下一代网络 CPL 解释器的实现①

Implementation for Interpreter of CPL Based on NGN

黄兴平　范冰冰　张奇支　(华南师范大学 计算机学院 广东 广州 510631)

摘　要： 简化业务开发和提供是下一代网络(NGN)发展的关键，业务描述工具 CPL 是 NGN 创建业务的重要工具。研究了 CPL 解释器的网络功能和基本工作机制，提出了 CPL 解释器的设计与实现框架，并从执行效率和扩展性两个方面对 CPL 解释器设计进行了深入讨论，通过 CPL 解释器工作流程对 CPL 解释器的工作机制进行了进一步的说明。所提出的 CPL 解释器原型在下一代网络业务应用服务器的开发中得到了应用，有效支持了业务开发的简化，获得了良好的应用效果。

关键词： 下一代网络 呼叫处理语言 解释器 业务生成环境

1　引言

下一代网络是业务驱动的网络，如何方便、快速、经济、有效地为用户提供业务，是下一代网络发展中的关键问题。

在传统电信网中，电信业务的提供都是由电信业务运营商根据市场的需求,制订业务技术规范，由专门的业务提供商使用特定工具例如 IN[1]技术开发的。但是这种方式有其明显的缺陷：1)市场反应速度慢，2)技术封闭、参与人员少，3)不能照顾细分市场和满足用户的个性化需要。

互联网开放业务环境和丰富的业务种类促进了互联网的迅速发展，也极大地冲击了电信网的发展。为了有效提高通信网络的使用效率和市场价值，提高运营商的竞争能力和用户的忠诚度，开放业务技术成为 NGN 研究的热点问题。基于 Parlay API[2]、JAIN API[3]开发业务的技术使用 IT 领域内的工具，无需电信领域的专门开发工具，使得第三方业务开发成为可能。同时，第三方业务开发带来的竞争与合作有利于缩短业务的开发周期，加快业务的部署。但这种开发方式仍要求业务开发人员有较多的电信领域知识和计算机知识。即使一些非常简单的业务，如根据自己的时间计划进行来电呼叫转移，客户也不能随意定制，这极大地限制了业务的应用。因此，对接口进一步封装,向最终业务用户开放，实现业务用户定制，促进这类简单业务得到更好的应用是满足人们需要和增加运营商收入的迫切需求。

在下一代网络和 IMS 体系中，SIP 协议占有主导地位。CPL(Call Processing Language)[4,5]是 IPTEL WG(IP Telephony Work Group) 推荐的在 H323/SIP 网络系统中使用用来解释和控制业务的呼叫处理语言。CPL 推出的目的是为了有效简化业务的开发，使得业务提供商、第三方业务开发商、终端用户都可以根据需要参与到业务的定义和使用中，从而有效地促进业务的繁荣和发展。

CPL 具有功能强大、资源消耗少、工作高效、容易实现、易于检验、运行安全、方便编写和处理、独立于操作系统/网络控制协议、可扩展性好等优点，由于具有以上突出优势，CPL 在业务开发中被广泛看好，被认为将成为下一代网络中创建业务的重要工具之一。

CPL 解释器是 CPL 业务实现和运行的核心，但是当前对于 CPL 解释器的实现技术研究较少，为了有效促进业务的发展，有必要对其解释器的实现进行研究，分析解决其关键技术，以有效促进下一代网络业务的发展。

本文首先对下一代网络业务脚本语言 CPL 的网络功能与结构进行了分析；然后研究了 CPL 解释器的基

① 基金项目:国家自然科学基金(60873016);国家高技术研究发展计划(863)(2009AA01Z102);广东省科技计划(2007B010200069)
收稿时间:2009-01-11

本工作机制，提出了一套完整的设计和实现方案，并从执行效率和扩展性两个方面对 CPL 解释器设计和实现进行深入讨论，最后结合业务流程对 CPL 解释器的运行过程进行了说明。

2 CPL功能与结构

2.1 CPL 网络功能

在下一代网络中，提供业务的应用服务器标准主要有三种，基于 Parlay/OSA 模型的应用服务器，基于移动 IN 标准的应用服务器以及基于会话发起协议(Session Initiation Protocol/SIP)[6]的应用服务器。CPL 本身是与协议无关的业务脚本语言，可以应用在各种应用服务器中，鉴于 SIP 在下一代网络中的重要地位，本文以 CPL 在 SIP 应用服务器中的应用为切入点研究 CPL 的功能和应用。

SIP 应用服务器业务的开发有多种方法。其中基于 SIP 协议的直接开发较为灵活和直接，一般而言，使用 SIP 的应用程序和服务逻辑可以使用任何编程语言开发。IETF 推荐开发 SIP 业务可以使用 3 种编程语言来实现基于 SIP 的业务逻辑：CPL、SIP-CGI 和 SIP Servlets(JSR116)。SIP 应用服务器的体系结构如图 1 所示。

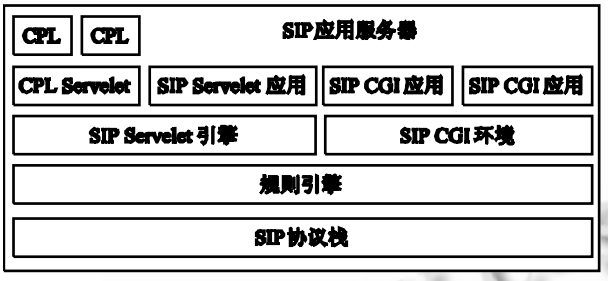

图 1 SIP 应用服务器体系结构

在图 1 所示的层次中，越往上，接口层次越抽象，其中 CPL 调用 SIP 封装后较为抽象的构件接口。SIP CGI 来源于 HTTP CGI，提供一种在 web 环境下创建新业务的机制；SIP Servlets 特定于 Java 的脚本语言，此语言相对较为原始，与 SIP 协议本身的协议接口耦合较为紧密。

CPL 是一种基于 XML 的脚本语言，主要用来描述和控制个性化的 Internet 电话业务(包括呼叫策略路由、呼叫筛选、呼叫日志等业务)，单个的脚本经过脚本解释器解释后才能被执行，因此可以通过设计应用服务器的脚本控制部分对 CPL 脚本进行严格的完整性、可靠性校验，从而保证由普通终端用户编写的 CPL 业务逻辑不会对应用服务器造成不可预知的损坏，因此可以同时面对服务器的开发人员和普通终端用户。

2.2 CPL 脚本实例与结构分析

下面举一个 CPL 脚本例子说明 CPL 内容结构以及对其进行结构特点进行分析:

```
< ?xml version =“1. 0”?>
< ! DOCTYPE cpl PUBLIC “ - / / IETF/ / DTD
RFCxxxx CPL 1. 0/ / EN”“cpl. dtd”>
< cpl >
< ancillary > < / ancillary >
< subaction id =“voicemail”>
<location url =“sip :jones@voicemail. example.
com”>
< proxy / >
< / location >
< / subaction >
< incoming >
< location url =“sip :jones@jonespc. example.
com”>
< proxy timeout =“8”>
< busy >
< sub ref =“voicemail”/ >
< /busy >
< noanswer >
< sub ref =“voicemail”/ >
< /noanswer >
< / proxy >
< / location >
< / incoming >
< / cpl >
```

这段 CPL 脚本可以分为 3 部分。第 1 部分是<ancillary > </ancillary >，说明一些辅助信息。第 2 部分是<subaction > </subaction > 之间的内容，定义可以被调用的子动作，这里主要定义了一个 ID 号为“voicemail”的 SIP 地址 jones@voicemail. example.com。第 3 部分是高层动作，代表了用户对呼叫业务的个人定制处理，在这个例子中是以呼叫来时的处理来说明的。关键字 incoming 代表了来话。<incoming > </incoming > 之间的内容，定义了

一个应答 SIP 地址: jones@jonespc.example.com 以及未获应答时调用第二部分 subaction 所定义动作的处理。

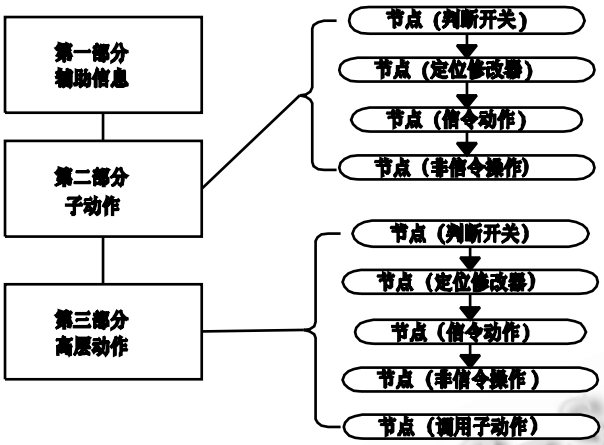

图 2　CPL 文档结构

在 IETF RFC3880[5]中，CPL 定义了 4 种功能节点:判断开关(用来提供施行 CPL 脚本的选择标准) 、定位修改器(用来确定、查询和删除地址定位表中的定位项) 、信令动作(引发相应的信令事件) 、非信令操作(触发辅助性事件)，其文档整体结构如图 2 所示。

可以看出，CPL 脚本可以抽象为节点的组合，由一系列的“功能”节点形成一个完整的 CPL 脚本。

3　CPL解释器设计和实现

3.1 CPL 解释器基本工作机制

CPL 解释器及其业务脚本作为一种业务运行在 SIP 应用服务器的业务逻辑执行环境(Service Logic Execute Environment/SLEE)中，在 CPL 业务作为业务加载到 SLEE 后，被激活时向 SLEE 登记其所关心的事件。CPL 解释器在 SIP 应用服务器中的执行情况如图 3 所示。

由图 3 所示，CPL 业务的运行过程可以分解为如下步骤:

1) 当产生 CPL 解释器所关心的事件时，核心网中软交换或 IMS 系统将它上报给 SLEE;

2) SLEE 通知 CPL 解释器有相应业务事件;

3) CPL 解释器根据上报的消息，查找相应的 CPL 脚本;

4) CPL 解释器解释脚本，控制业务的执行;

5) CPL 解释器根据对 CPL 脚本解释的结果，向 SLEE 发出下一步执行的请求;

6) SLEE 将请求转发给网络，由网络完成实际控制处理。

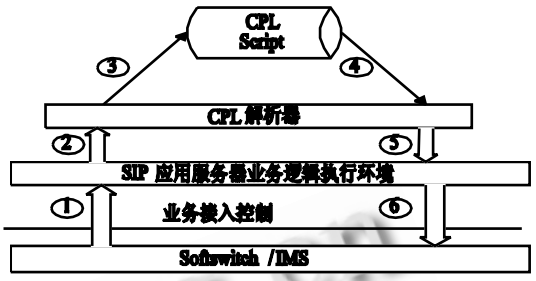

图 3　CPL 解释器工作机制

通过以上步骤，就完成了 CPL 业务的处理过程，可以看出，通过在 SIP 应用服务器之上叠加 CPL 解释器，可以很好地支持用户业务定制的功能要求，进一步分离了业务的执行和网络的控制功能。CPL 解释器在 CPL 业务的处理中处于核心地位,设计一个具有良好性能和可扩展性的解释器对于有效支持 CPL 业务的发展以及未来 CPL 的语义扩展演进具有非常重要的意义。

3.2 CPL 解释器设计

CPL 业务逻辑使用基于 XML 的 CPL 脚本语言描述，可以由业务生成环境或人工编写生成，由业务生成环境验证，以脚本文件或文档对象模型(Document Object Model/DOM)[7]树形式存放，由 CPL 解释器负责解释执行。SLEE 管理业务逻辑组件 API 以使得组件能被上层业务逻辑正确调用，因此，CPL 业务在 SLEE 上执行，必须通过 CPL 解释器，调用构件 API 提供的能力，完成业务逻辑处理。

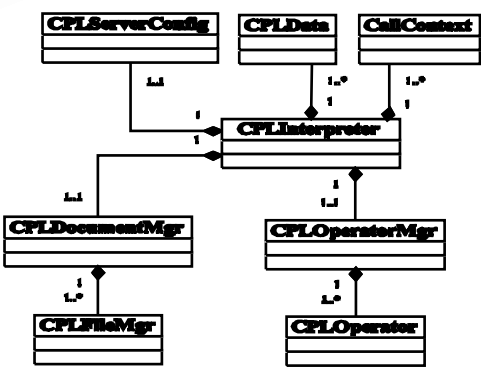

图 4　CPL 主控模块类设计

因此，CPL 解释器的主要功能是:

1) 根据网络侧触发请求找到与此业务对应的脚

本文件。

2) 将脚本文件转换成 DOM 对象。

3) 解释 DOM 树中的节点，根据节点规定的业务逻辑调用相应组件 API，执行对应操作。

4) 根据呼叫信息及脚本决定下一个执行节点，直至执行完 DOM 树的一个分支至叶子节点。

由上分析，CPL 解释器主控模块类设计如图 4 所示。图 4 描述了 CPL 解释器包含的主要类及其之间的相互关系，其中 CPLInterpreter 是解释器的主控核心类，它接收 SLEE 上报的事件并分发事件，依据脚本内容执行业务逻辑。

CPLServerConfig 类是 CPLInterpreter 初始化时调用，读取系统配置文件，完成 CPL 解释器的初始化工作。

CPLDocumentMgr 类的作用是维护脚本与 DOM 中 Document 对象之间一一对应关系，根据执行环境上报消息提取的 trigger 值返回相应的 Document 对象，其中这个 trigger 是从消息中提取的脚本文件标识，以确认对应到哪种业务。

CPLData 类的作用时保存当前待处理标记节点元素的信息以及呼叫相关的信息。

CallContext 类保存当前呼叫会话的上下文信息。

CPLFileMgr 类的作用就是从外部，例如本地文件、FTP 等处装载指定的脚本文件，供 CPLDocumentMgr 类调用。

CPLOperatorMgr 类的作用是维护着脚本中的节点 node 和与其对应的 operation 对象间的一一对应关系，管理维护脚本的执行顺序。

CPLOperator 类为所有标记操作的父类，定义了所有操作必须完成的方法。

3.3 hashtable 提高装载效率

在 CPL 业务的执行中，根据上报消息提取的 trigger 值寻找相应的业务文件并加载是比较费时间的。为了有效提高执行的效率，CPLDocumentMgr 可以在内存中维护一部分常用的脚本 DOM 对象，并映射到一个缓存散列表(hashtable)中，以提高查找执行的效率。因此，执行 CPL 业务时对脚本文件的查找与载入可以分为两类：

1) 若对象已存在 hashtable，直接返回 Document 对象；

2) 否则，根据脚本的存放位置(由配置文件确定)，通过 CPLFileMgr 类中的方法获得相应的 Document 对象。

3.4 State 模式优化 CPL 业务逻辑设计

在 CPL 的业务逻辑脚本中，变化最大的就是业务逻辑的节点操作和状态变化，可以体现在：

1) CPL 业务的执行是依照呼叫信息，按照给定的脚本执行业务逻辑。业务脚本反映的是业务状态的转移,每个标记就是一个状态。每一个业务脚本以及每个脚本的每个操作都有可能完全不一样；

2) CPL 的业务脚本操作的语义定义很有可能会有扩展的需要，为了保持软件演进的稳定性，在设计和实现时必须有充分的设计考虑。

针对第一种情况，在传统的软件设计中往往以 if-else 或 switch 条件分支结构来实现，往往导致结构复杂并且难以修改和维护。这里引入 State 模式[8]来优化 CPL 业务逻辑的操作控制的设计，可以较好解决上述问题，如图 5 所示。

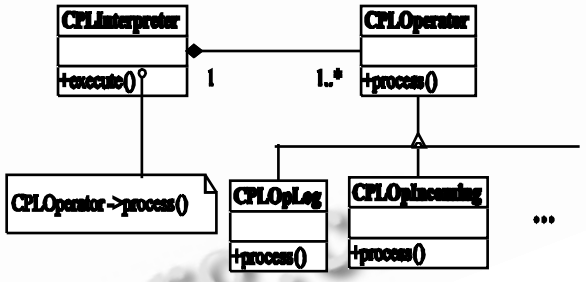

图 5 State 模式优化 CPL 设计

需要指出的是，CPLInterpreter 类是通过 CPL OperatorMgr 获取到 CPLOperator 类实例的，这里为了简化图形而直接以聚合关系表示。

以 State 模式实现 CPL 脚本的操作，把软件变化性[9]很好地控制在了 CPLOperator 类的子类上，为高层的 CPLInterpreter 类提供了一致的调用接口。业务逻辑的状态变化以及脚本语义的扩充例如扩充新的操作子类，对高层都不会有任何影响，很好地满足了前面提到的两点变化性需求，有效简化了 CPL 解释器的设计，同时也使得 CPL 解释器有很好的演进性，能满足未来的需求。

3.5 CPL 解释器工作流程

为了更清晰地理解 CPL 解释器的工作流程，图 6 阐述了 CPL 收到请求后的业务消息序列图。

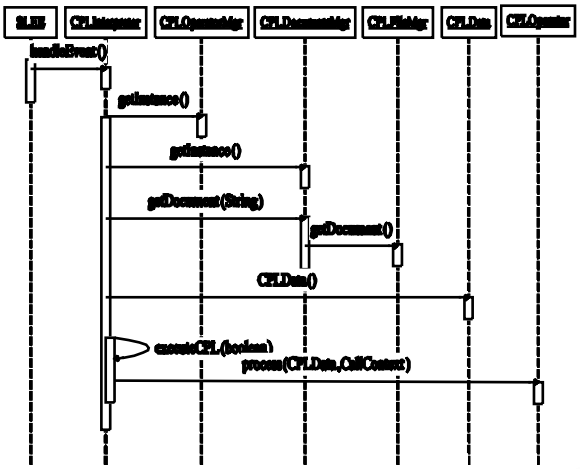

图 6　CPL 解释器工作流程

由图 6 所示，CPL 解释器解释执行业务的步骤可以分解为：

1) SIP 应用服务器产生事件通知；

2) CPLInterpreter 产生管理所有操作的实例；

3) CPLInterpreter 产生获取 Document 对象的实例；

4) 由 CPLDocumentMgr 实例依据呼叫上下文获得相应的 Document 对象；

5) 设置 CPLData；

6) 执行解释器 execute()方法；

7) 调用执行对应节点的 process()方法，完成业务执行。

从执行流程可以看出，本文提出的 CPL 解释器的设计方案逻辑流程清晰简洁，有很好的适应性。

4　总结

为了有效促进 CPL 业务的开发和应用，本文研究了 CPL 在 SIP 应用服务器中的应用，在分析 CPL 解释器工作机制和 CPL 业务文档结构特点的基础上，提出了 CPL 解释器的设计与实现方案。

为了使得 CPL 业务逻辑执行具有较高效率，针对脚本的寻找提出了采用缓存载入常用的业务逻辑脚本并以 Hashtable 查找的方式优化执行效率的方法。

为了有效控制 CPL 业务逻辑状态的变化性以及满足业务语义扩充演进的需要，提出以 State 模式实现 CPL 的节点操作的方法，简化了 CPL 解释器的设计，具有良好的扩充性。

本文设计的 CPL 解释器原型在实现 SIP 应用服务器的过程中得到了应用，有效支持了业务开发的抽象和简化，提高了下一代网络业务开发的效率，取得了良好的应用效果，能有效促进下一代网络业务的应用与发展。

参考文献

1 ITU-T Recommendation, Q.1224, IN, 1998.

2 ETSI ES 203 915-1 v1.2.1(2007-01). Open Service Access (OSA) API, Part1-Part5.

3 Java call control (JCC) application programming interface(API)-2002.http://java.sun.com/products/jain, 2002－09.

4 RFC2824, Call Processing Language Framework and Requirement, 2000.5.

5 RFC3880,Call Processing Language(CPL):A Language for User Control of Internet Telephony Services IETF, 2004.10.

6 RFC 3261, SIP: Session Initiation Protocol, 2002.6.

7 DOM Level 2 Core: W3C Document Object Model Level 2 Core Specification, November 2000.

8 Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns-Elements of Reusable Object Oriented Software, Addison-Wesley, 1995.

9 Bachmann F, Bass L. Managing Variability in Software Architectures. ACM SIGSOFT Software Engineering Notes, 2001,26:126－132.