# **PJSIP开发手册之总体设计（一）**

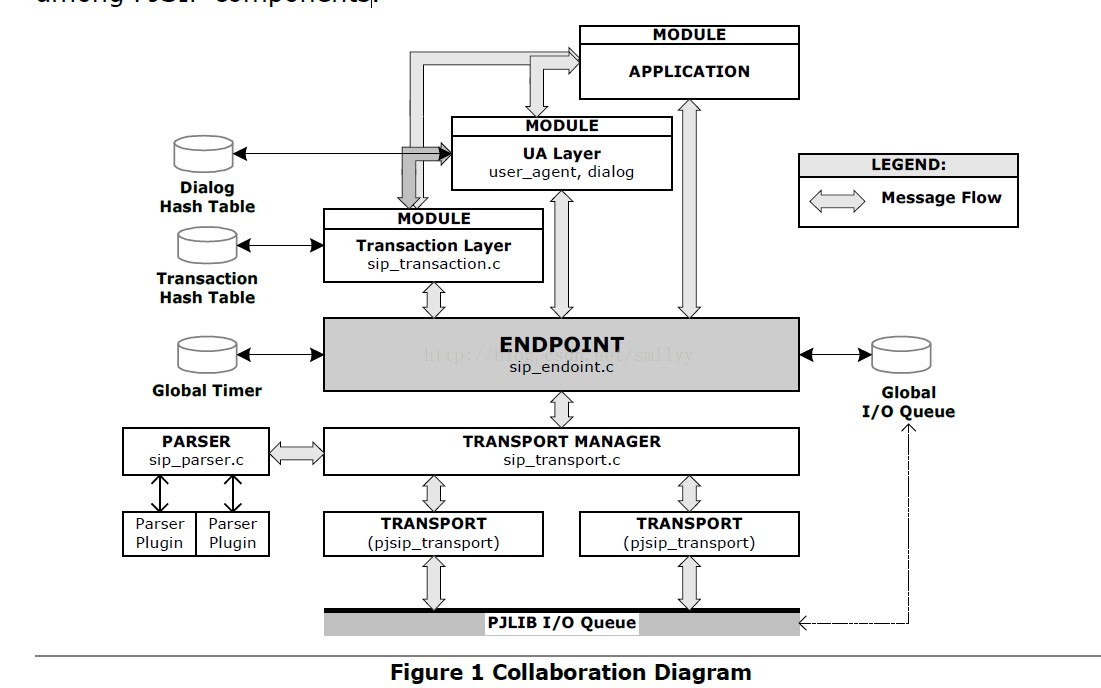
PJSIP是用C语言写的。

****第一章  总体设计****

****结构****

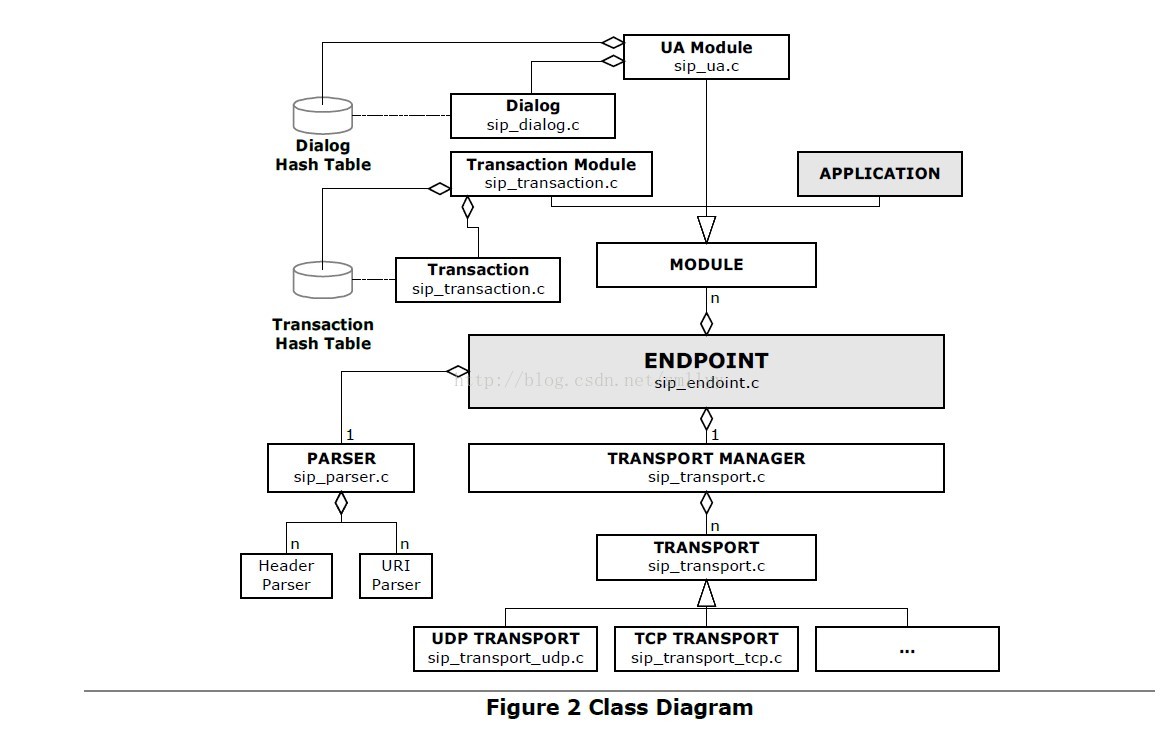
****通信图****

下面的图展示了SIP消息是如何在SIP组件之间传递的。

IMG_257

****类图****

下面的图展示的是PJSIP的类

IMG_259

****EndPoint****

EndPoint是SIP栈的核心，封装在pjsip\_endpoint数据结构中。Endpoint有以下属性和职责：

它有内存池工厂，负责给所有的SIP组件分配内存池。

它有定时器堆实例，为所有的SIP组件调度定时器

它有SIP传输管理实例，传输管理有SIP传输和控制消息的解析和打印

它拥有PJLIB io队列的唯一实例。Io队列是用来分发网络事件

它提供线程安全的轮询功能，这样的应用中的线程可以轮询定时器和socket事件（PJSIP本身不创建任何线程）

它管理模块，PJSIP模块是扩展除了消息解析和传输之外协议栈功能的主要方法

它负责从Transport Manager那里接收SIP消息，再分发给各个模块

****内存池分配和释放****

Endpoint负责所有的SIP组件的内存分配，为了保证整个应用的线程安全和策略的一致性。比如内存池缓存策略，就是把不使用的内存池缓存下来为将来的使用，而不是释放它

Endpoint提供分配和释放内存池的函数：

pjsip\_endpt\_create\_pool()

pjsip\_endpt\_release\_pool()

当endpoint被创建（pjsip\_endpt\_create()）之后，应用为endpoint指定内存池工厂。Endpoint在它整个生命周期内都将保管这个内存池工厂的指针，并且使用它去创建和释放内存池。

****定时器管理****

Endpoint拥有单一一个定时器堆实例，去管理定时器，完成所有定时器的创建和调度。

Endpoint提供以下函数去管理定时器：

pjsip\_endpt\_schedule\_timer()

pjsip\_endpt\_cancel\_timer()

Endpoint在轮询函数被调用时，会检查定时器的期限。

****轮询栈****

Endpoint提供一个函数（pjsip\_endpt\_handle\_events()）去检查定时器超时和网络事件的发生。应用可以指定等待这些事件发生的最长时间。

PJSIP栈不会创建线程。所有栈的执行都会在应用的主线程中，不管是使用API调用还是应用调用轮询函数。

轮询功能是基于定时器堆的内容来有优化等待时间的。例如，它知道一个定时器将在下一个5ms超时，它等待socket事件的时间就不会超过5ms；这样就减少了当没有socket事件发生时，应用所需要的等待时间。定时器的精确度依平台而定。

****线程安全与线程并发****

****线程安全****

线程安全是个复杂的问题。但是，幸运的是，以下的设计原则在整个栈中都一致使用了：

对象一定要是线程安全的，但是数据结构不一定是线程安全的。

通常，对象和简单数据之间没有明显的区别。但是下面的例子可以让你了解它们之间的区别：

数据结构是：

* PJLIB的数据结构，像列表、数组、哈希表、字符串和内存池
* SIP消息元素，像URIs，头部域和SIP消息

数据结构都是线程不安全的，它们的线程安全是由包含它们的对象保证的。如果把数据结构也设置为线程安全的。我们称之为对象的有：

PJLIB对象，像io队列的，那么将会严重影响栈的性能和浪费操作系统的资源

* 然而，SIP对象必须是线程安全
* PJSIP对象，像endpoint，transaction，dialogs和dialogusages等等

****线程并发****

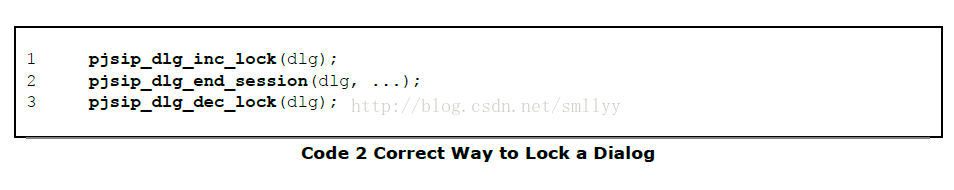
但是糟糕的是一些对象在头文件里暴露了它们的声明（如pjsip\_transaction和pjsip\_dialog）。虽然这些对象暴露的API是保证线程安全的。但是应用还是必须保证在访问这些数据结构之前获取到相应的锁，例如调用pj\_mutex\_lock()

更糟糕的是，一个dialog暴露了除了pj\_mutex\_lock()和pj\_mutex\_unlock()之外的不同的APi来锁一个dialog。应用应该调用pjsip\_dlg\_inc\_lock()和pjsip\_dlg\_dec\_lock()。这两种方法之前的区别，是dialog inc/dec锁方法在调用过程中并不会销毁dialog，不然dialog销毁会导致pj\_mutex\_unlock()崩溃。

考虑下面的例子：

IMG_261

上面的例子（假象的），应用执行到第3行可能会崩溃，由于pjsip\_dlg\_end\_session()在某种情况下，可能会销毁dialog，例如，当发出的初始的INVITE事务没有收到任何响应，因此事务将被销毁，同时dialog也将被销毁。Inc/dec方法通过临时增加会话计数来阻止dialog的销毁，因此在end\_session()方法中将不会销毁dialog。Dialog可能在dec\_lock()方法中被销毁。正确的顺序应该如下：

IMG_263

最后，真正糟糕的是，加锁必需在正确的顺序，否则将会造成死锁。例如，如果应用想要锁定同一个dialog中的dialog和一个事务，必须在获得transaction的锁之前先获得dialog的锁，否则当另一个线程以相反的顺序获取同一个dialog和transaction的锁时会导致死锁的发生。

****解决方法****

幸运的是，应用只有在极少的情况下需要直接获取对象的锁，所以上面的问题很少会发生。

应用应该使用对象的API去访问对象。APIs保证正确的执行顺序，避免死锁和崩溃。

回调函数通常都是在已经获取到对象的锁之后，才会被对象（transaction或dialog）调用。应用不用获取对象的锁，也可以安全地访问对象的数据结构。

# **PJSIP开发手册之模块（二）**

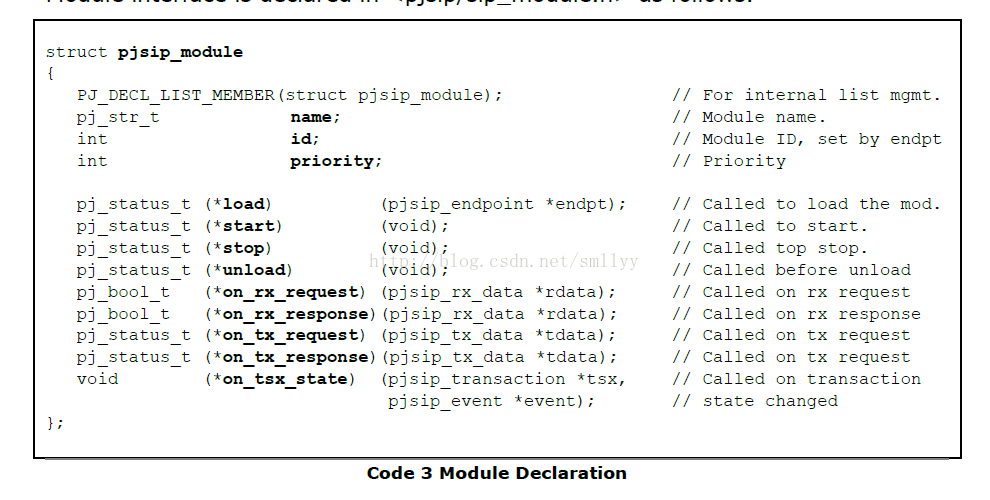
****第二章 模块****

模块框架是在PJSIP程序中各个软件组件之间派发SIP消息的主要方法。PJSIP中所有的软件组件，包括transaction层和dialog层，都是以模块实现的。如果没有模块，核心栈（pjsip\_endpoint和transport）就不知道如何处理SIP消息。

模块框架是基于一个简单但是功能强大的接口抽象。对于接收到的消息，Endpoint（pjsip\_endpoint）按照优先级向各个模块分发这个消息，直到其中一个模块处理了这个消息。对于发出的消息，Endpoint在电缆传输数据之前，会将这个消息分发到各个模块，允许它们对消息做最后的修改。

****模块声明****

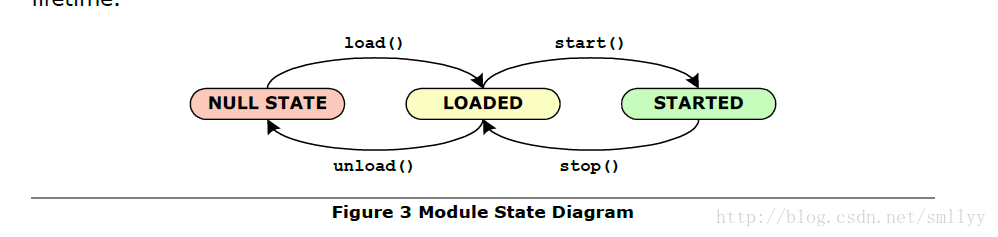
模块接口在<pjsip/sip\_module.h>中声明，如下：



所有的函数指针是可选的，如果它们没有被指定，它们将会被认为是成功返回的。

Load，start，stop，和unload这四个函数将会被Endpoint调用去控制模块状态。下图是模块状态的生命周期：

IMG_257



on\_rx\_request()和on\_rx\_response()函数指针是模块从Endpoint（pjsip\_endpt）或其他模块接收SIP消息的主要方法。这些回调函数的返回值很重要。如果一个回调函数返回非0（例如真的条件），表示这个模块已经处理了这个消息，这样，Endpoint就可以停止分发消息到其他模块。

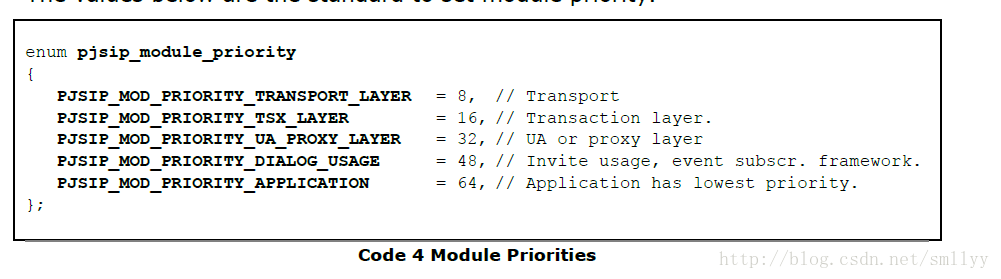
在一个消息被发送之前，on\_tx\_request()和on\_tx\_response()函数指针将会被传输管理调用。这就给了一些模块（比如sipcomp，message signing）机会去对消息做最后的修改。所有的模块必须返回PJ\_SUCCESS（例如0状态），否则传输将被取消。

on\_tsx\_state()函数指针用于接收由于接收到消息、发送消息、定时器事件或者网络错误事件引起的事务状态改变的通知。

****模块优先级****

模块的优先级决定了模块被调用来处理回调的顺序。高优先级的模块（例如，较小的优先级数）将会先调用它们的on\_rx\_request()和on\_rx\_response()，最后调用它们的on\_tx\_request()和on\_tx\_response()函数。

下面给出的是标准的模块优先级的值：（注意：数字越小优先级越高）

IMG_260

Transport Manager使用的是PJSIP\_MOD\_PRIORITY\_TRANSPORT\_LAYER优先级。这个优先级是用来控制消息的传输，例如，优先级比它低的模块（有更大的优先级数）的on\_tx\_request()和on\_tx\_response()函数将会在传输层处理消息之前调用（例如，目的地址的计算，在连续的缓存里打印消息等等）。而且高优先级的模块将会在传输层处理完成之后，才会调用回调函数。

Transaction层使用PJSIP\_MOD\_PRIORITY\_TSX\_LAYER优先级。Transaction层接收属于一个Transaction里的所有接收到的消息。

UA层（如dialog框架）或proxy层使用PJSIP\_MOD\_PRIORITY\_UA\_PROXY\_LAYER优先级。UA层吸收所有属于同一个dialog集的所有接收到的消息（这代表有分歧的响应）。

Dialog usages使用PJSIP\_MOD\_PRIORITY\_DIALOG\_USAGE优先级。目前PJSIP实现了两种dialog usages：INVITE会话和事件订阅会话（包括REFER订阅）。Dialog usage吸收属于一个特定的会话的同一个dialog的消息。

对于应用模块来说PJSIP\_MOD\_PRIORITY\_APPLICATION是一个合适的值，因为应用希望利用transactions、dialogs和dialog usages。

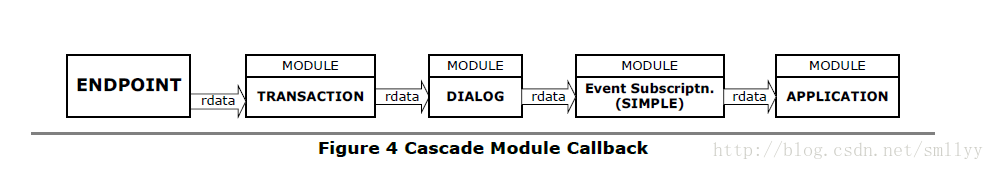
****到来消息的模块处理****

当消息到来时，它将被表示为接收消息缓存（struct pjsip\_rx\_data）。传输管理解析这个消息后，把解析后的数据结构存放在接收消息缓存中，并把消息传给Endpoint。

Endpoint通过调用on\_rx\_request()或on\_rx\_response()回调函数将接收消息缓存分发给各个已注册的模块。先分发到高优先级的模块，直到有个模块返回非零。当一个模块返回非零，Endpoint将停止分发消息到剩下的模块，假设这个模块已经开始处理这个消息。

回调函数返回非零的模块，自己可能将消息分发给其他模块。例如，transaction模块，当接收到匹配的消息后开始处理消息，然后把处理好的消息分发给transaction用户，transaction用户也是一个模块。Transaction通过调用transaction用户的回调函数on\_rx\_request()或on\_rx\_response()来将消息传给transaction用户模块。接收消息缓存区的事务字段将被设置为相应的值，用于区分事务内和事务外的消息。

下图显示模块如何连续调用其他模块：

IMG_262

****送出消息的处理****

送出消息被表示为一个传输数据缓存（pjsip\_tx\_data），它包括本身的消息结构，内存池和transport信息。

当pjsip\_transport\_send()被调用来发送一个消息，transport manager将会安装优先级从低到高开始调用所有模块的on\_tx\_request()或on\_tx\_response()。Transport层负责管理一个传输缓存内的信息：

* 传输信息
* 输出消息结构到连续的数据结构

优先级比PJSIP\_MOD\_PRIORITY\_TRANSPORT\_LAYER低的模块将在这些信息获取之前接收到消息。这意味着目的地址还没有计算出来，并且消息还没有打印到连续的缓存中。

如果模块要在消息输出到缓存之前修改消息结构，需要把模块的优先级设置比transport层高。如果模块想要得到真实在电缆上传输的字节包（例如，打印日志），可以把模块的优先级设置得比transport层高。

如果有一个模块的回调函数返回的不是PJ\_SUCCESS，那么传输将会被取消并且错误码将会返回给pjsip\_transport\_send()的调用者。

****Transaction用户和状态回调****

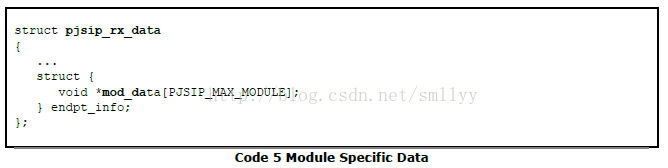
on\_tsx\_state()回调函数由于接收一个特定事务状态改变的通知。这个回调函数是唯一的，因为事务状态的改变可以又消息无关的事件引起（例如，定时器超时和transport错误）。

回调函数只有当模块在特定的事务中注册成为事物用户之后才会被调用。每个transaction只允许一个transaction用户。Transaction用户可以设置为每个transaction的基础。对于由dialog创建的事务，transaction用户设置为这个dialog的UA层模块。当应用创建transaction时，应用会把它们自己设置为transaction用户。

on\_tsx\_state()在接收到请求或响应重传时，不会被调用。注意传送或接收临时并不属于重传，因此传送或接收临时响应将会引起该回调函数。

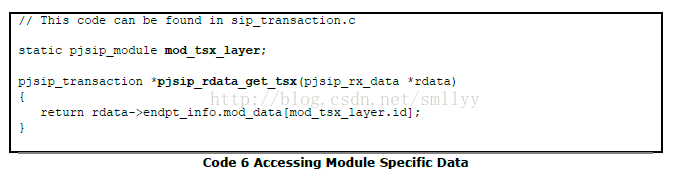
模块内特定数据

一些PJSIP组件有一个容器，可以用来存放模块的特定数据。这个容器叫做mod\_data，是一个void指针数组，用模块ID作为索引。例如，一个到来包缓存（pjsip\_rx\_data）有下面的模块特定容器的声明：



模块ID在模块注册到Endpoint时决定。

当一个到来包缓存（pjsip\_rx\_data）传递给模块时，该模块可以把特定的模块数据存放在mod\_data数组中对应的位置。之后，这些数据可以被模块或应用取出。例如，事务层会将匹配的事务实例存放到mod\_data中，UA层也会将匹配的dialog实例存放到mod\_data中。应用可以调用pjsip\_rdata\_get\_tsx()或者pjsip\_rdata\_get\_dlg()来获取这些数据，下面是一个简单数组查找：



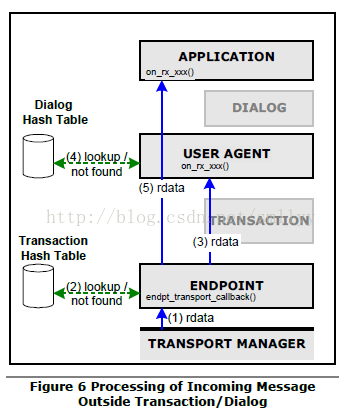
回调函数总结

下表总结了事件的发生以及它触发的特定的回调函数。但是，on\_tsx\_state()回调函数，只有在应用选择有状态地处理请求才会触发。

|  |  |  |
| --- | --- | --- |
| 事件 | on\_rx\_request()  或on\_rx\_response() | on\_tsx\_state() |
| 接收到新的请求或者响应 | 被调用 | 被调用 |
| 接收到请求或者响应的重传 | 只有当优先级数小于transaction层时才被调用 | 不被调用 |
| 发送一个新的请求或者响应 | 不被调用 | 被调用 |
| 重传一个请求或响应 | 不被调用 | 不被调用 |
| 传送超时 | 不被调用 | 被调用 |
| 其他的transaction失败事件（如DNS查询失败，transport失败） | 不被调用 | 被调用 |

回调图样例

事务和dialog外的到来消息



处理过程如下：

1) Transport Manager（pjsip\_tpmgr）将所有接收到的消息传给Endpoint（在消息解析之后）

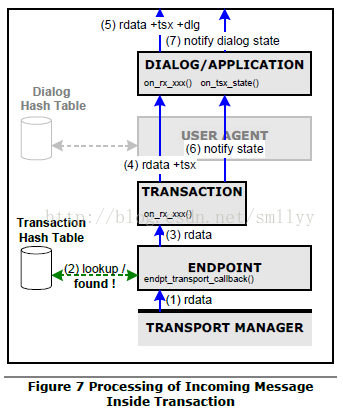
2) Endpoint将消息分发到所有已注册的回调。在回调表中的第一是transaction层。Transaction将在transaction表中查找消息，并且找不到一个匹配的事务

3) Endpoint分发消息到回调表中的下一个，也就是UA

4) UA在dialog哈希表中查找消息，并且找不到匹配的dialog

5) Endpoint将会继续将消息分发给下一个注册的回调直到找到应用。应用将会处理消息（如，无状态响应，创建UAS事务，代理请求，或创建dialog等）

事务内的到来消息



处理过程如下：

1) Transport Manager（pjsip\_tpmgr）将所有接收到的消息传给Endpoint（在消息解析之后）

2) Endpoint将消息分发到所有已注册的回调。在回调表中的第一是transaction层。Transaction将在transaction表中查找消息，并且找到一个匹配的事务

3) 因为transaction的回调返回PJ\_TRUE，Endpoint将不会再分发消息

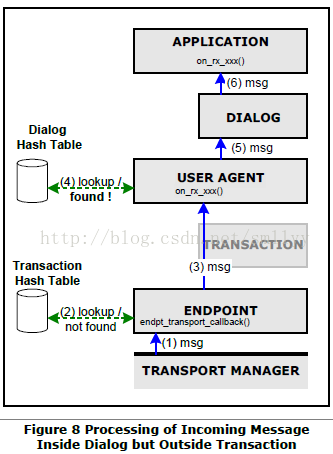
4) Transaction处理这个响应（如，更新FSM）。如果是重传消息，处理将停止在这步。否则，事务将把消息传给它的TU。TU可以是一个dialog或者一个应用

5) 如果TU是一个dialog，这个dialog将处理响应，并将响应传给它的DU（如应用）

6) 如果消息的到来改变了事务的状态，事务将通知它的TU关于这个新状态

7) 如果TU是一个dialog，它可能还要通知应用关于dialog的状态改变

Dialog内事务外的到来消息



1) Transport Manager（pjsip\_tpmgr）将所有接收到的消息传给Endpoint（在消息解析之后）

2) Endpoint将消息分发到所有已注册的回调。在回调表中的第一是transaction层。Transaction将在transaction表中查找消息，并且找不到一个匹配的事务

3) Endpoint分发消息到回调表中的下一个，也就是UA

4) UA在dialog哈希表中查找消息，并且找到一个匹配的dialog

5) UA模块将把消息传给对应的dialog

6) Dialog将为到来的请求创建事务，并通过调用dialog usage的on\_rx\_request()和on\_tsx\_state()函数，来将消息传给它们

模块管理

PJSIP的Endpoint（pjsip\_endpoint）管理模块。应用需要把它的每个模块注册到Endpoint上，便于协议栈来识别它们。

模块管理API

模块管理相关的API声明在<pjsip/sip\_endpt.h>中

IMG_268

向Endpoint注册一个模块。Endpoint将调用模块的load和start函数去正确地初始化模块，并给模块分配一个特定的模块ID。

IMG_269

向Endpoint注销一个模块。Endpoint将调用一个模块的stop和unload函数去关闭一个模块

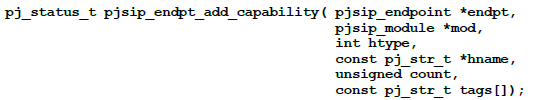
模块能力

模块可以向Endpoint声明一个新的能力。目前，Endpoint管理以下能力：

* 允许SIP方法（Allow头部域）
* 支持SIP扩展（Supported头部域）
* 支持的文本类型（Accepted头部域）

这些头部域将被自动地加到送出的请求和响应中。

一个模块通过调用pjsip\_endpt\_add\_capability()函数来声明一个新能力。



向Endpoint注册一个新功能。htype参数指定了向那个头部域中增加这个能力，可选的有PJSIP\_H\_ACCEPT，PJSIP\_H\_ALLOW，和PJSIP\_H\_SUPPORTED。hname参数是可选的，它只用来指出头部域中哪个能力是无法被核心协议栈所识别的。count和tags参数指定要加到头部域中的tag字符串数组。

IMG_271

获取一个能力头部域，其中包含了指定头部域向Endpoint已注册的所有能力。

# **PJSIP开发手册之消息元素（三）**

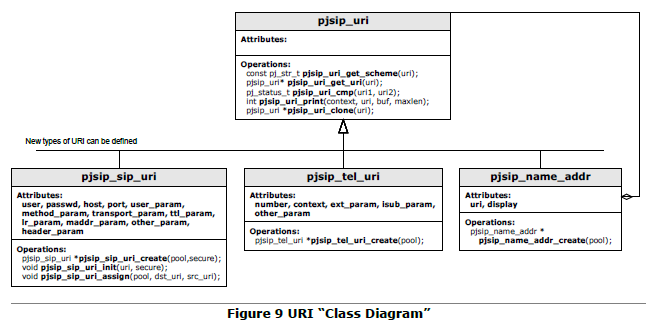
****第三章 消息元素****

****URI****

统一资源定位符（URI）几乎是仿照面向对象构造的（或者有人说是基于对象而不是面向对象的）。因此URI是可以被协议栈统一对待，而且新的URI类型也很容易引入。

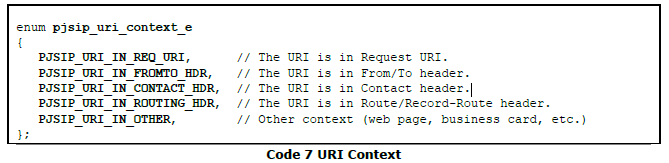
****URI**类图**

下图展示了URI对象的设计：



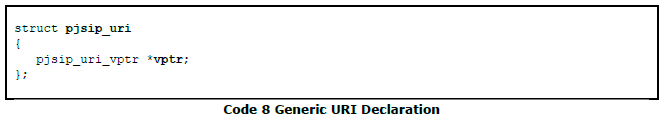
****URI上下文****

URI上下文指定了URI在哪使用（如在请求行，在From和To头部域等等）。上下文指定了此时什么URI元素是允许出现的。例如，transport参数不允许在From/To头部域出现等等。在PJSIP中，当要向一个缓存中输出URI时和比较两个URIs时，需要指定上下文。在这些情况下，在指定上下文中部允许出现的URI部分，将在输出和比较过程中忽略。

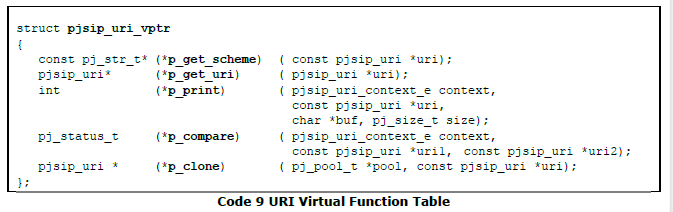


****基本URI****

pjsip\_uri数据结构中包含了所有类型URI共享的属性。因此，所有类型的URI都可以转化为pjsip\_uri，且可以统一的方式操作。



jsip\_uri\_vptr指定了虚函数表，其中的成员是由每一种类型的URI定义的。应用不推荐直接访问这些函数指针。应用推荐使用URI的API。因为它们可读性好，同时也减少一些输入。



下面的URI函数可以被所有类型的URI对象应用。这些函数通常实现为内联函数，这些内联函数调用URI虚函数表中对应的函数指针。

IMG_260

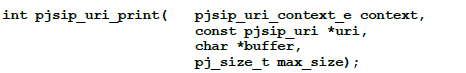
获取URI的机制字符串（如”sip”，”sips”，”tel”）。

IMG_261

获取URI对象。通常所有URI对象会返回它自身。除了命名地址（name  address），会返回命名地址对象内的URI。

IMG_262

根据指定的上下文比较uri1和uri2。指定上下文中不允许出现的参数将被忽略。如果URIs相同则返回PJ\_SUCCESS。



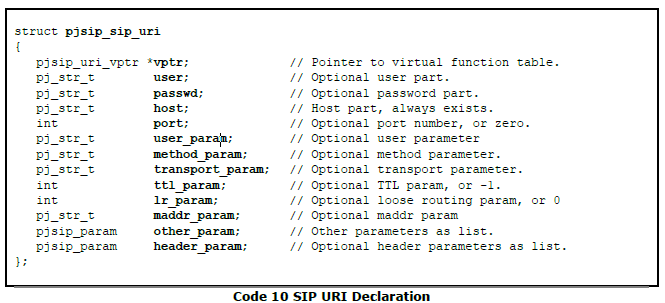
根据指定的上下文将uri输出到指定buffer。指定上下文中不允许出现的参数将不会被输出。

IMG_264

使用 指定的内存池，创建uri的深度拷贝。

****SIP URI**和SIPs URI**

pjsip\_sip\_uri代表SIP和SIPs URI。它声明在<pjsip/sip\_uri.h>中。



下面列出的函数是SIP/SIPS URI对象中特有的。除此之外，应用也可以使用基本URI中的函数来操作SIP/SIPS URI。

IMG_266

使用指定的内存池，创建一个新的SIP URI。如果secure标志不为零，则SIPS URI将会被创建。这个函数将会设置这个URL的vptr为SIP/SIPS vptr，并清空其他成员值。

IMG_267

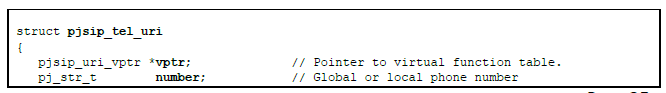
初始化一个SIP URI结构

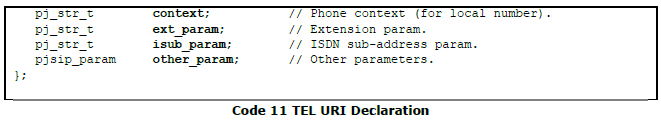
IMG_268

深度拷贝SIP/SIPS URI

****电话**URI**

pjsip\_tel\_uri结构代表tel URL，它声明在<pjsip/sip\_tel\_url.h>中。





下面的方法是TEL URI特有的。除此之外，应用也可以使用基本URI中的函数来操作SIP/SIPS URI。

IMG_271

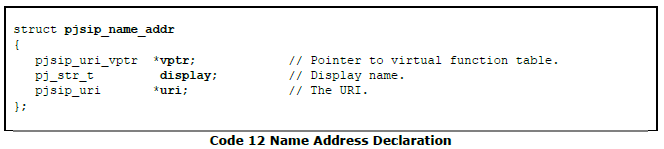
创建一个新的[tel:URI](http://blog.csdn.net/smllyy/article/details/tel:URI" \t "http://blog.csdn.net/smllyy/article/details/_blank)。

IMG_272

根据RFC3966中定义的规则来比较两个电话号码是否相等。在比较过程中，它可以识别全球和本地号码，并忽略分隔符。

****命名地址****

一个命名地址并不是真正定义的一个新类型的URI，只是封装了已有的URI（如SIP URI），并增加了展示名。



下面的方法是命名地址特有的。除此之外，应用也可以使用基本URI中的函数来操作SIP/SIPS URI。

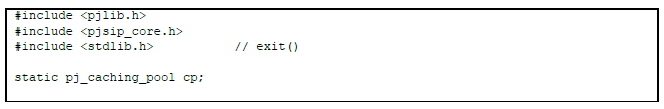
IMG_274

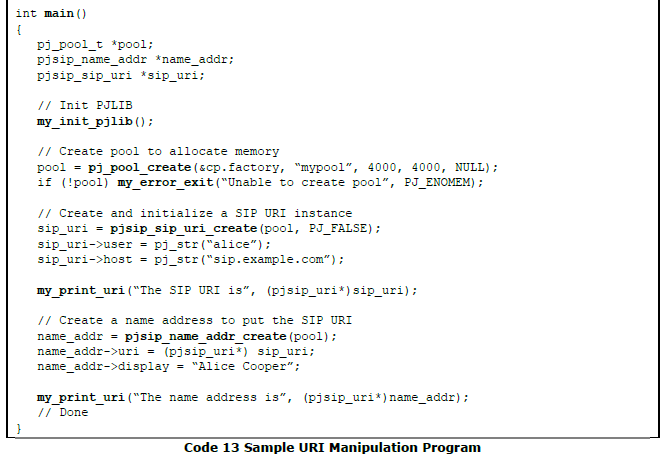
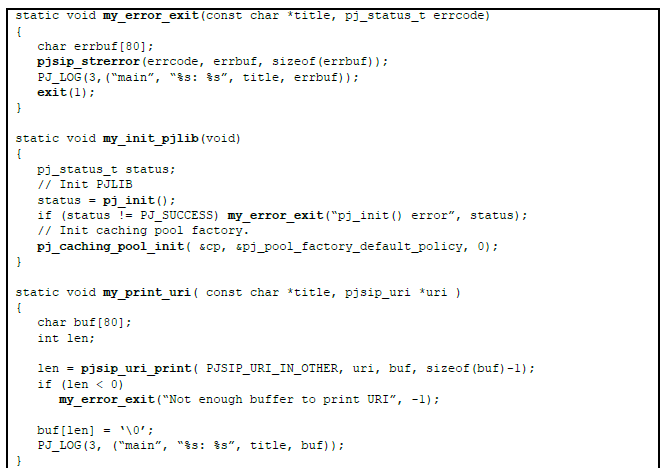
创建一个新的命名地址。它将初始化虚函数表的指针，设置空的展示名并把uri成员设置为NULL。

IMG_275

复制rhs到name\_addr。

****URI操作样例程序****

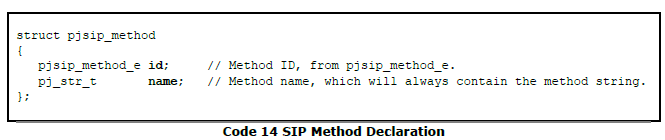
********

********

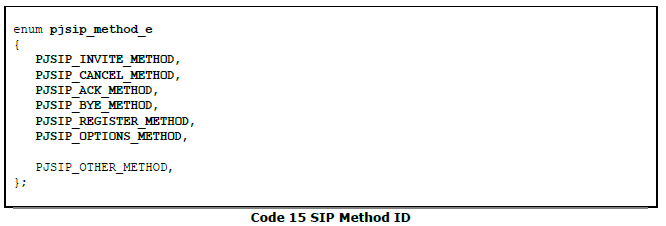
**SIP方法**

**SIP方法的表示（pjsip\_method**）

PJSIP中的SIP方法的表示是可扩展的；它可以在不用重新编译库的条件下支持新方法。



PJSIP核心库只声明了SIP核心标准（RFC3261）中指定的方法。对于这些核心方法，pjsip\_method中的id字段包含以下枚举中合适的值：



枚举中没有指定的方法，它的id值将被设置为PJSIP\_OTHER\_METHOD。在这种情况下，应用必须检查pjsip\_method的name字段来得到实际的方法。

****SIP**方法API**

以下函数可以用来操作PJSIP的SIP方法对象。

IMG_281

从字符串method\_name初始化SIP方法。它将初始化方法的id为正确的值。

IMG_282

从字符串method\_name初始化SIP方法（不复制字符串）。它将初始化方法的id为正确的值。np代表no pool。

IMG_283

从方法的ID枚举初始化SIP方法。name字段将被相应地初始化。

IMG_284

将rhs复制到method中

IMG_285

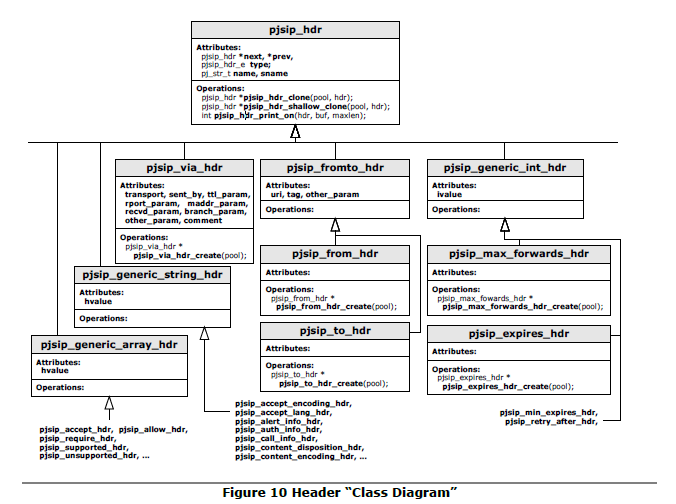
比较method1和method2的相等性。函数返回0则两个方法相等。返回+1如果method1大于method2；返回-1如果method1小于method2。

****头部域****

PJSIP中的所有的头部域有相同的头属性，如头类型，名字，短名字和虚函数表。因此，协议栈中所有头部域可以使用统一方法处理。

****头部域类图****

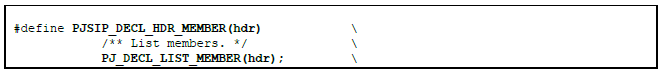
下面显示了PJSIP头部域的类图片段。这有比图中更多的头部域。PJSIP库实现了SIP标准（RFC3261）中指出的所有头部域。其他的头部域将会在相应的扩展模块中实现。

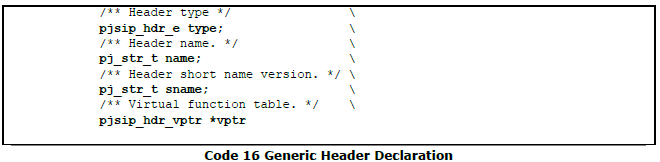


正如类图所示，每个指定的头部域通常只提供一个特定的函数。例如，创建头部域实例函数。

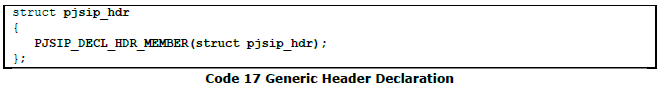
****头部域结构****

为了保证所有的包含共同的属性，并且这些属性在正确且相同的内存布局中。头部域必须调用PJSIP\_DECL\_HDR\_MEMBER宏作为头部域的第一个元素，指定头部域的名字为宏的参数。



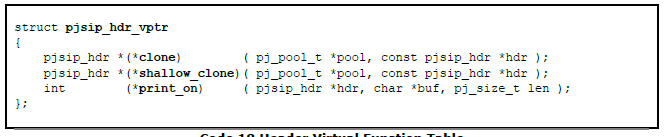


PJSIP中定义了pjsip\_hdr结构，它包含了头部域的共同属性。因此，所有的头部域可以转化为pjsip\_hdr类型。这样它们就可以统一操作。



****共同的头部域函数****

pjsip\_hdr\_vptr是一个虚函数表，它的实现是每种类型的头部域各自提供的。这个表包含的函数指针如下：



虽然应用可以直接调用pjsip\_hdr\_vptr中的函数指针，但是建议使用下面头部域API，因为它们具有较高的可读性。

IMG_291

对hdr进行深拷贝。

IMG_292

实现头部域的浅拷贝。浅拷贝会给一个指定的头部域创建一个新的精确的拷贝，但是它的大多数值仍指向原头部域。通常，浅拷贝只是简单地memcpy()。因此它的执行速度比深拷贝快。

但是浅拷贝时要小心，要知道新头部域和原头部域是共享相同指向值的指针的。因此，当包含原头部域的内存池被销毁时，即使新的头部域使用了不同的内存池来创建，都会变成无效的。或者原头部域的某些值被修改，浅拷贝的新的头部域中的值也会被修改。

尽管如此，库中还是广泛使用了浅拷贝。例如，一个dialog在同一个session期间有些头部域的值或多或少会保持一段时间不变（如From、To、Call-ID、Route和Contact）。当创建一个请求时，dialog可以只浅拷贝这些头部域到请求消息中。

IMG_293

将指定的头部域输出到一个缓存中（如，传输之前）。这个函数会返回输出到缓存的字节数或-1代表缓存满。

****支持的头部域****

标准的PJSIP头部域声明在<pjsip/sip\_msg.h>中。其它头部域声明在实现指定功能或SIP扩展的头文件中（如SIMPLE扩展的头部域等）。

每个头部域通常只定义了一个API来操作它们。例如，创建指定头部域的函数。其它APIs从虚函数表中输出。

APIs用来创建一个头部域，命名为头部域名加上\_create()。例如，调用pjsip\_via\_hadr\_create()来创建一个pjsip\_via\_hdr头部域实例。

****头部域数组元素****

大多数SIP头部域（例如，Require，Contact，Via等）通过逗号隔开组合成一个单一的头部域。例如：

IMG_294

注意：PJSIP不支持复杂头部域（如Contact，Via，Route，Record-Route）表示成数组元素。简单的字符串数组可以支持（如，Require，Supported等）。

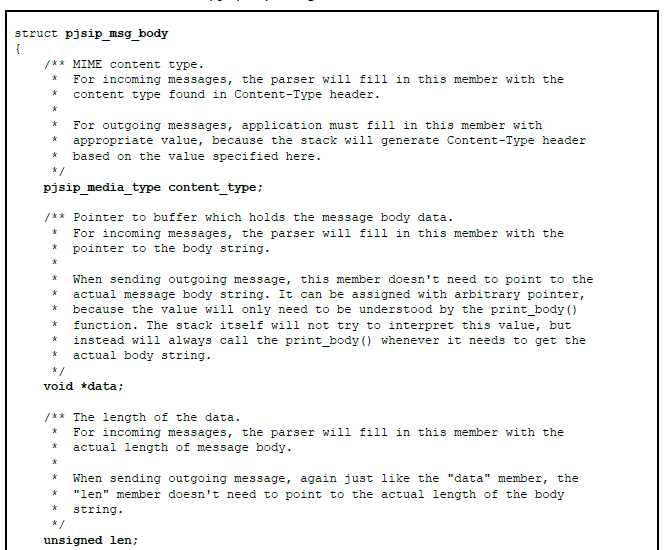
当解析器在头部域中遇到这样的数组时，它将按照出现的顺序把它们分隔成一个个单一的头部域。因此对于上面的例子，解析器将修改消息为：

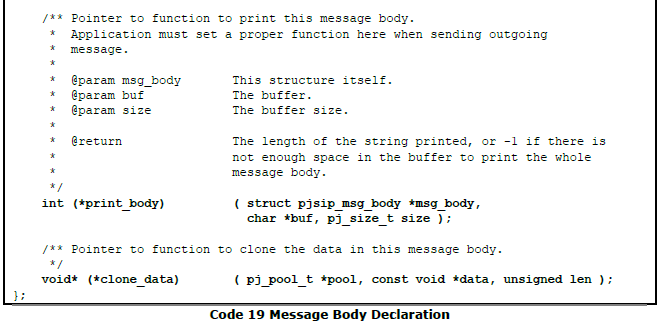
IMG_295

SIP标准规定不同种类的头部域表示应该同样处理。当头部域数组被支持，应用不仅要检查所有的头部域，还要检查一些头部域看看其中有没有包含数组。删除头部域数组的支持，应用只需要检查消息中的头部域主列表。

****消息体****

PJSIP中使用pjsip\_msg\_body来表示SIP消息体。这个结构定义在<pjsip/sip\_msg.h>中。





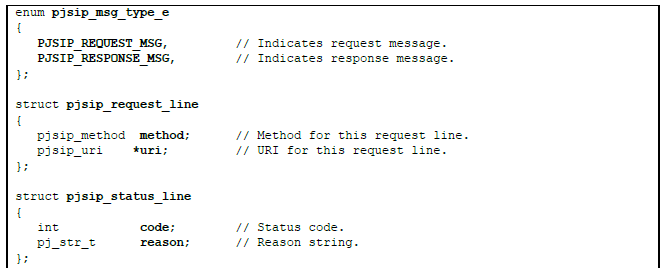
下面是操作SIP消息对象的APIs：

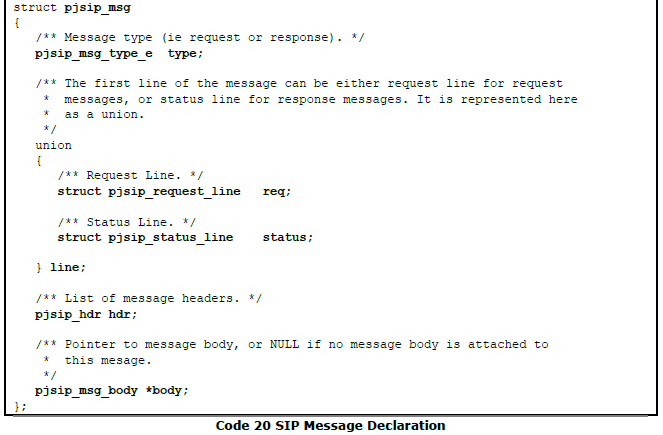
IMG_298

拷贝消息体。它将使用原消息体的clone\_data方法来复制消息体的内容。

****消息（pjsip\_msg**）**

PJSIP中，请求和响应都表示为pjsip\_msg结构。声明在<pjsip/sip\_msg.h>中。下面的代码片段展示了pjsip\_msg和其它支持的结构的声明：





下面是操作SIP消息对象的APIs。

IMG_301

IMG_302

根据类型来创建一个请求或响应。

IMG_303

从指定的开始位置，在消息中找出指定类型的头部域。如果start为NULL，该函数将从消息的第一个头部域找起，没找到则返回NULL。

IMG_304

从指定的位置开始，在消息的头部域列表中找出长、短名字符合指定名字的头部域。

IMG_305

在消息的头部域列表的最后加上hdr头部域。

IMG_306

在消息的头部域列表的第一个位置加上hdr头部域。

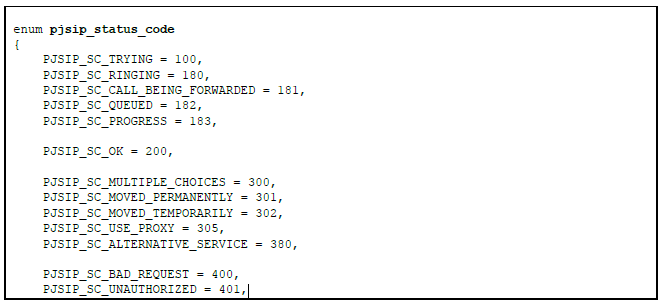
IMG_307

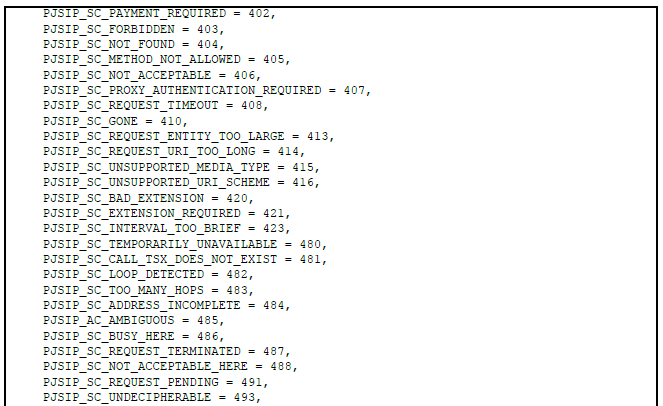
将指定的消息输出到指定的缓存中。这个函数会返回输出到缓存的字节数或-1代表缓存满。

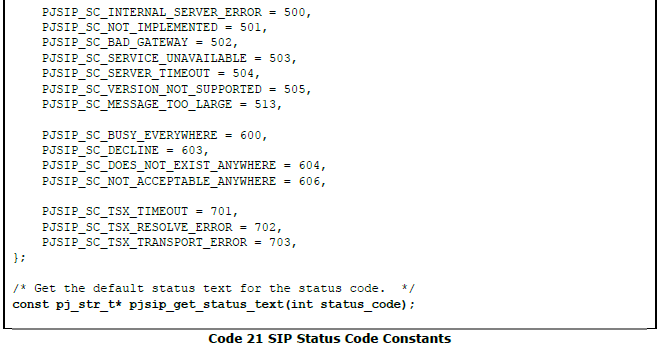
****SIP状态码****

SIP核心标准（RFC3261）中定义的SIP状态码表示为<pjsip/sip\_msg.h>中定义的枚举pjsip\_status\_code。除此之外，默认的原因短语可以通过调用pjsip\_get\_status\_text()返回。

下面的片段展示了PJSIP中的状态码的声明：







PJSIP中还定义了新的状态集（如7xx）代表消息处理过程中更大的错误状态（如transport错误，DNS错误等等）。

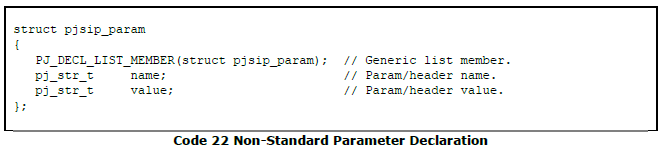
这个状态集只在内部使用，不会被传输。

****非标准参数元素****

在PJSIP中，已知的或者标准的参数（如URI参数，头部域参数）通常将被表示为相关的结构的单个属性/域。非标准参数将会被放入一个参数列表中，每个参数表示为pjsip\_param结构。非标准参数通常声明为所属结构的other\_param域。

****数据结构表示（pjsip\_param**）**

这个结构表示一个列表中的每个参数。



例如它的usage，请见pjsip\_sip\_uri声明中的other\_param和header\_param域（见前面3.1.4‘’SIP 和 SIPs URI‘）或者pjsip\_tel\_uri声明中的other\_param（见前面3.1.5“Tel URI”）。

****非标准参数的操作****

下面的函数用来帮助处理非标准的参数：

IMG_312

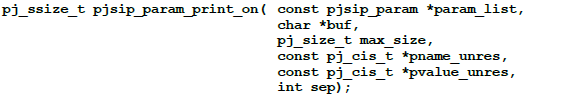
这个函数提供大小写不敏感的参数查找。

IMG_313

深度拷贝参数列表。

IMG_314

浅拷贝参数列表。



输出参数列表到指定的缓存中。pname\_unres和pvalue\_unres分别指出了pname和pvalue需要转义的字符。其他字符将不被转义。sep参数指定了参数之间的分隔符（普通参数之间用“；”分隔，头部域参数之间用“，”分隔）。

****转义规则****

PJSIP提供解析时自动的不转义和输出时的不转义，仅对以下消息元素实现。

1. 所有类型的URI以及它们的元素，依据它们各自的转义规则进行转义和反转义

2. 出现在消息元素中的参数（如URL，头部域中等等），自动进行转义和反转义

其它消息元素将由协议栈原样地，不加解释地传递。

# **PJSIP开发手册之解析器（四）**

****第四章 解析器****

下面PJSIP解析器的一些特征：

1. 它是自顶而下，手写的解析器。它使用了PJLIB的扫描器，此scanner速度很快，并减轻了解析器的复杂性。这样增加解析器可读性。

2. 如上所述，执行速度很快。在一台P412.6GHz机器上，1秒钟可以解析68k个800bytes的SIP消息或860k个80bytesde 的URIs。

3. 它是可重入的。这使它在多处理器的机器上具有扩展性。

4. 它具有可扩展性。可以使用模块向解析器插入新的URI或头部域。

解析器采用了许多可以想到的技巧来获取最高性能：

1. 它对所有消息元素使用零拷贝。即当一个元素（如pvalue）解析后，解析器并不会拷贝pvalue的内容到消息中的相应字段，而只是将指针和长度存放到消息中相应的字段中。这样做，是因为PJSIP在整个库中使用pj\_str\_t类型，不需要字符串以NULL结尾。

2. 它使用PJSIP的内存池（pj\_pool\_t）来为消息结构的内存分配。这比传统的malloc()函数快了好几倍。

3. 它使用零同步。解析器是可以重入的，因此不需要同步功能。

4. 它使用PJLIB的try/catch异常框架。这不仅大大简化了解析器的错误检查。使用异常框架，只需要在解析器的最顶部安装一个异常处理器即可。

PJSIP没有实现的一个特征是“懒解析”。在早期的设计阶段，我们决定不实现它的原因如下：

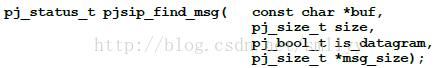
1. 它会使事情变复杂，特别是错误处理。使用懒解析后，在后期阶段程序需要访问特定的消息元素时，如果解析器解析失败，基本上程序的各个部分都要准备处理错误情况。

2. 后来，我们相信PJSIP解析器很快而且不需要懒解析。尽管这样说，PJSIP解析器对一些应用，有一些可以打开的开关，用来忽略对一些头部域的解析（如代理服务器它不需要检查一些头部域类型）。

****函数****

PJSIP主要的解析器在<pjsip/sip\_parser.h>中声明，在<pjsip/sip\_parser.c>中定义。库的其它部分可能提供其它解析功能和扩展解析器（如<pjsip/sip\_tel\_uri.c>提供了解析TELURI的函数，并把这个函数注册到主解析器）。

消息解析



检查缓存buf中到来的包是否包含一个合法的SIP消息。当一个合法的SIP消息被检测到，将消息的大小存到msg\_size中。如果指定了is\_datagram，则这个函数将一直返回PJ\_SUCCESS。

注意，此函数期望的缓存是以NULL结尾。

IMG_257

解析一个缓存到SIP消息。解析器要等到至少SIP请求/状态行已经被成功解析之后，才返回消息。如果err\_list参数不为NULL，解析过程中遇到错误将会放入err\_list参数中。

注意，此函数期望的缓存是以NULL结尾。

IMG_258

解析一个缓存到SIP消息。解析器要等到至少SIP请求/状态行已经被成功解析之后，才返回消息。除此之外，这个函数更新rdata中的msg\_info中的头部域指针。

注意，此函数期望的缓存是以NULL结尾。

****URI**解析**

IMG_259

解析一个缓存到SIP URI。如果option设置为PJSIP\_PARSER\_URI\_AS\_NAMEADDRESS，这个函数将包装成命名地址。如果option设置为PJSIP\_PARSER\_URI\_IN\_FROM\_TO\_HDR，且URI没有包含在括号内，函数将不会解析URI之后的参数（它将被认为是头部域参数，而不是URI参数）。

此函数可以解析任何PJSIP库可以识别的URI类型，并根据模式可以返回正确的URI实例。

注意，此函数期望的缓存是以NULL结尾。

****头部域解析****

****IMG_260****

根据头部域类型hname，以行的方式解析头部域内容（即冒号后面的头部域部分）。它返回正确的头部域实例。

注意，此函数期望的缓存是以NULL结尾。

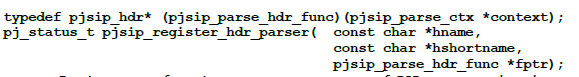
IMG_261

解析在input缓存中的找到的多个头部域，并把结果放入hdr\_list中。此函数期望头部域以换行符（如在SIP消息）或以’&‘分隔符（如在URI中）来分隔。最后一个头部域的分隔符是可选的。

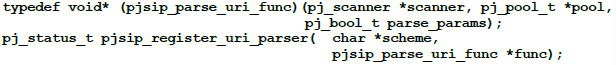
注意，此函数期望的缓存是以NULL结尾。

****扩展解析****

解析器可以注册新的函数指针来解析新的类型的头部域或新的类型的URI。



注册新的函数来解析一个新的类型的SIP消息头部域。



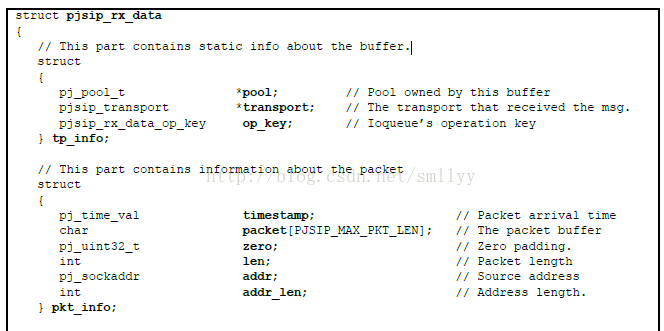
# **PJSIP开发手册之消息缓存区（五）**

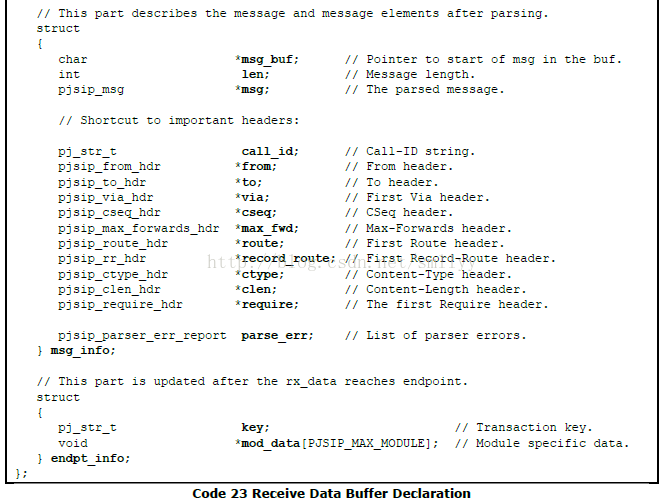
****第五章 消息缓存区****

****接收消息缓存区****

PJSIP接收到一个SIP消息后，将以pjsip\_rx\_data类型，而不是简单的消息本身，来传递到PJSIP的不同软件组件。这个结构（pjsip\_rx\_data）包含接收到的消息的所有信息。

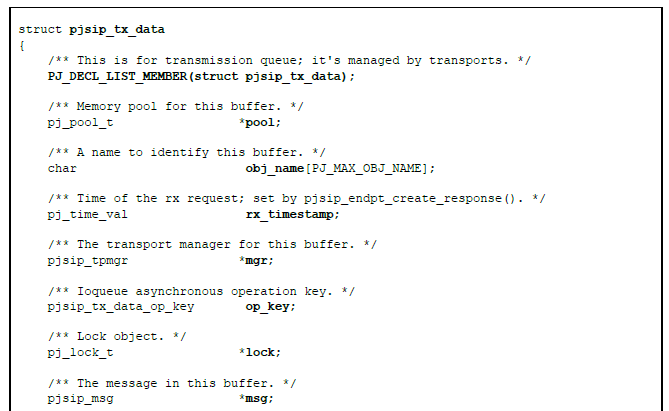
****接收数据缓存结构****

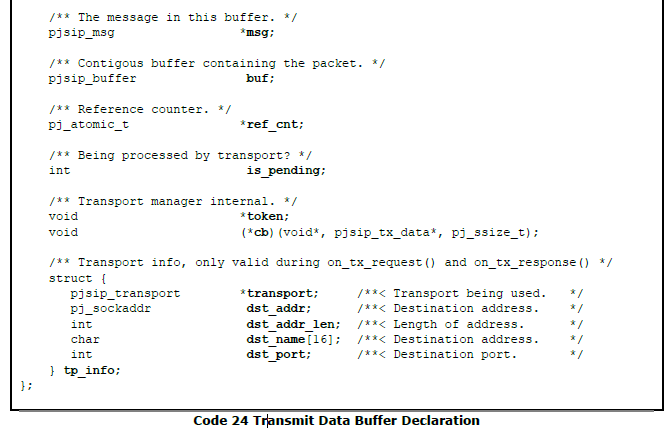
********

********

**传输数据缓存区（pjsip\_tx\_data**）

当PJSIP应用想要发送外出消息，它必须创建一个传输数据缓存区。传输数据缓存区提供了与消息字段内存分配有关的内存池，引用计数器，锁保护以及传输层用来处理消息的其它信息。





# **PJSIP开发手册之传输层（六）**

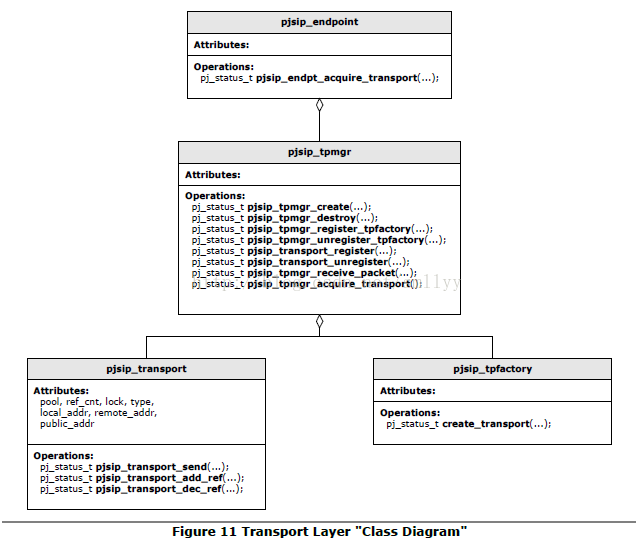
****第六章 传输层****

Transport是用来通过网络发送/接收消息的。PJSIP的transport框架是可扩展的，这意味着程序可以自己的transport来传输消息。

****传输层设计****

****类图****

下图展示了transport层的各个实例之间的关系。



****Transport Manager****

Transport Manager（pjsip\_tpmgr）管理所有的transport对象和工厂。它提供以下功能：

* 通过使用引用计数器和空闲的定时器管理transport的生命周期
* 管理transport工厂
* 从transport接收包，解析包并把SIP消息传给Endpoint
* 基于transport的类型和远端的地址，找到匹配的transport来给指定地点发送SIP消息
* 当不存在可用的transport来发送SIP消息时，动态创建transport

每个Endpoint只有一个transport manager。Transport manager通常是应用不可见的，应用需要使用Endpoint提供的函数。

****Transport**工厂**

Transport工厂（pjsip\_tpfactory）是用来创建与远端Endpoint的连接。这种类型的连接的一个例子就是TCP transport。每个目的地址需要创建一个TCP transport。

当transport manager检测到它需要为新的地址创建新的transport，它将会找到符合的transport工厂，并让工厂创建一个连接。

****Transport****

Transport对象用pjsip\_transport结构表示。每个实例代表一个socket处理（如UDP，TCP），但是transport层也支持non-socket的传输。

通用的transport操作

从框架角度来看，transport对象是一个活动的对象。框架中并不存在轮询transport对象的机制。因此传输对象必须找到自己的方式来从网络上接收数据包，以及将数据包提交到transport manager为之后的处理。

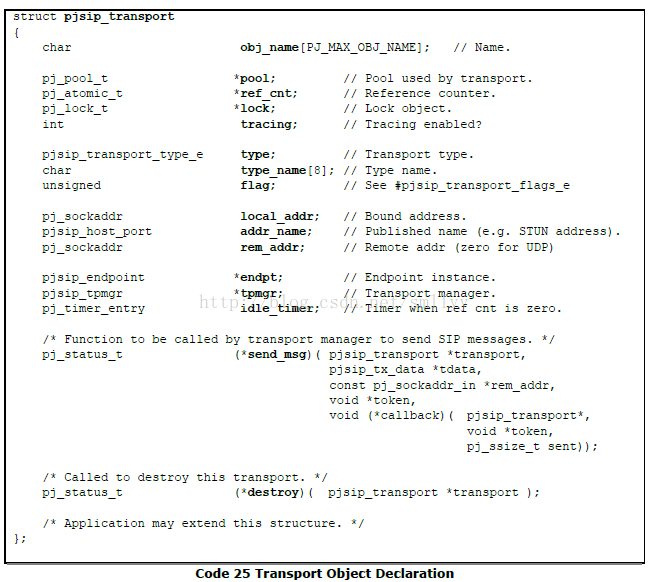
推荐的实现方法是将transport的socket处理注册到Endpoint的I/O队列（pj\_ioqueue\_t）。因此，当Endpoint轮询I/O队列，网络上的包将被transport对象接收。

一旦transport对象接收了一个包，它必须调用pjsip\_tpmgr\_receiver\_packet()函数将这个包传给transport manager，这样这个包将被解析并分发到栈剩下的部分。Transport对象必须初始化接收数据缓存tp\_info和pkt\_info。

每个transport对象有一个指向发送消息到网络上的函数指针（如send\_msg()）。应用（或栈）向网络发送消息通过调用pjsip\_transport\_send()函数，这个函数最终也会调用各个transport对象的send\_msg()函数。发送包可以异步完成，transport对象的send\_msg()必须返回PJ\_EPENDING，并且当消息到达目的地时，调用参数内指定的回调函数。

****Transport**对象**声明****

下面代码展示了transport对象的声明。



****Transport管理****

通过调用pisip\_transport\_register()函数来注册transport到transport manager。在调用此函数前transport结构中的所有成员必须被初始化。

Transport的生命周期由transport manager自动管理。每当transport的引用计数器为0时，一个空闲定时器将启动。当空闲的计数器超时时并且引用计数器仍为0，transport manager将调用pjsip\_transport\_unregister()来销毁transport。这个函数将transport从transport manager的哈希表上注销，并最终销毁它。

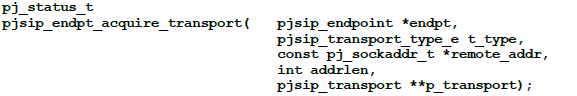
一些transport需要一直存在，即使没有被使用（如UDP transport，它是一个单例实例）。为了阻止那个transport被删除，它需要把引用计数器初始设置为1，这样，引用计数器将永不为0。

****Transport**错误处理**

Transport用户负责处理transport错误（如发送包失败或连接重置）。Transport对象自己不需要处理这种错误，除了在函数的返回值报出错误。特别地，它一定不要尝试去重连接一个失败/关闭的连接。

****使用transport****

****函数指南****



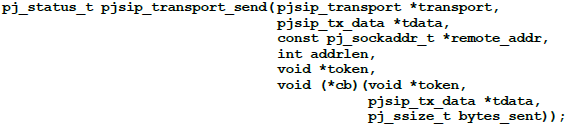
获取t\_type类型的transport来发送消息到目的地remote\_addr。注意，当transport被成功获取，这个transport的引用指针将增加。

IMG_259

增加transport的引用计数器。这个函数阻止transport被销毁，并且它可以取消活动的空闲计数器。

IMG_260

减少transport的引用计数器。每当transport的引用计数器为0时，一个空闲定时器将启动。当空闲的计数器超时时并且引用计数器仍为0，transport manager将调用pjsip\_transport\_unregister()来销毁transport。



发送tdata到remote\_addr使用transport。如果函数立刻完成且数据已经发送，则返回PJ\_SUCCESS。如果函数立刻完成但有错误，则返回一个非0的错误码。这两种情况下，回调函数都不会被调用。

如果函数没有立刻完成（如当下层socket缓存满），函数将返回PJ\_EPENDING，并且调用者将通过cb回调函数被通知。如果待发送的操作完成时有错误，则bytes\_sent是负值的错误码（使用”pj\_status\_t status = -bytes\_sent”来获取错误码）。

该函数只会照原样发送消息，不会对消息做任何验证。Via头部域也不会被修改。

****扩展transports****

PJSIP可以扩展自定义的transports。理论上，所有类型的transport，不局限于TCP/IP，都可以加入transport manager的框架中。

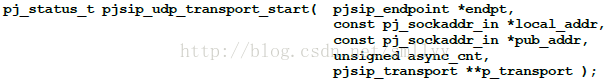
****初始化transports****

PJSIP是默认不会开始任何的transport（甚至内置的transport）；应用在需要时会被初始化并开始transports。

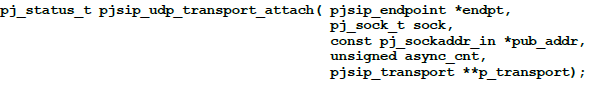
下面是内置的UDP和TCP transport的初始化。

****UDP transport**初始化**

PJSIP提供两种选择去创建、初始化和开始UDP transports。下面函数声明在<pjsip/sip\_transport\_udp.h>中：

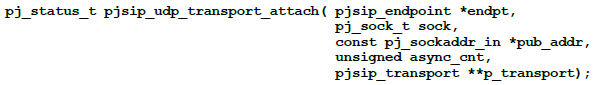


创建，初始化，注册和开始一个新的UDP transport。UDP socket将被绑定到local\_addr。如果Endpoint位于防火墙/NAT等端口转发设备后面，那么pub\_addr可以用作此transport对象的公共地址；否则pub\_addr应该与local\_addr相同。参数async\_cnt指定这个transport对象可以允许有多少个并发操作，为了性能最大化，这个参数应该等于节点的处理器数。



如果transport对象启动成功，函数返回PJ\_SUCCESS，且传输对象返回到p\_transport参数中。应用可以立刻使用这个transport对象，而不需要向transport manager注册，因为这个函数返回成功时，就已注册过了。

关于错误，函数将返回一个非0的错误码。



在UDP socket已经可用的情况下创建、初始化、注册和启动一个新的UDP transport对象。这个函数在如下的情况下非常有用，当应用已使用了STUN解析了socket的公共地址，这时可以不用关闭它后再重新创建一个新的socket，而是可以直接重用此socket。

****TCP transpor初始化****

待做

****TLS transport初始化****

待做

****SCTP transport**初始化**

待做

# **PJSIP开发手册之消息发送（七）**

****第七章 发送消息****

发送和接收消息是SIP应用的核心操作。每个模块里的on\_rx\_request()和on\_rx\_response()回调函数来处理接收到来的消息。

这章将介绍基本的发送外出消息的方法，例如，不使用transaction或dialog。

下一章Transaction介绍关于如何有状态地处理请求（到来请求和出去请求）。

****发送消息概述****

****创建消息****

PJSIP提供了多种API来创建请求和响应消息。下面是几种创建消息的方法：

* 对于响应消息，最简单的方法是调用pjsip\_endpt\_create\_response()函数
* 对于请求消息，你可以使用pjsip\_endpt\_create\_request()，pjsip\_endpt\_create\_request\_from\_hdr()，pjsip\_endpt\_create\_ack()或pjsip\_endpt\_create\_cancel ()函数
* 代理服务器可以根据到来的要转发的消息，来创建请求和响应，通过调用pjsip\_endpt\_create\_request\_fwd()和pjsip\_endpt\_create\_response\_fwd()函数
* 或者你可以创建请求或响应，通过pjsip\_endpt\_create\_tdata()创建传输缓存，pjsip\_msg\_create()来创建消息，pjsip\_msg\_add\_hdr()或pjsip\_msg\_insert\_first\_hdr()来插入头部域，设置消息体，等等。
* 高层的模块可以提供更多的方式去创建消息（如dialog层）。

所有的消息创建API（除了底层的pjsip\_endpt\_create\_tdata()）将设置传输缓存（pjsip\_tx\_data）的引用计数器为1，这意味着应用（或栈）在某个时刻必须减小引用计数器去销毁传输缓存。

所有的消息发送API将减小传输缓存（pjsip\_tx\_data）的引用计数器。这意味着如果应用没有对传输缓存的引用计数器做其他处理，这个缓存将在发送后销毁。

****发送消息****

调用pjsip\_endpt\_acquire\_transport()和pjsip\_transport\_send()函数是最基本的发送消息的方式。但是，在这种工作方式下，你需要知道目的地址（即sockaddr，不仅仅是主机名）。因为从得到消息和精确的socket地址有一些步骤要做（例如，决定使用哪个地址，执行RFC 3263的查找等），事实上这个函数是很底层的直接调用。

核心API发送消息使用的是pjsip\_endpt\_send\_request\_stateless()和pjsip\_endpt\_send\_response()函数。这是两个功能强大的函数，因为它们自动地处理transport层，并且是上层模块（例如，transation）的基础。

pjsip\_endpt\_send\_request\_stateless()函数是用来发送请求消息的，它的执行流程如下：

* 通过Request-URI和Route头部域的参数决定连接的目的地
* 使用RFC3263解析目的地服务器（定位SIP服务器）
* 选择和建立用来连接到服务器的transport
* 修改Via头部域的sent-by字段，用来反映所使用的头部域
* 使用当前的transport来发送消息
* 如果使用当前的transport无法连接到服务器则转到下一个服务器或transport

pjsip\_endpt\_send\_response()函数是用来发送响应消息的，它的执行流程如下：

* 根据RFC3261的18.2.2节来选择transport和发送响应的目的地地址
* 另外要遵循RFC3失败581关于rport参数的设置
* 使用所选择的transport来发送响应
* 如果使用所选择的transport发送响应失败或根据RFC3262解析服务器，则转到下个地址

因为可以异步发送消息（如，在TCP已经连接之后），所以所有函数提供通知应用关于传输状态的回调函数。这些回调函数也会通知应用关于失败处理是否发生，并且应用可以复写这些行为。

****函数指南****

****发送响应****

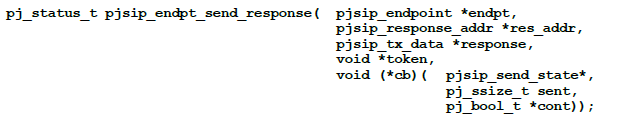
基本函数



根据rdata中的请求，利用状态码st\_code和状态短语st\_text来创建响应消息。如果st\_text为NULL，则使用默认的状态短语。

IMG_257

根据rdata中接收到的请求消息来决定发送响应的目的地址和transport。这个函数参照RFC 3261的18.2.2节和RFC 3581来计算目的地址和transport。目的地址和transport信息将返回在res\_addr参数中。



无状态地发送response，使用res\_addr中的目的地址和transport。响应地址信息（res\_addr）通常调用pjsip\_get\_response\_addr()来初始化。

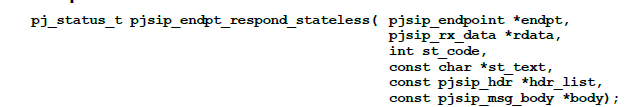
当回调函数cb被调用时，传输的状态和其他信息（包括原始的令牌）将会存储在pjsip\_send\_state中。如果消息发送成功，回调函数中的sent参数将是一个非0的正数。如果失败则sent参数是个负值，错误码是它不带负号的部分（即status=-sent）。如果cont的值不为0，则表示函数将要尝试其它的地址来发送这个消息（即失败处理）。应用可以通过设置这个参数为0，来不用尝试其它地址。

如果应用没有指定cb，这个函数将不会失败处理去尝试发送到下个地址，以防这个已选择的transport传送消息失败。

如果消息有效或一个非0错误码，这个函数将返回PJ\_SUCCESS。然而即使返回PJ\_SUCCESS，也不代表保证消息已被成功发送。

注意，回调函数也有可能在该函数返回之前被调用。

复合函数



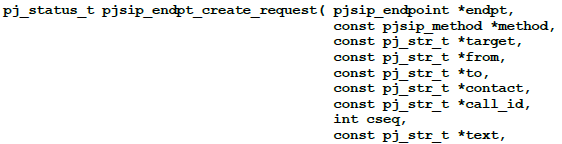
这个函数对一个到来的请求创建并发送一个响应。另外，调用者可以需要指定消息体和附加在响应消息中的头部域（存放在hdr\_list和body参数中）。如果没有附加的头部域和消息体，则这些参数设为NULL。

如果响应成功地创建并发送到transport层，或产生一个非0的错误码，则该函数返回PJ\_SUCCESS。然而即使返回PJ\_SUCCESS，也不代表保证消息已被成功发送。

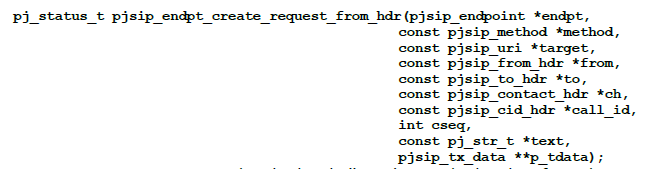
****发送请求****

****IMG_260****

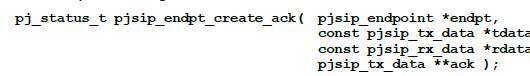
创建一个新的，空的传输数据。

IMG_262

创建指定method、target URI、from、to和contact头部域的请求消息。call\_id和cseq是可选择的。如果text被指定了，那么一个“text/plain”消息体将被加入。请求消息初始引用计数器为1，并返回给发送者在p\_tdata。



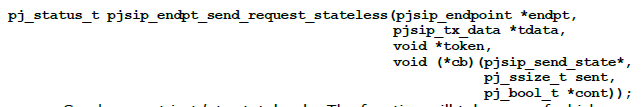
通过浅拷贝创建一个新的请求头部域。



根据rdata中接收到的响应从初始的请求消息中创建ACK请求消息。这个函数通常在INVITE请求接收到非成功的响应时被transaction使用。一个INVITE请求的成功响应的ACK请求消息通常是dialog的创建请求的函数生成的。

IMG_265

根据之前在tdata中发送的请求消息创建CANCEL请求。这个函数将创建一个新的传输数据缓存在p\_data中。



无状态地发送tdata中的请求消息。这个函数将关注使用哪个目的地址和transport，关注请求行和Route头部域中的URI。下面是这个函数执行的一些步骤：

* 根据Request-URI和Route头部域，决定连接的主机（pjsip\_get\_request\_addr()）
* 解析目的主机（pjsip\_endpt\_resolve()）
* 获取使用的transport（pjsip\_endpt\_acquire\_transport()）
* 发送消息（pjsip\_transport\_send()）
* 如果需要，失败处理转到下个地址或transport

当回调函数cb被调用时，传输的状态和其他信息（包括原始的令牌）将会存储在pjsip\_send\_state中。如果消息发送成功，回调函数中的sent参数将是一个非0的正数。如果失败则sent参数是个负值，错误码是它不带负号的部分（即status=-sent）。如果cont的值不为0，则表示函数将要尝试其它的地址来发送这个消息（即失败处理）。应用可以通过设置这个参数为0，来不用尝试其它地址。

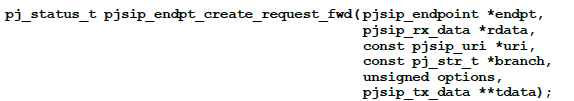
如果应用没有指定cb，这个函数将不会失败处理去尝试发送到下个地址，以防这个已选择的transport传送消息失败。

如果消息有效或一个非0错误码，这个函数将返回PJ\_SUCCESS。然而即使返回PJ\_SUCCESS，也不代表保证消息已被成功发送。

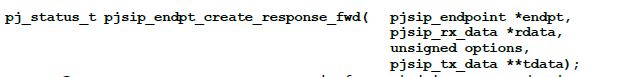
注意，回调函数也有可能在该函数返回之前被调用。

****无状态的代理转发****

代理服务器可能选择无状态地转发一个消息。当使用这种方式处理的时候，必须严格遵循RFC3261的16.11节。



创建一个新消息用来转发到上流的新的目的地址URI在uri中。这个新的消息是对rdata中的请求消息的深拷贝，除非有其他拷贝机制定义在options参数中。如果branch参数不为NULL，则将被用作Via头部域的branch参数。如果为NULL，则一个唯一的branch参数将会被使用。



代理服务器从rdata中的响应消息创建一个新的响应消息，用来转发到下流。注意，这个函数实际上将拷贝这个响应不检查响应的有效性或删除最顶端的Via头部域。这个函数将深拷贝响应，除非有其他拷贝机制定义在options参数中。

IMG_269

根据到来的请求消息的信息创建一个全局唯一的branch参数。这个函数保证重传的同样的请求会有相同的branch ID。

这个函数也能使用在循环检查过程中。如果相同的URI的相同的请求到达代理服务器，它将

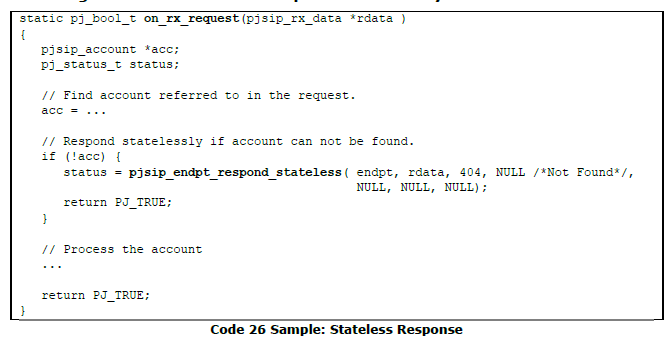
计算成同一个branch id。

注意，返回的字符串是从rdata的内存池中分配的。

****例子****

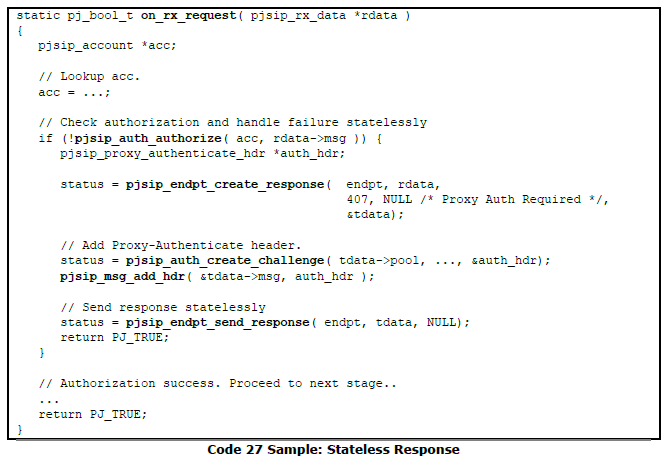
****发送响应****

未找到发送账号时无状态地发送响应

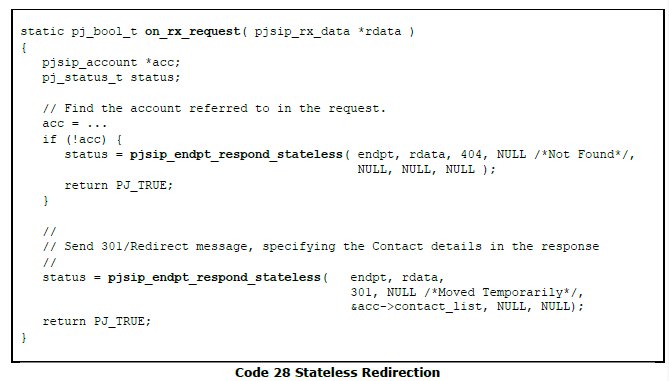


无状态地处理认证失败

其他方式来发送无状态的响应：

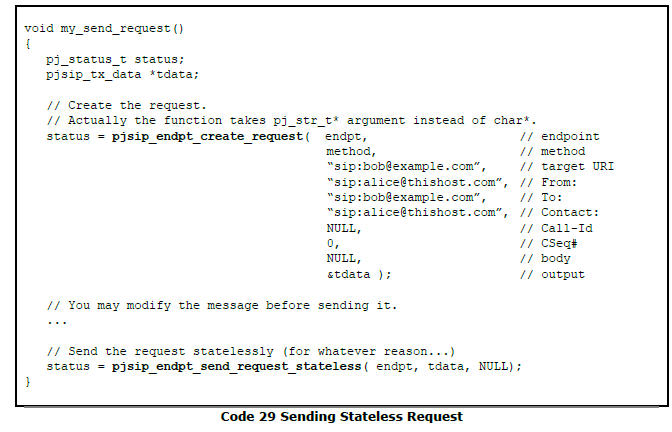


无状态的重定位

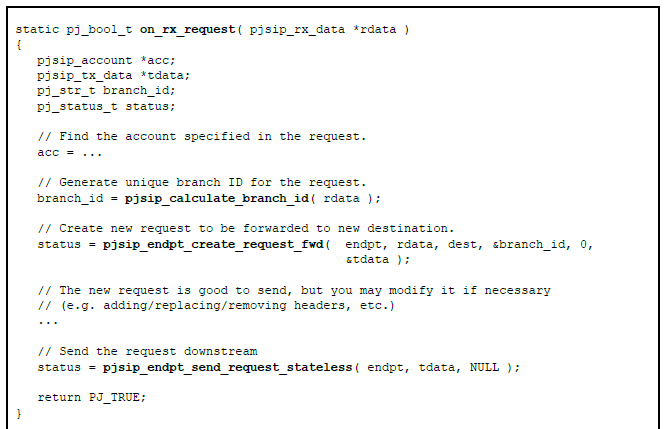


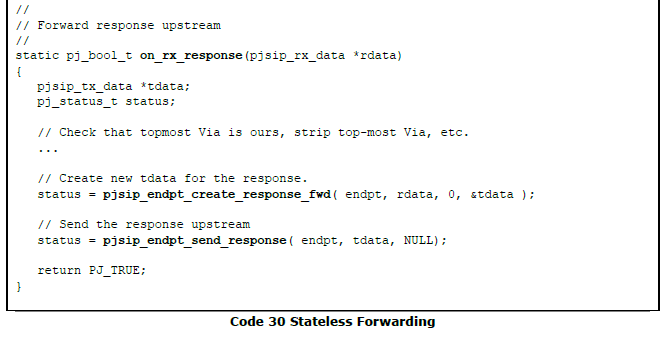
****发送请求****

无状态地发送请求



无状态地转发





# **PJSIP开发手册之事务（八）**

****第八章 Transaction****

****整体设计****

****介绍****

PJSIP中的transation表示为pjsip\_transaction结构，在<pjsip/sip\_transaction.h>中。Transaction的生命周期通常有以下几步：

* 通过pjsip\_tsx\_endpt\_create\_uac()/pjsip\_tsx\_create\_uas()创建
* 在初始化UAS之后，应用将需要调用pjsip\_tsx\_recv\_msg()传入初始的请求消息，这样这个transaction的状态就从NULL变为TRYING。后续的重传请求将被这个事务终止。
* 当应用想要使用这个Transaction发送请求或响应消息时，它将调用pjsip\_tsx\_send\_msg()
* 当有消息传给Transaction时（Endpoint传给的到来消息或TU传给的外出消息），或定时器超时，事务状态会自动改变，并且TU通过on\_tsx\_state()回调函数将被通知。
* 如果Transaction的状态变为PJSIP\_TSX\_STATE\_TERMINATED时，这个Transaction将被销毁。应用也可以调用pjsip\_tsx\_terminate()来强制终止这个Transaction。

****定时器和重传****

****Transaction****有两种类型的定时器：重传定时器和超时定时器。这两种定时器的值将被Transaction根据Transaction的类型（UAS或UAC），transport（可靠或不可靠）,和请求方法（INVITE和non-INVITE）来设置。

应用只能在全局的基础上改变定时器的时间间隔（可能只有在编译的时候）。

一个Transaction处理包括到来和外出的重传。到来的重传会被Transaction安静地终止和忽略；Transaction不会发出任何通知。Transaction在需要时，会自动重传外出消息；同样也不发出任何通知。

****INVITE最终响应和ACK请求****

失败的INVITE请求

注意：Transaction严格遵循RFC3261来处理失败的INVITE请求。

客户端Transaction：当一个客户端INVITE Transaction接收到了300-699的最终响应对于INVITE请求，它将自动发送ACK请求给响应。这个Transaction将会等待定时器D，在它超时前，任何到来的重传的300-699响应将以自动应答ACK请求。

服务器Transaction：当一个服务端INVITE Transaction被要求发送300-699最终响应，它将发送这个响应并且保持重传直到收到ACK请求或定时器H超时。当收到ACK请求时定时器H未超时时，Transaction将会转到Confirmed状态并等到定时器I超时时被销毁。如果是定时器H并没有收到一个有效的ACK，则Transaction将被销毁。

成功的INVITE请求

客户端Transaction：当一个客户端INVITE Transaction接收到了对于INVITE请求的2XX最终响应，它将在把响应传给它的TU（可以是一个dialog或一个应用）之后销毁自己。后来重传的2xx响应将会被直接传给dialog或应用。

无论如何，应用都必须基于接收的2xx响应发送ACK请求。

服务器Transaction：当一个服务端INVITE Transaction被要求发送2xx最终响应时，它将发送这个响应并且保持重传直到收到ACK请求或它被应用调用pjsip\_tsx\_terminate()终止。

对于实现的简单性，一个典型的UAS dialog通常将让Transaction来处理2xx INVITE响应的重传。但是代理的应用程序必须在接收和发送2xx响应后立即销毁UAS Transaction，来让2xx重传可以被两端的UA来处理。

注意：INVITE 服务端Transaction对于成功的INVITE请求的处理和RFC3261中规定的有些不同。RFC3261中规定服务端INVITE Transaction在发送了2xx响应后，立即被销毁。PJSIP的Transaction行为简化了dialog的实现，但是保持了RFC3261中对于代理应用的灵活性的兼容。

可以在INVITE服务端Transaction创建之后复写transaction->handle\_200resp为0（默认为非0），这样UAS INVITE Transaction将会在发送2xx响应后立刻被销毁。

****到来的ACK请求****

当INVITE 服务端Transaction完成非成功的最终响应，ACK请求将会被Transaction终止；TU也不会被通知。

当INVITE 服务端Transaction完成2XX的最终响应，第一个ACK请求将会被通知给TU。后来接收到的ACK重传将不会被通知给TU。

****服务器解析和Transports****

Transaction使用核心API pjsip\_endpt\_send\_request\_stateless()和pjsip\_endpt\_send\_response()来发送外出消息。这些函数提供服务器解析和发送消息的transport建立，和当检测到失败时替换transport。这个transaction使用这些函数提供的回调函数来监控传输过程和追踪使用的transport。

这个Transaction会增加使用的transport中的引用计数器。

TCP连接关闭

一个TCP连接的关闭不会自动导致这个transaction的失败。事实上，这个Transaction只有在它想要发送一个消息时，才会检查这个错误。当它这样做时，它允许使用替换的transport来发送消息。

****Via头部域****

Branch参数

UAC Transaction在Via头部域中的branch参数不存在时会自动生成一个唯一的branch参数。如果branch参数已存在，则这个Transaction将使用它作为一个标识，并遵循RFC3261和RFC2543的规则集。

Via Sent-By

Via头部域的Sent-By参数通过pjsip\_endpt\_send\_request\_stateless()和pjsip\_endpt\_send\_response()来设置。

****指南****

****基本函数****

****IMG_256****

初始化和注册一个Transaction层模块到指定的Endpoint。

IMG_257

获取Transaction层模块的实例

IMG_258

关闭Transaction层模块，并从Endpoint上注销。

IMG_259

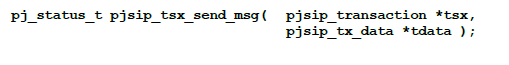
为tdata中的外出消息创建一个新的UAC Transaction，并设置TU为tsx\_user。这个Transaction将会自动初始化和注册到Transaction表中。注意，在调用这个函数之后，应用通常将会调用pjsip\_tsx\_send\_msg()来实际发送消息。

IMG_260

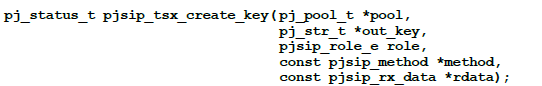
为rdata中的到来消息创建一个新的UAS Transaction，并设置TU为tsx\_user。这个Transaction将会自动初始化和注册到endpoint的Transaction表中。

IMG_261

应用必须在UAS Transaction创建之后调用这个函数，并把初始的请求消息传给Transaction，这样Transaction的状态会从NULL变成TRYING。TU的on\_tsx\_state()将会被调用。



通过Transaction发送消息。如果tdata为NULL，上个消息或创建期间指定的消息将被重传。当这个函数返回PJ\_SUCCESS时，将会减小tdata的引用计数器。



根据一个到来的请求或响应创建一个Transaction标识，并考虑消息是否遵循了RFC3261或RFC2543。这个key将用来在Endpoint的Transaction表中查找。

这个函数将key返回在out\_key参数中。role参数找到UAC或UAS Transaction，method参数包含消息的方法。

IMG_264

在Transaction表中找到指定的key的Transaction。如果lock参数是非0的，这个函数将会在返回Transaction之前锁住这个Transaction，这样其它线程就不能删除这个Transaction。

调用者负责在使用完这个Transaction后，调用pj\_mutex\_unlock()来解锁这个Transaction。

IMG_265

强制终止tsx这个Transaction，使用指定的状态吗st\_code。通常应用不需要调用这个函数，因为Transactions将会根据它们的状态机自动终止和销毁。

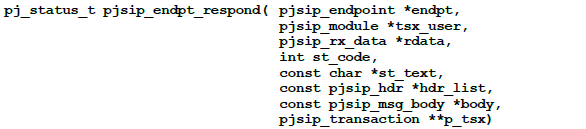
这个函数用在，如INVITE请求的200/OK响应接收或发送和UA层想要处理200/OK响应的重传。

这个Transaction将散发Transaction状态改变事件（状态改变到PJSIP\_TSX\_STATE\_TERMINATED），接着它会立刻被这个函数注销和销毁。

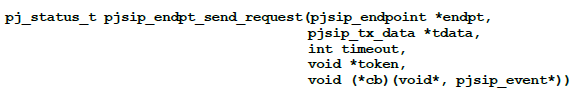
IMG_266

从一个到来的消息中获取一个Transaction对象。

****复合函数****



为到来的请求创建一个新的UAS Transaction来发送响应。

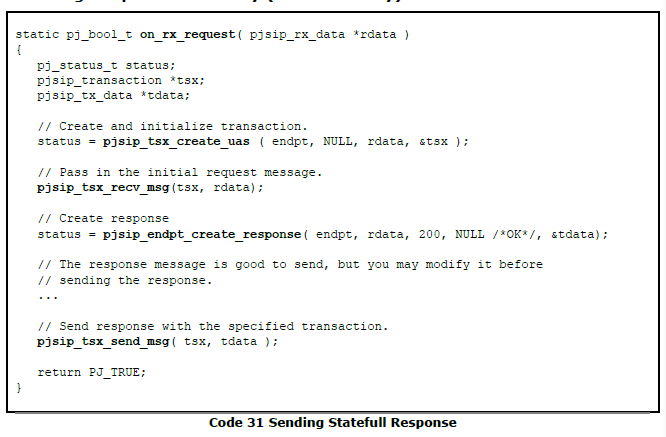


通过使用一个UAC transaction发送请求，和选择性地请求回调函数在Transaction完成时被调用。

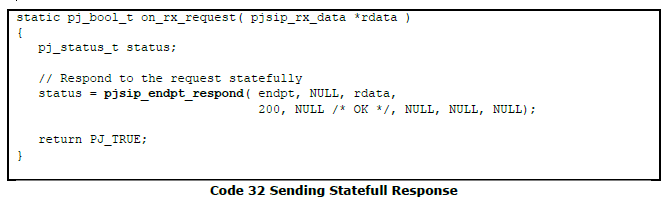
****发送有状态的响应****

****使用样例****

有状态地发响应（困难的方法）



有状态地发送响应（简单的方法）



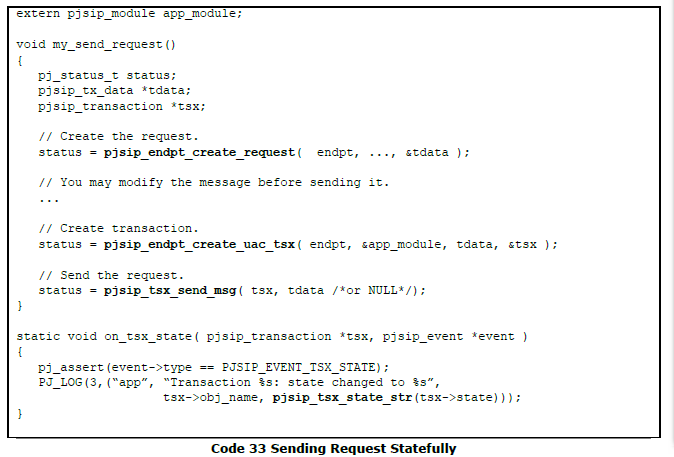
****有状态地发送请求****

两种方式：

* 使用pjsip\_endpt\_send\_request()
* 手动使用Transaction

****使用样例****

使用Transaction发送请求



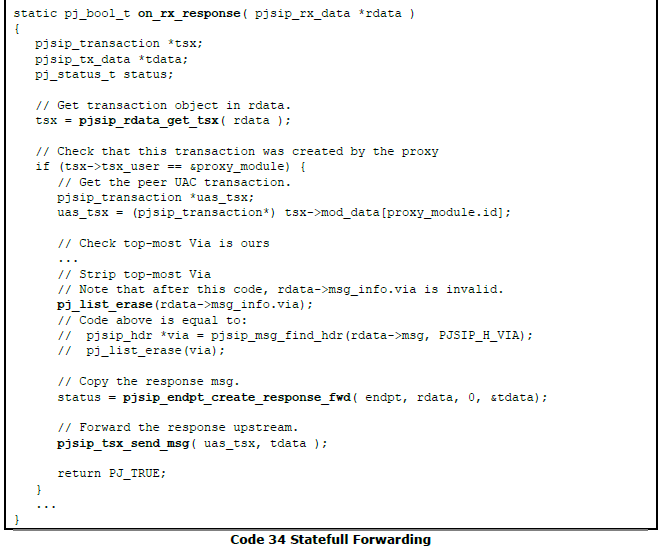
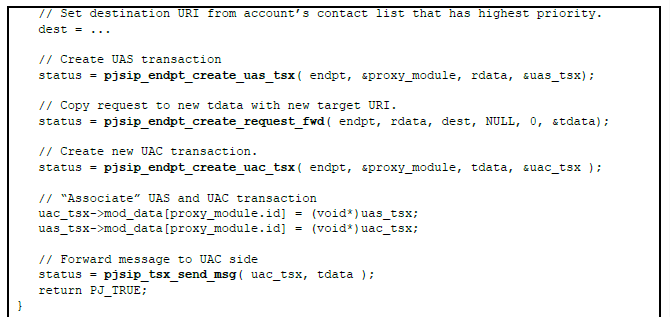
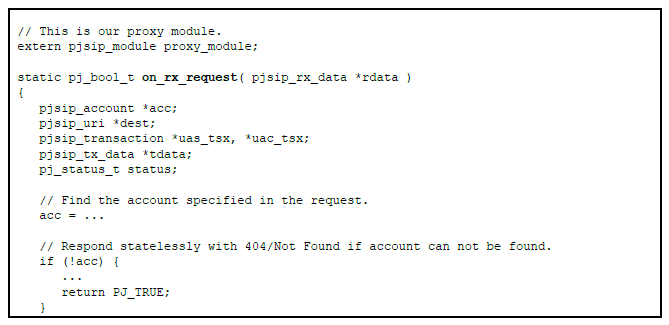
****有状态的代理转发****

****使用样例****

有状态的转发

下面的代码展示了一个示例的有状态转发的代理。这个代码为每边创建UAS和UAC Transaction，转发请求给UAC一边，转发所有响应从UAC边到UAS边。它也处理UAC边的Transaction定时器超时或其它错误，和发送响应到UAS边。

接收到的UAC边的CANCEL消息无法处理。

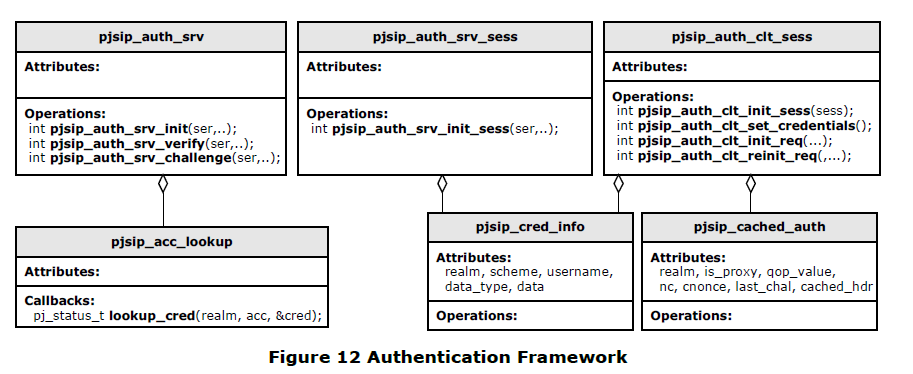


# **PJSIP开发手册之认证框架（九）**

****第九章 认证框架****

****PJSIP****的认证框架支持客户端和服务端的认证。这个认证框架默认支持HTTP摘要认证，但是其他认证机制也可以加入这个框架。

下图说明了该框架的“类图”：



****客户端认证框架****

客户端认证框架管理从客户端到所有下流的服务器的认证处理。它可以使用正确的证书（当这样的证书被提供）响应服务器的挑战，缓存认证信息和使用缓存的认证信息来初始化后续的请求。

****客户端认证框架指南****

认证APIs在<pjsip/sip\_auth.h>中声明。下面是认证所用的数据结构和函数的文档说明。

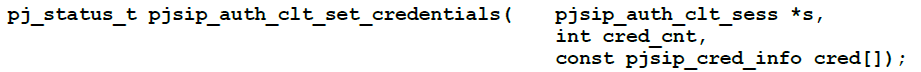
数据结构指南

|  |  |
| --- | --- |
| ****数据结构**** | ****描述**** |
| pjsip\_cred\_info | 这个数据结构描述了用来认证指定realm的证书。一个客户端在一个dialog或注册期间，可以有多个证书；每个证书的信息需要用来认证一个特定的下流的代理或服务器。  例如，客户端需要一个证书来认证它的outbound代理，和其它证书来认证端服务器。 |
| pjsip\_cached\_auth | 这个结构保存从一个特定服务器接收到的最近的挑战。客户端可以利用最近的挑战来创建一个请求。 |
| pjsip\_auth\_clt\_sess | 这个结构描述客户端认证会话。客户端将通常在一个dialog或客户端注册期间，一直保持这个结构。 |

函数指南

IMG_257

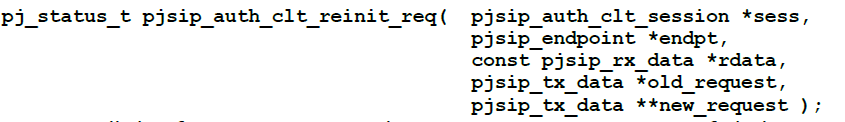
初始化客户端认证会话数据结构，并设置pool负责为该会话进行后续的内存分配。该版本的PJSIP中，options参数应该设置为0。



设置会话过程要使用的证书。这将使用客户端认证的内存池来复制指定的证书。

IMG_259

这个函数根据会话中缓存的信息，增加所有相关的认证头部域到一个新的外出请求tdata中。在调用这个函数前，请求消息的请求行必须有效。



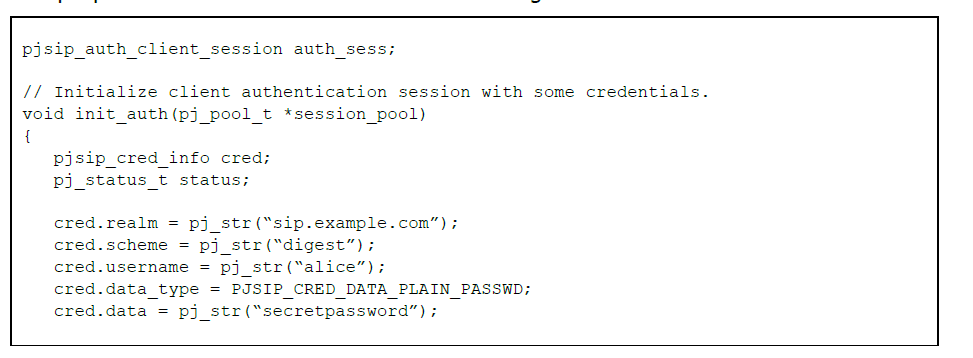
基于接收到失败的认证状态码（401/407响应），调用这个函数去重新初始化一个请求，这个函数将根据old\_request创建一个new\_request，和根据在rdata中找到的挑战增加适当的Authorization和Proxy-Authorization头部域。另外，这个函数也把相关的信息放进这个会话中。

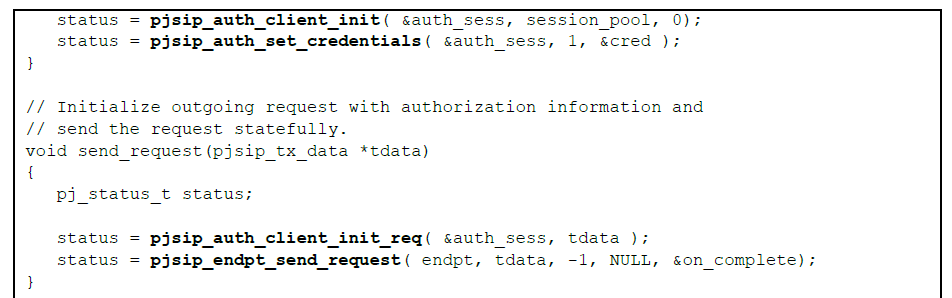
如果这个挑战的一个证书丢失，则这个函数将返回失败。注意这个函数会复用老的请求而不是创建一个新的。

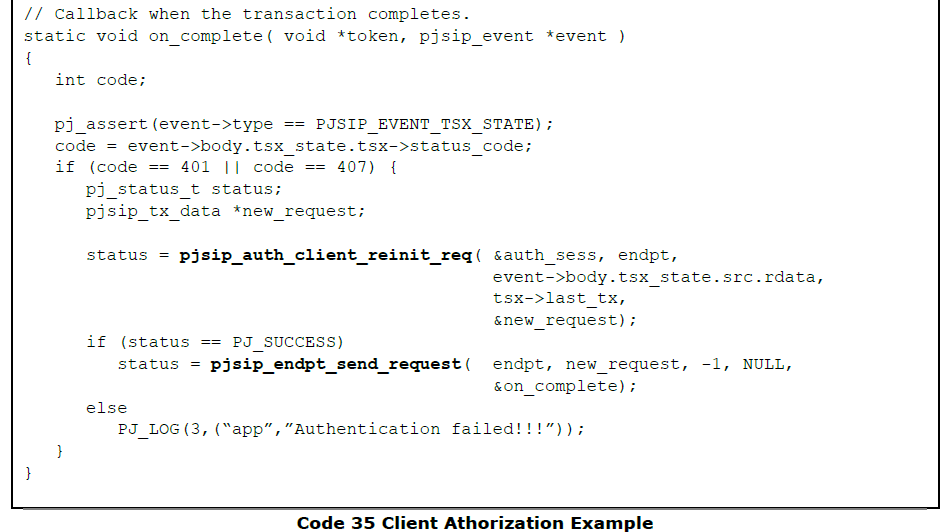
****例子****

客户端Transaction认证

下面的例子说明如何使用认证信息来初始化外出请求和如何处理接收到的服务器的挑战。为了简便起见，错误处理没有在下面的例子中展示。一个真实的应用需要各个情况的错误处理。







****服务端认证框架****

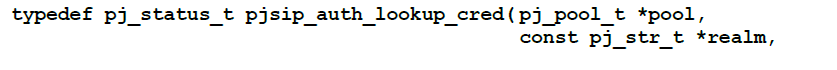
服务端认证框架提供两种类型的服务器认证机制：

* Session-less服务端认证，提供认证客户端的通用API。这个API提供基于Request-per-request的全局服务端认证机制，并且通常被无状态的代理应用程序所使用。
* 服务端认证会话，提供认证一个特定dialog或注册会话内的请求的APi。每个服务端的dialog或注册会话需要创建一个服务端认证会话实例。一个服务端认证会话将有唯一一个证书（初始的证书），并且这个证书必须在这个dialog或注册会话内被客户端使用。

服务端认证会话目前没有被实现。只有全局的Session-less服务端认证框架可用。

****服务端认证指南****

数据类型指南

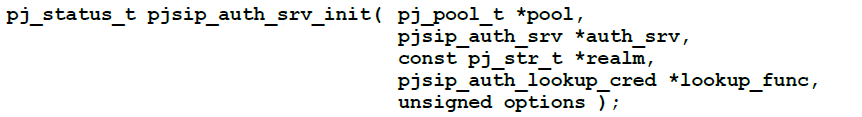


IMG_265

函数类型被注册到认证服务器上，用来查找在指定realm中的指定acc\_name的证书信息。当证书信息成功被恢复，这个函数将在cred\_info参数中填上证书并返回PJ\_SUCCESS。否则它返回以下一个错误码：

* PJSIP\_EAUTHACCNOTFOUND：没有找到指定realm的账号
* PJSIP\_EAUTHACCDISABLED：账号找到但不能使用。

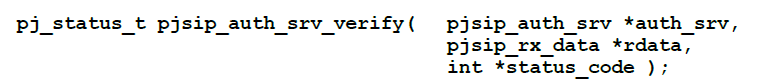
函数指南



初始化服务端认证会话数据结构来服务指定realm和使用lookup\_func函数来查找证书信息。

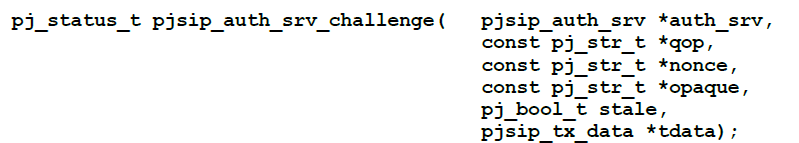
Options参数是下面值的位掩码组合：

* PJSIP\_AUTH\_SRV\_IS\_PROXY：指定这个服务器将作为一个代理服务器（而不是一个UAS）来认证客户端，这意味着Proxy-Authenticate将代替WWW-Authenticate被使用。



           请求服务端认证框架去核实在rdata中指定请求中的认证信息。如果status\_code不是NULL，它将被填上合适的响应状态码（401/407等等）。

           当请求中的认证信息可以被接受或当认证失败时出现下面的错误，这个函数将返回PJ\_SUCCESS:

* PJSIP\_EAUTHNOAUTH：在请求中没有认证的头部域被指定
* PJSIP\_EINVALIDAUTHSCHEME：无效/不支持的认证机制（目前只支持摘要认证）
* PJSIP\_EAUTHACCNOTFOUND或者PJSIP\_EAUTHACCDISABLED：是查找函数返回的错误码
* PJSIP\_EAUTHINVALIDDIGEST：无效的摘要
* 返回其它非0值代表系统错误

            向tdata中的外出消息中增加认证挑战的头部域。如果qop被指定，那么它将被加入到这个挑战中。应用也可以为应用指定自定义的nonce和opaque，               或者可以将这些值保留为NULL，让这个函数想这些值里填入随机的字符。

****扩展认证框架****

认证框架可以被扩展来支持除了HTTP摘要之外的其他认证机制。（例如，PGP等）

待做。

# **PJSIP开发手册之用户代理（十）**

****第十章基本用户代理层（UA）****

****基本Dialog概念****

基本的UA dialog提供管理SIP dialogs的基础设备和dialog usages，像dialog的状态，会话计数器，Call-ID，From，To，和Contact头部域，transactions中CSeq的排序，和路由集。

这个基本的UA dialog是不知道它正在使用哪种类型的会话（例如，INVITE会话，SUBSCRIBE/NOTIFY会话，REFER/NOTIFY会话，等等），并且它可以被用来同时在同一个dialog内建立多个不同类型的会话。

一个PJSIP dialog可以被认为只是一个存储常用dialog属性的被动数据结构。你不能讲dialog和INVITE会话混淆。一个INVITE会话是一个会话（也通常叫做dialog usage）在一个dialog内。同一个dialog内也可以有其他的会话、usages；它们共享共同的dialog属性（虽然每个dialog中可能只有一个INVITE会话）。

PJSIP dialog不知道它的会话的状态。它不知道是否一个INVITE会话已经被建立或者失去连接。事实上，PJSIP dialog甚至不知道在dialog中的是哪种类型的会话。它关心的只是这个dialog中有多少个活动的会话。Dialog初始的时候有一个活动的会话，并且当这个会话计数器到达0并且最后的transaction也终止了，这个dialog将被销毁。

增加和减小会话计数器是每个dialog usages的责任。

****Dialog会话****

Dialog会话在PJSIP的dialog框架中只是用一个引用计数器来表示。每当Dialog usages在一个特定的Dialog中创建或销毁一个会话时，它们将增加或减小这个引用计数器。

Dialog会话是Dialog usages创建的。在一个特定的Dialog中，一个Dialog usages可以创建多个会话（除了INVITE usage，它只能在一个Dialog中创建一个INVITE会话）。

关于“会话”，准确的表示将在Dialog usage模块中定义。基本的Dialog只关心Dialog中活动的会话数目。

****Dialog Usages****

Dialog usages是PJSIP的模块，注册到Dialog来接收Dialog的事件。多个模块可以注册到一个Dialog，因此Dialog可以有多个usages。每个Dialog usages模块负责处理一个指定的会话。例如，每当接收到一个新的SUBSCRIBE请求（和增加Dialog的会话计数器）时，subscribe usage模块将创建一个新的subscribe会话。当这个subscribe会话终止时减少这个会话计数器。

Dialog处理Dialog usages的过程和endpoint处理模块的过程类似；每当on\_rx\_request()和on\_rx\_response()事件发生，Dialog将按照Dialog usages的优先级从高到低，将事件传到每个usages，直到有一个模块返回true（例如，非0），在这种情况下，Dialog将停止分发事件。On\_tsx\_state()通知将分发到所有的Dialog usages。每个Dialog usage应该滤掉不属于它的transaction事件。

在它更底层的使用方面，应用可以直接管理Dialog，并且它是Dialog的“usage”（或用户）。在这种情况下，应用负责管理一个Dialog内的会话，即处理所有的请求和响应和建立/销毁会话。

在最后一章中，我们将学习管理会话的高层APIs。这些高层APIs是PJSIP模块并作为Dialog usages注册到Dialog。它们将处理各种类型会话中指定的各种类型的SIP消息（例如，一个INVITE usage模块将处理INVITE，PRACK，CANCEL，ACK，BYE，UPDATE和INFO，等等）。这些高层APIs根据会话的说明提供高层的回调函数。

在这章中，我们只学习基本，底层的Dialog usages。

****Dialog集****

每个Dialog包含在一个Dialog集中。每个Dialog集用一个共同的本地标签（如，From标签）标识。通常一个Dialog集只有一个Dialog。一个Dialog集有多个Dialog的情况，只有当外出的INVITE交叉时，在这种情况下，每个接收到含有不同的To便签的响应消息将在同一个Dialog集中创建一个新的Dialog。

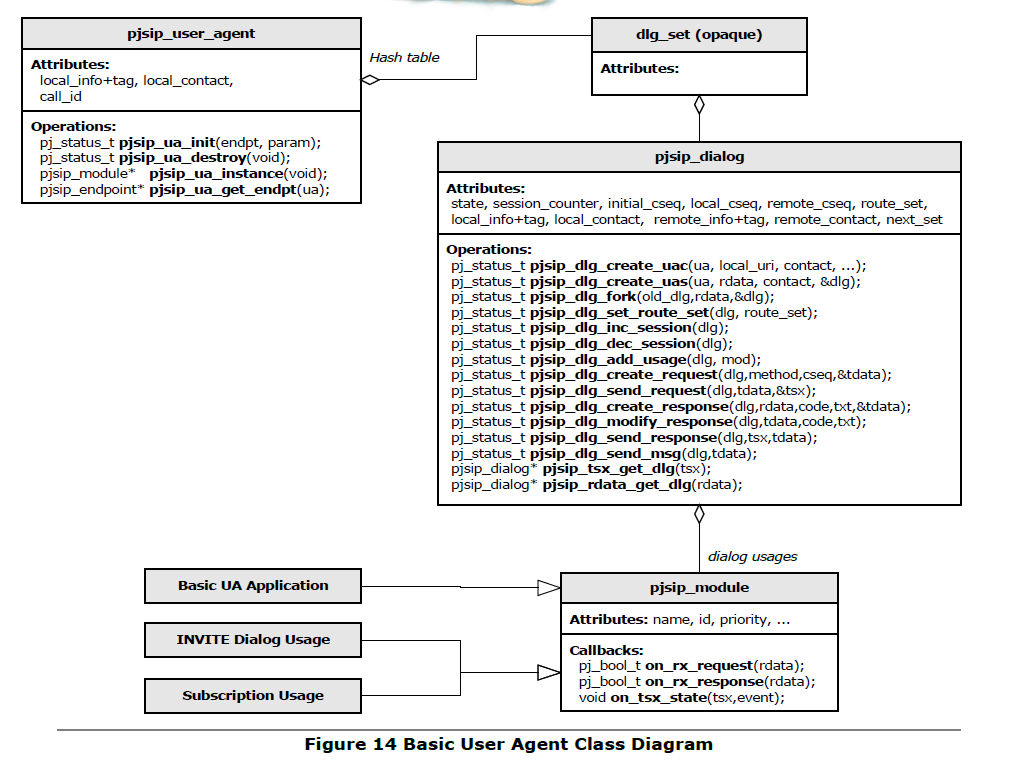
一个Dialog集在PJSIP中定义为一个不透明的类型（即void\*）。一个Dialog结构（pjsip\_dialog）有一个成员dlg\_set用来标识它所属的Dialog集。应用可以使用linked list API来保存一个Dialog的所有兄弟姐妹（在同一个会话中）。

****客户端认证****

一个Dialog维持一个客户端认证会话（pjsip\_auth\_clt\_sess），用来认证这个Dialog中发给下流服务器的请求。这个基本Dialog使用适当的认证头部域初始化每个外出的请求。然而，认证挑战必须由Dialog usages来处理；例如，这个基本Dialog当收到一个请求的401/407响应时不会自动重新尝试这个请求。

****类图****

下面的图说明了用户代理层和基本Dialog框架。



这个图展示了Dialog和它的usages之间的关系。在最基础/底层的情况下，应用模块是Dialog的唯一usage。在更高层的情况下，一些高层的模块（例如，pjsip\_invite\_usage和pjsip\_subscribe\_usage）可以作为dialog usages注册到一个dialog上，并且应用将从这些usages接收事件，而不是直接从dialog接收。

这个图也展示了PJSIP用户代理模块（pjsip\_user\_agent）。用户代理模块是所有dialogs的所有者；用户代理模块维持一个存放目前所有活动的dialog集的哈希表。

****交叉****

处理交叉情况

当用户代理检测到下流代理发来的响应的时候，交叉响应用户代理模块提供可以被应用注册的回调函数。一个交叉的响应（可以是临时的或者2xx响应）的dialog的To标签和其他已存在的dialog的To标签不同。当收到这样的响应时，用户代理将调用on\_dlg\_forked()回调函数接收到的响应和原来的dialog（应用初始创建的dialog）作为参数传入。

处理交叉情况完全是应用的责任。

接收到一个临时的交叉响应，应用可以：

* 忽略这个临时响应（可能一直等待，直到接收到一个最终的，交叉的2xx响应）；或者
* 创建一个新的Dialog（通过调用pjsip\_dlg\_fork()）。在这种情况下，从这个特定的call leg接收到的后续的响应，将进入这个新的Dialog中。

接收到一个交叉的2XX响应，应用可以：

* 终止这个特定的call leg。在这种情况下，应用将根据响应构建ACK请求，发送ACK，接着构建一个BYE transaction并发送它到call leg。应用在将ACK和BYE请求发给transaction/transport层之前，必须根据响应中的Record-Route头部域来构建ACK和BYE请求的Route头部域。
* 为这个特定的call leg创建一个Dialog（通过调用pjsip\_dlg\_fork()）。应用接着将构建和发送ACK请求到call leg，为了建立这个Dialog。在Dialog建立之后，应用可能通过发送BYE请求来终止这个Dialog。

创建交叉Dialog

应用通过调用pjsip\_dlg\_fork()函数来创建一个交叉Dialog。这个函数创建一个Dialog和执行以下步骤：

* 复制原始的Dialog真的所有属性（包括认证客户端会话）到新的Dialog
* 根据响应中To头部域的标签，分配不同的remote标签
* 注册这个新的Dialog到用户代理的Dialog集
* 如果原始的Dialog有一个应用定时器，它将复制这个定时器和更新新Dialog的定时器

注意：这个函数不会从原始Dialog复制Dialog usages（如，模块）。

在一个新的Dialog被创建之后，应用必须重新注册每个Dialog usage到这个新Dialog，通过调用pjsip\_dlg\_add\_usage()。

这个新Dialog必须作为回调函数的返回值被返回。这将导致用户代理分发这个消息到这个新的Dialog，导致Dialog usages（如，应用）接收到on\_rx\_response()的通知关于这个新的Dialog的行为。

使用定时器去处理失败的交叉Dialog

应用可以给Dialog安排指定的应用定时器通过调用pjsip\_dlg\_start\_app\_timer()函数。对于和一个Dialog相关的定时器，这个定时器比通用的定时器更完美，因为当这个Dialog被销毁时这个定时器将自动删除。

定时器对于处理失败的交叉Dialog很重要。一个交叉的早期的Dialog可能没有通过一个最终的响应来完成，因为交叉的代理在接收到2xx响应后，将不会转发300-699响应。因此终止这些早期Dialog的唯一方法就是为这些Dialog设置一个定时器。

最好的使用Dialog的应用定时器来处理失败的交叉早期Dialog的方法是，开始这个定时器在Dialog集中其他的交叉Dialog第一次接收到2xx响应时。当这个定时器超时并且没有接收到2xx响应时，这个Dialog应该被终止。

****Cseq排序****

当这个请求被发送时（与之相对应的，当这个请求被创建时），这个Dialog的本地cseq被更新。当CSeq的头部域存在在这个请求中时，这个值可能被更新为这个Dialog内这个被发送的请求。

当远端接收到这个请求时，远端的Dialog中的cseq被更新。如果Dialog的远端cseq为空，接收第一个请求将设置这个Dialog的远端cseq。对于后续的请求，当Dialog接收cseq小于Dialog记录的cseq的请求时，这个请求将被这个Dialog自动地无状态地响应一个500响应（内部服务器错误）。当这个请求的cseq大于这个Dialog的记录cseq，这个Dialog将自动更新远端的cseq（包括一个请求的cseq大于记录的cseq超过1）。RFC3261 Section 12.2.2

****Transactions****

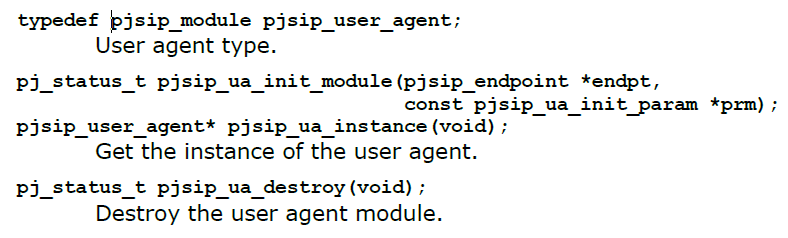
Dialog通常表现为有状态的。当有请求到来，它将自动创建UAStransaction，和当它被要求发送请求时，创建UAC transaction。

Dialog表现为无状态的唯一情况是当它接收到的请求的cseq小于当前的cseq时，这将导致以500（内部服务器错误）来响应这个请求。

当代表一个Dialog的一个transaction被创建（通过Dialog API，对UAC和UAS transaction），这个transaction的用户被设置为用户代理实例，并且这个Dialog实例将放入这个transaction的mod\_data中合适的索引下。这个索引是这个用户代理的模块ID。当事件或消息到达，这个transaction将向用户代理模块报道这个事件，这个用户代理模块将查找这个Dialog和将事件传给这个Dialog。

****基础UA API指南****

****用户代理模块API****



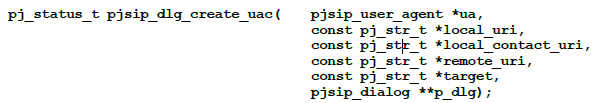
**Dialog结构**

**Dialog**结构和它的API声明在<pjsip/sip\_dialog.h>。下面的代码展示了pjsip\_diaog的声明：



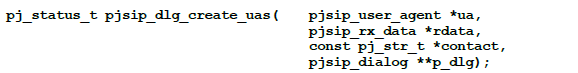
****Dialog创建API****

一个dialog可以通过调用以下任一个API来创建。



创建一个新的dialog并返回这个实例在p\_dlg参数中。在创建这个dialog之后，应用可以通过调用pjsip\_dlg\_add\_usage()增加模块作为dialog usages。

注意最初，这个dialog中的会话指针将初始化为0。



根据到来的创建一个dialog的请求（如，INVITE，REFER，或SUBSCRIBE）来初始化UAS dialog，并设置contact参数到本地的Contact中去。如果contact没有被指定，这个本地的contact将被初始化为请求的To头部域的URI。

如果这个请求有To标签参数，dialog的本地标签将被初始化为这个值。否则将生成一个全局唯一的本地标签。

如果请求中存在Record-Route头部域，这个函数也根据这个头部域将初始化dialog的路由集。

注意最初，这个dialog的会话计数必须初始化为0。

IMG_261

基于rdata中接收到的响应，创建一个新的（交叉的）dialog。这个函数拷贝original\_dlg（包括认证会话）到一个新的dialog，但是这个新的dialog将有一个从响应的To头部域中拷贝来的新的远端标签。返回时，new\_dlg将被注册到用户代理。应用只需要增加模块作为dialog的usages。

注意最初，这个dialog的会话计数必须初始化为0。

****Dialog终止****

一旦会话计数器到达0并且所有悬挂的会话已经被销毁，Dialog通常是自动被销毁。然而，这里有些情况，如果dialog usages需要早点销毁dialog，例如，当这个初始化失败时。

pjsip\_dlg\_terminate()函数被用来早点销毁dialog。这个函数通常被dialog usage调用。应用与公司使用合适的高层的API像pjsip\_inv\_terminate()来销毁这个会话和dialog。

IMG_262

销毁dialog并把它从UA模块的哈希表中注销。这个函数只有在会话计数器为0时被调用。

****Dialog会话管理API****

下面的函数被用来管理dialog的会话计数器。

IMG_263

增加dialog中会话的数量。注意最初（在创建之后）dialog已经将会话计数器设置为0。

IMG_264

减少dialog中会话的数量。一旦会话计数器到达0并且不存在悬挂的transaction，这个dialog将被销毁。注意，当这个函数被调用时，如果这里没有悬挂的transaction，这个函数可能立刻销毁这个dialog。

****Dialog usages API****

下面的函数被用来管理一个dialog中的dialog usages。

IMG_265

增加一个模块作为dialog usage，并选择性设置这个模块指定的数据。

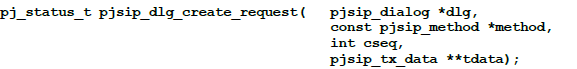
IMG_266

绑定模块指特定数据到这个dialog上。

IMG_267

得到之前绑定到这个dialog的模块特定数据。应用通过访问dlg->mod\_data[module\_id]可以直接得到值。

****Dialog请求和响应API****



创建一个基本/通用的请求，使用指定的method和选择性指定的cseq。cseq使用值

-1让dialog自动为这个请求放入下一个Cseq数。否则对于一些请求，如CANCEL和ACK，应用必须把原始的INVITE请求中的Cseq作为参数放入。这个函数也将放入Contact头部域。

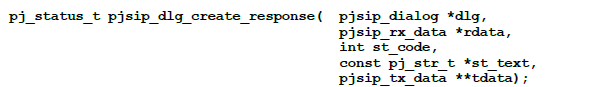
IMG_269

发送请求消息到远端。如果请求不是一个ACK请求，这个dialog将有状态地发送这个请求，通过创建一个UAC transaction和使用这个transaction发送这个请求。当请求不是ACK或CANCEL，这个dialog将增加它的本地Cseq数并更新请求中的Cseq。

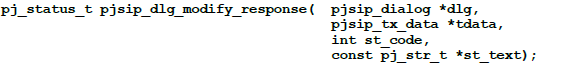
注意在这个函数返回之前dialog usages的on\_tsx\_state的回调函数可能被调用。

如果p\_tsx不是null，这个参数将被设置为发送这个请求的transaction的实例。

无论这个操作的状态如何，这个函数减少这个传输数据的引用指针。



为rdata中到来的请求创建一个响应消息，使用状态码st\_code和状态字段st\_text。这个函数和Endpoint的API pjsip\_endpt\_create\_response()不同，它可以在适当的时候在响应中加入Contact和Record-Route头部域。



使用其他状态码修改之前发送的响应。适当的时候会加入Contact头部域。

IMG_272

有状态地发送响应消息。这个transaction实例必须是on\_rx\_request()回调函数中报道的transaction。

无论这个操作的状态如何，这个函数减少这个传输数据的引用指针。

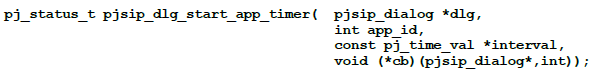
****Dialog辅助API****

IMG_273

设置dialog的初始路由集为route\_set。这个只能在任何请求发送之前，对于UAC dialog调用。在这个dialog被建立之后，这个路由集将被改变。

对于UAS dialog，这个路由集将被pjsip\_dlg\_create\_uas()从Record-Route头部域初始化。

这个route\_set参数是标准的Route头部域列表（即，带有标记）。



使用应用指定的app\_id和回调函数cb，和这个dialog开始应用定时器。应用可以只设置一个dialog一个应用定时器。这个定时器比dialog指定定时器更有用，因为它将自动被销毁一旦这个dialog被销毁。

注意这个定时器也将被拷贝到这个交叉的dialog。

IMG_275

停止应用指定定时器

IMG_276

得到到来rdata中的dialog实例。如果一个到来的消息和一个已存在的dialog匹配，这个用户代理必须已经将这个匹配的dialog实例放入rdata中，或者否则这个函数将返回NULL，如果没有找到可以匹配的dialog。

IMG_277

得到指定transaction中的dialog实例。

****例子****

****INVITE UAS Dialog****

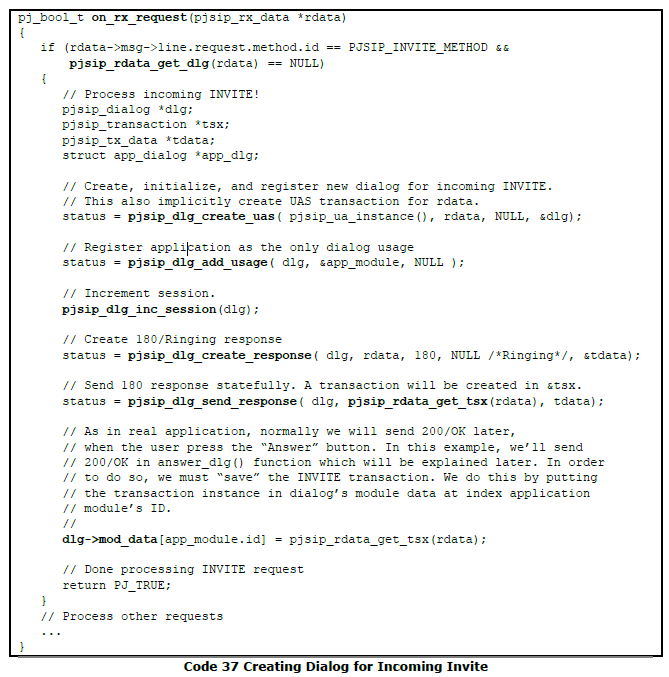
下面的例子使用基本/底层的Dialog API来处理一个到来的Dialog。这些展示如何：

* 创建和初始化到来的Dialog
* 创建UAS transaction来处理到来的INVITE请求和传输1xx响应
* 可靠地传输2xx响应到INVITE
* 处理到来的ACK

通常，大多数错误处理为了简洁被忽略。真实世界的应用应该准备处理过程中出现的所有错误情况。

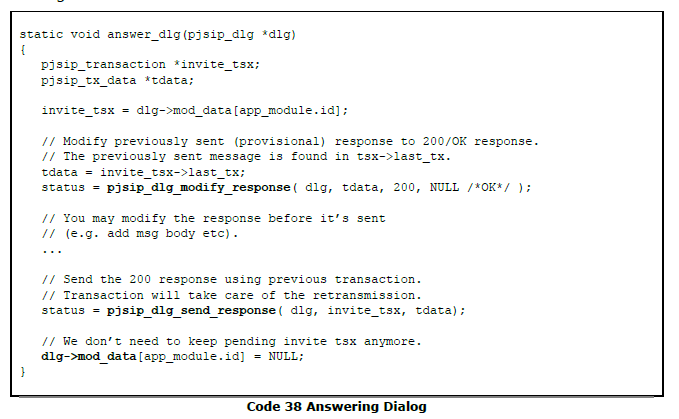
创建初始INVITE Dialog

在这个例子中，我们将了解如何为一个到来的INVITE请求创建一个Dialog和以180/Ringing临时响应响应这个Dialog。



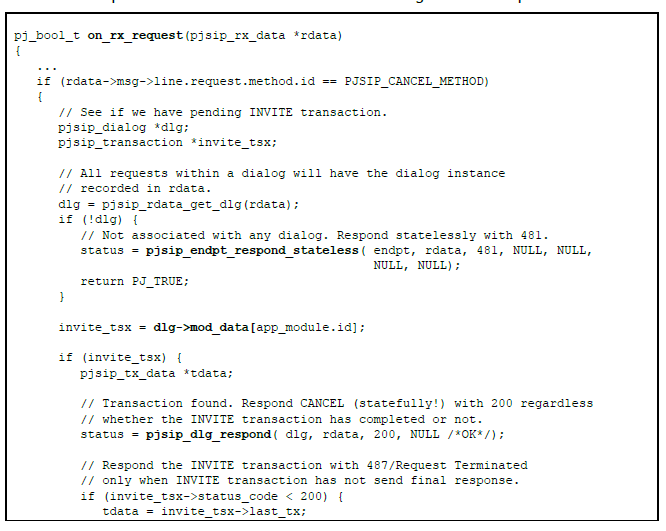
应答Dialog

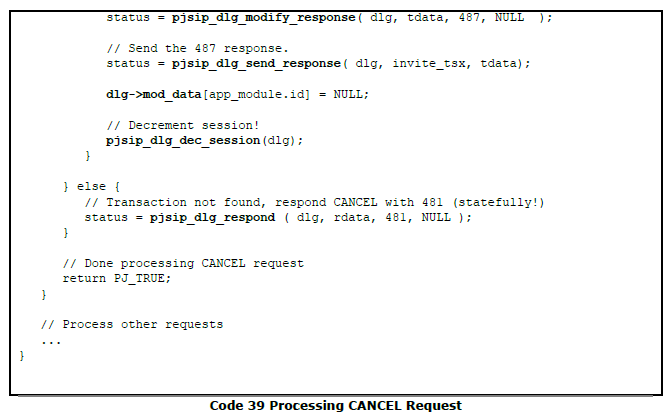
在这个例子中我们将了解如何发送200/OK响应来建立这个Dialog。



处理CANCEL请求

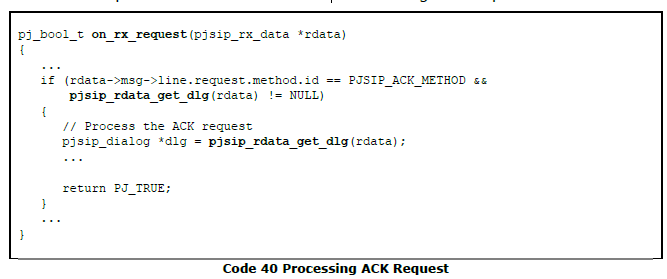
在这个例子中我们将了解如何处理到来的CANCEL请求。





处理ACK请求

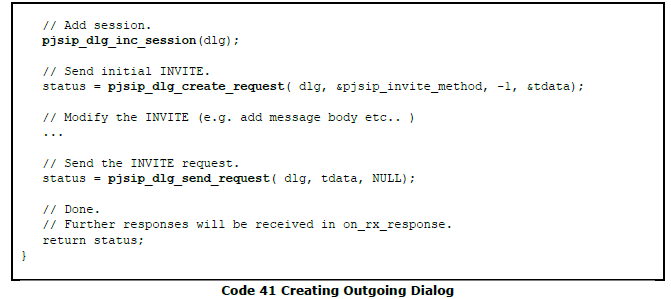
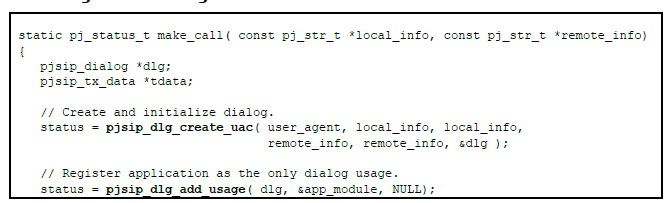
在这个例子中我们了解如何处理到来的ACK请求。



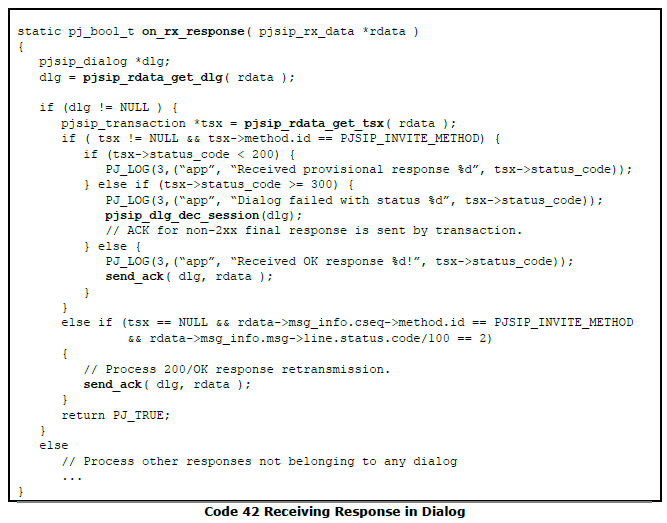
****外出的INVITE Dialog****

下面的例子们说明外出的INVITE Dialog如何处理。

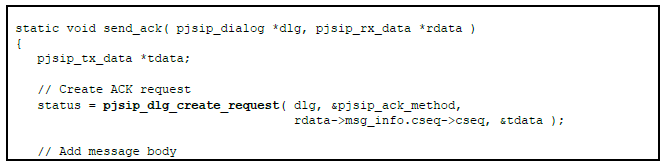
创建初始Dialog

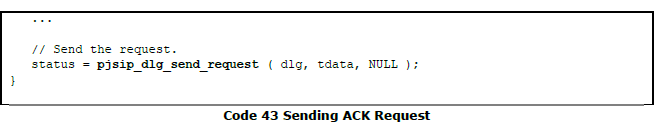


接收响应



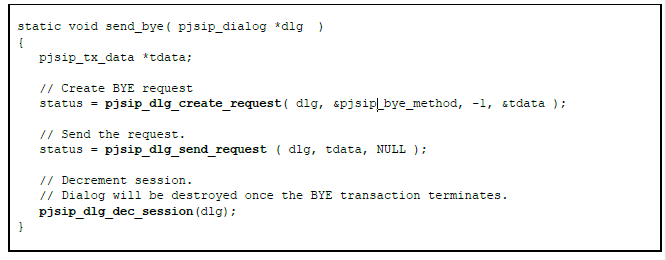
发送ACK





终止Dialog

下面的例子说明终止INVITE Dialog的一种方法，如，通过发送BYE。



# **PJSIP开发手册之SDP offer/answer框架（十一）**

****第十一章 SDP offer/Answer框架****

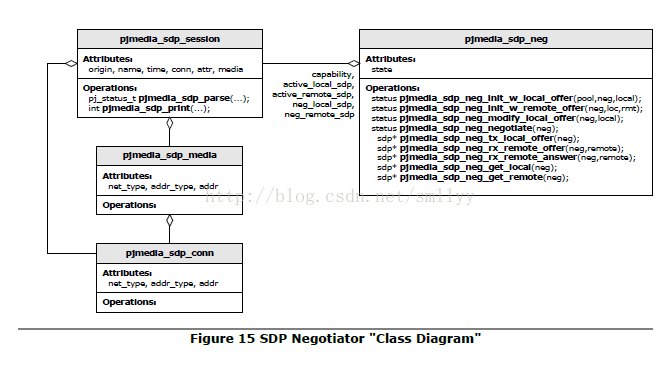
PJSIP中SDP offer/answer框架是基于RFC3264”An Offer/Answer模型使用会话描述协议（SDP）”。这个框架的主函数是为了促进本地和远端的媒体能力的协商，和在一个INVITE会话中使用哪个媒体集上达成共识。

注意尽管它主要被用在invite会话中，这个框架是基于通用SDP协商框架（pjmedia\_sdp\_neg），因此它应该可以被其他类型的应用使用。这个Dialog invite会话提供SDP offer/answer和SIP协议的整合；它正确地解释了相关消息中的消息体（如，INVITE，ACK，PRACK，UPDATE）和转化它们到SDP offer/answer协商。

这一章描述了底层SDP协商框架，它声明在<pjmedia/sdp\_neg.h>头文件。

****SDP协商结构****

pjmedia\_sdp\_neg结构代表通用的SDP offer/answer会话，并且用来协商本地和远端的SDP。



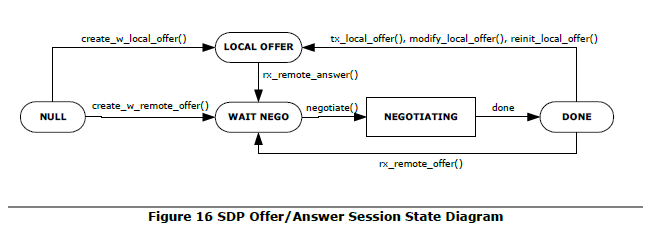
pjmedia\_sdp\_neg结构保持三种SDP结构：

* Initial\_sdp：本地Endpoint的初始能力。创建期间这个SDP将被传进协商中，并且这个内容在整个会话通常将不会改变（即使在协商之后）。当接收到远端发来的新的offer时，协商者协商时使用这个SDP（与之对应的，从远端接收到被更新的SDP）。
* active\_local\_sdp：包含和远端协商之后的本地SDP。这个Dialog必须使用它去开始本地的媒体而不是初始的SDP。
* active\_remote\_sdp：包含当前远端使用的SDP。

在协商过程中，协商者也有两个内部使用的其他SDP变量，叫做和neg\_remote\_sdp。这些是暂时的SDP描述，并且应用禁止参考这些变量。

****SDP协商会话****

SDP offer/answer会话的一般状态转化如下图：



协商会话从PJMEDIA\_SDP\_NEG\_STATE\_NULL开始。如果这个Dialog已经有一个本地媒体描述并且想要提供这个媒体给远端（通常这是当Dialog作为UAC时），它将通过把本地的SDP传给pjmedia\_sdp\_neg\_create\_w\_local\_offer()函数来创建这个SDP协商者。这个函数将设置本地Endpoint的初始能力，和设置这个协商会话的状态为PJMEDIA\_SDP\_NEG\_STATE\_LOCAL\_OFFER。接着这个初始的SDP可能被将包含在INVITE请求中发送给远端。一旦Dialog已经接收到远端的SDP，它必须调用pjmedia\_sdp\_neg\_rx\_remote\_answer()。这个协商函数接着将被调用。

如果这个Dialog已经有远端媒体描述（通常这种情况当Dialog作为UAS），它可以通过将本地和远端SDP传给pjmedia\_sdp\_neg\_create\_w\_remote\_offer()函数来创建SDP协商会话。在这之后，这个协商函数将被调用。

在这个会话被建立之后，本地和远端可能修改这个会话。协商者可能处理下面情况之一：

* 这个Dialog已经接收到远端的SDP。在这种情况下，这个Dialog将调用pjmedia\_sdp\_neg\_rx\_remote\_offer()和把这个远端SDP传给这个函数。在这个协商函数被调用之后，这个协商函数的返回值决定是否需要对本地媒体做修改。
* 想要发送SDP到远端。Dialog可以选择以下两种方式：
  1. 如果它只想不加修改直接发送目前活动的本地SDP，它应该调用pjmedia\_sdp\_neg\_tx\_local\_offer()来获取活动的本地SDP，发送这个SDP，接着等待远端应答。
  2. 如果他想要修改当前活动的本地媒体（如，改变流方向，改变活动的编解码器等），它应该使用pjmedia\_sdp\_neg\_get\_local()得到活动的本地媒体，修改它，并调用pjmedia\_sdp\_neg\_modify\_local\_offer()去更新这个offer，发送这个本地SDP，接着就等待远端的应答。
  3. 这个Dialog可能想要完整地改变本地媒体（如，改变IP地址，改变编解码集，增加新的媒体行）。这和上面的更新当前的媒体描述不同，因为它将改变initial\_sdp，所以未来的协商将基于这个新的SDP。如果这个Dialog想要做这个，它要调用pjmedia\_sdp\_neg\_reinit\_local\_offer()并将新的本地SDP作为参数，发送这个SDP，接着等待远端的应答。

在这个Dialog发送offer到远端之后，它应该接收到远端的应答。这个Dialog必须提供远端的SDP给协商者，这样这个协商函数可以被调用。这个Dialog通过调用pjmedia\_sdp\_neg\_rx\_remote\_answer()来提供这个远端的应答。

如果远端已经拒绝本地的offer（如，返回488/”Not Acceptable Here”响应），Dialog必须继续调用pjmedia\_sdp\_neg\_rx\_remote\_answer()函数并把远端SDP参数设置为NULL，和调用这个协商函数这样这个协商会话可以恢复到之前活动的会话描述。

****SDP协商函数****

在已经提供了本地和远端的SDP之后（即协商者的状态为PJMEDIA\_SDP\_NEG\_STATE\_WAIT\_NEGO），Dialog调用pjmedia\_sdp\_neg\_negotiate()来协商offer和answer。这个函数可能返回以下中的一种结果：

* PJ\_SUCCESS（即0），如果它已经成功在本地和远端SDP之间达成共识。在这种情况下，本地和远端的SDP都将保存在会话中为之后的使用，并且应用可以查询这些活动的SDP来开始本地媒体。
* PJMEDIA\_ESDPNOCHANGE，如果目前使用的SDP不需要修改（本地和远端）。在这种情况下，之前同意的SDP会话将也不会被修改
* PJMEDIA\_ESDPFAIL，如果它不能在本地个远端能力之间达成共识。在这种情况下，如果这个会话保持了一个之前已经同意的SDP，这些SDP（本地和远端）将不会被修改。如果在会话中，Dialog作为UAS它应该以488/”Not Acceptable Here”响应这个请求。
* PJMEDIA\_ESDPNOOFFER，如果协商者没有发送或接收到任何offer
* PJMEDIA\_ESDPNOANSWER，如果没有接收到任何远端的answer。
* 或者其他值指示其他错误。

在所有情况下，这个协商函数将设置这个协商者的状态为PJMEDIA\_SDP\_NEG\_STATE\_DONE。

# **PJSIP开发手册之Dialog Invite会话和Usage（十二）**

****第十二章 Dialog Invite会话和Usage****

****介绍****

Dialog invite会话是一个高层的invite会话管理，它可以被应用用来管理invite会话（包括SDP管理）。这个invite会话封装了抽象的基本Dialog，因此应该不需要使用基本Dialog的API，当它使用invite会话API时。

一个Dialog INVITE会话是可以基于每个对话被应用创建的。Dialog INVITE会话被Dialog INVITE usage管理。Dialog INVITE usage将事件从这个Dialog转发到相应的INVITE会话，并且也处理交叉Dialog。

DialogINVITE会话和usage在单独的静态库中实现，即pjsip-ua库。应用必须包括<pjsip-ua/sip\_inv.h>来使用Dialog INVITE会话/usage功能。或者应用可以只包括一个单独的头文件<pjsip-ua.h>来获得pjsip-ua库中的所有功能。

****术语****

****DialogINVITE会话****是一个在一个Dialog中的INVITE会话。如果应用决定使用高层的INVITE会话管理，它需要为每一个Dialog创建唯一一个Dialog INVITE会话实例。

DialogINVITE usage是一个注册到endpoint的PJSIP模块。当一个Dialog有Dialog INVITE session时，这个模块需要作为Dialog usage注册到特定的Dialog。这将在INVITE会话创建的时候自动实现。

****特性****

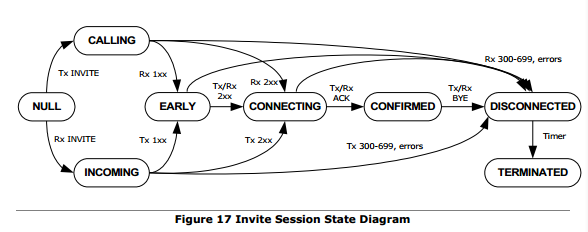
****DialogINVITE****会话提供下面的特性：

* 会话进度报告（例如，会话progressing，connected，confirmed，disconnected）
* 自动认证处理（例如，接收到401/407响应后重试这个请求）
* SDP offer和answer处理
* 高层的交叉处理
* 会话超时（即Expires头部域）
* 会话扩展，像会话定时器，和可靠的临时响应

****INVITE会话状态****

DialogINVITE usage提供回调函数来通知应用会话的进度。对于电话应用这是非常有用的，因为通常会话的状态和电话拨打的状态相关。

一个INVITE会话的进度如下状态图所示。



每个状态的描述如下：

|  |  |
| --- | --- |
| PJSIP\_INV\_STATE\_NULL | 会话第一次被创建时的状态。在状态时，没有消息已经被发送或接收。 |
| PJSIP\_INV\_STATE\_CALLING | 第一个INVITE消息发送后，在收到任何临时响应之前的会话状态。 |
| PJSIP\_INV\_STATE\_INCOMING | 接收到第一个INVITE消息后，在没有发送任何临时响应之前的会话状态。 |
| PJSIP\_INV\_STATE\_EARLY | 在Dialog已经发送或接收到INVITE请求的临时响应之后的会话状态，仅当To标签存在时。 |
| PJSIP\_INV\_STATE\_CONNECTING | 2xx响应被发送或者接收之后的会话状态 |
| PJSIP\_INV\_STATE\_CONFIRMED | ACK请求被发送或者接收之后的会话状态 |
| PJSIP\_INV\_STATE\_DISCONNECTED | 当会话已经失去连接时，或者INVITE的非成功的响应或BYE请求时的会话状态 |

****INVITE会话的创建****

对于外出的Dialog（即拨打电话的），应用需要使用pjsip\_dlg\_create\_uac()创建一个UAC Dialog。应用接着调用pjsip\_inv\_create\_uac()来创建invite会话，并将这个UAC实例作为这个函数的一个参数传入。应用禁止在invite会话被创建之前发送INVITE请求，或者否则这个INVITE会话将宽松一些事件。

对于到来的Dialog，应用可以通过调用pjsip\_inv\_verify\_request()来验证是否这个请求可以被接收。这个函数验证Supported，Require，和请求的消息体来确认这个请求是否可以被接收。如果这个请求不能被接收，应用将才拒绝的响应。如果这个请求可以被接收，应用创建通过调用pjsip\_dlg\_create\_uas()来创建UAS Dialog。应用接着调用pjsip\_inv\_create\_uas()来为这个Dialog创建invite会话，并将这个UAS Dialog实例，作为一个参数传入。应用禁止在invite会话创建之前发送任何响应，否则这个会话将宽松一些事件。

当一个外出的Dialog交叉时，并且如果在原始的Dialog中存在一个invite会话，这个invite usage将自动为这个新的交叉的Dialog创建invite会话。通过一个回调函数，应用将被调整新的会话的创建。

Invite会话创建函数（即pjsip\_inv\_create\_uac()和pjsip\_inv\_create\_uas()函数）将自动注册invite会话usage到这个dialog。应用不需要调用pjsip\_dlg\_add\_usage()来这吃这个invite usage模块到这个Dialog。

****消息处理****

Invite会话处理所有能改变会话状态的SIP方法。对于这版本的PJSIP，这个invite会话处理INVITE，BYE，ACK，CANCEL，UPDATE和PRACK方法。

应用必须使用invite会话API来创建和发送请求和响应消息。这是确保请求和响应消息被正确处理并且它包含了适当的被这个会话正在使用的特性（如可靠临时响应）必须的。

应用可以继续使用基本的Dialog API来创建和发送除了上面列出的之外的方法的请求和响应消息。例如，应用可以使用基本的Dialog API来创建和发送一个Dialog中的MESSAGE请求。

****扩展Dialog****

如上所述，invite会话处理发生在一个Dialog内的INVITE，BYE，ACK，CANCEL，UPDATE和PRACK消息。当应用想要支持或处理其他类型的消息，它必须注册作为Dialog usage注册到Dialog。这将使应用可以处理之前存在的Dialog usages不能处理到来的请求。

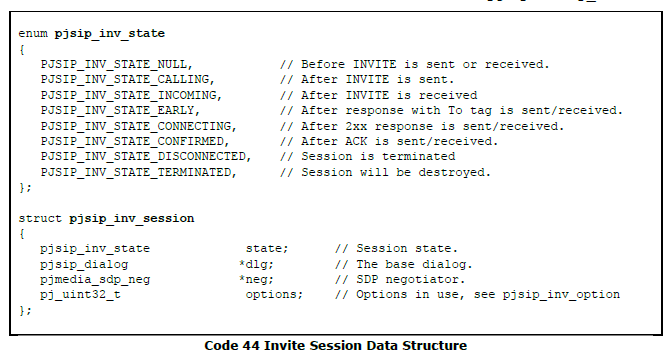
应用正确地设置它的模块的优先级。应用的优先级应该被设置成PJSIP\_MOD\_PRIORITY\_APPLICATION。Invite usage将模块优先级设置为（PJSIP\_MOD\_PRIORITY\_APPLICATION-1）。这些将确保这个invite usage可以在应用之前首先检查到来的请求。

****扩展Invite会话****

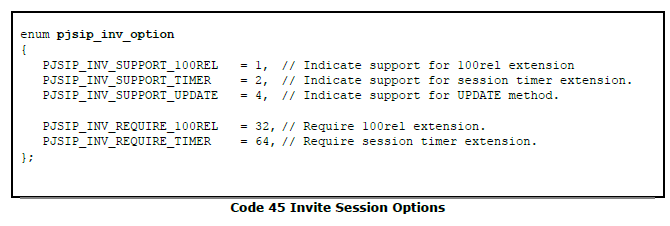
未来，invite会话可以被扩展来支持更多的SIP扩展，像呼叫转移，Dialog targetting等等。现在，应用可以通过手动地构建消息来实现这些特性。

****指南****

Invite会话功能声明在<pjsip-ua/sip\_inv.h>中。



下面代码展示了可以应用到会话中的不同的选项。当创建一个会话时，这些选项的位遮掩组合需要被指定。在一个Dialog被建立之后（包括早期），这个pjsip\_inv\_session的options展示了Endpoint的共同的能力。



****Invite usage模块****

invite usage模块必须在任何invite会话被创建之前被初始化。

IMG_259

初始化invite usage模块并把它注册到Endpoint。Callback参数包含在invite会话中事件发生时将会调用的函数的指针。

IMG_260

得到invite usage模块的实例。

****会话回调****

pjsip\_inv\_callback结构包含函数指针，这些指针将被应用注册到invite usage模块用来接收invite会话事件的通知。

回调中的函数如下。

IMG_261

这个回调在invite会话状态改变时被调用。应用应该检查会话的状态（inv\_sess->state）来获取当前的状态。

这个回调是强制的。

IMG_262

这个回调在当外出请求交叉时，invite usage模块创建一个新的Dialog和invite会话后被调用。

这个回调是强制的。

IMG_263

这个回调在会话内的transaction发生状态改变时被调用。应用可以实现这个回调，如去检测一个外出消息的进度。

这个回调是可选的。

IMG_264

这个函数在invite会话接收到一个新的offer时被调用。应用使用pjsip\_inv\_set\_sdp\_answer()来设置本地answer。这个函数将不会发送外出消息。它只是为SDP协商过程保持这个answer，并且被包含在后续的响应或消息里被发送。

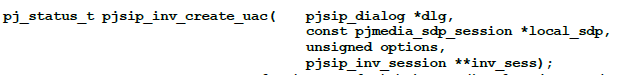
这个回调是可选的。当它没有指定，默认的行为是使用会话的初始能力来协商远端offer。

IMG_265

这个函数在SDP offer/answer会话完成之后被调用。Status参数指定了offer/answer的状态，pjmedia\_sdp\_neg\_negotiate()返回的。

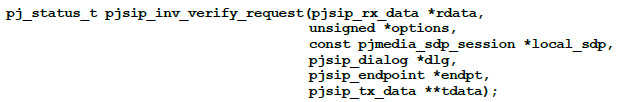
这个回调函数是可选的（从框架的角度来看），但是所有有用的应用通常需要实现这个回调。

****会话的创建和终止****



为指定的dlg创建UAC invite会话。如果应用已经决定它的媒体能力，它可以指定local\_sdp中的SDP。否则它将设置为NULL，让远端的UAS指定一个offer。Options参数是pjsip\_inv\_options的位遮掩组合。

返回成功时，这个invite会话将被放入inv\_sess参数并且这个函数将返回PJ\_SUCCESS。否则将返回适当的错误码。



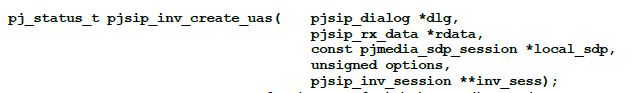
应用应该在接收到最初的rdata的INVITE请求之后，在invite会话创建之前（或者Dialog）调用这个函数来验证这个invite会话是否能处理这个INVITE请求。这个函数验证本地Endpoint是否能处理请求中需要的SIP扩展（即Require头部域）和媒体（如果媒体描述在消息中存在）。

当调用这个函数时，options参数应该包含想要应用到会话中的扩展。当返回时，这个参数将包含在考虑了Supported，Require，和Allow头部域之后将被应用到会话中的SIP扩展。

如果本地媒体能力已经被决定，并且应用想要验证它是否能处理到来的INVITE请求中的媒体offer，它应该验证local\_sdp中的本地媒体能力。如果它没有被指定，媒体验证将在这个函数中不会被执行。

如果所有事都协商成功后，这个函数将返回PJ\_SUCCESS。否则它将返回失败的原因。

这个函数当验证失败时是可以创建合适的响应消息的。如果tdata被指定，当验证失败时那么一个非2xx最终响应将被创建并返回时会放入tdata参数中。如果在调用这个函数之前一个Dialog已经被创建，那么它必须被指定在dlg参数中。否则应用必须指定endpt参数（这是很有用的，如当应用想要无状态地发送这个响应时）。



创建指定dlg的UAS invite会话。应用必须指定接收到的rdata中的INVITE请求。这个invite会话需要检测这个接收到的请求来看这个请求是否包含它支持的特性。

应用应该在调用这个函数之前调用验证函数，来确保可以成功创建这个会话。

如果应用已经决定它的媒体能力，它能指定这个能力在local\_sdp。如果在初始的INVITE中接收到的SDP，这个UAS能力指定在local\_sdp不需要匹配这个接收到的offer；SDP协商者能够重新排列answer中的媒体行来匹配这个offer。

Options参数是pjsip\_inv\_options中的SIP特性的位遮掩。

返回成功时，这个invite会话将被放入inv\_sess参数并且这个函数将返回PJ\_SUCCESS。否则将返回适当的错误码。

IMG_269

早点终止INVITE会话和销毁下层的Dialog（如果这个Dialog没有其他usage）。这个函数应该只有当INVITE会话初始失败时被调用。通常情况下，应用必须通过调用pjsip\_inv\_end\_session()来终止这个INVITE会话。

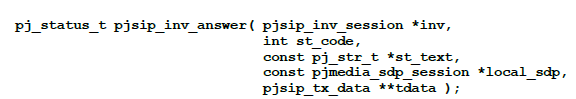
St\_code参数指定了失去连接的原因的SIP状态码。如果notify为true，应用的回调将被调用。

****会话操作****

IMG_270

为会话创建初始的INVITE请求。这个函数只能被UAC会话调用。如果创建成功，初始的INVITE请求将被放入tdata参数中。

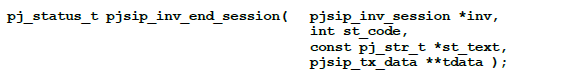
如果invite会话创建之前，本地媒体能力被指定，这个函数将在外出的INVITE请求中放入一个SDP offer。否则外出的请求将不包含SDP体。



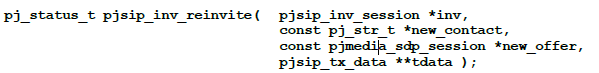
为INVITE请求创建一个响应消息。St\_code包含要发送的临时或最终响应的状态码。如果自定义了状态文本，应用可以指定st\_text；否则如果这个参数为NULL，默认的状态文本将被使用。

如果应用在UAS invite会话创建期间已经指定了它的媒体能力，local\_sdp参数必须为NULL。这是因为应用不能在一个INVITE transaction中执行多于一个SDP offer/answer。

如果应用在UAS invite会话创建期间没有指定它的媒体能力，它可能或必须在local\_sdp中指定它的能力，并依据st\_code来判断是否是一个2xx最终响应。



创建一个SIP消息来初始化INVITE会话的终止。根据会话的状态，这个函数可能返回CANCEL请求，一个非2xx响应，或者一个BYE请求。如果这个会话没有被到来的INVITE应答，这个函数创建这个非2xx最终响应使用指定的st\_code和st\_text。



创建一个re-INVITE请求。如果应用想要更新它的本地Contact和通知对端刷新关于这个新联系人，它能指定这个新的联系人在new\_contact中，否则这个参数必须为NULL。

当没有悬挂的answer被发送或接收时，应用可以初始化请求中的一个新的SDP offer/answer会话。它能通过观察SDP协商者的状态来检测这个条件。如果新的offer应该被发送到远端，这个offer必须在new\_offer中指定，否则这个参数必须为NULL。

IMG_274

发送tdata中的请求或响应消息。token是将被放入transaction的mod\_data数组对应的模块索引下的任意应用数组。

****辅助API****

****IMG_275****

获取与dlg相关的invite session实例，或者NULL。

IMG_276

获取与tsx相关的invite session实例，或者NULL。

# **PJSIP开发手册之SIP事件通知（十三）**

****第十三章 SIP特定的事件通知****

SIP事件特定的通知在RFC3265“Session Initiation Protocol-SpecificEvent Notification”描述。这个核心协议定义了建立事件订阅的两种SIP方法，即SUBSCRIBE和NOTIFY，尽管其他方法也可以被定义来建立订阅（如REFER）。

这章描述了PJSIP的设计和基于基本的Dialog框架来创建基本和通用的事件通知框架的实现。这章可以用来实现高层的事件包如presence和call transfer（使用REFER）。

PJSIP的事件通知框架的实现被打包成一个静态库pjsip-simple，在pjsip目录下。为了使用它的功能，应用应该包括头文件<pjsip\_simple.h>和连接到pjsip-simple静态库。

****基本概念****

所有类型的JSIP事件通知会话表示为pjsip\_evsub对象。这个对象管理订阅的生命周期，和转化到来的请求和响应到适合的回调函数的调用。

PJSIP事件通知会话使用基本的Dialog框架来管理下层的Dialog和维护Dialog属性（像请求目标，CSeq排序，等）。因为基本的Dialog的设计允许Dialog被多个会话共用，多个事件订阅会话可能使用相同的Dialog，并且它也可以和invite会话共用这个Dialog。

为了订阅一个事件通知，应用需要创建一个事件订阅对象，指定下层的Dialog和接收订阅事件的回调函数。

到来的订阅请求（如SUBSCRIBE或REFER）将到达一个Dialog或者应用，根据这个请求是否在一个Dialog内。应用必须检测请求中的Event id并使用合适的包的API来处理订阅。例如，当请求是REFER，应用调用pjsip\_xfer\_create\_uas()来创建服务端订阅。当SUBSCRIBE请求中的Event id是“presence“时，应用调用pjsip\_pres\_create\_uas()来创建服务端订阅。

****事件包****

事件包描述了事件订阅的语义。在PJSIP中，在带有指定的事件ID的会话被创建之前事件包必须首先被注册到事件框架上。通过调用pjsip\_evsub\_register\_pkg()可以把事件包注册到这个框架。这个函数通常当这个PJSIP模块实现的这个事件包被初始化时。

这个事件包主要负责向NOTIFY请求提供消息体。例如，PJSIP presence事件包创建所有外出的NOTIFY请求消息体使用内容类型”application/pidf+xml”或者”application/xpidf+xml”。

****头部域****

事件框架基于已注册的事件包提供的信息来管理外出请求的Accept，Allow-Events，Event，Expires和Subscription-State头部域的内容。它也检查到来请求中的Expires和Subscription-State头部域的内容，并相应地更新它自己的状态。

所有其他头部域（和Dialog外的头部域）必须被事件包或者应用处理。例如，refer事件包管理外出的REFER请求的Refer-To头部域。

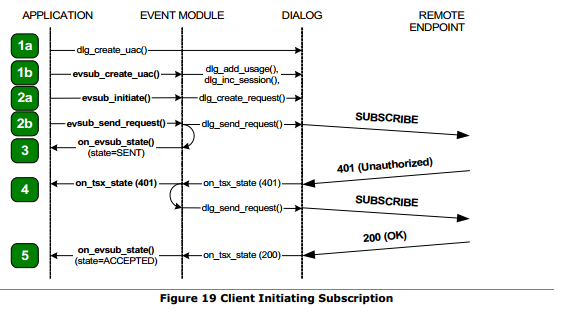
****基本的操作****

这个会话描述如何使用核心的PJSIP事件订阅框架。正如你将在后面章节见到，高层的事件包（如presence和call transfer）的操作，将和核心事件框架的操作类似。

注意：为了节省空间，图中省略了”pjsip\_”头。

****客户端初始化订阅****

客户端通过构建和发送SUBSCRIBE请求建立Dialog来初始化订阅。客户端应该将合适的证书放入这个Dialog中，这样认证的挑战可以被evsub模块自动处理。客户端应该在Dialog中设置合适的路由集。



描述：

1.        应用（或者事件包）初始化客户端订阅通过先创建一个UAC Dialog(1a)，接着创建这个客户端订阅会话(1b)。应用可以1a和1b步骤之间设置Dialog的证书和路由集。

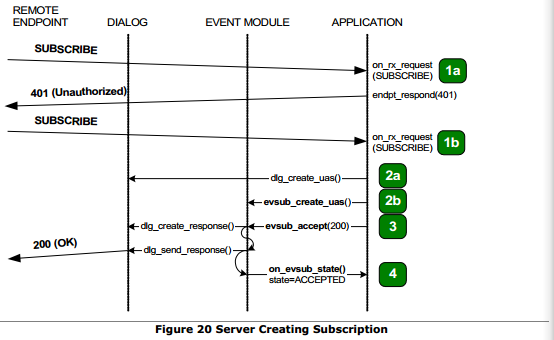
2.        通过创建请求(2a)和发送请求(2b)，应用发送初始的SUBSCRIBE（或者其他建立订阅的方法，像REFER）。

3.        上面2步骤中的SUBSCRIBE请求的发送将触发on\_evsub\_state()回调函数被调用。这甚至可能在evsub\_send\_request()函数返回之前发生。

4.        步骤4中应用接收on\_tsx\_state()中的任何transaction状态的进度。这个回调函数是可选的，并且只为提供信息。如果这个请求被挑战，并且证书已经在这个Dialog里设置了，这个事件框架将使用正确的证书提交这个请求。

5.        当接收到2xx响应，on\_evsub\_state()回调将被调用。应用可以通过调用pjsip\_evsub\_get\_state()函数得到这个订阅状态，当接收到2xx响应时，这个函数返回PJSIP\_EVSUB\_STATE\_ACCEPTED。当接收到非2xx最终响应时，这个订阅状态将被设置为PJSIP\_EVSUB\_STATE\_TERMINATED。

****服务端接收到来的订阅****

********

描述：

1．会话外的到来请求将总会到达应用，应用最终决定如何处理这个请求。对于到来的SUBSCRIBRE请求，如果应用想要认证这个请求，它可以有状态或无状态地发送401/407响应（步骤1a）来响应这个请求。这必须在任何Dialog或者订阅实例被创建之前完成。当应用满意这个请求（步骤1b），它可以接着创建这个服务端订阅实例（步骤2a和2b）。

如果这个请求在一个Dialog内（例如，REFER请求），应用接收这个Dialog的回调函数

on\_tsx\_state()中的请求。在这种请况下，步骤1到2a是不需要的，并且应用直接执行步骤2b。

2．服务端的事件订阅需要一个Dialog。应用可能为这个订阅创建一个新的UAS Dialog（步骤2a），或者可能使用现有的Dialog（例如处理Dialog内的到来REFER请求）。应用接着创建服务端的事件订阅（步骤2b），并传入这个Dialog实例。

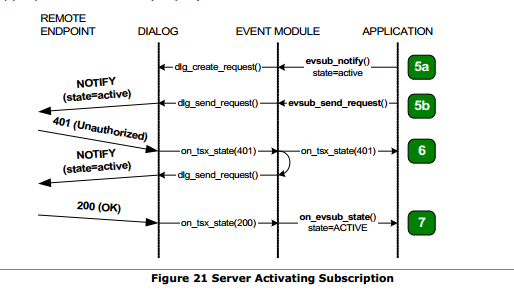
3．应用调用pjsip\_evsub\_accept()来发送响应（步骤3），并将传入的状态码放入这个响应中。这个状态码必须是2xx的。

4．步骤3中发送2XX响应将触发on\_evsub\_state()回调函数的调用（步骤4）。

服务端接着必须立刻发送初始的NOTIFY请求，下面将会介绍。

服务端激活订阅（发送NOTIFY）

服务端激活服务端订阅通过发送NOTIFY请求（见下面的第5步）。如果服务端想要NOTIFY请求被认证，那么它必须在创建UAS Dialog时设置证书。如果这个NOTIFY请求是一个挑战，那么只要这个Dialog中有一个正确的证书，那么evsub模块将自动重新使用合适的证书提交NOTIFY请求（步骤6）。

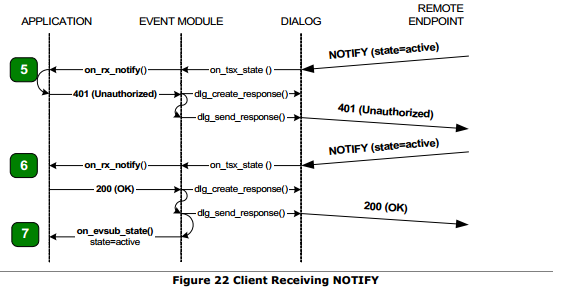


****客户端接收NOTIFY请求****

当接收到NOTIFY请求，应用可以通过在on\_rx\_notify()回调函数中返回401来挑战这个请求（见下面步骤5）。客户端接着将等待NOTIFY请求的立即提交。默认地，订阅框架等待NOTIFY请求重提交5秒，如果到了5秒还没有收到NOTIFY请求它将发送带有0 Expires值的SUBSCRIBE请求来终止这个订阅。

on\_rx\_notify()回调函数是可选的。默认的是以200响应响应到来的NOTIFY请求。

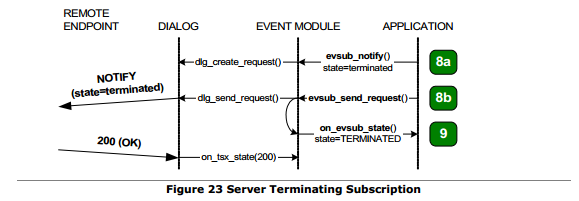
注意如果应用应答这个NOTIFY请求以2xx响应的话，事件框架只更新它的状态（根据到来的NOTIFY请求中的状态）。



****服务端终止订阅****

服务端终止订阅通过发送带有Subscription-State为terminated的NOTIFY请求（下面步骤8）。服务端可能在它接收到建立这个会话的初始请求之后的任何时间发送NOTIFY请求。

特别地，当这个订阅超时后，服务端应该发送Subscription-State为terminated的NOTIFY请求。

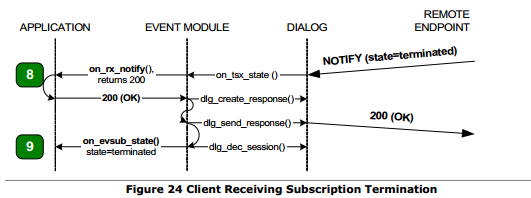


无论响应是什么，只要发送带有Subscription-State为terminated的NOTIFY请求将触发on\_evsub\_state()回调函数的调用（步骤8b）。然而，当这个NOTIFY请求被挑战时，这个框架将通过重新发送带有合适的证书的请求来响应这个挑战（如果这样的证书存在）。

注意：另外，接收到481（Call/TransactionDoes Not Exist），408（Request Timeout），transaction超时，或者transport错误事件时也将终止这个服务端订阅，并会触发on\_evsub\_state()回调函数。

****客户端接收订阅终止****

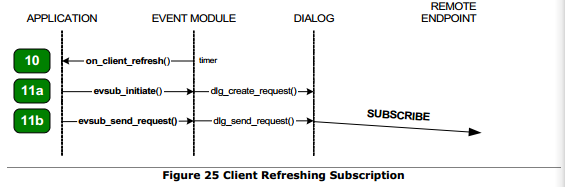
当接收到发送带有Subscription-State为terminated的NOTIFY请求，只有客户端给服务端的响应是2xx时，on\_evsub\_state()回调函数将被调用（下面步骤8）。



****客户端刷新订阅****

当要刷新这个订阅时，事件框架将触发on\_client\_refresh()回调函数。应用必须通过调用pjsip\_evsub\_initiate()函数创建请求和pjsip\_evsub\_send\_request()发送请求来刷新订阅。

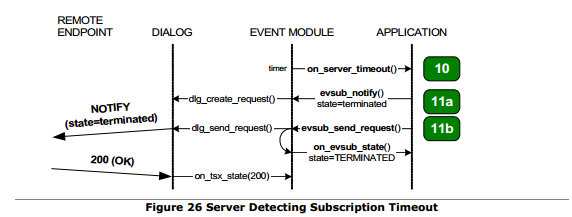
当PJSIP事件包中的presence或refer被使用时，这些包为这个回调函数提供了默认的实现。这个默认的实现使用了最近的Expires值。因此如果应用正在使用这些包，它不需要实现这个回调。



****服务端检查刷新超时****

当订阅的expires期间内没有接收到订阅的刷新，服务端订阅将触发on\_server\_timeout()回调。应用必须通过发送终止状态的NOTIFY来终止订阅。

当PJSIP事件包中的presence或refer被使用时，这些包为这个回调函数提供了默认的实现。这个默认的实现通过带发送终止状态和上次消息体的NOTIFY请求来终止订阅。因此如果应用正在使用这些包，它不需要实现这个回调。



****指南****

****模块管理****

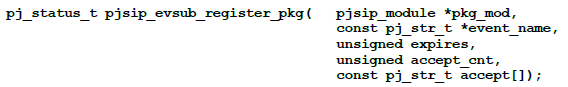
IMG_264

初始化事件通知模块并注册这个模块到指定的Endpoint。这个函数必须在任何其他的事件订阅函数之前被调用。

IMG_265

获取事件通知模块的实例。

****事件包管理****

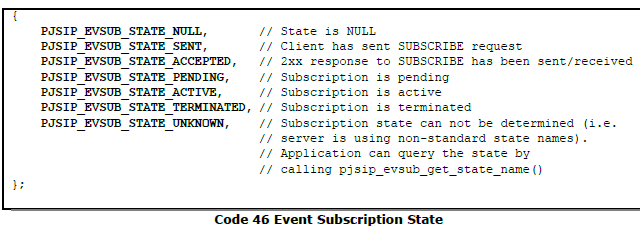


注册事件包到事件订阅框架。pkg\_mod参数指定了事件包注册到的模块。event\_name指定了事件包的名字，如“presence”（RFC3856）。accept\_cnt和accept参数指定描述可接受的媒体类型的数组。Presence包可接收的媒体类型比如“application/pidf+xml”和”application/xpidf+xml”。

****事件订阅状态****

事件订阅状态表示为pjsip\_evsub\_state中的值，定义在<pjsip-simple/evsub.h>如下：

IMG_267



注意：

* 当状态达到PJSIP\_EVSUB\_STATE\_TERMINATED时，应用必须释放与订阅相关的资源，因为在这之后订阅将被销毁并且将没有任何通
* PJSIP\_EVSUB\_STATE\_UNKNOWN发生当服务器发送Subscription-State头部域无法识别。

****事件订阅会话****

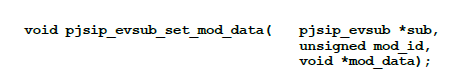
PJSIP中的事件通知会话表示为pjsip\_evsub结构。下面的函数用来查询这个结构的属性。

IMG_269

获取事件订阅状态。

IMG_270

返回代表状态的字符串，或者当状态是PJSIP\_EVSUB\_STATE\_UNKNOWN时，返回服务器发送的状态字符串。



把用户定义的数据mod\_data放进mod\_id位置中。

IMG_272

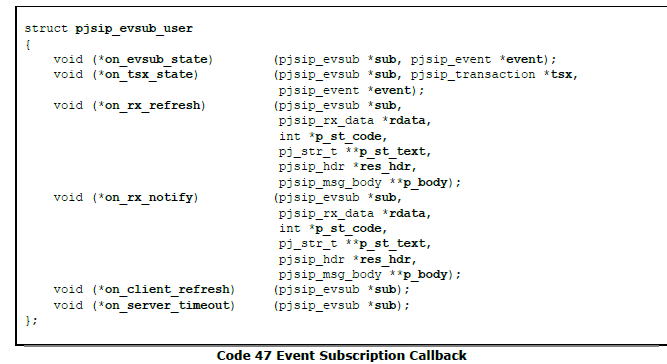
恢复之前设置在mod\_id位置的用户定义数据。

****通用的事件订阅回调****

通用的事件订阅回调用户包含用来从事件框架或者正在使用的事件包接收到通知的函数回调。

当应用正在使用一个包，应用通常要为这个包注册这个回调函数，而不是注册回调到事件框架。

通用的事件订阅回调函数声明如下。

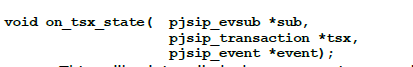


每个回调函数的描述如下：

IMG_274

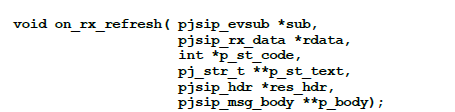
这个回调函数当订阅状态改变时被调用。应用必须准备接收NULL事件和其他类型的除了PJSIP\_EVENT\_TSX\_STATE的事件。

这个回调函数是可选的，虽然通常应用想要实现这个函数。



这个回调当transaction状态改变时被调用，对于属于这个订阅的transaction（即带有可以创建订阅和NOTIFYtransaction的请求）。

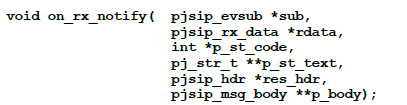
这个回调是可选的，因为它只用来提供信息。



这个回调当到来的SUBSCRIBE（或者任何第一个建立这个订阅的方法）被接收之后被调用。它允许应用指定发送什么响应，并且要放什么附加的头部域和消息体。

当应用正在使用PJSIP的事件包像presence或call transfer时，这个回调是可选的；这个包的默认实现是发送200（OK）和包含当前订阅状态的NOTIFY。

然而，如果应用实现这个回调（即回调的值不为NULL），它必须当接收到这个回调时发送NOTIFY请求。建议调用pjsip\_evsub\_last\_notify()来创建NOTIFY请求，因为这个函数考虑未订阅的请求和计算合适的过期时间间隔。



这个回调当客户端/订阅用户接收到到来的NOTIFY请求。它允许应用指定发送什么响应，并且要放什么附加的头部域和消息体。

这个回调是可选的，当它没有实现时，默认的是发送200（OK）来响应到来的NOTIFY请求。

IMG_278

这个回调当客户端刷新这个订阅被调用。

当应用正在使用PJSIP的事件包像presence或call transfer时，这个回调是可选的；这个事件包将通过发送时间间隔设置为当前/最近时间间隔的SUBSCRIBE请求来刷新订阅。

然而，如果应用实现这个订阅（即回调的值不为NULL），它必须自己发送刷新订阅。

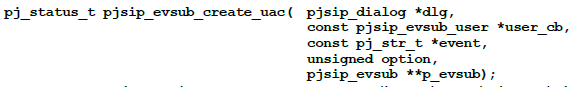
IMG_279

这个回调当服务器在指定的订阅时间段之后没有接收到订阅刷新时被调用。

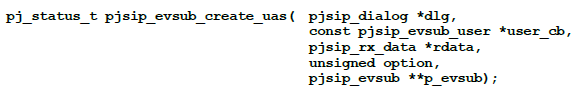
当应用正在使用PJSIP的事件包像presence或call transfer时，这个回调是可选的；事件包将发送NOTIFY来终止这个订阅。

然而，如果应用实现这个订阅（即回调的值不为NULL），它必须自己处理超时，并且建议发送状态设置为终止的NOTIFY。

****事件订阅API****



创建客户端订阅会话，使用dlg作为下层的dialog。event参数指定要使用的事件包，并且这必须在之前就已经注册到这个事件框架了。option参数当前只被refer订阅使用，并且其他类型的包这个参数应该设置为0。

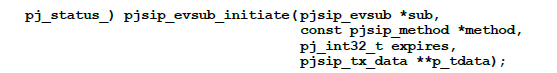


创建服务端订阅会话，，使用dlg作为下层的dialog。rdata参数指定到来的请求。option参数当前只被refer订阅使用，并且其他类型的包这个参数应该设置为0。

IMG_282

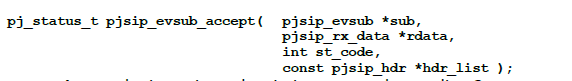
强制销毁事件订阅。这个函数应该只在初始化失败时被调用。对于通常情况，订阅在终止状态时被自动销毁。

当dialog没有其他usage时，这个函数可能销毁下层的dialog。

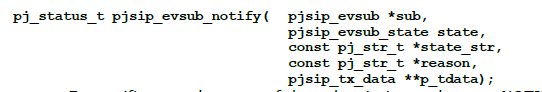


调用这个函数来创建请求来初始化订阅，刷新订阅，或请求订阅终止。method参数必须是建立订阅的方法，像SUBSCRIBE或REFER。如果参数为NULL，那么SUBSCRIBE将被使用。expires参数将被放入请求的Expires头部域。如果这个设置为0，这将是取消订阅的请求。如果这个值是负数，默认的包中定义的过期时间间隔将被使用。

应用接着必须调用pjsip\_evsub\_send\_request()来发送订阅请求。



发送2XX响应给SUBSCRIBE或REFER请求来接受到来的订阅请求。st\_code参数必须指定2xx。在hdr\_list是可选的放入响应中的头部域列表。



对于通知者，设置订阅的状态和创建NOTIFY请求给订阅者。state\_str参数是可选的，它只被用在当这个状态设置为PJSIP\_EVSUB\_STATE\_UNKNOWN。当订阅的状态设置为PJSIP\_EVSUB\_STATE\_TERMINATED时，reason参数必须被设置。应用应该使用RFC3265中定义的值，如“noresource”，“timeout”，“giveup”，”rejected”，”probation”和”deactivated”。PJSIP不翻译这个原因字符串，它将只是把这个字符串放到外出的NOTIFY请求中。

注意当NOTIFY请求被发送时，这个订阅的状态将被设置。

IMG_286

对于通知者，创建一个NOTIFY请求来反映当前订阅的状态。这个函数通常被包实现者使用，而不是直接被应用使用。

IMG_287

发送tdata中之前创建的外出请求。

****辅助API****

IMG_288

获取和指定的transaction相关的事件订阅实例。

# **PJSIP开发手册之Presence事件包（十四）**

****第十四章 Presence事件包****

****介绍****

Presence相关的SIP定义在RFC3856” A Presence Event Package fortheSession Initiation Protocol (SIP)”。这个presence事件包允许一个Endpoint来订阅一个URI的presence状态（如buddy）。

这章描述了PJSIP的presence事件包的设计和实现。这个实现使用了PJSIP的通用事件订阅框架，和使用事件名“presence”注册一个事件包。

PJSIP的presence实现打包为一个静态库pjsip-simple，在pjsip目录下。为了使用这个功能，应用应该包括<pjsip\_simple.h>头文件并连接pjsip-simple静态库。

****指南****

这个presence API和核心事件API相似，并且行为也是一致的。请查看头文件<pjsip-simple/presence.h>获取更多信息。

应用必须在任何使用presence功能之前调用pjsip\_pres\_init\_module()。这个函数支持这个presence模块Endpoint，并且也注册事件包“presence”到事件框架。

# **PJSIP开发手册之Refer事件包（十五）**

****第十五章 Refer事件包****

refer事件包声明在<pjsip-ua/sip\_xfer.h>。应用必须在它使用它的这些功能之前调用pjsip\_xfer\_init\_module()。这个函数注册mod-xfer模块到Endpoint，并注册refer事件包到事件框架。

这个refer事件包API和核心事件API相似。

# **PJSIP开发手册之即时消息（十六）**

****第十六章 即时消息****

PJSIP基于即时消息可以被用来促进pager，描述在RFC3428（Session InitiationProtocol(SIP)Extension For Instance Messaging）。

另外，PJSIP支持消息组合如RFC3994（Indication of Message Compositionfor Instant Messaging）所述。

****即时消息****

应用通过构造一个MESSGAE请求来发送即时消息。PJSIP没有定义一个指定的API来组合MESSAGE请求，因为基本上这个流程和创建其他类型的请求相同并且MESSAGE请求不需要特殊的处理。

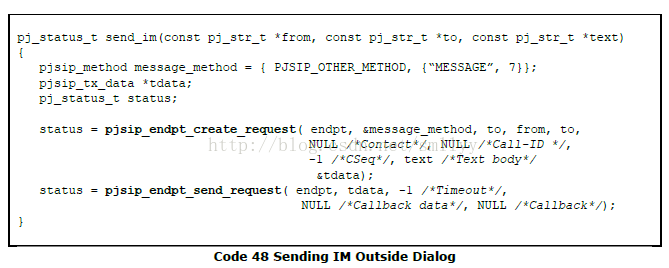
应用可以选择在一个dialog内还是外发送MESSAGE请求。例如，当一个语音交流的INVITE会话已经被建立，MESSAGE请求可能dialog内被交换。然而，RFC3428明确规定实现时不应该为了将MESSAGE请求和另外一个dialog关联而创建dialog。

****发送消息****

Dialog外

为了发送dialog外的MESSAGE请求，应用通常构建一个新的请求，调用pjsip\_endpt\_create\_request()。这个函数可以接受一个”text/plain”的消息体。应用接着调用pjsip\_endpt\_send\_request()来有状态地发送这个请求。

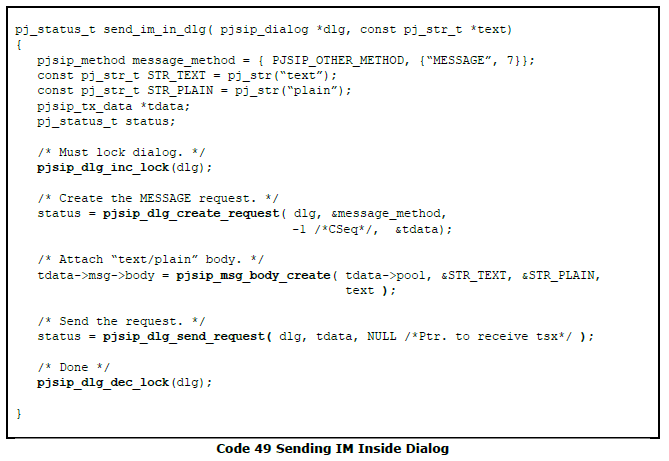
下面代码片段展示了如何实现。



Dialog内

为了发送dialog内的MESSAGE请求，应用通常构建一个dialog内的新的请求，调用pjsip\_dlg\_create\_request()。应用接着绑定一个”text/plain”的消息体到请求，并调用pjsip\_dlg\_send\_request()来发送这个请求。

下面代码片段展示了如何创建一个dialog内的MESSAGE请求。



****接收消息****

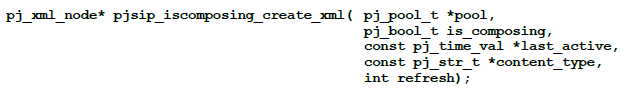
Dialog外的到来MESSAGE请求将被应用模块接收。

Dialog内的到来MESSAGE请求将通过这个dialog的回调函数on\_tsx\_state()通知给dialog usage。

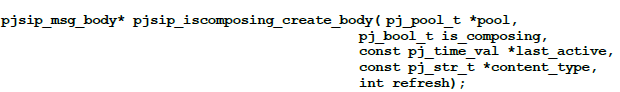
****消息组成指示****

PJSIP SIMPLE静态库提供构成和解析消息组成指示消息体的助手。消息组成指示助手函数声明在<pjsip-simple/iscomposing.h>头文件中。应用可以包含这个头部域，或者包含< pjsip-simple .h>来包含SIMPLE的所有功能。

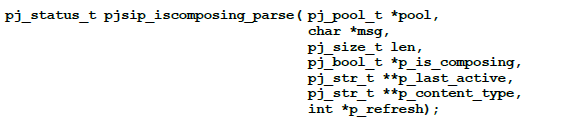
PJSIP消息组成指示头文件声明如下函数：



创建XML文件服从“application/im-iscomposing+xml”的说明。唯一需要的参数是pool和is\_composing状态。其他参数是可选的加入这个XML文件的属性。last\_active属性指示了上次键入的时间。将这个参数赋值为NULL来忽略它。content\_type参数指定了正在组成的消息类型。将这个参数赋值为NULL来忽略它。Refresh参数指示接受者可以期望发送者来刷新这个状态。将这个参数赋值为-1来忽略它。



创建一个SIP消息体包含消息组成指示的XML文件。



解析一个包含消息组成指示的XML文件的缓冲。文件中的值将返回在p\_is\_composing, p\_last\_active,p\_content\_type, 和p\_refresh参数中，这些都是可选的。