<<Java   >>

<<Java   >>

13 ISBN   9787564119300

10 ISBN   7564119306

            2010-1

        305

                    PDF

            http://www.tushu007.com

<< Java   >>

When I was presented with the opportunity to revise Java Message Service, I jumped at the chance. The first edition, published by O'Reilly in 2000, was a bestseller and without a doubt the definitive source for JMS and messaging in general at that time. Writing the second edition was an exciting chance to breath new life into an already great book and add new content that was relevant to how we use messaging today. What I failed to fully realize when I took on the project was just how much messaging(or, more precisely, how we use messaging) has changed in the past 10 years. New messaging techniques and technologies have been developed, including messagedriven beans (as part of the EJB specification), the Spring messaging framework, Event Driven Architecture, Service-Oriented Architecture, RESTful JMS interfaces, and the Enterprise Service Bus (ESB), to name a few. The somewhat minor book project that I originally planned quickly turned into a major book project. My original intent was to preserve as much of the original content as possible in this new edition. However, based on changes to the JMS specification since the first edition was written, as well as the development of new messaging techniques and technologies, the original content quickly shrank. As a result, you will find that roughly 75% of this second edition is new or revised content. The JMS specification was updated to version 1.1 a couple of years after the printing of the first edition of this book. While not a major change to the JMS specification, the JMS 1.1 specification was nevertheless a significant step toward fixing some of the deficiencies with the original JMS specification. One of the biggest changes in the specification was the joining of the queue and topic APl under a unified general APl, allowing queues and topics to share the same transactional unit of work. However, the specification change alone was not the only factor that warranted a second edition of the book. As the Java platform has matured, so has the way we think about messaging. From new messaging technologies and frameworks to complex integration and throughput requirements, messaging has changed the way we think about and design systems, particularly over the past 10 years. These factors, combined with the specification changes, are the reasons for the second edition.

<<Java   >>

Java   API   ——

        JMS

        JMS 1.1   messaging
  IBM   MQ   Progress   SonicMQ   ActiveMQ
      Java
                                                              beans   Enterprise JavaBeans(EJB)
        JMS   RESTful   Spring

Java

　　　　(Mark Richards)

Richard Monson-Haefel　O'Reilly　　　　　Enterprise JavaBeans　　　Java

David A. Chappell　Oracle　　　　　　SOA　　　　　　　　　　Java Web Services　　　Java
　　　　　　O'Reilly

<< Java   >>

<<Java   >>

Messaging Design Considerations Internal Versus External Destination   Internal Destination Topology   External Destination Topology Request/Reply Messaging Design Messaging Design Anti-Patterns   Single-Purpose Queue   Message Priority Overuse   Message Header MisuseA. The Java Message Service APIB. Message HeadersC. Message PropertiesD. Installing and Configuring ActiveMQIndex

Event-Driven Architecture   EDA   is an architecture style that is built on the premisethat the orchestration of processes and events is dynamic and very complex, and there-fore not feasible to control or implement through a central orchestration component.When an action takes place in a system, that process sends an event to the entire systemstating that an action took place   an event   . That event may then kick off other pro-cesses, which in turn may kick off additional processes, all decoupled from each other.Some good examples of EDA include the insurance domain and the defined benefitsdomain. Both of these industry domains are driven by events that happen in the system.For example, something as simple as changing your address can affect many aspects ofthe insurance domain, including policies, quotes, and customer records. In this case,the driving event in the insurance application is an address change. However, it is notthe responsibility of the address change module to know everything that needs to happen as a result of that event. Therefore, the address change module sends an eventmessage letting the system know that an address has changed. The quoting system willpick up that event and adjust any outstanding quotes that may be present for thatcustomer. Simultaneously, the policy system will pick up the change address event andadjust the rates and policies for that customer.Another example of EDA is within the defined benefits domain. Getting married orchanging jobs triggers events in the system that qualify you for certain changes to yourhealth and retirement benefits. Many of these systems use EDA to avoid using a large,complex, and unmaintainable central processing engine to control all of the actionsassociated with a particular "qualifying event."Messaging is the foundation for systems based on an Event-Driven Architecture. Eventsare typically implemented as empty payload messages containing some informationabout the event in the header of the message, although some pass the application dataas part of the event. Not surprisingly, most architectures based on EDA leverage thepub/sub model as a means of broadcasting the events within a system.

<<Java   >>

"   Java   JMS API
                                    Enterprise Java
"   ——Tim Berglund   August

<<Java   >>

Java   （ 2 ·   ）

<<Java   >>

PDF

:http://www.tushu007.com