揭开 J2EE 集群的神秘面纱

2005 年八月

转载自http://blog.csdn.net/esoftwind
翻译自 TSS 的文章。-- Uncover the hood of J2EE Clustering
原文：http://www.theserverside.com/tt/articles/article.tss?l=J2EEClustering
整理：枫远雅客http://www.fyyk.com

1.序言

越来越多的关键应用运行在 J2EE（Java 2, Enterprise Edition）中，这些诸如银行系统和账单处理系统需要高的可用性（High Availability, HA），同时像 Google 和 Yahoo 这种大系统需要大的伸缩性。高可用性和伸缩性在今天高速增长的互连接的世界的重要性已经证实了。eBay 于 1999 年 6 月停机 22 小时的事故，中断了约 230 万的拍卖，使 eBay 的股票下降了 9.2 个百分点。

J2EE 集群是用来提供高可用性和伸缩性服务，同时支持容错处理的一种流行的技术。但是，由于 J2EE 规范缺乏对集群的支持，J2EE 供应商实现集群的方法也各异。这给 J2EE 架构师和开发人员带来了很多困难。以下是几个常见的问题：

- 为什么带集群功能的商业 J2EE 服务器产品如此昂贵？（10 倍于不带集群功能的产品）
- 为什么基于单服务器环境构建的应用不能在集群中运行？
- 为什么应用在集群环境中运行得很慢，但在非集群环境中却快得多？
- 为什么集群的应用移植到其他服务器中失败？

理解这些限制和要素的最佳方法是学习他们的实现方式。

2.基本术语

在我们讨论不同的集群实现之前，先谈谈几个概念。这有助于理解不同的 J2EE 集群产品不同的设计结果和概念：

伸缩性（Scalability）：

在一些大的系统中，预测最终用户的数量和行为是非常困难的，伸缩性是指系统适应不断增长的用户数的能力。提高这种并发会话能力的一种最直观的方式就增加资源（CPU，内存，硬盘等），集群是解决这个问题的另一种方式，它允许一组服务器组在一起，像单个服务器一样分担处理一个繁重的任务。

高可用性（High availability）：

单一服务器的解决方案并不是一个健壮方式，因为容易出现单点失效。像银行、账单处理这样一些关键的应用程序是不能容忍哪怕是几分钟的死机。它们需要这样一些服务在任何时间都可以访问并在可预期的合理的时间周期内有响应。集群方案通过在集群中增加的冗余的服务器，使得在其中一台服务器失效后仍能提供服务，从而获得高的可用性。

负载均衡（Load balancing）：

负载均衡是集群的一项关键技术，通过把请求分发给不同的服务器，从而获得高可用性和较好的性能。一个负载均衡器可以是从一个简单的 Servlet 或 Plug-Ins（例如一个 Linux box 利

用 ipchains 来实现），到昂贵的内置 SSL 加速器的硬件。除此之外，负载均衡器还需执行一些其他的重要任务，如“会话胶粘”让一个用户会话始终存在一个服务器上，“健康检查”用于防止将请求分发到已失效的服务器上。有些负载均衡器也会参与我们下面将要谈到“失效转移”过程。

容错（Fault tolerance）：

高可用性意味着对数据正确性的要求不那么高。在 J2EE 集群中，当一个服务器实例失效后，服务仍然是有效的，这是因为新的请求将被冗余服务器处理。但是，当一个请求在一个正在失效的服务器中处理时，可能得到不正确的结果。不管有多少个错误，容错的服务应当能确保有严格的正确的行为。

失效转移（Failover）：

失效转移是集群中用来获取容错能力的另一项关键的技术。当一个结点失效后，通过选择集群中的另一个结点，处理将会继续而不会终止。转移到另一个结点可以被显式的编码，或是通过底层平台自动地透明地路由到另一个服务器。

等幂方法（Idempotent methods）：

等幂方法是指这样一些方法：重复用相同的参数调用都能得到相同的结果。这些方法不会影响系统状态，可以重复调用而不用担心改变系统。例如：getUsername()就是等幂的，而 deleteFile 就不是。当我们讨论 HTTP Session 失效转移和 EJB 失效转移时，它是一个重要的概念。

3.什么是 J2EE 集群

一个天真的问题，不是吗？但我仍要用几句话和图来回答它。通常，J2EE 集群技术包括"负载均衡"和"失效转移"。

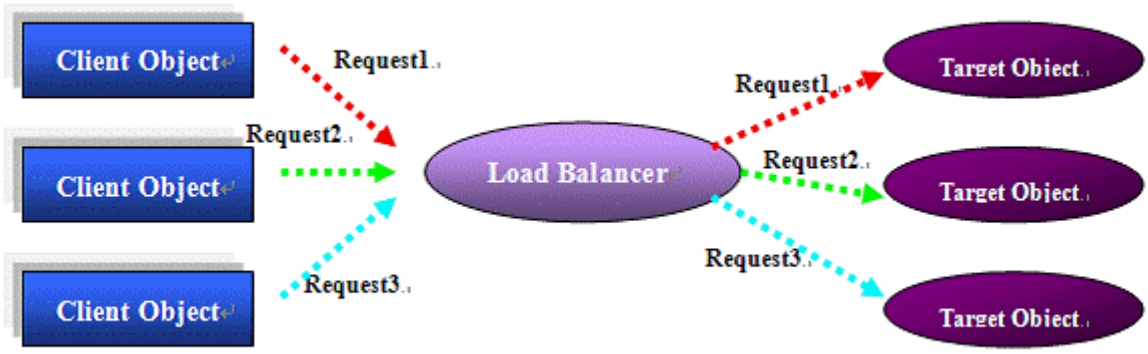

图 1 负载均衡

如图 1 所示，负载均衡意味着有许多客户端向目标对象同时发出请求。负载均衡器在调用者和被调用者之间，分发请求到与原始对象相同的冗余对象中。伸缩性和高可用性就是这样得到的。

图 2 失效转移

如图 2 所示，失效转移与负载均衡不同。有时客户端会连续发请求到目标对象，如果请求中间目标对象失效了，失效转移系统将检测到这次失败，并将请求重定向到另一个可用的对象。通过这种方式可以获得容错能力。

如果你想知道更多的有关 J2EE 集群的知识，你就会问到一个基本的问题，“什么对象可以集群？”和“在我的 J2EE 代码中哪里会发生负载均衡和失效转移呢？”。这些都是用来理解 J2EE 集群的非常好的问题。实际上，并不是所有的对象都能被集群的，并且负载均衡和失效转移并不是在 J2EE 代码所有地方都能发生。看看下面的例子代码：

```
public class A {
……
public void business() {
    B instance1 = new B();
    instance1.method1();
    instance1.method2();
    ……..
}
……..
}
```

```
public class B {
……
public void method1() {
    ……
}

public void method2() {
    ……
}
```

图 3 例子代码

在 Class A 的 bussiness()方法中，instance1 可以负载均衡吗？或是当其失效，可以失效转移到其他 B 的实例上吗？我想是不行的！对负载均衡和失效转移来说，必须要有个拦截器在调用者和被调用者之间分发或重定向请求到不同的对象上。Class A 和 Class B 的实例是运行在一个 JVM 中紧密耦合的，在方法调用间加入分发逻辑非常困难。

什么类型对象可以被集群？——只有那些可以被部署到分布式拓朴结构中的组件。

在我的 J2EE 代码中，什么地方会有负载均衡和失效转移？——只在你调用分布式组件的方法时。

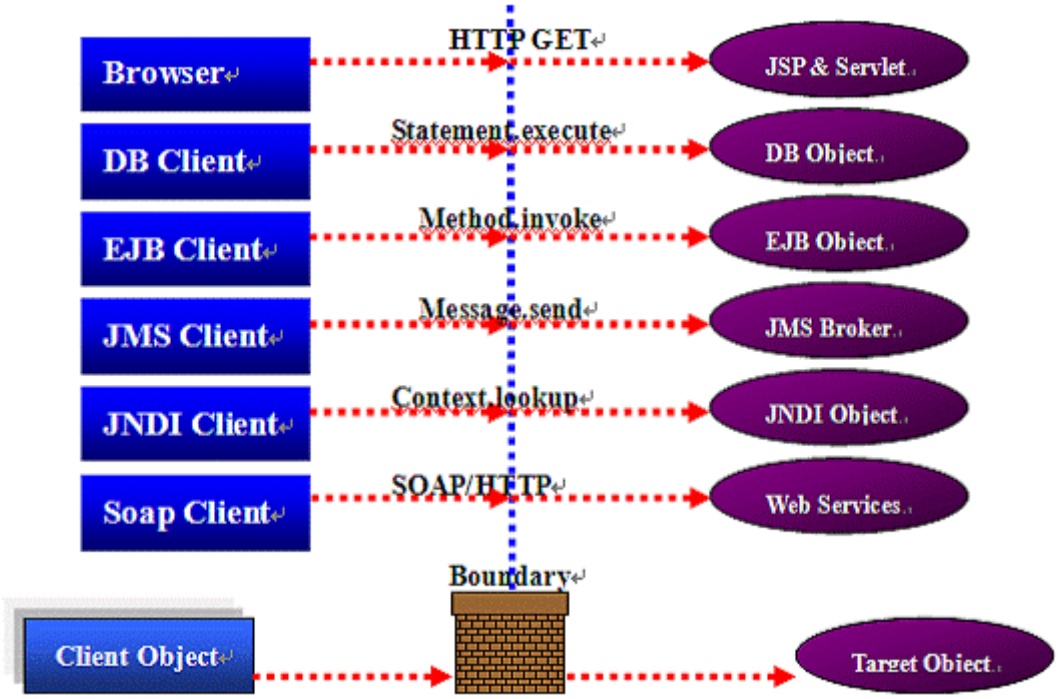

图 4 分布式对象

在如图 4 所示的分布式环境中，调用者和被调用者被分离在有明显边界的不同的运行容器中，这个边界可以是 JVM，进程和机器。

当目标对象被客户端调用时，目标对象的功能是在容器中运行的（这就是为什么我们说它是分布式的原因）。客户端和目标对象通过标准的网络协议通信。这些特性就为一些机制提供了机会可以介入到方法调用之间实现负载均衡和失效转移。

如图 4，浏览器通过 HTTP 协议调用 JSP 对象，JSP 运行在 WEB 服务器中，浏览器只需要返回结果而不关心它是怎么运行的。在上述场景中，一些东西就可以在浏览器与 WEB 服务器之间实现负载均衡和失效转移的功能。在 J2EE 平台，分布式技术包括：JSP（Servlet），JDBC，EJB，JNDI，JMS，WEB Service 等。负载均衡和失效转移就发生在这些分布式方法被调用时。在后续部分我们将详细讨论这些技术。

4. WEB 层集群实现

WEB 层集群是 J2EE 集群的重要且基本的功能。WEB 集群技术包括 WEB 负载均衡和 HTTP Session 失效转移。

4.1. WEB 负载均衡

J2EE 提供商实现 WEB 负载均衡有许多方式。基本上，都一个负载均衡器被插入到浏览器和 WEB 服务器之间，如下图所示。

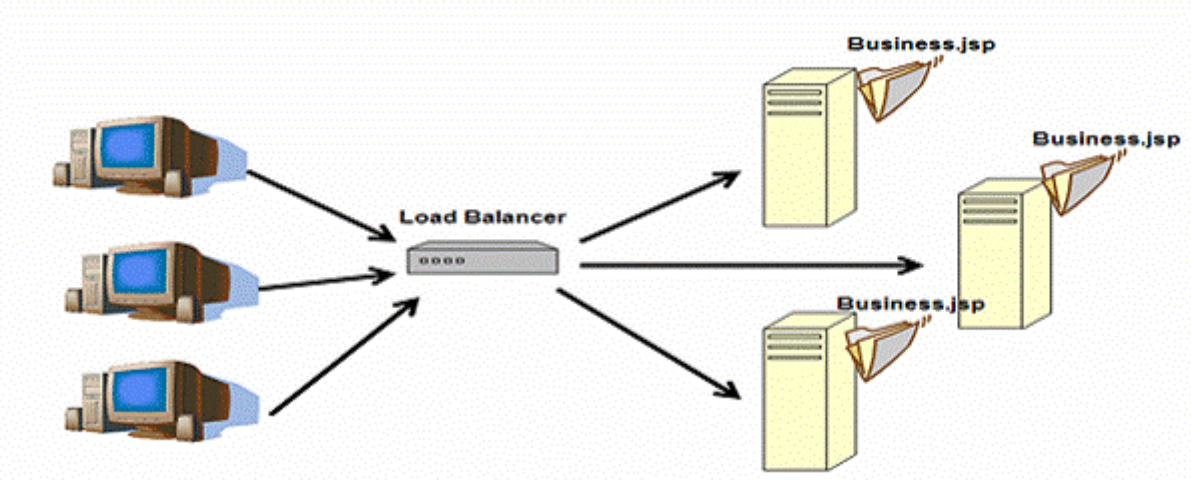

图 5 WEB 负载均衡

负载均衡器可以是一台硬件，如 F5 负载均衡器，或仅仅是另一台有负载均衡 Plug-Ins 的 WEB 服务器，一个简单的带 ipchains 的 Linux box 可以很好的实现负载均衡。不管采用哪种技术，负载均衡器都有以下特性：

- **实现负载均衡算法**

  当客户请求到来时，负载均衡器需要决定将如何分发到后台服务器。流行的算法是 Round-Robin、Random 和 Weight Based。负载均衡器尽力使每台服务器实例都获得相同的负载，但是上述算法没有一个可以获得理想的负载相同，因为它们仅仅是依据发送到特定服务器的请求的个数。一些精密的负载均衡器实现了特殊的算法。它在分发请求之前将检测服务器的工作负载。

- **健康检测**

  当一台服务器失效了，负载均衡器应当检测出失效并不再将请求分发到这台服务器上。同样，它也要检测服务器是否恢复正常，并恢复分发请求。

- **会话胶粘**

  几乎所有的 WEB 应用程序都有一些会话状态，可能是简单的记住用户是否登陆，或是包含你的购物车信息。因为 HTTP 本身是无状态的，会话状态应当存在服务器的某个地方并与你当前浏览会话相关联，这样当你下次再请求相同 WEB 应用程序的页面时可以很容易的重新获取。当负载均衡时，最佳的选择就是将特定的浏览器会话分发到上次相同的服务器实例中，否则，应用程序可能不能正确工作。

因为会话状态存储在特定 WEB 服务器的内存中，“会话胶粘”对于负荷均衡非常重要。但是，如果其中某台服务器实例因为某种原因失效了（比如关机），那么这台服务器的会话状态将要丢失。负载均衡器应当检测到这个失效并不再将请求分发给它，但这些请求的会话状态都因为存放在失效的服务器中而丢失了所有信息，这就将导致错误。会话的失效转移因此而生。

4.2. HTTP Session 失效转移

几乎所有流行的 J2EE 供应商都在他们的集群产品中实现了 Http Session 失效转移，用来保障当某台服务器失效后会话状态不会丢失，使客户端请求能被正确处理。如图 6 所示，当浏览器访问有状态的 WEB 应用程序（第 1 ,2 步），这个应用程序可能在内存创建了会话对象用于保存信息以供后面的请求使用，同时，发送给浏览器一个唯一的 HTTP Session ID 用于标识这个会话对象（第 3 步），浏览器将这个 ID 保存 Cookie 中，并当它下次再请求同一 WEB 应用

程序的页面时，会将 Cookie 发还给服务器。为了支持会话失效转移，WEB 服务器将在一定的时候把会话对象备份到其他地方以防止服务器失效后丢失会话信息（第 4 步）。负载均衡器检测到这个失败（第 5，6 步），并将后续的请求分发到装有相同应用程序的服务器实例中（第 7 步），由于会话对象已经备份到其他地方了，这个新的服务器实例可以恢复会话（第 8 步）正确地处理请求。

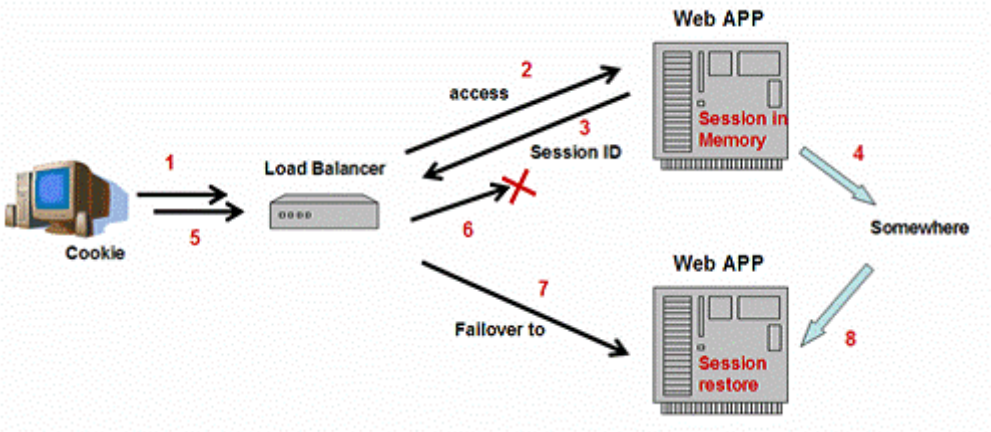

图 6 HTTP Session 失效转移

为了实现上述功能，HTTP Session 失效转移将带来以下问题：

- **全局HTTP Session ID**

  如上所述，HTTP Session ID 用于在特定的服务器实例中标识唯一的内存会话对象，在 J2EE 平台，HTTP Session ID 依赖于 JVM 实例，每个 JVM 实例拥有几个应用程序，每个应用程序都为不同的用户管着许多会话，HTTP Session ID 是在当前 JVM 实例用于访问相关会话的关键。在会话失效转移的实现中，要求不同的 JVM 实例不能产生两个相同的 HTTP Session ID，这是因为当失效转移发生时，一个 JVM 的会话将要备份并恢复到另一个中，这样，必须建立全局 HTTP Session ID 产生机制。

- **如何备份会话状态**

  如何备份会话状态是区别 J2EE 服务器好坏的关键因素。不同的供应商有不同的实现，在后续部分我再详细解释。

- **备份的频率和粒度**

  会话的备份是消耗性能的，包括 CPU，内存，网络带宽和写入磁盘和数据库的 I/O，备份的频率和备份对象的粒度将严重影响性能。

5. 数据库备份方式

几乎所有的J2EE集群产品都允许选择将你的会话对象通过JDBC备份到关系数据库中。如图 7 所示，这种方式可以让服务器实例非常简单的在正确的时间序列化会话内容并写到数据库中。当发生会话转移时，另一台可用的服务器接过已失效的服务器工作，从数据库中恢复所有的会话状态。序列化对象是关键点，它使得内存会话数据可以持久化和传输。要了解更多有关Java对象序列化知识，请参考http://java.sun.com/j2se/1.5.0/docs/guide/serialization/index.html。

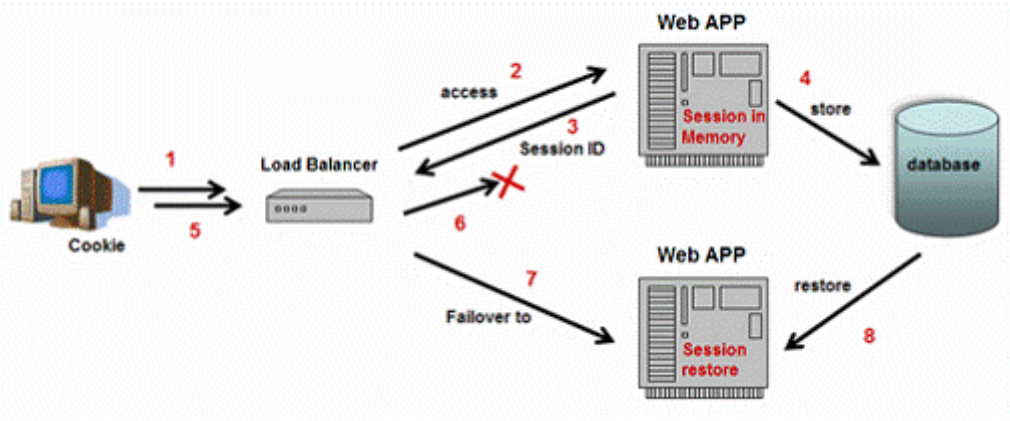

图 7 备份会话数据到数据库

由于数据库交易是非常昂贵的，这种方法主要缺点是当在会话中保存大量的或大的对象时限制了伸缩性，大多数使用数据库会话持久化的服务器产品都提倡尽量减少用 HTTP 会话存储对象，但这限制了你的应用程序的架构和设计，特别是你要使用 HTTP 会话缓存用户数据时。

数据库的方式也有一些优点：

- 简单，容易实现。分离的请求处理和会话备份处理使集群更好管理和健壮。
- 会话可以失效转移到任何一台服务器，因为数据库是共享的。
- 当整个集群失效时，会话数据依旧幸免。

6. 内存复制方式

因为性能的原因，一些 J2EE 服务器（Tomcat，Jboss，WebLogic，WebSphere）提供了另一种实现：内存复制

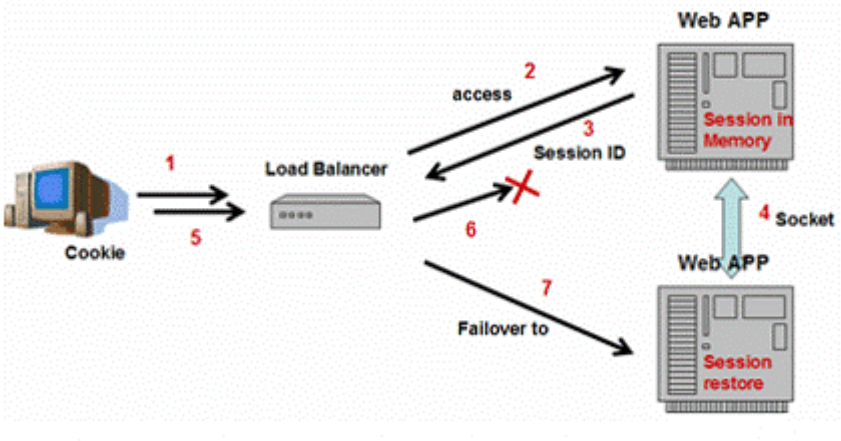

图 8 对会话状态进行内存复制

基于内存的会话持久化将会话信息保存在一台或是多台备份服务器中，而不是保存数据库中（如图 8）。这种方式因为性能高而非常流行。同数据库方式相比，直接在原服务器和备份服务器之间网络通信是非常轻量的。同时注意在使用方式中，数据库方式中的“恢复”阶段是不需要的，因为在备份后，所有会话数据都已经存在备份服务器的内存中了，已经可以处理请

求。

“Java Groups”是当前Tomcat和Jboss集群所使用的通信层。Java Groups是用于实现可靠组通信和管理的工具包。它提供了诸如“组成员协议”和“消息广播”等核心特性，这些都对集群的工作非常有用。有关Java Groups的信息，请参考：http://www.jgroups.org/javagroupsnew/docs/index.html。

Tomcat方式：多服务器复制

内存复制也存在许多不同的方式，第一种方法就是将会话数据复制到集群中的所有结点，Tomcat5 采用的就是这种方式。

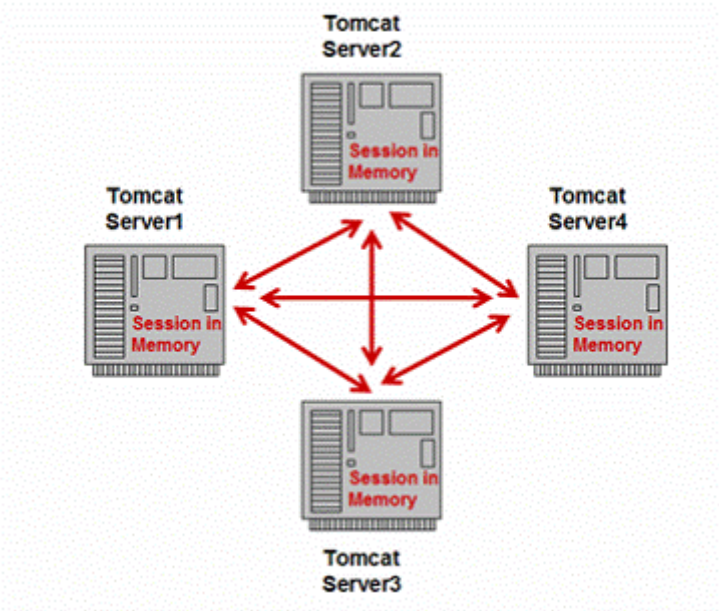

图 9 多服务器复制

如图 9 所示，当一个服务器实例的会话改变后，将备份到其他所有的服务器上。当一台服务器失效后，负载均衡器可以选择其他任何一台可用的服务器实例。但这种方式限制了伸缩性，如果集群中有很多的服务器实例，那么网络通信的代价就不能被忽略，这将严重降低性能，并且网络也将成为系统的瓶颈。

WebLogic，Jboss和Websphere的方式：对等服务器复制

由于性能和伸缩性的原因，WebLogic，Jboss 和 Webshpere 采用了其他方式实现内存复制。每台服务器任意选择一台服务器备份其内存中的会话信息。如图 10 所示。

在这种方式中，每台服务器都有一台自己的对等服务器，而不是其他所有的服务器，这种方式消除在集群中加入过多服务器实例的话影响伸缩性的问题。

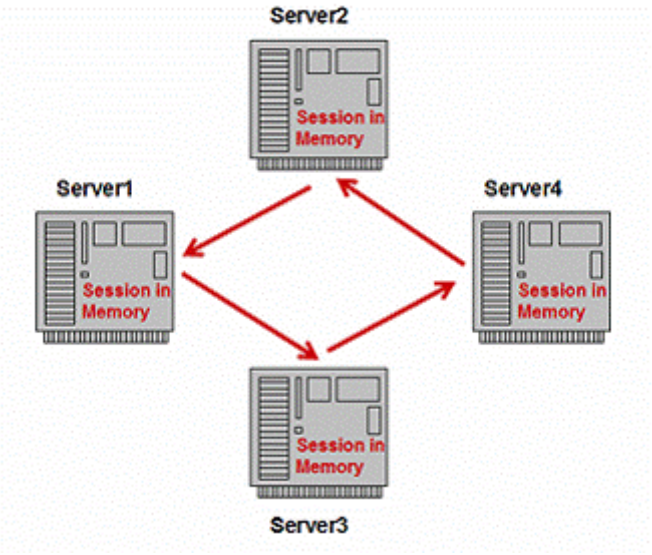

图 10 对等服务器复制

尽管这种方式实现失效转移有很高的性能和伸缩性，但它仍有一些限制：

- 它给负载均衡器带来了更多的复杂性。当一台服务失效后，负载均衡器必须知道那台服务是这台已失效服务器的对等备份服务器。这将缩小了负载均衡器的选择范围，同时有些硬件也不能满足这种要求。
- 除了处理正常的请求外，服务器还将负责复制的任务。由于备份会话数据的任务也需要占用 CPU 的周期，所以每台服务器的请求处理能力也降低了。
- 在没有发生失效转移的时候，备份服务器上大量用于备份的内存是个浪费。同时这也将增加了 JVM GC 的负担。
- 集群中的服务器实例构成了复制对。这样，当会话所在主服务器失效后，负载均衡器将会话转移到备份服务器，使备份服务器处理两倍的请求，这将造成备份服务器的性能问题。

为了克服上面的 4 点问题，不同的软件供应商采用了不同的方法，WebLogic 采用的复制对不是对每台服务器，而是对每个会话。当一台服务器实例失效后，会话数据已经分散备份到多个备份服务器上，使失效的负载均匀地分布。

IBM的方式：中央状态服务器

Websphere 采用不同的方式实现内存复制：备份会话信息到中央的状态服务器，如图 11 所示：

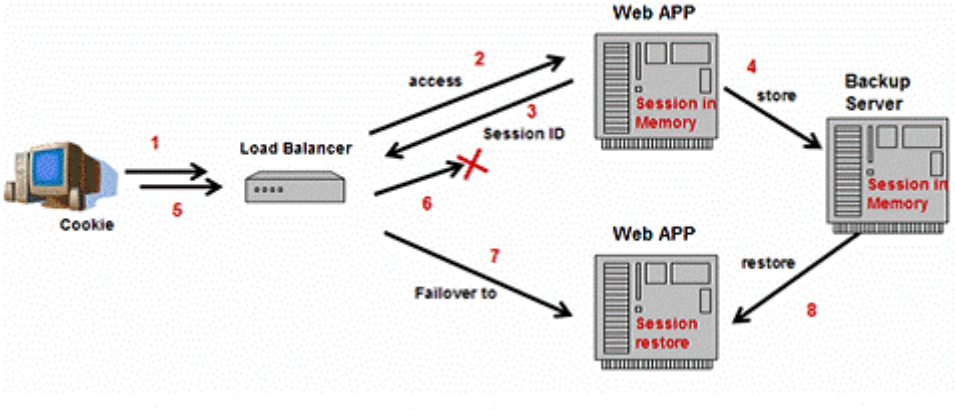

图 11 中央状态服务器复制

这与数据库的解决方案非常类似，不同之处在于专用的“会话备份服务器”代替了数据库服务器，这种方式结合了数据库和内存复制两种方式的优点。

- 将请求处理和会话备份处理分开使用集群更加健壮。
- 所有的会话数据都备份到专用的服务器上，无需服务器浪费内存用于备份其他服务器的会话。
- 因为会话备份服务器是在服务器之间共享的，所有失效后可以转移到任何一台服务器上，这样大多数据软硬件负载均衡器都可以使用，更重要的是当一台服务器失效后，负载将均匀的分布到所有实例上。
- 与重量级的数据库连接相比，应用服务器与备份服务器之间 Socket 通信是轻量的。这样就比数据库的解决方案有更好的性能和更好的伸缩性。

然而，由于有恢复失效服务器会话数据的这么一个阶段，因此其性能肯定不如两台服务器直接复制解决方案，另外，多出来一台备份服务器也增加了管理的复杂性。也可能由于单台备份服务器造成性能瓶颈。

Sun的方式：特殊数据库

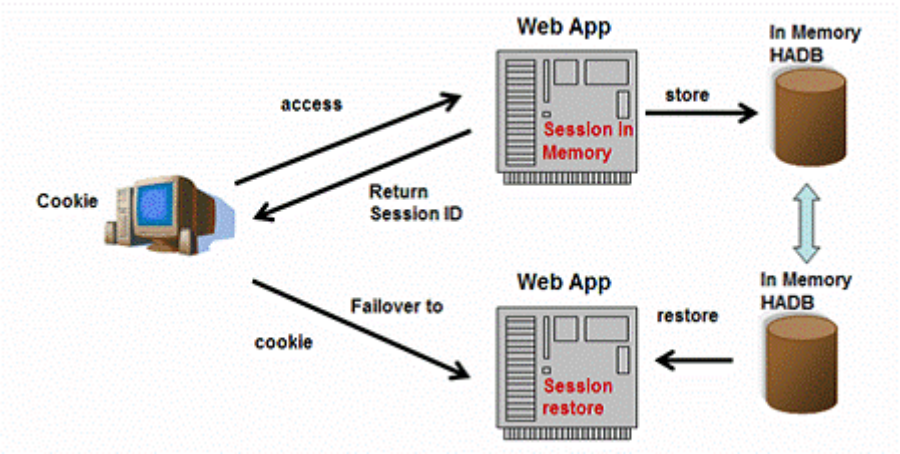

图 12 特殊数据库复制

Sun JES 应用服务器采用了别的方式实现会话失效转移，如图 12 所示，它看上去很像数据

库的方式，因为它采用关系数据库存储会话并通过 JDBC 访问所有会话数据。但是 JES 内部所使用的关系数据库称为 HADB，已经为访问会话做了特别优化，并且将几乎所有的会话数据存在内存中。这样，你可以说它更像中央状态服务器的方式。

7. 性能因素

考虑如下问题：一台 WEB 服务器中可能运行着许多 WEB 应用，它们中每一个都可能被成百的并发用户访问，而每个用户都会产生浏览器会话用于访问特定的应用。所有会话信息都将备份以便服务器失效后能转移到其他服务器实例中。更糟的是，会话会由于一次次的发生以下情况而变化，包括创建、失效、增加属性、删除属性、修改属性值。甚至是什么都没变，但由于有新的访问而使最后访问时间变了（由此判断什么时候失效会话）。因此，性能在会话失效转移的解决方案中是个很大的因素。供应商通常会提供一些可调的参数改变服务器行为，使之适应性能需求。

备份时机

当客户端的请求被处理后，会话随时改变。由于性能因素，实时备份会话是不明智的。选择备份频率需要平衡。如果备份动作发生得太频繁，性能将急剧下降。如果两次备份的间隔太长，那么在这间隔之间服务器失效后，很多会话信息将会丢失。不管所有的方式，包括数据库和内存复制，下面是决定备份频率的常用的选项。

- **按WEB请求**

  在 WEB 请求处理结束后，发生响应之前保存数据。这种方式能够保证在失效后备份的会话数据是最新的。

- **按固定的时间间隔**

  会话在后台按固定的时间间隔保存。这种方式不能保证备份的会话数据是最新的。但由于不需在每次请求之后备份数据，因而有更好的性能。

备份粒度

当备份会话的时候，你还需要决定多少会话状态需要保存。以下是不同产品所有采用的常用的策略。

- **整个会话**

  每次都将保存所有会话。这种方式为正确保存分布式 WEB 应用的会话提供了最好保证。这种方式简单，内存复制和数据库存储方式都默认采用这种方式。

- **修改过的会话**

  当会话被修改过后，则备份整个会话。当“session.setAttribute()”或 “session.removeAttribute()”被调用后，则认为会话被修改过。必须保证这些方法调用才修改会话属性，这并不是 J2EE 规范后要求的。但却是正确实现这种方法所需要的。备份修改过的会话减少了会话存储的数量，那些仅仅是读取会话属性的请求将不会触发会话备份的动作。这将带来比备份整个会话更好的性能。

- **修改过的属性**

  仅仅是保存被修改过的会话属性而不是整个会话。这将最小化备份的会话数据。这种方式带来最好的性能及最小的网络通信。为了保证这种方式工作的正确性，必须依据以下的要点。首先，只能通过调用“setAttribute()”方法修改会话状态，并且会话数据要被序列化和备份。其次，确保属性之间没有交叉引用。每个唯一的属性键值下的对象图应当被独立地序列化和保存。如果两个独立的键值下的对象存在交叉引用，它们将不能够正确的序列化和反序列化。例如图 13 所示，在一

个内存复制的集群中，会话中存有“student”和“school”对象，同时“school”对象含有到“student”对象的引用，某一个时候“school”被修改后备份到备份服务器中。在序列化和反序列化之后，在备份服务器的被还原的“school”对象的版本将包含整个对象图，包括其引用的“student”对象。但是“student”对象可以被独立的修改，当它被修改后，仅仅只有它自己被备份。在序列化和反序列化之后，在备份服务器中还原“student”对象，但在此时，它将丢失与“school”对象的连接。尽管这种方式有最好的性能，但上述限制将影响 WEB 应用程序的架构和设计。特别是需要用会话保存缓存的复杂的用户数据。

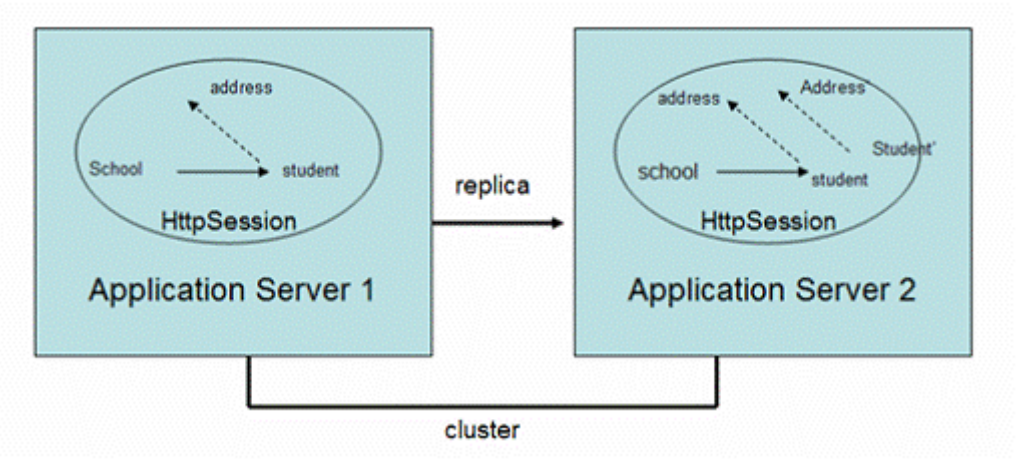

图 13 会话复制中的交叉引用

8. 其他失效转移的实现

如前面章节所提到的，当会话备份时粒度对于性能非常重要。然而，当前的一些实现，不管是数据库存储还是内存复制，都将使用 Java 对象序列化技术来传输 Java 对象。这就打了个大印子，影响系统的性能，并给 WEB 应用程序的架构和设计带来很多的限制。一些 J2EE 供应商便寻找一些特别的手段来更为轻量地，小印子地实现 WEB 集群，提供细粒度的分布式对象共享机制用于提高集群的性能。

Jrun的Jini技术

Jrun4 将它的集群解决方案构在Jini技术之上。Jini为分布式计算而生，它允许在一个单一的分布式计算空间内创建“联合”的设备或组件。Jini提供查找，注册，租用等分布式系统服务，这对集群环境非常有用。另一种称为JavaSpace的技术构建于Jini之上，它提供了一些用于实现集群非常有价值的特性，如对象处理，共享，迁移等。要了解有关Jini和JavaSpace更多的信息，请参考http://java.sun.com/products/jini/2_0index.html。

Tangosol的分布式缓存

Tangosol Coherence提供了一个分布式数据管理平台，它可以嵌入到大多数流行的J2EE服务器中用于实现集群环境。Tangosol Coherence同时也是提供了分布式缓存系统用于在不同的 JVM 之间有效地共享 Java 对象。要了解有关 Tangosol 更多的信息，请参考 http://www.tangosol.com。

9. JNDI 集群实现

J2EE 规范要求所有的 J2EE 容器必须提供 JNDI 规范的实现。JNDI 在 J2EE 应用程序中的主要角色是用来提供一个间接层，这样资源可以很容易被找到，而不用关心细节。这使得 J2EE 组件更加可复用。

拥用全特性的集群的 JNDI 对于 J2EE 集群是非常重要的。所有的 EJB 调用都开始于在 JNDI 树上查找它的 Home 接口，J2EE 供应商根据他们的集群结构采用不同的方式实现 JNDI 集群。

9.1. 共享全局 JNDI 树

WebLogic 和 Jboss 都有一个全局的，共享的，集群范围的 JNDI Context 供客户端查找和绑定对象，绑定的全局 JNDI Context 中对象将通过 IP 广播的方式在集群中复制，这样当一台服务器实例停机后，被绑定的对象仍然可供查找。

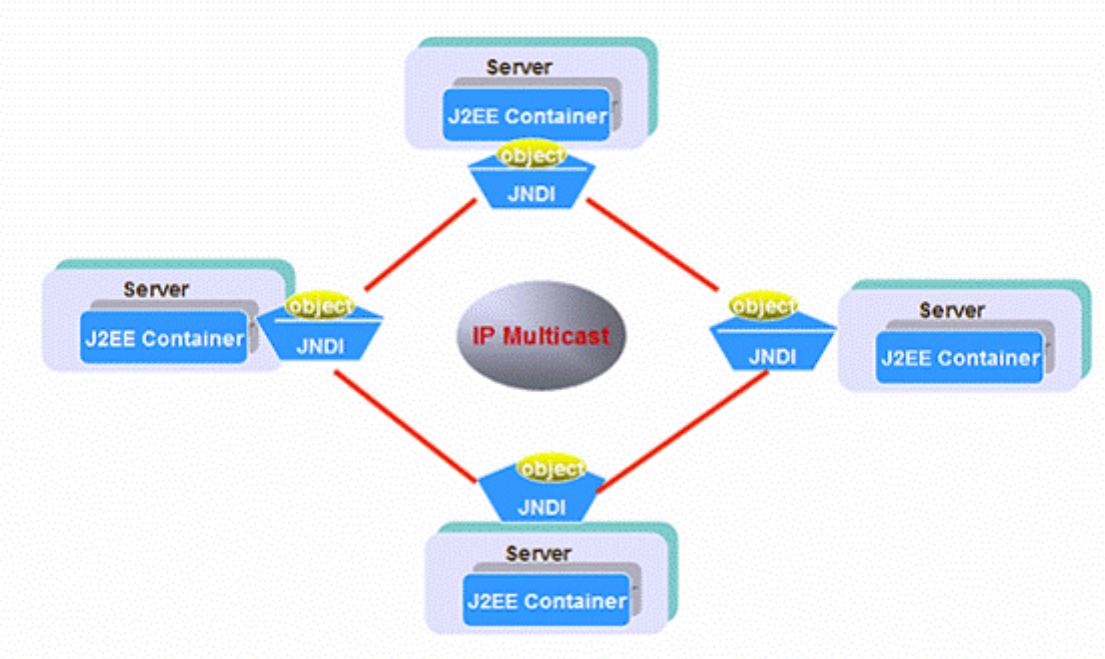

图 14 共享的全局 JNDI

如图 14 所示，共享的全局 JNDI 树实际包括了所有本地 JNDI 结点上绑定的对象。集群上每个结点都拥有自己的名称服务器，它们与集群中其他服务器相互复制所有的东西。这样每个名称服务器上都拥有其他名称服务器对象树的拷贝。这种冗余结构使得全局 JNDI 树高可用。

实际上，集群的 JNDI 树可以被用做两个目的。可以被管理员用来部署对象和服务。在一台服务中部署完 EJB 模块或 JDBC&JMS 服务后，JNDI 树上的所有对象都将复制到其他服务器实例中。在运行期，程序可以 JNDI API 访问 JNDI 树存储或者获取对象，这些对象也将被全局复制。

9.2. 独立 JNDI

Jboss 和 WebLogic 都采用了共享全局 JNDI 树的方式，Sun JES 和 IBM WebSphere 等采用了每个服务器都拥有独立的 JNDI 树的方式。在使用独立 JNDI 树的集群中，成员服务器不必知道或关心集群中其他服务器。这是否意味着不想把 JNDI 集群呢？所有 EJB 访问都开始于在

JNDI 树上查找它们的 Home 接口，如果没有可集群的 JNDI 树，集群就根本无用。

实际上，如果 J2EE 应用程序是相似的，独立的 JNDI 树仍然可以获得高可用性。当集群中所有实例都有相同的设置以及都部署相同的应用程序集，我们说集群是“相似的”，这种情况下，一种被称为代理的特殊管理工具可以帮助我们获取高可用性，如图 15 所示：

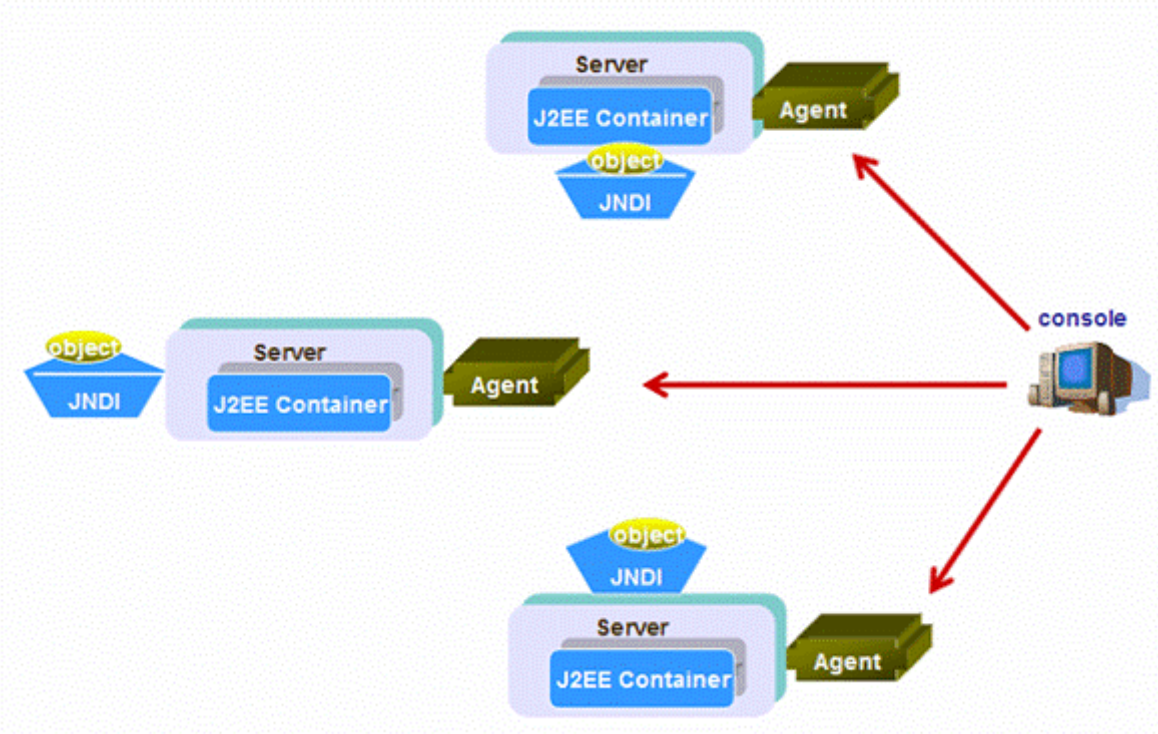

图 15 独立 JNDI

不管是 Sun JES 还是 WebSphere 都在集群的实例上安装了结点代理，当部署 EJB 模块或绑定其他 JNDI 服务，管理控制员可以向所有的代理发出命令，以此实现与共享全局 JNDI 相同的效果。

但是独立 JNDI 的方案不能支持复制在运行期绑定或获取的对象。有以下几个原因：JNDI 在 J2EE 应用程序中的主要角色是用来提供管理外部资源的间接层，并不是用来做数据存储。如果有这样的需求，可以采用具有 HA（高可用性）特性的独立的 LDAP 服务器或数据库。Sun 和 IBM 自己都有这样拥有集群特性的独立的 LDAP 服务器产品。

9.3. 中心 JNDI

少数 J2EE 产品使用了中心 JNDI 方案，这种方案中名称服务器驻留在单一服务器中，所有的服务器实例都注册它们相同的 EJB 组件和管理对象到单一的服务器中。

名称服务器实现了高可用性，这对客户端是透明的。所有的客户端都在这台服务器中查找 EJB 组件，但是这种结构意味着复杂的安装和管理，慢慢被多数供应商抛弃。

9.4. 初始化访问 JNDI 服务器

当然客户端要访问 JNDI 服务器的时候，它们需要知道远程 JNDI 服务器的主机名/IP 地址和端口，在全局或是独立 JNDI 树的方案中，有多个 JNDI 服务器。客户端第一次访问时应该连

接哪个呢？如何获得负载均衡和失效转移呢？

从技术上说，一个软件或硬件负载均衡器可以设在远程客户端和所有的 JNDI 服务器之间执行负载均和失效转移。但是，很少有供应商实现这种方式，这里有些简单的方案：

- Sun JES 和 Jboss 实现 JNDI 集群是在“java.naming.provider.url”属性中设置一列用逗号分隔的 URL，如 java.naming.provider.url=server1:1100,server2:1100:server3.1100:server4.1100 客户端将挨个联系列表中的服务器，一旦其中一个响应了便中止。
- Jboss 同时也实现了自动发现的特性，当设置属性串“java.naming.provider.url”为空时，客户端将试图通过网络广播来发现在一个 JNDI 服务器。

10. EJB 集群实现

EJB 是 J2EE 技术中重要的部分，并且 EJB 集群是实现 J2EE 集群最大的挑战。

EJB 技术是为分布式计算而生。它们可以在独立的服务器中运行。Web 服务器组件或富客户端可以从其他的机器通过标准协议（RMI/IIOP）来访问 EJB。你可以象调用你本地 Java 对象的方法一样调用远程 EJB 的方法。实际上，RMI/IIOP 完全掩盖了你正在调用的对象是本地的还是远程的，这被称作本地/远程透明性。

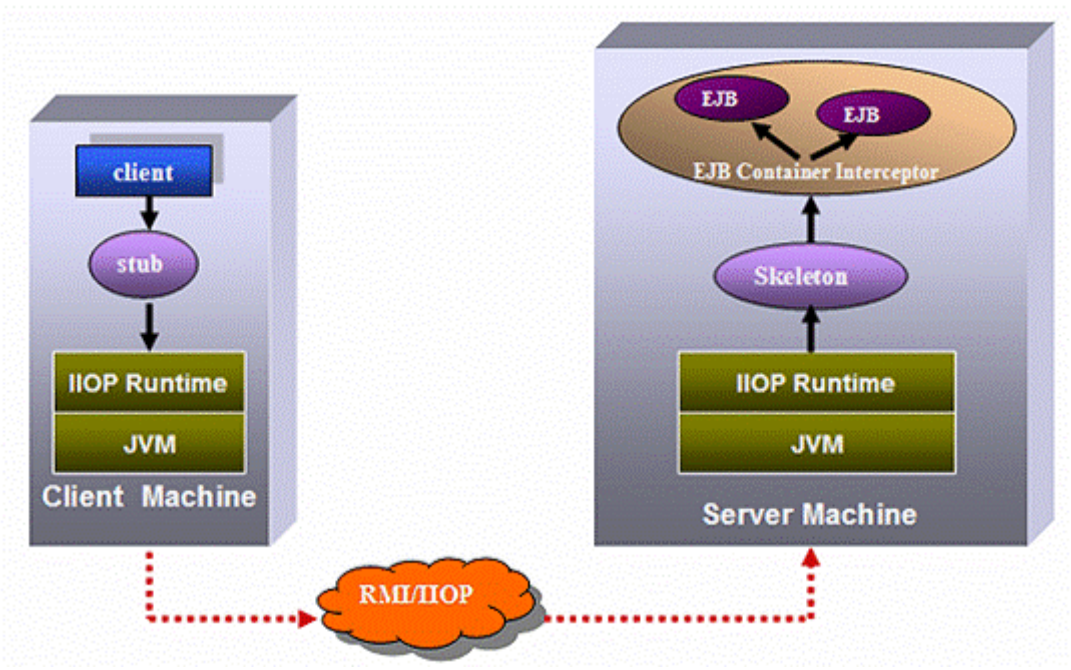

图 16 EJB 调用机制

上图显示了远程EJB的调用机制。当客户端想使用EJB，它不能直接调用，相反，客户端只能调用一个被称为“stub”的本地对象，它扮演了到远程对象代理的角色，并且有远程对象相同的接口。这个对象负责接受本地方法调用，并且这些调用通过网络代理到远程EJB。这些对象在客户JVM中运行，并且知道如何通过RMI/IIOP跨过网络查找真正的对象。要了解有关EJB更多的信息，请参考http://java.sun.com/products/ejb。

为解释 EJB 集群的实现，我们先看看在 J2EE 代码中如何使用 EJB 的。为了调用 EJB，我

们需要

- 在 JNDI 服务器中查找 EJBHome stub
- 使用 EJBHome stub 查找或创建 EJB 对象，这样获得一个 EJBObject stub
- 在 EJBObject stub 上调用方法

负载均衡和失效转移可以在 JNDI 查找时发生，这我们已在上面阐述了。在 EJB stub（包括 EJBHome 和 EJBObject）的方法调用时，供应商采用以下三种方式实现 EJB 负载均衡和失效转移。

10.1. 智能存根（Smart stub）

正如我们所知，客户端可以通过存根对象（stub）来访门远程的 EJB，这个对象可以通过 JNDI 树获取，甚至客户端可能透明地从 WEB 服务器上下载存根类文件。

这样存根就可以在运行期动态地用程序生成，而其类文件也不必在客户端环境的 classpath 或库中。（因为它是可以被下载的）

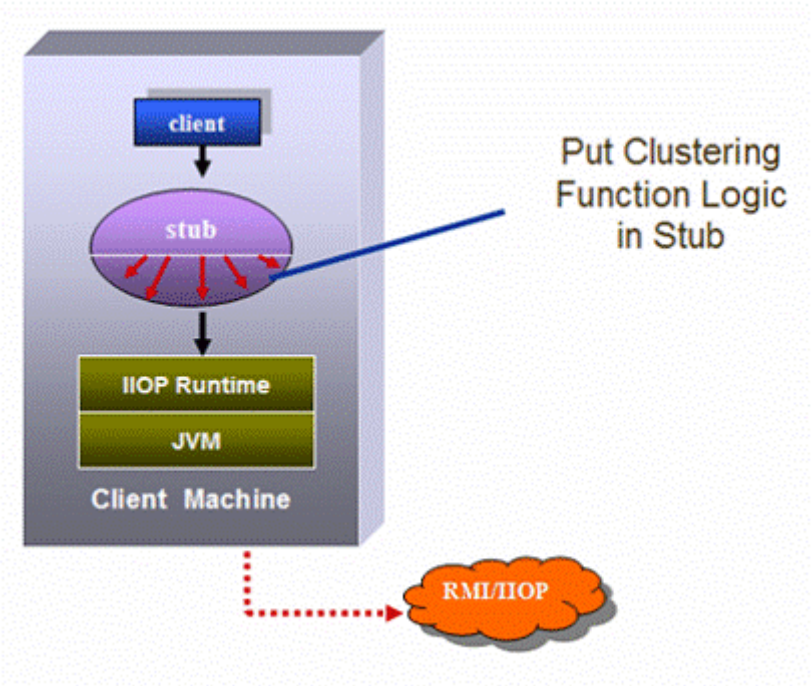

图 17 智能存根

如图 17 所示，BEA WebLogic 和 Jboss 通过在存根代码中组合几种行为来实现 EJB 集群，而这些都是在客户端透明运行的（客户端不需要了解这些代码）。这种技术叫做智能存根。

智能存根确实很聪明，它包含一组它可以访问的目标实例，可以检测这些实例的任何失效，它也包含了复杂的负载均衡和失效转移的逻辑，用来分发请求到目标实例。而且，如果集群拓朴改变了的话（比如新增或删除了服务器实例），存根可以更新它的目标实例清单来反映新的拓朴，而不需要手工重新配置。

使用智能存根实现集群有以下优点：

- 因为 EJB 存根运行在客户端，所以它可以节省许多服务器资源。
- 负载均衡是在客户端代码中，并且与客户端的生命周期相关。这样就消除了负载均衡

器的单点失效。如果负载均衡器失效了，就意味着客户端也失效了，这种情况是能接受的。

- 存根可动态的自动下载更新，这意味着零维护。

10.2. IIOP 运行期库

Sun JES Application Server 使用了另一种方法实现 EJB 集群。负载均衡和失效转移逻辑是在 IIOP 运行库中实现的。比如，JES 修改了“ORBSocketFactory”的实现，使它能支持集群。如图 18 所示。

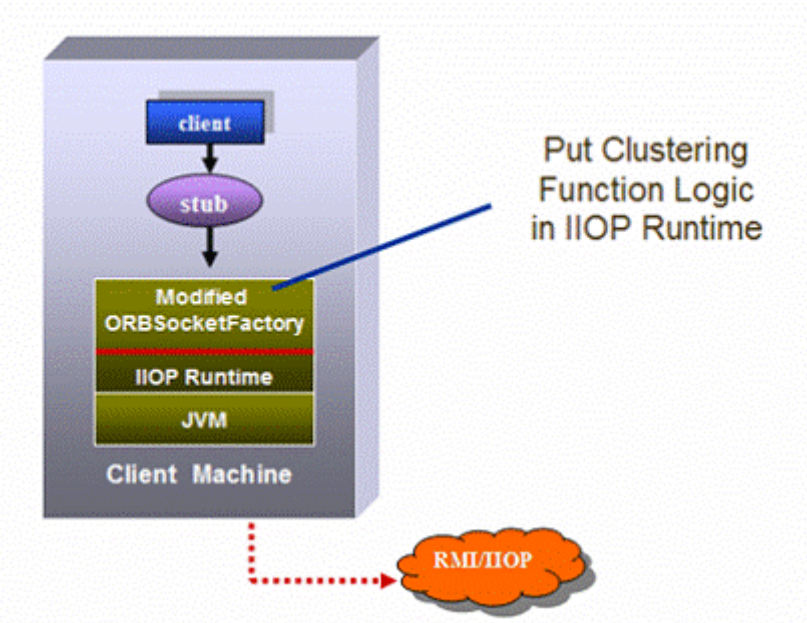

图 18 IIOP 运行期

“ORBSocketFactory”的修改版有实现负载均衡和失效转移的所有逻辑和算法，这样就保持了存根干净和小。因为是在运行库中实现的，这样就比存根的方式更容易获得系统资源。但这种方式需要在客户端运行一个特殊的运行库，这样就给与其他 J2EE 产品进行互操作时带来了麻烦。

10.3. 拦截器代理

IBM Webshpere 采用了定位服务精灵（Location Service Daemon-LSD）,它对 EJB 客户端扮演了拦截器代理的角色，如图 19 所示。

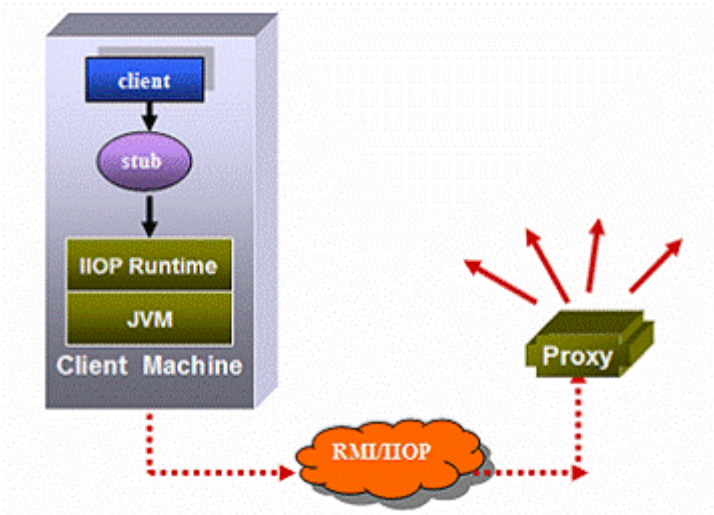

图 19 拦截器代理

在这种方式中，客户端通过 JNDI 查找获取存根，这个存根将信息路由到 LSD，而不是运行了 EJB 的应用程序服务器。这样 LSD 接收到所有进来的请求，根据负载均衡和失效转移的策略来判断应该将它发送到哪个服务器实例。这种方式增加的安装和维护集群的额外的管理工作。

10.4. EJB 的集群支持

要调用 EJB 的方法，要涉及到两种存根对象，一是 EJBHome 接口，另一个是 EJBObject 接口。这意味着 EJB 潜在的需要在两层上实现在负载均衡和失效转移。

- 当客户端使用 EJBHome 存根创建或查找 EJB 对象时
- 当客户端调用 EJB 对象上的方法时。

EJBHome存根支持集群

EJBHome 接口用于在 EJB 容器中创建和查找 EJB 实例，而 EJBHome 存根是 EJBHome 接口的客户端代理。EJBHome 接口不需维持任何客户端的状态信息。这样，来自不同 EJB 容器中的相同 EJBHome 接口对于客户端来说是一致的。当客户端执行 create()或 find()调用的时候，home 存根根据负载均衡和失效转移算法从多个相同的服务器实例中选择一个，并将调用发送到该服务器的 home 接口上。

EJBObject存根支持集群

当 EJBHome 接口创建一个 EJB 实例，它将返回 EJBObject 的存根给客户端，并让用户调用 EJB 的业务方法。集群中已有一组可用的服务器实例，并在上面的部署了 bean，但是不能将 EJBObject 存根对象向 EJBObject 接口发出调用转发到任意一个服务器实例上，这取决于 EJB 的类型。

无状态会话 Bean 是最简单的情况，因为不涉及到状态，所有的实例都可以认为是一样的，这样对 EJBObject 的方法调用可负载均衡和失效转移到任意一个服务器实例上。

集群的有状态会话 Bean 与无状态会话 Bean 有一点不同，正如我们所知，有状态会话 Bean 对于客户端连续的请求会持有状态信息。从技术上来说，集群有状态会话 Bean 与集群 HTTP Session 是一样的。在常规时间，EJBObject 存根将不会把请求负载均衡到不同的服务器实例。相反，它将胶粘到最初创建的服务器实例上，我们称这个实例为“主实例”。在执行过程中，

会话状态会从主实例备份到其他服务上去。如果主实例失效了，备份服务器将接管它。

实体 Bean 本质上是无状态的，尽管它可以处理有状态的请求。所有的信息都将通过实体 Bean 本身的机制备份到数据库中。这样看起来实体 Bean 可以象无状态会话 Bean 一样很容易获取负载均衡和失效转移。但实际，实体 Bean 多数情况下是不负载均衡和失效转移的。根据设计模式的建议，实体 Bean 被会话外观包装。这样，多数对实体 Bean 的访问都是进程内会话 Bean 通过本地接口完成的，而不是远程客户端。这使得负载均衡和失效转移没有意义。

11. JMS 和数据库连接的集群支持

除 JSP，Servlet，JNDI 和 EJB 之外，在 J2EE 中还有其他的分布式对象。这些对象在集群的实现中可能支持，可能不支持。

当前，一些数据库产品，如 Oracle RAC，支持集群环境并可以部署到多复制，同步的数据库实例中。然而，JDBC 是一个高度状态化的协议并且它的事务状态直接与客户端和服务器的 Socket 连接绑定，所以很难获取集群能力。如果一个 JDBC 连接失效了，与该失效连接相关的所有 JDBC 对象也就失效了。客户端代码需要进行重连的动作。BEA WebLogic 使用 JDBC 多池（multipool）技术来简化这种重连过程。

JMS 被多数 J2EE 服务器所支持，但支持得并不完全，负载均衡和失效转移仅仅被 JMS 代理所实现，很少有产品在 JMS Destination 中的消息有失效转移的功能。

12. J2EE 集群的神话

12.1. 失效转移可以完全避免错误——否定

在 Jboss 的文档中，整个章节都在警告你“你真的需要 HTTP 会话的复制吗？”。是的，有时没有失效转移的高可用性的解决方案也是可接受并且是廉价的。失效转移并不是你想象的那么强壮。

那么失效转移到底给你带来了什么？你可能想失效转移可以避免错误。你看，没有会话的失效转移，当一个服务器实例失效后，会话数据将丢失而导致错误。通过失效转移，会话可以从备份中恢复，而请求可以被其他服务器实例所处理，用户根本意识不到失效。这是事实，但这是有条件的！

回想一样我们定义的“失效转移”，我们定义了一个条件是失效转移是在“两个方法调用之间”发生的。这是说如果你有两个连续的对远程对象的方法调用，失效转移只会在第一调用成功后并且第二调用的请求发出前才能发生。

这样，当远程服务器在处理请求的过程中失效了会发生什么呢？答案是：多数情况处理将会停止而客户端将会看到错误信息。除非这个方法是等幂的（Idempotent），只有这个方法是等幂的，一些负载均衡器更智能，它会重试这些方法并将它失效转移到其他实例上。

为什么“等幂”重要呢，因为客户端不知道当失效发生的时候请求被执行到什么地方。是才刚刚初始化还是差不多就要完成了？客户端没法判断！如果方法不是等幂的，在相同方法上的两次调用可能会两次修改系统的状态，而使得系统出现不一致的情形。

你可能想所有在事务中的方法都是等幂的，毕竟，如果错误发生，事务将被回滚，事务状态的改变都将被复位。但事实上事务的边界可能不包括所有的远程方法调用过程。如果事务已经在服务器上提交了而返回给客户端时网络崩溃怎么办呢？客户端不知道服务器的事务是否是成功了。

在一些应用程序中，将所有的方法都做成等幂的是不可能的。这样，你只能通过失效转移减少错误，而不是避免它们。拿在线商店为例，假设每台服务器可以同时处理 100 个在线用户的请求，当一台服务器失效了，没有失效转移的解决方案将丢失 100 个用户的会话数据并激怒这些用户。而有失效转移的解决方案中，当失效发生的时候有 20 个用户正在处理请求，这样 20 个用户将被失效激怒。而其他 80 个用户正处于思考时间或在两个方法调用之间，这些用户可以透明地获得失效转移。这样，你就需做以下的考虑：

- 激怒 20 个用户和激怒 100 个用户造成影响的区别。
- 采用失效转移和不采用失效转移产品成本的区别

12.2. 独立应用可以透明的迁移到集群结构中——否定

尽管一些供应商宣称他们的 J2EE 产品有这样的灵活性。不要相信他们！事实你要在开始系统设计时就要准备集群，而这将影响开发和测试的所有阶段。

Http Session

在集群环境中，如我前面提到的，使用 HTTP Session 有很多限制，这取决于你的应用程序服务器采用了那种会话失效转移的机制。第一个重要的限制就是所有保存的 HTTP Session 中的对象必须是可序列化的，这将限制设计和应用程序结构。一些设计模式和 MVC 框架会用 HTTP Session 保存一些不序列化的对象（如 ServletContext，EJB 本地接口和 WEB 服务引用），这样的设计不能在集群中工作。第二，对象的序列的反序列化对性能的影响很大，特别是数据库保存的方式。在这样的环境中，应该避免在会话中保存大的或是众多的对象。如果你采用了内存复制的方式，如前所述你必须小心在会话中属性的交叉引用。其他在集群环境中的主要区别是在会话不管任何属性修改，你必须调用“setAttribute()”方法。这个方法调用在独立的系统中是可选的。这个方法的目的是区别已修改的属性和那些没用到属性，这样系统可以只为失效转移备份必要的数据，从而提高性能。

缓存

我经历过的大多数 J2EE 项目都用了缓存来提高性能，同时流行的应用程序服务器也都提供了不同程度的缓存用来加快应用程序的速度。但这些缓存都是为那些典型的独立环境设计的，只能在单 JVM 实例中工作。我们需要缓存是因为一些对象很“重”，创建它需花费大量的时间和资源。因此我们维护了对象池用于重用这些对象，而不需要在后面创建。我们只有当维护缓存比创建对象更廉价时才能获得性能的提高。在集群环境，每个 JVM 实例都要维护一份缓存的拷贝，这些拷贝必须同步以维持所有服务器实例状态的一致性。有时这种类型的同步会比没有缓存带来更糟的性能。

Static变量

当我们设计 J2EE 应用程序时，在架构上经常会使用一些设计模式。这些如“Singleton”的设计模式会用到静态变量来在多对象之间共享状态。这种方式在单服务中工作得很好，但在集群环境将失效。集群中的每个实例都会在它的 JVM 实例中维护一份静态变量的拷贝，这样就破坏了模式的机制。一个使用静态变量的例子就是用它来保持在线用户数。用静态变量来保存在线用户数是一个很简单的办法，当用户进入或离开时就增加和减少它。这种方式在单服务器

中绝对是好的，但在集群环境将失效。在集群中更好的方式是将所有状态保存到数据库。

外部资源

尽管 J2EE 规范不支持，但为各种目的仍然会用外部 I/O 的操作。例如，一些应用会使用文件系统来保存用户上传的文件，或是创建一个动态配置的 XML 文件。在集群环境是没有办法来在其他实例之间来复制这些文件的。为了在集群中工作，办法是用数据库作为外部文件的存放点，另外也可以使用 SAN（存储区域网，Storage Area Network）作为存放点。

特殊服务

一些特殊的服务只在独立的环境中才有意义，定时服务就一个很好例子，这种服务可以在一个固定的间隔时间有规律的触发。定时服务常用于执行一些自动化管理任务。如日志文件滚动，系统数据备份，数据库一致性检查和冗余数据清理等。一些基于事件的服务也很难被迁移到集群环境中。初始化服务就是个好例子，它只在整个系统启动时才发生。邮件通知服务也一样，它在一些警告条件下触发。

这些服务是被事件而不是被请求触发的，而且只被执行一次。这些服务使得负载均衡和失效转移在集群中没多少意义。

一些产品准备了这些服务，如 Jboss 使用了“集群单例设施”来协调所有实例，保证执行这些服务一次且仅有一次。基于你所选择的平台，一些特殊的服务可能会是把你的应用迁移到集群结构中的障碍。

12.3. 分布式结构比并置结构更灵活——不一定

J2EE 技术，尤其是 EJB，天生就是用来做分布式计算。解耦业务功能，重用远程组件，这些使得多层应用非常流行。但是我们不能将所有的东西都分布。一些 J2EE 架构师认为 Web 层与 EJB 层并置得越近越好。这些计论后面会继续。先让我解释一下。

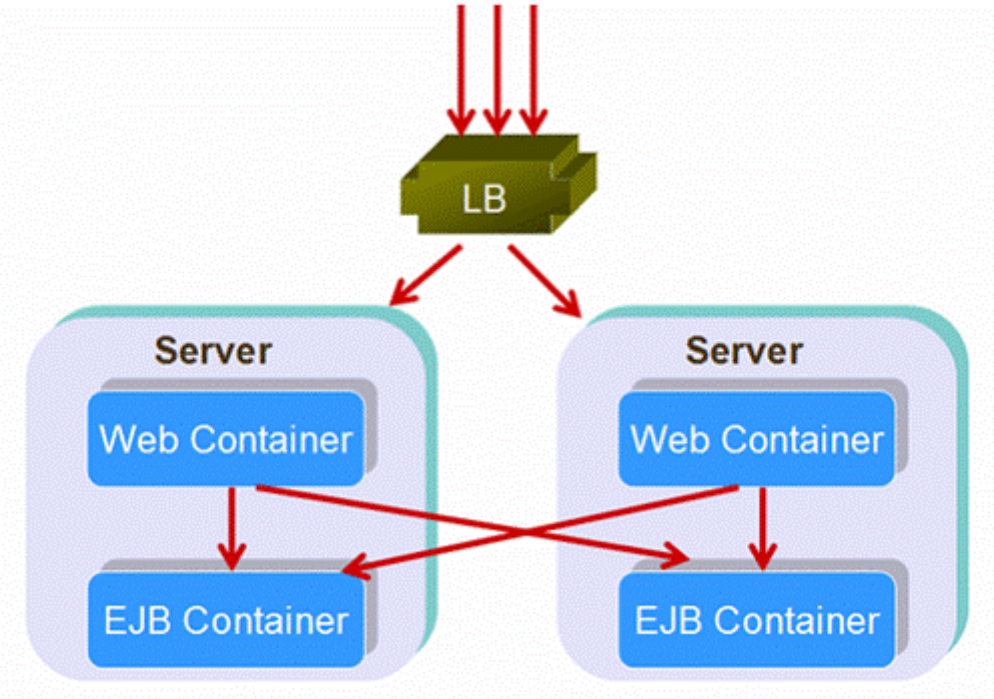

图 20 分布式结构

如图 20 所示，这是一个分布式结构。当请求来了，负载均衡器将请求分发到不同服务器中的不同 WEB 容器，如果请求包含了 EJB 调用，WEB 容器将重发 EJB 调用到不同的 EJB 容器。这样，请求将被负载均衡和失效转移两次。

一些人看分布式结构，他们会指出：

- 第二次负载均衡没有必要，因为它不会使任务分配更平坦。每个服务器实例都有它们自己的 WEB 容器和 EJB 容器。把 EJB 容器用来处理来自其他实例 WEB 容器的请求比只在服务器内部调用并没有什么优势。
- 第二次失效转移没有必要，因为它不能是高可用性。多数供应商实现 J2EE 服务器都会在同一服务器中运行的 WEB 容器和 EJB 容器放在一个 JVM 实例中。如果 EJB 容器失效了，多数情况下在同一个 JVM 中的 WEB 容器也将同时失效。
- 性能将下降。想像一下对你的应用的一次调用包含一组对 EJB 的调用，如果你负载均衡了这些 EJB，这将跨越每个服务器实例，导致许多不必要的服务器到服务器的交互。还有，如果这个方法在事务范围内，那么事务边界将包含许多服务器实例，这将严重影响性能。

实际上在运行期，多数的供应商（包括 Sun JES，WebLogic 和 Jboss）都会优化 EJB 调用机制，使请求首先选择同一个服务器中的 EJB 容器。这样，如图 21 所示，我们只在第一层（WEB 层）做负载均衡，然后调用相同服务器上的服务。这种结构我们称之为并置结构。技术上说，并置结构是分布式结构的一种特例。

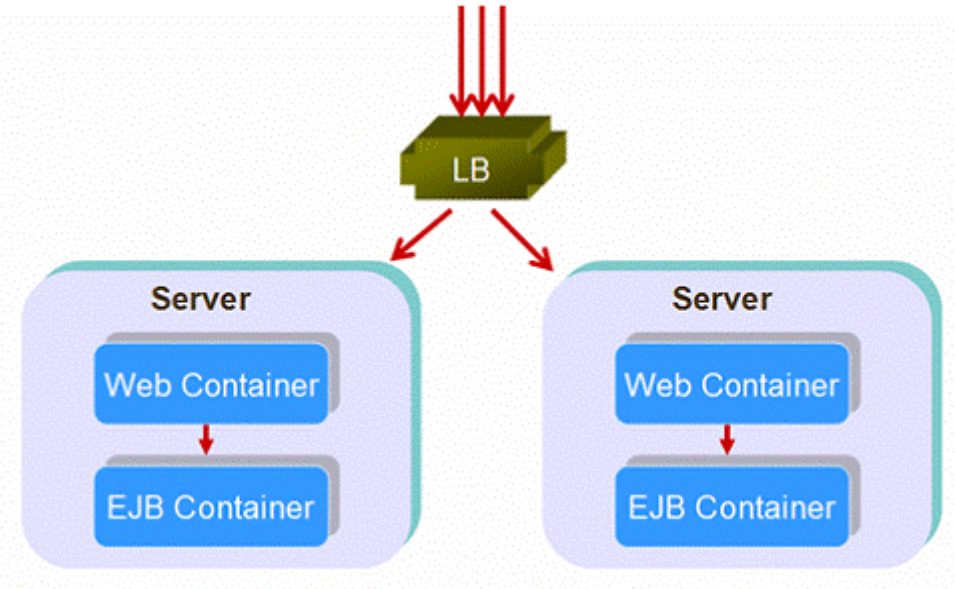

图 21 并置结构

一个有趣的问题是，既然多数的部署在运行期都演进成了并置结构，为什么不用本地接口代替远程接口，这将大提高性能。你当然可以，但是记住，当你使用本地接口后，WEB 组件和 EJB 耦合得很紧，而方法调用也是直接的而不通过 RMI/IIOP。负载均衡和失效转移分发器没有机会介入本地接口调用。“WEB+EJB”整体处理负载均衡和失效转移。

但不幸的是，在集群中使用本地接口在多数 J2EE 服务器中有局限性。使用本地接口的 EJB 是本地对象，是不可序列化的，这一个限制就使本地引用不能保存在 HTTP Session 中。一些产

品，如 Sun JES，会将本地接口区别看待，使它们可以序列化。这样就可以用在 HTTP Session 中。

另一个有趣的问题是，既然并置结构这么流行并且有好的性能，为什么还要分布式结构呢？这在多数情况下是有道理的，但有时分布式结构是不可替代的。

- EJB 不仅被 WEB 容器使用，富客户端也会使用它。
- EJB 组件和 WEB 组件需在不同的安全级别上，并需要物理分离。这样防火墙将被设置用于保护运行 EJB 的重要机器。
- WEB 层和 EJB 层极端不对称使得分布式结构是更好的选择。比如，一些 EJB 组件非常复杂并且很消耗资源，它们只能运行在昂贵的大型服务器上，另一方面，WEB 组件（HTML，JSP 和 Servlet）简单得只需廉价的 PC 服务器就能满足要求。在这种情况下，专门的 WEB 服务器可以用来接受客户端连接请求，很快处理静态数据（HTML 和图像）和简单的 WEB 组件（JSP 和 Servlet）。大型服务器只被用来做复杂计算。这将更好的利用投资。

13. 结论

集群与独立环境不同，J2EE 供应商采用不同的方法来实现集群。如果你的项目为做到高伸缩性而使用集群，你应该在你的项目开始的时候就做准备。选择符合你的需求的正确的 J2EE 产品。选择正确的第三方软件和框架并确保它们能支持集群。最后设计正确的架构使得能从集群中受益而不是受害。