


个人资料



ty3219

访问：3849次

积分：131

等级：BLOG 2

排名：千里之外

原创：8篇

转载：3篇

译文：1篇

评论：2条

文章搜索

文章分类

Linux系统 (6)

android (3)

Linux编程 (4)

文章存档

2016年02月 (1)

2016年01月 (1)

2015年12月 (2)

2015年09月 (1)

2015年08月 (1)

展开

推荐文章

\* 2016 年最受欢迎的编程语言是什么？

【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

dbus介绍与例子

标签： dbus 例子 signal method dbus-daemon

2015-08-08 16:50 420人阅读 评论(0) 收藏 举报

分类：Linux编程 (3)

版权声明：本文为博主原创文章，未经博主允许不得转载。

D-bus是一个进程间通信的工具，优点不在此赘述。

网上很多关于dbus的帖子都是基于dbus-glib或者QT D-bus的，直接使用dbus的教程比较少。也难怪，因为连D-bus的官网都说：“If you use this low-level API directly, you're signing up for some pain.”

但实际上，直接使用D-bus也没有想象中难。本文将对直接使用D-bus做一个介绍。

本文参考了其他一些网站的帖子或者介绍

官网：<http://www.freedesktop.org/wiki/Software/dbus/>

经典例子：<http://www.matthew.ath.cx/articles/dbus>

不错的帖子：<http://blog.csdn.NET/flowingflying/article/details/4527634>

一、概念介绍

这里虽然说是概念介绍，其实只是我个人对D-bus的一个理解，不一定完整准确。

1. 首先，D-bus可以分成三部分来看，

(1) dbus-daemon，一个dbus的后台守护程序，用于多个应用之间消息的转发；

(2) libdbus.so，dbus的功能接口，当你的程序需要使用dbus时，其实就是调用libdbus.so里面的接口；

(3) 高层封装，如dbus-glib和QT D-bus，这些其实都对D-bus的再封装，让你使用起来更方便。

从D-bus官网下载到源码，其实只包含上面所说的 1 和 2 两部分，libdbus.so里面的接口也就是官网说的 low-level API。

2. 关于address、bus name、path。。。。

D-bus里面提到了一些概念，刚开始不太好理解，这些概念也很容易混淆。这些概念的权威解释可以看[这里](#)。

首先，运行一个dbus-daemon就是创建了一条通信的总线Bus。当一个application连接到这条Bus上面时，就产生了Connection。

每个application里面会有不同的Object。这里Object的概念，可以简单地理解为C++里面一个类的实例。从D-bus的概念上说，通信双方是Object，不是application，一个application是可以包含很多个Object的。

- \* 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



上海嵌入式培训

嵌入式主板

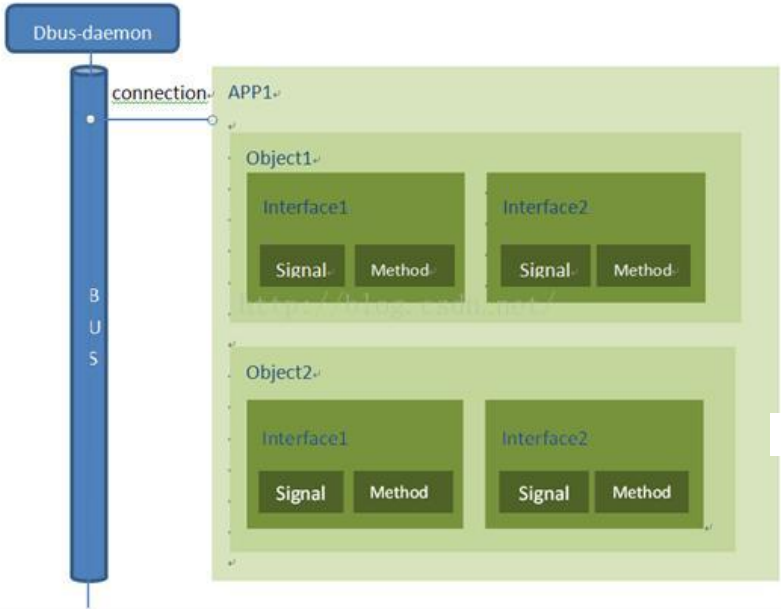
linux红帽认证

app开发

而一个Object里面又会有不同的Interface，这个Interface我把它理解为Object里面的一个类的成员。这些Interface其实是通信方式的集合。

这里又牵扯出来一个通信方式，D-bus里面支持的通信方式有两种，一种叫signal，一种叫method。signal简单地讲，其实就是广播，就是一对多的通信方式，可以从app1向其他所有的app发消息，但其他的app是不会对signal进行回复的。method则是一对一的通信，一问一答。这种方式有点像远程调用，app1调用app2的method并传递参数给这个method，获取到这个method返回的结果。

上面把D-bus通信里面的几个重要元素都介绍了一下，大概的关系是这样的：



几个重要的元素之间的关系都画出来了，那么在程序里面怎么去标识这些元素呢？这里又提出了一些名词address、bus name、path、Interface name。

（1）address是用来标识dbus-daemon的。当一个dbus-daemon运行以后，其他的app该怎么连接到这个dbus-daemon，靠的就是address。address的格式要求像这样：unix:path=/var/run/dbus/system\_bus\_socket。

（2）bus name是用来标识application的。当一个app1连接上dbus-daemon以后，相当于有了一个Connection，但其他的app2、app3怎么找到app1，靠的就是bus name。这个bus name标识了app1的Connection，也就相当于标识了app1。bus name由两种，一种是已冒号开头的唯一标识，像:34-907这样；另一种是通用的标识，是方便人看的，像com.mycompany.TextEditor。

（3）path用于标识Object。当app1的Object1要跟app2的Object2通信时，Object1要和Object2通信时，就要告诉dbus-daemon，Object2的path。path的格式像这样，/com/mycompany/TextFileManager，已“/”开头。

（4）每个Interface都会有自己的名字，也就是interface name，我们通过这个interface name就可以找到这个interface。interface name像这样org.freedesktop.Hal.Manager

（5）Signal和Method也有自己的名字，这个名字没什么特别的格式要求，随便改个名字就可以了。

官网上对这些标识列了一个表，如下：

A...	is identified by	which looks like...	and is chosen by...
Bus	address	unix:path=/var/run/dbus/system_bus_socket	system configuration
Connection	bus name	:34-907 (unique) or com.mycompany.TextEditor (well-known)	D-Bus (unique) or the owning program
Object	path	/com/mycompany/TextFileManager	the owning program
Interface	interface name	org.freedesktop.Hal.Manager	the owning program
Member	member name	ListNames	the owning program

二、例子

我在Matthew Johnson和恺风的例子基础上做了修改，如下：

```
[cpp]

01. #include <stdio.h>
02. #include <stdlib.h>
03. #include <string.h>
04. #include <unistd.h>
05. #include <dbus/dbus.h>
06.
07.
08.
09. /*
10.  * listen, wait a call or a signal
11.  */
12. #define DBUS_SENDER_BUS_NAME      "com.ty3219.sender_app"
13.
14. #define DBUS_RECEIVER_BUS_NAME    "com.ty3219.receiver_app"
15. #define DBUS_RECEIVER_PATH        "/com/ty3219/object"
16. #define DBUS_RECEIVER_INTERFACE   "com.ty3219.interface"
17. #define DBUS_RECEIVER_SIGNAL      "signal"
18. #define DBUS_RECEIVER_METHOD      "method"
19.
20. #define DBUS_RECEIVER_SIGNAL_RULE "type='signal',interface='%s'"
21. #define DBUS_RECEIVER_REPLY_STR   "i am %d, get a message"
22.
23. #define MODE_SIGNAL                1
24. #define MODE_METHOD                2
25.
26. #define DBUS_CLIENT_PID_FILE       "/tmp/dbus-client.pid"
27.
28. /**
29.  *
30.  * @param msg
31.  * @param conn
32.  */
33. void reply_method_call(DBusMessage *msg, DbusConnection *conn)
34. {
35.     DbusMessage *reply;
36.     DbusMessageIter reply_arg;
37.     DbusMessageIter msg_arg;
38.     dbus_uint32_t serial = 0;
39.
40.     pid_t pid;
41.     char reply_str[128];
42.     void *__value;
43.     char *__value_str;
44.     int __value_int;
45.
46.     int ret;
47.
48.     pid = getpid();
49.
50.     //创建返回消息reply
51.     reply = dbus_message_new_method_return(msg);
52.     if (!reply)
53.     {
54.         printf("Out of Memory!\n");
55.         return;
56.     }
57.
58.     //在返回消息中填入参数。
59.     snprintf(reply_str, sizeof(reply_str), DBUS_RECEIVER_REPLY_STR, pid);
60.     __value_str = reply_str;
61.     __value = &__value_str;
62.
63.     dbus_message_iter_init_append(reply, &reply_arg);
64.     if (!dbus_message_iter_append_basic(&reply_arg, DBUS_TYPE_STRING, __value))
65.     {
66.         printf("Out of Memory!\n");
67.         goto out;
```

```

68.     }
69.
70.     //从msg中读取参数, 根据传入参数增加返回参数
71.     if (!dbus_message_iter_init(msg, &msg_arg))
72.     {
73.         printf("Message has NO Argument\n");
74.         goto out;
75.     }
76.
77.     do
78.     {
79.         int ret = dbus_message_iter_get_arg_type(&msg_arg);
80.         if (DBUS_TYPE_STRING == ret)
81.         {
82.             dbus_message_iter_get_basic(&msg_arg, &__value_str);
83.             printf("I am %d, get Method Argument STRING: %s\n", pid,
84.                 __value_str);
85.
86.             __value = &__value_str;
87.             if (!dbus_message_iter_append_basic(&reply_arg,
88.                 DBUS_TYPE_STRING, __value))
89.             {
90.                 printf("Out of Memory!\n");
91.                 goto out;
92.             }
93.         }
94.         else if (DBUS_TYPE_INT32 == ret)
95.         {
96.             dbus_message_iter_get_basic(&msg_arg, &__value_int);
97.             printf("I am %d, get Method Argument INT32: %d\n", pid,
98.                 __value_int);
99.
100.            __value_int++;
101.            __value = &__value_int;
102.            if (!dbus_message_iter_append_basic(&reply_arg,
103.                DBUS_TYPE_INT32, __value))
104.            {
105.                printf("Out of Memory!\n");
106.                goto out;
107.            }
108.        }
109.        else
110.        {
111.            printf("Argument Type ERROR\n");
112.        }
113.    } while (dbus_message_iter_next(&msg_arg));
114.
115.    //发送返回消息
116.    if (!dbus_connection_send(conn, reply, &serial))
117.    {
118.        printf("Out of Memory\n");
119.        goto out;
120.    }
121.
122.    dbus_connection_flush(conn);
123.
124.    out:
125.    dbus_message_unref(reply);
126. }
127.
128. /* 监听D-Bus消息, 我们在上次的例子中进行修改 */
129. void dbus_receive(void)
130. {
131.     DBusMessage *msg;
132.     DBusMessageIter arg;
133.     DBusConnection *connection;
134.     DBusError err;
135.
136.     pid_t pid;
137.     char name[64];
138.     char rule[128];
139.

```

```
140. const char *path;
141. void *__value;
142. char *__value_str;
143. int __value_int;
144.
145. int ret;
146.
147. pid = getpid();
148.
149. dbus_error_init(&err);
150. //创建于session D-Bus的连接
151. connection = dbus_bus_get(DBUS_BUS_SESSION, &err);
152. if (!connection)
153. {
154.     if (dbus_error_is_set(&err))
155.         printf("Connection Error %s\n", err.message);
156.
157.     goto out;
158. }
159.
160. //设置一个BUS name
161. if (0 == access(DBUS_CLIENT_PID_FILE, F_OK))
162.     snprintf(name, sizeof(name), "%s%d", DBUS_RECEIVER_BUS_NAME, pid);
163. else
164.     snprintf(name, sizeof(name), "%s", DBUS_RECEIVER_BUS_NAME);
165.
166. printf("i am a receiver, PID = %d, name = %s\n", pid, name);
167.
168. ret = dbus_bus_request_name(connection, name,
169.                             DBUS_NAME_FLAG_REPLACE_EXISTING, &err);
170. if (ret != DBUS_REQUEST_NAME_REPLY_PRIMARY_OWNER)
171. {
172.     if (dbus_error_is_set(&err))
173.         printf("Name Error %s\n", err.message);
174.
175.     goto out;
176. }
177.
178. //要求监听某个signal: 来自接口test.signal.Type的信号
179. snprintf(rule, sizeof(rule), DBUS_RECEIVER_SIGNAL_RULE, DBUS_RECEIVER_INTERFACE);
180. dbus_bus_add_match(connection, rule, &err);
181. dbus_connection_flush(connection);
182. if (dbus_error_is_set(&err))
183. {
184.     printf("Match Error %s\n", err.message);
185.     goto out;
186. }
187.
188. while (1)
189. {
190.     dbus_connection_read_write(connection, 0);
191.
192.     msg = dbus_connection_pop_message(connection);
193.     if (msg == NULL)
194.     {
195.         sleep(1);
196.         continue;
197.     }
198.
199.     path = dbus_message_get_path(msg);
200.     if (strcmp(path, DBUS_RECEIVER_PATH))
201.     {
202.         printf("Wrong PATH: %s\n", path);
203.         goto next;
204.     }
205.
206.     printf("Get a Message\n");
207.     if (dbus_message_is_signal(msg, DBUS_RECEIVER_INTERFACE, DBUS_RECEIVER_SIGNAL))
208.     {
209.         printf("Someone Send me a Signal\n");
210.         if (!dbus_message_iter_init(msg, &arg))
211.         {
```

```

212.         printf("Message Has no Argument\n");
213.         goto next;
214.     }
215.
216.     ret = dbus_message_iter_get_arg_type(&arg);
217.     if (DBUS_TYPE_STRING == ret)
218.     {
219.         dbus_message_iter_get_basic(&arg, &__value_str);
220.         printf("I am %d, Got Signal with STRING: %s\n",
221.             pid, __value_str);
222.     }
223.     else if (DBUS_TYPE_INT32 == ret)
224.     {
225.         dbus_message_iter_get_basic(&arg, &__value_int);
226.         printf("I am %d, Got Signal with INT32: %d\n",
227.             pid, __value_int);
228.     }
229.     else
230.     {
231.         printf("Argument Type ERROR\n");
232.         goto next;
233.     }
234. }
235. else if (dbus_message_is_method_call(msg, DBUS_RECEIVER_INTERFACE, DBUS_RECEIVER_METHOD))
236. {
237.     printf("Someone Call My Method\n");
238.     reply_method_call(msg, connection);
239. }
240. else
241. {
242.     printf("NOT a Signal OR a Method\n");
243. }
244. next:
245.     dbus_message_unref(msg);
246. }
247.
248. out:
249.     dbus_error_free(&err);
250. }
251.
252. /*
253.  * call a method
254.  */
255. static void dbus_send(int mode, char *type, void *value)
256. {
257.     DBusConnection *connection;
258.     DBusError err;
259.     DBusMessage *msg;
260.     DBusMessageIter arg;
261.     DBusPendingCall *pending;
262.     dbus_uint32_t serial;
263.
264.     int __type;
265.     void *__value;
266.     char *__value_str;
267.     int __value_int;
268.     pid_t pid;
269.     int ret;
270.
271.     pid = getpid();
272.
273.     //Step 1: connecting session bus
274.     /* initialise the erroes */
275.     dbus_error_init(&err);
276.
277.     /* Connect to Bus*/
278.     connection = dbus_bus_get(DBUS_BUS_SESSION, &err);
279.     if (!connection)
280.     {
281.         if (dbus_error_is_set(&err))
282.             printf("Connection Err : %s\n", err.message);
283.

```

```

284.     goto out1;
285. }
286.
287. //step 2: 设置BUS name, 也即连接的名字。
288. ret = dbus_bus_request_name(connection, DBUS_SENDER_BUS_NAME,
289.                               DBUS_NAME_FLAG_REPLACE_EXISTING, &err);
290. if (ret != DBUS_REQUEST_NAME_REPLY_PRIMARY_OWNER)
291. {
292.     if (dbus_error_is_set(&err))
293.         printf("Name Err : %s\n", err.message);
294.
295.     goto out1;
296. }
297.
298.
299. if (!strcasecmp(type, "STRING"))
300. {
301.     __type = DBUS_TYPE_STRING;
302.     __value_str = value;
303.     __value = &__value_str;
304. }
305. else if (!strcasecmp(type, "INT32"))
306. {
307.     __type = DBUS_TYPE_INT32;
308.     __value_int = atoi(value);
309.     __value = &__value_int;
310. }
311. else
312. {
313.     printf("Wrong Argument Type\n");
314.     goto out1;
315. }
316.
317.
318. if (mode == MODE_METHOD)
319. {
320.     printf("Call app[bus_name]=%s, object[path]=%s, interface=%s, method=%s\n",
321.           DBUS_RECEIVER_BUS_NAME, DBUS_RECEIVER_PATH,
322.           DBUS_RECEIVER_INTERFACE, DBUS_RECEIVER_METHOD);
323.
324.     //针对目的地地址, 创建一个method call消息。
325.     //Constructs a new message to invoke a method on a remote object.
326.     msg = dbus_message_new_method_call(
327.         DBUS_RECEIVER_BUS_NAME, DBUS_RECEIVER_PATH,
328.         DBUS_RECEIVER_INTERFACE, DBUS_RECEIVER_METHOD);
329.     if (msg == NULL)
330.     {
331.         printf("Message NULL");
332.         goto out1;
333.     }
334.
335.     dbus_message_iter_init_append(msg, &arg);
336.     if (!dbus_message_iter_append_basic(&arg, __type, __value))
337.     {
338.         printf("Out of Memory!");
339.         goto out2;
340.     }
341.
342.     //发送消息并获得reply的handle 。
343.     //Queues a message to send, as with dbus_connection_send(), but also returns a DbusPendingCall i
344.     if (!dbus_connection_send_with_reply(connection, msg, &pending, -1))
345.     {
346.         printf("Out of Memory!");
347.         goto out2;
348.     }
349.
350.     if (pending == NULL)
351.     {
352.         printf("Pending Call NULL: connection is disconnected ");
353.         goto out2;
354.     }

```

```

355.     dbus_connection_flush(connection);
356.     dbus_message_unref(msg);
357.
358.     //waiting a reply, 在发送的时候, 已经获取了method reply的handle, 类型为
DBusPendingCall.
359.     // block until we receive a reply. Block until the pending call is completed.
360.     dbus_pending_call_block(pending);
361.     // get the reply message.
Gets the reply, or returns NULL if none has been received yet.
362.     msg = dbus_pending_call_steal_reply(pending);
363.     if (msg == NULL)
364.     {
365.         printf("Reply Null\n");
366.         goto out1;
367.     }
368.
369.     // free the pending message handle
370.     dbus_pending_call_unref(pending);
371.
372.     // read the Arguments
373.     if (!dbus_message_iter_init(msg, &arg))
374.     {
375.         printf("Message has no Argument!\n");
376.         goto out2;
377.     }
378.
379.     do
380.     {
381.         int ret = dbus_message_iter_get_arg_type(&arg);
382.         if (DBUS_TYPE_STRING == ret)
383.         {
384.             dbus_message_iter_get_basic(&arg, &__value_str);
385.             printf("I am %d, get Method return STRING: %s\n", pid,
386.                 __value_str);
387.         }
388.         else if (DBUS_TYPE_INT32 == ret)
389.         {
390.             dbus_message_iter_get_basic(&arg, &__value_int);
391.             printf("I am %d, get Method return INT32: %d\n", pid,
392.                 __value_int);
393.         }
394.         else
395.         {
396.             printf("Argument Type ERROR\n");
397.         }
398.
399.     } while (dbus_message_iter_next(&arg));
400.
401.     printf("NO More Argument\n");
402. }
403. else if (mode == MODE_SIGNAL)
404. {
405.     printf("Signal to object[path]=%s, interface=%s, signal=%s\n",
406.         DBUS_RECEIVER_PATH, DBUS_RECEIVER_INTERFACE, DBUS_RECEIVER_SIGNAL);
407.
408.     //步骤3:发送一个信号
409.     //根据图, 我们给出这个信号的路径(即可以指向对象), 接口, 以及信号名, 创建一个Message
410.     msg = dbus_message_new_signal(DBUS_RECEIVER_PATH,
411.         DBUS_RECEIVER_INTERFACE, DBUS_RECEIVER_SIGNAL);
412.     if (!msg)
413.     {
414.         printf("Message NULL\n");
415.         goto out1;
416.     }
417.
418.     dbus_message_iter_init_append(msg, &arg);
419.     if (!dbus_message_iter_append_basic(&arg, __type, __value))
420.     {
421.         printf("Out of Memory!");
422.         goto out2;
423.     }
424.

```



```

425. //将信号从连接中发送
426. if (!dbus_connection_send(connection, msg, &serial))
427. {
428.     printf("Out of Memory!\n");
429.     goto out2;
430. }
431.
432.     dbus_connection_flush(connection);
433.     printf("Signal Send\n");
434. }
435.
436. out2:
437.     dbus_message_unref(msg);
438. out1:
439.     dbus_error_free(&err);
440. }
441.
442. static void usage(void)
443. {
444.     #define USAGE "usage: ./dbus-client [send | receive] <param>\n" \
445.         "\treceive -- listen, wait a signal or a method call\n" \
446.         "\t\tif you want to test signal broadcast, run two receiver like this:\n" \
447.         "\t\ttrm -f /tmp/dbus-client.pid\n" \
448.         "\t\t./dbus-client receive &\n" \
449.         "\t\ttecho > /tmp/dbus-client.pid\n" \
450.         "\t\t./dbus-client receive &\n" \
451.         "\tsend [mode] [type] [value] -- send a signal or call a method\n" \
452.         "\t\tmode -- SIGNAL | METHOD\n" \
453.         "\t\ttype -- STRING | INT32\n" \
454.         "\t\tvalue -- string or number\n" \
455.         "\t\texample:\n" \
456.         "\t\t./dbus-client send SIGNAL STRING hello\n" \
457.         "\t\t./dbus-client send METHOD INT32 99\n" \
458.         "\n"
459.     printf(USAGE);
460. }
461.
462. int main(int argc, char *argv[])
463. {
464.     if (argc < 2)
465.     {
466.         usage();
467.         return -1;
468.     }
469.
470.     if (!strcmp(argv[1], "receive"))
471.     {
472.         dbus_receive();
473.     }
474.     else if (!strcmp(argv[1], "send"))
475.     {
476.         if (argc < 5)
477.         {
478.             usage();
479.         }
480.         else
481.         {
482.             if (!strcasecmp(argv[2], "SIGNAL"))
483.                 dbus_send(MODE_SIGNAL, argv[3], argv[4]);
484.             else if (!strcasecmp(argv[2], "METHOD"))
485.                 dbus_send(MODE_METHOD, argv[3], argv[4]);
486.             else
487.                 usage();
488.         }
489.     }
490.     else
491.     {
492.         usage();
493.     }
494.
495.     return 0;
496. }

```

### 三、运行

想要运行上面的例子，还需要一些步骤。

#### (1) 运行dbus-daemon

dbus-daemon的运行需要一个配置文件，这个配置文件稍微有点复杂，这里提供一个最简单的，无任何权限检查的例子debug-allow-all.conf

```
[html]
01. <!-- Bus that listens on a debug pipe and doesn't create any restrictions -->
02.
03. <!DOCTYPE busconfig PUBLIC "-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
04. "http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
05. <busconfig>
06.   <type>session</type>
07.
08.   <listen>unix:tmpdir=/tmp</listen>
09.
10.   <standard_session_servicedirs />
11.
12.   <policy context="default">
13.     <!-- Allow everything to be sent -->
14.     <allow send_destination="*" eavesdrop="true"/>
15.     <!-- Allow everything to be received -->
16.     <allow eavesdrop="true"/>
17.     <!-- Allow anyone to own anything -->
18.     <allow own="*" />
19.     <allow user="*" />
20.   </policy>
21.
22. </busconfig>
```

执行下面的命令

```
[plain]
01. ./dbus-daemon --config-file=/path/to/debug-allow-all.conf --fork --print-address
```

此时，dbus-daemon就会打印出一句类似这样的话

```
unix:path=/tmp/dbus-UXeqD3TJHE,guid=88e7712c8a5775ab4599725500000051
```

其实这个就是dbus-daemon的地址，我们需要把这个地址设置到环境变量里面，当你运行app的时候，libdbus.so就会读取这个环境变量，然后连接到这个dbus-daemon上。

设置环境变量

```
[plain]
01. export DBUS_SESSION_BUS_ADDRESS=unix:path=/tmp/dbus-
    UXeqD3TJHE,guid=88e7712c8a5775ab4599725500000051
```

#### (2) 这个时候你就可以运行上面例子编译出来的程序

```
[plain]
01. ./dbus-app
```

此时，会打印出一些参数信息。这个例子程序其实既可收也可以发，作为接收方时运行

```
[plain]
01. ./dbus-app receive &
```

可以运行多个dbus-app作为接收方，这样测试signal时就可以看到多个dbus-app同时受到这个signal了。  
作为发送方时，发送signal

```
[plain]
01. ./dbus-app send SIGNAL STRING hello
```

作为发送方时，调用method

```
[plain]
01. /dbus-app send METHOD INT32 30
```

至此，一个dbus的例子就可以运行起来了，想详细了解这个例子需要自己去看例子的源码。

顶

1

踩

0

上一篇 [glib交叉编译](#)

下一篇 [ubuntu 12.04 amd显卡驱动安装](#)

我的同类文章

Linux编程（3）

• [glib交叉编译](#)

2015-07-18 阅读 818

• [C程序自动构建工具](#)

2015-05-10 阅读 219

• [Linux 下系统调用的三种方法](#)

2014-07-09 阅读 133

参考知识库



**.NET**知识库

870 关注 | 635 收录

猜你在找

- Apple Watch开发入门
- Maemo Linux手机平台系列分析7 Maemo平台开发之
- 征服JavaScript高级程序设计与应用实例视频课程
- 浅析 Linux 初始化 init 系统第 2 部分 UpStart
- HTML5开发手机App之：CSS全教程
- Linux系统init过程之 UpStart
- 基于WebRTC的跨平台实时语音通信解决方案
- 浅析 Linux 初始化 init 系统第 2 部分 UpStart
- Part 17: Cocos2d-x开发实战-基于Node.js的Socket.
- 浅析 Linux 初始化 init 系统第 2 部分 UpStart



PLAY NOW

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