学生应知信息知识

Java知识手册（五）

狄登峰 主编

目　　录

J2EEWeb 服务客户端质量报告

概要

本文实现了记录 J2EE（Java2 平台企业版）Web 服务的客户端响应次数的一个通用的结构。记录的响应次数是真实的客户端响应次数，所以它们实际上反映了用户对服务质量的看法。实验的样品是使用 SunONE（开放式网络环境）应用服务器和 IDE 建立起来的，但是这个方法很普通，很容易推广到其它 J2EE 实现上。

Web 服务正迅速的成为实现客户端-服务器系统的首选结构。它的优点是：企业可以正式的定义一组服务，然后生成通讯用的完整的客户端和服务器的代码库，从而简化新的客户端对合法的 Web 资源的访问。

但是，Web 服务在简化客户端-服务器系统的建立的同时，监控服务质量就变得很重要。假设有一个在用户的立场上提交处理的客户端应用程序。而企业事务通常要调用好几个 Web 服务：初始调用递交工作内容，接下来的调用检查实现，最终调用得出结果。一个调用就是一个特殊的 HTTP/SOAP(简单对象访问协议)交换。假设你是 IT 部门专门负责监控服务器装载和预测未来需要的工作人员，你必须得回答这个基本问题："我现在管理我的客户端怎样呢？对于将来的管理，我还需要什么东西？"

如果你只有 HTTP 日志的话，就很难回答上述问题了。客户端只关心事务的处理，但是因为每个事务包括几个 HTTP 请求，对于评估服务质量，你最多只能开发自定义数据采集软件，该软件可通过 HTTP 日志做出指示并建立用户事务处理的模型。就算是这样，你所拥有的信息仍然有限，因为它不能反映网络传送或者客户端应用程序的内务操作。

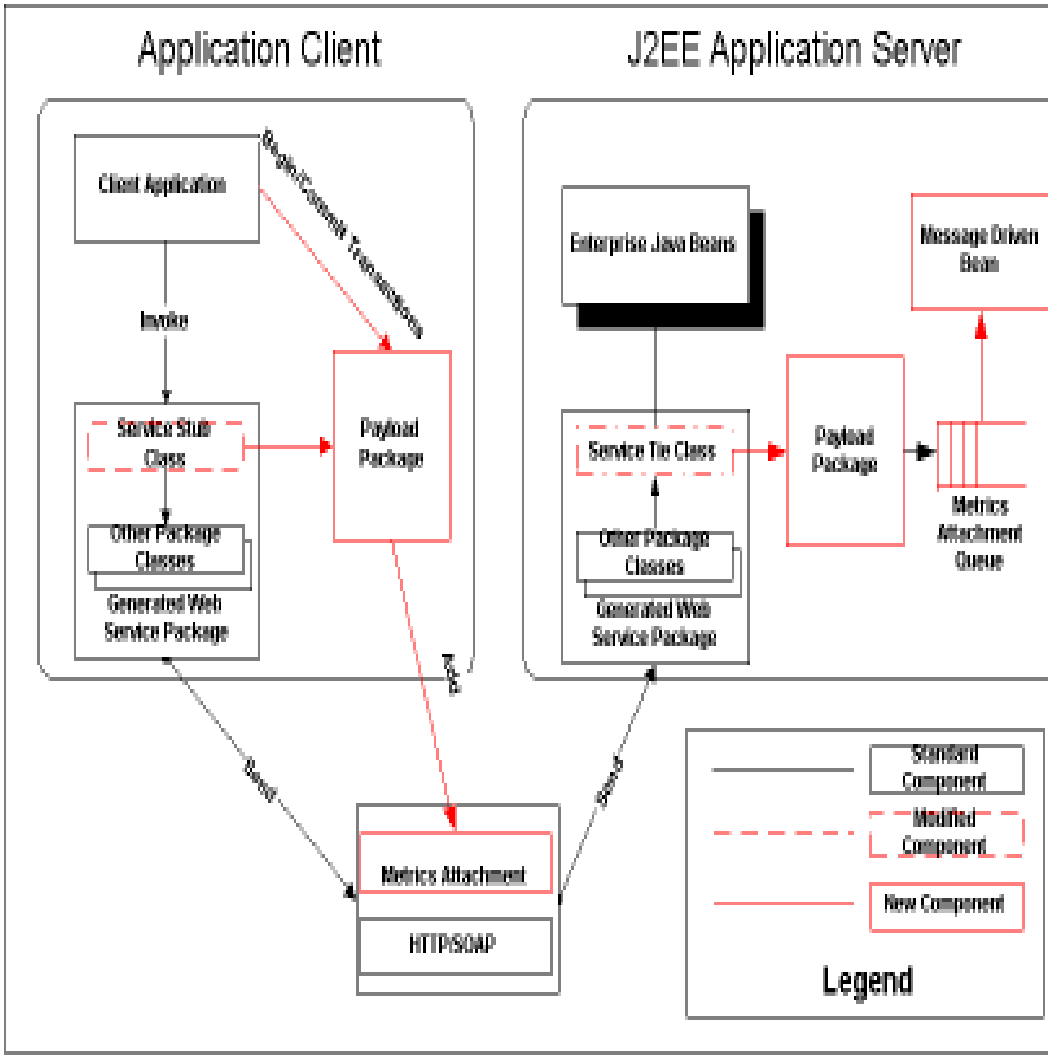
本文的中心思想是：事务服务质量用客户端评估最好。这儿采用的方法就是允许客户端记录实际的事务响应次数。客户端应用程序通过将响应时间报告添加到下一个弹出的事务处理请求上，从而上传响应时间报告给服务器。服务器取出这些附件并将他们排队储存和在线分析。

结构

基于客户端的频率记录结构的目标就是：记录用的下部构造必须是轻型的，它不但有利于实现运行的内务开销还可以减轻添加它到一个现有的实现的负担。我们也希望该结构对提供的服务没有限制，我们很想可以将服务添加到一个现有的、可以尽可能容易地使用 Web 服务的客户端-服务器系统上。

我们的结构的另一个目标是：企业应用程序自身的可靠性不要太差。我们将引入一些新的、容易做到的步骤到应用程序的处理工作流程中。而且我们可以保证这些新步骤中的任何故障都可以得到处理，我们不会因为不能将频率用于程序就让企业事务的处理失败

下图显示了一个典型的 J2EE(Java2 平台企业版)Web 服务的客户端-服务器应用程序。典型的组件用粗线标明；我们添加的新组件用于收集频率，它采用红线标明。



J2EEWebservices:Metrics-gatheringarchitecture

"J2EEApplicationServer"区域表示现有的服务器资源。他们是用来处理客户端请求的企业 JavaBeans(EJB)

组件。工具可自动的生成 Web 服务软件包。EJB 组件和相关的 Web 服务模块被当作 J2EE 应用程序配置到 J2EE 服务器上。当应用程序配置时，客户端可以通过调用程序 WSDL(Web 服务描述语言)服务来判断一个服务。WSDL 服务对程序提供的服务给出正式的定义。

"ApplicationClient"区域由程序组件和 Web 服务软件包组成。程序组件实现商业逻辑和用户界面。Web 服务软件包可自动地通过 WSDL 编译器和 J2EE 服务器程序提供的 WSDL 服务创建出来。

从理论上来说，整个客户端-服务器系统有两层。应用程序层在服务器端拥有 EJB 组件，在客户端拥有一个应用程序。Web 服务层有一个服务器实现和一个客户端实现，两者都可以自动产生。

典型地，用户的商业事务处理包括许多个服务期调用。第一个调用初始化事务，返回一个"handle"给客户端。后来的调用查询事务的完成--客户端使用句柄调用服务来检查事务是否得到处理。通常最后一个调用可得到完成的事务的状态。因此，一个商业模型，可在客户端程序内实现的商业模型，总是使得事务与低级别的服务器调用联系起来。

我们可以将收集频率的组件添加到我们的标准 J2EEWeb 服务结构上。上图中的 Payload 软件包在服务器和客户端都有配置。稍后我会详细的讨论这个软件包，但是从结构的意义上来讲,这个软件包提供了几个服务。例如：客户端程序可以使用 beginTransaction() 和

commitTransaction()来定界事务和记录次数。客户端 Web 服务软件包使用 Payload 软件包来连载次数报告给 SOAP 信息附件。服务器端的 Web 服务软件包使用 Payload 软件包将 SOAP 附件从引入的请求中剥离出来，并将它列队登记和报告。

这个实现中的系统操作很少因为客户端和服务器不交换任何新的通信--完成的事务的频率报告与下一个客户端请求一起运送。引入的唯一的新的处理就是客户端上的一些连载和服务器端排队等待的附件。整个实现很轻便，因为只要添加一行代码到每个程序 Web 方法上，并且代码还是一样的--如果 Web 方法的标记不变的话他也不会发生变化。

引入的最后一个组件就是信息驱动的 EJB 组件，它可读取连载的频率附件。典型的，这些附件将会记录到一个数据库中所以企业可以保存事务服务质量的历史纪录。企业可以使用这个数据库将真实的事务响应次数与服务器资源的使用联系起来，从而可以鉴定性的判断出哪个服务器组件才是关键的服务瓶颈。因为附件是排队等待的，所以频率读取器 EJB 组件应该在不同的 J2EE 服务器上运行，我们不希望使用商业 EJB 组件纪录的频率附件竞争资源。

实现

在这个部分，我将会告诉你如何将频率代码集成到简单的 J2EE 客户端-服务程序上。所有的代码都可从 Resources 处下载；下面部分将告诉你如何使用

SunONE(开放网络环境)应用程序框架来建立和运行频率代码。

应用服务原型

在本例中，服务器应用包括一个单一的会话期 bean。这无损于应用程序的一般性因为内部的服务 EJB 设计不影响记录频率的结构。即使服务器中有许多不同的 EJB 组件也可以使用同样的频率方法。

XactBeanEJB 暴露了三个商业方法：submitWork()、checkWork()、和 getResult()。每个方法都各有不同。客户端应用使用这三个方法来模拟一个客户端，该客户端可提交多个 Web 请求以执行用户的事务。

代表服务器应用的会话期 bean 显示如下：

```
packageTransactionProcessor;

importjavax.ejb.*;

importjava.rmi.server.*;

importjava.util.*;

/**

*CreatedMay19,200310:07:39PM

*CodegeneratedbytheSunONEStudioEJBBuilder
```

```
*@authorBrianConnollybrian@ideajungle.com
*/
publicclassXactBeanimplementsjavax.ejb.SessionBean{
privatejavax.ejb.SessionContextcontext;
privateintmRandom;
/**
*@seejavax.ejb.SessionBean#setSessionContext(javax.ejb.SessionCon
*/
publicvoidsetSessionContext(javax.ejb.SessionContextaContext){
context=aContext;
}
/**
*@seejavax.ejb.SessionBean#ejbActivate()
*/
publicvoidejbActivate(){
}
```

```
/**
*@seejavax.ejb.SessionBean#ejbPassivate()
*/
publicvoidejbPassivate(){
}
/**
*@seejavax.ejb.SessionBean#ejbRemove()
*/
publicvoidejbRemove(){
}
/**
*Seesection7.10.3oftheEJB2.0specification
*/
publicvoidejbCreate(){
Randomr=newRandom();
mRandom=r.nextInt(10000);
```

```
}
publicjava.lang.StringSubmitWork(java.lang.StringWork){
returnnewInteger(mRandom).toString();
}
publicbooleanCheckWork(java.lang.StringXact){
returntrue;
}
publicjava.lang.StringGetResult(java.lang.StringXact){
returnnewInteger(mRandom).toString();
}
}
```

这三个方法可模拟客户端事务处理。在 submitWork() **中，我们产生了一个作为任意号码使用的句柄--实际上它是唯一的事务标记符。**checkWork()**总是返回**"真"**。在实际的系统中，客户端传递事务标记符，所以** checkWork() **使用一个回溯的事务管理器检查事务是否已经完成。类似的，在实际的系统中，**getResult()**返回一个复杂的事务完成纪录。**

服务器 Web **服务软件包**

服务器 Web **服务软件包可自动生成。在** SunONEStudio **中，** Web **模块的创建只要选择一组** EJBJava **方法即可，并且** Web **服务软件包的类可由** Web **模块创建。**

该软件包包含许多类和接口。这里最关键的一个就是＜ServiceName**＞**ServantInterface_Tie **类，在这个类中服务名就是＜**ServiceName**＞。类** Tie **是** Web **服务模块最上面的堆栈；它将引入的服务调用绑定到创建它的** EJB **组件上。我们只需修改类** Tie **就可以添加次数纪录。**

Tie **包括许多方法，但是我们只需修改与** EJB **商业方法** invoke_＜X＞**关联的那一个方法。在方法** invoke_＜X＞**中，＜**X**＞表示** EJB **商业方法的名称。我们添加一个** importPayload.*;**到类** Tie **上，并对每个商业方法作了一个小小的修改。让我们看看下面的方法** invoke_SubmitWork()**：**

```
/*
*Thismethoddoestheactualmethodinvocationforoperation:SubmitWork
*/
privatevoidinvoke_SubmitWork(StreamingHandlerStatestate)throwsEx
TransactionService.XactServiceGenServer.
```

```
XactServiceServantInterface_SubmitWork_RequestStruct
myXactServiceServantInterface_SubmitWork_RequestStruct=null;
ObjectmyXactServiceServantInterface_SubmitWork_RequestStructObj
state.getRequest().getBody().getValue();
/*Lineaddedtogeneratedmethod:*/
Serializer.queueFirstAttachmentText(state.getMessageContext());
if(myXactServiceServantInterface_SubmitWork_RequestStructObj
instanceofSOAPDeserializationState){
myXactServiceServantInterface_SubmitWork_RequestStruct=
(TransactionService.XactServiceGenServer.
XactServiceServantInterface_SubmitWork_RequestStruct)
((SOAPDeserializationState)
myXactServiceServantInterface_SubmitWork_RequestStructObj)
.getInstance();
}else{
myXactServiceServantInterface_SubmitWork_RequestStruct=
```

```
(TransactionService.XactServiceGenServer.
XactServiceServantInterface_SubmitWork_RequestStruct)
myXactServiceServantInterface_SubmitWork_RequestStructObj;
}
java.lang.Stringresult=
((TransactionService.XactServiceGenServer.XactServiceServantInterfa
getTarget()).SubmitWork
(myXactServiceServantInterface_SubmitWork_RequestStruct.getString
TransactionService.XactServiceGenServer.
XactServiceServantInterface_SubmitWork_ResponseStruct
myXactServiceServantInterface_SubmitWork_ResponseStruct=
newTransactionService.XactServiceGenServer
.XactServiceServantInterface_SubmitWork_ResponseStruct();
SOAPHeaderBlockInfoheaderInfo;
myXactServiceServantInterface_SubmitWork_ResponseStruct.setResul
SOAPBlockInfobodyBlock=newSOAPBlockInfo
```

(ns1_SubmitWork_SubmitWorkResponse_QNAME);

bodyBlock.setValue(myXactServiceServantInterface_SubmitWork_Res

bodyBlock.setSerializer

(myXactServiceServantInterface_SubmitWork_ResponseStruct_SOAPS

state.getResponse().setBody(bodyBlock);

}

我们添加了一个单行到 invoke_SubmitWork()**上：**

Serializer.queueFirstAttachmentText(state.getMessageContext());

getMessageContext() **返回实现接口** javax.xml.rpc.handler.soap.SOAPMessageContext **的对象。该对象提供对当前** SOAP **信息的访问。我们传递实现接口** SOAPMessageContext **的对象到** Payload.Serializer **中的一个静态方法上。该静态方法从第一个信息附件中获取** XML **字符串并将它排队等待次数处理器** EJB **组件的调用。**

我们对每个 invoke_＜X＞**方法作了同样的修改。**

Payload **软件包**

Payload 软件包可用于客户端，也可用于服务器。它包含三个类：ClientReport、CurrentReport、和 Serializer。

ClientReport 表示一个客户端次数报告：

```
packagePayload;

importjava.io.*;

importjava.util.*;

/**

*

*@authorBrianConnollyBrian@ideajungle.com

*/

publicclassClientReportimplementsSerializable{

publicDateclientStartDateTime;

publicDateserverStartDateTime;

publiclongclientElapsedMS;

publicStringtype;

publicStringstatus;

publicStringtransactionID;
```

```
publicStringclientID;

//DefaultpublicconstructorforWSDL

publicClientReport(){

}

/*

...Get,setpropertymethodsarenotshown

*/
```

在上述代码中，clientStartDateTime **记录客户端初始化事务的时间。**serverStartDateTime **当前没有使用；它的用途是保存事务的服务器开始时间以便事务次数可与服务器资源使用的随时间的变化关联起来。**

ClientElapsedMS **是我们记录的主要工具：从客户端开始记录新事务到它收到最后一个** Web **服务调用的结果为止这段时间的毫秒数。**

Type **允许客户端使用类型特征化事务。通常，事物系统提供许多种类型的事务。我们期望某些类型对于服务器来说相对容易一些，某些类型相对难一些，这样当我们分析响应次数和测量服务器资源时我们能够将他们辨别出来。**