XJoin:基于 SIP 应用服务器的电信应用框架①

XJoin: Telecommunication Framework Based on SIP Application Server

滕圣波　廖建新　王　纯　朱晓民　(北京邮电大学网络与交换技术国家重点实验室 北京 100876)

摘　要: 文章介绍一套基于构件的电信应用框架及其实现,适用于 SIP 应用服务器,实践和测试表明,该框架可以降低 NGN(Next Generation Network)网络的业务开发和维护成本。

关键词: 应用服务器 电信应用框架 构件 SIP NGN

1　引言

NGN(Next Generation Network)网络是业务驱动的网络,开放的业务开发和运营模式是其发展的趋势[1],因此作为增值业务主要平台的应用服务器在 NGN 网络中格外重要。根据网络接口的不同,NGN 网络中的应用服务器可分为 SIP(Session Initiation Protocol)应用服务器和 Parlay 应用服务器两类,本文所讨论的应用服务器指 SIP 应用服务器。

NGN 网络中的业务层出不穷且生命周期短,需要尽可能的简化业务开发。而现有的 SIP 应用服务器都未提供单独的业务开发框架,目前支持的业务开发方式都是直接基于开放的应用服务器底层 API(Application Programming Interface)以及脚本配置,需要业务开发者熟悉这些下层 API 以及通信协议,且需要业务开发者编写几乎全部的业务逻辑,未能很好的满足快速开发的需求。如今 Web 领域的应用框架已经逐渐成熟,WebWork 就是一个事件驱动的比较成功的 Web 应用框架[2],文献[3]则提供了在 J2EE(Java 2 platform enterprise edition)应用服务器上集成 Web 容器框架的较为理想的实现方式。借鉴 Web 的成功经验,开发一套电信领域的适合各种业务的应用框架,成为迫切的需求。北京邮电大学网络与交换技术国家重点实验室网络智能研究中心自主研发并实现了基于 SIP 应用服务器的电信应用框架 XJoin,为业务的快速开发以及代码重用提供了有效的支持。

2　现有业务开发方式分析

应用服务器是 NGN 网络中进行业务开发的主要平台。目前 JCP(Java Community Process)共推出了两套应用服务器规范,JAi(Java API for Integrated Networks) SLEE(Service Logic Execution Environment)以及 SIP Servlet。

JAiSLEE1.1(JSR 240)是一个获得 JCP 批准的容器标准[4],该标准类似于 EJB(Enterprise Java Bean),只是专门用于各种面向事件的应用程序。JSLEE 提出了 SBB(Service Building Block)的概念,SBB 就是单个或多个功能组件。如果业务开发者直接采用 JAiSLEE 1.1 的应用服务器开发业务,需要开发多个包含业务逻辑的 SBB,然后通过复杂的 XML(Extensible Markup Language)配置脚本将这些 SBB 加载到应用服务器中,组成一个完整的业务。开发 SBB 的具体办法是继承 SBB 抽象类并实现特定的抽象方法。

SIP Servlet 1.1(JSR 289)是对 HTTP Servlet APIs 和容器模型的扩展,也是一个容器标准[5],专用于 SIP 协议[6],特定于 Java 语言。BEA 公司的 WLSS(WebLogic SIP Server)就是支持 SIP Servlet 的应用服务器。如果业务开发者直接采用 SIP Servlet 标准的应用服务器开发业务,就类似在 HTTP Servlet 容器(比如 Tomcat)上直接开发业务,通过开发一系列的 SIP Servlets 来处理 SIP 消息流和请求。具体开发办法为继承 SipServlet 接口并实现 void service(ServletRequest request, ServletRe-

① 基金项目:国家杰出青年科学基金(No. 60525110);国家 973 计划项目(No. 2007CB307100,2007CB307103);新世纪优秀人才支持计划(No. NCET-04-0111);电子信息产业发展基金项目(基于 3G 的移动业务应用系统)

sponse response)等方法。

以上两种规范定义的业务开发方式都有共同的缺点,就是业务开发者的开发量比较大,且开发的业务代码不能重用。两种规范都不含任务的业务逻辑,业务开发者需要自行开发全部的业务逻辑。开发业务逻辑的过程也比较复杂,两种规范都提供了一系列的 API,业务开发者需要熟悉相关规范以调用这些 API。"不要重复发明轮子",已经成了软件界的一句经典名言,而现有的基于应用服务器规范的业务开发,却在不停地"重复发明轮子"。

3 电信应用框架研究

3.1 概念的提出

框架是一个骨架,它封装了某领域内处理流程的控制逻辑,所以说框架是一个半成品的应用。框架分为两种,通用框架和应用框架[7]。通用框架所解决的是所有类型的应用都关心的"普遍"问题,比如负责数据持久化的 ORM(Object Relational Mapping)框架,而应用框架解决的是某一特定类型的应用关心的问题,比如 Web 框架。应用框架需要针对特定领域的特点进行设计,才能发挥其优势。遗憾的是,目前尚未有专用于开发 NGN 网络中电信业务的应用框架。

我们把为基于 SIP 应用服务器的电信业务专门设计的,包含了核心业务逻辑与通用工具的应用框架,称为电信应用框架。电信应用框架介于应用服务器和业务逻辑之间。应用服务器将事件报给电信应用框架,电信应用框架负责执行业务逻辑。业务逻辑,电信应用框架,应用服务器三者的关系如图 1 所示。

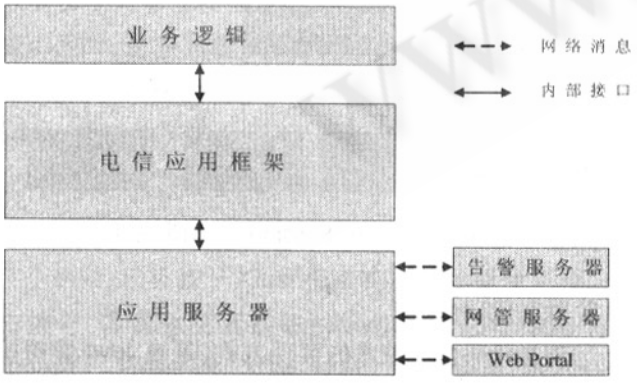

图 1 层次图

3.2 基于构件的电信应用框架架构

传统的面向对象框架存在着以下三点不足:框架的"过度增殖问题",许多硬编码被不断的加入到框架中,框架的维护越来越困难[8];"脆弱的基类"问题,框架开发者对框架中基类的扩展可能会影响到框架使用者所做的扩展;"隐式的体系结构"问题,框架使用者需要了解但无法了解隐藏在类的实现中的框架体系结构。因此,XJoin 电信应用框架采用了基于构件的软件架构,将各个模块做成构件的形式,通过用构件接口的调用来替代类方法的重载,将"继承"变为"组装",避免了传统的面向对象框架的问题[9]。

XJoin 由四部分构件组成:容器路由构件,业务加载构件,业务组件构件,协议相关构件。其体系结构如图 2 所示。

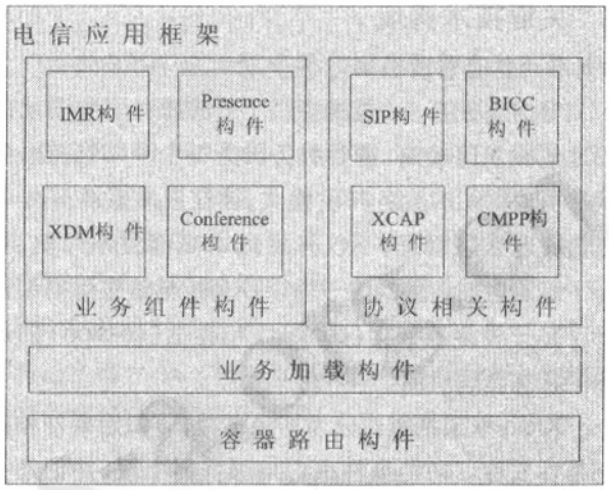

图 2 体系结构图

业务加载构件和容器路由构件是 XJoin 的核心构件,与业务逻辑和业务的承载协议无关。这两个构件的设计参考了文献[10]提出的框架应该满足的控制反转模式的要求,由业务加载构件在业务激活或者升级时读取业务配置文件,根据配置文件指定的规则创建特定的 Java Bean,并将这些 Java Bean 放入容器路由构件提供的一个控制反转容器中。之后,容器路由构件负责管理这些 Java Bean 的生命周期和约束关系,将应用服务器上报的事件路由给配置文件中指定的 Handler 进行处理,实现了控制反转。

协议相关构件提供了 NGN 网络中常用的业务承载协议的资源和能力,比如 SIP ,BiC (Bearer Independ-

ent Call Control Protocol)以及XCAP(XML Configuration Access Protocol)的消息收发的基本功能,并以可复用构件的形式供业务开发者直接在配置文件中配置使用这些功能。

业务组件构件的设计目的是针对几类特定的业务,抽象出公共的业务逻辑,把事件和消息收发进一步封装,使业务开发者不必关注采用什么协议,细节如何实现,只需要通过编写核心的配置文件以及少量的代码,就可以开发成功整个业务。比如,对于IMR, Presence, XDM以及Conference功能的支持。

协议相关构件和业务相关构件都以可热插拔的形式设计,只需要指定的jar包放入指定的文件夹,就可以实现指定构件的加载。

4 关键技术实现

4.1 通用状态管理机制

电信业务的一个主要特点,在于其对于会话状态管理的要求比较高,需要服务器为每个呼叫维持一个状态机。以往的业务开发模式,往往要求业务开发者自己实现状态管理,不仅开发成本高而且不可复用。为了满足电信业务的这一需求,XJoin电信应用框架设计了基于事件上下文(context)和会话(session)的通用的状态管理机制。

XJoin框架把事件分为初始事件,非初始事件和内部事件三种。初始事件由应用服务器上报,并创建一个上下文。非初始事件也由应用服务器上报,关联一个已经创建好的上下文。XJoin内部产生的事件称为内部事件,不创建也不关联上下文。

每一个业务在运行时对应一棵以Session作为节点的运行树。这棵运行树的根节点是一个特殊的Session,叫做Application,整个业务唯一。Session在收到初始事件时创建,在收到结束事件(属于非初始事件)时销毁。图2所示的为背靠背的用户代理业务(Back to Back User Agent)的运行树,此时同时有两组呼叫。在收到主叫拨号事件以后,Application创建一个一级Session,一级Session创建一个主叫Session以及一个被叫Session,双方挂机以后,对应该次呼叫的Session全部被销毁。

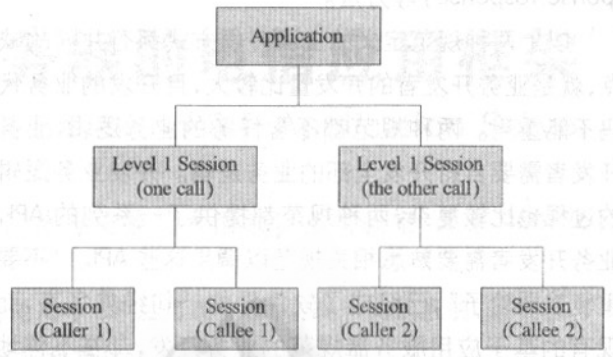

图3 B2BUA业务的运行树

每个Session节点可以关联一个或者多个上下文,表示接收该上下文上的所有事件,但是一个上下文只能被一个Session关联。每个Session提供了一个集合可以保存状态数据,该集合在Session的生存期间有效。业务开发者可以根据接收到的事件获得当前的Session,从而获得状态数据。这种状态管理机制为业务开发者管理会话状态提供了便利。

某些特殊业务,比如会议电话业务,需要把多方呼叫合并在一起进行状态处理,这就需要Session的合并。XJoin支持两个一级Session的合并,合并规则为:两个一级Session合并为一个一级Session,原有的两个一级Session的子Session作为合并后的一级Session的子Session。图3中的Level 1 Session(one call)和Level 1 Session(the other call)合并后的运行树如图4所示。

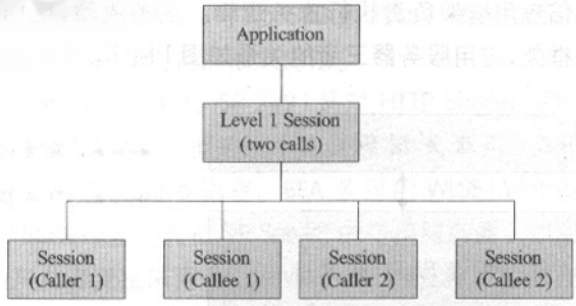

图4 两个一级Session的合并结果

4.2 动态代码生成

动态代码生成技术的实现方式,是在Java虚拟机启动以后,动态的生成Java字节码文件(.class文件),或者修改现有的Java字节码文件,然后把对应的

Java 类加载到虚拟机里面，这样就可以在业务运行过程中动态创建或者更改 Java 类。

XJoin 利用 Java 字节码工程库 ASM3.0 实现了动态代码生成技术[11]，以支持 Java 注解到 Java 代码的转换。比如，业务开发者需要发送一条 SIP 请求，如果要指定该请求的 from 头，只需要给自己编写的 Handler 类增加一条注解后的属性：

```
@from
public String fromHeader;
```

XJoin 会在业务激活时动态生成一个代理类，拦截该 HandeIr 的处理，在这个 Handler 执行结束以后取被注解的 fromHeader 的值作为要发送的 SIP 请求的 from 头。

类似的注解在 XJoin 框架里还有很多，这些注解的使用为业务开发者屏蔽了框架的实现细节，使业务开发者无需掌握底层 API 就能使用框架的资源和能力，简化了业务开发的难度，也减少了业务开发者的代码量。

4.3 Spring 扩展

XJoin 框架的业务加载构件利用了开源项目 Spring 框架对于 XML 文件的解析能力。Spring 2.0 增加了一个新的特性，就是允许开发者自定义 XML 标签和自定义命名空间，通过自己写解析代码来替换 Spring 默认的解析方式[12]。

XJoin 自定义了 16 个 XML 标签，比如 application，session，event，handler 等等，用于业务逻辑的搭建。XJoin 实现了这些标签的解析、类生成和加载，并在加载过程中按照类之间的约束关系执行预处理，从而使这些自定义标签具备了一定的智能，简化了业务的配置。同时，这些自定义的标签和扩展的代码可以和 Spring 2.0 无缝的结合，即可以在自定义的标签中使用 Spring 原有的标签。

5 应用和测试

目前，已经基于 XJoin 框架完成了多媒体彩铃、点击拨号、多媒体交互应答（Interactive Multimedia Response），Presence，XDM（XML Document Management）等多项业务的开发和测试，证明了该框架的通用性和易用性。

在 XJoin 实现以前，直接基于应用服务器进行了点击拨号业务的开发和测试。在 XJoin 实现以后，进行了点击拨号业务在 XJoin 框架上的移植和测试。前后两种开发方式的代码量对比如表 1 所示。

表 1 代码统计

	基于应用服务器的点击拨号	基于 XJoin 的点击拨号
代码行数（不含空行、注释）	4366	2159
配置文件行数	559	1656

从表 1 的对比结果可以看出，采用 XJoin 框架以后，业务开发的工作量有明显的减少。代码大约减少到之前的 1/2 左右，减少的大量的代码由框架提供的可复用构件来支持。同时，配置文件的行数明显增多，这一方面意味着开发配置文件的工作量增加，另一方面也说明了业务的可维护性和灵活性的增加。

对直接基于应用服务器的点击拨号和基于 XJoin 的点击拨号分别进行了性能测试，使用相同的测试环境：惠普 DL380 PC Server，Intel（R）Xeon（TM）CPU 3.00GHz（超线程），JDK1.5，为 Java 虚拟机分配 1G 的内存。使用相同的测试用例，测试时长 1200 分钟，40CAPS（Call Attempt Per Second）的稳定呼叫速率，每次呼叫的通话时间是 90 秒，由话务量公式 $A = C \times t$ 计算得话务量为 $40 \times 90 = 3600$ 爱尔兰。两种开发方式的性能测试结果如表 2 所示。

表 2 性能测试结果

	基于应用服务器的点击拨号	基于 XJoin 的点击拨号
呼损	0.20%	0.11%
CPU 占用率	72.5%	73.4%

从性能测试的结果看，采用 XJoin 之后，整体性能有较大改善。在同样的测试条件下，CPU 占用率没有明显变化，呼损比直接基于应用服务器降低了二分之一左右，这证明了 XJoin 框架是成功的。框架提供了统一的经过优化以后的业务构件，在一定程度上提高了软件的可靠性。

6 结论

XJoin 电信应用框架借鉴了 Web 框架的优点，在 SIP 应用服务器的基础上作了进一步的能力扩充，为业

务开发者提供了简单强大的 XML 配置方式以及丰富易用的通用资源，适合于开发 NGN 网络尤其是 3G 网络的各种增值业务，具有现实意义。虽然 XJoin 框架在一定程度上增加了业务开发人员学习框架的培训成本，但是相比于业务的开发维护成本的降低，以及应用程序稳定性的提高，这部分成本是值得的。

文献[13]指出了现有的基于构件的软件架构的两点不足，各个构件是固定和被动的，构件间的连线是"笨拙"的，并提出了一种新的基于移动 Agent 技术的构件软件框架，具有较强的网络环境的动态适应性。但是，该技术目前尚不成熟，XJoin 未加以采用。可以对移动 Agent 技术进行进一步研究，并利用该技术对 XJoin 框架加以完善和扩展，以进一步增加框架的灵活性。

参考文献

1 刘韵洁. 下一代网络的发展趋势——融合与开放. 电信科学，2005，2：1－6.

2 Patrick Lightbody, Jason Carreira, et al. WebWork in Action. http://www.opensymphony.com/webwork.

3 林泊，周明辉等. 一个 J2EE 应用服务器的 Web 容器集成框架. 软件学报，2006，17(5)：1198－1203.

4 Sun Microsystems, Inc. JSR 240: JAiSLEE 1.1 Specification, Proposed Final Draft. JAi，24 Aug, 2007

5 Nasir Khan, BEA Systems Inc. JSR 289: SIP Servlet Specification v1.1. 30 Jan, 2007.

6 Rosenberg, J., Schulzrinne, H., Camarillo, G.. SIP: Session Initiation Protocol. RFC 3261, June 2002.

7 Zhuweisky。NET 通信框架的设计、实现与应用. http://zhuweisky.cnblogs.com.

8 Barry D, Smaragdakis Y. Object－Oriented frameworks and product－lines. In: Donohoe P, ed. Proceedings of the lst Software Product Line Conference. Dordrecht: Kluwer，2000，227－247.

9 刘瑜，张世琨，王立福，杨芙清. 基于构件的软件框架与角色扩展形态研究. 软件学报，2003，14(8)：1365－1370.

10 Douglas C. Schmidt, Aniruddha Gokhale, Balachandran Natarajan. Leveraging application framework. ACM Queue, July/August 2004：66－75.

11 Eric Bruneton. ASM 3.0: A Java bytecode engineering library. 2007. http://download.forge.objectweb.org/asm/asm－guide.pdf

12 Rod Johnson, Juergen Hoeller, et al. Spring reference: Spring java/j2ee Application Framework. http://static.springframework.org/spring/docs/2.0.x/reference/index.html

13 吕建，张鸣，廖宇，陶先平. 基于移动 Agent 技术的构件软件框架研究. 软件学报，2000，11(8)：1018－1023.