基于“空间”的编程

作者：Bernhard ANGERER　中文顾问：周念军，刘英博　翻译：王彤
Bernhard ANGERER : Gigaspaces Technologies Inc., 317 Madison Avenue, Suite 1220, New York, NY 10017. www.angerer.com. 周念军：IBM Corporation, 19 Skyline Dr., Hawtorne, New York 10532. jzhou@ibm.com. 刘英博：北京市海淀区清华大学软件学院, 邮编:100084. lyb01@mails.tsinghua.edu.cn. 王彤：Rosensteel & Beckmann LLC, 90 Park Avenue, Suite 1710, New York, NY 10016. twang@rosebeck.com.

摘　　要: 当前中间件的体系结构以层次为基础，由构成不同层次的模块组成，诸如通信层，业务层，数据层等。虽然该模式具有广泛的适用性，但随着应用服务器的规格要求，复杂性和分布程度持续增高，它的效率则越来越低。本文意在全面地介绍一种基于“空间”的中间件的编程方法。这是一种以分布式虚拟共享内存为技术基础设施的新型编程方法。其重要性在于它将分布计算的语意环境和问题领域的语意环境在极大的程度上相分离，从而可以为今天软件体系结构中存在的种种问题提供具有持续可靠性的解决方案。

关键词: 协调中间件; 基于“代理”的体系结构; 基于“空间”的体系结构, 数据网格; tuple 空间，java 空间

作者简介： Bernhard，1970 年生，男，奥地利公民，计算机学博士，“基于空间”的系统软件工程师。他的专长包括面向对象的编程，面向服务的编程，组件体系结构，虚拟共享内存系统和语意环境计算系统（如资源描述框架）。

中图分类号: TP311.11　**文献标志码:**　**文章编号:**

Space-Based Programming

Author: Bernhard ANGERER　Chinese Advisor: ZHOU Nian-jun, LIU Ying-bo　Translator: WANG Tong
Bernhard ANGERER : Gigaspaces Technologies Inc., 317 Madison Avenue, Suite 1220, New York, NY 10017. www.angerer.com. ZHOU Nian-jun: IBM Corporation, 19 Skyline Dr., Hawtorne, New York 10532.. jzhou@ibm.com. LIU Ying-bo: Tsinghua University Software Research Institute, Haidian Dist. Beijing, China 100084. lyb01@mails.tsinghua.edu.cn. WANG Tong: Rosensteel & Beckmann LLC, 90 Park Avenue, Suite 1710, New York, NY 10016. twang@rosebeck.com.

Abstract: Current middleware architectures follow the tier-based approach with different sets of building blocks that are separated into modules such as messaging-tier, business-tier and data-tier. Whereas this model is a sufficient fit for a whole range of applications, it becomes more and more deficient as the requirements, complexities and degree of distribution for the application server continues to rise. The intention of this article is to give a comprehensive overview about the space based approach to middleware. This model introduces a distributed, virtual shared memory infrastructure that comes along with a new type of programming model. The significance is that it greatly decouples the semantics of distributed computing from the semantics of the problem domain and therefore provides sustainable answers to vivid problems in today’s software architectures.

Keywords: coordination middleware, agent-based architecture, space-based architecture, data grids, tuple-spaces, javaspaces

Biography: Bernhard, born 1970, male, Austrian, PhD in computer science, a software engineer focusing on space based systems. His field of expertise covers object-orientation, service orientation, component architectures, virtual shared memory systems and semantic computing such as the resource description framework.

导言

“基于空间的编程”代表着分布式应用软件的新型设计方法。当前占统治地位的分布式编程是以远程过程调用（RPC）为基础。这种方法突出地体现在 CORBA 技术平台中，而且也是 Web 服务标准（Web-Services standards）下的基本设计方法。“空间体系结构”（space-based architechture）则用一种极其简捷的模式完全取代了远程过程调用。其固有的简洁的特性决定了它具有广泛的适用性，并赋予其模块性（modularity），可扩展性（scalability）和源代码经济性（source code economy）的优点。

八十年代早期，为了支持“分布式的应用软件”，耶鲁大学的 Gelernter 教授开发了一种名为 Linda 的编程语言[1]，从而迈出了走向“空间体系结构”的第一步。Linda 由一套数量不多的操作（operations）与一个全局共享的持续存储 – 即所谓的“tuple 空间”（名为“tuple 空间”的技术基础设施–infrastructure 诞生并存在于大学的实验室里；九十年代后期，Sun Microsystems 采纳了“空间”的概念，创造了 JavaSpaces 即“java 空间”）- 相结合构成。其协调规则与经典的编程语言特性互补因而可以轻松地融合于其中。研究结果显示，使用这种体系结构，并行和分布计算中的大量问题都可以得到优雅地解决。

今天，“基于空间的编程”已进入代表现代网格（grid）系统的主流范畴，深深地影响着所有有价值的应用软件（applications）的工程设计，尤其是在高同步分布式（distributed）高性能的系统环境中：如何将不同的元素（进程-processes，对象-objects，服务程序-services，代理程序-agents，人）放入到这样的一个工作系统中并使其在开放的和动态的环境中完成富有成效地通讯、集成与合作，仍是当前最重要和最具挑战性的技术问题之一。

随着基于“空间”的普通协调中间件（middleware）成功地产业化地运用于复杂应用软件的工程设计[2]，2003 年发表的同名原文逐渐得到了越来越多的关注[3]。本文是 2003 年文章的更新版本，保留了原文的主要结构但进一步丰富了一些细节，并添加了一个例子来说明基于“空间”的体系结构的运行和编程风格，同时补充了该技术应用方面的最新发展现状。

1 分布式的共享的内存

所谓“空间”指的是一个虚拟的、分布式的、共享的内存。实际上，这种内存系统的思想本身已是长期研发的对象，并不是什么新东西。“新”的是对于“基于空间”的编程模式的技术基础设施的全面理解。这与“面向对象”的编程思想的发展过程很相似，即编程的工具先出现，但产生深远影响的严谨理论则是在许多年以后才产生的（Jini 和 JavaSpaces 的区分与共通–视见解不同–为这种转化增添了额外的障碍[4]）。

“空间”的应用编程接口（API）本质上包含四种方法：*写入，读出，提取，通知*。“写入”和“读出”分别用来存储和访问对象。“提取”用来在“读出”之后消除对象，“通知”用来宣布“空间”的变化。

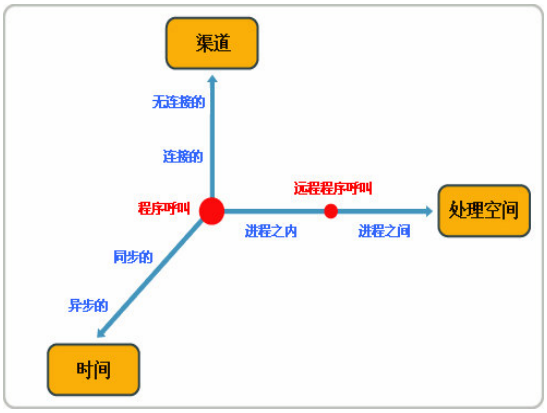


图 1：全方位的自由

这种设计思想使得分布式系统的模型大为简化[5]。在一个远程过程调用的模型中，对象们通过显式（explicit）的过程调用来相互通讯并进行同步（synchronize）。如果没有额外的辅助措施，这种方法的可扩展性不会很好；一般来讲，参加的对象越多，它们之间的通信线路就越长越复杂[6]。然而，在一个“空间”的模型中，“空间”本身在参加者（participants）之间起到同步的作用。在这种典型的“黑板通讯”模式中，参加者：

- 无需彼此了解（它们之间的交流是非连接的和匿名的）
- 不需要共享同样的进程或机器（跨进程的-interprocess）
- 暂时地不依赖于彼此（异步的）

这种在所有三个维度上的全面自由度（见图 1）会导致非常好而灵活的系统设计效果。

从实现层面上讲，“空间”由“空间服务程序”提供，这种服务程序一般具备下列基本功能：

- **透明数据共享性**：“空间”全透明地满足一个网络（network）当中多进程的并发的数据共享操作。由于应用软件的开发者不需要知道相互之间交流的细节，“分布式的数据结构（data structures）”[7]很容易建立。实际上的数据交付机制（复制，传播）发生在幕后并因执行而异。

- **持续性**：“空间”维护着一个存储机制以确保对象的存在，直到某个进程明确地将它们消除，或由“空间”在租借期满时将其消除。这样一来，象“聊天”这样的应用软件在执行过程中就不再需要同时调用其参与程序。

- **相关性**（Associativity）：对象在“空间”中采用联想的方式定位。“读出”操作通过描述数据域的内容来寻找某一对象（也许采用空白的通配符）。“空间”的搜索引擎则通过将值和类型相匹配来返回需要的对象。

- **交易性**（Transaction）：“空间”通过事务处理（transaction）模型来保证一个或多个“空间”中的一个或多个操作在执行过程中的原子性（atomicity）。

- **代码转移性**（Code Transfer）：对象作为被动数据以序列化的形式存在于“空间”中。然而，一经“读出”，一个对象包含的数据就可以被修改，其“方法（methods）”也可以被调用。在 Java 中对象加载的任务由类加载程序完成。

“空间”服务程序的关键在于其所提供的这种基本范式（paradigm）的简单性：它非常容易把握，不需要复杂而曲折的学习过程；最低限度的应用层面编程可以极大地降低开发成本；此外，它通过将“发送者（sender）”与“接收者（receiver）”相分离，把联接它们的应用逻辑变得更简单，更灵活，更稳定。这些特点尤其有助于大型应用软件的构造，分析，测试和再利用。

2 主人-工人的模式

主人–工人的模式（the master-worker pattern）[8]完美地示范了“空间”的方法在开发一个 Web 应用软件中的运用。网络服务器在收到客户的请求（例如，向不同的档案库提出问询）后，将其作为请求对象异步地放置到“空间”中（见图 2）。相关的工作进程（worker processes）则并行反

应处理请求，并将处理结果作为答复对象返回到“空间”中。网络服务器在接到答复通知后，再将答案按照用户界面的要求重新包装提供给客户。

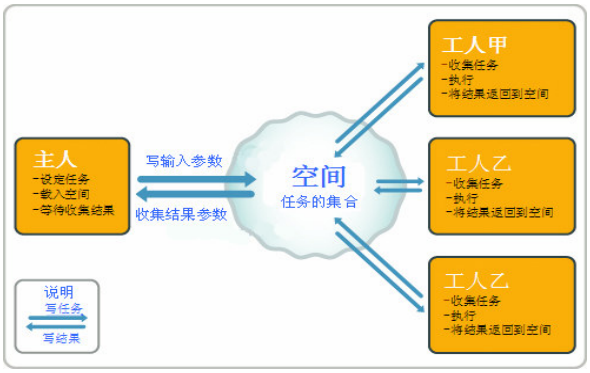


图 2：主人-工人的模式

通过以“空间”作为通讯媒体以及交流进程之间的完全分离，这一执行过程可以线形地伸缩。如果并发请求激增，只要根据需要添加额外的网络服务器和工人机器（廉价的标准硬件）就可以了。

一个工人程序代表一个档案适配器。如要增加一个档案库，只要重新编写一个适配器就可以将其引入一个正在运作的系统中。这种“热拔插”的能力不仅有助于系统维护而且有助于系统开发和调试。

3 分布式的高速缓存

“空间”的另一个固有的内在特征是分布式的高速缓存（caching）。工作进程可以事先将答复对象存放在“空间”里，只需维持着答案与请求之间的映射关系，当一个匹配的请求到达时即可将答复对象立刻返回。

众所周知，从成本效益比的角度考虑，要在一个系统中实现最佳的负载（load）平衡通常不是件容易的事。对此，“空间”同样提供了一条“自然的” 捷径[9]。因为工人程序非常清楚它们各自机器的负载，所以可以“授权”“空间”来负责对请求的再分配。只有在一个有足够资源的工人程序存在的条件下，一个请求才会被“空间”接受。负载就这样在工作进程和机器之间自行调整，而无需发布有关负载本身的信息。

概括起来，基于“空间”的编程方法具有以下优点：

- 更快的开发周期
- 通过添加廉价标准硬件实现的高度可扩展性
- 负载从网络服务器转移到同时运行的工作进程上
- 界面之间完全分离
- 分布式的高速缓存
- 异构系统（heterogenous landscape）之间的简单集成

罗克希德马丁公司（Lockheed Martin）的首席技术官 Bill Rawlings 曾宣称：他利用“空间”的编程方法得以将源代码（相对于 J2EE 来说）减少三分之二[10]。不难想象这对于项目预算和时间表会带来什么样的影响。

“基于空间的编程”的这些优点从根本上要归功于以下因素：

- **基于黑板的体系结构**：“空间”代表一种新型的信息总线（bus），与整个的异构系统之间只有一个界面。进程们因此而同步触发。
- **应用程序内“活”的对象**：由于考虑到可扩展性的需要，数据库通常是唯一能够永久存放状态信息的地方（无状态的编程）。有了“空间”以后，技术和业务对象都可以在“空间”中存储其状态，设计得以大大地简化。“阻抗失配”的问题失去其尖锐性。
- **最低限度的应用层面编程**：“空间”接口的简单性是其所有显著特点的关键。它避免了复杂的接口描述语言（例如 CORBA IDL）-- 参加者只须遵循固定的接口编程范式，因为他们都居于同一地址的“空间”。

上述因素可以被总结为基于“空间”的“编程模式”。请注意，这些模式的实现离不开下列基于“空间”的技术基础设施，即：

- **对等式（peer to peer）引擎**：因为“空间”是一个由“引擎的联邦”创造的分布式被共享的内存，它没有中心服务器（因而没有单点失败以及瓶颈问题）。
- **集群（clustering）和高速缓存模块的融合**：在传统的 n 层的体系结构中，操作系统，数据库和中间件都各有自己的高可用性和一致性模型。而在一个基于“空间”的系统中，所有与高可用性，可扩展性和高速缓存相关的问题都由同一套技术基础设施来解决。
- **通讯和协调模块的融合**：在传统的分布式系统中，进程同步，远程过程调用和信号传递分别由不同的模块来操作。而在一个基于“空间”的系统中，所有软件之间的通讯和协调都由同一个机制来处理，不管它们是在同一地址的“空间”中对话，还是以同步或异步的方式远程通话。

4 举例

为了说明一个基于“空间”的体系结构是如何工作的，让我们看一下贷款处理的例子。贷款处理涉及以下步骤：**收集有关申请人的数据**；**检验申请人数据**（即，核实就业情况和银行/投资帐户余额）；**信用打分**；**风险分析**和**定价**；**保险**；以及**完成**。用基于“空间”的方法来实现这一过程，不同的任务被清晰地分成彼此通信的工人程序或服务程序。大家也可以将此看作是一个单一登录的问题。不同的服务程序需要联络各种各样的应用软件来完成任务,彼此分离，但仍需要一个共同区域来共享它们的请求和结果。然

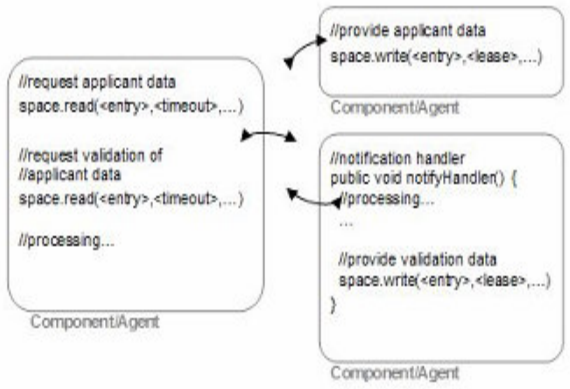
而，在现有的系统中，这样的一个操作比看上去要复杂的多，因为每个层次都需要有自己的高可用性和一致性模型。

目前这样的一个应用软件只能以“面向服务”的方式来设计和开发。其产物就是所谓的“工作流程应用软件”，其中每个服务程序同时既要维持多个对话，又要发送和接收消息。另外，鉴于使用环境（服务之间的组织-service orchestration）或网络环境（局域网/广域网）有可能发生变化，服务程序的反应时间经过一段时期也会发生变化。

当前开发的这种应用软件大多数成效不高。仅拿一个方面来看，每当消息进来的时候，程序都必须从数据库中调出状态信息，采取相应的行动，然后再将其存储回去。这需要大量的重复代码。同时还必须将其写成一个由一套消息处理器（handler）组成的事件驱动型程序，很难寻找逻辑流程。

“空间”通过让你对有待分布的事实近乎忽略不计，将这样的应用软件变得非常容易写。通讯和协调的部分被抽象化了，因为代码是在分布于幕后的数据结构上工作的。所谓的局部变量变成了“空间”入口。请求对象或条目被放入到“空间”中请求答复。通知对象也被放入到“空间”中以便在答案到达时被通知。在某个特定请求之前可能还有其他请求需要先处理。由于编程的复杂性被减少到“几乎”只有一个单一内存，对协调的需要也就不知不觉地自行化解了。

比如，下面是我们的贷款处理一例的开始部分：



一定数量的软件组件以一种完全分离的方式相互通讯。一个组件提供申请人数据。另一个组件在这些对象到达时得到通知并提供相应的检验数据。第三个组件在等侯这些数据以继续进程链。

以上体现了“空间”的最基本的应用编程接口。条目的对象被写入“空间”。“事件/数据流”的特点主要通过超时（timeout）设定（等侯条目到达的时间）和租期（lease）设定（自动驱逐条目的时间）来决定。有关使用 JavaSpaces 的应用接口编程来实现分布式的数据结构的详细描述参阅 JavaSpaces Principles, Patterns, and Practice （见参考文献 7-2）。

图 3：贷款处理（伪代码）

虽然图 3 描述的情形非常容易写，但传统上则视之为不可取，因为：

- 每一个对话都要求有一个带有某种同步机制的牵涉好几个机器的线程（thread）
- 没有处理更多负载的可扩展性，需要另外想办法
- 如果虚拟机（JVM - Java Virtual Machine 或者 CLR - The Common Language Runtime）关闭，所有有关正在进行的对话的信息都会丢失

5 结论

“空间”系统结构的魅力在于它在保证简捷性的同时提供强大的功能[11]。与开发分布式应用软件的其他模式相比，它设计简单，节约开发和调试成本，使系统更健壮，维护和集成更容易。“空间”的体系结构通过将复杂性从执行阶段转移到配置阶段，成功地将分布计算的语义环境和问题领域的语义环境相分离。在此请注意，在以“空间”为基础的系统结构中，配置不仅与分布计算环境相关，而且是解决性能和可扩展性问题的主要手段。“空间”的引擎之所以能够在差别很大的条件下（比如，读操作密集或是写操作密集）运行良好，这种灵活性都是由管理员通过配置来实现的。

“空间体系结构”的上述特征使其得以在虚拟程度大大提高的环境下实施和操作。这种环境将不仅抽象化地提供分布式，成批次，规模化，可用性等等服务，而且将开启一扇通向前所未有的快捷性和灵活性的大门，将设计和编程人员从当前只能靠漫长的时间和巨额投资才能解决的技术束缚之中解脱出来。

“当 Linda 语言在逻辑上可以良好地适用于以下所有操作的时候，为什么我们还要用三个工具箱来分别处理并行应用（如消息传递），单处理器（uniprocessor）（如带锁的共享内存），和跨网络（trans-network）通讯（如远程过程调用）呢？”- David Gelenter （见参考文献 1-1）

“我设计和编写了 SPECTRE 来与当前电子商务（E-business）中流行的典型的“三层”体系结构相比较。“三层”体系结构需要 12000 多行代码，两个半人工和六个月的时间才能写完。而与其性能相同的 SPECTRE 只需不到 90 行代码，我用三个星期就写完了。如果“Java 空间”需要一些“重磅炸弹”才能被接纳和认可的话，这些数字应该是足够的了。”- Bill Rawlings （见参考文献 10）

“警告：变革即在眼前。如果您想知道在将来应当如何设计和编写服务器应用软件。。。那么您应当往下读。。。注意：拒绝是没有用的。”- Sing Li（见参考文献 4）

“迄今为止，站在最前沿的用户们都是通过对各类最好的技术加以整合来累加起具有高性能和高扩展性的应用软件平台。但新一代博大精深的标准化的 XTP 平台将从网格技术，事件驱动型的结构体系，分布式的高速缓存，和高性能通讯的融合（convergence）之中诞生。”- Massimo Pezzini / Gartner（见参考文献 15）

“思维上的量子飞跃。”- Bill Joy（见参考文献[GUTH98]）

图 4：有关“空间”的评论

目前已有一定数目的“基于空间”的系统投产使用。大多数这样的系统所解决和处理的复杂程度远远超出传统应用服务器所能处理的范围，因为传统服务器遵循以分层为基础的模型[12]，需要在应用层面，数据层面和通讯层面上分别

集群[13]。Gartner 注意到这一新发展并开始赋予这类应用软件一个新名词，即 XTP（extreme transaction processing）[14]。Gartner 给 XTP 下的定义是：网格技术，事件驱动型的结构体系，分布式的高速缓存平台，和高性能通讯的融合[15]。这同样也是对于“基于空间的系统”的恰当定义。

参考文献

[1] 参阅，D. GELERNTER, N. CARRIERO, Coordination Languages and their Significance [J]. Communications of the ACM, Feb. 1992, Vol. 35, No.2；
D. GELERNTER. Mirror Worlds: Or the Day Software Puts the Universe in Shoebox...How It Will Happen and What It Will Mean [M]. Oxford University Press, 1992, ISBN-13: 978-0195079067

[2] 参阅，GARTNER. GigaSpaces Positioned in Visionaries Quadrant [R]. GRIDtoday, 2006, http://www.gridtoday.com/grid/959632.html；
GRIDTODAY, Grids Gets Transactional [J]. GRIDtoday, 2006, http://www.gridtoday.com/grid/1150800.html；
IBM, Scaling the Grid, Case Study [R]. Gigaspaces, 2006, http://www03.ibm.com/servers/deepcomputing/cod/pdf/fsscasestudy.pdf

[3] 参阅，B. ANGERER. Space based Programming [J]. O’Reilly – ONJava.com, 2003, http://www.onjava.com/pub/a/onjava/2003/03/19/java_spaces.html

[4] 参阅，S. LI, P. HOULE, M. WILCOX, R. PHILLIPS, Professional Java Server Programming: with Servlets, JavaServer Pages (JSP), XML, Enterprise JavaBeans (EJB), JNDI, CORBA, Jini and Javaspaces (平装) [M]. Wrox Press 1999

[5] 参阅，B. VENNERS, Designing as if Programmers are People [J]. java.net, 2003, http://today.java.net/pub/a/today/2003/06/10/design.html

[6] 参阅，B. ANGERER, A. ERLACHER. Loosely Coupled Communication and Coordination in Next-Generation Java [J]. Middleware, java.net, 2005, http://today.java.net/pub/a/today/2005/06/03/loose.html

[7] 比较，E. FREEMAN, S. HUPFER. "Make room for JavaSpaces - Part 1" [J]. JavaWorld/IBM-developerworks 1999, http://www-128.ibm.com/developerworks/java/library/j-space/；和 E. FREEMAN, S. HUPFER, K. ARNOLD. JavaSpaces Principles, Patterns, and Practice [M]. Addison-Wesley 1999

[8] 参阅，T. WHITE. How To Build a ComputeFarm [J]. java.net 2005, http://today.java.net/pub/a/today/2005/04/21/farm.html

[9] 参阅，S. LI, High-impact Web tier clustering, Part 2: Building adaptive, scalable solutions with JavaSpaces [J]. IBM developerWorks, Wrox Press 2003, http://www.ibm.com/developerworks/java/library/j-cluster2/

[10] 参阅，B. RAWLINGS. JavaSpace mailing list, 003081 or 001588 [EB]. http://swjscmail1.sun.com/archives/javaspaces-users.html

[11] 见上 5.

[12] 见上 2-3.

[13] 参阅，N. SHALOM, Space-Based Architecture and The End of Tier-based Computing (白皮书) [R]. GigaSpaces Technologies 2006, http://www.sun.com/third-party/srsc/resources/gigaspaces/GigaSpacesEndofTierBasedComputingwp.pdf

[14] 参阅，M. PEZZINI, The Challenges of Extreme Transaction Processing in a World of Services and Events [R]. Gartner 2006, http://mediaproducts.gartner.com/reprints/oracle/131036.html

[15] 参阅，GRIDTODAY, GigaSpaces Releases eXtreme Application Platform [J]. GridToday 2007, http://www.gridtoday.com/grid/1611358.html

[Guth98] R. GUTH, More than just another pretty name - Sun's Jini opens up a new world of distributed computer systems [J]. SunWorld 1998, http://sunsite.uakom.sk/sunworldonline/swol-08-1998/swol-08-jini.html