J2EE 资源集成的性能优化研究*)

郑 蕾[1,3] 官荷卿[1,2,3] 陈宁江[1,2,3]
(中国科学院软件研究所 软件工程技术研发中心 北京100080)[1]
(中国科学院软件研究所 计算机科学重点实验室 北京100080)[2]
(中国科学院研究生院 北京100039)[3]

摘 要 J2EE 应用服务器基于 JCA 规范为企业资源系统提供了一个统一的资源集成解决方案。在 JCA 框架下,应用服务器提供连接管理、事务管理及安全管理等服务,这些服务是影响系统性能的重要因素。本文主要研究了应用服务器端在支持资源集成中与性能相关的几个关键问题,包括连接池管理和事务管理,提出了行之有效的优化策略。本文的工作已经应用到中科院软件所自主开发的 J2EE 应用服务器 OnceAS 中,取得了很好的效果。

关键词 JCA,资源集成,连接池,事务管理,J2EE 应用服务器

Research on Performance Optimization in J2EE Resource Integration

ZHENG Lei[1,3] GUAN He-Qing[1,2,3] CHEN Ning-Jiang[1,2,3]
(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100080)[1]
(Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080)[2]
(Graduate School of the Chinese Academy of Sciences, Beijing 100039)[3]

Abstract Based on JCA specification, J2EE Application Server gives a standard way to integrate enterprise resources. Under the connector architecture, the application server provides different services, including transactions, security and connection management for resource adapters, and these services are important to the system performance. This paper studied several problems in application server's resource integration implementation, including connection management and transaction management. Some available optimizations and the implementations are given. These optimizations have been successfully employed in the J2EE Application Server OnceAS.

Keywords JCA, Resource integration, Connection pool, Transaction management, J2EE application server

1 引言

三层/多层的分布式体系结构已成了当今互联网时代企业应用的主流。三层结构中的数据层主要包括数据库、ERP、CRM 系统等,它们统称为企业信息系统(Enterprise Information Systems,EIS)。EIS 为企业提供了基础的信息设施和服务,承担着数据和资源管理的核心功能。由于大部分 EIS 是在不同的时期或者面向不同需求开发的,它们之间的互连以及与当前的主流应用系统之间的互连都存在着困难,因此如何对 EIS 资源进行集成和管理已成为目前企业应用中的关键问题。SUN 公司的 JCA(J2EE Connector Architecture)1.0[1]规范作为 J2EE 1.3[2]规范的一部分为异构 EIS 资源的集成提供了一个标准的体系结构,遵循 J2EE 1.3规范的应用服务器必须提供合适的环境并支持必要的 JCA 功能,包括缓冲池、事务和集成的安全机制等。资源集成的性能问题对 J2EE 应用服务器以及应用系统的正确运行和性能具有重要影响。本文结合中科院软件所自主研制的 OnceAS 应用服务器[3]的工作,在分析资源集成的性能因素的基础上,在连接管理、事务管理等方面提出了性能优化的方案。第2节简要介绍了 J2EE 连接器结构及实现存在的一些问题;第3节对现有连接池结构的设计优化;第4节讨论了本文在事务管理中对规范的扩展,最后小结全文。

2 JCA 概述

JCA 是为了简化 J2EE 服务器和 EIS 之间的集成而提出的一种规范。它的实现依赖于 EIS 供应商提供的资源适配器端和应用服务器提供商提供的应用服务器端对规范的共同支持,本文的研究主要针对应用服务器端的实现。

如图1所示,JCA 规范主要定义了两部分内容:Service Provider Interface(简称 SPI)和 Common Client Interface(简称 CCI)。SPI 定义了资源适配器(Resource Adapter)和应用服务器之间的系统级协议,它包括连接管理协议、事务管理协议和安全管理协议。对于应用服务器开发商来说,这些系统级协议实现的好坏直接影响到整个服务器的性能。在实际应用中对连接和事务的管理是最常用的,对应用服务器性能的影响也最大。连接的创建和释放是一项耗费资源的操作,连接池是连接管理中一种常见的优化手段,本文针对已有的连接池调度所存在的问题,提出了新的优化方法。JCA 规范在涉及事务管理[4]性能的一些技术问题上未做详细规定。例如,在连接共享的情况下,应用组件可能会根据需要缓存连接句柄,按照规范,这种缓存连接句柄需要和实际的物理连接相关联,从而造成资源浪费;又如,在一个事务作用域内可能会获取一些并未使用到的资源连接,而这种资源连接的获取与注册又是很耗时的操作。因此,我们在 JCA 事务管理协议的基础上进行了扩展,使得对事务的管理更加灵活高效。

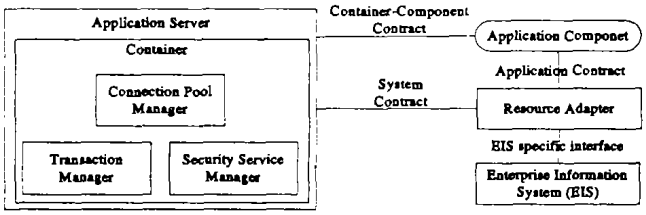

图1 JCA 体系结构

3 连接池的优化

应用服务器在获取资源连接时，每个新的资源连接请求都会导致很大的开销，通过使用连接池可以有效提高系统的性能和资源利用率。JCA 规范没有规定具体怎样实现连接池。在 OnceAS 中，我们设计并实现的连接池结构如图2所示。

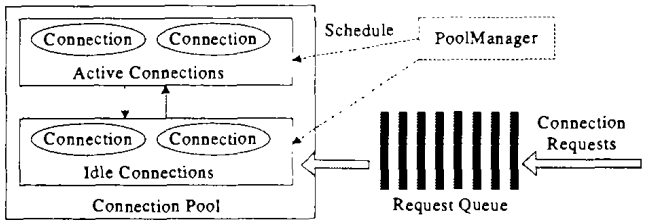

图2 连接池的结构

连接池中的连接分为两种：活动连接（Active Connection）是正在被应用程序使用的连接，闲置连接（Idle Connnetion）是当前没有被使用的连接。连接池管理器处理应用程序对连接的请求，并根据实际情况在这两种连接之间进行调度，适时创建和释放连接。

3.1 应用程序连接请求队列

在连接池的实现中，通常使用 Java 自带的向量（Vector）、数组链表（ArrayList）、栈（Stack）作为存储和管理应用程序连接请求的数据结构。这存在两个问题：

1）考虑到连接池的效率，池管理器（例如栈作为容器的情况）可能会在一个连接可用时，将该连接分配给后到达的连接请求，从而造成先到达的请求反而后得到连接的不公平现象。

2）在保证应用程序连接请求的先到先得的公平性的前提下，池管理器（例如向量作为容器的情况）通常使用标准 add 方法将连接请求放入容器，当池中有连接可用时使用标准 remove 方法取出连接请求并和可用连接进行匹配。这种实现方式会引起大量 Java 对象的整体拷贝和相应的不可预知的垃圾收集，降低系统的性能。

针对这两个问题，我们使用自己实现的请求队列（Request Queue）作为存储和管理应用程序连接请求的容器。这是一种循环队列结构，它充分利用了队列结构的特性："先进先出"（FIFO）的存储结构使得数据元素只能从队尾进入，从队首取出，保证了连接请求先到先得的公平性；首尾维护指针的使用使得队列中的元素可以任意增减，且又不会带来 Java 对象的整体拷贝，数据元素的次序也不会改变。这种请求队列是我们在分析了 Java 本身容器的实现方法的基础上以同样的方法实现的，从而保证了它的性能并不低于传统容器。这虽然只是一点简单的改进，却明显改善了连接池的性能。

3.2 基于 Reactor 模式的优化

应用服务器为每个可用的资源维护一个对应的连接池。每个连接池都会自动调整池中的连接数，尽量释放长期不用的连接，以便将对应的系统资源（内存、网络连接、线程等）释放；同时也保持一定的连接数，以便在连接使用的高峰期降低创建和释放连接的开销。从单个连接池的角度看，这种策略是有效的。

在很多应用场景下，连接池要频繁地创建大量连接，并在一段时间的闲置后大量释放连接。这样的例子包括证券结算系统、网络论坛等，它们的共同特点在于对资源的连接请求的数量在一个相对较短的时间内变化很大。

连接池中的每个连接对应一个系统线程，连接池管理器创建和释放连接的操作都属于同步阻塞式系统调用。在同步阻塞情况下，线程将一直阻塞，直到连接创建或释放的操作彻底完成。对于同时为多个资源维护多个连接池的应用服务器而言，大量连接池中的线程都处于同步阻塞状态带来了不必要的性能开销，这些等待连接创建和释放的线程造成了系统性能的瓶颈。为了减轻大量线程同步阻塞对系统性能的影响，我们利用 Reactor 设计模式[5]的思想，对连接池进行优化。

Reactor 模式的核心是选择器（Selector）和通道（Channel）两个类。通道表示服务器和客户机之间的一种通信机制，在此我们将它视为资源连接，它可以被异步地关闭和中断。选择器类是通道的多路复用器，它将可用的资源连接多路复用并将它们分派到各自的请求应用程序。Reactor 模式实现的序列图如图3所示。

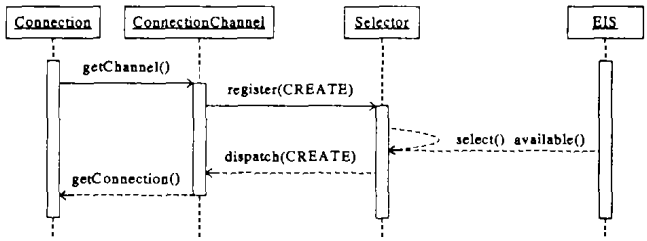

图3 Reactor 模式实现的序列图

Reactor 模式在处理这种大量同步阻塞线程问题上的优势体现在：所有资源连接通道被注册到一个选择器对象上，由选择器等待着任何通道的活动，而相应的资源连接线程不必在此阻塞等待。当某个资源连接可用时，选择器对象将会发出通知，将连接分派到相应的注册线程，从而实现异步非阻塞的资源连接的创建与释放。这种方式只用一个线程进行多个连接的创建和释放，而不是通常的每个资源连接对应一个线程。尽管选择器本身不能对连接进行操作，但它可以监听连接可用的事件并越过较慢的连接通道，从而减少了多个线程的同步阻塞，提高了系统性能。

3.3 连接池的性能优化分析

我们将从理论上分析 Reactor 模式的应用对连接池性能的影响。在不考虑一些重要的环境因素（如线程同步、上下文切换、内存换页和缓冲区大小等）的情况下，我们对需要考虑的因素做了如下假定：

（1）服务器和应用程序之间共有 N 个并发的资源连接请求；

（2）对于每个资源连接请求，服务器和应用程序之间传输的消息（执行数据库查询、修改等）大小为 S 字节；

（3）网络连接的应用程序（客户端）带宽是 B 字节/秒；

（4）对于每个资源连接请求，服务器执行请求分析等操作的时间是常量 C 秒；

（5）磁盘 I/O 的速度总是快于网络 I/O 的速度，服务器的带宽总是大于应用程序的带宽之和，并且服务器 CPU 始终未满载。因此不考虑服务器端带宽、缓冲和磁盘速度等影响因素。

对于通常的没有使用 Reactor 模式的连接池，假设最多可以支持 X 个并发线程（对应于最多可容纳 X 个资源连接），由于同一时刻最多只能处理 X 个请求，服务器的总处理时间 T_{old} 依赖于池中资源连接的数量，如式（1）所示。

$$T_{old}=\frac{N\times S}{B\times \min(N,X)}+N\times C \qquad (1)$$

对于使用了 Reactor 模式的连接池，由于其支持任意数量的资源连接，因此服务器的总处理时间 T_{new} 不再依赖于系统最大并发线程的数量 X，而直接依赖于应用程序的带宽 B、消息大小 S 和常量 C，如式（2）所示。

$$T_{new}=\frac{S}{B}+N\times C \tag{2}$$

两种连接池的性能对比可以简单比较服务器的总处理时间，即 T_{new} 和 T_{old} 的比值 K，如式(3)所示。

$$K=\frac{T_{new}}{T_{old}}=\frac{\frac{S}{B}+N\times C}{\frac{N\times S}{B\times\min(N,X)}+N\times C} \tag{3}$$

通常情况下，在实际应用中可以将消息大小 S、应用程序的带宽 B、服务器能够支持的最大线程(资源连接)数量 X 与 C 值一样视为常数。当 N 趋向无穷大时，K 趋向于一个极限值，如式(4)所示。

$$\lim_{N\to\infty}K=\lim_{N\to\infty}\frac{\frac{S}{B}+N\times C}{\frac{N\times S}{B\times\min(N,X)}+N\times C}=\frac{C}{\frac{S}{B\times X}+C} \tag{4}$$

因此，在需要处理很多资源连接的情况下，除了资源连接的最大数量和常量开销，连接的消息传输的时长 S/B 对 K 值具有非常重要的影响。连接传输消息持续的时间越长(通过连接进行的操作越复杂)，K 值越小，使用 Reactor 模式的连接池的优势也就越明显。表1给出了在系统支持的最大线程(资源连接)数量 $X=100$ 和常量开销 $C=0.01$ 秒的条件下，对 S 和 B 的不同取值得到的极限的 K 和 $1/K$ 的值。

表1 消息大小及应用程序带宽对 K 值的影响

S(字节)	B(字节/秒)	K	1/K
1000	8000	0.889	1.125
10000	8000	0.444	2.250
100000	8000	0.074	13.500
1000000	8000	0.008	126.000
1000	10000000	1.000	1.000
10000	10000000	0.999	1.001
100000	10000000	0.990	1.010
1000000	10000000	0.909	1.100

从表1中可以看出，当 $S=1$ 兆字节，$B=8$ 千字节/秒时，使用 Reactor 模式的连接池要比通常的连接池快125倍。这意味着如果应用程序使用资源连接的时间很长的话，基于 Reactor 模式的连接池将表现出巨大的优势。同时，当使用资源连接的时间较短时，如果通过连接进行较复杂的操作，则基于 Reactor 模式的连接池会表现出一定的优势；如果通过连接进行的操作很简单，则优势不明显。

4 资源连接的事务管理

应用组件对 EIS 资源的连接操作往往是处在某一事务范畴之内的。应用服务器对资源连接的事务管理分为两种情况：如果一个应用组件的事务中包含了对多个资源连接的操作，则使用服务器端的事务管理器通过调用资源适配器对连接进行事务管理；如果一个事务只涉及到对某一资源的连接，则使用资源适配器内部提供的本地事务接口对连接进行事务管理。

4.1 动态连接关联机制

在实际的事务处理过程中经常会碰到这样的情况：两个应用组件 A 和 B 都使用事务。组件 A 的方法打开一个数据库连接，并对其中某个表进行了更新，在关闭连接前组件 A 调用了组件 B 的某个方法。组件 B 打开同一个数据库的连接，也对同一个表进行了更新。按照普通的连接处理方式，这两个连接句柄关联的是两个不同的物理连接，不但占用系统资源，而且所执行的数据库操作将会造成无法恢复的死锁。为了解决这一问题，JCA 使用连接共享机制(Connection Sharing)通过在 getConnection 方法时传递一个描述连接属性的 ConnectionSpec 对象让组件 A 和 B 所获得的连接句柄实际上都指向同一个物理连接(ManagedConnection，MC)对象。

对于这种共享的物理连接对象，应用组件可能会将连接对象保存在其实例变量中(即缓存连接句柄)，从而避免多次 JNDI[6] 查找所带来的开销，并在多个事务间重复使用。这种连接缓存机制有两个问题：首先，要使这些连接句柄在执行了 ManagedConnection 的 cleanup 方法之后不会变得无效，应用服务器就必须保留一个有效的 ManagedConnection 对象，使它与这些未关闭的连接句柄相关联，这将消耗一个后台资源的物理连接。如果组件在使用一次之后就很少被用到，那么系统资源将会被浪费。其次，应用服务器不知道将在哪个打开的连接上执行交互，所以它必须与所有缓存的连接句柄相关联，并为每个句柄都跟踪相关必要的信息。因此，管理这些可共享的资源连接的缓存句柄将导致应用服务器的极大开销。我们使用一种动态连接关联机制(Dynamic Connection Association Optimization)对该问题进行优化，使与连接句柄的关联只有在需要的时候才发生。

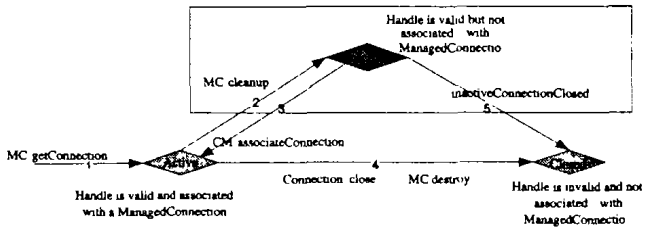

图4 动态连接句柄关联机制

如图4所示，我们为连接句柄引入一种 Inactive 状态，在这种状态下连接句柄是有效的，但与实际的物理连接没有任何关联；并为连接管理器(Connection Manager，CM)接口增加一个 associateConnection 方法。当应用程序需要与处在 Inactive 状态的连接执行交互时，资源适配器或连接句柄本身调用该方法把连接句柄激活。同时，在连接管理器接口上增加 inactiveConnectionClosed 方法关闭 Inactive 状态的连接句柄。具体的状态转换过程如下所述：

1)连接管理器通过调用 MC.getConnection 方法为应用组件获得一个活动的(Active)连接句柄对象。

2)应用组件使用该连接句柄操作后台的 EIS 资源。操作结束后，为了在下一个事务中重复使用该连接句柄，应用组件并不将其关闭，而只是通知应用服务器调用 MC.cleanup 方法释放实际的物理连接资源。此时该连接句柄仍可继续使用，但并不关联任何物理资源，即处于非活动(Inactive)状态。

3)在另一事务中，应用组件需要再次操作同一个 EIS 资源，此时，它并不需要通过重新查找 JNDI 等一系列过程获得相关的连接句柄，而是直接使用这个处于非活动状态的连接句柄。同时，通过服务器内部的具体实现，应用组件调用连接管理器的 CM. associateConnection 方法，将这个连接句柄再次与一个实际的物理连接相关联，于是该连接句柄又从非活动状态变成了活动状态。

4)应用服务器如果确定该连接句柄不会再被使用到，并想将其释放，则在活动状态下调用 Connection.close 方法即可将该连接从有效状态变为关闭状态，此时，该连接句柄既不能被再次使用也不与任何实际的物理连接相关。

5)在非活动状态下调用 CM.inactiveConnectionClosed

(下转第235页)

过一个 UML 状态图的实例，给出了本文所描述的6条测试准则的具体应用。最后还讨论了这6条测试准则之间的包含关系。

众所周知，数据流覆盖主要是用在基于程序的测试中。在未来，希望能把数据流覆盖也应用到基于状态图的测试中。这样就能使基于状态图的测试准则更加充分。

参 考 文 献

1 Zhu H, Hall P AV, May J HR. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 1997, 29(4): 366～427

2 Haworth B, Kirsopp C, Roper M, et al. Towards the development of adequacy criteria for object-oriented systems. In: Proc. of the 5th European Conf. on Software Testing Analysis and Review, Edinburgh, Scotland, Nov. 1997. 417～427

3 Haworth B. Adequacy criteria for object testing. In: Proc. of the 2nd Intl. Software Quality Week Europe 1998, Brussels, Belgium, Nov. 1998

4 Offutt A J, Xiong Y, Liu S. Criteria for Generating Specification-Based Tests. In: Proc. of 5th IEEE Intl. Conf. on Engineering of Complex Computer Systems (ICECCS'99), Las Vegas, Nevada, USA, Oct. 1999. 119～129

5 Offutt A J, Abdurazik A. Generating tests from UML specifications. In: Proc. of the Second IEEE Intl. Conf. on the Unified Modeling Language (UML99), Fort Collins, CO, IEEE Computer Society Press, 1999. 416～429

6 Abdurazik A, et al. Evaluation of Three Specification-based Testing Criteria. In: Sixth IEEE Intl. Conf. on Engineering of Complex Computer Systems (ICECCS '00), Tokyo, Japan, Sep. 2000

7 Ammann P E, Black P E. A Specification-Based Coverage Metric to Evaluate Test Sets. International Journal of Reliability, Quality and Safety Engineering, World Scientific Publishing, Singapore, 2001, 8(4): 275～300

8 Binder R V. Testing Object-Oriented System: Models, Patterns, and Tools. Boston: Addison Wesley Longman, Inc. 2000

9 Kim Y G, Hong H S, Bae D H, Cha S D. Test Cases Generation from UML State Diagram, IEEE Proceedings - Software, 1999, 146(4): 187～192

10 郑人杰，殷人昆，陶永雷. 实用软件工程. 北京：清华大学出版社(第二版)，1997

(上接第208页)

方法将由无效状态变为关闭状态，释放资源。此时，该连接句柄既不能被再次使用也不与任何实际的物理连接相关。

4.2 连接事件通知机制及其扩展

在 JCA 规范中，应用服务器通过连接事件通知机制进行连接及事务管理，其中定义了五种类型的连接事件，包括：CONNECTION_CLOSED、CONNECTION_ERROR_OCCURRED、LOCAL_TRANSACTION_STARTED、LOCAL_TRANSACTION_COMMITED 和 LOCAL_TRANSACTION_ROLLEDBACK。

应用服务器为每个物理连接对象注册一个连接事件监听器(ConnectionEventListener)，每当监听到连接事件发生时就根据所发生的事件进行相应处理。例如，如果应用组件关闭了某一连接句柄，这一事件以资源适配器所定义的方式传递给它所关联的物理连接对象，该对象使用监听器的 connectionClosed 方法通知所有注册的应用组件，并传递一个 CONNECTION_CLOSED 类型的连接事件给应用服务器。应用服务器收到这一事件后判断是否需要将该连接返回连接池并调用事务管理器将代表该物理连接对象的资源从事务管理器中注销。同样，本地事务也是通过向应用服务器发送事件通知来告知该事务已经开始、提交或是回滚。

在商业应用中，经常有一些包含在全局事务中的应用组件调用了无需在这个特定的事务作用域中使用的连接，或者应用组件并不会事务性地使用到这些连接。在这些情况下，根据 JCA 规范的要求在进入事务作用域时主动获得这些连接的资源可能是没有必要的，而且如果一个事务中只用到了一个单独的连接却占用了多个连接，则可能阻止该事务的单阶段提交优化。因此，希望这些连接只有在需要它们的时候才被征用。我们利用连接事件通知机制的思想对实际应用中的事务管理进行了优化。通过增加一个连接延期(CONNECTION_PENDING)事件通知可以做到这一点，资源适配器可以用 CONNECTION_PENDING 事件来通知连接管理器何时要使用 ManagedConnection 以及何时需要在一个全局事务中使用它。在 OnceAS 中我们对 JCA 规范的标准接口 ConnectionEventListener 进行了扩展，如图5所示。

```
public interface javax.resource.spi.ConnectionEventListener {
  public void connectionClosed(ConnectionEvent event);
  public void connectionErrorOccurred(ConnectionEvent event);

  // Local Transaction Management related events
  public void localTransactionStarted(ConnectionEvent event);
  public void localTransactionCommitted(ConnectionEvent event);
  public void localTransactionRolledback(ConnectionEvent event);
}
public interface ConnectionListener extends javax.resource.spi.ConnectionEventListener{
  public static final int CONNECTION_PENDING;
  public abstract void connectionPending(ConnectionEvent) throws ResourceException;
}
```

图5 使用连接延期事件扩展接口

同时，通过在资源适配器中将事务资源设置为动态注册，使得应用服务器可以监视交互挂起事件，这样就可以使得物理连接对象只有在必要的时候才在全局事务作用域中被使用，从而提高了系统性能。

结论 本文研究了 J2EE 应用服务器使用 JCA 进行资源集成的性能问题及其优化。我们对连接池提出了应用程序连接请求队列，并使用 Reactor 模式对连接请求进行了优化，分析了基于 Reactor 模式的连接池对系统性能的提高。我们对事务管理的规范进行了扩展，提高了系统的灵活性及资源利用效率。本文提到的方法已应用到应用服务器 OnceAS 中，取得了很好的效果。

参 考 文 献

1 Sun Microsystems Inc. Java 2 Platform, Enterprise Edition Connector Architecture Specification. http://java.sun.com.

2 Sun Microsystems Inc. Java 2 Platform Enterprise Edition Specification. Version 1.3.2001.7

3 中科院软件所软件工程技术中心. OnceAS 1.0应用服务器技术白皮书. 2004.5

4 Bernstein P, Newcomer E. Principles of Transaction Processing. Morgan Kaufmann Publishers, 1997

5 Schmidt D C. Experience Using Design Patterns to Develop Reuseable Object-Oriented Communication Software. Communications of the ACM, 1995, 38(10): 65～74

6 Sun Microsystems Inc. Java Naming and Directory Interface Application Programming Interface. 1999