整合 RTSJ 的动态实时 OSGi 服务部署机制研究

张 奕 蔡皖东
（西北工业大学计算机学院 西安 710072）

摘 要 针对当前 OSGi 规范没有对服务的实时性提出具体标准和详细解决方案的问题，试图将 Java 实时规范（Real-Time Specification for Java，RTSJ）整合到 OSGi 架构中，以提供一种在动态实时嵌入式环境下部署实时组件和服务的解决方案。通过分析 RTSJ 对 OSGi 框架的影响，针对 OSGi 事件机制在 RTSJ 中不能满足实时计算要求的现状，提出了基于 RTSJ 实时线程的事件机制，解决了 OSGi 在 RTSJ 中自适应环境变化所导致的服务之间的实时切换问题，从而确保了基于 OSGi 架构的实时嵌入式系统在动态不确定环境下的实时性。

关键词 自适应，Java 实时规范，OSGi 框架，事件队列，实时线程

中图法分类号 TP311 **文献标识码** A

Research of Real-time OSGi Service Deployment Mechanism Based on RTSJ Event Mechanism

ZHANG Yi CAI Wan-dong
(School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract OSGi specifications provide dynamic services management framework, but there are no real-time criterion and detailed solutions on the specifications. This paper presented a solution of providing a dynamic real-time framework for deploying the real-time components and services by integrating RTSJ into OSGi. This paper first analyzed the impact of RTSJ on the OSGi framework, and then for the reason that the existing OSGi event mechanism in the RTSJ can not satisfy the real-time computing requirements, provided a new event mechanism solution based on the RTSJ real-time threads. It resolves the problem of real-time switching between the services in the RTSJ-based OSGi framework and ensures OSGi-based time predictability of real-time embedded systems in the dynamic environment.

Keywords Adaptive, RTSJ, OSGi framework, Event queue, Real-time thread

1 引言

随着实时嵌入式系统的广泛应用，其运行环境发生了深刻的变革。系统运行时的共享资源、应用负载等随时变化，使得在系统开发时无法对系统运行时的各种性能参数进行精确、定量化的描述。因此，基于静态前提约束的系统资源分配和调度算法，难以对动态、不确定条件下的复杂应用环境进行有效的资源管理。于是设计一种能屏蔽底层运行环境、自适应外界环境变化的实时嵌入式中间件，成为解决上述问题的有效手段之一，同时自适应实时嵌入式中间件对软件体系结构的动态性提出了更高的要求。

面向服务组件的模型将面向服务的计算[1]引入到组件模型中，使得组件模块之间通过服务进行交互。服务提供者和请求者是在运行时动态绑定的，服务提供者可随时注册和注销服务，因此基于面向服务的组件模型的软件具有很好的动态性，为自适应实时嵌入式中间件的开发提供了支持。

OSGi(Open Service Gateway Initiative)[2]定义了一个基于 Java 平台的面向服务组件的编程模型，它可以作为部署和管理组件模块的基础平台。OSGi 具有在运行时动态安装、更新和卸载组件模块的能力，近年在嵌入式系统中得到了广泛的应用。

OSGi 规范虽然提供了动态服务的管理框架，却没有对服务的实时性提出具体的标准和详细的解决方法，使其在实时嵌入式系统中的应用受到很大限制。本文试图将 Java 实时规范（Real-Time Specification for Java，RTSJ）[3]整合到 OSGi 中，以提供一个适合于部署实时组件和服务的动态实时环境。通过 RTSJ 的实时性保证 OSGi 中单个服务的实时性，并通过 OSGi 的动态性来实现自适应环境下的实时服务之间的切换，从而保证实时嵌入式系统中关键任务的实时运行。

2 相关研究

随着实时嵌入式系统的开发复杂度急剧提高，为了降低这类软件的开发成本，面向组件编程技术在实时嵌入式系统中的应用得到了重视和研究，一些基于 RTSJ 的组件编程模型被同时提出。

D. Dvorak 等人[4]提出的用于管理 RTSJ 内存的实时组件模型，解决了 RTSJ 的内存管理，但对实时性的其它方面没有涉及；J. Etienne 等人[5]定义了一种基于实时 Java 的分层组件模型；J. Hu 等人[6]提出了一种基于 RTSJ 的轻量级中间件框架，组件之间通过事件进行交互；A. Plšek 等人[7]提出了

到稿日期：2010-09-22 返修日期：2010-12-04 本文受国家高技术发展计划 863 项目（2009AA01Z424）资助。

张 奕（1977－），女，博士生，讲师，CCF 会员，主要研究方向为安全/任务关键型系统及中间件，E-mail：zywait@nwpu.edu.cn；蔡皖东（1951－），男，博士，教授，主要研究方向为信息安全。

一种基于动态自适应的实时组件模型。

考虑到标准化、动态性、易复用、易扩展和易部署等方面的问题，这些中间件技术直接应用到实时嵌入式系统的效果都不是十分理想。于是尝试将一些已经取得成功的组件编程框架(如 OSGi)应用到实时嵌入式系统中。

W. E. Hong 等人[8]提出了一种基于 RT-linux 和 OSGi 的面向互联网的实时嵌入式框架；N. Gui 等人[9]提出了一种基于 OSGi 框架的实时组件模型。这两个框架将实时任务与非实时任务分离，实时任务由实时操作系统控制，而非实时任务在 OSGi 框架中运行。而本文通过 RTSJ 来实现实时组件，实时任务和非实时任务同在实时 Java 虚拟机中运行，使得资源管理更为方便、高效。

3 OSGi 实时改造可行性分析

3.1 RTSJ 实时线程模型

实时计算的核心内容是可预测性，即保证系统始终在要求的期限内执行。受 Java 语言规范和虚拟机规范中固有的不确定性影响，标准 Java 不能应用于实时系统。RTSJ 作为 Java 在实时性方面的扩展规范，为开发和部署有实时性要求的 Java 应用程序提供了新的线程模型及线程类[3]。

RTSJ 中定义的 Schedulable 接口，提供了对可调度对象的时间特性进行赋值和控制预期行为的方法。RTSJ 同时提供了一个默认的基本调度器 PriorityScheduler，它能按固定优先级，以可抢占方式调度，即最高优先级的调度对象总能抢占处理器而获得优先处理，从而保证调度的完成时限。

RTSJ 定义了 3 类可调度对象：RealtimeThread(RTT)，NoHeapRealtimeThread (NHRT) 和 AsyncEventHander (AEH)，它们之间的类关系如图 1 所示。

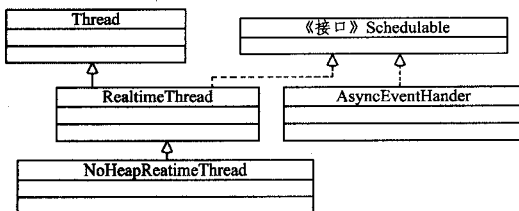

图 1 RTSJ 可调度对象类关系

3.2 RTSJ 对 OSGi 框架的影响

OSGi 的服务由 Java Archive 压缩文件 Bundle 实现。为了使耦合度最小化和可管理，OSGi 提供一种面向服务的架构来实现 Bundle 间的动态交互。通过 OSGi 提供的服务注册、服务发现和服务绑定机制，Bundle 可以方便地对服务之间的依赖关系和状态进行监听、管理。服务的提供者和服务的请求者在运行时动态绑定，在软件保持运行的情况下，提供同一服务接口的不同服务提供者可以进行透明替换，其序列关系如图 2 所示。

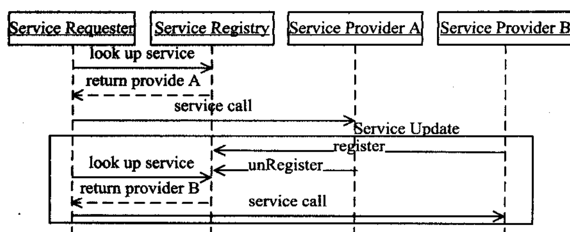

图 2 服务替换序列关系图

为了提供一个适合部署实时组件和服务的动态实时环境，本文将 OSGi 规范提供的动态服务管理框架和 RTSJ 提供的实时性整合为一体，其系统架构如图 3 所示。

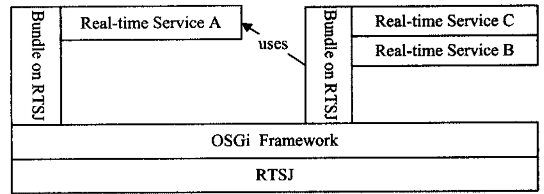

图 3 基于 RTSJ 的 OSGi 架构

RTSJ 的实时 API 实现实时任务并打包成 Bundle，以实时服务的形式发布在 OSGi 框架上。当外部环境发生变化或对系统进行远端控制时，实时服务进行动态切换，以保证实时嵌入式系统关键任务的实时运行。

4 基于 RTSJ 的实时 OSGi 实现

4.1 OSGi 事件机制改造

实现将 RTSJ 整合到 OSGi 中的关键，是如何在 RTSJ 的线程调度模型中支持 OSGi 的事件通知机制。原始 OSGi 通过维护一个事件队列，使得在服务提供者的状态发生变化时，服务使用者能得到相应的事件通知。当服务和 Bundle 的状态变化时，OSGi 框架将触发相关事件(将发生的变化信息包装成事件)添加到事件队列，并由事件派发线程进行派发。事件添加到事件队列后触发过程随即结束，随后的事件处理在事件派发线程上等待执行。

而原始 OSGi 事件队列模型应用于 RTSJ 的缺点是事件派发线程不能在时序上对 java. lang. Thread(JLT)线程提供保证，导致事件队列中事件的派发和处理的时间具有不可预测性，破坏了系统的实时性。本文通过以下 3 个方面的改进来保证系统的实时性：

(1)提供一种可行的事件机制：利用具有精确调度语义、优先级更高、且不被垃圾收集器所抢断的 NHRT 线程替换 JLT 线程来实现事件派发线程，从而保证事件派发的实时性。本文采用服务器算法中的 PS 算法(Polling Server)[10]来处理异步事件，即通过一个具有最高优先级的轮询任务，以固定的周期来处理异步事件。如果当前系统中没有异步事件，则将轮询任务挂起直到下一周期，已分配给异步事件的处理时间将被放弃。异步事件的响应时间的最大值可定义为：

$$R_i \leqslant (\overline{T}_s - \overline{C}_s) + \left\lceil \frac{C_i + B_i}{\overline{C}_s(i)} \right\rceil \overline{T}_s \qquad (1)$$

式中，R_i 为异步事件的响应时间，$\overline{T}_s$ 为轮询任务的发布周期，$\overline{C}_s$ 为轮询任务所分配的 CPU 时间，C_i 为处理异步事件的时间，B_i 为线程阻塞的时间。

(2)解决基于 JLT 的事件触发线程和基于 NHRT 的事件派发线程对事件队列进行同步访问的问题：原始 OSGi 框架基于监视器的方法可能会导致优先级反转。RTSJ 提供了优先级置顶算法[11]和优先级继承算法[12]来避免优先级反转。但 RTSJ 规定 NHRT 线程需要比垃圾回收线程具有更高的执行机率，从而导致 NHRT 线程不能与 JLT 线程同步访问同一个对象。为了解决这一问题，通过 RTSJ 提供的 Wait-Free 队列[13](如图 4 所示)来实现事件队列，解决了由 NHRT 线程、垃圾回收线程和优先级反转所产生的问题。

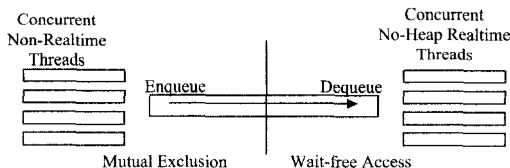

图 4 Wait-Free 队列

(3)改进 OSGi 中事件的处理模式：在 OSGi 现有实现中，事件处理线程和派发线程是同一线程。事件从事件队列中取出之后，立即在当前线程中处理，处理完毕才取下一事件继续循环处理，这种模式的缺点是派发线程长时间被事件处理逻辑所占据。为了提高派发线程的性能，需对原始的单线程事件模型进行改造，即仅让 NoHeapRealtimeThread 线程负责从事件队列中获取事件，而事件的处理则交由 RTSJ 的异步事件处理器 AEH 执行。由于 AEH 是由 RTSJ 调度器调度的对象，因此保证了事件处理的实时性。

4.2 OSGi 实时线程模型

基于 RTSJ 的实时事件模型可分解成事件队列和服务器线程两个不同的部分，本文采用 UPPAAL 工具进行时间自动机建模：

(1)事件队列(Event Queue,EQ) 亦即事件处理请求队列。EQ 中声明了一个 Wait-Free 事件队列，并提供了处理队列中事件的方法：getEventHandler()、putEventHandler()和 getFront()。EQ 的状态更新关系如图 5 所示。

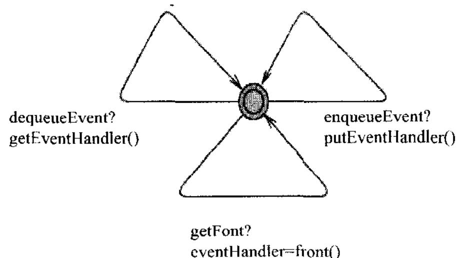

图 5 事件队列模型

(2)服务器线程(Server Thread,ST) 即事件队列的派发线程。ST 具有空闲、就绪、执行和阻塞 4 种状态，ST 的自动机状态更新关系如图 6 所示。

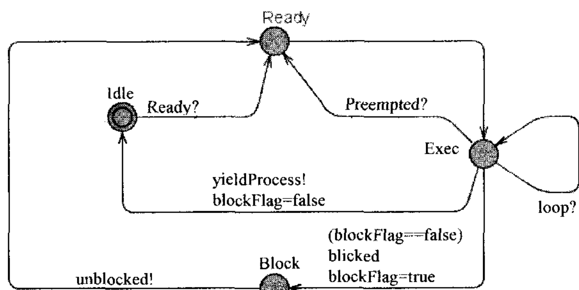

图 6 服务器线程模型

4.3 OSGi 实时线程实现

OSGi 作为一个开放规范，存在多种实现。Felix[14] 实现了 OSGi R4 规范，具有代码规模小且性能稳定的特点，适合在嵌入式设备上运行。本文基于 Felix 实现 OSGi 的实时事件机制，利用 RTSJ 的 NHRT 线程替换原始 JLT 线程，并用 Wait-Free 队列替换原始 java. util. ArrayList 对列。当服务器线程从事件队列获得事件后，事件的处理交由 RTSJ 的 AEH 来执行，实时 OSGi 的线程改造算法如图 7 和图 8 所示。

```
static WaitFreeReadQueue queue= new WaitFreeReadQueue(QUEUE_SIZE,0);
private void fireEventAsynchronously(…){
    …
    //将事件请求写入 Wait-Free 对列
    ImmortalMemory. instance(). executeInArea(new Runnable()
    {public void run(){queue. write(request);}});
}
```

图 7 基于 WaitFreeReadQueue 的事件添加

```
static WaitFreeReadQueue queue = new WaitFreeReadQueue
(QUEUE_SIZE,0);
static immortal= ImmortalMemory. instance();
public void run(){
  while(true) {
      while(! queue. isEmpty){ //从 Wait-Free 对列中取出请求
        req=(Request)queue. read();
        if(req! = null) { //将请求转化为 AEH
          handler=(AsyncEventHandler)immortal. newInstance
          (eventClass);
          event = (AsyncEvent) immortal. newInstance (AsyncEvent. class);
          event. addHandler(handler);
          event. bindTo(bindName);
          event. fire();
        }
        RealTimeThread. waitForNextPeriod();//等待下一次发布
      }
   }
}
```

图 8 NoHeapRealtimeThread 事件派发处理

5 实验

为了验证上述方法的有效性，将改进后的 OSGi 框架和原始 OSGi 框架在不同的垃圾收集频率下对 OSGi 事件的反应时间抖动进行对比。

实验采用 Pentium(R)1. 5GHz 处理器，512MB 内存。底层操作系统为 SUSE Linux Enterprise Real-Time 10，RTSJ 平台为 Sun Java Real-Time System2. 2，所有 OSGi 代码均在 Felix2. 0. 4 下实现。

实验针对 10 个不同的垃圾收集频率，对 OSGi 事件的响应时间各收集 12 组数据，并对 12 组数据分别求标准差和最大离均差以比较它们与平均值的偏离程度。为了更清晰地显示实验结果，将标准差和最大离均差与平均值相除得出倍率关系并转化为百分率，如图 9 和图 10 所示。其中横坐标表示垃圾收集频率，纵坐标具体定义为：

$$y=\frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})^2}}{\bar{x}}*100\% \tag{2}$$

式中，$\bar{x}=\frac{1}{N}\sum_{i=1}^{N}x_i$。

$$y=\frac{\max(|x_i-\bar{x}|)}{\bar{x}}*100\% \tag{3}$$

(下转第 192 页)

[8] Saitta L, Zucker J-D. A Model of Abstraction in Visual Perception[J]. Applied Artificial Intelligence, 2001, 15(8): 761-776

[9] 孙善武，王楠，欧阳丹彤. 广义 KRA 抽象模型[J]. 吉林大学学报：理学版，2009，47，(3)：537-542

[10] Wang Nan, OuYang Dan-tong, Sun Shan-wu, et al. Formalizing the Modeling Process of Physical Systems in MBD[C]// The 2009 International Conference on Web Information Systems and Mining (WISM'09) and the 2009 International Conference on Artificial Intelligence and Computational Intelligence (AICI'09). 2009: 685-695

[11] 王楠，欧阳丹彤，孙善武. 基于本体的分层抽象模型[J]. 计算机科学(已录用)

[12] Wang Nan, OuYang Dan-tong, Sun Shan-wu. Formalizing Ontology-based Hierarchical Modeling Process of Physical World[C]// The 2010 International Conference on Web Information Systems and Mining (WISM'10) and the 2010 International Conference on Artificial Intelligence and Computational Intelligence (AICI'10). 2010: 18-24

[13] Sun Shan-wu, Wang Nan. Formalizing the Multiple Abstraction Process Within the G-KRA Model Framework[C]// 2010 International Conference on Intelligent Computing and Integrated Systems (ICISS2010). 2010: 281-284

(上接第 149 页)

式中，$\bar{x}=\frac{1}{N}\sum_{i=1}^{N}x_i$。

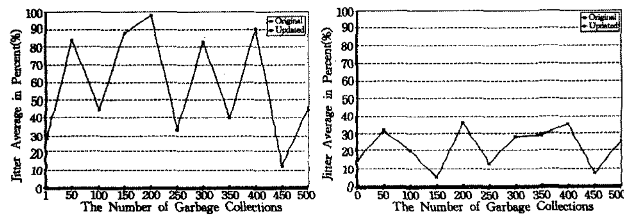

图 9 响应时间的的标准差比较　图 10 响应时间的最大离均差比较

由实验结果可以看出，原始的 OSGi 事件响应时间具有较大的不确定性，修改后的 OSGi 框架在 120 个实验数据中，标准差与平均值的比率最大值为 0.03%，考虑到线程切换的时间开销，我们认为修改后的 OSGi 框架的响应时间基本无抖动，具有确定性。由此得：基于实时事件机制的 OSGi 框架可以满足嵌入式实时系统的实时性要求。

结束语 本文通过分析 RTSJ 对 OSGi 框架的影响，并针对 OSGi 事件派发线程的实时性无法保证的问题，提出了基于实时线程的事件机制，解决了 OSGi 为了自适应环境变化所导致的实时服务之间无法切换的问题，从而保证了整个实时嵌入式系统的实时性。

我们的研究目标是提出一个基于 OSGi 的自适应实时中间件，而将 RTSJ 整合到 OSGi 中的工作目前只完成了第一步。下一步还将对以下两个方面进行深入的研究：(1) 基于 RTSJ 的 OSGi 各实时组件间的运行期隔离；(2) 基于实时 OSGi 的端到端的自适应 QoS 协商机制。

参考文献

[1] Erl T. Service-Oriented Architecture: Concepts, Technology, and Design [M]. Prentice Hall, 2005

[2] OSGi Service Platform Core Specification v4.0 [EB/OL]. http://www.osgi.org, 2010-4-2

[3] Belliardi R, Brosgol B, Dibble P, et al. The real-time specification for Java-version 1.0.2 [M]. USA: Addison-Wesley, 2004

[4] Dvorak D, Bollella G, Canham T, et al. Project Golden Gate: Towards Real-Time Java in Space Missions [C]// Proceeding of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. Vienna: IEEE Press, 2004: 15-22

[5] Etienne J, Cordry J, Bouzefrane S. Applying the CBSE Paradigm in the Real-time Speci_cation for Java [C]// Proceedings of the 4th interna-tional workshop on Java technologies for real-time and embedded systems. USA: ACM Press, 2006: 218-226

[6] Hu J, Gorappa S, Colmenares J A, et al. Compadres: A Lightweight Component Middleware Framework for Composing Distributed, Real-Time, Embedded Systems with Real-Time Java [C]// Proceeding of ACM/IFIP/USENIX 8th Int'l Middleware Conference (Middleware 2007). vol. 4834. Newport Beach: Springer, 2007: 41-59

[7] Plˇsek A, Loiret F, Merle P, et al. A Component Framework for Java-based Real-Time Embedded Systems [C]// Proceeding of ACM/IFIP/USENIX 9th International Middleware Conference. Leuven: Springer, 2008: 124-143

[8] Hong W E, Ku B J, Lee M J, et al. Combined Approach of OSGi and RTLinux Framework for Supporting Software Architecture of Internet Embedded Real-Time System [C]// Proceeding of IEEE Real-Time and Embedded Technology and Applications Symposium. Toronto: IEEE Press, 2004: 296-305

[9] Gui N, Florio V D, Sun H, et al. A Hybrid real-time component model for reconfigurable embedded systems [C]// Proceedings of the 2008 ACM symposium on Applied computing. Fortaleza: ACM Press, 2008: 1590-1596

[10] Tia T, Liu J W, Shankar M. Aperiodic request scheduling in fixed-priority preemptive systems [R]. UIUCDCS-R-94-1859. Dept. Computer Science, University of Illinois at Urbana-Champaign, 1994

[11] Goodenough J B, Sha L. The priority ceiling protocol: A method for minimizing the blocking of high priority Ada tasks [C]// Proceedings of the second international workshop on Real-time Ada. vol. 7. Devon: ACM Press, 1988: 20-31

[12] Sha L, Rajkumar R, Lehoczky J P. Priority Inheritance Protocols: An Approach to Real-Time Synchronization [J]. IEEE Transactions on Computers, 1990, 39(9): 1175-1185

[13] Tsigas P, Zhang Y, Cederman D, et al. Wait-Free Queue Algorithms for the Real-time Java Specification [C]// Proceeding of the12th IEEE Real-Time and Embedded Technology and Applications Symposium. San Jose: IEEE Press, 2006: 373-383

[14] Apache Felix [EB/OL]. http://felix.apache.org/site/index.html, 2010-3-2