事务：在控制之中吗？--附一些高深内容 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/461/2021_2022__E4_BA_8B_E5_8A_A1_EF_BC_9A_E5_c104_461594.htm BEA的WebLogic Platform 8.1的发布引起了行业分析家和IT从业者的极大热情，他们认识到它的潜力使J2EE平台的强大功能为更多的开发人员所使用。 这使得J2EE架构师可以做它最擅长的工作--体系设计和技术问题的解决，同时允许"普通"开发人员使用由专家设计的体系结构，这些开发人员迄今为止还被限制于构建部门级的应用程序，因为他们缺少体系结构，从而缺乏可伸缩性来支持过去常使用的易用工具来生成的应用程序。 WebLogic Platform的两个关键特征可以把高级业务开发人员和高级J2EE架构师的职责分开--BEA WebLogic Workshop的运行时（runtime），它利用高级体系结构命令对资深业务人员施加影响；Workshop开发环境，它允许开发者通过一致的图形抽象与环境相互作用，只有在需要表达业务规则而不是组装应用程序时才转变为代码行。IDE同样有帮助架构师的特性--它们可以产生骨架式的应用程序环境，植入标准组件等，并可以把这一环境以模板的形式向其他开发者发布，这使得业务程序员可以通过预先打包的方式和自动的方式快速启动工程（重要的是，这是一个与开发标准及其他开发工程一致的快速启动）。 运行时框架的关键部分是控件体系结构--控件提供了封装业务逻辑的方法。业务开发者可以编写业务规则，使用控件以组件的形式提供业务规则，或者封装某些资源所需的复杂逻辑和基础结构。J2EE架构师可以实现这一"硬核心管道化（plumbing）"并把它包装在控件中，这样应用程序

开发者就可以很容易地像使用他们自己的逻辑那样地使用控件。这也解决了大型开发部门经常遇到的另一问题--J2EE编码者编写了许多非常优秀的基础结构，但因为业务开发者不知道如何去使用它，所以它们只能被其他J2EE专家所使用。从总体上来说，以J2EE开发者的角度来看，他们开始尽量避免做沉闷的、易错的剪切和粘贴工作--让我们做这些工作吧，它只不过是形成前端的另一个struts，或者只不过是另一个代码摘录来在JNDI查找JMS队列并通过它发送消息，没有人愿意干这种事。从非J2EE开发者的角度来说，用之前存在的组件组装有用的应用程序（比如以一种新的方法组合现有子系统的另一套Web页面）就变为可能，而这并不需要通过挑灯夜战来学习J2EE知识。 事务：您失去了计划吗？ 对于事务我们究竟要做什么？什么都不需要，它只是背景。如果不讨论它，我又怎么写文章？它是什么？哦…… 事务、Workshop和控件 控件只是简单的有注释的Java对象--这些注释提供了允许控件用户在高于通常的J2EE接口的抽象级工作的能力。当部署控件（实际上可能是多个控件的组合）时，注释对生成运行时联结（它是真正被部署的部分）起推动作用。与其他任何Java对象一样，控件从调用者中继承了事务上下文。因为控件没有远程接口的概念（至少当前的版本没有），所以调用者通常是由包含在EJB中的WebLogic Workshop生成的轻重量级控件容器。如果您查看与这个（WebLogic Workshop管理的）EJB关联的部署描述符，就会看到它有"容器"的事务策略--这样控件的事务上下文可以由EJB容器使用JTA提供，就像该控件是您编写的并从EJB代码调用的简单陈旧的Java对象。 Workshop中最接近远程接口的事务是Web服务--它很容易

得到控件并把控件表现为Web服务（潜在的和对话Web服务），而不仅仅是一些鼠标的单击，这样剩下的问题就是……什么是Workshop Web Service的默认事务行为？ 当对Web服务方法进行调用时，从消息到达调用的联接已经通过大量J2EE（这完全取决于您在注释里的声明）机制完成，最后到达具有容器管理事务的EJB中。在执行方法期间， JTA事务将会运行。 如果方法成功，事务会进行提交。如果方法失败（抛出Exception），事务将会回滚。就是这么简单。 回忆这一点，即使WebLogic Workshop Web服务是可会话的（如果注释是这么说）。会话状态保存在数据库的表中，这一持久状态是如何与任何应用程序管理的持久状态关联的呢？它们包含在同一事务的上下文中。这样如果您的方法失败了，就好像这段会话从来没有发生过。精细自动的行为事务是受欢迎的。这对部署意味着，在默认情况下，对话状态通过cgDataSource数据源保存。如果您的应用程序状态保存在别的地方，您可能会得到错误提示：不能通过事务影响数据源，因为它已经影响了作为cgDataSource底层的cgPool。您可以通过两种方法来修正该错误：要么更改cgPool来使用xa的数据库访问并得到两阶段的提交，要么让会话状态和应用程序状态通过相同的连接池保留在同一数据库实例中（Workshop的jws-config.properties文件从WorkShop角度进行控制）来避免两阶段的提交需要。 如果一个Workshop Web服务调用另一个Web服务，在事务的上下文不会被传播，这样被调用的服务将根据我刚才概述的规则来运行它自己新的事务。如果您希望失败的服务调用回滚到调用者的事务，请重新向该架构抛出异常。如果想不管Web服务调用的失败来保存调用者（

包括应用程序和会话)的状态，则可以捕捉到异常但不重新抛出。 当然，现在应用的是事务的通常规则。如果您想保存一些数据（比如审计记录）而不考虑事务最终的结果，则您需要用到TransactionManager对象，在调用之前挂起事务，并在稍后进行恢复。对非J2EE的高手来说这是很常见的，因为这会使该对象成为J2EE架构设计师应该实现并提供给应用程序开发人员作为预先构建的控件。 关键是，它没关系！ 最后的几百字对于许多应用程序开发者来说有些高深（至少是接近于高深）。而这就是关键。如果J2EE架构设计师理解了这些内容，并且当他们在做应用程序开发时把模板和开发方针一起使用时想到了这些内容，那么应用程序的开发人员就不需要担心--框架代表他们做正确的事情，它们也会很快地开发出更多应用程序，与没有指引正确方向的框架和模板相比，这些应用程序的行为更具有一致性。这意味着IT部门在下一次业务需求再次调整的紧急关头可以面带微笑并说没问题，而不是嘀咕着更新自己的简历。 