第三部分 协 议 层

第14章 无线会话协议规范

14.1 范围

无线应用协议（Wireless Application Protocol，WAP）是WAP论坛经过不断努力得到的成果，它提供了一个业界技术规范，以便开发出适用于各种无线通信网络的应用和业务。 WAP论坛的工作范围就是为各种业务和应用制定一系列的技术规范。无线市场正在快速增长，新的用户不断增多，新的业务不断涌现。为了给运营商和生产者提供一个面对先进业务、多种类业务和快速灵活的业务生成的商机， WAP制定了一系列传输层、会话层和应用层协议。有关WAP体系结构更多的信息，请参阅“无线应用协议体系结构规范” (Wireless Application Protocol Architecture Specification) [WAP]。

在WAP体系结构中，会话层协议族被称作无线会话协议（ WSP）。WSP为处于WAP较高层次上的应用层提供了两个会话业务的一致接口。一个是连接模式业务，运行在事务处理协议层（WTP）上面；另一个是非连接业务，运行在安全或非安全数据报传输业务之上。有关事务处理和传输等业务的更多信息，请参考无线应用协议 [WAP]：无线事务处理协议规范[WAPWTP]和无线数据报协议规范[WAPWDP]。

WSP目前能够提供最大限度适应浏览应用（ WSP/B）的业务。WSP/B提供HTTP1.1功能，并且具有新的特点。例如具有较长会话存活期，能够提供数据推、能力协商、会话挂起 /恢复等通用工具。WSP族中的协议适用于具有相对的较长延迟时间的低带宽承载网络。

14.2 WSP体系结构概述

WSP是会话级协议族，它为客户端、服务器或代理之间的远程操作提供服务。

14.2.1 参考模型

WAP的协议分层模型如图14-1所示，WAP协议及其功能的分层方法类似于ISO OSI参考模型[ISO7498]。协议初始化，配置和出错情况管理（例如由于移动台漫游出覆盖范围而引起的连接丢失）是由层管理实体处理的，而并不是由协议本身处理的。

WSP是为处理事务和数据等业务而设计的。安全层是可选层，位于传输层的上面，具有传输业务的接口，事务处理、会话或应用管理实体应能够提供额外的支持，以建立安全的上下文和安全连接功能。但这一支持并不是由 WSP直接提供的，就这一点来说，安全层是可选的。WSP本身并不需要安全层，但是，使用 WSP协议的应用可能需要安全层。

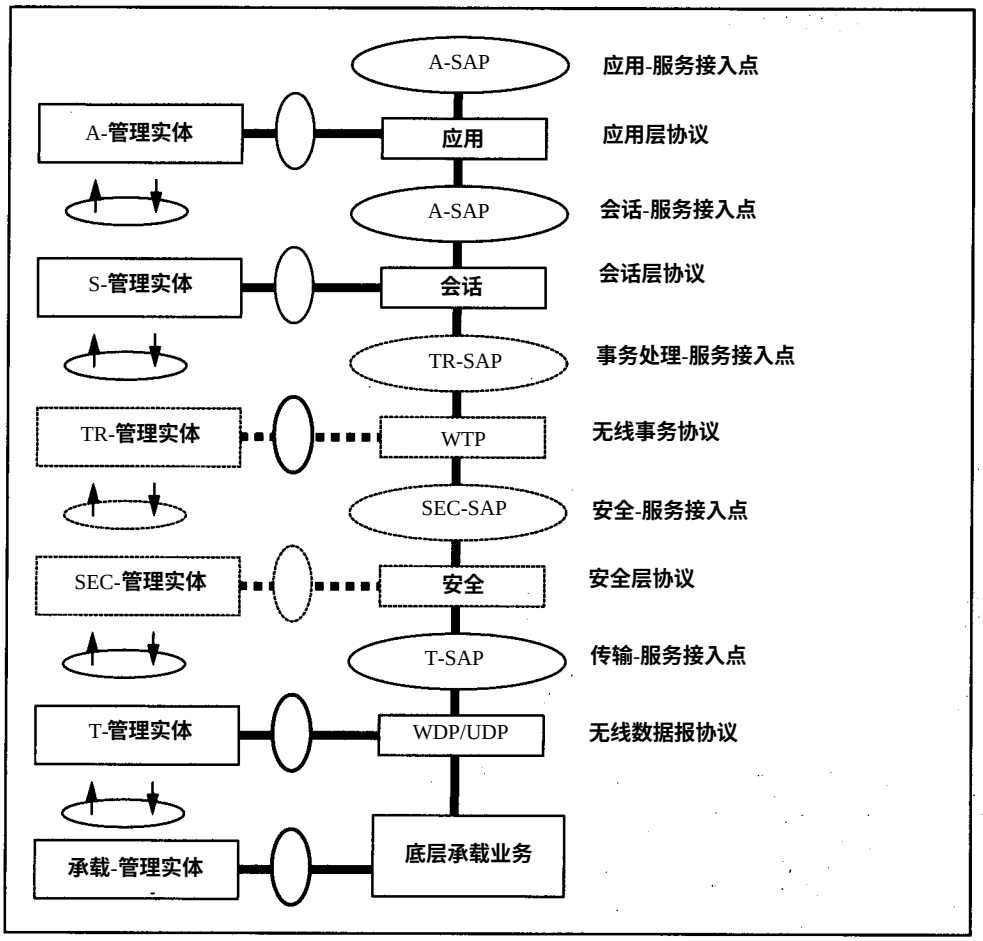

图14-1 无线应用协议参考模型

14.2.2 WSP/B特点

WSP提供了一种方法，使得相互协作的客户端 /服务器应用程序之间进行有组织的数据交换。特别值得一提的是，WSP提供了以下几种具体应用方法：

- 建立可靠的从客户端到服务器的会话，并有序释放该会话。
- 使用能力协商，在协议功能的通用级别上达成一致。
- 使用压缩编码，在客户端和服务器之间交换数据。
- 挂起/恢复会话。

目前定义的业务和协议（WSP/B）最适于浏览类型的应用。 WSP/B通常定义了两个协议：一个协议提供连接方法的会话业务，建立在事务处理的业务之上； 另一个协议提供非连接方法非确认的业务，建立在数据报传输业务之上。非连接方法业务很适合应用于不需数据的可靠传输和并不关心确认的情况，实际上不需建立会话即可使用非连接方法业务。

除了一般特点之外，WSP/B还提供如下方法：

- 提供HTTP/1.1功能：

可扩展请求/应答方法、复合对象、内容类型协商

- 交换客户端/服务器会话报头。
- 中断事务处理。
- 从服务器至客户端异步推（Push）内容。
- 协商支持多个同时发生的异步事务处理。

1. 基本功能

WSP/B的核心设计是HTTP二进制形式的，因此，发往服务器的请求和传给客户端的响应可以既包括报头也包括数据，WSP/B支持所有HTTP定义的格式。另外，它使用能力协商约定了一系列扩展请求方法，从而达到对HTTP/1.1的完全兼容。

WSP/B为应用层提供键入数据的传输。它使用 HTTP1.1内容报头可扩展方法定义内容类型、字符集编码、语言等等，而为知名报头定义压缩二进制编码是用来减少协议报头大小的。WSP/B同样定义了为复合数据对象每一部分提供内容报头的压缩复合数据格式，从语意上来说，该格式等同于HTTP1.1中所使用的多重或混合的二进制格式。

WSP/B本身并不解释请求/应答的内容报头信息。请求/应答的内容报头作为会话创建进程的一部分，在整个会话期间保持一致，它们能在客户端 /服务器之间交换，交换内容包括可接受内容类型、字符集、语言、设备性能等其他静态参数。 WSP/B既可以毫无增删的传递客户端/服务器会话报头，也可以传递请求和响应报头。

WSP/B会话存活期同低层的传输没有直接关系，会话在空闲时也可以挂起，以释放网络资源或节省电池。轻型的会话重建协议使会话即使在没有充分的会话建立报头的情况下同样能够被恢复，会话同样可以在不同的承载网络上恢复。

2. 扩展功能

WSP/B允许对等实体之间通过协商拥有扩展性能，这使得在获得简单基本的实现的同时也易于获得高质、全面的实现。 WSP/B提供了一种可供选择的机制，使内容报头信息加到事务处理的确认中，这就允许客户端应用程序传递已结束事务处理的具体信息回服务器。WSP/B提供推（PUSH）和拉（PULL）两种数据传输，利用HTTP1.1请求/响应机制可推数据，并且WSP/B提供了3种推数据机制来进行数据传输：

- 在会话上下文已经存在的情况下，进行确认的数据推操作。
- 在会话上下文已经存在的情况下，进行非确认的数据推操作。
- 在会话不存在的情况下，进行非确认的数据推操作。

经确认数据推机制允许服务器在会话期任何时间内向客户端推数据，并且服务器收到推数据已传输的确认信息。

在活动会话内未经确认的数据推提供的功能同可靠数据推提供的功能类似，只是未经确认，即使不在活动会话内也一样可以进行未经确认的数据推。在这种情况下，是假定已有默认的会话上下文，没有会话上下文的未经确认的数据推用来在不可靠的传输链路上发送单方向的数据。

WSP/B可以有选择性地支持多个异步请求，以便客户端能够向服务器同时提交多个请求。由于多个请求和响应能够合成为较少的报文，所以这就提高了传播时间的利用效率。同样，由于每个请求一旦有效就可以发给客户，所以这也提高了反应时间。

WSP/B把知名报头字段名划分成报头代码页，每一代码页为知名字段名定义的编码数量

相当有限，这使得字段名的表示看起来更紧凑。因为 WSP/B规定了代码页之间转换的机制，所以即使不能为某个代码页上每个知名字段名都加上标识，这也不会造成问题。

14.3 WSP层间通信元素

WAP会话层提供了连接方法和非连接方法业务，它们由抽象描述术语来定义，而这一术语是建立在源于[ISO10731]服务原语基础之上的。一些用来描述通讯机制的术语和概念源于[ISO7498]，用来描述运行和可操作数据对象的术语则是基于[RFC2068]。

这项业务定义了WAP会话层必须提供给其用户的最小功能集合。由于这个定义是抽象的，所以它并未规定编程接口或具体实现。事实上，这项业务可使用不同的协议来传递。

14.3.1 使用的符号

1. 服务原语和参数定义

在会话层内的实体之间以及层与层之间的通信是由许多种服务原语完成的。服务原语抽象地描述了会话层与其邻接层之间逻辑上的信息交换与控制，它由命令和对各命令的响应组成，其响应同提供的服务类型有关。一个原语的一般句法是：

X-Service.type （Parameters ）[X-服务.类型（参数）]

X表示提供服务的层。在本规范中，X指会话层，用“S”表示。

服务原语并不同于应用编程接口（API），也不是使用API的一种特殊方法，它是解释说明由协议层向上层提供的服务的一种抽象方法。特别值得一提的是，服务原语及其参数并不包含具体实现向每个实现对象传递该原语时所需的信息，其中，该实现对象与抽象用户或业务提供者的实例进行通信。这些概念与实际API函数的映射和与实际API的语义关联，它们是实现方面的问题，超出了本规范的范围。

2. 时间序列表

服务原语的使用特点可由[ISO10731]中描述的时间序列表来解释说明。

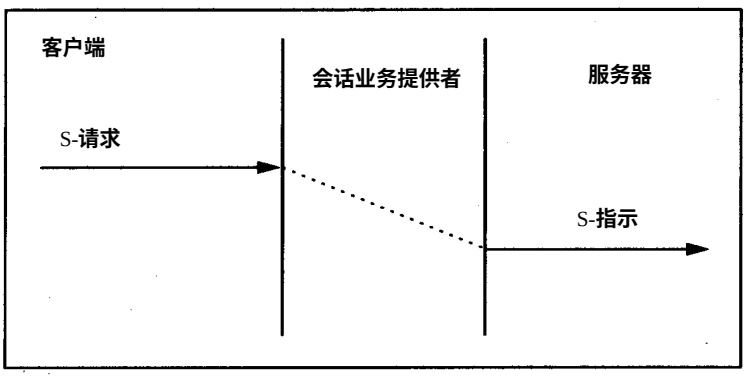

图14-2 一个未确认服务

图14-2表明客户端使用请求原语引起一个简单未确认服务，并在对端中产生了一个指示原语。虚线表示经过提供者一段时间的传播，这种传播由表示原语的两个箭头之间的纵向间隔代表。如果客户端和服务器的标志（label）在图表中，则表示两端都不能产生原语；如果

图表中省略了客户端和服务器的标志，则表示两端都可以产生该原语。

3. 服务原语类型

在规范中定义的服务原语类型如表14-1所列。

表14-1 规范中定义的原语类型

类 型	简 写	描 述
请求	req	应用于高层向邻近低层请求服务时
指示	ind	向用户提供这一服务原语类型的层通知邻近高层通信另一方的有关行为（例如请求原语的引发）或服务提供者的有关行为（例如一协议产生事件）
响应	res	该层使用响应原语类型来向邻近低层确认接收到的指示原语类型
确认	cnf	提供服务请求的层使用确认原语以通知服务已结束

4. 原语参数表

服务原语如下表中所定义。表中标明服务原语可能使用哪些参数以及不同服务原语是如何使用这些参数的。如果某一原语类型不能使用，则表中略去相应栏。在原语类型栏中定义的项由表14-2定义。

表14-2 参数用法表

M	必须有该参数
C	有无该参数取决于其他参数值
O	有无该参数由用户选择
P	有无该参数由服务提供者选择，实现可能不需要该参数
—	没有该参数
*	有无该参数取决于低层协议
(=)	该参数值同前面的服务原语相应的参数值相同

例如，简单的确认原语的定义见表14-3。

表14-3 简单确认原语的定义

	原语 S—PRIMITIVEX			
参数	Req	Ind	Res	cnf
参数1	M	M（=）	—	—
参数2	—	—	O	C（=）

如前面的示例中定义所示，参数 1常出现在 S-PrimitiveX.requset中，并与 S—PrimitiveX.indication的参数相一致。参数2可能在S-PrimitiveX.reponse中定义。在这种情况下，它必须与对应的S-PrimitiveX.confirm中的参数等值。否则，它根本就未定义。

一个简单的原语示例如表14-4所列。

表14-4 简单的原语示例

	原语 S-PRIMITIVEY	
参数	REQ	IND
参数2	-M	

在这个示例中，S－PrimitiveY.request没有参数，但是相应的S－PrimitiveX.indication必须有参数2。因为S－PrimitiveX.response和S－PrimitiveX.confirm在示例栏中未定义，所以相应类型的服务原语永不会发生。

14.3.2 服务原语参数类型

本节描述了在服务原语定义中使用的各抽象参数的类型，这些类型的实际格式与编码等具体实现方面的问题，不在本服务原语定义中讨论。

在原语的描述中，类型出现在参数名中，并经常有一个附加的限定词来指出该参数将如何或者在何处使用。例如，参数 Push Body的类型是Body，参数Client Address的类型是Address。

1. 地址（Address）

会话层直接使用下一层的地址结构。服务器和客户端的地址共同形成了通信地址的四重组（the peer address quadruplet），它将标识出本地低层通信使用的业务接入点。在引发会话层业务前，本业务接入点必须为通信作好准备。该项准备需要用户和管理实体相互作用才能完成，完成的过程不属于本规范的范围。

2. 报文（Body）和报头（Header）

报文类型相当于HTTP中的entity－body[RFC2068]。报头类型代表了一系列属性项，它们相当于HTTP中的报头。

3. 性能（Capabilities）

性能类型表示了一些与服务提供者的操作有关的服务工具和参数设置。预定义的性能参数将在14.3.3节“已定义性能”中描述，但是服务提供者可以识别补充性能参数。

4. 推标识符（Push Id）

推标识符代表了一抽象值，它被用来唯一地区分会话中的推事务处理，该事务在业务接口上被延迟。

5. 原因（Reason）

服务提供者使用原因类型通告某一具体指示原语的起因。服务提供者可以定义补充原因值，但服务使用者必须识别表14-5中所列的几种。

表14-5 原因值及其描述

原 因 值	描 述
PROTOERR	当前状态下，协议禁止通信方执行操作。例如，使用过的PDU被禁止
DISCONNECT	操作仍在进行时，切断会话
SUSPEND	操作仍在进行时，挂起会话
RESUME	操作仍在进行时，恢复会话
CONGESRION	由于缺乏资源，通信方不能发出请求
CONNECTERR	一个阻止会话生成的错误
MRUEXCEEDED	请求的SDU大小大于通信双方协商确定的最大接收单元
MOREXCEEDED	同时存在的未完成方法请求或PUSH请求的数量超过了规定的上限
PEERROR	服务方请求的操作被放弃
NETERR	低层的网络错误导致请求无法完成
USERREQ	指示由本地服务使用者的操作引起

6. 请求URI（Request URI）

这一参数类型试图与HTTP方法请求[RFC2068]中的Request－URI有相同的用法。然而，会话用户可以视情况选择使用这种参数类型，甚至可以置它为空或者包含与URI语法不一致的二进制数据。

7. 状态（Status）

状态参数类型等同于HTTP1.1的状态代码值[RFC2068]。

8. 事务标识符（Transcation Id）

事务标识符代表一个抽象值，它被用来唯一地区分挂在业务接口的一个会话的方法请求事务。

14.3.3 连接模式会话业务

1. 概述

连接模式会话业务被划分成各种工具，其中一些是可选的，大多数工具是非对称的，这样使会话所连接的服务器与客户端所进行的操作可以不同。提供的工具如下：

- 会话管理工具（Session Management facility）
- 方法调用工具（Method Invocation facility）
- 异常情况报告工具（Exception Reporting facility）
- 推工具（Push facility）
- 确认推工具（Confirmed Push facility）
- 会话恢复工具（Session Resume facility）

会话管理工具和异常情况报告工具总是可用的，其他的工具由能力协商在会话建立期间决定。

会话管理工具使客户端能与服务器相连接，并使二者在所使用的工具和协议选项上达成一致。服务器可以拒绝连接，并将客户端重定向至另一个服务器。在会话建立期间，客户端和服务器也可以交换属性信息，这些信息需要在会话期间一直保持有效。服务器和客户端都可以结束会话，并通知对方。如果由于服务提供者或管理实体的原因结束了会话，同样要通知用户。

方法调用工具使得客户端能够向服务器请求操作并返回操作结果。可执行的操作是HTTP方法[RFC2068]或由用户自己定义的扩展操作，这些操作符合请求－应答模式或事务处理模式。不管事务处理是否成功，都应该通知客户端或服务的提供者。被用户和服务提供者放弃的操作会造成事务处理的失败。

异常情况报告工具让服务提供者能够通知用户一个与具体的事务无关的事件。这个工具不改变会话状态。

推工具使服务器能够利用客户端与服务器的公用会话信息，向客户端主动发送非请求信息（unsolicited information）。这个工具是非确认的，所以信息传输是不可靠的。

确认推工具与推工具类似。但是客户端需要确认信息的接收，同时也可能会通知服务器放弃PUSH操作。

会话恢复工具包含挂起会话的方法，这种方法保存当时的会话状态。但是通信双方都清楚只有客户端才能恢复该会话，使通信继续。这种机制也适用于如下情况：双方不再进行通信，

直到服务提供者发现服务使用者或管理实体采取了相应的矫正措施。同样，利用这一机制可以把会话转换至拥有更多匹配资源的承载网络上去，从而保证给定承载网络的负载更加合理。

2. 性能

使用性能可以管理与会话服务提供者操作有关的信息。性能应用广泛，例如表示选定的服务工具集，设置特定的协议参数，建立会话双方使用的代码页和扩展方法名等。

(1) 能力协商

能力协商用于服务的双方，用来约定一个双方都可接受的服务水平，并根据用户的实际需要来优化服务提供者的操作。能力协商只能用于可协商的能力，通告性的能力只能无修改地传送到双方的服务使用者。

发起能力协商的一方称为倡议方（initiator），另一方称为响应方（responder）。单向的能力协商定义是：倡议方提出一系列性能要求，响应方进行响应这些要求。能力协商过程由倡议方控制，所以如果性能设置超出了倡议方的建议和服务提供者可以支持的功能级别，那么响应方将不会对此作出应答，能力协商通常适用于所有已知的性能。如果服务原语未包含某一性能，则意味着服务原语的发起者（originator）希望使用目前的性能设置，或者希望使用默认值，或者是在能力协商过程中商定。然而，响应方仍可能会以一不同的性能值作出应答，只要该值并未包含更高级别的功能。

单方向能力协商过程如下：

- 倡议方的服务使用者提出一系列性能初值。
- 倡议方的服务提供者修改性能，以使性能不超过倡议方服务提供者实际能够支持的服务功能级别。
- 响应方的服务提供者修改性能，以使性能不超过响应方服务提供者实际能够支持的服务功能级别。
- 响应方的服务使用者接收已修改的性能，并进行响应，响应时所使用的性能应能够反映服务使用者实际希望使用的功能级别。如果响应方的服务使用者应答时并未包含某一性能，则意味着服务使用者希望使用对方提出的性能设置。
- 响应方的服务使用者所选的性能集合由倡议方的服务使用者指定。这些性能成为默认设置，并且在会话的下一个能力协商中仍然有效。

如果由服务原语指定的、传送性能信息的操作失败，在该操作之前有效的性能设置仍然保持有效。

如果可协商的性能值是一个正整数，那么最终性能设置的取值由服务使用者给出，这个值是双方的服务提供者都能够支持的最小值。

如果可协商的性能值是一个集合，那么最终性能集将只包括那些由服务使用者建议使用的子集和服务提供双方都能支持的性能。

(2) 已定义性能

服务使用者和服务提供者可以识别如下的性能：

3. 服务原语

本节列出了业务所支持的所有抽象的服务原语并给出了它们的含义。

(1) S－connect

表14-6列出了已定义的性能。

表14-6 已定义性能

性能名称	级 别	类 型	描 述
替换符	I	地址列表	当前会话中，服务提供者使用相应地址可访问同一服务使用者实例。可任意排列地址顺序，一般把最常用的替换符放在最前面。例如，当恢复一个会话时，此信息可便于切换到一个新的承载体
客户端SDU大小	N	正整数	客户端和服务器使用该功能约定，会话过程中可能发往客户端的最大事务处理数据单元的大小
扩展方法	N	方法名称集合	使用该性能约定扩展方法（这在HTTP1.1中未定义）。客户端和服务器均支持这些方法，并用在随后的会话中
头代码页	N	代码页名称集合	使用该功能可约定扩展报头代码页。客户端和服务器均支持这些代码页并在随后的会话中使用这些代码页
最大未完成方法请求数目	N	正整数	客户端和服务器使用该功能约定，在会话过程中同时活动的最大方法请求数目
最大未完成推请求数目	N	正整数	客户端和服务器使用该功能约定，在会话中同时活动的确认PUSH请求的最大数目
协议选项	N	工具和特征集合	该性能置设选定的服务工具和特征。 它可包含以下内容： 推操作，确认推操作，会话恢复和确认报头。如某一项出现在该性能中，则表明相应工具或特征有效
服务器SDU大小	N	正整数	客户端和服务器使用该功能约定，在会话过程中发往服务器的最大事务处理服务数据单元的大小

附注 在上表级别栏中：N表示可协商的，I表示通告性的。

这个服务原语用于会话建立的初始化并通知会话已建立，它使得作为倡议方的客户端和作为响应方的服务器能够进行单方向的能力协商。它是会话管理工具的一部分（见表 14-7）。

表14-7 原语S-connect及其含义

参 数	REQ	IND	RES	CNF
服务器地址	M	M（=）	—	—
客户端地址	M	M（=）	—	—
客户端报头	0	C（=）	—	—
请求的性能	0	M	—	—
服务器报头	—	—	O	C（=）
协商的性能	—	—	O	M（=）

服务器地址（Server Address）标识了将与哪一方建立会话。

客户端地址（Client Address）标识了会话的发起者。

客户端报头（Client Headers）和服务器报头（Server Headers）标识的属性与HTTP消息

报头[RFC2068]相一致，在服务使用者之间进行通信时不会被修改。它们可作为应用级的参数或作为高速缓存请求报头和应答报头，在会话中始终都保持不变。但是，如何理解和使用它们完全取决于服务使用者。如果未提供这些参数，应用利用与其有关的默认会话报头，为这个会话提供一个静态的格式。

请求性能（Requested Capabilties）和协商性能（Negotiated Capabilties）可实现在14.3.3节中所描述的能力协商过程。如果违反了能力协商的规则，会导致会话建立过程中相关操作的失败。

在会话建立期间，服务使用者可以发起某些服务原语，但这些服务原语未必用于最终选择的会话功能。当会话建立和相应的能力协商结束时，这些服务请求会被放弃并向服务使用者报告错误。在会话建立以后，如果再出现发起原语的错误，相应的行为仅与本地的具体实现有关。

图14-3指出了在会话建立成功的过程中所使用的原语。服务使用者在会话建立时可能已经请求了一个方法调用，相关原语由图中虚线表示。

在会话建立期的任何时间内，服务提供者均可产生一个断开指示。

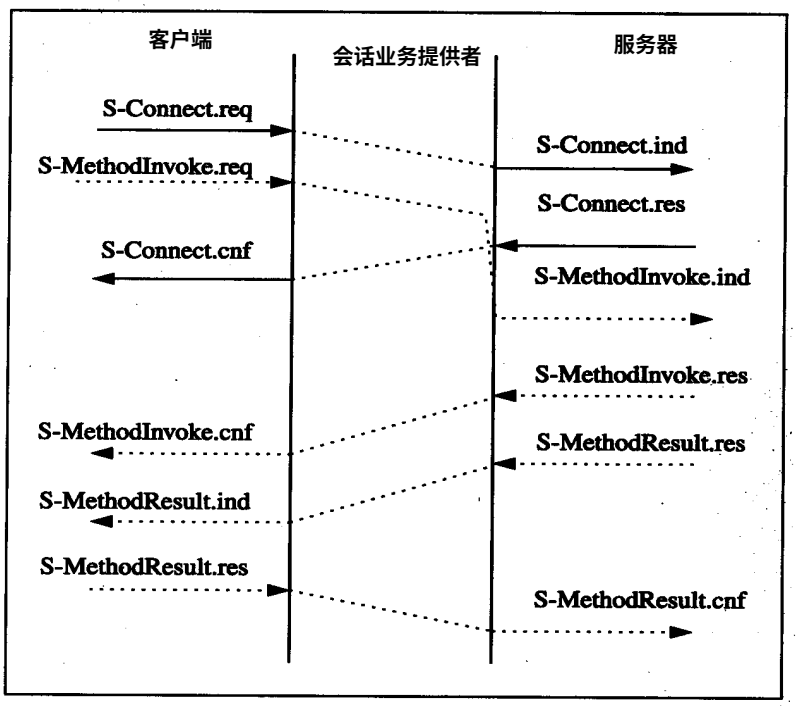

图14-3 成功的会话建立

(2) S-Disconnect

这个服务原语用于断开会话，并通知会话用户该会话不能建立或被断开，它是会话管理工具的一部分（见表14-8）。不管是本地服务使用者，还是对等端的服务使用者或是服务提供者都可以断开会话，一经检测到会话结束就发出该原语。在断开指示之前，会话服务提供者必须放弃所有未完成方法和推事务处理。在断开指示之后，与这个会话相关的服务原语不会再出现。

表14-8 原语S-Disconnect 及其含义

参 数	REQ	IND
原因代码	M	M（=）
重定向安全	C	C（=）
重定向地址	C	C（=）
错误报头	O	P（=）
错误报文	O	P（=）

参数原因代码（Reason Code）指示了断开的原因，可能的取值为原因和状态参数类型。

如果原因代码指示客户端正重定向至一个新的服务器地址，那么必须要有重定向安全（Rediect Security）和重定向地址（Redirect Addresses）两个参数。

重定向安全指示了当重定向至新的服务器时客户端是否重新使用当前的安全会话，或者它是否必须使用一个不同的安全会话。

重定向地址是可变地址，客户端使用该参数来与客户端最初试图连接的服务器建立会话。如果原因代码指示客户端只是暂时的被重定向，那么一旦重定向的会话结束，该客户端应该使用最初的服务器地址，试着再建立一个会话服务。如果原因代码指示客户端是永久性的重定向，那么该客户端应该使用重定向地址参数中的一个地址，试着建立一个会话服务。

如果原因代码参数的取值为状态类型，那么除了原因代码之外，还应包括错误报头（Error Headers）和错误报文（Error Body）两个参数，以报告错误信息。该报文和报头的大小不应使SDU超过双方选定的当前最大的接收单元。服务提供者可以不向服务使用者传递错误报文和错误报头。

当服务器拒绝会话或重定向会话时，使用的原语如图 14-4所示。服务使用者可以在会话建立之前就请求一个方法调用，相关的原语用虚线表示。

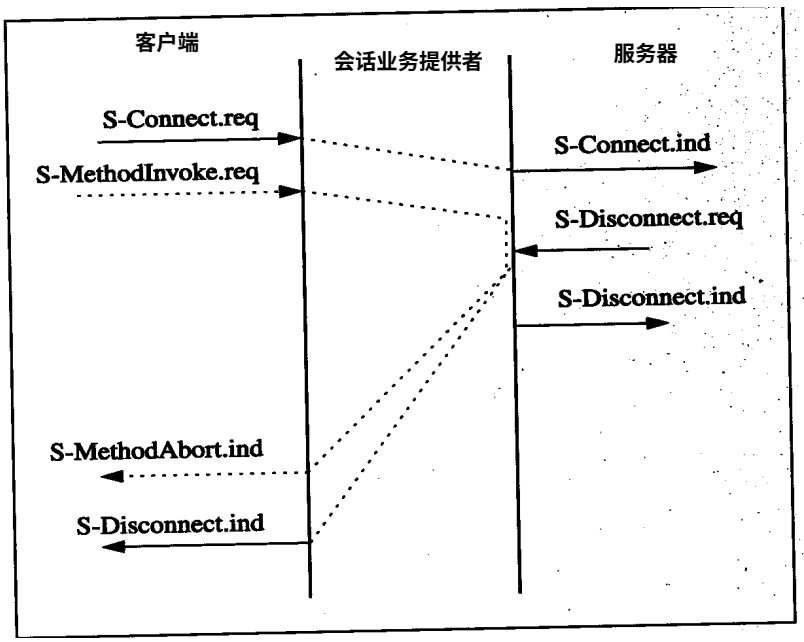

图14-4 拒绝会话建立

在会话的任何时间内，服务提供者均可产生一个断开指示。

图14-5给出了活动会话结束时所使用的原语。S－Disconnect.indication指示断开会话，并且不再发出任何指示。在S－Disconnect之前，服务提供者会放弃一切未完成的事务。

服务使用者必须随时为断开会话作好准备。如果用户希望继续通信，它必须重新建立会话并且重试被放弃的方法调用。

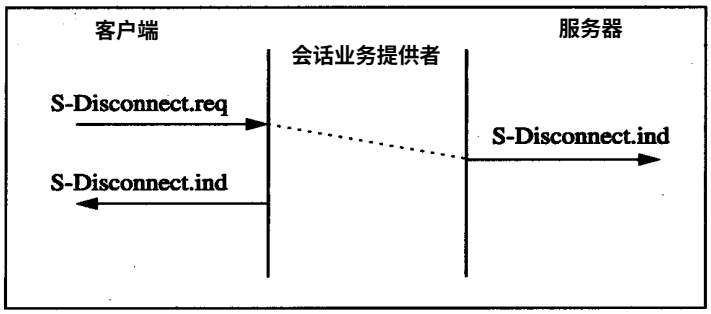

图14-5 活动会话结束

(3) S－Suspend

这个原语用来请求会话挂起，这样，在会话被恢复或被断开之前没有任何其他的活动发生。在会话被挂起时，会话服务提供者必须放弃所有的未完成方法和推事务处理。该原语是会话恢复工具的一部分（见表14-9）。

表14-9 原语S_SUSPEND及其含义

参 数	REQ	IND
原因	－	M

原因（reason）参数提供了挂起的原因。它由服务使用者提出请求，或者由服务提供者对它进行初始化。

图14-6给出了可能用到的原语流。

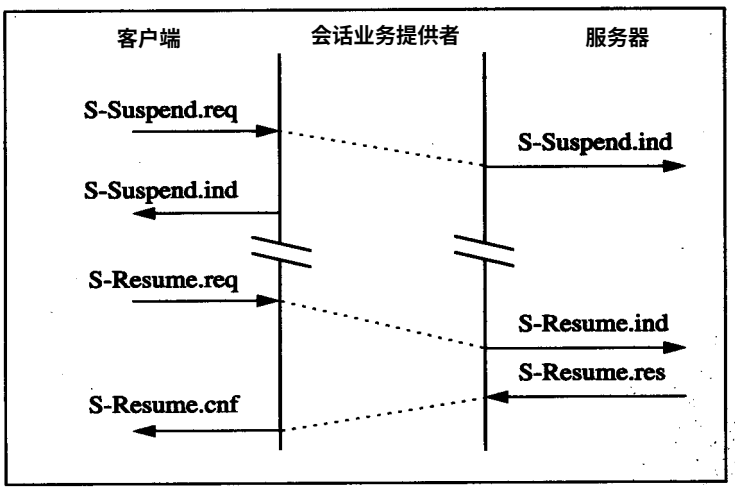

图14-6 会话挂起和恢复

特别是，当客户端不能响应数据推操作时，例如因为在低层的承载网络中客户端关闭了数据链路，那么客户端会挂起会话。S－Suspend.request可能会导致整个数据传输事务处理被立即放弃。

服务提供者可让已建立的会话随时被挂起。例如，在承载网络不能用的时候就可以挂起会话。图14-7给出了仅有一方被告知（在这个示例中是客户端）挂起会话的情况，当客户端试图恢复挂起的会话时，服务器拒绝请求并断开会话（例如，此时服务器可能认为正在使用的承载网络是不合适的）。

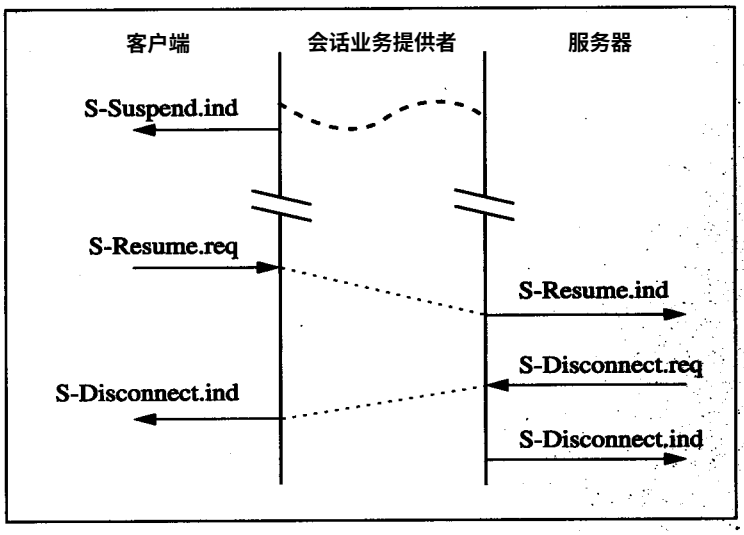

图14-7 服务提供者挂起会话并拒绝恢复

图14-8给出了在通知双方的服务使用者会话挂起时所发生的事件序列。在这个示例中，有一方服务使用者决定断开会话，而不是打算恢复会话。服务使用者随时可通过发起 S－

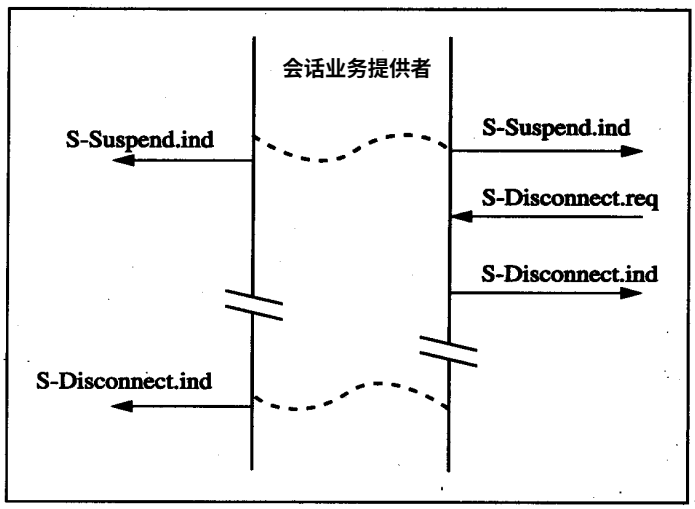

图14-8 挂起会话结束

Disconnect. request原语来关闭自己一方的会话。这时，由于服务双方有效的通信路径已经不存在，因此不必通知另一方。图 14-8表明，服务提供者最终会挂起会话，而何时挂起会话则是具体实现方面的事情。

(4) S－Resume

这个原语用于请求恢复会话，新的服务接入点由地址参数指明。该原语是会话恢复工具的一部分（见表14-10）。

表14-10 原语S-RESUME及其含义

参 数	Req	Ind	Res	cnf
服务器地址	M	M（=）	－	－
客户端地址	M	M（=）	－	－

服务器地址标识了与哪一方恢复会话。

客户端地址标识了当前会话发起点。

服务器地址和客户端地址均可以不同于会话被挂起前的地址。如果服务器地址与挂起前不同，服务使用者必须提供一个地址，以便联络先前使用的同一服务实例。

(5) S－Exception

这个原语用于报告一些事件，这些事件与具体的事务无关，也不会引起会话的断开和挂起，它是异常情况报告工具的一部分（见表 14-11）。

表14-11 原语S-EXCEPTION及其含义

参 数	IND
异常情况数据	M

异常情况数据（Exception Data）包含了来自服务提供者的信息。发生异常情况有很多原因：

- 低层传输状态的改变（例如，漫游出覆盖区）。
- 服务质量的改变。
- 安全层（Security Layer）状态的改变或出现问题。

(6) S－MethodInvoke

这个原语用于请求服务器执行一项操作。它只能与 S－MethodResult原语一起使用，是方法调用工具的一部分（见表14-12）。

表14-12 原语 S－METHODINVOKE及其含义

参 数	REQ	IND	RES	CNF
客户端事务标识符	M	－	－	M（=）
服务器事务标识符	－	M	M（=）	－
方法	M	M（=）	－	－
请求URI	M	M（=）	－	－
请求报头	O	C（=）	－	－
请求报文	C	C（=）	－	－

客户端的服务使用者可用客户端事务标识符（Client Transcation Id）来区分待处理的事务。

服务器的服务使用者可用服务器事务标识符（Server Transaction Id）来区分待处理的事务。

方法（Method）参数标识了被请求的操作：或者是HTTP方法[RFC2068]，或者是在能力协商过程中建立的扩展方法。

请求URI（Request URI）规定了对哪一个实体进行操作。

请求报头（Request Headers）是一个属性信息列表，语义上它等同于HTTP报头[RFC2068]。

请求报文（Request Body）是与请求有关的数据，语义上它等同于HTTP实体报文(HTTP entity body)。

如果请求方法（request Method）中没有说明可以使用实体报文，则不必提供请求报文参数[RFC2068]。

(7) S－MethodResult

这个原语用于对操作请求进行响应，只有在执行S－MethodInvoke原语操作后它才会被引发。它是方法发起工具的一部分（见表14-13）。

表14-13 原语 S－METHODRESULT及其含义

参 数	REQ	IND	RES	CNF
服务器事务标识符	M	－	－	M（=）
客户端事务标识符	－	M	M（=）	－
状态	M	M（=）	－	－
响应报头	O	C（=）	－	－
响应报文	C	C（=）	－	－
确认报头	－	－	O	P（=）

客户端服务使用者使用客户端事务标识符来区分待处理的事务。这个标识符必须与前面的S－MethodInvoke.request客户端事务ID相一致，以便随后产生的S－MethodResult.indiction能够使用它。

服务器的服务使用者可使用服务器事务标识符来区分待处理的事务。这个标识符必须与先前的S－MethodInvoke.response服务器事务ID相一致，以便随后产生的S－MethodResult.request能够使用它。

状态（Status）参数在语义上等同于HTTP状态代码[RFC2068]。

响应报头（Response Headers）参数是一个属性信息的列表，语义上等同于HTTP报头[RFC2068]。

响应报文（Response Body）参数是与响应有关的数据，语义上等同于HTTP实体报文。如果状态参数指明当前状态有错误，响应报文参数应该另外给出错误的信息，并显示给用户。

确认报头（Acknowledgment Headers）参数可用于给服务器返回信息。但是，提供者可能忽略这个参数或者只支持数量有限的信息的传输。

图14-9给出了在整个事务处理中用到的原语流。

只要放弃了事务处理，就会给服务使用者发送S－MethodAbort.indication。这个原语可代替先前的指示原语或确认原语，或者代替在它们后面将要出现的任何原语。一旦发出放弃原语，那么就不能再出现与该事务处理有关的原语。

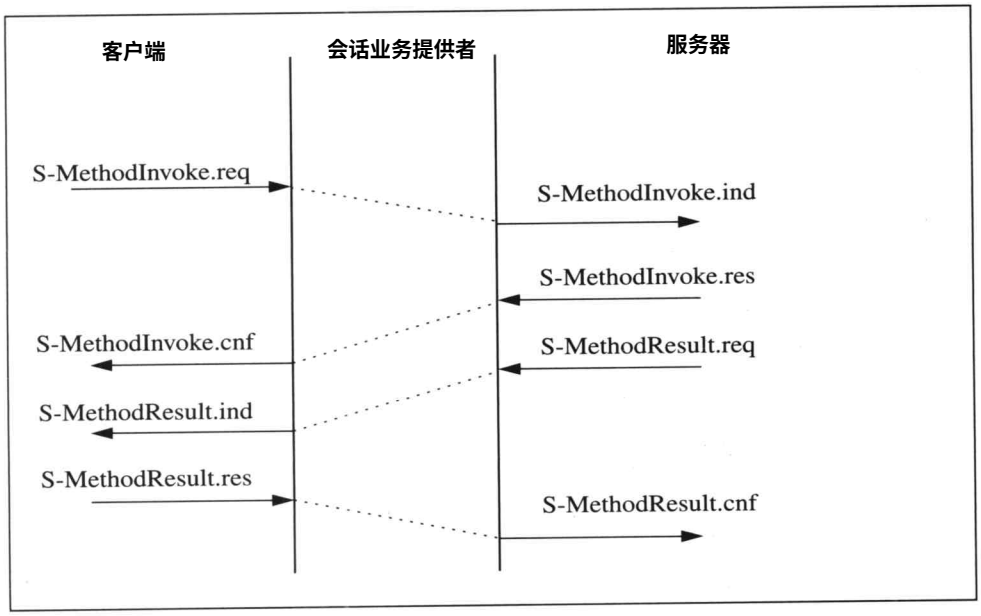

图14-9 完整的事务处理

会话层不支持多个方法的重叠发起，因为这样会导致指示原语的传递顺序与对应的请求原语传递顺序不同，这也同样适用于响应原语、确认原语以及对应的 S－MethodResult原语。图14-10给出了方法调用的结果与对应的原始请求传递顺序不同的情况（为了清楚起见，响应原语和确认原语不在其中）。

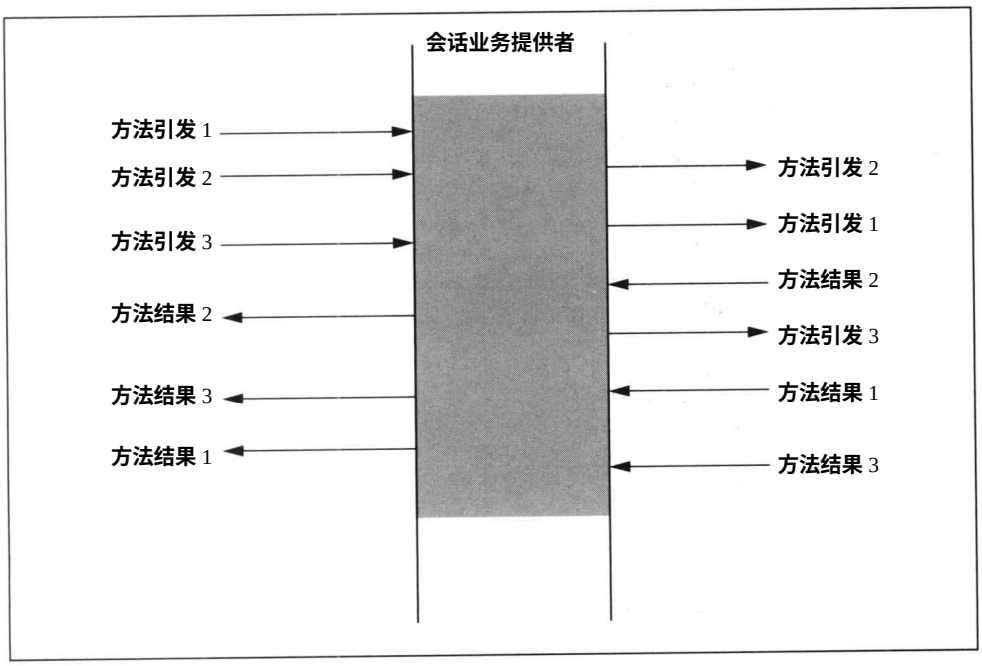

图14-10 无序异步请求

(8) S－MethodAbort

这个原语用于放弃尚未完成的操作请求。只有在S－MethodInvoke原语请求发生之后，才可以发起这个原语。该原语是方法调用工具的一部分（见表14-14）。

表14-14 原语S－METHODABORT及其含义

参 数	REQ	IND
事务处理标识符	M	M
原因	－	M

客户端的服务使用者在发起S－MethodAbort.request原语时，使用事务标识符区别待处理的事务。这个标识符必须与先前的S－MethodInvoke.request的客户端事务ID相一致，以便随后产生的S－MethodResult.response能够使用它。在这种情况下，服务器中S－MethodAbort.indication原语的事务ID，应与处理这个事务的服务器事务ID相一致。

服务器的服务使用者在引发S－MethodAbort.request原语时，使用事务处理标识符区别待处理的事务。这个标识符必须与先前的S－MethodInvoke.indication的服务器事务ID相一致，以便随后产生的S－MethodResult.confirm能够使用它。在这种情况下，服务器中S－MethodAbort.indication原语的事务ID要与处理该事务的客户端事务ID相一致。

原因参数给出放弃事务处理的原因。如果对方发起了S－MethodAbort.request，原因参数的类型是PEERREQ。

根据发出原语的时间，会出现两种情况。

第一种情况如图14-11所示。在S-MethodInvoke.indication原语发出之前，提交放弃请求，而此时方法调用正在同提供者进行通信，在这种情况下，并未通知用户，这个事务处理就被放弃了。

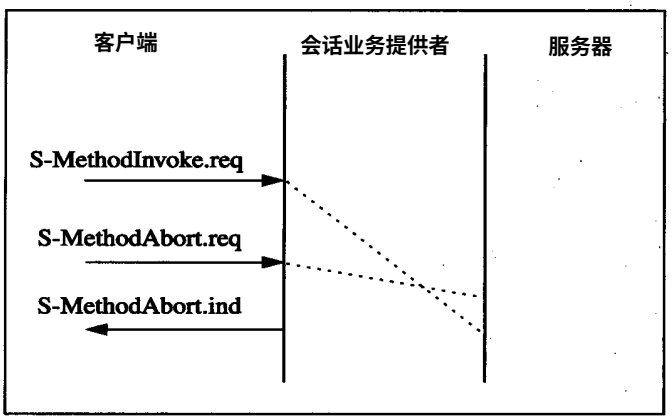

图14-11 在S-MethodInvoke.indication原语之前放弃事务处理

第二种情况如图14-12所示。放弃请求在S-MethodInvoke.indication原语发出后传送给服务提供者，在这种情况下，S-MethodAbort.indiction原语也将发生，这时应用不必再发起被放弃事务处理的S-MethodInvoke或S-MethodResult原语。

对于将要被放弃的事务处理来说，在S-MethodInvoke.request和S-MethodResult.response之间，客户端的S-MethodAbort原语会随时出现。同样，在S-MethodInvoke.indication和

S-MethodResult.confirm之间，服务器的S-MethodAbort原语也可能会随时出现。

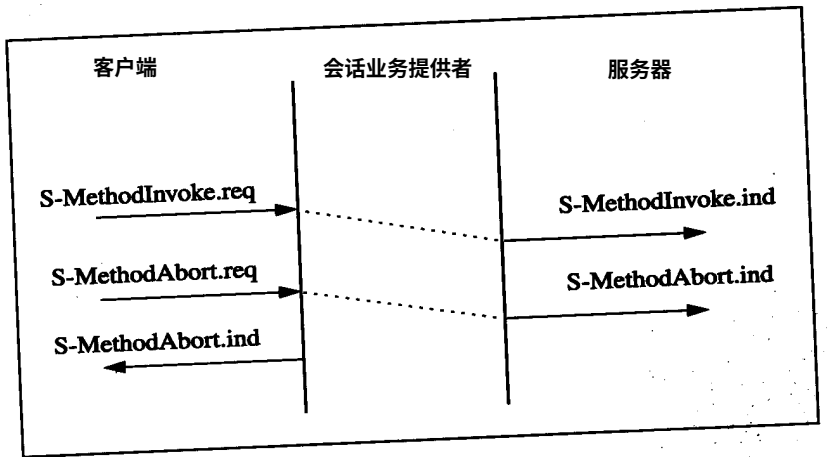

图14-12 在S-MethodInvoke.indication原语之后放弃事务处理

(9) S－Push

这个原语用于服务器在一个会话的上下文中以不确认的方法发出非请求信息。如图 14-13所示。该原语是PUSH工具的一部分（见表14-15）。

表14-15 原语S－PUSH及其含义

参 数	REQ	IND
PUSH报头	O	C（=）
PUSH报文	O	C（=）

如果被推的实体位置需要指出，那么为确保交互操作的正确性，在 PUSH报头中应包括内容位置报头（Content-Location header）[RFC2068]。

由于无法保证向对方传递信息的正确性，所以可能会出现图 14-14中的情景。

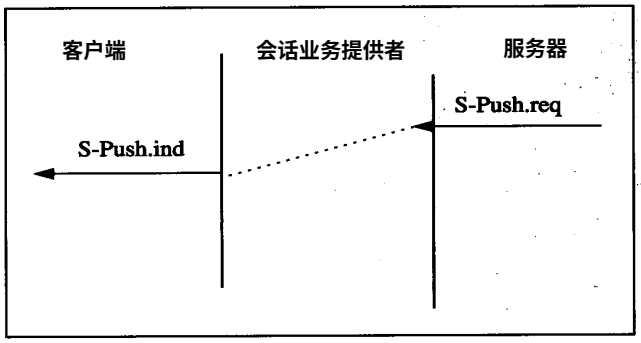

图14-13 不确认数据推原语

(10) S－ConfirmedPush

这个原语用于服务器在一个会话的上下文中以确认的方法发出非请求信息，如图1 4-14所示。该原语是经确认PUSH工具的一部分（见表14-16）。

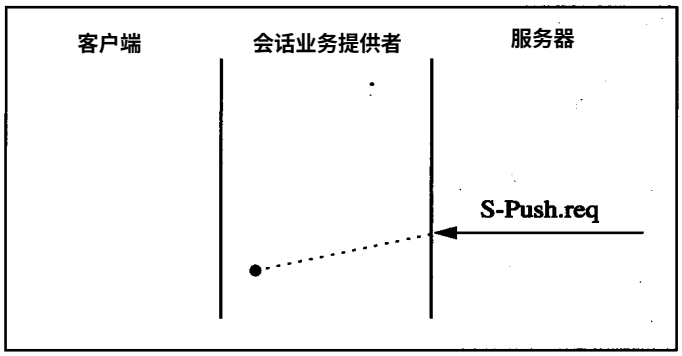

图14-14 失败的未确认数据推原语

表14-16 原语S－CONFIRMEDPUSH及其含义

参 数	REQ	IND	RES	CNF
服务器PUSH标识符	M	－	－	M（=）
客户端PUSH标识符	－	M	M（=）	－
PUSH报头	O	C（=）	－	－
PUSH报文	O	C（=）	－	－
确认报头	－	－	O	P（=）

服务器的服务使用者可用服务器PUSH标识符（Server Push Id）来区分待处理的推操作。

客户端的服务使用者可用客户端PUSH标识符（Client Push Id）来区分待处理的推操作。

如果被推的实体位置需要指出，那么为确保交互操作的正确性，在PUSH报头参数中应包括内容位置报头[RFC2068]。

确认报头参数用于向服务器返回信息，但是，提供者可以忽略这个参数或者仅支持数量有限的数据传输。

(11) S－PushAbort

这个原语用于拒绝一个推操作，它是确认PUSH工具的一部分（见图14-15和表14-17）。

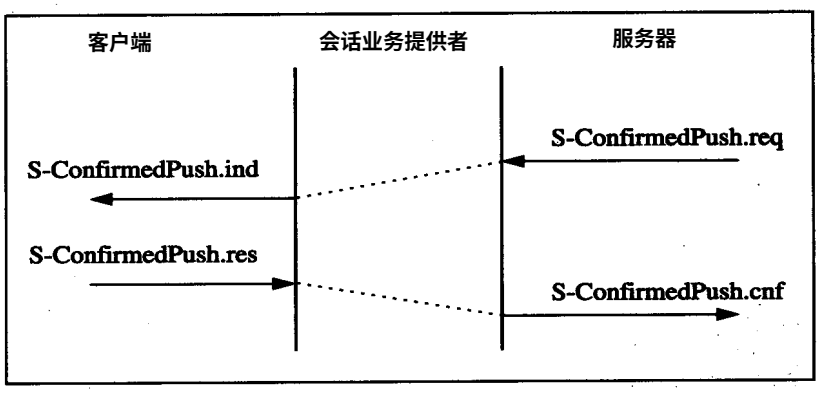

图14-15 确认数据推原语

表14-17 原语S－PUSHABORT及其含义

参 数	REQ	IND
客户端PUSH标识符	M	－
服务器PUSH标识符	－	M
原因	M	M（=）

客户端的服务使用者用客户端PUSH标识符（Client Push Id）来区分待处理的推操作，这个标识符必须与前面S－ConfirmedPush.indication原语的客户端PUSH标识符相一致。

服务器的服务使用者用服务器PUSH标识符（Server Push Id）来区分待放弃的推操作，这个标识符必须与前面由于被放弃而未被确认或指示的 ConfirmedPush.request原语的服务器PUSH标识符相一致。

原因（Reason）参数指明了放弃推操作的原因，这个值或者由对方的服务使用者提供，或者由服务提供者给出一个原因代码。

图14-16标识了S－PushAbort的使用方法，它只能在S－ConfirmedPush.indication之后被请求，代替了S－ConfirmedPush.response原语。

如果提供者发出放弃请求，即使用户未请求这个操作，同样要发出S－PushAbort.indication。

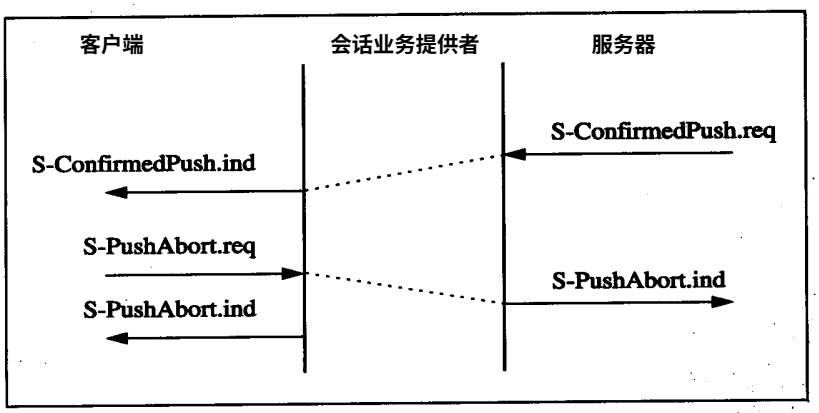

图14-16 放弃的确认数据推原语

4. 使用服务原语的限制

表14-18至表14-20定义了在服务接口上禁止使用的原语序列。因为服务是非对称的，所以客户端和服务器各有一个单独的列表。

表中仅列出了允许使用的原语，为了简单起见，省略了层前缀。表中的一些栏目解释如下：

客户端和服务器中事务的生存期由表14-21和表14-22定义。再次需要说明的是，表中只列出了允许使用的原语。

客户端和服务器的确认PUSH事务生存活期由表14-23和表14-24定义。再次需要说明的是，表中只列出了允许使用的原语。

表14-18 列表各项图示说明

表 项	描 述
—	不能产生指示和确认原语
N/A	发起这个原语是一个错误，下一步采取的动作与本地的具体实现有关
STATE_NAME	允许使用这个原语并转移业务接口界面到指定的状态
[1]	如果未完成事务处理的数目等于选定的 MOM请求的值，那么发起这个原语是错误的。下一步采取的动作与本地的具体实现有关，在得到允许之前传送该原语会被延迟
[2]	如果不存在与事务ID相匹配的未完成事务，那么发起该原语是错误的。下一步采取的动作与本地的具体实现有关。
[3]	如果能力协商中没有选择确认 PUSH工具，那么发起该原语是错误的。如果不存在与PUSH标识符相匹配的未完成 PUSH事务，那么发起该原语同样是错误的。下一步采取的动作与本地的具体实现有关
[4]	只有在能力协商中选择了 PUSH工具，才能使用
[5]	只有在能力协商中选择了确认推工具，才能使用
[6]	如果在能力协商中没有选择 PUSH工具，那么发起该原语是错误的。下一步采取的动作与本地的具体实现有关
[7]	如果在能力协商中没有选择确认 PUSH工具，那么发起该原语是错误的。下一步采取的动作与本地的具体实现有关
[8]	如果在能力协商中没有选择确认 PUSH工具，那么引发该原语是错误的。下一步采取的动作与本地的具体实现有关。同样， 如果未完成处理的PUSH事务数目等于选定的MOP请求的值，那么引发该原语是错误的。下一步采取的动作与本地的具体实现有关，在得到允许之前传送这个原语会被延迟
[9]	如果在能力协商中没有选择会话恢复工具，那么引发该原语是错误的。下一步采取的动作与本地的具体实现有关
[10]	只有在能力协商中选择了会话恢复工具，才能使用

表14-19 可用客户端会话层原语

客 户 端	会 话 状 态						
S－PRIMITIVE	无效	连接	接通	在关闭	正被挂起	已被挂起	恢 复
Connect.req	连接	N/A	N/A	N/A	N/A	N/A	N/A
Di sconnect.req	N/A	在关闭	在关闭	N/A	在关闭	在关闭	在关闭
Me thodInvoke.req	N/A	[1]	[1]	N/A	N/A	N/A	[1]
Me thodResult.res	N/A	N/A	[2]	N/A	N/A	N/A	N/A
Me thodAbort.req	N/A	[2]	[2]	N/A	N/A	N/A	[2]
ConfirmedPush.res	N/A	N/A	[3]	N/A	N/A	N/A	N/A
PushAbort.req	N/A	N/A	[3]	N/A	N/A	N/A	N/A
Suspend.req	N/A	N/A	挂起 [9]	N/A	N/A	N/A	挂起
Resume.req	N/A	N/A	恢复 [9]	N/A	恢复 [9]	恢复 [9]	N/A
Connect.cnf	—	接通	—	—	—	—	—
Exception.ind	—	连接	接通	在关闭	挂起	—	恢复
Di sconnect.ind	—	无效	无效	无效	无效	无效	无效
Me thodInvoke.cnf	—	—	接通	—	—	—	—
Me thodResult.ind	—	—	接通	—	—	—	—
Me thodAbort.ind	—	连接	接通	关闭	正被挂起	—	恢复
Push.ind	—	—	接通 [4]	在关闭[4]	正被挂起 [4]	—	—
ConfirmedPush.ind	—	—	接通 [5]	—	—	—	—
PushAbort.ind	—	—	接通 [5]	—	正被挂起 [5]	—	—
Suspend.ind	—	—	已被挂起 [10]	—	已被挂起[10]	—	已被挂起[10]
Resume.cnf	—	—	—	—	—	—	接通[10]

表14-20 可用服务器会话层原语

服务器	会话状态						
S-PRIMITIVE	无 效	连 接	接 通	在 关 闭	正被挂起	已被挂起	恢 复
Connect.res	N/A	接通	N/A	N/A	N/A	N/A	N/A
Disconnect.req	N/A	在关闭	在关闭	N/A	在关闭	无效	在关闭
MethodInvoke.res	N/A	N/A	[2]	N/A	N/A	N/A	N/A
MethodResult.req	N/A	N/A	[2]	N/A	N/A	N/A	N/A
MethodAbort.req	N/A	N/A	[2]	N/A	N/A	N/A	N/A
Push.req	N/A	N/A	[6]	N/A	N/A	N/A	N/A
ConfirmedPush.req	N/A	N/A	[8]	N/A	N/A	N/A	N/A
Resume.res	N/A	N/A	N/A	N/A	N/A	N/A	接通 [9]
Connect.ind	连接	—	—	—	—	—	—
Exception.ind	—	连接	接通	在关闭	正被挂起	—	恢复
Disconnect.ind	—	无效	无效	无效	无效	无效	无效
MethodInvoke.ind	—	—	接通	—	—	—	—
MethodResult.cnf	—	—	接通	—	—	—	—
MethodAbort.ind	—	—	接通	在关闭	正被挂起	—	—
ConfirmedPush.cnf	—	—	接通 [5]	—	接通 [5]	—	—
PushAbort.ind	—	—	接通 [5]	在关闭[5]	接通 [5]	—	—
Suspend.ind	—	—	已被挂起 [10]	—	已被挂起 [10]	—	已被挂起 [10]
Resume.ind	—	—	恢复 [10]	—	恢复	恢复 [10]	—

表14-21 可用客户端事务处理原语

客 户 端	事务状态				
S—PRIMITIVE	无 效	请 求	等 待	完 成	放 弃
MethodInvoke.req	请求	N/A	N/A	N/A	N/A
MethodResult.res	N/A	N/A	N/A	无效	N/A
MethodAbort.req	N/A	放弃	放弃	放弃	N/A
MethodInvoke.cnf	—	等待	—	—	—
MethodResult.ind	—	—	完成	—	—
MethodAbort.ind	—	无效	无效	无效	无效

表14-22 可用服务器事务处理原语

服务器	事务状态				
S—PRIMITIVE	无 效	请 求	处 理	应 答	放 弃
MethodInvoke.res	N/A	处理	N/A	N/A	N/A
MethodResult.req	N/A	N/A	应答	N/A	N/A
MethodAbort.req	N/A	放弃	放弃	放弃	N/A
MethodInvoke.ind	请求	—	—	—	—
MethodResult.cnf	—	—	—	无效	—
MethodAbort.ind	—	无效	无效	无效	无效

表14-23 可用服务器经确认推原语

服务器 S－PRIMITIVE	确认PUSH状态 无效	推
ConfirmedPush.req	推	N/A
ConfirmedPush.cnf	－	无效
PushAbort.ind	－	无效

表14-24 可用客户端经确认推原语

客户端 S－PRIMITIVE	无效	确认PUSH状态 接收	放弃
ConfirmedPush.res	N/A	无效	N/A
PushAbort.req	N/A	放弃	N/A
ConfirmedPush.ind	接收	－	－
PushAbort.ind	－	无效	无效

5. 错误处理

连接模式的会话服务提供者把错误和其他异常情况处理分为四级：

- 如果异常情况与特定的事务无关，那么它只用异常情况工具提出报告，并不干扰整个会话状态。
- 与某一特定的事务有关的错误，会用适当的原因代码引起方法或 PUSH放弃指示，并不干扰整个会话状态。
- 如果选择了会话恢复工具，那么禁止会话双方进行通信的情况会引发挂起指示。如果没有选择会话恢复工具，会引发断开指示。
- 其他错误会以相应的原因代码引起断开指示。

某些竞争的情况会把一个方法或 PUSH事务的放弃原因代码报告成断开 (DISCONNECT)，但这不能被解释成表示会话断开，会话断开通常只能使用 S－Disconnect原语表示。

14.3.4 无连接模式会话服务

1. 概述

无连接模式会话服务利用无确认的工具，完成层用户之间内容实体的交换，所提供的服务是非对称的，类似于连接模式的会话服务。

在无连接模式中，只有方法调用工具和 PUSH工具是可用的。由于这些工具不带确认，所以对等实体间的通信可能并不可靠。

2. 服务原语

定义服务原语所使用的类型请参考服务原语参数类型等有关章节。

(1) S-Unit-MethodInvoke

在服务器中，这个原语被用于发起一种无确认的方法，它是方法调用工具的一部分（见表14-25）。

表14-25 原语S-UNIT-METHODINVOKE及其含义

参数	REQ	IND
服务器地址	M	M（=）
客户端地址	M	M（=）
事务标识符	M	M（=）

（续）

参 数	REQ	IND
方法	M	M（=）
请求URI	M	M（=）
请求报头	O	C（=）
请求报文	C	C（=）

服务器地址（Server Address）标明了请求发往哪一方。

客户端地址（Client Address）标明了请求的发起方。

服务使用者可以利用事务标识符（Transaction Id）区分不同的事务。这个事务标识符在服务用户之间透明地传输。

方法（Method）标明了请求的操作，它必须是HTTP方法[RFC2068]之一。

请求URI（Request URI）规定了操作实施的实体对象。

请求报头（Request Headers）是一个属性信息的列表，语义上等同于HTTP报头[RFC2068]。

请求报文（Request Body）是与请求相关的数据，语义上等同于HTTP实体报文。如果请求的方法参数中没有规定允许有实体报文，则原语中不应有请求报文参数[RFC2068]。

(2) S-Unit-MethodResult

该原语用于返回服务器方法引发的结果，但未予确认，它是方法引发工具的一部分（见表14-26）。

表14-26 原语S-UNIT-METHODRESULT及其含义

参 数	REQ	IND
客户端地址	M	M（=）
服务器地址	M	M（=）
事务标识符	M	M（=）
状态	M	M（=）
响应报头	O	C（=）
响应报文	C	C（=）

客户端地址标识了结果发往哪一方。

服务器地址标识了发送结果的发起方。

服务使用者利用事务标识符区别不同的事务。

状态（Status）语义上等同于HTTP状态代码[RFC2068]。

响应报头是一个属性信息列表，语义上等同于HTTP报文[RFC2068]。

响应报文（Response Body）是与响应有关的数据，语义上等同于HTTP实体报文。如果状态参数表明出现了错误，响应报文应提供关于这个错误的补充说明信息，并显示给用户。

(3) S-Unit-Push

该原语用于从服务器到客户端以未经确认的方法发送非请求信息，它是推工具的一部分（见表14-27）。

表14-27 原语S-UNIT-PUSH及其含义

参 数	REQ	IND
客户端地址	M	M（=）
服务器地址	M	M（=）
PUSH标识符	M	M（=）

（续）

参 数	REQ	IND
PUSH报头	O	C（=）
PUSH报文	O	C（=）

客户端地址标识了PUSH将要发往哪一方。

服务器地址标识了PUSH的发起方。

服务器用户使用PUSH标识符（Push Id）来区分不同的推操作。

如果被推实体的位置需要指出，那么为确保操作的正确，在 PUSH报头参数中应包括内容位置头[RFC2068]。

3. 使用服务原语的限制

一旦底层为通信作好准备，服务使用者可以随时发起允许的请求原语。管理实体之间如何进行相应交互操作超出了本规范的范围。服务使用者不遵守这些规定会造成错误，下一步采取的动作与本地的具体实现有关。

服务提供者在得知对等端用户实体发起了一个请求原语后，应发出一个指示原语。

表14-28定义了客户端和服务器允许发起的原语。

不遵守这些规定会造成错误，下一步采取的动作与本地的具体实现有关。

表14-28 无连接服务原语

	类 型				
一般名称	REQ	IND	RES	CNF	描 述
S－Unit－MethodInvoke	C	S	－	－	在服务器引发一个原语，不需确认
S－Unit－MethodResult	S	C	－	－	从服务器返回应答，不需确认
S－Unit－Push	S	C	－	－	推内容，不需确认。
－ 可能不发生原语					
C 可能在客户端发生原语					
S 可能在服务器发生原语					

4. 错误处理

如果请求不能传递给对方的提供者，那么无连接模式会话服务提供者就不能产生任何指示原语。异常情况的检测和如何进行正确操作是具体实现方面的事情。

14.4 WSP/B协议操作

本节描述了会话服务双方所用的协议，以实现抽象服务接口定义描述的功能。

14.4.1 连接模式WSP/B

这一节描述了在WTP事务服务[WAPWTP]之上的WSP/B操作。

1. WTP的使用

每一个WSP工具使用的WTP事务级别总结在表14-29中。

表14-29 WTP的使用

WSP 工具	WTP事务级别	WSP 工具	WTP事务级别
会话管理	级别0和级别2	PUSH	级别0
方法调用	级别2	确认PUSH	级别1
会话恢复	级别0和级别2		

连接模式的WSP/B客户端必须支持WTP级别0和级别2事务的初始化，客户端也应能接收从服务器发起的级别0事务请求，以使服务器能够明确地断开会话。如果客户端支持PUSH工具，那么客户端必须接收的事务处理应是表14-29定义PUSH工具使用的级别。

2. 协议描述

以下各图标识了通过会话工具使用事务处理服务。协议的具体细节参考下面14.4.1节中的状态表，图表与状态表之间的任何不符之处皆以状态表为准。

短箭头代表了WTP协议消息，把确认PDU和WSP/B PDU作为数据来传输，平行箭头所表示的消息可能会被合成为一个传输数据报来传输。

(1) 会话管理工具

图14-17标识了一般会话的创建过程，期间没有发生任何错误和重定向。

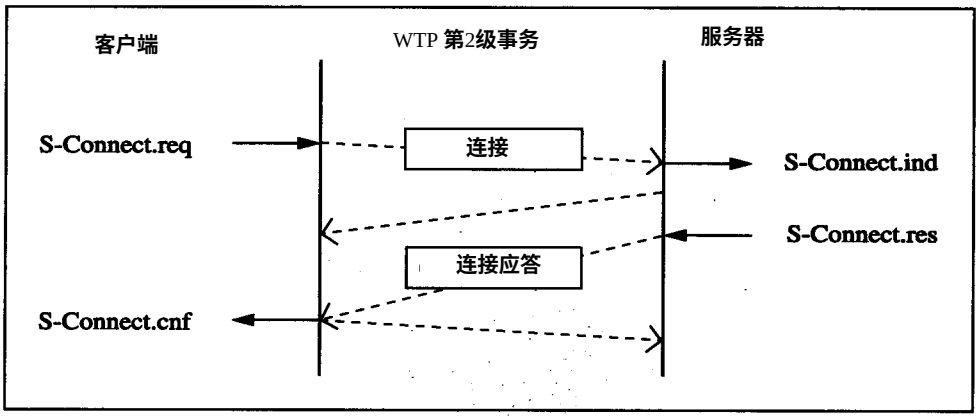

图14-17 一般会话创建过程

图14-18标识了客户端的会话创建被重定向至另一个服务器。

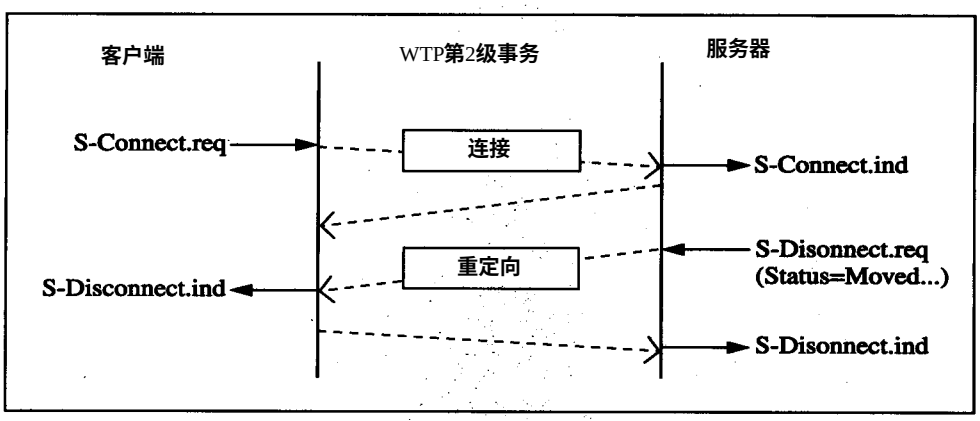

图14-18 带有重定向的会话创建

图14-19标识的会话创建中服务器会话用户拒绝接受该会话。

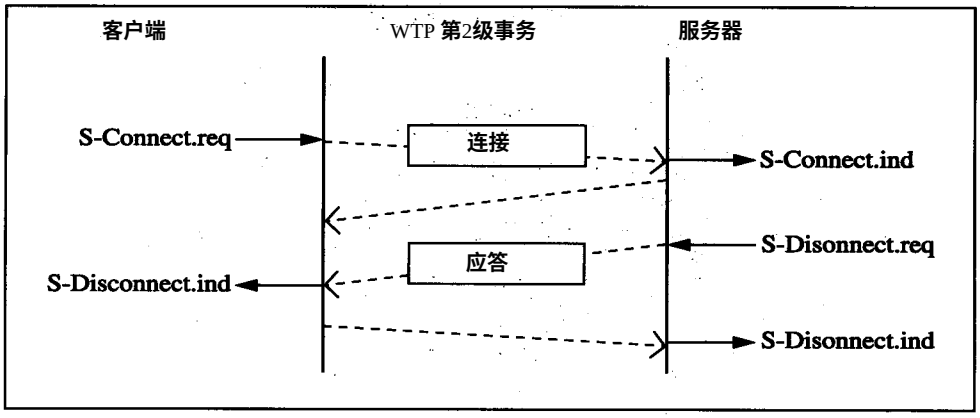

图14-19 带有服务器错误的会话创建

图14-20标识了会话结束。

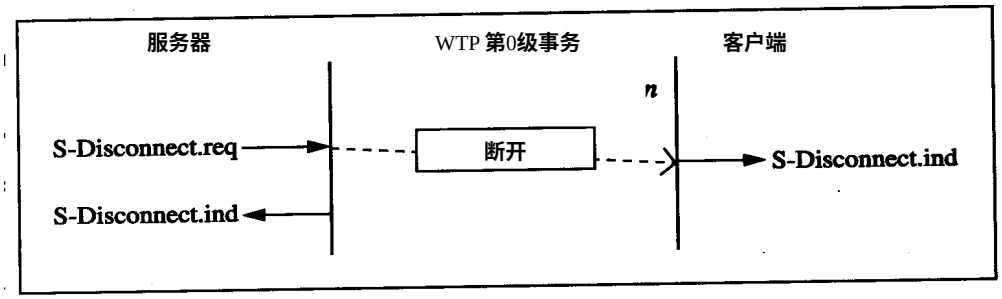

图14-20 会话结束

(2) 会话恢复工具

图14-21标识了会话挂起。

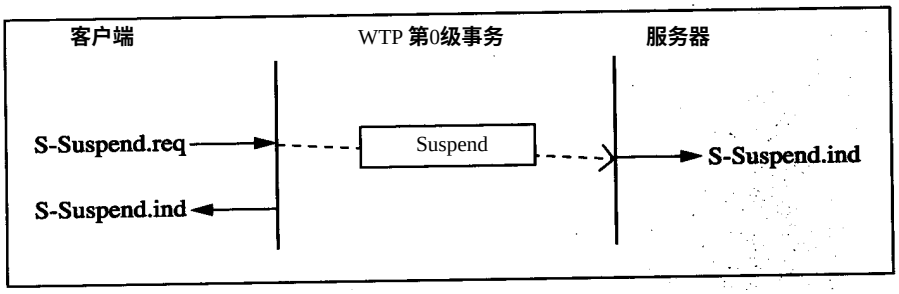

图14-21 会话挂起

图14-22标识了会话的成功恢复。

图14-23表明在会话恢复时服务器会话用户拒绝恢复该会话的情况。

(3) 方法引发工具

图14-24标识了方法引发。

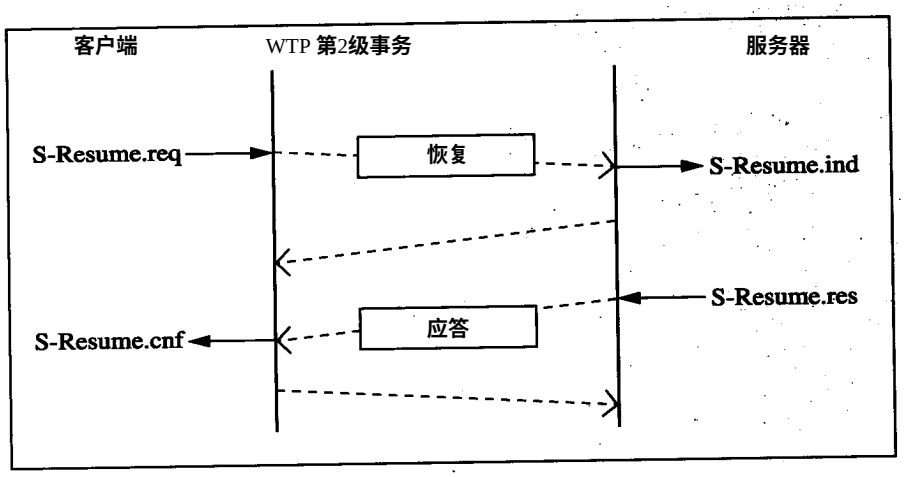

图14-22 正常会话恢复

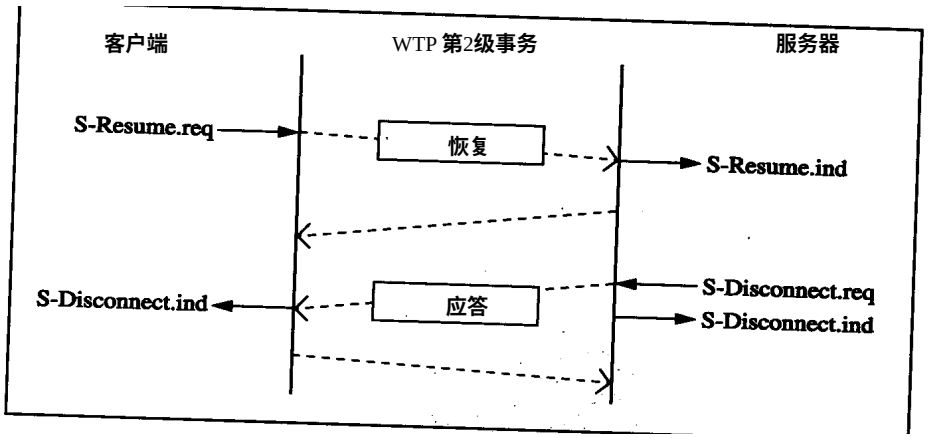

图14-23 带有服务器错误的会话恢复

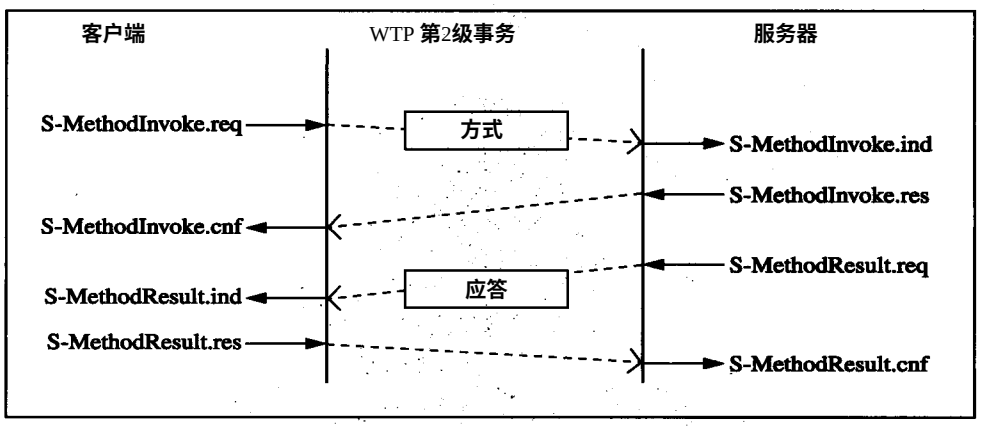

图14-24 正常方法引发

(4) 推工具

图14-25标识了一个未予确认的推。

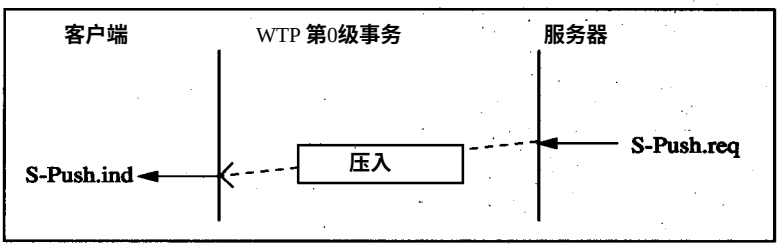

图14-25 推引发

(5) 经确认推工具

图14-26标识了一个经确认的推。

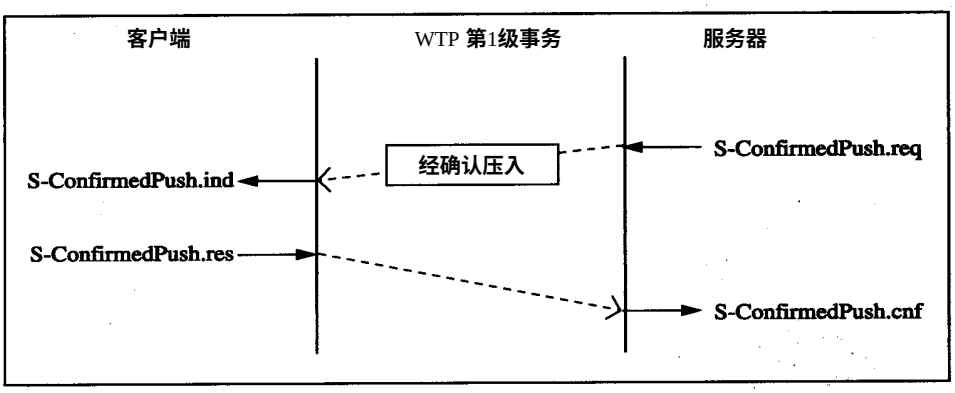

图14-26 经确认的推引发

3. 协议参数

协议状态机使用如下参数。

(1) Maximum Receive Unit（MRU）

MRU是会话层能接受低层服务提供者最大 SDU的大小。在14.5.3节中的“默认性能”规定了其初始值被置为默认的SDU大小，该值可在能力协商中修改。

(2) Maximum Outstanding Method Requests （MOM）

MOM是指在给定的时间内未完成方法请求事务处理的最大数目。在14.5.3节中的“默认性能”规定了其初始值被置为默认的MOM大小，该值可在能力协商中修改。

(3) Maximum Outstanding Push Requests （MOP）

MOP是指在给定的时间内未完成推请求事务处理的最大数目。 在14.5.3节中的“默认性能”规定了其初始值被置为默认的MOP大小，该值可在能力协商中修改。

4. 变量（Variables）

协议状态机使用如下变量。

(1) N_Methods

N_Methods记录在服务器进行的方法事务处理的数量。

(2) N_Pushes

N_Pushes记录在客户端进行的推事务处理的数量。

(3) Session_ID

Session_ID作为客户端和服务器的会话标识符由服务器指定。

用来指定标识符的方法在所用的传输网络中传递消息期间不应被重复。否则，会话管理逻辑会被破坏。

5. 事件处理

会话与后面的地址四重组有关：客户端地址、客户端端口、服务器地址和服务器端口。根据地址四重组，把将要进行的事务处理指定给某一个会话，所以，地址四重组是一个会话唯一真正的协议级别标识符。一个会话一次只能绑定一个地址四重组。

根据某个地址四重组创建一个新会话，但该会话已存在，这时，服务器会话提供者必须允许该会话的重新创建。它作为会话另一个受限制的实例用来在服务器进行会话创建事务处理（例如，Connect和ConnectReply PDU），这一点在下表中有详细说明。

事务处理层的指示和确认被称作事件，根据协议状态表置某一事件有效并执行这个事件。协议状态表同样可使用伪事件来引发协议实现本身状态的改变，被认为合理的伪事件由协议状态机或是由实现本身产生。例如，伪事件可以表示某一管理操作，来断开长时间不活动的会话。伪事件名字由斜体字表示，其定义见表 14-30。

表14-30 伪事件及其定义

伪 事 件	描 述
*Abort*放弃	放弃一个方法或推事务处理
*Release*释放	允许执行一个方法事务处理
*Suspend*挂起	挂起会话
*Disconnect*断开	断开会话

新的事务请求在执行之前根据状态表被置为有效。如果不再进行其他操作，只进行如下测试，将根据状态表进行事件处理（见表 14-31）。

表14-31 测试及其处理

测 试	动 作
TR-Invoke.ind SDU大小 > MRU	TRAbort.req放弃该 TR-Invoke，原因是MRUEXCEEDED
级别2 TR-Invoke.ind，在服务器上，Connect PDU	1) 创建一个新的原始会话（proto－session），负责进行剩下的连接事务处理 2) 原始会话发出 S-Connect.indication通知会话用户 3) 如果会话用户通过引发 S-Connect.response接受这个新的会话，则 proto－session就会成为这个新的会话，其地址为这个相应的地址四重组。而绑定该地址四重组的任一个旧的会话会被断开
级别2 TR-Invoke.ind，在服务器上，Resume PDU	在会话恢复PDU中，传递给由SessionId标识的会话，而不是由地址四重组标识的会话。如果 SessionId并不有效，例如，该会话不存在，则 TR-Abort.req断开该 TR-Invoke

（续）

测 试	动 作
级别1-2 TR-Invoke.ind，没有会话与 地址四重组匹配	TR-Abort.req断开该 TR－Invoke，原因是DISCONNECT
级别1-2 TR-Invoke.ind PDU不在状态表中	TR-Abort.req禁止执行该 TR-Invoke，原因是PROTOERR
级别0 TR－Invoke.ind PDU不在状态表中	忽略
不在状态表中的任何其他事件	如果是放弃以外的其他事务处理，则执行 TR－Abort.req，原因是PROTOERR 放弃所有的方法和推事务处理，原因是 PROTOERR 执行S-Disconnect.ind，原因是PROTOERR

由低层事务处理层提供地服务并不能可靠地探查到对方已经丢弃了会话状态信息，除非进行了方法或推事务处理，这会最终造成大量不再具有通信对方的会话存在，具体实现应该能够断开处在这种状态的会话。

6. 状态表

下面的状态表定义了连接方法 WSP/B的活动。因为多重的方法和推事务处理可以同时发生，所以为客户端和服务器定义了三种状态表：一种定义是为了会话状态，一种是为了方法状态，另一种是为了推状态。

表中所用的状态与为抽象业务接口定义的状态尽管名字相似，但逻辑上是完全不同的。特别是业务接口中的某一状态以相同的名字映射到一个协议状态，同样可以映射到多个协议的状态或根本没有协议的状态。

单个的事件在条件列中可能有几个表项，在这种情况下，条件应该从上到下一行一行的作比较，把最特殊的条件放在最前面。单重条件表项可能包含几个条件，这时要用逗号分开，在这种情况下，为使条件正确应遵守以上规定。

(1) 客户端会话状态表

表14-32～表14-36分别标识了会话状态和客户端使用事务处理服务时发生的事件处理：

表14-32 客户端会话无效

事 件	条 件	动 作	下一状态
S－Connect.req		断开和该地址四重组相关的所有会话 TR－Invoke.req（级别2， Connect）N_PUSHES = 0	连接

表14-33 客户端会话连接

事 件	条 件	动 作	下一状态
S-Disconnect.req		TR－Abort.req（原因是 DISCONNECT） 断开连接 放弃（原因是 DISCONNECT）所有未完成方法事务处理 S－Disconnect.ind（USERREQ）	无效
Disconnect		TR－Abort.req断开连接，原因是DISCONNECT 放弃所有未完成方法事务处理 S－Disconnect.ind，原因是DISCONNECT	无效

（续）

事 件	条 件	动 作	下一状态
S-MethodInvoke.req		开始一个新的方法事务处理（参考方法状态表）	
S-MethodAbort.req		参考方法状态表	
Suspend		TR-Abort.req挂起连接 放弃所有的方法事务处理 S－Disconnect.ind，原因是SUSPEND	无效
TR-Invoke.ind	级别1，ConfirmedPush PDU	TR－Abort.req断开该TR-Invoke	
TR-Result.ind	连接事务处理，SDU 大小>MRU	执行TR－Abort.req，原因是MRUEXCEEDED 放弃所有的未完成方法事务处理，原因是CONNECTERR S-Disconnect.ind，原因是MRUEXCEEDED	无效
	连接事务处理，ConnectReply PDU	TR－Result.res Session_ID = SessionId from PDU S－Connect.cnf	接通
	连接事务处理，Redirect PDU	TR－Result.res 放弃所有未完成方法事务处理，原因是CONNECTERR S-Disconnect.ind（Redirect参数）	无效
	连接事务处理，Reply PDU	TR－Result.res 放弃所有方法事务处理，原因是 CONNECTERR S-Disconnect.ind（Reply参数）	无效
	其他	TR－Abort.req，原因是PROTOERR 放弃所有未完成方法事务处理，原因是CONNECTERR S-Disconnect.ind，原因是PROTOERR	无效
TR-Invoke.cnf	连接事务处理，方法事务处理	忽略 放弃方法事务处理，原因是DISCONNECT	
TR－Abort.ind	连接事务处理	放弃所有的未完成方法事务处理，原因是CONNECTERR S－Disconnect.ind(放弃原因)	无效
	方法事务处理	参考方法状态表	

表14-34 客户端会话接通

事 件	条 件	动 作	下一状态
S－Disconnect.req		放弃所有方法和推事务处理，原因是 DISCONNECT TR－Invoke.req，原因是级别0，Disconnect S－Disconnect.ind，原因是USERREQ	无效
Disconnect		放弃所有方法和推事务处理，原因是DISCONNECT S-Disconnect.ind，原因是DISCONNECT	无效
S-MethodInvoke.req		开始一个新的方法事务处理	

（续）

事 件	条 件	动 作	下一状态
S-MethodResult.res		参考方法状态表	
S-MethodAbort.req		参考方法状态表	
S-ConfirmedPush.res		参考推状态表	
S-PushAbort.req		参考推状态表	
S-Suspend.req		放弃所有方法和推事务处理，原因是SUSPEND TR-Invoke.req，原因是级别0，Suspend S-Suspend.ind，原因是USERREQ	已被挂起
Suspend	会话挂起工具无效	放弃所有方法和推事务处理，原因是SUSPEND S-Disconnect.ind，原因是SUSPEND	无效 已被挂起
	会话挂起工具有效	放弃所有方法和推事务处理，原因是SUSPEND S－Suspend.ind，原因是SUSPEND	
S-Resume.req		放弃所有方法和推事务处理，原因是USERREQ 捆绑会话至新的通信地址四重组 TR－Invoke，原因是级别2，Resume	恢复
TR－Invoke.ind	级别0，Disconnect PDU	放弃所有方法和推事务处理，原因是DISCONNECT S－Disconnect.ind，原因是DISCONNECT	无效
	级别0，Push PDU，推工具有效	S-Push.ind 开始一个新的推事务处理	
	级别1，Confirmed-PushPDU，经确认推工具有效		
TR-Result.ind	方法事务处理	参考方法状态表	
TR-Invoke.cnf	方法事务处理	参考方法状态表	
TR-Abort.ind	方法事务处理 推事务处理	参考方法状态表 参考推状态表	

表14-35 客户端会话挂起

事 件	条 件	动 作	下一状态
S-Disconnect.req		S-Disconnect.ind，原因是USERREQ	无效
Disconnect		S-Disconnect.ind，原因是DISCONNECT	无效
S-Resume.req		TR-Invoke.req，原因是级别2， Resume	恢复
TR－Invoke.ind	级别0，Disconnect PDU 级别1，ConfirmedPushPDU，被确认推工具有效	S-Disconnect.ind，原因是DISCONNECT TR－Abort.req放弃该 TR－Invoke，原因是SUSPEND	无效
TR－Invoke.cnf	忽略		
TR－Abort.ind	忽略		

表14-36 客户端会话恢复

事 件	条 件	动 作	下一状态
S－Disconnect.req		TR－Abort.req断开恢复操作，原因是DISCONNECT 放弃所有未完成方法事务处理，原因是DISCONNECT S-Disconnect.ind，原因是USERREQ	无效
Disconnect		TR-Abort.req断开恢复操作，原因是DISCONNECT 放弃所有未完成方法事务处理，原因是DISCONNECT S-Disconnect.ind，原因是DISCO-NNECT	无效
S-MethodInvoke.req		开始一个新的方法事务处理（参考方法状态表）	
S-MethodAbort.req		参考方法状态表	
S-Suspend.req		TR-Abort.req挂起恢复操作，原因是SUSPEND 放弃所有未完成方法事务处理，原因是S USPEND TR-Invoke.req，原因是级别 0，Suspend S-Suspend.ind原因是USERREQ	已被挂起
Suspend		TR-Abort.req挂起恢复操作，原因是SUSPEND 放弃所有未完成方法事务处理，原因是SUSPEND S-Suspend.ind，原因是SUSPEND	已被挂起
TR-Invoke.ind	级别0，Disco-nnect PDU	TR-Abort.req断开恢复操作，原因是DISCONNECT 放弃所有未完成方法事务处理，原因是DISCONNECT S-Disconnect.ind，原因是DISCONNECT	无效
	级别1，Confir-medPush PDU，经确认推工具有效	TR－Abort.req挂起TR－Invoke，原因是SUSPEND	
	恢复事务处理，		
TR－Result.ind	SDU尺寸> MRU	TR－Abort.req放弃 TR－Result，原因是MRUEXCEEDED 放弃所有未完成方法事务处理，原因是SUSPEND S-Suspend.ind，原因是 MRUEX-CEEDED	无效
	恢复事务处理 Reply PDU（st-atus ＝OK）	TR-Result.res S-Resume.cnf	接通
	恢复事务处理 Reply PDU（st-atus ! ＝OK）	TR-Result.res 放弃所有未完成方法事务处理，原因是DISCONNECT	无效

(续)

事 件	条 件	动 作	下一状态
		TR-Abort.req放弃TR－Result，原因是PROTOERR	已被挂起
	其他	放弃所有未完成方法事务处理，原因是SUSPEND S-Suspend.ind，原因是PROTOERR	
TR-Invoke.cnf	恢复事务处理	忽略	
	方法事务处理	放弃方法事务处理，原因是SUSPEND	
	恢复事务处理 Reason ＝ DISC-ONNECT	放弃所有方法事务处理，原因是DISCONNECT S-Disconnect.ind，原因是DISCONNECT	无效
TR-Abort.ind	恢复事务处理	放弃所有未完成方法事务处理，原因是SUSPEND S-Suspend.ind（放弃原因）	已被挂起
	方法事务处理	参考方法状态表	

(2) 客户端方法状态表

表14-37～表14-40分别标识了方法状态和客户端使用事务处理服务时发生的事件处理：

表14-37 客户端方法NULL

事 件	条 件	动 作	下一状态
S-MethodInvoke.req		TR-Invoke.req（级别2， Method） 注：“方法”意味着可以使用分配给特定方法的PDU类型得到或传递PDU	请求

表14-38 客户端方法请求

事 件	条 件	动 作	下一状态
S-MethodAbort.req		TR-Abort.req放弃该方法，原因是PEERREQ S-MethodAbort.ind（原因是USERREQ）	无效
Abort		TR-Abort.req（放弃原因）该方法 S-MethodAbort.ind，原因是USERREQ	元效
TR－Invoke.cnf		S-MethodInvoke.cnf	等待
TR－Abort.ind	Reason ＝DISCONNECT	断开会话	无效
	Reason ＝SUSPEND	挂起会话	无效
	其他	S－MethodAbort.ind（放弃原因）	无效

表14-39 客户端方法保持

事 件	条 件	动 作	下一状态
S-MethodAbort.req		TR-Abort.req放弃该方法，原因是PEERREQ S-MethodAbort.ind，原因是USERREQ	无效
Abort		TR-Abort.req（放弃原因）方法 S-MethodAbort.ind（放弃原因）	无效

(续)

事 件	条 件	动 作	下一状态
TR-Result.ind		TR－Abort.req（MRUEXCEEDED）	无效
	SDU size>MRU		
		S-MethodAbort.ind （MRUEXCEEDED）	完成
	响应PDU	S－MethodResult.ind	
	其他	TR－Abort.req（PROTOERR）	无效
		S－MethodAbort.ind（PROTOERR）	
TR－Abort.ind	Reason＝DISCONNECT	断开会话	无效
	Reason＝SUSPEND	挂起会话	无效
	其他	S-MethodAbort.ind（放弃原因）无效	

表14-40 客户端方法完成

事 件	条 件	动 作	下一状态
S-MethodResult.res		TR-Result.res（退出信息 =确认报头）	无效
		注意：支持确认报头是可选的	
S-MethodAbort.req		TR－Abort.req放弃该方法，原因是PEERREQ	无效
		S-MethodAbort.ind，原因是USERREQ	
Abort		TR-Abort.req（放弃原因）放弃该方法	无效
		S-Method Abort.ind（放弃原因）	
TR－Abort.ind	Reason＝DISCONNECT	断开会话	无效
	Reason＝SUSPEND	挂起会话	无效
	其他	S-MethodAbort.ind（放弃原因）	无效

(3) 客户端推状态表

表14-41和表14-42分别标识了推状态和客户端使用事务处理服务时发生的事件处理。

表14-41 客户端推无效

事 件	条 件	动 作	下一状态
TR－Invoke.ind	级别1，ConfirmedPush PDU，N_PUSHES ＝MOP	TR－Abort.req放弃该TR－Invoke，原因是MOREXCEEDED	无效
	级别1，ConfirmedPush PDU，N_PUSHES < MOP	增加N_PUSHES S-ConfirmedPush.ind	接收

表14-42 客户端推接收

事 件	条 件	动 作	下一状态
S-ConfirmedPush.res		TR-Invoke.res（退出信息=确认报头）	无效
		注意：是否支持确认报头是可选项	
		减少N_PUSHES	
S－PushAbort.req		TR-Abort.req（放弃原因）该TR－Invoke	无效
		减少N_PUSHES	
Abort		TR－Abort.req（放弃原因）该TR－Invoke	无效
		减少N_PUSHES	

（续）

事 件	条 件	动 作	下一状态
TR－Abort.ind	Reason＝DISCONNECT	断开会话	无效
	Reason＝SUSPEND	挂起会话	无效
	其他	S-PushAbort.ind（放弃原因） 减少N_PUSHES	无效

(4) 服务器会话状态表

表14-43～表14-50分别标识了会话状态和服务器使用事务处理服务时发生的事件处理。

表14-43 服务器会话无效

事 件	条 件	动 作	下一状态
TR－Invoke.ind	级别2， Connect	TR－Invoke.res N_Methods = 0 S－Connect.ind	连接

表14-44 服务器会话连接

事 件	条 件	动 作	下一状态
S－Connect.res		断开同这个地址四重组有关的其它会话 分配一个Session_ID给这个会话 TR－Result.req（ConnectReply） 释放所有处于保持状态的方法事务处理	连接_2
S-Disconnect.req	Reason＝~~M~~oved Permanently or Moved Temporarily	TR－Result.req（Redirect） 放弃（原因是DISCONNECT）所有方法事务处理 S-Disconnect.ind（原因是USERREQ）	结束
	其他	TR-Result.req （Reply（status= reason）） 放弃（原因是DISCONNECT）所有方法事务处理 S-Disconnect.ind（原因是USERREQ）	结束
Disconnect		TR-Abort.req（原因是DISCONNECT）连接事务处理 放弃（原因是DISCONNECT）所有方法事务处理 S-Disconnect.ind（原因是DISCONNECT）	无效
Suspend		TR-Abort.req（DISCONNECT） 连接事务处理 放弃(原因是DISCONNECT)所有方法事务处理 S-Disconnect.ind（SUSPEND）	无效
TR－Invoke.ind	级别2，方法 级别2，恢复	开始新的方法事务处理（参见方法状态表） TR－Abort.req（原因是DISCONNECT）放弃这个TR－Invoke	
TR－Abort.ind		放弃（原因是DISCONNECT）所有方法事务处理 S－Disconnect.ind（放弃原因）	无效

表14-45 服务器会话结束

事 件	条 件	动 作	下一状态
Disconnect		TR－Abort.req（原因是DISCONNECT） 继续传输事务处理	无效
Suspend		TR－Abort.req（原因是SUSPEND） 继续传输事务处理	无效
TR－Result.cnf		忽略	无效
TR－Abort.ind		忽略	无效

表14-46 服务器会话连接_2

事 件	条 件	动 作	下一状态
S－Disconnect.req		TR-Abort.req（原因是DISCONNECT） 这个连接事务处理 放弃（原因是DISCONNECT）所有的 方法和推事务处理 TR-Invoke.req（级别0，原因是 Disconnect） S-Disconnect.ind（原因是USERREQ）	无效
Disconnect		TR-Abort.req（原因是DISCONNECT）这 个连接事务处理 放弃（原因是DISCONNECT）所有的 方法和推事务处理 S-Disconnect.ind（原因是DISCONNECT）	无效
S-MethodInvoke.res		参考方法状态表	
S-MethodResult.req		参考方法状态表	
S－Push.req		TR－Invoke.req（级别0， Push）	
S－ConfirmedPush.req		开始新的推事务处理（参考推状态表）	
Suspend	会话恢复工具无效	TR－Abort.req（原因是DISCONNECT） 放弃这个连接事务处理 放弃（原因是DISCONNECT）所有方 法和推事务处理 S-Disconnect.ind（原因是SUSPEND）	无效
	会话恢复工具有效	TR-Abort.req（原因是SUSPEND） 放弃这个连接方法事务处理 放弃（原因是SUSPEND）所有方法和 推事务处理 S-Suspend.ind（原因是SUSPEND）	已被挂起
TR-Invoke.ind	级别2，方法	开始新的方法事务处理（见方法状态表） 释放这个新的方法事务处理	
	级别2，恢复，会话恢复工具无效	TR－Abort.req（原因是DISCONNECT） 放弃TR-Invoke TR－Invoke.res	恢复
	级别2，恢复，会话恢复工具有效	TR－Abort.req（原因是RESUME）放弃 这个连接事务处理 放弃（原因是RESUME）所有方法和推 事务处理 S-Suspend.ind（原因是RESUME） S-Resume.ind	

（续）

事 件	条 件	动 作	下一状态
	级别0，断开	TR-Abort.req（原因是DISCONNECT）	无效
	级别0，挂起	连接事务处理	已被挂起
	会话恢复工具有效	放弃（原因是DISCONNECT）所有方法和推事务处理	
		S-Disconnect.ind（原因是DISCONNECT）	
TR-Invoke.cnf	推事务处理	参考推状态表	
TR-Result.cnf	连接事务处理	参考方法状态表	接通
	方法事务处理		
TR－Abort.ind	连接事务处理	放弃（原因是DISCONNECT）所有方法和推事务处理	无效
		S－Disconnect.ind（放弃原因）	
	推事务处理	参考推状态表	
	方法事务处理	参考方法状态表	

表14-47 服务器会话连接

事 件	条 件	动 作	下一状态
S-Disconnect.req		放弃（原因是DISCONNECT）所有方法和推事务处理	无效
		TR-Invoke.req（级别0，Disconnect）	
		S-Disconnect.ind（原因是USERREQ）	
Disconnect		放弃（原因是DISCONNECT）所有方法和推事务处理	无效
		S-Disconnect.ind（原因是DISCONNECT）	
S-MethodInvoke.res		参考方法状态表	
S-MethodResult.req		参考方法状态表	
S-Push.req		TR-Invoke.req（级别0，Push）	
S-ConfirmedPush.req		开始新的推事务处理（参考推状态表）	
Suspend	会话恢复工具无效	放弃（原因是SUSPEND）所有方法和推事务处理	无效
	会话恢复工具有效	S-Disconnect.ind（原因是SUSPEND）	
		放弃（原因是SUSPEND）所有方法和推事务处理	已被挂起
		S-Suspend.ind（原因是SUSPEND）	
TR-Invoke.ind	级别2，方法	开始新的方法事务处理（参考方法状态表）释放新的方法事务处理	
	级别2，恢复会话恢复工具无效	TR-Abort.req（原因是DISCONNECT）放弃TR-Invoke	恢复

（续）

事 件	条 件	动 作	下一状态
	级别2，恢复，会话恢复工具有效	TR-Invoke.res 放弃（原因是RESUME）所有方法和推事务处理 S-Suspend.ind（原因是RESUME） S-Resume.ind	恢复
	级别0，断开	放弃（原因是DISCONNECT）所有方法和推事务处理 S-Disconnect.ind（原因是DISCONNECT）	无效
	级别0，挂起，会话恢复工具有效	放弃（原因是SUSPEND）所有方法和推事务处理 S-Suspend.ind（原因是SUSPEND）	正被挂起
TR-Invoke.cnf	推事务处理	参考推状态表	
TR-Result.cnf	方法事务处理	参考方法状态表	
TR－Abort.ind	推事务处理	参考推状态表	
	方法事务处理	参考方法状态表	

表14-48 服务器会话挂起

事 件	条 件	动 作	下一状态
S-Disconnect.req		S-Disconnect.ind（原因是USERREQ）	无效
Disconnect		S-Disconnect.ind（原因是DISCONNECT）	无效
TR-Invoke.ind	级别2，方法	TR-Abort.req（原因是SUSPEND） 放弃TR－Invoke	
	级别2，恢复	TR-Invoke.res S-Resume.ind	恢复
	级别0，断开	S-Disconnect.ind（原因是DISCONNECT）	无效

表14-49 服务器会话恢复_1

事 件	条 件	动 作	下一状态
S-Disconnect.req		TR-Abort.req（原因是DISCONNECT）放弃这个恢复事务处理 放弃（原因是DISCONNECT）所有方法事务处理 TR-Invoke.req（级别0，Disconnect） S-Disconnect.ind（原因是USERREQ）	无效
Disconnect		TR-Abort.req（原因是DISCONNECT）放弃这个恢复事务处理 放弃（原因是DISCONNECT）所有方法事务处理 S-Disconnect.ind（原因是DISCONNECT）	无效
S-Resume.res		断开同这个地址四重组有关的会话 捆绑会话至一个新的地址四重组	恢复_2

（续）

事 件	条 件	动 作	下一状态
		TR-Result.req（原因是Reply） 释放所有处于保持状态的方法事务处理	
Suspend		TR-Abort.req（原因是SUSPEND） 放弃这个恢复事务处理 放弃（原因是SUSPEND）所有方法事务处理 S-Suspend.ind（原因是SUSPEND）	已被挂起
TR－Invoke.ind	级别2，方法	开始新的方法事务处理（参考方法状态表）	
	级别2，恢复	TR-Invoke.res TR-Abort.req（原因是RESUME）放弃这个旧的恢复事务处理 放弃（原因是RESUME）所有方法事务处理 S-Suspend.ind(原因是RESUME) S-Resume.ind	
	级别0，挂起	TR-Abort.req（原因是SUSPEND） 放弃这个恢复事务处理 放弃（原因是SUSPEND）所有方法事务处理 S-Suspend.ind（原因是SUSPEND）	已被挂起
	级别0，断开	TR-Abort.req（原因是DISCONNECT） 放弃这个恢复事务处理 放弃（原因是DISCONNECT）所有方法事务处理 S-Disconnect.ind（原因是DISCONNECT）	无效
TR-Abort.ind	恢复事务处理	放弃（原因是SUSPEND）所有方法事务处理 S-Suspend.ind（放弃原因）	已被挂起

表14-50 服务器会话恢复_2

事 件	条 件	动 作	下一状态
S-Disconnect.req		TR-Abort.req（原因是DISCONNECT） 放弃这个恢复事务处理 放弃（原因是DISCONNECT）所有方法和推事务处理 TR-Invoke.req（级别0，Disconnect） S-Disconnect.ind（原因是USERREQ）	无效
Disconnect		TR-Abort.req（原因是DISCONNECT） 放弃这个恢复事务处理 放弃（原因是DISCONNECT）所有方法和推事务处理 S-Disconnect.ind(原因是DISCONNECT)	无效
S-MethodInvoke.res		参考方法状态表	
S-MethodResult.req		参考方法状态表	

（续）

事 件	条 件	动 作	下一状态
S-Push.req		TR-Invoke.req（级别0，Push）	
S-ConfirmedPush.req		开始新的推事务处理（参考推状态表）	
Suspend		TR-Abort.req（原因是SUSPEND）	已被挂起
		放弃这个恢复事务处理	
		放弃（原因是SUSPEND）所有方法和推事务处理	
		S-Suspend.ind（原因是SUSPEND）	
TR-Invoke.ind	级别2，方法	开始新的方法事务处理（参考方法状态表）	恢复
		释放这个新的方法事务处理	
	级别2，恢复	TR-Invoke.res	
		TR-Abort.req（原因是RESUME）	
		放弃这个旧的恢复事务处理	
		放弃（原因是RESUME）所有的方法和推事务处理	
		S-Suspend.ind（原因是RESUME）	
		S-Resume.ind	
	级别0，挂起	放弃（原因是SUSPEND）所有方法和推事务处理	已被挂起
		S-Suspend.ind（原因是SUSPEND）	
	级别0，断开	TR-Abort.req（原因是DISCONNECT）	无效
		放弃恢复事务处理	
		放弃（原因是DISCONNECT）所有方法和推事务处理	
		S-Disconnect.ind（原因是DISCONNECT）	
TR-Invoke.cnf	推事务处理	参考推状态表	
TR-Result.cnf	恢复事务处理	参考方法状态表	接通
	方法事务处理		
TR-Abort.ind	恢复事务处理	放弃（原因是SUSPEND）所有的方法和推事务处理	已被挂起
		S-Suspend.ind（放弃原因）	
	推事务处理	参考推状态表	
	方法事务处理	参考方法状态表	

(5) **服务器方法状态表**

表14-51～表14-55分别标识了服务器使用事务处理服务时发生的方法状态和事件处理。

表14-51 服务器方法无效

事 件	条 件	动 作	下一状态
TR-Invoke.ind	级别2，方法PDU，N_Methods ═MOM	TR-Abort.req（MOREXCEEDED）	无效
	级别2，方法PDU	Increment N_Methods	保持

表14-52 服务器方法保持

事 件	条 件	动 作	下一状态
Release		S-MethodInvoke.ind	请求
Abort		Decrement N_Methods	无效
		TR-Abort.req（放弃原因）放弃该方法	
TR-Abort.ind	Reason=DISCONNEC	断开会话	无效
	Reason=SUSPEND	挂起会话	
	其他	减少N_Methods	

表14-53 服务器方法请求

事 件	条 件	动 作	下一状态
S-MethodInvoke.res		TR-Invoke.res	处理
S-MethodAbort.req		减少N_Methods	无效
		TR-Abort.req（原因是PEERREQ）放弃该方法	
		S-MethodAbort.ind（原因是USERREQ）	
Abort		减少N_Methods	无效
		TR-Abort.req（放弃原因）放弃该方法	
		S-MethodAbort.ind（放弃原因）	
TR-Abort.ind	Reason=DISCONNECT	断开会话	无效
	Reason=SUSPEND	挂起会话	
	其他	减少N_Methods	
		S-MethodAbort.ind（放弃原因）	

表14-54 服务器方法处理

事 件	条 件	动 作	下一状态
S-MethodResult.req		TR-Result.req	应答
		减少N_Methods	无效
S-MethodAbort.req		TR-Abort.req（原因是PEERREQ）放弃该方法	
		S-MethodAbort.ind（原因是USERREQ）	
		减少N_Methods	
Abort		TR-Abort.req（放弃原因）放弃该方法	无效
		S-MethodAbort.ind（放弃原因）	
	Reason=DISCONNECT	断开会话	
TR-Abort.ind	Reason=SUSPEND	挂起会话	无效
	其他	减少N_Methods	
		S-MethodAbort.ind（放弃原因）	

表14-55 服务器方法应答

事 件	条 件	动 作	下一状态
S-MethodAbort.req		减少N_Methods	无效
		TR-Abort.req（原因是PEERREQ）放弃该方法	
		S-MethodAbort.ind（原因是USERREQ）	
Abort		减少N_Methods	无效
		TR-Abort.req（放弃原因）放弃该方法	
		S-MethodAbort.ind（放弃原因）	

（续）

事 件	条 件	动 作	下一状态
TR-Result.cnf		减少N_Methods S-MethodResult.cnf （确认报头=退出信息） 注意： 可以选择是否支持确认报头	无效
TR-Abort.ind	Reason=DISCONNECT Reason=SUSPEND 其他	断开会话 挂起会话 减少N_Methods S-MethodAbort.ind（放弃原因）	无效

(6) 服务器推状态表

表14-56和表14-57分别标识了服务器使用事务处理服务时发生的推状态和事件处理。

表14-56 服务器推状态-无效

事 件	条 件	动 作	下一状态
S－ConfirmedPush.req		TR-Invoke.req（级别1，推）	推

表14-57 服务器推状态-推

事 件	条 件	动 作	下一状态
Abort		TR-Abort.req（放弃原因）放弃这个推事务处理 S-PushAbort.ind（放弃原因）	无效
TR-Invoke.cnf		S-ConfirmedPush.cnf （确认报头=退出信息） 注意：可以选择是否支持确认报头	无效
TR-Abort.ind	Reason=DISCONNECT Reason=SUSPEND 其他	断开会话 挂起会话 S-PushAbort.ind（放弃原因）	无效

14.4.2 无连接模式WSP/B

本节适用于会话服务提供者直接使用传输SAP协议的情况，并且同样适用于安全SAP协议的使用。在无连接模式传输原语（WAPWDP）与安全原语之间存在一个一一映射。例如，T－Dunitdata.request直接映射到SEC－UnitData.request。考虑到这种两义性，在原语名字前删除层前缀（“T－D”或“SEC－”）。

无连接模式WSP/B协议并不需要状态机。如表14-58所示，无连接模式WSP/B业务接口的每一原语同低层的Unitdata原语一起直接发送WSP/B PDU。

表14-58 无连接模式WSP/B

事 件	条 件	动 作
S-Unit-MethodInvoke.req		Unitdata.req（原因是Method） 注意：“方法”表示的是Get PDU或者是Post PDU,它使用分配给这个方法的PDU类型

（续）

事 件	条 件	动 作
S-Unit-MethodResult.req		Unitdata.req（Reply）
S－Unit－Push.req		Unitdata.req（Push）
T－DError.ind		Ignore
Unitdata.ind	方法PDU 注意：“方法”表示的是Get PDU或者是Post PDU，它使用分配给这个方法的PDU类型	S-Unit-MethodInvoke.ind
	应答PDU	S-Unit-MethodResult.ind
	推PDU	S-Unit-MethodPush.ind

协议的参数由服务使用者双方共同约定，例如 MRU和持续有效的会话报头，并不需要特殊的机制来定义它们，但是，服务器的知名端口需要指出相应的参数设置。

14.5 WSP/B数据单元结构和编码

本节讲述了在客户端和服务器之间交换 WSP/B数据单元所用的数据结构。

14.5.1 数据格式

使用如下数据类型定义数据格式（如表 14-59所示）。

表14-59 数据类型格式定义

数 据 类 型	定 义
bit	一位数据
octet	不透明8位数据
uint8	8位无符号整数
uint16	16位无符号整数
uint32	32位无符号整数
uintvar	可变长无符号整数（参考下一节）

1. 原语数据类型

多字节整数值的网络传输次序是高字节优先。换句话说，最高有效字节在网络中优先传输，随后是较低有效字节。

一个字节中各二进制位的传输次序也是高字节优先。换句话说，最先描述的数据位放置在最高有效位上并在网络中优先传输，随后是较低有效位。

2. 可变长无符号整数

数据单元格式中一些字段是可变长的，特别是有一个字段规定了可变长字段的大小。为使数据单元格式尽可能小，使用可变长无符号整数编码来规定长度。无符号整数越大，相应的编码的大小越大。

每一可变长无符号整数的字节由 1个连续位和 7个负荷位（payload）组成。如图14-27所示。

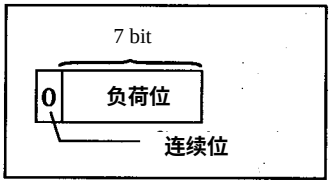

图14-27 可变长整数八位组

在编码一较大无符号整数时，按其7位划分并分别放入各字节的装载（payload）中。最高有效位放在第一个字节中，最低有效位放于最后一个字节中。除最后一个字节外，所有字节的连续位置1，而最后一个字节的连续位置0。

例如，如图14-28所示，数值0x87A5（1000 0111 1010 0101）的编码放于三个字节中。无符号整数应以最可能小的编码方法来编码。换句话说，编码值的第一个字节的值不应为 0x80。

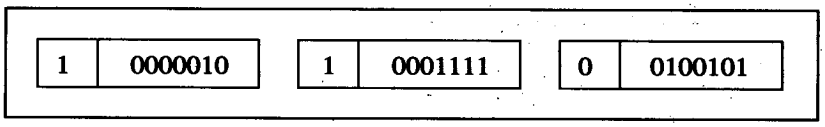

图14-28 长整数长度

在数据单元格式描述中，数据类型 unitvar用来指示变量长度的整数字段。一个 uintvar的最大位数为32位，它的编码不超过5个字节。

14.5.2 协议数据单元结构

WSP/B产生的WTP SDU包含了一个单独的WSP/B协议数据单元，每一PDU在协议中具有某种具体的功能并包含某种具体的类型信息。

1. PDU通用字段

本节讲述了所有或许多PDU中的通用字段（如图14-29所示）。

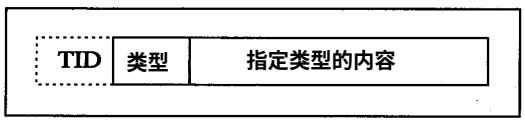

图14-29 PDU结构

每一PDU都以一个有条件的事务处理标识符和类型标识符开头（参考表 14-60）。

表14-60 PDU报头字段

名 字	类 型	信 源
TID	uint8	S－Unit－MethodInvoke.req ∶∶ Transaction Id 或 S－Unit－MethodResult.req ∶∶ Transaction Id 或 S－Unit－Push.req ∶∶ Push Id
Type	uint8	PDU类型

TID字段用来在无连接模式会话服务中连接请求和应答，使用 TID是有条件的。在无连接模式的PDU/B中它必须存在，但是在连接方法的 PDU中它却不能存在。在非连接 WSP/B中，TID作为会话原语的Transaction ID或PUSH ID参数传送给会话用户或从会话用户传送出去。

Type字段规定了PDU的类型和功能。 PDU的其他部分被看作为内容，包含某种特定类型信息。

下面各节讲述了各PDU类型的内容格式。为简短起见，以下各节中在描述 PDU时，省略了该PDU的报头。

2. 会话管理工具

(1) **连接**（Connect）

初始化会话的创建要发出Connect PDU（**参考表**14-61）。

表14-61 Connect**字段**

名 字	类 型	源 于
Version	uint8	WSP/B**协议的版本**
CapabilitiesLen	Uintvar	Capabilities**字段长度**
HeadersLen	Uintvar	Headers**字段长度**
Capabilities	CapabilitiesLen **八位组**（octets）	S-Connect.req：：Requested Capabilities
Headers	HeadersLen **八位组**	S-Connect.req：：Client Headers

Verson**字段标识了** WSP/B**协议的版本，它被用来确定该** PDU**以及所有后续** PDU**的格式。版本号如下编码：高** 4**位存储了版本的主要号码，低** 4**位存储了版本的次要号码。本规范所用的版本号是**1.0**，也就是**0x10**。**

CapabilitiesLen**字段规定了性能字段的长度。**

HendersLen**字段规定了报头字段的长度。**

Capabilities**字段包含发送方请求的已编码性能设置。每一性能所具有的特定性能参数都同该字段有关。关于该字段其他的编码信息，参考** 14.5.3**节“ 性能编码 ”。**

Headers**字段包含客户端发给服务器的报头，在整个会话中都使用该报头。**

(2) **连接应答**（ConnectReply）

发送ConnectReply PDU**是响应**Connect PDU**的**（**参考表**14-62）。

表14-62 ConnectReply**字段**

名 字	类 型	信 源
ServerSessionId	Uintvar	Session_ID variable
CapabilitiesLen	Uintvar	Capabilities**字段的长度**
HeadersLen	Uintvar	Headers**字段的长度**
Capabilities	CapabilitiesLen **八位组**	S-Connect.res：：Negotiated Capabilities
Headers	HeadersLen **八位组**	S-Connect.res：：Server Headers

ServerSessionId**包含了服务器会话标识符。在随后发出的用于会话管理的** PDU**中，它用以标识相应的会话。特别是，如果客户端在低层传输发生变化后试图恢复会话时，将使用该会话标识符。**

CapabilitiesLen**字段规定了性能字段的长度。**

HeadersLen**字段规定了报头字段的长度。**

Capabilties**字段或为**0**或为发送方接受的性能设置。关于性能的其他信息，参考** 14.5.3**节“ 性能编码 ”。**

Headers**字段包含的报头在整个会话中都适用。**

(3) **重定向**（Redirect）

当建立会话企图被拒绝时，应返回 Redirect PDU**以响应**Connect PDU**。在会话创建时，当服务器地址改变或需要执行加载平衡方法时，可使用** Redirect PDU**从服务器转移客户**（**如表**14-63）。

表14-63 Redirect字段

名 字	类 型	信 源
Flags	uint8	S-Disconnect.req ∷ Redirect Security和S－Disconnect.req ∷ Reason
Redirect Addresses	multiple 八位组	S-Disconnect.req ∷ Redirect Addresses

Flags字段标识了重定向的种类。未分配的标志必须由服务器置为0并且客户端忽略该标志。表14-64列出了标志的定义。

表14-64 未分配标志定义

标 志 位	描 述
0x80	永久重定向
0x40	重用安全会话

如果永久重定向标志被置值，客户端应保存Redirect Addressess，在以后所有同服务器建立会话时都使用被保存的Redirect Addressess。如果重用安全会话标志被置值，当客户端向重定向所至的服务器请求会话时，可使用当前的安全会话。

Redirect Addresses字段包含一个或多个新的服务器地址，随后的Connect PDU应被送往这些地址，而不是引起Redirect PDU发送的服务器地址，Redirect Addresses字段的长度由低层传输报告的SDU大小决定。每一重定向地址以以下格式编码（如表14-65所示）。

表14-65 AddressType

名 字	类 型	用途/目的
NetworkType Included	1 bit	指示包含NetworkType字段的标志
PortNumber Included	1 bit	指示包含PortNumber字段的标志
Address Len	6 bits	Address字段的长度
BearerType	uint8	所用承载网络的类型
PortNumber	uint16	使用的端口号
Address	AddressLen 八位组	所用的承载网络地址

BearerType Included字段和PortNumber Included字段分别标识了BearerType字段和PortNumber字段的内容。如果会话建立企图被重定向至的承载网络类型和信宿端口号分别和用于初始连接PDU的承载网络类型和信宿端口号相同，那么不应包含BearerType字段和PortNumber字段。

AddressLen字段包含地址字段的长度。

BearerType字段标识了所使用的承载网络的类型。承载类型码在[WAPWDP]中定义。

PortNumber字段包含了信宿端口号。

Address字段包含了所使用的承载地址。

BearerType字段同样指出了用于对该字段进行编码的承载（bearer－dependent）地址格式，编码应使用可应用的承载器规范中定义的本地地址传输格式。如果该格式使用的数据位数不是8的倍数，那么该地址应被编码成高字节优先的多字节整数，在最高有效字节中应包含必要的0填充位，所使用的承载地址格式在WAPWDP中同承载类型码一起定义。

(4) 断开（Disconnect）

发送断开PDU以结束会话（如表14-66所示）。

表14-66 Disconnect字段

名 字	类 型	信 源
ServerSessionId	Uintvar	Session_ID 变量

ServerSessionId包含了将要断开的会话的会话标识符

(5) 应答（Reply）

应答PDU由会话创建工具使用，在随后的14.5.2节“应答”中定义。

3. 方法引发工具

可用两种PDU在服务器引发一个方法：Get和Post，这视所要求的参数而定。

HTTP1.1（RFC2068）定义的方法被分配给一个具体的 PDU类型号，但是，方法 PDU类型号并不是HTTP1.1定义的，而是在能力协商时建立的。该方法使用Get PDU或使用Post PDU，将视该方法是否包含请求内容来定。使用 Get的方法的PDU类型号范围是 0x40-0x5F，使用Post的方法的PDU类型号范围是0x60-0x7F。

(1) Get

Get PDU适用于HTTP1.1GET、OPTIONS、HEAD、DELETE和TRACE方法，同样也适用于扩展方法，该扩展方法并不给服务器发送请求信息（如表 14-67所示）。

表14-67 Get Fields

名 字	类 型	信 源
URILen	uintvar	URI字段的长度
HeadersLen	uintvar	Headers字段的长度
URI	URILen 八位组	S-MethodInvoke.req ∶∶ Request URI或S-Unit-Method-Invoke.req ∶∶ Request URI
Headers	HeadersLenoctets	S-MethodInvoke.req ∶∶ Request Headers or S-Unit-MethodInvoke.req ∶∶ Request Headers

URLLen字段规定了URL字段的长度。

HeaderLen字段规定了Header字段的长度。

URL字段包含了URL。如果URI是正常存储的空结束字符串，那么实现在该字段中一定不包括该空串。

Header字段包含了同请求相关的报头。

(2) Post

Post PDU适用于HTTP1.1POST和PUT方法，同样适用于扩展方法。该扩展方法向服务器发送请求信息（如表14-68所示）。

表14-68 Post字段

名 字	类 型	信 源
UriLen	Uintvar	URI字段的长度
HeadersLen	Uintvar	ContentType和Headers字段的组合长度
Uri	UriLen 八位组	S-MethodInvoke.req ∶∶ Request URI或S-Unit-Method - Invoke.req ∶∶ Request URI

（续）

名 字	类 型	信 源
ContentType	multiple 八位组	S-MethodInvoke.req：：Request Headers或 S-Unit-MethodInvoke.req ：：Request Headers
Headers	（HeadersLen-length of ContentType）八位组	S-MethodInvoke.req ：：Request Headers或 S-Unit-MethodInvoke.req ：：Request Headers
Data	multiple 八位组	S-MethodInvoke.req ：：Request Body或S-Unit-Method - Invoke.req ：：Request Body

UriLen字段规定了URI字段的长度。

HeaderLen字段规定了ContentType和Headers字段的组合长度。

URI字段包含了URI。如果URI是正常存储的空结束字符串，那么实现在该字段中一定不包括该空串。

ContentType字段包含了数据的内容类型，它同下面 14.5.4节“ 内容类型字段 ”中规定的内容类型值编码相一致。

Header字段包含了同请求有关的报头。

Data字段包含了同请求有关的数据。数据字段的长度取决于低层传输提供或报告的 SDU大小，数据字段紧接在Header字段之后，并一直到SDU的末尾。

(3) 应答（Reply）

应答是一般响应PDU，用来从服务器返回对一个请求的响应信息。应答在 S－Connect原语中用于会话创建时指示错误（如表 14-69所示）。

表14-69 Reply字段

名 字	类 型	信 源
Status	uint8	S-MethodResult.req ：：Status或S-Disconnect.req ：：Reason或S-Unit－MethodResult.req ：：Status
HeadersLen	uintvar	ContentType和Headers字段的组合长度
ContentType	multiple 八位组	S-MethodResult.req ：：Response Headers或S-Disconnect.req ：：Error Headers或S-Unit-MethodResult.req ：：Response Headers
Headers	（HeadersLen-length of ContentType）八位组	S-MethodResult.req ：：Response Headers或S-Disconnect.req ：：Error Headers或S-Unit-MethodResult.req ：：Response Headers
Data	multiple 八位组	S-MethodResult.req ：：Response Body或S-Disconnect.req ：：Error Body或S-Unit-MethodResult.req ：：Response Body

Status字段包含了在试图理解和响应请求时的结果代码。 Status代码在HTTP1.1[RFC2068]中定义，并映射成单字节值，这些值在表 14-21已分配号栏中列出。

HeadersLen字段规定了ContentType和Headers字段的组合长度。

ContentType字段包含了数据的内容类型，它同在 14.5.4节“ 内容类型字段 ”中规定的内容类型值编码相一致。

Headers字段包含了应答报头。

数据字段包含了从服务器返回的数据，它的长度取决于低层传输提供或报告的 SDU的大小，其紧接在Header字段之后，并一直到SDU的末尾。

(4) 确认报头（Acknowledgment Headers）

确认报头并不是真正的PDU，它可能由 TR－Result原语的Exit Info参数携带。业务提供者使用它来携带确认报头可选的特征所需的数据（如表 14-70所示）。

表14-70 Acknowledgment Headers字段

名 字	类 型	信 源
Headers	multiple 八位组	S-MethodResult.res :: Acknowledgement Headers或 S-ConfirmedPush.res :: Acknowledgement Header

Header字段包含了以 14.5.4节“ 报头编码 ”中定义的方法编码的信息，该字段的大小由Transaction Exit数据的大小确定。

4. 推和经确认推工具

(1) 推和经确认推

Push和ConfirmedPush PDU从服务器向客户端传送未经请求的信息。这两种 PDU的格式是一样的，只是PDU的类型是不同的（如表14-71所示）。

表14-71 推和经确认推字段

名 字	类 型	信 源
HeadersLen	Uintvar	ContentType和Headers的组合长度
ContentType	multiple 八位组	S-Push.req :: Push Headers或S-ConfirmedPush.req :: Push Headers或S-Unit-Push.req :: Push Headers
Headers	（HeadersLen-length of ContentType）八位组	S-Push.req :: Push Headers或S-ConfirmedPush.req :: Push Header或S-Unit-Push.req :: Push Headers
Data	multiple 八位组	S-Push.req :: Push Body或S-ConfirmedPush.req :: Push Body或S-Unit-Push.req :: Push Body

HeadersLen字段规定了ContentType字段和Headers字段的组合长度。 ContentType字段包含了数据的内容类型，它同14.5.4节“ 内容类型字段 ”中规定的内容类型值编码相一致。

Headers字段包含了Push报头。

Data字段包含了从服务器推的数据。

Data字段的长度取决于低层传输提供或报告的 SDU的大小。 数据字段紧接在Header字段之后，并一直到SDU的末尾。

(2) 确认报头

如果业务提供者要以经确认推工具来实现确认报头可选特征，那么确认报头要携带相关数据。

5. 会话恢复工具

(1) 挂起（Suspend）

发送Suspend PDU用来挂起会话（如表14-72所示）。

表14-72 挂起字段

名 字	类 型	信 源
SessionId	Uintvar	Session_ID变量

SessionId字段包含了将要挂起的会话的会话标识符。

(2) 恢复（Resume）

如表14-73所示，在低层传输协议改变时，发送恢复 PDU用来恢复业已存在的会话。

表14-73 恢复字段

名 字	类 型	用途/目的
SessionId	Uintvar	Session_ID变量

SessionId字段包含了在会话最初创建时从服务器返回的会话标识符，服务器根据会话标识符寻找相应的会话。然后，绑定会话至事务处理业务实例，该实例由携带该 PDU的事务处理的通信地址四重组标识。

(3) 应答（Reply）

应答PDU为会话恢复工具所使用，并在14.5.2节“应答”中定义。

14.5.3 性能编码

性能使客户端和服务器能够在协议特征和扩展方法进行协商。一般性能格式定义：忽略不能理解的性能。

性能值的集合被编码成性能结构序列，该性能结构在下一节讲述。

如果发送方希望提供给接收方可以选择的某一性能的可选值集合，那么发送方发送性能的多个实例，每一个实例具有不同的参数，并把最优先选择的参数置于最前面。响应方不能对性能值编码和发送，除非知道倡议方承认该性能值，这一点或者由会话协议的版本号指出或者由早已在会话中发送了该性能的倡议方指出。

当能力协商倡议方给一个性能编码，该性能在下面的 14.5.3节“性能定义”中定义，并且该编码值等于目前有效的性能设置（默认值或协商值），那么性能结构可以省略。在这种情况下响应方视其有相同的性能结构，并作同样处理，就好象响应方已经接收到了明确的编码值。当响应方给一个性能编码，该性能在下面的 14.5.3节“性能定义”中定义，并且该编码值等于倡议方提出的性能设置，那么性能结构可以省略。在这种情况下倡议方视其有相同的性能结构，并做同样处理，就好象倡议方已经接收到了某个编码值。

1. 性能结构

描述性能格式的表类似于在 PDU定义中使用的表（如表14-74所示）：

表14-74 性能字段

名 字	类 型	用途/目的
Length	Uintvar	Identifier和Parameters字段的组合长度
Identifier	multiple 八位组	性能标识符
Parameters	(Length-length of Identifier) 八位组	关于性能的参数

Length字段规定了标识符和参数字段的组合长度。

标识符字段确定了性能。在这个协议版本中定义的性能标识符的值请参考表 14-94的分配的号码列。它的编码方法与报头字段名相同，使用字段名 BNF规则，这一点在14.5.4节“报头”中有详细说明。

Parameters字段（如果非空）包含了具体性能的参数。

如果在能力协商中接收到一未知标识符字段的性能，该值必须被忽略。响应方同样要以相同的性能作应答，但 Parameters字段为空，该空 Parameters字段表明该性能由于不确定所以未生效。结果是，必须为任何具体提供者的附加性能选择编码，而空 Parameters字段是非法的（例如性能具有整数值），或者它表明没有扩展功能被激活。

2. 性能定义

(1) 业务数据单元大小

有两种业务数据单元大小性能，一种是针对客户端的，另一种是针对服务器的：

- Client-SDU-Size
- Server-SDU-Size

这些性能具有相同的参数格式（如表 14-75所示）。

表14-75 SDU大小性能字段

名 字	类 型	用途/目的
MaxSize	Uintvar	能够取的最大值

MaxSize字段规定了客户端或服务器接收或发出的最大 SDU大小，这一点取决于性能的内容，MaxSize置为0表示SDU大小没有限制。当客户端发送 Client－SDU－Size性能时，它指出了客户端能接收到的SDU大小的最大值（即客户端 MRU）。

当服务器方发送Client-SDU-Size性能时，它指出了服务器能发送的 SDU大小的最大值。

当客户端发送Server-SDU-Size性能时，它指出了客户端能发送的 SDU大小的最大值。

当服务器发送 Server-SDU-Size性能时，它指出了服务器能接收到的 SDU大小的最大值（即服务器MRU）。

默认的SDU大小在14.5.3节“性能默认”中规定，默认SDU大小被看作具体实现的最小值，否则会话建立时发出的方法请求可能被放弃，这是由于直到会话建立时服务器才能指示出真正的MRU。

(2) 协议选项（Protocol Options）

协议选项性能用来激活扩展可选协议功能（如表 14-76所示）。

表14-76 协议选项性能字段

名 字	类 型	用途/目的
Flags	Multiple 八位组	可选标志

当客户端发送协议选项性能给服务器时，标志（ Flag）字段规定了客户端会接受的选项。当服务器返回协议选项性能给客户端时，标志字段规定了服务器会执行的选项。尽管标志字段可以多字节，但目前定义的标志位数为一个字节。所有未定义的位被置为0，接收方忽略这些0，包括所有随后的附加字节。将来定义更多的标志位，在该字段就可能要有更多的附加字节。

标志位置为 1表示激活相关的可选功能，标志位清 0表示撤销该功能。标志有如下定义（见表14-77）：

当客户端激活经确认推和 /或推工具时，这表明客户端能够并且也希望接受数据推。如果客户端能够接收数据推，但是服务器的业务提供者不能发送推，那么当（服务器）以协商的性能应答时，相应的推标志应清 0；如果服务器的业务用户不发送任何类型的数据推，那么相

应的推标志在应答中应清0。这就使得客户端能够释放另外用于接收数据推的资源。

表14-77 标志的定义

标 志 位	描 述
0x80	经确认推工具
0x40	推工具
0x20	会话恢复工具
0x10	确认报头

当客户端激活会话恢复工具时，这表明客户端希望挂起和恢复会话。如果服务器不能或不愿支持会话恢复工具，那么当服务器以协商的性能应答时，它应对会话恢复工具标志位清0。

当客户端确认报头标志位置值时，这表明客户端对是否希望发送确认报头进行确定，服务器在应答中利用确认报头标志来指出它是否能够处理该报头。如果服务器不能处理该报头，客户端就不应发送它们；如果客户端仍然发送它们，那么该报头会被忽略。

(3) 未完成请求的最大数目 （MOR）

有两种MOR性能，一种是用于方法的，一种是用于推的：

• Method-MOR

• Push-MOR

Method-MOR和Push-MOR性能分别表示了同时存在的未完成方法或推事务处理的数目（如表14-78所示）。

表14-78 MOR性能字段

名 字	类 型	用途/目的
MOR	uint8	未完成请求的最大数目

当客户端能够同时提交多个未完成方法请求，它指出了它能在 Method-MOR请求中同时发送请求的最大数目。服务器能够响应的数目低于客户端 Method-MOR请求的数目，也低于服务器能够同时处理的方法事务处理的数目。

类似地，当客户端能够同时处理多个未完成 push请求时，它指出了它能在Push-MOR功能中能够同时处理的请求的最大数目。服务器能够响应的数目低于客户端Push-MOR请求的数目，也低于服务器能够同时发出的推事务处理的最大数目。

(4) 扩展方法（Extended Methods）

扩展方法功能指明了在会话中用到的扩展方法集合并分配 PDU类型给相应的扩展方法（如表14-79所示）。

表14-79 扩展方法性能字段入口

名 字	类 型	用途/目的
PDU Type	uint8	用于方法的PDU类型
Method Name	Multiple 八位组	空结束方法名

当在Connect PDU中，从客户端到服务器为扩展方法性能发送一定的性能参数时，相应参

数以0或其他PDU类型给各方法名赋值。

分配表长度取决于在性能长度中规定的性能长度值，每一性能分配都包含了一个 PDU类型和一个方法名。针对使用 Get PDU格式的方法，客户端分配的 PDU类型范围是PDU类型0x50－0x5F，针对使用Post PDU格式的方法，客户端分配的PDU类型范围是0x70－0x7F。方法名是空结束字符串。

当在连接应答PDU中从服务器到客户端为扩展方法功能发送一定的性能参数时，相应参数或为0或为其他服务器能接收并接受的PDU类型代码（不包括方法名）。

(5) 报头代码页（Header Code Pages）

报头代码页性能详细说明了在会话中使用的报头代码页集合并分配页代码给相应的报头代码页（如表14-80所示）。

表14-80 报头代码页性能字段入口

名 字	类 型	用途/目的
Page Code	uint8	报头页代码
Page Name	Multiple 八位组	报头页名

当在连接PDU中，从客户端到服务器为报头代码页性能发送一定的性能参数时，相应参数以0或以其他报头页名给代码赋值，赋值表长度取决于在性能长度中规定的性能长度值。每一性能赋值都包含了一个页代码和一个页名，页名是空结束字符串。

当在连接应答PDU中，从服务器到客户端为报头代码页性能发送一定的性能参数时，相应参数或为0或为其他服务器能够并且会使用的页代码（不包括页名）。

当客户端发送该性能时，它指明了它需要使用的指定报头代码页。服务器的响应指明了在剩余的会话时间内实际上应使用的代码页。

一旦商定使用扩展报头代码页，属于该页的报头必须使用由代码页定义的二进制句法来编码并发送出去。如果服务器拒绝使用某一报头代码页，该（应用明确的）报头必须以本文的格式发送，除非其他代码页为该报头定义编码的句法。

如果服务器同意使用报头代码页，在变换代码页次序情况下需要区别报头代码页时，会在会话的剩余时间内使用客户端选定的页代码。

(6) 替换符（Aliases）

替换符性能详细说明了发送方可选择的地址列表（如表 14-81所示）。

表14-81 替换符性能字段

名 字	类 型	用途/目的
Addresses	Multiple 八位组	可选择的地址

地址字段编码格式同14.5.2节“重定向PDU”中重定向地址字段的格式一样。当恢复会话时可利用服务器发送的地址促使会话转换至另一承载网络，客户端发送的地址可简化无连接模式业务的使用。

3. 默认性能

除非对某一具体承载网络或一具体应用端口有了不同的规定，否则默认性能为表 14-82中的值：

表14-82 默认性能的取值

名 称	设 置	名 称	设 置
替 换 符	None	协议选项	0x00
客户端SDU大小	1400 字节	MOM	1
扩展方法	None	MOP	1
报头代码页	None	服务器SDU大小	1400 字节

14.5.4 报头编码

1. 概述

WSP/B报头包含在WSP/B PDU中或在多部分数据对象中。报头字段包含一般信息、请求信息、响应信息或者实体信息，每一报头字段由字段名组成，后跟一个字段值（如图 14-30）。

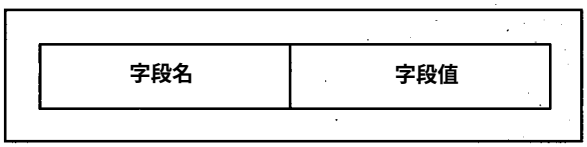

图14-30 由字段名和字段值组成的报头字段

WSP/B给报头字段编码定义了一个压缩格式，这与 HTTP1.1报头字段是兼容的。减少报头大小，可进行下面步骤：

1) 知名记号映射成二进制数值。

2) 数据值、整数值、质量因子（quality factor）和第四位值置成二进制编码。

3) 清除冗余信息。

编码利用了HTTP报头文本字符串第一个字节有代表性的范围（范围是32～126），除了文本字符串以8位的字符值（例如，national characters）初始化等极少的情况。范围0～31和127～255可用于二进制值、引用的字符和二进制数据长度指示器，这就使二进制数据和文本字符串有效混合成为可能，有利于对HTTP1.1报头的普通部分编码。

(1) 字段名

分配有整数编码值的字段名必须用整数值编码，没有分配整数值的字段名必须用文本编码。整数编码划分成报头代码页，使编码更加紧凑。

每一报头代码页编码一直到128个知名的字段名，所以整数编码值用一个字节来表示。最知名的报头名在默认报头代码页中定义，但是附加的编码值在代码页之间变换也能被置成有效。

在会话中使用的报头代码页由数字代码区别。报头代码 1是默认页，常常在一系列报头中首先被激活，转换到一个新的代码页通过在两个报头字段之间发送一个转换序列即可完成。这个过程适用于每一个WSP/B PDU的报头字段，同样也适用于多部分实体中每一实体的报头字段。

默认报头代码页定义了所有的 HTTP1.1字段名和WAP特殊报头字段。报头代码页号码分配如下：

- 默认报头代码页，包括HTTP1.1和WAP特殊报头。

• 2～15，为特殊WAP报头代码页保留。

• 16～27，为专用代码页保留。

• 128～255，为将来使用保留。

专用报头代码页由一个文本名（字符串）确定。然而，当使用能力协商约定在会话中使用的扩展报头代码页集合时，每一专用代码页同样分配得到一个在为其保留号码范围内的号码标识，该标识保持有效一直到会话结束，并且用来在变换序列中识别该页。

如果能力协商在使用报头代码页上有了约定，那么发送专用字段名时必须使用该页定义的知名单字节值，如果在使用报头代码页上没有约定，那么必须使用记号文本规则对该专用字段名进行编码。

例如，知名报头和专用报头序列可构造如下：

<WSP 报头 1>

.

.

<WSP报头 n>

<转移到专用代码页>

<专用报头 1>

.

.

<专用报头 m>

(2) 字段值

编码的字段值的句法由字段名定义。知名字段值必须使用根据报头句法定义的压缩二进制格式来编码，只有当没有其他可用的编码时才使用原文值（ textual value）。对WSP字段值这样编码可以使字段值是可定长的，即使并不知道某一字段值的具体格式，这同样使得可以跳过个别的报头信息而不用解释具体内容。 14.5.4节“ 报头句法 ”中定义的报头句法规定的所有字段值的第一个字节说明如表 14-83所示：

表14-83 报头句法规定的字段值的第一个字节说明

值	第一个字节的说明（INTERPRETATION）
0～30	该字节后跟其他数据字节的指示号码（0～30）
31	该字节后跟uintvar，指示了后继数据字节的号码
32～127	该值是文本字符串，以零字节（零字符）结尾
128～255	该值是一个7位编码值，该报头没有其他数据

直到应用时才定义如何对专用字段值进行编码，但是该编码必须附加到前表描述的一般格式中。

如果在先前的扩展报头代码页上客户端和服务器有了一个相互约定，那么在对这些代码页定义的专用字段值如何进行编码上也应有一个约定，在这种情况下，必须根据代码页定义的句法规则对可用的字段值进行编码。如果客户端和服务器在能力协商期间不能在报头代码页的使用上达成一致，那么必须使用专用值规则对专用字段值进行编码。

(3) 列表值编码

如果具有知名字段名的报头字段的句法使用 1#规则允许用逗号分隔的列表，该句法由

RFC2068定义，那么该报头必须转换成一个报头序列。该序列中每一报头都有一个源字段名，并且包含一个源列表的值，序列中报头的次序同源列表中相应值的次序是相同的。只有在转换之后，知名报头的编码规则才适用。

2. 报头句法

本节定义了WSP/B中所有HTTP1.1报头字段的句法和语义。本文档规定的机制在增强型BNF中描述，该增强型BNF同RFC2068使用的BNF类似。

符号<字节 N>用来表示一个单字节，其中值N是十进制数。符号<任意字节 M－N>用来表示一个单字节，其值范围是从M到N。

(1) 基本规则

在本规范中使用下列规则讲述了基本的分列结构。记号、文本和字节的规则同 RFC2068中的定义是一样的。

```
Text－string = [Quote] TEXT    End-of－string
```

如果文本中第一个字符在128～255范围内，那么在该文本之前有一个引用字符。否则，没有该字符。引用字符不属于文本内容部分。

```
Token-text = Token    End-of-string
Quoted-string = <Octet 34>    TEXT    End-of-string
```

文本对RFC2068 Quoted-string编码，引号标记除外。

Short-integer = OCTET

在0～127范围内的整数编码是一个字节，最高有效位置1（1xxx xxxx），其他有效位为编码值。

```
Long-integer = Short-length Multi-octet-integer
```

Short-length指示多字节整数的长度。

```
Multi-octet-integer = 1*30 OCTET
```

多字节内容应是一未分配的整数值，最高有效位首先被编码，依次一直到最低有效位。

```
Uintvar －integer = 1*5 OCTET
```

该编码与0节中为uintvar定义的编码相同。

```
Constrained-encoding = Token-text | Short-integer
```

如果记号值没有知名的二进制编码，或知名编码的分配数目可同短整数匹配，那么可使用该编码。

```
Quote = <Octet 127>
End-of-string = <Octet 0>
```

(2) 长度

使用下列规则对长度指示器进行编码：

```
Value-length = Short-length |(Length-quote Length)
```

Value length用来指示后面值的长度。

```
Short-length = <Any octet 0-30>
Length-quote = <Octet 31>
Length = Uintvar-integer
```

(3) 参数值

使用下列规则对参数值进行编码：

```
No-value = <Octet 0>
```

指示该参数实际上没有值，例如：“；foo=xxx；bar；baz=xyzzy”中的参数”bar”。

```
Text-value = No-value | Token-text | Quoted-string
Integer-Value = Short-integer | Long-integer
Date-value = Long-integer
```

日期应以1970-01-01，00：00：00 GMT开始的秒的方法进行编码。

```
Delta-seconds-value = Integer-value
Q-value = 1*2   OCTET
```

该编码同Uintvar－integer中的一样，但是其大小有限。当质量因子0和有一个或两个小数位的质量因子进行编码时，它们应乘以100并加1递增，所以它们的编码值仅有一个字节，范围是1～100。例如：0.1的编码为11（0x0B），0.99的编码为100（0x64）。有三个小数位的质量因子应乘以1000然后加上100，结果的编码应是一个或两个字节uintvar。例如，0.333的编码为0x83 0x31。质量因子1是默认值并且不会发送该值。

```
Version-value = Short-integer | Text-string
```

Short-integer值最高有效位的头三位用来对主版本号编码，其范围是1～7，其余四位有效位则包含了次版本号，范围是0～14。如果只有一个主版本号，则有效位最低四位以值15代替。如果将要被编码的版本遵循这些约定，则应使用Short-integer，否则，应使用Text-string。

```
Uri-value = Text-string
```

URI值应经由RFC2068编码，但是业务用户可使用不同的格式。

(4) 参数

使用如下规则对参数进行编码：

```
Parameter = Typed-parameter | Untyped-parameter
Typed －parameter = Well-known －parameter  －token Typed－value
```

知名参数（well－known parameter）包含了值的实际期望类型。

```
Well －known －parameter  －token = Integer－value
```

参数使用的编码值在分配号码一节中规定。

```
Typed －value = Compact－value | Text－value
```

除期望类型之外，再也没有其他值。如果该值不能用期望类型编码，那么它就应作为文本编码。

```
Compact －value = Integer－value |
          Date －value |Delta－seconds －value | Q－value | Version－value |
          Uri －value
Untyped －parameter = Token－text Untyped－value
```

值的类型未知，但是如果可能的话，可以作为整数进行编码。

```
Untyped －value = Integer－value | Text－value
```

(5) 鉴权

以下通用规则用于鉴定和授权。

```
Credentials =  ( Basic Basic－cookie )  |  ( Authentication  －scheme *Auth
```

```
param )
    Basic = <Octet 128>
    Basic -cookie = User-id Password
    User -id = Text-string
    Password = Text-string
```

注意：用户ID和口令不能以64为基数进行编码

```
    Authentication  -scheme = Token-text
    Auth -param = Parameter
    Challenge = ( Basic Realm-value )  |  ( Authentication  -scheme Realm-value
    *Auth -param )
    Realm -value = Text-string
```

在相应的RFC2068 Quoted-string**中编码应该不包括引号符“ "”**

(6) **报头**

使用以下规则对报头进行编码：

```
    Header = Message-header | Shift-sequence
    Shift -sequence = ( Shift -delimiter Page-identity )  | Short-cut -shift -
    delimiter
    Shift -delimiter = <Octet 127>
    Page -identity = <Any octet 1-255>
    Short -cut -shift -delimiter = <Any octet -131>

    Message -header = Well-known -header | Application-header

    Well -known -header = Well-known -field -name Wap-value

    Application  -header = Token-text   Application-specific  -value

    Field -name = Token-text | Well-known -field -name
    Well -known -field -name = Short-integer
    Application  -specific  -value = Text-string
    Wap -value =
              Accept -value |
              Accept -charset -value |
              Accept -encoding -value |
              Accept -language -value |
              Accept -ranges -value |
              Age -value |
              Allow -value |
              Authorization  -value |
              Cache -control -value |
              Connection  -value |
              Content -base -value |
              Content -encoding -value |
              Content -language -value |
              Content -length -value |
              Content -location -value |
              Content -MD5 -value |
```

```
Content -range -value |
Content -type -value |
Date |
Etag -value |
Expires -value |
From -value |
Host -value |
If -modified -since -value |
If -match -value |
If -none -match -value |
If -range -value |
If -unmodified -since -value |
Location -value |
Last -modified |
Max -forwards -value |
Pragma -value |
Proxy -authenticate -value |
Proxy -authorization -value |
Public -value |
Range -value |
Referer -value |
Retry -after -value |
Server -value |
Transfer -encoding -value |
Upgrade -value |
User -agent -value |
Vary -value |
Via -value |
Warning |
WWW -authenticate -value |
Content -disposition -value
```

(7) **接受字段**

使用以下规则对接受字段进行编码：

```
Accept -value = Constrained-media | Accept-general -form
Accept -general -form = Value-length Media-range [Accept-parameters]
Media -range = (Well -known -media | Token-text) *(Parameter)
Accept -parameters = Q-token Q-value *(Accept -extension)
Accept -extension = Parameter
Constrained -media = Constrained-encoding
Well -known -media = Integer-value
```

两者都使用分配号码中内容类型分配表的值进行编码。

```
Q-token = <Octet 128>
```

(8) **接受字符字段**

使用以下规则对接受字符字段进行编码：

```
Accept -charset -value = Constrained-charset | Accept-charset -general -form
Accept -charset -general -form = Value-length (Well -known -charset | Token-
```

```
text ) [Q－value]
    Constrained －well－known －charset = Constrained-encoding
    Well－known －charset = Integer-value
```

两者都使用分配号码中字符集合分配表的值进行编码。

(9) **接受编码字段**

使用以下规则对接受编码值进行编码：

```
Accept －encoding －value = Content-encoding －value
```

(10) **接受语言字段**

使用以下规则对接受语言值进行编码：

```
Accept －language －value = Constrained-language | Accept-language －general －
form
Accept －language －general －form = Value-length (Well－known －language | Text
－string ) [Q－value]
Constrained －language = Any-language | Constrained-encoding
Well－known －language = Any-language | Integer-value
```

两者都使用分配号码中字符集合分配表的值进行编码。

```
Any－language = <Octet 128>
```

等同于专用RFC2068**语言范围“*”**

(11) **接受范围字段**

使用以下规则对接受范围值进行编码：

```
Accept －ranges －value = (None | Bytes | Token-text )
None = <Octet 128>
Bytes = <Octet 129>
```

(12) **年限字段**

使用以下规则对年限值进行编码：

```
Age －value = Delta-seconds －value
```

(13) **许可字段**

使用以下规则对允许值进行编码：

```
Allow －value = Well-known －method
Well－known －method = Short-integer
```

范围0x40～0x7F**内的任何知名方法或扩展方法。**

(14) **鉴权字段**

使用以下规则对鉴权值进行编码：

```
Authorization －value = Value-length Credentials
```

(15) Cache**控制字段**

使用以下规则对Cache**控制值进行编码：**

```
Cache －control －value =     No-cache |
                              No－store |
                              Max－stale |
                              Only－if－cached |
                              Private |
```

```
                                Public |
                                No-transform |
                                Must-revalidate |
                                Proxy-revalidate |
                                Cache-extension |
                                Value-length Cache-directive
    Cache-directive =No-cache 1*(Field-name) |
                     Max-age   Delta-second-value |
                     Max-stale   Delta-second-value |
                     Min-fresh   Delta-second-value |
                     Private 1*(Field-name) |
                     Cache-extension Parameter
    No-cache = <Octet 128>
    No-store = <Octet 129>
    Max-age = <Octet 130>
    Max-stale = <Octet 131>
    Min-fresh = <Octet 132>
    Only-if-cached = <Octet 133>
    Public = <Octet 134>
    Private = <Octet 135>
    No-transform = <Octet 136>
    Must-revalidate = <Octet 137>
    Proxy-revalidate = <Octet 138>
    Cache-extension = Token-text
```

(16) **连接字段**

使用以下规则对连接值进行编码：

```
Connection -value = (Close | Token-text)
Close = <Octet 128>
```

(17) **基本内容字段**

使用以下规则对基本内容值进行编码：

```
Content -base -value = Uri-value
```

(18) **内容编码字段**

使用以下规则对内容编码值进行编码：

```
Content -encoding -value = (Gzip | Compress | Deflate | Token-text)
Gzip = <Octet 128>
Compress = <Octet 129>
Deflate = <Octet 130>
```

(19) **内容语言字段**

使用以下规则对内容语言值进行编码：

```
Content -language -value = (Well-known-language | Token-text)
```

(20) **内容长度字段**

使用以下规则对内容长度值进行编码。内容长度报头中的信息一般是冗余的，可能不发送。内容长度在PDU中有效或在传输层提供PDU大小时计算该长度。

如果PDU根本没有包含实体报文（对应于报头报文），那么内容长度应在报头字段中编码

以便客户端能够获悉实体大小。

```
Content －length －value = Integer－value
```

(21) 内容位置字段

使用以下规则对内容位置值进行编码：

```
Content －location －value = Uri－value
```

(22) 内容MD5字段

使用以下规则对内容MD5值进行编码：

```
Content －MD5－value = Value－length  Digest
```

128位MD5摘要同[RFC1864]，注意该摘要不能以64为基数进行编码。

```
Digest = 16*16 OCTET
```

(23) 内容范围字段

使用以下规则对内容范围值进行编码。在 HTTP1.1报头中用到的Last－byte－pos是冗余的。内容范围在PDU中有效或在传输层提供PDU大小时计算该范围。 Last－byte－pos值可通过First－byte－pos值与内容范围值之和得到。

```
Content －range = Value－length  First－byte －pos  Entity－length
First －byte －pos = Uintvar－integer
Entity －length = Uintvar－integer
```

(24) 内容类型字段

使用以下规则对内容类型值进行编码。只有当知名媒体在 0～27或文本字符串的范围内时才可使用内容类型值的短形式，在其他情况下，必须使用一般形式。

```
Content －type －value = Constrained－media | Content－general －form
Content －general －form = Value－length  Media－type
Media －type = (Well －known －media | Token－text) *(Parameter)
```

(25) 数据字段

使用以下规则对数据值进行编码：

```
Date = Date－value
```

(26) 实体标签字段

使用以下规则对实体标志值进行编码：

```
Etag －value = Text－string
```

该值编码应如[RFC2068]。

(27) 终止字段

使用以下规则对终止值进行编码：

```
Expires －value = Date－value
```

(28) From字段

使用以下规则对From值进行编码：

```
From －value = Text－string
```

该值应如[RFC822]一样编码成一个e－mail地址。

(29) 主机字段

使用以下规则对主机值进行编码：

```
Host ―value = Text―string
```

该值编码应如[RFC2068]。

(30) If Modified Since字段

使用以下规则对If Modified Since值进行编码：

```
If ―modified ―since ―value = Date―value
```

(31) If Match字段

使用以下规则对if match值进行编码：

```
If ―match ―value = Text―string
```

该值编码应如[RFC2068]。

(32) If None Match字段

使用以下规则对if none match值进行编码：

```
If ―none ―match ―value = Text―string
```

该值编码应如[RFC2068]。

(33) If Range字段

使用以下规则对if range值进行编码：

```
If ―range = Text―string | Date―value
```

该值编码应如[RFC2068]。

(34) If Unmodified Since字段

使用以下规则对if unmodified since值进行编码：

```
If ―unmodified ―since ―value = Date―value
```

(35) Last Modified字段

使用以下规则对last modified值进行编码：

```
Last ―modified ―value = Date―value
```

(36) 位置字段

使用以下规则对位置值进行编码：

```
Location ―value = Uri―value
```

(37) Max Forwards字段

使用以下规则对max forwards值进行编码：

```
Max ―forwards ―value = Integer―value
```

(38) 附注字段

使用以下规则对附注值进行编码：

```
Pragma ―value = No―cache | Parameter
```

引用的文本字符串编码应如[RFC2068]。

(39) 代理鉴别字段

使用以下规则对代理鉴别值进行编码：

```
Proxy ―authenticate ―value = Value―length Challenge
```

(40) 代理授权字段

使用以下规则对proxy authorization**值进行编码：**

```
Proxy－authorization  －value = Value－length Credentials
```

(41) **公共字段**

使用以下规则对公共值进行编码：

```
Public －value = (Well－known －method | Token－text)
```

(42) **范围字段**

使用以下规则对范围值进行编码：

```
Range －value = Value－Length (Byte －range －spec | Suffix－byte －range －spec)
Byte －range －spec = Byte－range  First－byte －pos [ Last－byte －Pos ]
Suffix －byte －range －spec = Suffix－byte －range  Suffix－length
First －byte －pos = Uintvar－integer
Last －byte －pos = Uintvar－integer
Suffix －length = Uintvar－integer
Byte －range = <Octet 128>
Suffix －byte －range = <Octet 129>
```

(43) **参考字段**

使用以下规则对参考值进行编码：

```
Referer －value = Uri－value
```

(44) **重试字段**

使用以下规则对重试值进行编码：

```
Retry －after －value = Value－length (Retry －date －value | Retry－delta －
seconds)
Retry －date －value = Absolute－time  Date－value
Retry －delta －seconds = Relative－time  Delta－seconds －value
Absolute －time = <Octet 128>
Relative －time = <Octet 129>
```

(45) **服务器字段**

使用以下规则对服务器值进行编码：

```
Server －value = Text－string
```

该值编码应如[RFC2068]

(46) **传递编码字段**

使用以下规则对传递编码值进行编码：

```
Transfer －encoding －values = Chunked | Token－text
Chunked = <Octet 128>
```

(47) **升级字段**

使用以下规则对升级值进行编码：

```
Upgrade －value = Text－string
```

该值编码应如[RFC2068]。

(48) **用户代理字段**

使用以下规则对用户代理值进行编码：

```
User －agent －value = Text－string
```

该值编码应如[RFC2068]。

(49) **变更字段**

使用以下规则对变更值进行编码：

```
Vary －value = Field－name
```

(50) Via**字段**

使用以下规则对Via**值进行编码：**

```
Via －value = Text－string
```

该值编码应如[RFC2068]。

(51) **警告字段**

使用以下规则对警告代码值进行编码。警告代码值在 [RFC2068]**中定义。**

```
Warning = Warn－code | Warning－value
Warning －value = Value－length   Warn－code   Warn－agent   Warn－text
Warn －code = Short－integer
Warn －agent = Text－string
```

该值编码应如[RFC2068]。

```
Warn －text = Text－string
```

(52) WWW**鉴别字段**

使用以下规则对WWW**鉴别值进行编码：**

```
Proxy －authenticate  －value = Value－length Challenge
```

(53) **内容配置字段**

使用以下规则对内容配置字段进行编码（该字段用于提交表格数据时）。

```
Content －disposition  －value = Value－length   Disposition *(Parameter )
Disposition = Form－data | Attachment
Form －data = <Octet 128>
Attachment = <Octet 129>
```

14.5.5 多部分数据

HTTP1.1**采用**MIME**多部分格式传输复合数据对象（例如，多部分 /混合）。**WSP/B**定义了一种**MIME**（多用途的网际邮件扩充协议）多部分实体压缩二进制形式。在多部分实体和内容类型之间有一个直接转换，转换后，多部分/混合实体成为**X－wap.multipart/mixed**实体。这样，所有**MIME**多部分/内容类型都可以转换成“** X－wap.multipart/* **”内容类型，在转换中不会丢失信息。**

1. X－WAP.Multipart**格式（见图**14-31）

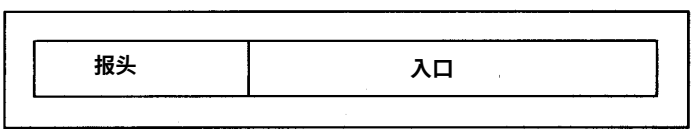

图14-31 X-WAP.Multipart**格式**

X－WAP.Multipart内容类型由一个后跟0或更多入口的报头组成。

2. 多部分报头（Multipart Header）

多部分报头格式如表14-84所示。nEntries字段规定了多部分实体的入口数量。

表14-84 多部分报头字段

名 字	类 型	用途/目的
nEntries	uintvar	多部分实体的入口数目

3. 多部分入口（Multipart Entry）

多部分入口格式如表14-85所示。

表14-85 多部分入口字段

名 字	类 型	用途/目的
HeadersLen	uintvar	ContentType和Headers字段的组合长度
DataLen	uintvar	Data字段的长度
ContentType	multiple 八位组（octets）	数据的内容类型
Headers	（HeadersLen length of ContentType）八位组（octets）	报头
Data	DataLen 八位组（octets）	数据

HeadersLen字段规定了ContentType和Headers字段的组合长度。

DataLen字段规定了多部分入口中的Data字段的长度。

ContentType字段包含数据的内容类型，它同14.5.4节“内容类型字段”规定的内容类型值编码一致。

Headers字段包含入口的报头。

Data字段包含入口的数据。

14.6 术语定义

本规范中使用的如下定义。

承载网络（Bearer Network） 承载网络用来携带在物理设备之间传递的传输层协议最终是会话层协议的消息。在会话存活期期间，有可能使用几个承载网络。

性能(Capability) 性能是在14.3.3节“性能”中介绍的术语，涉及到客户端或服务器支持的会话层协议工具和配置参数。

能力协商(Capability Negotiation) 能力协商是在14.3.3节“能力协商”中定义的机制，能力协商是使会话的功能与所选协议相互一致的一种机制。会话能力要在会话建立初期进行协商，通过能力协商，服务器的应用程序能够确定客户端是否支持某些特定的协议软件和配置。

客户端和服务器(Client and Server) 采用客户端和服务器这两个术语是为了使WSP与众所周知的已有系统相对应。客户端是发起会话请求的设备（或应用程序）。服务器是被动地等待客户端会话请求的设备，它既可以接受会话请求也可以拒绝会话请求。

为了使运行规模最小，实现WSP协议的设备可以只具备客户端或服务器的几项功能。客户端或服务器可以仅支持协议软件的一个子集，而子集的应用情况可以通过能力协商机制说明。

无连接模式会话业务(Connectionless Session Service) 无连接会话服务是一种不可靠的会话服务。在这种模式下，服务的使用者只能使用请求原语，服务的提供者只能使用指示原语。

连接模式会话服务(Connection-Mode Session Service) 连接模式会话服务(见14.3.3节)是一种可靠的会话服务。在这种模式下，服务的使用者可以使用请求和响应原语，服务的提供者可以使用指示和确认原语。

内容(Content) 同请求和响应一起发送的实体报文被归作内容。它由实体报头字段定义的格式和编码来进行编码。

内容协商(Content Negotiation) 内容协商是一种机制。当服务器为一个请求提供服务时，利用这种机制来选择响应的内容类型和编码方法。任何响应的内容类型和编码方法都可以进行协商，内容协商使服务器的应用程序能够确定一个客户端是否支持某种特定的内容格式。

实体(Entity) 实体是请求或响应传送的有效负荷信息。实体由实体头字段中的元信息和实体主体中的内容组成。

报头(Header) 头包含元信息。需要特别指出的是，会话报头包含了一个会话在其生存期内一直不变的综合信息。而实体头包含了关于某一请求、响应或实体主体（内容）的元信息。

层实体(Layer Entity) 在OSI体系结构中，参与提供层服务的层内有效（active）元素被称为层实体。

方法（Method） 方法是客户端请求的类型，它的定义类似于HTTP1.1（如Get、Post等）。一个WSP客户端使用方法和扩展方法调用服务器上的服务。

空结束字符串(Null Terminated String) 一个非零八位组（octet）序列，该序列以一个零八位组（octet）结尾。

对等地址四重组(Peer Address Quadruplet) 会话与一个特定的客户端地址、一个特定的客户端端口、一个特定的服务器地址和一个特定的服务器端口有关，这四个值的组合在这个规范中被称为对等地址四重组。

代理（Proxy） 是一个既能作服务器又能作客户端的中介程序。作为客户端，它代表其他的客户端提出请求。请求可以在代理内部进行处理，也可以在进行适当的解释之后，传送给其他的服务器进行处理。

Push和Pull数据传输(Pull and Push Data Transfer) 推和拉是因特网世界中的常用语言，它们分别描述了推操作事务处理和方法事务处理。服务器通过调用 WSP/B推服务，把数据“推”到客户端；反之，客户端通过调用WSP/B方法服务，把数据从服务器中“拉”出来。

会话(Session) 会话是为了进行事务处理而在两个应用程序之间建立的长生存期的通信上下文，用于事务处理和分类数据传送。

会话业务访问点(Session Service Access Point， S-SAP) 会话服务接入点是向上一层提供会话服务的一个逻辑点。

会话服务提供者(Session Service Provider) 会话服务提供者是一个层实体，它通过会话服务接入点S-SAP积极地参与提供会话服务。

会话服务用户(Session Service User) 会话服务用户是一个层实体，它通过会话服务接入点S-SAP向会话服务的提供者请求服务。

事务处理(Transaction) 这里定义了三种形式的事务。我们在描述事务这个术语时，没有使用与一般数据库事务相关的语义。

- 方法事务是一种由客户端发起的请求-响应-确认机制，用于调用服务器上的方法。
- 推（PUSH）事务是一种由服务器发起的请求-确认机制，用于把数据推到客户端。
- 传输事务是一种低级的事务处理原语，它由事务服务的提供者提供。

14.7 缩略语

本规范使用了如下缩写：

API	Application Programming Interface	应用编程接口
A-SAP	Application service Access Point	应用业务接入点
HTTP	Hypertext Transfer Protocol	超文本传输协议
ISO	International Organization for Standardization	国际标准化组织
MOM	Maximum Outstanding Method requests	最大未完成方法请求
MOP	Maximum Outstanding Push requests	最大未完成推进请求
MRU	Maximum Receive Unit	最大接收单元
OSI	Open System Interconnetion	开放系统互连
PDU	Protocol Data Unit	协议数据单元
S-SAP	Session Service Access Point	会话业务接入点
SDU	Service Data Unit	服务数据单元
SEC-SAP	Security Service Access Point	安全业务接入点
T-SAP	Transport Service Access Point	传输业务接入点
TID	Transaction Identifier	事务处理标识器
TR-SAP	Transaction Service Access Point	事务处理业务接入点
WDP	Wireless Datagram Protocol	无线数据报协议
WSP	Wireless Session Protocol	无线会话协议
WSP/B	Wireless Session Protocol - Browsing	无线会话协议/浏览
WTP	Wireless Transaction Protocol	无线事务协议

14.8 参考标准

[WAPARCH] "WAP Architecture Specification"，WAP Forum，30-April-1998
URL： http：//www.wapforum.org/

[WAPWDP] "Wireless Datagram Protocol Specification"，WAP Forum， 30-April-1998
URL： http：//www.wapforum.org/

[WAPWTP] "Wireless Transaction Protocol Specification"，WAP Forum，30-April-1998
URL： http：//www.wapforum.org/

[RFC2119] "Key Words for Use in RFCs to Indicate Requirement Levels"，Bradner， S.， March 1997
URL： ftp：//ftp.isi.edu/in-notes/rfc2119.txt.

[RFC2068] "Hypertext Transfer Protocol—HTTP/1.1"，Fielding，R.，et.al .，January 1997
URL：ftp：//ftp.isi.edu/in-notes/rfc2068.txt.

[RFC1521] "MIME （Multipurpose Internet Mail Extensions）Part One：Mechanisms for Specifying and Describing the Format of Internet Message Bodies"，Borenstein，N.，et. al.，September 1993
URL：ftp：//ftp.isi.edu/in-notes/rfc1521.txt.

[RFC2047] "MIME （Multipurpose Internet Mail Extensions） Part Three：Message Header Extensions for Non-ASCII Text"，Moore，K.，November 1996
URL：ftp：//ftp.isi.edu/in-notes/rfc2047.txt.

[RFC822] "Standard for the Format of ARPA Internet Text Messages"，Crocker，D.，August 1982
URL：ftp：//ftp.isi.edu/in-notes/rfc822.txt.

14.9 参考资料

[ISO7498] "Information Technology—Open Systems Interconnection—Basic Reference Model：The Basic Model"，ISO/IEC 7498-1：1994

[ISO10731] "Information Technology Open Systems Interconnection Basic Reference Model Conventions for the Definition of OSI Services"，ISO/IEC 10731：1994

[RFC1630] "Universal Resource Identifiers in WWW，a Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World Wide Web"，Berners-Lee，T.，June 1994
URL：ftp：//ftp.isi.edu/in—notes/rfc1630.txt.

[RFC1738] "Uniform Resource Locators （URL）"，Berners-Lee，T.，et.al.，December 1994
URL：ftp：//ftp.isi.edu/in-notes/rfc1738.txt.

[RFC1808] "Relative Uniform Resource Locators"，Fielding，R.，June 1995
URL：ftp：//ftp.isi.edu/in-notes/rfc1808.txt.

[RFC1864] "The Content-MD5 Header Field"，Meyers，J. and Rose，M.，October 1995
URL：ftp：//ftp.isi.edu/in—notes/rfc1864.txt.

14.10 号码分配

本节包含了WSP/B**号码分配表（如表**14-86**～表14-**94**）。**WAP**体系结构组织负责管理号码的分配。使用**IANA**分配的**MIBEnum**值对字符集进行编码，并在** URL：ftp：//ftp.isi.edu/in—notes/iana/assignments/character—sets**处进行登记注册。表**14-86**仅提供了一个快速的引用。

表14-86 PDU类型分配

名 字	分配的号码
Reserved	0x00
Connect	0x01
ConnectReply	0x02
Redirect	0x03
Reply	0x04
Disconnect	0x05
Push	0x06
ConfirmedPush	0x07
Suspend	0x08
Resume	0x09
Unassigned	0x10～0x3F
Get	0x40
Options（Get PDU）	0x41
Head（Get PDU）	0x42
Delete（Get PDU）	0x43
Trace（Get PDU）	0x44
Unassigned（Get PDU）	0x45～0x4F
Extended Method（Get PDU）	0x50～0x5F
Post	0x60
Put（Post PDU）	0x61
Unassigned（Post PDU）	0x62～0x6F
Extended Method（Post PDU）	0x70～0x7F
Reserved	0x80～0xFF

表14-87 放弃原因代码分配

名 字	分 配 描 述	号 码
PROTOERR	协议错误，收到不合法PDU	0xE0
DISCONNECT	会话断开	0xE1
SUSPEND	会话挂起	0xE2
RESUME	会话恢复	0xE3
CONGESTION	通信堵塞，不能处理该SDU	0xE4
CONNECTERR	会话连接失败	0xE5
MRUEXCEEDED	超过最大接收单元数目	0xE6
MOREXCEEDED	超过最大未完成请求数目	0xE7
PEERREQ	对等请求	0xE8
NETERR	网络错误	0xE9
USERREQ	用户请求	0xEA

表14-88 状态代码分配

HTTP状态代码	描 述	分配的号码
空	保留	0x00－0x0F
100	继续	0x10
101	协议转换	0x11

（续）

HTTP状态代码	描　述	分配的号码
200	成功	0x20
201	创建	0x21
202	接受	0x22
203	未赋权信息	0x23
204	没有内容	0x24
205	内容重置	0x25
206	部分内容	0x26
300	多重选择	0x30
301	永久转移	0x31
302	暂时移动	0x32
303	另外的参考根据	0x33
304	未改变	0x34
305	使用代理	0x35
400	服务器不能理解的错误请求	0x40
401	未赋权	0x41
402	请求付费	0x42
403	禁止的操作被拒绝	0x43
404	未发现	0x44
405	不允许的方法	0x45
406	不能接受的	0x46
407	请求代理鉴权	0x47
408	请求超时	0x48
409	冲突	0x49
410	离开	0x4A
411	需要的长度	0x4B
412	预处理失败	0x4C
413	请求的实体太大	0x4D
414	请求的URI太大	0x4E
415	未支持的媒体类型	0x4F
500	服务器内部错误	0x60
501	未实现	0x61
502	错误的网关	0x62
503	无效的业务	0x63
504	网关超时	0x64
505	不支持的HTTP版本	0x65

表14-89 性能分配

性　能	分配的号码
Client－SDU－Size	0x00
Server－SDU－Size	0x01
Protocol options	0x02
Method－MOR	0x03
Push－MOR	0x04
Extended methods	0x05
Header code pages	0x06
Aliases	0x07
Unassigned	0x08～0x7F

表14-90 知名参数分配

记 号	分配的号码	值的BNF格式
Q	0x00	Q－value
Charset	0x01	Well－known－charset
Level	0x02	Version－value
Type	0x03	Integer－value
Uaprof	0x04	Untyped－value
Name	0x05	Text－string
Filename	0x06	Text－string
Differences	0x07	Field－name
Padding	0x08	Short－integer

表14-91 报头字段名分配

名 称	分配的号码	名 称	分配的号码
Content－Type	0x11	User－Agent	0x29
Date	0x12	Vary	0x2A
Etag	0x13	Via	0x2B
Expires	0x14	Warning	0x2C
From	0x15	WWW－Authenticate	0x2D
Host	0x16	Content－Disposition	0x2E
If－Modified－Since	0x17		

表14-92 内容类型分配

内 容 类 型	分配的号码	内 容 类 型	分配的号码
/	0x00	application/x－wap.wmlc	0x14
text/*	0x01	application/x－wap.wmlscriptc	0x15
text/html	0x02	application/x－wap.wta－eventc	0x16
text/plain	0x03	application/x－wap.uaprof	0x17
text/x－hdml	0x04	application/x－wap.wtls－ca－certificate	0x18
text/x－ttml	0x05	application/x－wap.wtls－user－certificate	0x19
text/x－vCalendar	0x06	application/x－x509－ca－cert	0x1A
text/x－vCard	0x07	application/x－x509－user－cert	0x1B
text/x－wap.wml	0x08	image/*	0x1C
text/x－wap.wmlscript	0x09	image/gif	0x1D
text/x－wap.wta－event	0x0A	image/jpeg	0x1E
multipart/*	0x0B	image/tiff	0x1F
multipart/mixed	0x0C	image/png	0x20
multipart/form－data	0x0D	image/x－wap.wbmp	0x21
multipart/byteranges	0x0E	x－wap.multipart/*	0x22
multipart/alternative	0x0F	x－wap.multipart/mixed	0x23
application/*	0x10	x－wap.multipart/form－data	0x24
application/java－vm	0x11	x－wap.multipart/byteranges	0x25
application/x－www－form－urlencoded	0x12	x－wap.multipart/alternative	0x26
application/x－hdmlc	0x13	Unassigned	0x27～x7F

表14-93 ISO 639语言分配集

语 言	简 写	分配的号码	语 言	简 写	分配的号码
Afar		0x01	Maori		0x47
Abkhazian		0x02	Macedonian	mk	0x48
Afrikaans	af	0x03	Malayalam		0x49
Amharic		0x04	Mongolian		0x4A
Arabic		0x05	Moldavian		0x4B
Assamese		0x06	Marathi		0x4C
Aymara		0x07	Malay		0x4D
Azerbaijani		0x08	Maltese		0x4E
Bashkir		0x09	Burmese		0x4F
Byelorussian	be	0x0A	Nauru		0x50
Bulgarian	bg	0x0B	Nepali		0x51
Bihari		0x0C	Dutch	nl	0x52
Bislama		0x0D	Norwegian	no	0x53
Bengali；Bangla		0x0E	Occitan		0x54
Tibetan		0x0F	(Afan) Oromo		0x55
Breton		0x10	Oriya		0x56
Catalan	ca	0x11	Punjabi		0x57
Corsican		0x12	Polish	po	0x58
Czech	cs	0x13	Pashto，Pushto		0x59
Welsh		0x14	Portuguese	pt	0x5A
Danish	da	0x15	Quechua		0x5B
German	de	0x16	Rhaeto－Romance		0x5C
Bhutani		0x17	Kirundi		0x5D
Greek	el	0x18	Romanian	ro	0x5E
English	en	0x19	Russian	ru	0x5F
Esperanto		0x1A	Kinyarwanda		0x60
Spanish	es	0x1B	Sanskrit		0x61
Estonian		0x1C	Sindhi		0x62
Basque	eu	0x1D	Sangho		0x63
Persian		0x1E	Serbo－Croatian		0x64
Finnish	fi	0x1F	Sinhalese		0x65
Fiji		0x20	Slovak	sk	0x66
Faeroese	fo	0x21	Slovenian	sl	0x67
French	fr	0x22	Samoan		0x68
Frisian		0x23	Shona		0x69
Irish	ga	0x24	Somali		0x6A
Scots Gaelic	gd	0x25	Albanian	sq	0x6B
Galician	gl	0x26	Serbian	sr	0x6C
Guarani		0x27	Siswati		0x6D
Gujarati		0x28	Sesotho		0x6E
Hausa		0x29	Sundanese		0x6F
Hebrew (formerly iw)		0x2A	Swedish	sv	0x70
Hindi		0x2B	Swahili		0x71
Croatian	hr	0x2C	Tamil		0x72
Hungarian	hu	0x2D	Telugu		0x73
Armenian		0x2E	Tajik		0x74

（续）

语 言	简 写	分配的号码	语 言	简 写	分配的号码
Interlingua		0x2F	Thai		0x75
Indonesian（formerly in）	id	0x30	Tigrinya		0x76
Interlingue		0x31	Turkmen		0x77
Inupiak		0x32	Tagalog		0x78
Icelandic	is	0x33	Setswana		0x79
Italian	it	0x34	Tonga		0x7A
Inuktitut		0x35	Turkish	tr	0x7B
Japanese	ja	0x36	Tsonga		0x7C
Javanese		0x37	Tatar		0x7D
Georgian		0x38	Twi		0x7E
Kazakh		0x39	Uighur		0x7F
Greenlandic		0x3A	Ukrainian	uk	0x81
Cambodian		0x3B	Urdu		0x82
Kannada		0x3C	Uzbek		0x83
Korean	ko	0x3D	Vietnamese		0x84
Kashmiri		0x3E	Volapuk		0x85
Kurdish		0x3F	Wolof		0x86
Kirghiz		0x40	Xhosa		0x87
Latin		0x41	Yiddish（formerly ji）		0x88
Lingala		0x42	Yoruba		0x89
Laothian		0x43	Zhuang		0x8A
Lithuanian		0x44	Chinese	zh	0x8B
Latvian，Lettish		0x45	Zulu		0x8C
Malagasy		0x46			

表14-94 字符集分配示例

字 符 集	分配的号码	IANA MIBENUM值
big5	0x07EA	2026
iso－10646－ucs－2	0x03E8	1000
iso－8859－1	0x04	4
iso－8859－2	0x05	5
iso－8859－3	0x06	6
iso－8859－4	0x07	7
iso－8859－5	0x08	8
iso－8859－6	0x09	9
iso－8859－7	0x0A	10
iso－8859－8	0x0B	11
iso－8859－9	0x0C	12
shift_JIS	0x11	17
us－ascii	0x03	3
utf－8	0x6A	106
gsm－default－alphabet	**未分配**	**未分配**

14.11 报头编码举例

本节举出几个示例来说明如何进行报头编码。

14.11.1 报头值

对报头编码的赋值符合HTTP1.1句法，并同相应的WSP/B报头编码字节流一起给出。

1. 原语值编码

HTTP1.1报头：Accept：application/x－wap.wmlc

报头编码：

0x80 知名字段名“ Accept ”作为一个短整数编码

0x94 知名媒体“ Application/x－wap.wmlc ”作为一个短整数编码

2. 结构值编码

HTTP1.1 报头：Accept－Language：en; q=0.7

报头编码：

0x83 知名字段名“ Accept－Language ”

0x02 值长度，必须进行普通编码

0x99 知名语言“ English ”

0x47 质量因子是0.7 （0.7 * 100 + 1 = 0x47）

3. 知名列表值编码

HTTP1.1 报头：Accept－Language：en，sv

报头编码：

0x83 知名字段名“ Accept－Language ”

0x99 知名语言“ English ”

0x83 知名字段名“ Accept－Language ”

0xF0 知名语言“ Swedish ”

4. 数据值编码

HTTP1.1 报头：Date：Thu，23 Apr 1998 13：41：37 GMT

报头编码：

0x92 知名字段名“ Date ”

0x04 多字节整数长度

0x35 4字节数据的编码应是1970－01－01， 0：00：00 GMT 开始的秒。最高有效字节在最前面

0x3f －

0x45 －

0x11

5. 内容范围编码

HTTP1.1 报头：Content－range： bytes 0－499/1025

报头编码：

0x90 知名字段名“ Content－range ”

0x03 长度值

0x00 首字节的位置
0x88 实体长度
0x01 实体长度
6. 未赋值记号编码
HTTP1.1报头：Accept－ranges：new－range－unit
报头编码：
0x84 知名字段名“ Accept－ranges ”
'n"e"w"-"r"a"n"g"e"-"u"i"n"t'0x00 记号作为一个零结束文本字符串来编码
7. 未赋值报头字段名的编码
HTTP1.1报头：X－New－header： foo
报头编码：
'X"-'N"e"w"-"h"e"a"d"e"r'0x00 字段名作为零结束文本字符串来编码
'f"o"o'0x00 字段值作为零结束文本字符串来编码
8. 未赋值列表值报头的编码
HTTP1.1报头：X－New－header：foo，bar
报头编码：
'X"-'N"e"w"-"h"e"a"d"e"r'0x00 字段名作为零结束文本字符串来编码
'f"o"o'，"b"a"r'0x00 字段值作为零结束文本字符串来编码

14.11.2 转换报头代码页

本节举例说明了如何转换报头代码页。
1. 转换序列
转换到报头代码页64
编码转换序列：
0x7F 转换分隔符
0x40 页标识
2. 快捷方法
转换到报头代码页16
编码转换序列：
0x10 快速转换分隔符

14.12 实现注释

下列实现的注释能够对WSP协议的具体执行效果产生深刻影响，这些注释对协议的实施者具有指导作用。

14.12.1 确认推事务处理和确认延迟

WTP的一个特点就是延迟发出对事务处理的确认信息，这可能大大减少在承载网络上发送的消息数量。但是，这个特点同样会大大降低推事务的吞吐量，特别是如果服务器在开始下一个确认推事务处理之前需要等待前一个推事务处理的确认信息时，这种情况更为严重。

延迟确认，会造成进行一个推事务所耗费的时间，不仅包括消息来回传递的时间，还包括确认延迟的时间。结果是加剧了承载网络的长时间延迟，最终导致 WTP要使用一个较大的延迟确认时间值。

会话层协议没有讨论这个问题，这是因为 WTP业务接口并没有提供改变确认延迟定时器的手段，如何控制定时器是一个具体实现的问题。如果很看重具体实现的性能，那么具体实现应能够使服务器用户可为每一个推事务处理规定可接受的最大确认延迟时间。为获得一个好的推事务处理吞吐量，对确认延迟定时器赋一个尽可能小的值并不是一个好的解决办法，这会妨碍优化WTP方法请求的通信，从而造成在空中接口上发送的消息数量加倍。

14.12.2 竞争管理

连接方法WSP/B在WTP中提供业务顶端分层， WTP并不能保证事务请求和结果到达通信对方的顺序和服务器用户提交它们的顺序是一致的。在这种情况下，如果方法或推事务已被初始化，而会话创建进程却还没有完成，就会导致竞争。为减少协议复杂度， WSP/B处理竞争的方法比较简单，在大多数情况下只是选择放弃引起竞争的事务。结果是：放弃事务的原因（reason）会被报告成DISCONNECT。换句话说，这会理解成该会话不存在，而事实上该会话是存在的并且可以使用该会话，在这种情况下，服务器用户仅仅需要重试对该事务的请求。

采用这种规则是由于竞争的情况并不多见，不值得用增加协议复杂度的方法来解决竞争。然而，如果很看重这个问题，那么可以采用某种实现策略来解决这个问题。首先，如果会话管理、方法和推等事务处理几乎同时被引发，那么就会产生竞争。这些 WTP组装进程会把产生的PDU拼接在一起成为一个传输数据报。 WTP管理这些组装和拆卸同样可以采用如下的方法：在同一传输数据报中的 PDU的顺序同相应操作的顺序是一致的，这就使得 WSP/B状态机在完成会话创建之前不必对方法和推事务处理等原语作出反应。

如果具体实现完全禁止资源的竞争，那么具体实现可以推迟引发方法和推事务处理直到完成会话创建。就协议方而言，这是合法的，但是，如果这个承载网络的往返时间很长，所产生的延迟时间会让用户难以忍受。

14.12.3 会话断开和会话挂起的优化

协议要求当通信一方断开或挂起会话时，所有挂起的方法和推事务处理都将被放弃，因此在传输断开 PDU或挂起PDU的同时，还应传输事务处理放弃 PDU。这些PDU都很短，因此可以组装成为一个传输数据报。具体实现应该可以在 WTP级别组装这些数据报，至少在这个例子中是可以的，这可使得网络的影响减至最小。

14.12.4 报头编码的解译

WSP/B为HTTP1.1报头定义了压缩二进制编码。一种方法是根据语境信息来定义如何解释某一具体编码，而不是直接编码。例如，报头字段名暗示了报头字段值的格式，如果是一个结构类型值，每一项的位置就暗示了该项的类型。即使二进制编码是一样的，但由于位置的不同所以代表的类型值也不同，最显而易见的方法是使用从上到下的策略来解释报头编码。具体实现能够利用这种方法支持上述编码技术。

14.12.5 增加知名参数和记号

由WSP/B定义的报头编码对报头字段值规定了严格的句法。在报头字段内，只有对已确定为知名二进制形式的数值才能进行压缩编码。如果应用程序广泛使用记号值，特别是参数，而它们是不能预知的，所以会禁止对所需文本进行编码。如果并不把 WSP/B升级成一个新版本，那么在WSP/B框架内可实施更有效的编码方案。应用程序可能会使用扩展报头代码页，它规定其必须符合标准 HTTP1.1报头的句法，以便识别所需要的新的知名数值，然后应用双方就可使用WSP/B能力协商在如何使用这个新的代码页方面进行约定。一旦达成约定，应用就可以修改它的报头处理，这样就可以使用在新代码页上定义的报头，以代替标准报头，但报头名字并未改变。转换到新代码页只需额外的一个字节，这会抵消数值压缩编码所节省的开销。