# SIP协议基本概念

1）使用UTF-8编码的文本协议；

2）协议分为请求消息和响应消息两种。请求消息为C2S，应答消息为S2C；

3）除选用的字符集（UTF-8）和语法定义外，消息的结构样式采用RFC2822（可联想http报文那种样式）；

4）SIP协议和HTTP协议的样式很像，包括很多字段的含义和可选值都一样，但是SIP和HTTP都是应用层协议，相互不存在任何依赖关系；

# **SIP协议报文格式**

起始行

N个头字段

可选的消息体

* 起始行：

请求报文：

Request-Line = Method Request-URI SIP-Version CRLF

响应报文：

Status-Line = SIP-Version Status-Code Reason-Phrase CRLF

* N个头字段

基本上和HTTP的头字段一致，参见RFC 2616和RFC2822。

头字段是分类的，有些只服务于请求报文，有些只服务于响应报文，当然也有二者都服务的字段，如果把字段用在了不恰当的地方，协议会自动忽略这个字段的存在。

* 可选的消息体

略，类似于HTTP的格式

# **SIP消息帧**

可采用任何非可靠协议（UDP，UIPC）作为消息帧的载体。

# **用户代理（UA/user agent）**

所谓的用户代理，是指SIP协议栈的代理者，这里一般指使用SIP协议的应用层软件，UA分为两种：UAC（UA client）和UAS（UA sever）。

UAC可以根据某些外部激励组装请求报文，并且可以接收对应的响应报文。

UAS接收UAC的请求报文，把处理结果反馈给UAC。

UAC和UAS之间还存在鉴权机制，消息体使用S/MIME实现加解密。

# **UAC的消息结构和处理方法（请求 & 应答）**

* 请求消息的结构

前面说到了消息的基本组成：起始行+N个头字段+报文体。

UAC请求报文格式简介：

起始行：遵从通用格式，Method Request-URI SIP-Version CRLF；

N个头字段：至少应该包含如下：**To**、From、CSeq、Call-ID、Max-Forwards、Via、Contact、Supported、Require。

报文体：由MIME编码，内容视情况而定。

* 请求消息的发送

RFC3263

* 应答消息的处理

应答消息先由传输层（IP）处理，然后上传到事务层（UDP、TCP），然后由应用层（SIP）处理具体的消息。

事务层错误：

某些情况下，事务层处理后反馈给应用层的并不是SIP格式的消息，而是事务层处理的结果。比如，当事务层超时错误时，返给应用层的就是408状态码；如果发生了严重错误，那可能返回503状态码。

无法识别的应答消息：

必须有相应的处理机制，而不是当做没发生

应答头中有多个Via

非法应答，丢弃此报文。

* 3xx响应码处理

3xx为交互式响应状态码，代表需要做多次交互，那么前面说到的头字段中的Contact字段就可以发挥作用了，用来标注交互双方的地址。具体见协议文档描述。

* 4xx响应码处理

4xx为发生错误状态码，代表发生错误了，需要针对处理，具体见协议文档描述。

# **UAS的消息结构和处理方法（请求 & 应答）**

UAS处理非会话请求时，处理规则和请求报文中的Method头字段无关。处理来自UAC的请求具有原子性。

UAS处理UAC请求报文的默认流程：

鉴权===》检查请求Method头字段===》检查其他头字段

* 检查其他头字段
  + **头字段To**：

请求消息的原始接收者，它由From字段代表的终端指定，由于可能存在proxy，那么经过proxy的路由转发后，最终处理这个报文的UAS当前UAC压根不知道其URI地址。

当某个UAS收到了不是发给自己的报文时（To字段不是自己），建议依旧接受这个报文（类似于UDP的广播模式，虽然不是发给自己的，但是也能收得到）；当然，UAS也可以选择拒绝接收，此时要反馈一个403状态码的报文回去。

* + 头字段Request-URI：

标识了处理这个请求的URI。如果请求消息的Request-URI使用了UAS所不支持的URI方案，那么返回416应答码；如果请求消息的Request-URI标识的不是UAS用来想要用来接收请求的地址，那么返回404应答码。

***一些假想：****结合To和Request-URI字段的描述，猜测Reuqest-URI字段在编码端是人为指定的，如果存在路由策略，那么很可能是路由器来完成自动填充；相反，To一定是人为指定的。*

* + 头字段Require：

当所有的检查都做完，确认可以处理此报文时，UAS会检查Require字段，此为扩展选项标签，具体见协议文档

* 重叠请求

如果某一个请求报文，再网络中存在多份，那么只有第一份报文会被UAS处理，UAS会给其余的报文应答482错误码。

* 发送临时响应

UAS仅对INVITE（一种Method）请求发送临时响应，对于其他Method都立刻产生最终响应。

* 头字段和标签

响应消息的From、Call-ID和Cseq必须和请求消息的From、Call-ID和Cseq，值一样。

同时，对于Via字段，除了值要一样外，所有值得排序还要一样。

# **重定向服务器**

重定向服务器负责消息的路由。

重定向，UAS把某个请求消息应该使用到的路由信息放在应答消息中，再反馈给UAC，UAC收到应答消息后，会向路由信息中标示的新URI再次发送请求。进而提高了精准度和效率。

重定向服务器包含一个定位服务器，这个服务器实际上就是一个数据库。它包含了单个URI到一个或者多个联系地址之间的映射。

重定向服务器不发送任何SIP请求。

重定向服务器决不能将请求重定向到一个与请求的Request-URI 相同的URI 地址。如果重定向的目的地URI 所指并非当前重定向服务器，它可能将请求向重定向目的地转发，或用一个404（未找到）响应来拒绝请求。

# **请求的取消**

UAC和重定向服务器可使用CANCLE来取消自己发送的前一个请求。

UAS已经应答的请求无法被CANCLE。

CANCLE最多的使用场景就是INVITE请求，如果UAS收到了针对INVITE的CANCLE，而且还没有对INVITE发出应答（就是还没处理完INVITE），那么UAS将“停止振铃”，然后使用487来应答INVITE来表示“这个INVITE我已经CANCLE了”。

* 客户端行为（UAC & 重定向服务器）

一般情况下，CANCLE只用来处理INVITE请求。

既然CANCLE请求是用来请求撤销某个消息的，那么就需要一套唯一关键字来定位特定消息，这里用的是Request-URI、Call-ID、To、CSeq的数字部分、From、各个标签，SAC发出的CANCLE报文中的如上字段必须要和它试图CANCLE的报文一样。

如果待取消报文中有Route字段，那么CANCLE报文中也要有一模一样的Route。

CANCLE请求不能包含任何Require或Proxy-Require头字段。

构造完CANCLE请求后，在发出去之前需要先自验是否收到的UAS回的应答报文。

如果UAC在64\*T1秒内没有收到来自UAS的应答消息，那么UAC就要主动发起一个CANCLE请求来取消上一次请求消息。（可以理解为在这个时间间隔内，UAS不反馈，默认UAS处理异常，UAC此时要主动要求撤销上一次申请）

* 服务端行为（UAS）

UAS（UAS & 代理服务器）处理收到的CANCLE时，有两种情况：1、无状态代理服务器将简单地转发请求；2、有状态代理服务器会对其作出应答并产生自己的CANCLE请求，路由到目的地UAS。

如果UAS没有找到与CANCLE匹配的事务，那么会返回481错误码；如果匹配到了，还要检查是否已经发送应答消息，如果发送了，那么UAS当什么事都没发生；如果匹配到了，还没发送应答消息，那就应答当前的CANCLE消息。

# **查询能力**

UA（UAS & UAC & 代理服务器）之间可以通过OPTIONS方法互相查询对方的部分信息。

应用：可以通过发送一些列包含递增的Max-Forwards值得OPTIONS请求来查询路径中每个服务器的能力。

如果发出OPTIONS后，长时间没应答，事务层（UDP、TCP等）会返回一个超时错误以表示目标不可达。

* OPTIONS请求的构造

可以包含一个Contact头字段，应该包含一个Accept字段，作用范围是初始请求的Request-URI

* OPTIONS请求的处理；

在对话中收到OPTIONS请求时，应回复200响应，且对当前会话不产生任何影响。

如果UAC发送OPTIONS给代理服务器，那么代理服务器反馈的响应中应详细描述自己的功能，但是，不能有报文体。

*注：代理服务器不能识别方法，仅扮演转发路由，以及类似DNS服务器的功能，不具备处理报文的能力。*

# **对话（dialog）**

对话是两个UA（UAC、UAS、代理服务器）之间的点对点的SIP链接。

对话用来描述下列事物：解析SIP报文的上下文环境。

同一个SIP链接上，两端的UA各自持有自己的对话，因此各自的对话ID也是不同的。

对话的ID是唯一的，由Call-ID、一个本地标签、一个远程标签，三个要素构成。

对话中包括一些对话中的后续消息所需的状态，包括：对话ID、本地序列号、远端序列号、本地URI、远端URI、远端目的、布尔型标记“secure”和路由集。

* 对话ID的计算

UAC：Call-ID 由消息的Call-ID 头字段设置，远端标签（tag）由To 头字段的tag 设置，本地标签由于From 头字段的tag 设置。

UAS： Call-ID 由消息中的Call-ID 头字段设置，远端标签由消息From 头字段的tag 设置，本地标识符于消息To 头字段的tag 设置。

# **对话的创建**

对话在请求消息得到了明确的非失败的响应之后才被创建。当一个对话创建以后，它处于“初始”状态，也称为初始对话。还可以通过其他方法定义对话的创建。

* UAS的动作

UAS在收到来自UAC的某个可建立对话的请求报文时，会作出一系列的动作，但是所有动作只有一个目的：组装响应报文。

* UAC的动作

两个动作：1.组装请求报文；2.解析应答报文，更新本地数据结构

# **对话中的请求**

当对话被创建后，双端的UA都可以主动发起一个新“事务”，此时，在此“事务”中，主动发起的一端叫做UAC，被动接收的一端叫做UAS。

* UAC发送请求报文

略（组报文的规则）

* UAS接收请求报文

略（报文中不同字段的不同值需要怎么处理）

* UAS发送应答报文

略（根据实际情况组应答报文）

* UAC处理应答报文

略（根据实际情况处理应答报文）

# **会话（session）**

UA通过INVITE请求申请与服务器（UAS）建立会话，UAC和UAS可能不是直连的，需要代理服务器提供路由服务，此时代理服务器提供转发，然后最终会找到对应的UAS。如果UAS允许建立会话，那么返回2xx的响应；若不允许，返回3xx、4xx、5xx、6xx的响应。在UAS发送最终响应之前，UAS还能发送1xx的临时通知来告知UAC当前处理进度。

软件层面，如果UAS对于当前INVITE请求做了fork动作，那么出于fork函数的特殊性，相当于UAS两个进程都受到了报文且持有了底层链接。这样的话UAC可能会收到两个2xx的应答，与此同时，也会创建两个对话（dialog），但是这两个对话属于同一个呼叫。

前面说到了“会话是由INVITE请求创建的”，那么只要关注INVITE请求在UAC和UAS之间的交互流程和规则即可：

* UAC创建INVITE请求的规则

略

* UAS处理INVITE请求的规则

略

* UAS创建INVITE应答的规则

略

* UAC处理INVITE应答的规则

略

# **会话更改**

会话建立之后，任何一方都可以发起会话更改请求，修改这些会话的某些属性。具体的方法为：在当前已经建立的对话（dialog）中，再发送一次INVITE，并在INVITE中指定新的属性。这个过程又称为reINVITE。

# **会话更改的过程**

* UAC的处理

略

* UAS的处理

略

# **会话结束**

尚未建立的会话：对INVITE请求返回拒绝错误码

已经建立的会话：向对方发送BYE请求

# **代理服务器**

代理服务器：将请求消息路由到UAS以及将相应消息路由到UAC的实体。

代理服务器会修改请求消息和响应消息中的部分字段，以用来完成路由转发。

请求消息和响应消息的路由过程是正好相反的，一个是去路，一个是回路。

代理服务器既可以是“路由器”，也可以是UAS，具体要看服务器上跑的软件实现了怎么样的功能，如果会给UAC应答，那么就是UAS；如果只是实现了路由功能，那么就是个路由器。

* 代理服务器的工作模式
  + 无状态模式

此时，代理服务器只提供路由功能，通过解析响应字段并结合路由选路技术，将请求报文和应答报文送到下一个路由节点或目的地。

完成自己的任务后就会把报文信息丢掉。

* + 有状态模式

有状态服务器会记住经过自己的每个请求报文的信息，这些信息会用来判断后续报文的处理方法。

有状态服务器可以fork，这回导致输入一个请求，路由出去两个或更多请求。此时这一个请求可能被发送到多个目的地。

代理服务器不应主动发送CANCLE请求。

# **有状态代理服务器（之前很多假设在这里都被推翻了，需要重新整理）**

参考资料：

http://www.cnblogs.com/my\_life/articles/2282364.html

工作在有状态模式下的代理服务器，有状态代理服务器是一个“事务处理引擎”。

组成：一个服务端事务，一个或多个客户端事务，将服务端事务和客户端事务关联起来的代理服务器核心。

对于一个有状态代理服务器，分工为：

* 服务端事务负责处理请求报文；
* 接着，交由代理服务器核心完成网络层的工作；
* 然后，把前两者处理后的结果通过客户端事务以请求报文的形式发送出去；
* 最后，客户端事务再把响应报文的处理结果交由代理服务器核心，进而交给服务端事务做进一步决策。

代理服务器必须具备的功能：

* 确认请求的有效性（参见本规范16.3 节）

如果任何一项检查失败，代理服务器必须象UAS 那样处理（参见本规范8.2 节）并用一个错误码来应答。

如果一个请求的Request-URI 使用了代理服务器不能理解的URI 方案，代理服务器应当用一个416（不支持的URI 方案）响应拒绝该请求。

Max-Forwards 头字段要么大于0，要么不存在。

Proxy-Require检查，属于扩展内容，UAC可能对于代理服务器有特殊要求，那么需要把这些要求写入Proxy-Require头字段，但是要在业务层预先商量好这些新增字段该如何处理。如果代理服务器处理不了这些字段，那么返回420错误码。

和http一样，SIP也存在证书，也存在鉴权。

* 预处理路由信息（参见本规范16.4 节）

代理服务器必须检查请求报文的Request-URI字段。

如果是路由功能的代理服务器向另一个代理服务器发送报文，那么Request-URI字段值需要被替换成Record-Route字段中的某个值。

* 确定请求发送的目的地（参见本规范16.5 节）

略

* 将请求向每个目的地转发（参见本规范16.6 节）

略

* 处理所有的响应消息（参见本规范16.7 节）

略

# **SIP事务层**

SIP是事务型协议，一个SIP事务由请求和请求对应的响应（临时响应&最终响应）组成，事务是针对某一端而言的，对于同一个业务，双端都有自己的事务与之对应。

对于INVITE请求，临时响应为ACK响应，最终响应为处理的结果响应（2xx成功，4xx失败）。

事务分为两类：客户端事务 和 服务端事务。

小结：所谓的事务，就是指完成某些功能的软件的合集。最重要的是了解哪些功能属于客户端事务，哪些功能属于服务端事务，见A82和A83。

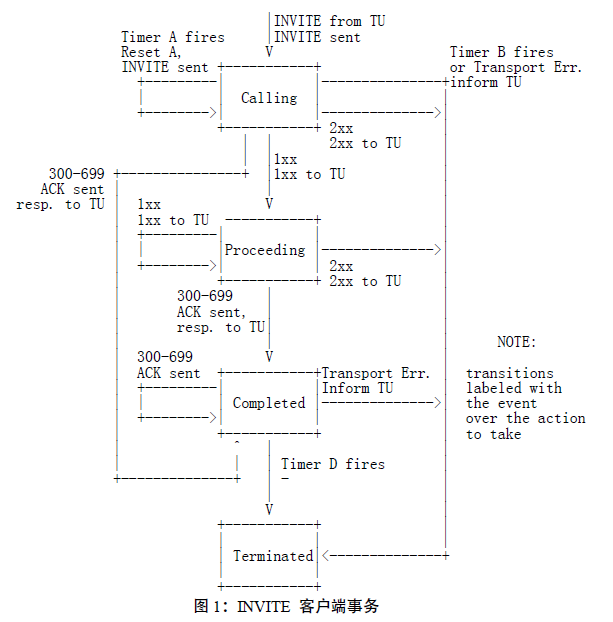
# **客户端事务**

通过状态机模型来描述客户端事务的业务流程。也可以认为一个状态机流程就是一个事务，出状态机就代表事务的终结。

前面说了，对于客户端事务，状态机模型有两套：INVITE状态机 和 非INVITE状态机。

如果使用可靠控制协议（TCP）作为控制层，那么不存在重传；如果使用不可靠控制协议（UDP），需要有重传机制。

* 客户端INVITE事务状态机



两个定时器：

A：非可靠有，T1为基准时间，每次翻倍，计时满则重传

B：可靠&费可靠，64\*T1为基准，计时满，整个事务超时

更多细节，参考官方文档

* 客户端非INVITE事务状态机

更多细节，参考官方文档

# **服务端事务**

服务端INVITE事务状态机

服务端非INVITE事务状态机

# **传输**

传输层负责通过网络传输请求和响应。

由于收发双方都是UA，地位是相等的，所以如果使用TCP作为传输层，那么收发双方都要有自己的监听端口，相当于两个连接通路。这样双方都能发起事务。

本规范中强制UA使用TCP。

更多内容，详见文档

# **普通的报文成分**

Sips是sip的安全版，类比http和https

# **头字段**

略

# **响应码**

略