提供了一些代理对象，调用其他进程里面的远端对象就像是在本地进程中的调用一样。应用调用代理上的方法，代理将构造一个方法调用消息给远端的进程。
- 2.在DBUS的底层接口中，应用需要自己构造方法调用消息（method call message），而不能使用代理。
- 3.方法调用消息里面的内容有：目的进程的bus name，方法的名字，方法的参数，目的进程的对象路径，以及可选的接口名称。
- 4.方法调用消息是发送到bus daemon中的。
- 5.bus daemon查找目标的bus name，如果找到，就把这个方法发送到该进程中，否则，daemon会产生错误消息，作为应答消息给发送进程。
- 6.目标进程解开消息，在dbus底层接口中，会立即调用方法，然后发送方法的应答消息给daemon。在dbus高层接口中，会先检测对象路径，接口，方法名称，然后把它转换成对应的对象（如GObject，QT中的QObject等）的方法，然后再将应答结果转换成应答消息发给daemon。
- 7.bus daemon接受到应答消息，将把应答消息直接发给发出调用消息的进程。
- 8.应答消息中可以包容很多返回值，也可以标识一个错误发生，当使用绑定时，应答消息将转换为代理对象的返回值，或者进入异常。

bus daemon不对消息重新排序，如果发送了两条消息到同一个进程，他们将按照发送顺序接受到。接受进程并需要按照顺序发出应答消息，例如在多线程中处理这些消息，应答消息的发出是没有顺序的。消息都有一个序列号可以与应答消息进行配对。

Emitting a Signal - Behind the Scenes

在dbus中一个信号包含一条信号消息，一个进程发给多个进程。也就是说，信号是单向的广播。信号可以包含一些参数，但是作为广播，它是没有返回值的。

信号触发者是不了解信号接受者的，接受者向daemon注册感兴趣的信号，注册规则是“match rules”，记录触发者名字和信号名字。daemon只向注册了这个信号的进程发送信号。


信号的一般流程如下：

- 1.当使用dbus底层接口时，信号需要应用自己创建和发送到daemon，使用dbus高层接口时，可以使用相关对象进行发送，如Glib里面提供的信号触发机制。
- 2.信号包含的内容有：信号的接口名称，信号名称，发送进程的bus name，以及其他参数。
- 3.任何进程都可以依据“match rules”注册相关的信号，daemon有一张注册的列表。
- 4.daemon检测信号，决定哪些进程对这个信号感兴趣，然后把信号发送给这些进程。
- 5.每个进程收到信号后，如果是使用了dbus高层接口，可以选择触发代理对象上的信号。如果是dbus底层接口，需要检查发送者名称和信号名称，然后决定怎么做。

Glib绑定接口在“dbus/dbus-glib.h”头文件中定义。

dbus和glib的数据类型映射如下：

D-Bus type signature	Description	GType	C typedef	Free function	Notes
D-Bus basic type					
BYTE		G_TYPE_UCHAR			
BOOLEAN		G_TYPE_BOOLEAN			
INT16		G_TYPE_INT			Will be changed to a G_TYPE_INT16 once GLib has it
UINT16		G_TYPE_UINT			Will be changed to a G_TYPE_UINT16 once GLib has it
INT32		G_TYPE_INT			Will be changed to a G_TYPE_INT32 once GLib has it
UINT32		G_TYPE_UINT			Will be changed to a G_TYPE_UINT32 once GLib has it
INT64		G_TYPE_GINT64			
UINT64		G_TYPE_GUINT64			
DOUBLE					
STRING				g_free	
OBJECT_PATH				g_object_unref	The returned proxy does not have an interface set; use dbus_g_proxy_set_interface to invoke methods



Container type mappings

dbus数据也有容器类型，像DBUS_TYPE_ARRAY 和 DBUS_TYPE_STRUCT，dbus的数据类型可以是嵌套的，如有一个数组，内容是字符串的数组集合。

但是，并不是所有的类型都有普通的使用，DBUS_TYPE_STRUCT应该可以包容非基本类型的数据类型。glib绑定尝试使用比较明显的方式进行声明。

D-Bus type signature	Description	GType	C typedef	Free function	Notes
as	Array of strings	G_TYPE_STRV	char **	g_strfreev	
v	Generic value container	G_TYPE_VALUE	GValue *	g_value_unset	The calling conventions for values expect that method callers have allocated return values; see below.

同时定义了新的数组类型集合。

D-Bus type signature	Description	GType	C typedef	Free function	Notes
ay	Array of bytes	DBUS_TYPE_G_BYTE_ARRAY	GArray *	g_array_free	
au	Array of uint	DBUS_TYPE_G_UINT_ARRAY	GArray *	g_array_free	
ai	Array of int	DBUS_TYPE_G_INT_ARRAY	GArray *	g_array_free	
ax	Array of int64	DBUS_TYPE_G_INT64_ARRAY	GArray *	g_array_free	
at	Array of uint64	DBUS_TYPE_G_UINT64_ARRAY	GArray *	g_array_free	

D-Bus type signature	Description	GType	C typedef	Free function	Notes
a{db}	Array of double	DBUS_TYPE_G_DOUBLE_ARRAY	GArray *	g_array_free	
ab	Array of boolean	DBUS_TYPE_G_BOOLEAN_ARRAY	GArray *	g_array_free	

定义了字典类型

D-Bus type signature	Description	GType	C typedef	Free function	Notes
a{ss}	Dictionary mapping strings to strings	DBUS_TYPE_G_STRING_STRING_HASHTABLE	GHashTable *	g_hash_table_destroy	

client app开发报价单

软件工程师薪水



我们需要连接上dbus，使用dbus_g_bus_get获得dbus连接。然后可以创建

需要调用方法的时候，可以有两种方式：1.同步调用，使用dbus_g_proxy_call发送方法请求到远端对象，dbus会阻塞等待远端对象的回应，输出参数里将会带有相应的回应数据,以G_TYPE_INVALID作为终止符。2.异步调用，使用dbus_g_proxy_begin_call，它将返回一个DBusGPendingCall对象，可以使用dbus_g_pending_call_set_notify连接到自己的处理函授中。

可以使用dbus_g_proxy_add_signal 和 dbus_g_proxy_connect_signal来连接信号，dbus_g_proxy_add_signal用来声明信号处理函数，属于必须被调用的接口，dbus_g_proxy_connect_signal可以调用多次。

Generated Bindings

使用内置的xml文件，可以很方便地自动创建出易于使用的dbus代理对象。如下的一个xml文件定义了一个方法：

```
<?xml version="1.0" encoding="UTF-8" ?>
<node name="/com/example/MyObject">
  <interface name="com.example.MyObject">
    <method name="ManyArgs">
      <arg type="u" name="x" direction="in" />
      <arg type="s" name="str" direction="in" />
      <arg type="d" name="trouble" direction="in" />
      <arg type="d" name="d_ret" direction="out" />
      <arg type="s" name="str_ret" direction="out" />
    </method >
  </interface >
</node >
```

“in”标识输入参数，“out”标识输出参数。
使用dbus-binding-tool工具来生成头文件，如dbus-binding-tool --mode=glib-client my-object.xml > my-object-bindings.h，会产生如下的内联函数原型：

```
/* This is a blocking call */
gboolean
com_example_MyObject_many_args (DBusGProxy *proxy, const guint IN_x,
const char * IN_str, const gdouble IN_trouble,
gdouble* OUT_d_ret, char ** OUT_str_ret,
GError **error);

/* This is a non-blocking call */
DBusGProxyCall*
com_example_MyObject_many_args_async (DBusGProxy *proxy, const guint IN_x,
const char * IN_str, const gdouble IN_trouble,
com_example_MyObject_many_args_reply callback,
gpointer userdata);
```



```
the non-blocking callback */
args_reply)
OUT_d_ret, char * OUT_str_ret,
userdata);
```

所有DBusProxy对象，一般是使用dbus_g_proxy_new_*函数创建出来的。客户端发起方法调用时，可以添加标记，目前只有org.freedesktop.DBus.GLib.NoReply标记，dbus可以不要回应消息，没有“out”参数，这样运算速度会快一点。

server端的编写

在GLib中，通过dbus表现出GObject，必须写XML文件描述这个对象的方法等属性。像上一篇文章中提到的例子：

```
<?xml version="1.0" encoding="UTF-8" ?>
<node name="/com/example/MyObject">
<interface name="com.example.MyObject">
<method name="ManyArgs">
<arg type="u" name="x" direction="in" />
<arg type="s" name="str" direction="in" />
<arg type="d" name="trouble" direction="in" />
<arg type="d" name="d_ret" direction="out" />
<arg type="s" name="str_ret" direction="out" />
</method >
</interface >
</node >
```

一旦写完XML，运行dbus-binding-tool工具，如 dbus-binding-tool --mode=glib-server my-object.xml > my-object-glue.h.

然后在本地代码中include产生的头文件，调用dbus_g_object_class_install_info进行类的初始化，传递对象和对象信息进去，如 dbus_g_object_type_install_info (COM_FOO_TYPE_MY_OBJECT, &com_foo_my_object_info);每个对象类都需要这样做。

为了执行方法，需要定义一个C函数，如my_object_many_args，需要遵守的规则如下：

- 1.函数返回gboolean，true表示成功，false标识失败。
- 2.第一个参数必须是对象实例的指针。
- 3.跟在实例指针后面的参数是方法的输入参数。
- 4.输入参数后面是输出参数。
- 5.最后一个参数必须是GError **，如果函数返回失败，必须使用g_set_error填充该错误参数。

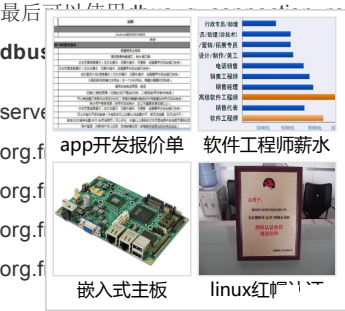
如下的xml文件

```
<method name="Increment">
  <arg type="u" name="x" />
  <arg type="u" direction="out" />
</method>
```

对应的函数定义为：

```
gboolean
my_object_increment (MyObject *obj, gint32 x, gint32 *ret, GError **error);
```

最后，在my_object.c中，使用g_object_register_g_object输出一个对象，如
g_object_register_g_object (connection, "/com/foo/MyObject", obj);



dbus启动问题

首先需要启动守护进程

```
dbus-daemon --system --print-pid --print-address
```

结果提示 Failed to start message bus: Could not get UID and GID for username "message"

dbus需要有一个messagebus用户，创建该用户即可，useradd messagebus，问题解决。

执行一个dbus测试程序，提示：D-Bus library appears to be incorrectly set up; failed to read

machine uid: Failed to open "/usr/var/lib/dbus/machine-id": No such file or directory

没有machine-id文件，查了一下，需要给它定义一个id，使用dbus-uuidgen

```
>/usr/var/lib/dbus/machine-id
```

产生这个文件，该问题解决。

再次执行测试程序，又有问题：Couldn't connect to session bus: Failed to execute dbus-launch to

autolaunch D-Bus session,看了帮助<http://dbus.freedesktop.org/doc/dbus-launch.1.html>

AUTOMATIC LAUNCHING一节，需要设置DBUS_SESSION_BUS_ADDRESS环境变量的值，先执行 dbus-launch,获得了DBUS_SESSION_BUS_ADDRESS值，再export一下，最后执行测试程序，OK了

在dbus帮助中有一篇关于dbus-launch的文章，可以在脚本中启动dbus-launch，同时自动设置

DBUS_SESSION_BUS_ADDRESS环境变量，脚本文件rundbus如下：

```
if test -z "$DBUS_SESSION_BUS_ADDRESS" ; then
## if not found, launch a new one
eval `dbus-launch --sh-syntax --exit-with-session`
```



```
echo "D-Bus per-session daemon address is: $DBUS_SESSION_BUS_ADDRESS"
fi
```

执行.rundbus即可。

顶

0

踩

0

- 上一篇
- What's in Android source package
- 下一篇
- Dbus组成和原理

我的同类文章

openk

- 7个Lin
- dbus实
- Dbus结



阅读 2979

• dbus实例讲解3

2009-12-24 阅读 2351

阅读 2766

• dbus实例讲解1

2009-12-24 阅读 7387

阅读 11270

参考知识库



Docker知识库
4194 关注 | 200 收录



Java EE知识库
7161 关注 | 705 收录



Java SE知识库
14281 关注 | 459 收录



Java 知识库
16228 关注 | 1305 收录

猜你在找

- JavaSE高级篇——（IO流+多线程+XML+Socket+swing）
- XML全面剖析加实战应用视频教程
- 在VC2015里学会使用tinyxml库
- XML技术
- 用redis 搭建大数据 热门排行榜，用户推荐系统
- uboot readme
- android linux 基础知识总结
- uboot readme
- viv代码分析一
- vivi源代码最为详细分析一

 JIGUANG | 极光

 JPush | 极光推送

极光推送 短信补充
降低通知成本 提升到达率

查看评论

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题

Hadoop

AWS

移动游戏

Java

Android

iOS

Swift

智能硬件

Docker

OpenStack

VPN

Spark

ERP

IE10

Eclipse

CRM

JavaScript

数据库

Ubuntu

NFC

WAP

jQuery

BI

HTML5

Spring

Apache

.NET

API

HTML

SDK

IIS

Fedora

XML

LBS

Unity

Splashtop

UML

components

Windows Mobile

Rails

QEMU

KDE

Cassandra

CloudStack

FTC

coremail

OPhone

CouchBase

云计算

iOS6

Rackspace

Web App

SpringSide

Maemo

Compuware

大数据

aptech

Perl

Tornado

Ruby

Hibernate

ThinkPHP

HBase

Pure

Solr

Angular

Cloud Foundry

Redis

Scala

Django

Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持
京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved

12



app开发报价单



软件工程师薪水



嵌入式主板



linux红帽