摘要： openwrt中使用ubus实现进程通信ubus为openwrt平台开发中的进程间通信提供了一个通用的框架。它让进程间通信的实现变得非常简单，并且ubus具有很强的可移植性，可以很方便

ubus为openwrt平台开发中的进程间通信提供了一个通用的框架。它让进程间通信的实现变得非常简单，并且ubus具有很强的可移植性，可以很方便的移植到其他linux平台上使用。本文描述了ubus的实现原理和整体框架。

ubus源码可通过git库 git://nbd.name/luci2/ubus.git 获得，其依赖的ubox库的git库：

git://nbd.name/luci2/ubox.git。

1. ubus的实现框架

ubus实现的基础是unix socket，即本地socket，它相对于用于网络通信的inet socket更高效，更具可靠性。unix socket客户端和服务器的实现方式和网络socket类似，读者如果还不太熟悉可查阅相关资料。

我们知道实现一个简单的unix socket服务器和客户端需要做如下工作：

1. 建立一个socket server端，绑定到一个本地socket文件，并监听clients的连接。
2. 建立一个或多个socket client端，连接server。
3. client和server相互发送消息。
4. client或server收到对方消息后，针对具体消息进行相应处理。

ubus同样实现了上述组件，并对socket连接以及消息传输和处理进行了封装：

1. ubus提供了一个socket server：**ubusd**。因此开发者不需要自己实现server端。
2. ubus提供了创建socket client端的接口，并且提供了三种现成的客户端供用户直接使用：

    1) 为shell脚本提供的client端。
    2) 为lua脚本提供的client接口。
    3) 为C语言提供的client接口。

    可见ubus对shell和lua增加了支持，后面会介绍这些客户端的用法。

3. ubus对client和server之间通信的消息格式进行了定义：client和server都必须将消息封装成**json**消息格式。

4. ubus对client端的消息处理抽象出“对象（**object**）”和“方法（**method**）”的概念。一个对象中包含多个方法，client需要向server注册收到特定json消息时的处理方法。对象和方法都有自己的名字，发送请求方只需在消息中指定要调用的对象和方法的名字即可。

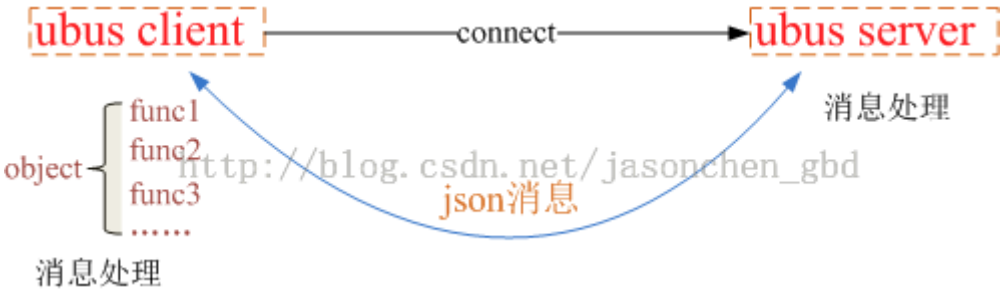
使用ubus时需要引用一些动态库，主要包括：

libubus.so：ubus向外部提供的编程接口，例如创建socket，进行监听和连接，发送消息等接口函数。

libubox.so：ubus向外部提供的编程接口，例如等待和读取消息。

libblobmsg.so，libjson.so：提供了封装和解析json数据的接口，编程时不需要直接使用libjson.so，而是使用libblobmsg.so提供的更灵活的接口函数。

ubus中各组件的关系如下图所示：

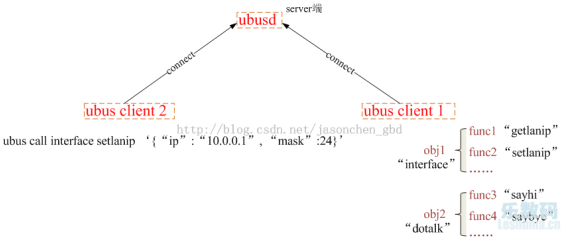


使用ubus进行进程间通信不需要编写大量代码，只需按照固定模式调用ubus提供的API即可。在ubus源码中examples目录下有一些例子可以参考。

2. ubus的实现原理

下面以一个例子说明ubus的工作原理：

下图中，client2试图通过ubus修改ip地址，而修改ip地址的函数在client1中定义。



client2进行请求的整个过程为：

1. client1向ubusd注册了两个对象："interface"和"dotalk"，其中"interface"对象中注册了两个 method："getlanip"和"setlanip"，对应的处理函数分别为func1()和func2()。"dotalk"对象中注册了两个 method："sayhi"和"saybye"，对应的处理函数分别为func3()和func4()。
2. 接着创建一个client2用来与client1通信，注意，两个client之间不能直接通信，需要经ubusd（server）中转。
3. client2就是在前面讲到的shell/lua/C客户端。假设这里使用shell客户端，在终端输入以下命令：

   ubus call interface setlanip '{"ip":"10.0.0.1", "mask":24}'

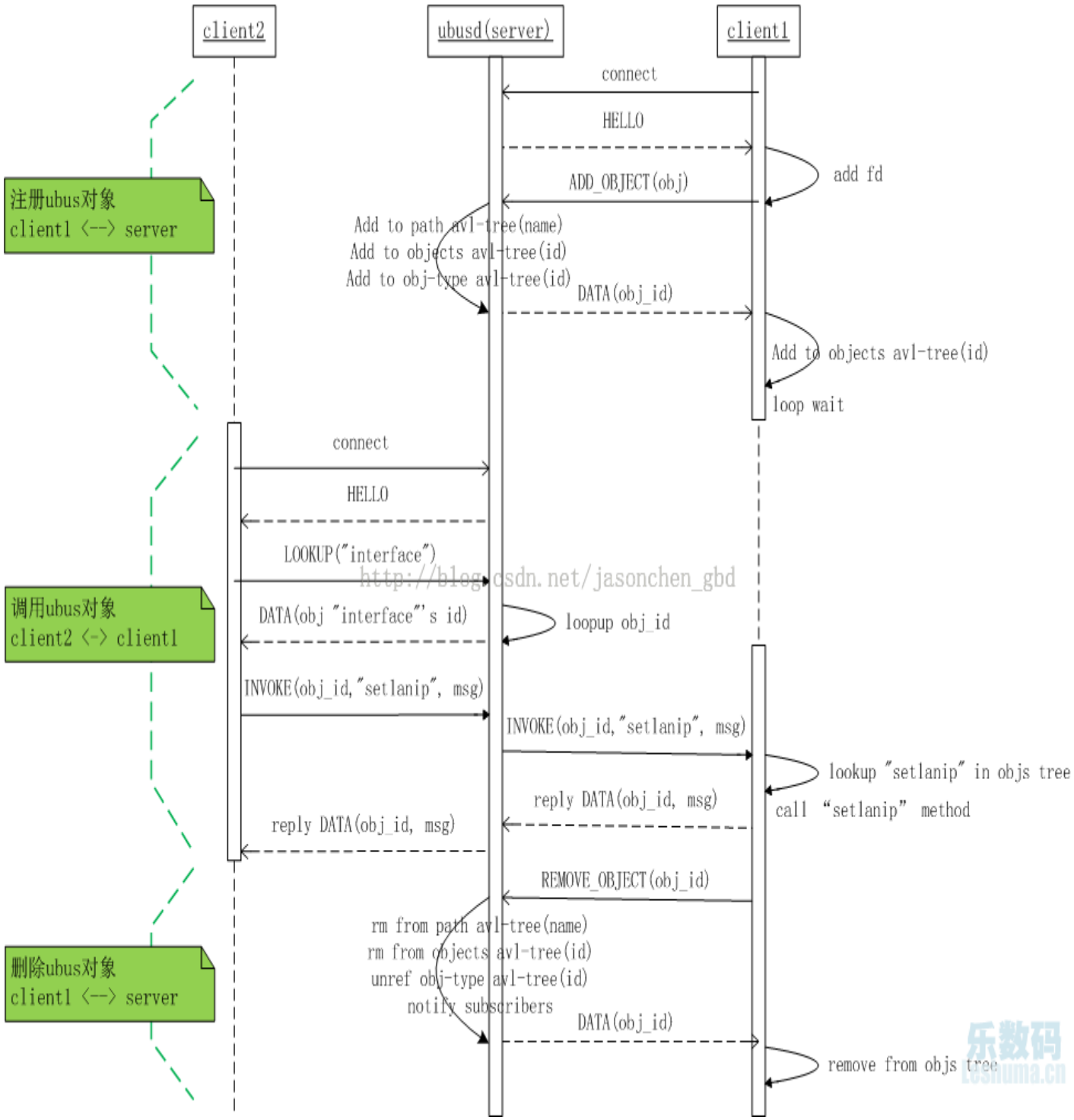
   ubus的call命令带三个参数：请求的对象名，需要调用的方法名，要传给方法的参数。

4. 消息发到server后，server根据对象名找到应该将请求转发给client1，然后将消息发送到client1，client1进而调用func2()接受参数并处理，如果处理完成后需要回复client2，则发送回复消息。

接下来介绍一下上述过程中，ubus内部的处理机制，虽然使用ubus进行进程间通信不需要关注这些实现细节，但有助于加深对ubus实现原理的理解。

下图中，client1注册对象和方法，其实可认为是服务提供端，只不过对于ubusd来讲是一个socket client。

client2去调用client1注册的方法。



3. ubus的应用场景和局限性

ubus可用于两个进程之间的通信，并以类似json格式进行数据交互。ubus的常见场景为：

“客户端--服务器”形式的交互，即进程A注册一系列的服务，进程B去调用这些服务。

ubus支持以“订阅 -- 通知”的方式进行进程通信，即进程A提供订阅服务，其他进程可以选择订阅或退订该服务，进程A可以向所有订阅者发送消息。

由于ubus实现方式的限制，在一些场景中不适宜使用ubus：

1. ubus用于少量数据的传输，如果数据量很大或是数据交互很频繁，则不宜用ubus。经过测试，当ubus一次传输数据量超过60KB，就不能正常工作了。

2. ubus对多线程支持的不好，例如在多个线程中去请求同一个服务，就有可能出现不可预知的结果。
3. 不建议递归调用ubus，例如进程A去调用进程B的服务，而B的该服务需要调用进程C的服务，之后C将结果返回给B，然后B将结果返回给A。如果不得不这样做，需要在调用过程中避免全局变量的重用问题。

4. ubus源码简析

下面介绍一下ubusd和ubus client工作时的代码流程，这里为了便于理解，只介绍大致的流程，欲了解详细的实现请读者自行阅读源码。

4.1 ubusd工作流程

ubusd 的初始化所做的工作如下：

1. epoll_create(32)创建出一个poll_fd。

2. 创建一个UDP unix socket，并添加到poll_fd的监听队列。

3. 进行epoll_wait()等待消息。收到消息后的处理函数定义如下：

```
1. static struct uloop_fd server_fd = {
2. .cb = server_cb,
3. };
```

即调用server_cb()函数。

4. server_cb()函数中的工作为：

(1)进行accept()，接受client连接，并为该连接生成一个client_fd。

(2)为client分配一个client id，用于ubusd区分不同的client。

(3)向client发送一个HELLO消息作为连接建立的标志。

(4)将client_fd添加到poll_fd的监听队列中，用于监听client发过来的消息，消息处理函数为client_cb()。

也就是说ubusd监听两种消息，一种是新client的连接请求，一种是现有的每个client发过来的数据。

当**ubusd**收到一个**client**的数据后，调用**client_cb()**函数的处理过程：

1. 先检查一下是否有需要向这个client回复的数据（可能是上一次请求没处理完），如果有，先发送这些遗留数据。

2. 读取socket上的数据，根据消息类型（数据中都指定了消息类型的）调用相应的处理函数，消息类型和处理函数定义如下：

```
1. static const ubus_cmd_cb handlers[__UBUS_MSG_LAST] = {
2. [UBUS_MSG_PING] = ubusd_send_pong,
3. [UBUS_MSG_ADD_OBJECT] = ubusd_handle_add_object,
4. [UBUS_MSG_REMOVE_OBJECT] = ubusd_handle_remove_object,
5. [UBUS_MSG_LOOKUP] = ubusd_handle_lookup,
6. [UBUS_MSG_INVOKE] = ubusd_handle_invoke,
7. [UBUS_MSG_STATUS] = ubusd_handle_response,
8. [UBUS_MSG_DATA] = ubusd_handle_response,
9. [UBUS_MSG_SUBSCRIBE] = ubusd_handle_add_watch,
10. [UBUS_MSG_UNSUBSCRIBE] = ubusd_handle_remove_watch,
11. [UBUS_MSG_NOTIFY] = ubusd_handle_notify,
12. };
```

例如，如果收到invoke消息，就调用ubusd_handle_invoke()函数处理。

这些处理函数可能是ubusd处理完后需要回发给client数据，或者是将消息转发给另一个client（如果发送请求的client需要和另一个client进行通信）。

3. 处理完成后，向client发送处理结果，例如UBUS_STATUS_OK。（注意，client发送数据是UBUS_MSG_DATA类型的）

4.2 client的工作流程

ubus call obj method的工作流程：

1. 创建一个unix socket(UDP)连接ubusd，并接收到server发过来的HELLO消息。

2. ubus call命令由ubus_cli_call()函数进行处理，先向ubusd发送lookup消息请求obj的id。然后向ubusd发送invoke消息来调用obj的method方法。

3. 创建epoll_fd并将client的fd添加到监听列表中等待消息。

4. client收到消息后的处理函数为ubus_handle_data()，其中UBUS_MSG_DATA类型的数据receive_call_result_data()函数协助解析。

被**call**的**client**的工作流程：

和ubus客户端的流程相似，只是变成了接受请求并调用处理函数。