第34卷 第10期 计 算 机 工 程 2008年5月
Vol.34 No.10 Computer Engineering May 2008
·软件技术与数据库· 文章编号：1000—3428(2008)10—0033—02 文献标识码：A 中图分类号：TP31

关键任务系统中的任务热插拔技术

叶海智，王慧强，赖积保
(哈尔滨工程大学计算机科学与技术学院，哈尔滨 150001)

摘 要：服务连续性和自适应性是许多关键任务系统的客观要求，为此提出采用任务热插拔的方法予以实现。介绍任务热插拔的基本思想，分析实现任务组件插入和替换的关键技术，总结了任务热插拔的技术实现路线，并开展了初步的实验研究。结果表明，该方法实现了任务组件的动态替换，可用于提高关键任务系统的可用性及适应性。

关键词：任务热插拔；运行监测；动态装载；静态建立；动态插入

Mission Hot-swapping Technique in Mission-critical System

YE Hai-zhi, WANG Hui-qiang, LAI Ji-bao
(College of Computer Science and Technology, Harbin Engineering University, Harbin 150001)

【Abstract】 Service continuity and self-adaptive capability are the objective requests for many mission-critical systems, so mission hot-swapping technology is proposed. The basic principle of mission hot-swapping is introduced, and key technologies to implement interposition and replacement of the mission components are analyzed. Technical routes of mission hot-swapping are summarized and elementary experimental research is carried out. The experimental results show that the method has implemented the dynamic replacement of the mission components, and it can be used to improve the availability and adaptability of the mission-critical systems.

【Key words】 mission hot-swapping; running monitoring; dynamic loading; quiescent state establishment; dynamic interposition

在软件的更新和演化方面，研究人员已经开展了一定的研究工作。文献[1]提出了运行于一个扩展虚拟机上的动态更新组件系统DUCS，具备了一定的组件更新能力，但其扩展虚拟机实现复杂，缺乏对环境的自适应能力；适应性代码[2]虽能对运行环境的变化作出积极反应，但代码复杂度高、性能开销大。据此，本文提出采用任务热插拔技术实现任务组件的动态插入和替换，以满足关键任务系统高可用性的需求。

1 任务热插拔及其关键技术实现

任务热插拔是针对关键任务系统的可用性而提出的。它将完成某任务的应用软件分为若干个相对独立、能完成特定子任务的功能组件，这些功能组件简称任务组件。在系统运行中通过对运行环境的监测和诊断，动态地进行任务组件的插入和替换，从而实现系统的可用性和对环境的适应性。

插入指在已有的任务组件之间插入新的组件。当系统出现错误或环境发生变化时，通过监测代码和诊断代码的插入来检测错误产生原因，或通过任务组件的插入来实现系统功能任务的扩展。

替换指在运行时将控制应用程序资源的活动任务组件失效，取而代之的是另一个任务组件，或用新的任务组件来升级旧的组件。

任务热插拔在提高系统性能、可用性、适应性及可维护性等方面有许多潜在的优势。如，通常系统为了支持多种适应性策略，必须实现所有可供选策略的组件，这无疑增加了系统在性能监测和调试方面的复杂性。而应用任务热插拔，每条策略可作为分离的、独立的组件来实现，并根据需要对其进行动态插入或替换，这种关键点的分离既保证了对运行环境的适应性，大大简化软件的整体结构，也提高了系统的可用性。

需提供不间断服务的关键任务系统大致可分为科学计算和服务器2类系统。这2类系统虽然实现的目标不同，但都有提供服务固定、运行状态具有相当规律性的特点。任务热插拔的实现，首先是根据这些特点来判定系统运行环境的变化问题；其次是对这些变化作出适应性反应，也就是任务组件的插入和替换。要解决上述问题就必须对任务热插拔中的一些关键实现技术进行深入研究，如运行监测、动态加载和静态建立机制等。

1.1 运行监测

对系统的运行监测是实现任务热插拔的基础。支持任务热插拔的关键任务系统虽然需要对运行环境变化作出积极响应以提高性能，但并非将所有的任务组件一次性全部装入系统中，而是根据对运行环境的监测结果动态地进行加载，从而避免给系统的运行带来较大的运行开销，这也是任务热插拔同自适应代码的重要区别。

监测是追踪运行环境的变化，并通过插入更详细的监测代码来判断和触发任务组件的热插拔。对需提供不间断服务的关键任务系统而言，由于其往往提供几种固定的服务，因此监测目标相对容易确定，如监测目前任务组件能否正常提供或满足所需的服务或某些资源的使用是否超过其规定门限等。

基金项目：高等学校博士学科点专项科研基金资助项目(20050217007)
作者简介：叶海智(1963－)，男，副教授、在职博士研究生，主研方向：计算机网络，可信分布式系统；王慧强，教授、博士生导师；赖积保，博士研究生
收稿日期：2007-06-07 **E-mail：**yhz87@163.com

1.2 动态装载

要实现任务组件的插入或替换以及监测代码的插入，就需要一种动态装载技术来进行组件或代码的动态加载。当今许多应用软件都是基于 Java 进行设计和开发的，而 Java 自身的类装载机制就为组件的动态装载提供了有力的技术支持。Java 虚拟机中的类装载器负责装载一个程序的所有代码类，并根据需要动态装载这些类，而不是一开始就完全装载这些类。它对类的查找和装载通过使用一种代理机制(delegation)来进行，当 Java 虚拟机 JVM 要求类装载器装载一个类时，该类装载器首先将请求转发给他的父装载器，只有当父装载器没有装载或无法装载这个类时，该类装载器才可以装载这个类。这样的装载机制既保证每个 Java 类在运行时都由一个装载器对其进行查找和动态加载，也有效地解决了类装载中的安全和效率问题。

1.3 动态插入

动态插入是任务热插拔的关键实现技术之一，它在关键任务系统的运行监测和系统功能扩展等方面具有重要作用。动态插入的一种可行方案是通过由包装器Wrapper和拦截器Interposer组成的中间调节对象来实现[3]。包装器充分利用了所有访问都要通过组件公共接口的特性，是插入在特定目标组件接口及其实现之间并封装了代码的功能单元。拦截器与包装器类似，但它实现了通用的接口。由于包装器可根据需要封装不同的代码，为组件的升级和功能的扩展创造了条件，而拦截器实现的通用接口又为组件的插入提供了方便。

例如，当需要插入新组件时，可将拦截器和封装了组件的包装器通过动态装载技术进行插入。由于拦截器提供了通用的接口，因此其他组件可通过该接口对新组件进行访问，并按顺序号进行。同样通过包装器封装新的代码可对目前的组件代码进行扩展和升级。其原理如图 1 所示。此外，包装器也可根据需要封装不同的函数或方法，用于对组件调用前后的状态进行追踪(如，Precall, Poscall 函数可记录当前通过拦截器访问组件的引用计数，组件替换时阻塞请求调用等)。

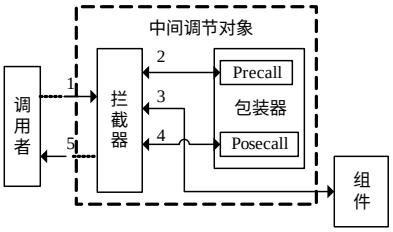

图 1　插入过程

1.4 静态建立

组件替换是任务热插拔中的最重要环节，涉及请求缓冲(request buffering)、状态迁移(state transfer)和请求重定向(request redirection)等方面内容[4]。其中状态迁移是有状态组件(如Java 会话EJB)进行替换的关键，它是指把老组件的状态(如程序栈、对象属性值等)拷贝到新组件中去以保证整个软件运行结果的一致性。而为了实现组件的状态迁移，就必须首先使被替换组件处于静止状态(quiescent state)，即所有对该组件的访问活动已结束，否则就会导致组件在进行状态迁移时，由于活动的调用改变了组件的状态而产生不可预料的行为结果。一种技术实现是在系统运行中通过装载技术动态插入一个通用的中间调节对象(Mediator)[5]，由其对被替换组件的请求调用情况进行跟踪并逐步建立其静止状态。其建立过程可分为传递、阻塞 2 个阶段。在传递阶段，该中间调节对象追踪所有调用旧组件的线程，并仍将调用传递给旧组件。追踪是对线程进行标识和计数，并存储于一个哈希表中。在阻塞阶段，中间调节对象等待已追踪线程的完成时，暂时阻塞缓存新到来的对旧组件的调用请求，并检查表中的已追踪的调用标识和计数。一旦计数到达零，表明组件中不再有活动调用，组件的静止状态已经建立，就可进行新旧组件间的状态迁移。被替换组件的静态建立既保证了状态迁移的安全性，也为请求缓存和请求的重定向实现奠定了基础。

需要说明的是，组件替换未必都要进行状态迁移。对于一些无状态组件，如 J2EE, Web services 中的无状态实体 Bean，在替换时就不需要进行状态的迁移。

2　技术实现路线

综上所述，关键任务系统中任务热插拔的技术实现路线可概括为如图 2 所示。

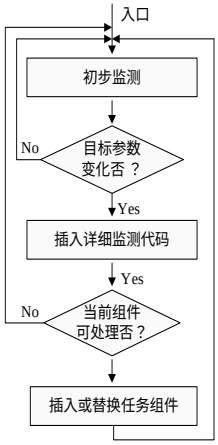

图 2　任务热插拔技术实现流程

首先对系统的运行环境进行监测，即根据关键任务系统的特点确立运行时的监测目标，由基础监测代码对目标进行初步监测。一旦发现监测目标相关参数的变化超过一定的范围，就利用动态装载技术插入更为详细的监测代码来进一步检测产生变化的原因并做出相应的行动决策，如仍提交给当前组件处理，还是插入或替换相应组件并由该组件进行处理。这样，将关键任务系统的整体任务设计为若干相互独立的子任务组件后，通过对系统运行环境的监测及任务组件和代码插入、替换等，既实现了系统的高可用性和对环境的适应性，又避免给系统带来较大的运行负担。

3　初步实验研究

前已述及，任务组件的替换是关键任务系统实现任务热插拔的关键环节。本实验以 J2EE 中 JavaBean API 为实现开发环境，并根据长时间运行的关键任务系统的特点和不同数组排序方法间的性能(如选择排序平均时间复杂度小而堆排序方法稳定性好)差异，以实现堆排序、选择排序的功能组件 Bean1、Bean2 作为任务组件，在程序执行过程中对两者进行动态替换来验证任务热插拔中组件的替换可行性。

在主程序内定义 2 个线程 Thread1, Thread2，它们互斥地随机生成长度不同的数组来模拟系统环境的变化，而由监测代理内的一小段基础监测代码来对系统运行状况进行监测，以数组长度的变化作为监测目标，互斥是在 2 个线程的控制按钮中加进相关的控制信号量来实现的。在系统运行中，当互斥的线程发生切换时，随机生成的数组的长度将发生变化，监测代理内始终运行的基础监测代码将监测到这些变化，并自动装载决策代码来决策确定需替换进来的排序组件，实现对新数组进行排序，从而触发替换过程的发生。程序整体框架如图 3 所示。

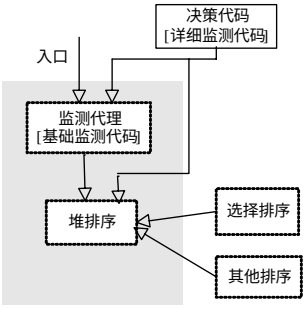

图 3　程序框架

(下转第 37 页)