# 中间件技术
# Middleware Technology

## 第六章　Web组件和容器

赖永炫　博士/教授
厦门大学　软件工程系

# 大纲

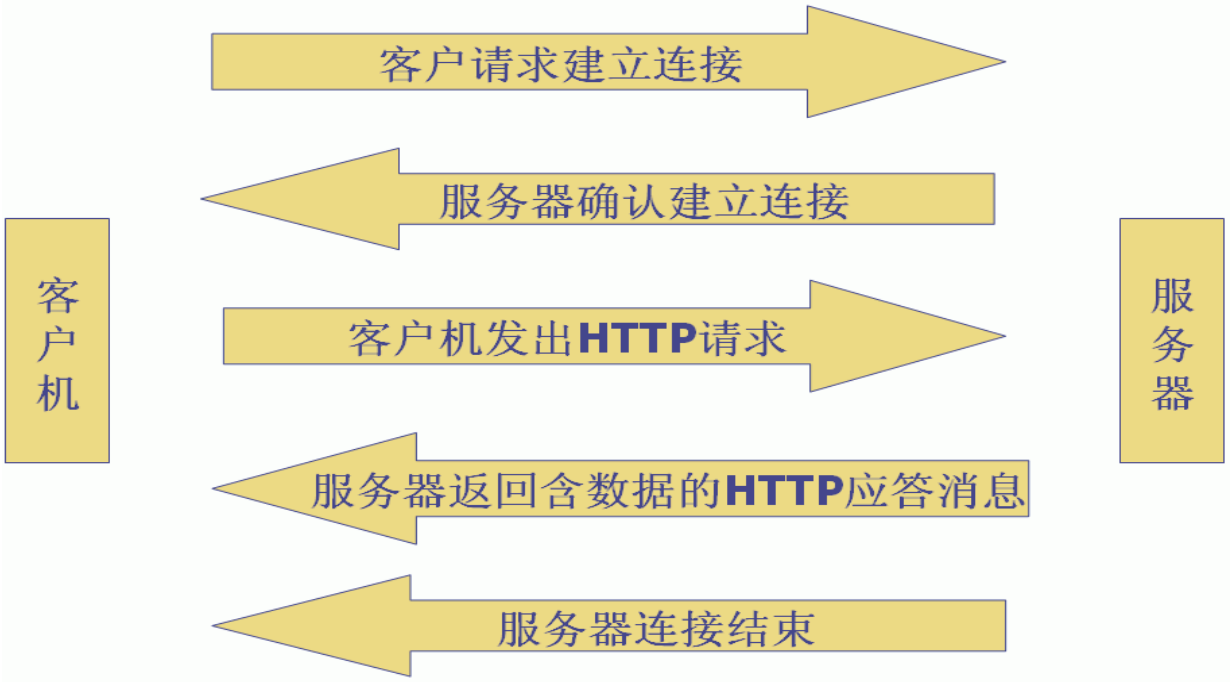
- 概念简介
- Java EE和Spring框架介绍
- 相关技术介绍
  - ✓ 反射
  - ✓ 注解
  - ✓ 依赖注入
  - ✓ AOP编程
- EJB编程

# Web服务器的概念

- Web服务器即网站服务器，是指驻留于因特网上某种类型计算机的程序。
- 可向浏览器等Web客户端提供文档，显示界面。
- Web服务器专门处理HTTP请求，并为应用程序提供商业逻辑。

# 工作过程

- 服务器的工作过程一般可分成如下4个步骤：连接过程、请求过程、应答过程以及关闭连接



# 网页访问提供中间件

- 连接过程、请求过程、和关闭连接都是较为标准的例行程序，关键点在于应答过程中，即如何根据请求返回各种各样的结果网页。
- Web服务器：分离抽象出服务器的基本执行流程，例行的标准化的流程由服务器来处理，而用户更多的负责定义个性化的、非标准化的流程。
  - ✓SSH架构（Spring、Struts、Hibernate），Ruby on Rails架构……
  - ✓Tomcat/Weblogic / WildFly/ IIS

# 网页的分类

- 网页可分为静态网页和动态网页两种:
  - ✓静态网页是预先存在服务器上的固定文件;
  - ✓动态网页则是服务器根据用户的请求动态组装而成。不同的请求，动态网页返回的结果一般不同。

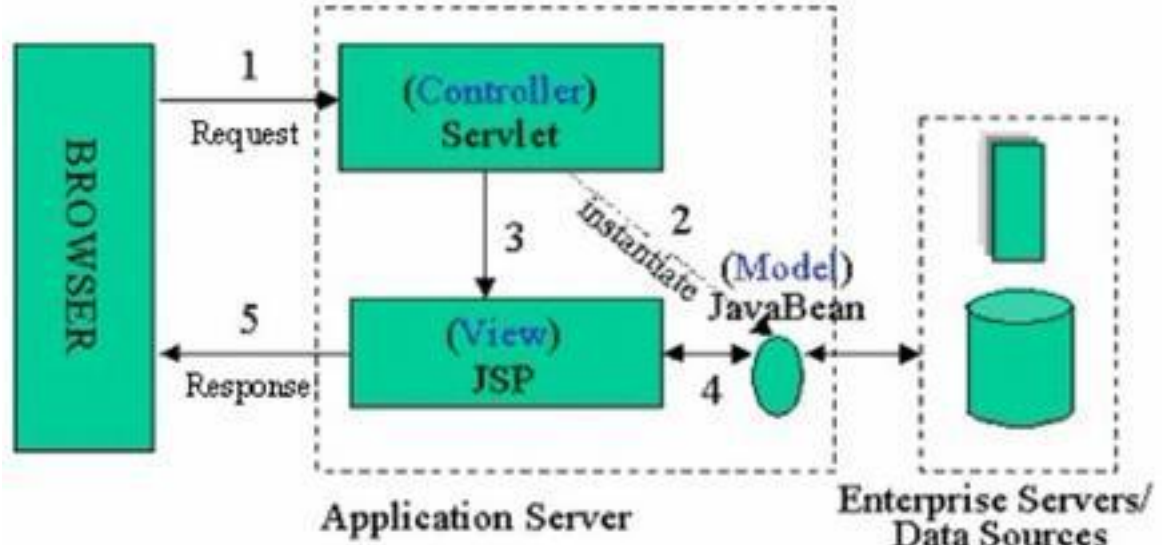
- 对于动态网页而言，用户访问web服务器时，服务器会在内存中生成各种对象来处理请求和处理业务逻辑。

# MVC框架

- MVC被大多数网页服务器采用
- Model View Controller即很好的抽象了客户端和服务器的访问流程图。
  - ✓ Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。通常模型对象负责在数据库中存取数据。
  - ✓ View（视图）是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。
  - ✓ Controller（控制器）是应用程序中处理用户交互的部分。

# MVC与Web服务器

- 通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。
- 作为MVC的一个具体实现，JavaEE的组件则大体也可以按照模型-视图-控制器的方式来进行划分。
- Web服务器也提供了很多处理请求、分配任务、封装显示界面的类。用户继承这些类，覆写相应的方法，即可加入个性化处理，同时符合基本的网页连接和处理流程。



# Web容器的概念

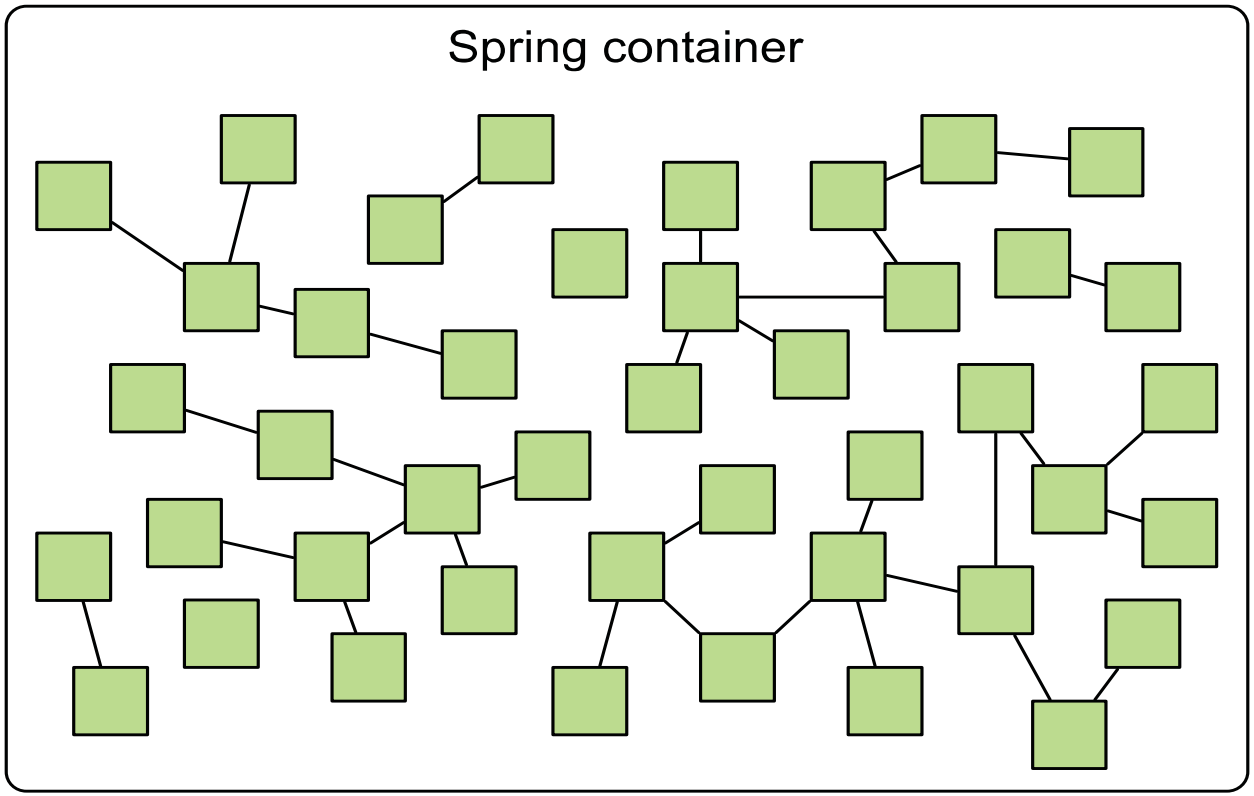



**Figure 1.4** **In a Spring application, objects are created, are wired together, and live in the Spring container.**

# Web容器的概念

- 容器可以管理对象的生命周期、对象与对象之间的依赖关系。
- 通过容器的配置（如XML配置文件），可以定义对象的名称、产生方式、对象间的关联关系等。
- 在启动容器之后，容器自动的生成这些对象，而无需用户编码来产生对象，或建立对象与对象之间的依赖关系。Web容器是自动化例行程序，减少用户工作量的一个有效方法。

# JAVA EE作为Web服务器

- Java EE（Java Platform，Enterprise Edition）是sun公司推出的企业级应用程序版本。
  - ✓这个版本以前称为 J2EE（1.2～1.4 版本），从 1.5 开始正式使用 Java EE 名字。
- 能够帮助我们开发和部署可移植、健壮、可伸缩且安全的Web应用和企业应用。

# JAVA EE

- Java EE（Java Platform，Enterprise Edition）是sun公司推出的企业级应用程序版本。这个版本以前称为 J2EE（1.2～1.4 版本），从 1.5 开始正式使用 Java EE 名字。
- 能够开发和部署可移植、健壮、可伸缩且安全的Web应用和企业应用，具备快速响应性和适应用户需求增长的伸缩性。

通常设计为多层的分层应用，包括：Web框架的前端层，提供安全和事物的中间层，提供持久性服务的后端层。

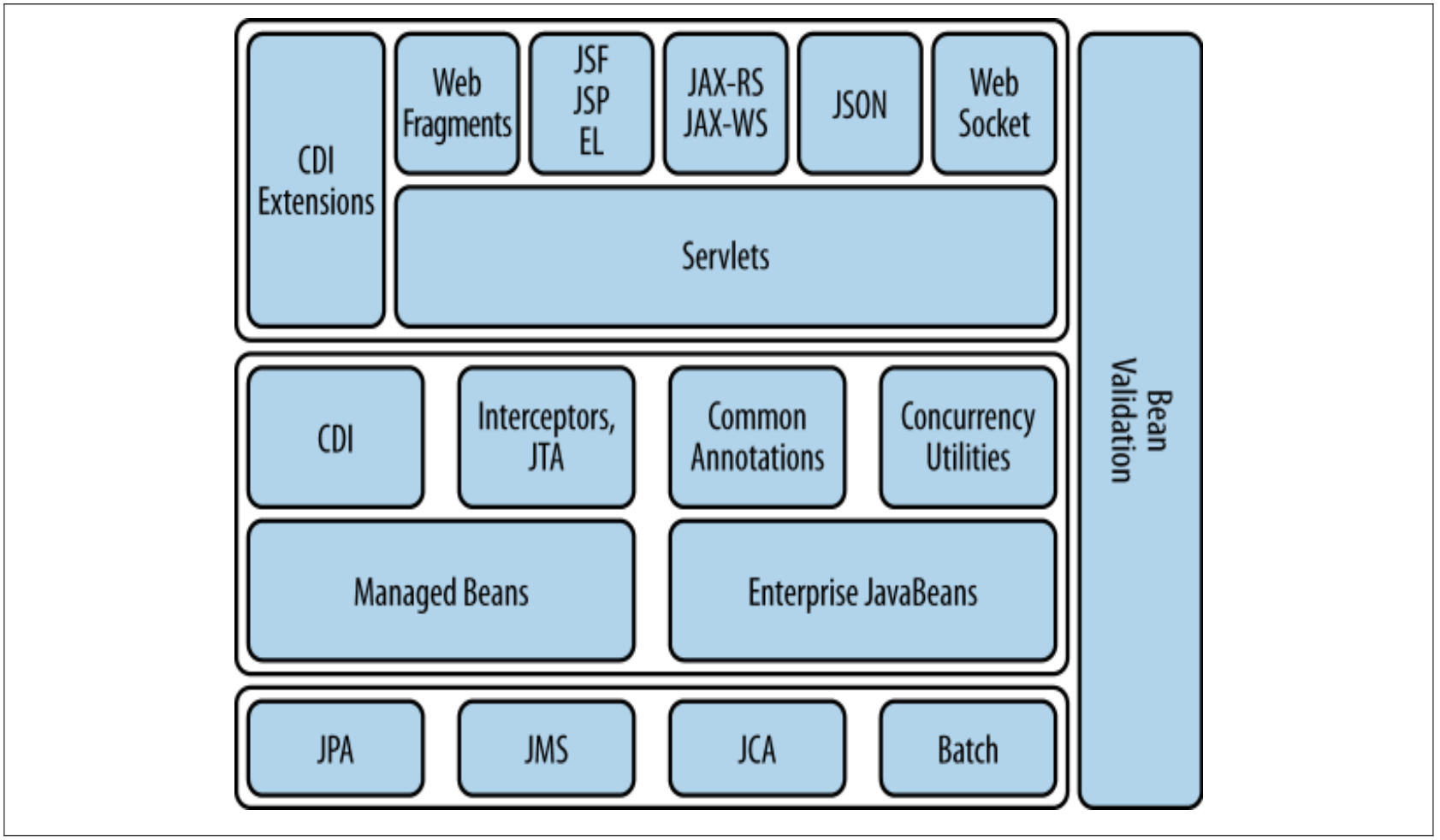



*Figure 1-1. Java EE 7 architecture*

# Java EE 应用

- 典型的Java EE应用，至少应该包括三部分：表现层，业务逻辑层和数据持久层。
  - ✓ 为了更加容易地创建企业应用程序，许多的框架涌现出来。
  - ✓ 表现层：Struts, JSF, Tapestry, WebWork, Velocity等
  - ✓ 业务逻辑层：普通的JAVA Beans，也可以用EJB(Session Bean);
  - ✓ 数据持久层：JDBC, ORMapping tools(Hibernate,toplink等),SQLMapper tools(Ibatis),JDO, EJB(Entity Bean)等。

# Java EE平台

- Java EE平台由一整套服务（Services）、应用程序接口（APIs）和协议构成，它对开发基于Web的多层应用提供了功能支持。

- JDBC（Java Database Connectivity）提供连接各种关系数据库的统一接口，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC为工具/数据库开发人员提供了一个标准的API，据此可以构建更高级的工具和接口，使数据库开发人员能够用纯 Java API 编写数据库应用程序，同时，JDBC也是个商标名。

- EJB(Enterprise JavaBeans)使得开发者方便地创建、部署和管理跨平台的基于组件的企业应用。
- Java RMI(Java Remote Method Invocation)用来开发分布式Java应用程序。一个Java对象的方法能被远程Java虚拟机调用。这样，远程方法调用可以发生在对等的两端，也可以发生在客户端和服务器之间，只要双方的应用程序都是用Java写的。

- EJB(Enterprise JavaBean)提供了一个框架来开发和实施分布式商务逻辑，由此很显著的简化了具有可伸缩性和高度复杂的企业级应用程序的开发。
  - ✓EJB规范定义了EJB组件在何时如何与它们的容器进行交互作用。容器负责提供公用的服务，例如目录服务，事务管理，安全性，资源缓冲池以及容错性。
  - ✓EJB并不是实现Java EE的唯一路径。正是由于Java EE的开放性，使得所有的厂商能够以一种和EJB平行的方式来达到同样的目地。

- JMS(Java Message Service)提供企业消息服务，如可靠的消息队列、发布和订阅通信、以及有关推拉（Push/Pull）技术的各个方面。

- JTS(Java transaction Service)提供存取事务处理资源的开放标准，这些事务处理资源包括事务处理应用程序、事务处理管理及监控。

- Annotation(Java Annotation) 提供一种机制，将程序的元素如:类，方法，属性，参数，本地变量，包和元数据联系起来。这样编译器可以将元数据存储在Class文件中。这样虚拟机和其它对象可以根据这些元数据来决定如何使用这些程序元素或改变它们的行为。

- JMF(Java Media Framework API)可帮助开发者把音频、视频和其他一些基于时间的媒体放到Java应用程序或applet小程序中去，为多媒体开发者提供了捕捉、回放、编解码等工具，是一个弹性的、跨平台的多媒体解决方案。
- javaFX利用 JavaFX 编程语言开发富互联网应用程序(RIA)。JavaFX Script编程语言一种声明性的、静态类型脚本语言。

- JMX（Java Management Extensions）即Java管理扩展，是一个为应用程序、设备、系统等植入管理功能的框架。
  - ✓ JMX可以跨越一系列异构操作系统平台、系统体系结构和网络传输协议，灵活的开发无缝集成的系统、网络和服务管理应用。
- JPA (Java Persistence API)JPA通过JDK 5.0注解或XML描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中。

# Spring开源框架

- Spring是一个开源框架，是为了解决企业应用开发的复杂性而创建的。
- Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。可以独立或在现有的应用服务器上运行，用于服务器端的开发。
- 从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。

# Spring起源

- EJB是sun公司的JavaEE服务器端组件模型，设计目标与核心应用是部署分布式应用程序。简单来说就是把已经编写好的程序（即"类"）打包放在服务器上执行，用C/S 形式的软件客户端对服务器上的"类"进行调用。
- J2EE应用程序的广泛实现是在2000年左右开始的，它的出现带来了诸如事务管理之类的核心中间层概念的标准化，但是在实践中并没有获得绝对的成功，因为开发效率，开发难度和实际的性能都令人失望。

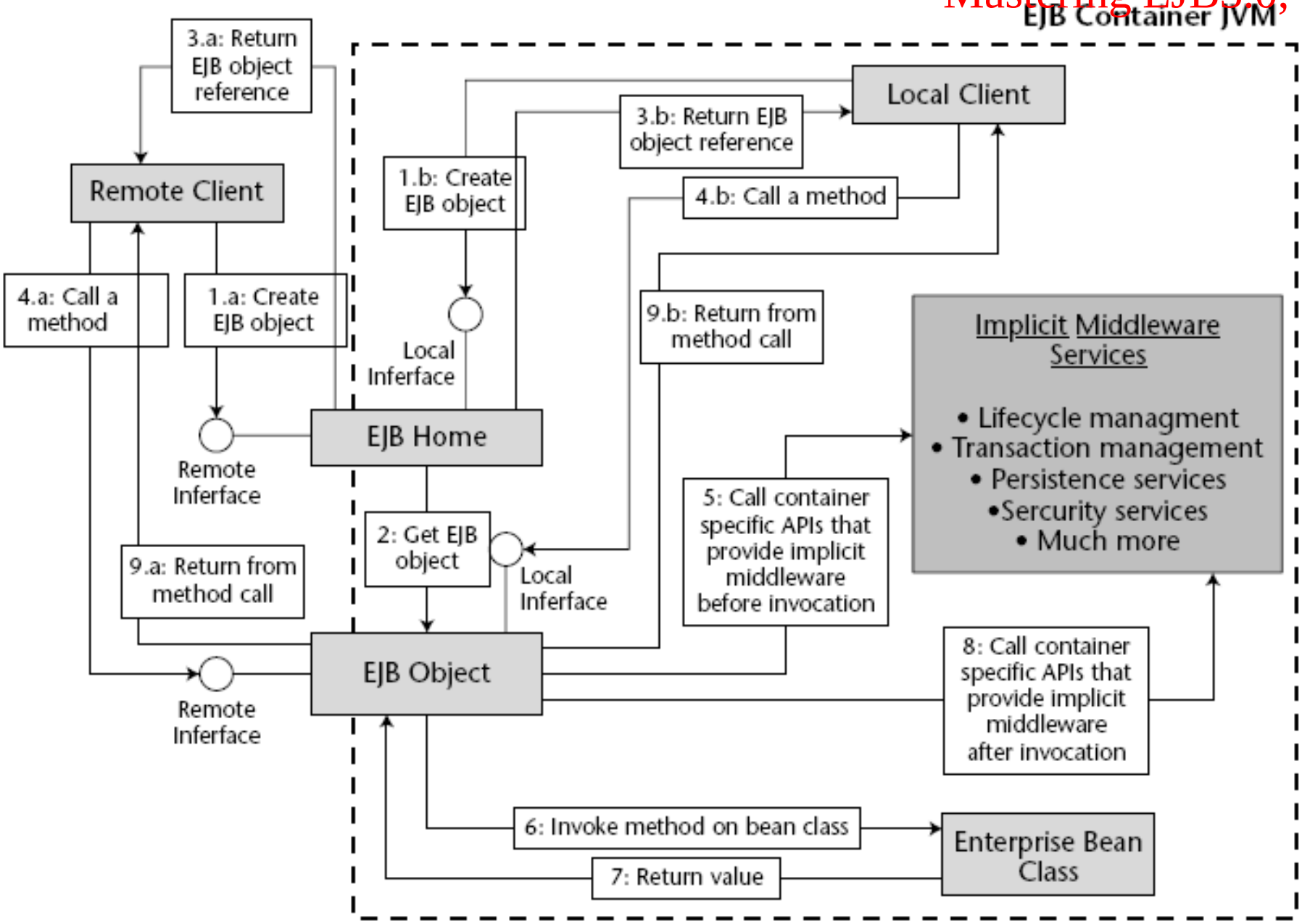



**Figure 3.3** Pre–EJB 3.0 programming model.

# Pre-EJB 3.0: The World That Was

- Dissect EJB 2.x
  - ✓ Development Complexities
  - ✓ Deployment Complexities
  - ✓ Debugging and Testing Complexities

EJB要严格地继承各种不同类型的接口，类似的或者重复的代码大量存在，而配置也是复杂和单调。因此，对于大多数初学者来说，学习EJB实在是一件代价高昂的事情，且较低的开发效率，极高的资源消耗，都造成了EJB的使用困难

# Spring的出现

- Spring出现的初衷就是为了解决类似的这些问题。它是一个轻量级控制反转(IoC)和面向切面(AOP)的服务器容器框架，采用依赖注入(Dependency Injection (DI))的设计模式。
- Spring一出现即引起了开发者极大兴趣并得到了大量正面的反馈，以未来企业Java应用首选框架的姿态展现。
- EJB也从各个框架中不断吸取养分，引入了一系列新的特性，也在不断的改版进化。出现了Spring 和EJB互相统一融合的趋势。

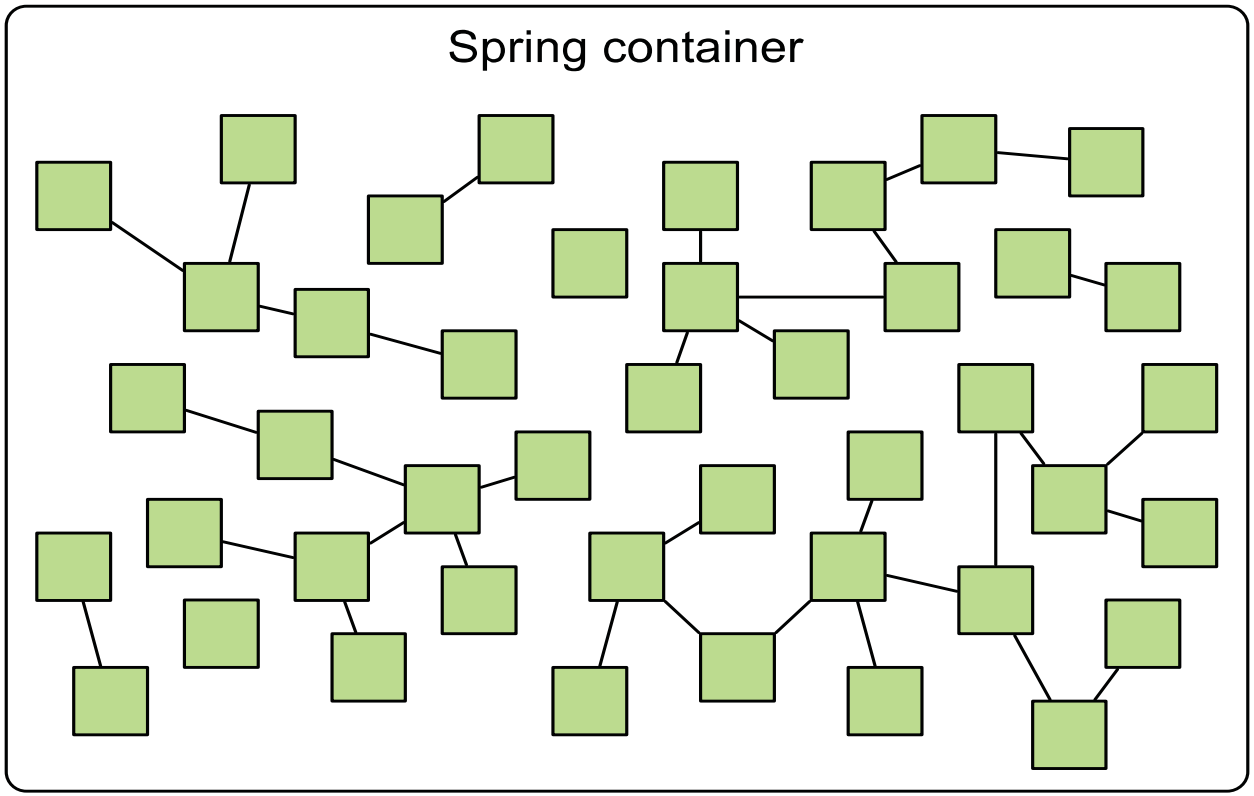



**Figure 1.4** **In a Spring application, objects are created, are wired together, and live in the Spring container.**

# POJO对象

- Spring和EJB框架结构都有一个共同核心设计理念：将中间件服务传递给耦合松散的POJO对象 (Plain Old Java Objects)

- POJO类没有任何特别之处，无需装饰，不继承自某个类。但是，它却可以作为组件(Component)使用，发挥强大的功能。

# POJO 对象

```
public class Student {
    private Service service;
    public Student(Service s){
     service=s;
    }
    public void learn(){
            service.serve();
    }
}
```

# 配置的例子

```
package soundsystem;

public class BlankDisc implements CompactDisc {

  private String title;
  private String artist;

  public BlankDisc(String title, String artist) {
    this.title = title;
    this.artist = artist;
  }

  public void play() {
    System.out.println("Playing " + title + " by " + artist);
  }

}
```

```
<bean id="compactDisc" class="soundsystem.BlankDisc">
  <constructor-arg value="Sgt. Pepper's Lonely Hearts Club Band" />
  <constructor-arg value="The Beatles" />
  <constructor-arg>
    <set>
      <value>Sgt. Pepper's Lonely Hearts Club Band</value>
      <value>With a Little Help from My Friends</value>
      <value>Lucy in the Sky with Diamonds</value>
      <value>Getting Better</value>
      <value>Fixing a Hole</value>
      <!-- ...other tracks omitted for brevity... -->
    </set>
  </constructor-arg>
</bean>
```

```
@Bean
public CompactDisc sgtPeppers() {
  return new SgtPeppers();
}
```

# 与web容器相关的技术

- 反射 Reflection
- 注解 Annotation
- 依赖注入Dependency Injection
- 面向切面的编程 Aspect Oriented Programming

# 反射概念

- 反射的概念是由Smith在1982年首次提出的，主要是指程序可以访问、检测和修改它本身状态或行为的一种能力。
- 它首先被程序语言的设计领域所采用，并在Lisp和面向对象方面取得了成绩。
- Common Lisp

《黑客与画家》

# 反射概念

- Reflection 是 Java 程序开发语言的特征之一，它允许运行中的 Java 程序对自身进行检查，或者说"自审"，并能直接操作程序的内部属性。

- Java反射的功能：
  - ✓ 可以判断运行时对象所属的类
  - ✓ 可以判断运行时对象所具有的成员变量和方法
  - ✓ 通过反射甚至可以调用到private的方法
  - ✓ 生成动态代理

# 实现Java反射的类

- 1)Class：它表示正在运行的Java应用程序中的类和接口
- 2)Field：提供有关类或接口的属性信息，以及对它的动态访问权限
- 3)Constructor：提供关于类的单个构造方法的信息以及对它的访问权限
- 4)Method：提供关于类或接口中某个方法信息

Class类是Java反射中最重要的一个功能类，所有获取对象的信息(包括：方法/属性/构造方法/访问权限)都需要它来实现

```
import java.lang.reflect.*;
public class DumpMethods {
        public static void main(String args[]) {
                try {
                        Class c = Class.forName(args[0]);
                        Method m[] = c.getDeclaredMethods();
                        for (int i = 0; i < m.length; i++){
                                System.out.println(m[i].toString());
                } catch (Throwable e) {
                        System.err.println(e);
                }
        }
}
```

```
import java.awt.*;
class SampleGet {
        public static void main(String[] args) {
                Rectangle r = new Rectangle(100, 325);
                printHeight(r);
        }
        static void printHeight(Rectangle r) {
                Field heightField;
                Integer heightValue;
                Class c = r.getClass();
                try {
                        heightField = c.getField("height");
                        heightValue = (Integer) heightField.get(r);
                        System.out.println("Height: " +
heightValue.toString());
                } catch (NoSuchFieldException e) {
                        System.out.println(e);
                } catch (SecurityException e) {
                        System.out.println(e);
                } catch (IllegalAccessException e) {
                        System.out.println(e);
                }
```

# 注解（Annotation）

- 也叫元数据，一种代码级别的说明。它是JDK1.5及以后版本引入的一个特性，与类、接口、枚举是在同一个层次。它可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明，注释。
- 注解是以'@注解名'在代码中存在的，不会直接影响到程序的语义，只是作为注解存在。
- 可以通过反射机制编程实现对这些元数据（用来描述数据的数据）的访问。

# 注解的例子

- 标记注解（marker annotation ）
  - ✓ @Override
- 配置注解
  - ✓ @Table(name = "Customer", schema = "APP")
- 从某种角度来说，可以把注解看成是一个XML元素，该元素可以有不同的预定义的属性。
- 属性的值是可以在声明该元素的时候自行指定的。
- 在代码中使用注解，就相当于把一部分元数据从XML文件移到了代码本身之中，在一个地方管理和维护。

# 正确使用 注解

- 在一般的开发中，只需要通过阅读相关的API文档来了解每个注解的配置参数的含义，并在代码中正确使用即可。

- 在有些情况下，可能会需要开发自己的注解。这在库的开发中比较常见。
- 注解的定义有点类似接口。

# 注解定义

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Assignment {
    String assignee();
    int effort();
    double finished() default 0;
}
```

@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型。可以通过default来声明参数的默认值

# 依赖注入的概念

- 在应用中，这些组件及对象生存在容器中，而容器将负责对象的创建、对象间的关联，并管理对象的生命周期。

- 负责对象创建和关联的，就是“依赖注入”（Dependency Injection）技术。

# 依赖注入的概念

- DI通过截取执行上下文或在运行时信息，为组件和对象注入它们之间的依赖关系。
- DI能提供类似胶水的功能，自动的把某些对象"缠绕"起来实现对象之间的协作。
- 对象本身并不关心这种"缠绕"，对这种框架结构也没有什么依赖。

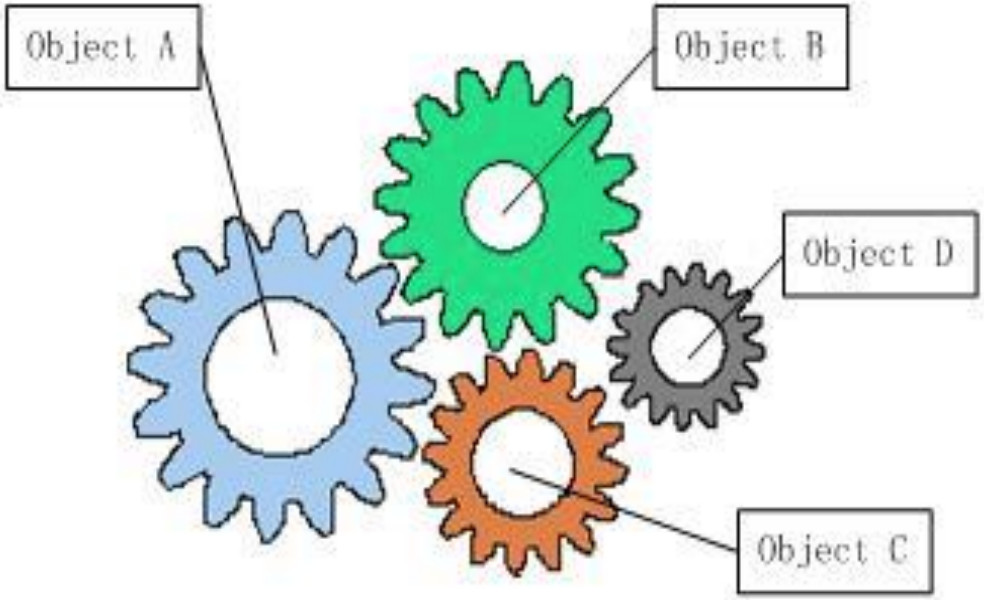
# 依赖注入和解耦合

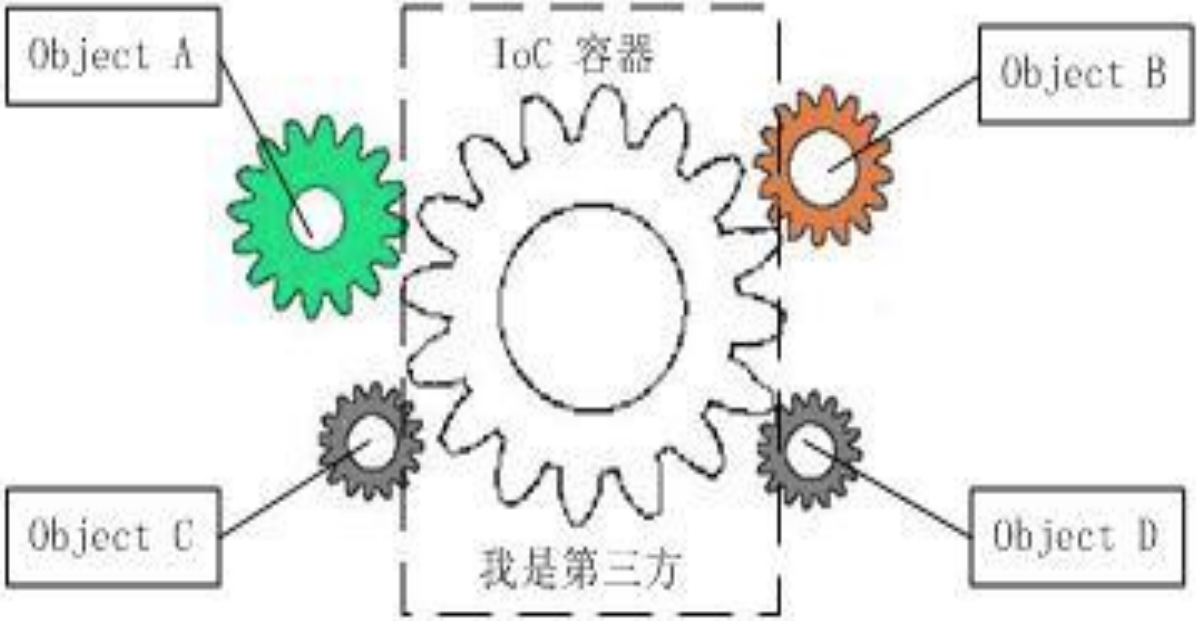
- 开发者可专注于业务逻辑， POJO类对象可脱离框架，进行单元测试。

- POJO类并不需要继承框架的类或实现其接口，开发者能够极其灵活地搭建继承结构和建造应用。

# 解耦合

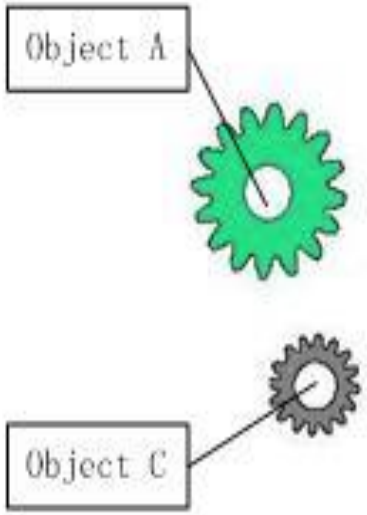
- 耦合关系不仅会出现在对象与对象之间，也会出现在软件系统的各模块之间，以及软件系统和硬件系统之间。
- 如何降低系统之间、模块之间和对象之间的耦合度，是软件工程永远追求的目标之一。
- 为了解决对象之间的耦合度过高的问题，软件专家Michael Mattson提出了IOC（反转控制）理论，用来实现对象之间的"解耦"。

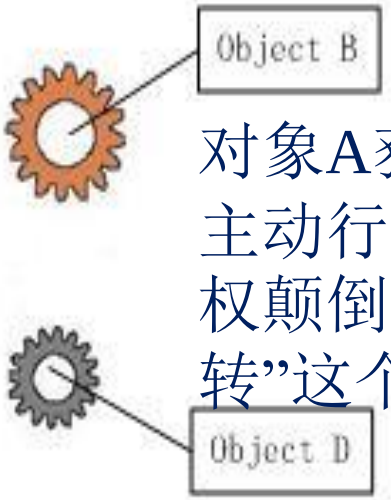

Object A
Object B
Object D
Object C


Object A
IoC 容器
Object B
Object C
我是第三方
Object D

# 控制反转(IoC)

- 简单来说就是把复杂系统分解成相互合作的对象，这些对象类通过封装以后，内部实现对外部是透明的，从而降低了解决问题的复杂度，而且可以灵活地被重用和扩展。




对象A获得依赖对象B的过程，由主动行为变为了被动行为，控制权颠倒过来了，这就是“控制反转”这个名称的由来。

# 例子说明: DITest

# 面向切面的编程

- AOP即为Aspect Oriented Programming的缩写，意为面向切面的编程，是通过预编译方式和运行期动态代理的方式，实现程序功能的统一维护的一种技术。

# 方面的概念

- AOP将应用系统分为两部分:
  - ✓核心业务逻辑（Core business concerns）
  - ✓横向的通用逻辑：所谓的"方面"(Crosscutting enterprise concerns)

- 持久化管理（Persistent）、事务管理（Transaction Management）、安全管理（Security）、日志管理（Logging）和调试管理（Debugging）

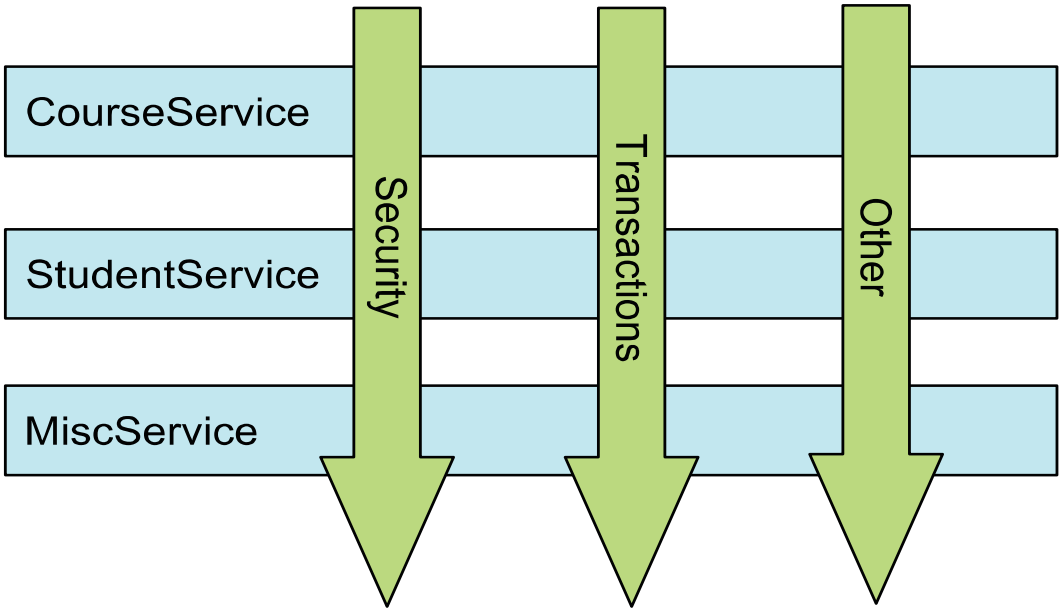



**Figure 4.1** **Aspects modularize cross-cutting concerns, applying logic that spans multiple application objects.**

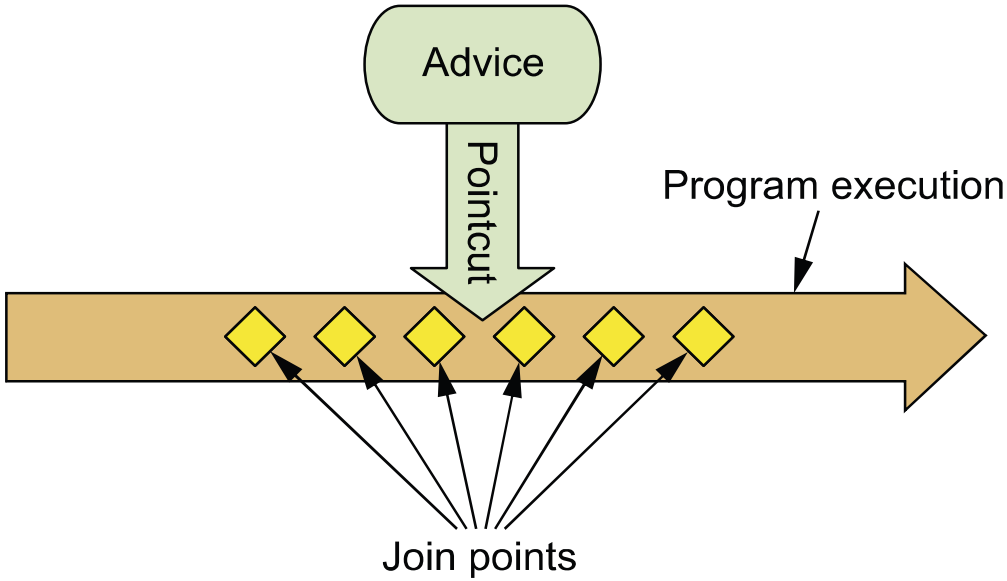
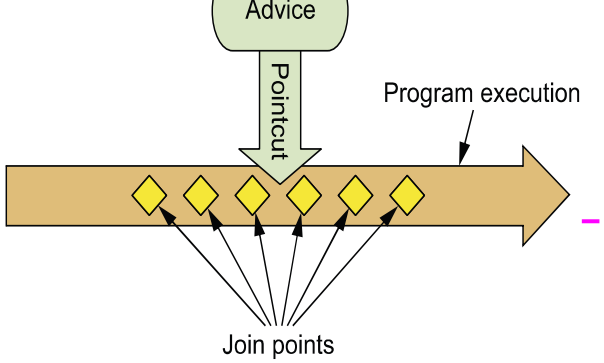



**Figure 4.2** **An aspect's functionality (advice) is woven into a program's execution at one or more join points.**

# 基本概念




- 连接点（Joinpoint）：
  - ✓ 程序执行过程中某一个方面可以切入（plug in）的点，如特定方法的调用或特定的异常被抛出。
- 通知（Advice）：（通知定义了"什么"和"何时"）
  - ✓ 在特定的连接点，AOP框架执行的动作。
  - ✓ 各种类型的通知包括"around"、"before"和"throws"通知。许多AOP框架包括Spring都是以拦截器做通知模型，维护一个"围绕"连接点的拦截器链。
- 切入点（Pointcut）：（切点就定义了"何处"）
  - ✓ 指定通知将被引发的一系列连接点的集合。AOP框架必须允许开发者指定切入点。例如，使用正则表达式来匹配连接点的集合。

# 基本概念

- 方面（Aspect）
  - ✓通知和切入点合在一起称为方面，是一个关注点的模块化，这个关注点实现可能横切多个对象。比如，事务管理、安全管理等都是横切关注点例子。方面用Spring的Advisor或拦截器实现。
- 引入（Introduction）
  - ✓引入允许添加新的方法或字段到被通知的类。Spring允许引入新的接口到任何被通知的对象。例如，你可为一个类引入IsModified接口和保存其状态数据的对象来记录对象被修改的状态，而不必更改原有类的代码。

# 基本概念

- 目标对象（Target Object）
  - ✓包含连接点的对象，也被称作被通知或被代理对象。
- AOP代理（AOP Proxy）
  - ✓AOP框架创建的对象，包含通知。在Spring中，AOP代理可以是JDK动态代理或CGLIB代理。
- 编织（Weaving）
  - ✓把方面应用到目标对象，从而创建一个新的代理对象的过程。编织可以在编译时完成（例如使用AspectJ编译器），也可以在运行时完成。Spring和其他纯Java AOP框架一样，在运行时完成织入。

# Spring AOP概念

| 概念 | 含义 |
| --- | --- |
| Aspect | 切面 |
| Join Point | 连接点，Spring AOP里总是代表一次方法执行 |
| Advice | 通知，在连接点执行的动作 |
| Pointcut | 切入点，说明如何匹配连接点 |
| Introduction | 引入，为现有类型声明额外的方法和属性 |
| Target object | 目标对象 |
| AOP proxy | AOP 代理对象，可以是 JDK 动态代理，也可以是 CGLIB 代理 |
| Weaving | 织入，连接切面与目标对象或类型创建代理的过程 |

- 例子讲解： Teacher & Student
- 另外，请参考 ASpectJ 语言的编程

# EJB（Enterprise Java Bean）

# EJB

- EJB（Enterprise Java Bean），即企业JavaBean是一个可重用的，可移植的Java EE组件。
- EJB由封装了业务逻辑的多个方法组成。多个远程和本地客户端可以调用这个方法。

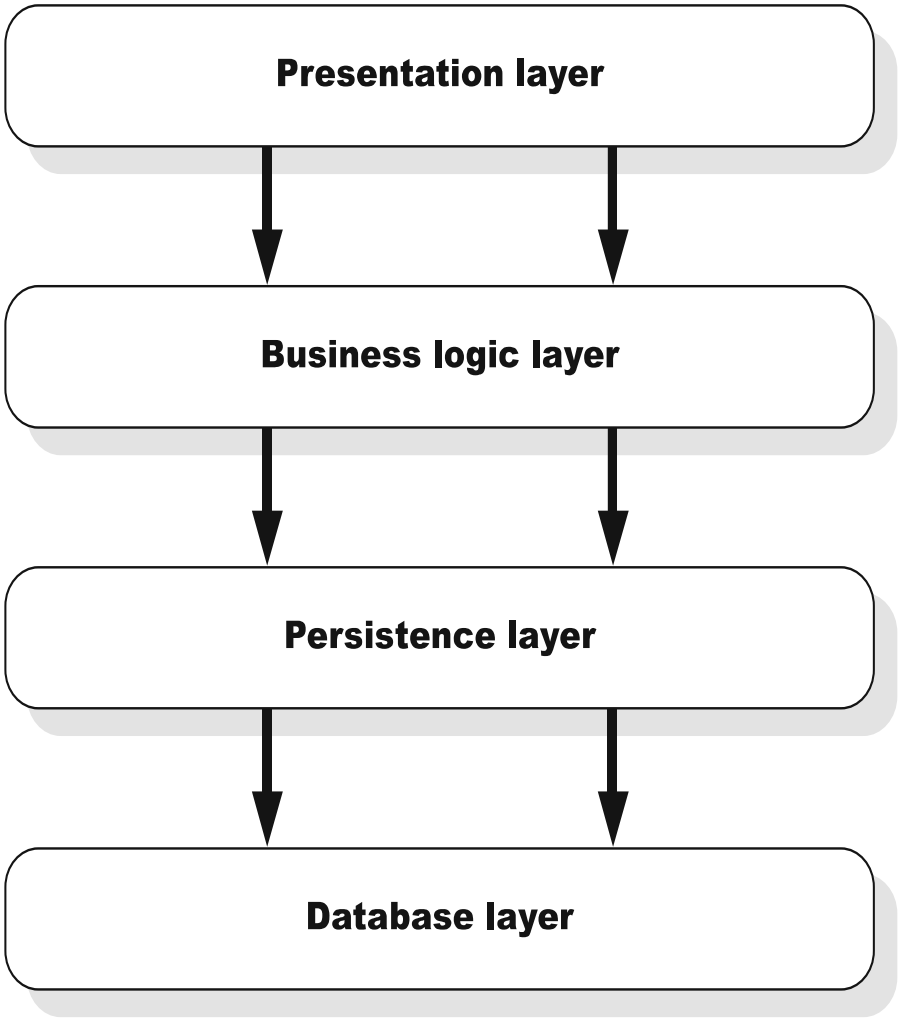



**Figure 1.1 Most traditional Enterprise applications have at least four layers: the presentation layer is the actual user interface and can either be a browser or a desktop application; the business logic layer defines the business rules; the persistence layer deals with interactions with the database; and the database layer consists of a relational database such as Oracle database that stores the persistent objects.**

# EJB

- EJB运行在一个容器里，允许开发者只关注与bean中的业务逻辑而不用考虑象事务支持，安全性和远程对象访问等复杂和容易出错的事情。

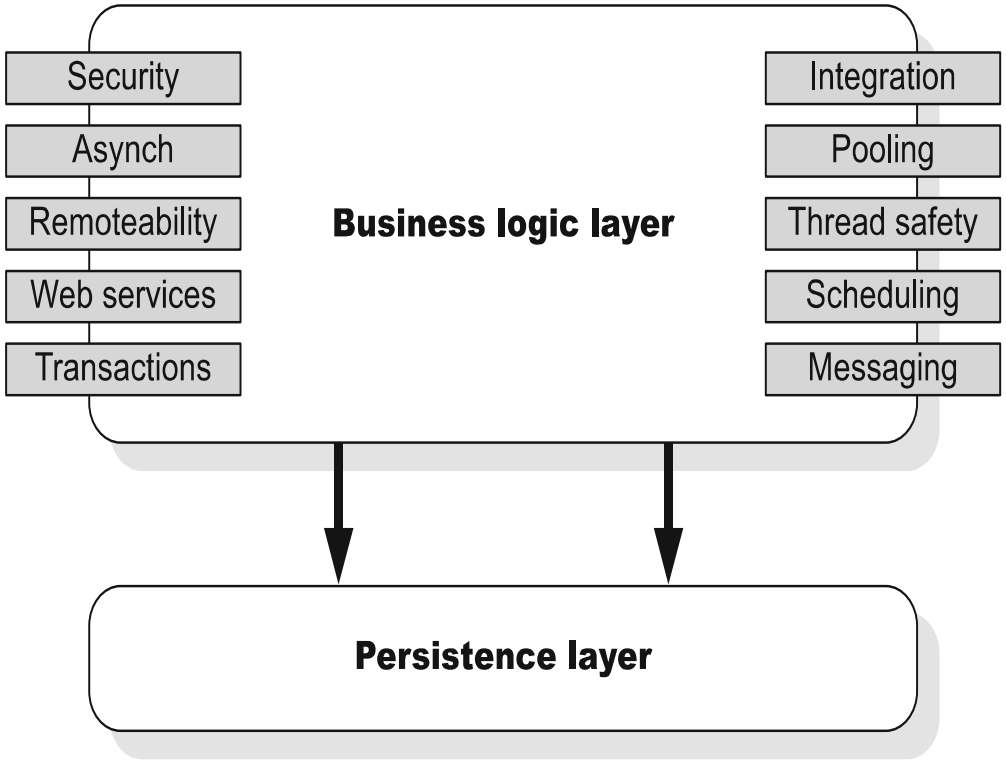



**Figure 1.2 The component services offered by EJB 3 at the supported application layer. Note that each service is independent of the others, so you are (for the most part) free to pick the features important for your application.**

- EJB以POJO对象形式开发，开发者可以用元数据注释（注解）来定义容器如何管理这些Bean。

- EJB3中包含有两种类型：会话Bean和消息驱动Bean
  - ✓ 会话Bean完成一个清晰的解耦的任务，例如检查客户账户历史记录。
  - ✓ 消息驱动Bean用于接收异步JMS消息。

# 会话Bean

- 会话Bean(Session Bean)被客户端调用用于处理某一具体的操作业务，比如代表着业务流程中象"处理订单"这样的动作。
- "会话"意味着Bean只存在于某一时间段，而当服务器容器关闭或故障时将被销毁。
- 基于是否维护过度状态，会话Bean可分为有状态或者无状态。

# 无状态会话Bean

- 无状态会话Bean不维护会话状态，其服务任务须在一个方法调用中结束。
- 其没有中间状态，也不保持追踪方法传递的信息。
- 一个无状态业务方法的每一次调用都独立于它的前一个调用。

# “会话池”技术

- 会话池中的无状态会话Bean能够被共享，当客户端请求一个无状态的Bean实例时，它可以从池子中选一个空闲状态的Bean实例进行调用处理。
- 当请求达到了会话池设置的最大数量，新请求将被加入队列等待无状态Bean的服务。
- EJB通过添加@Stateless标注来指定一个Java Bean作为一个无状态会话Bean被部署和管理。

# 有状态的会话Bean

- 维护一个跨越多个方法调用的会话状态。当一个客户端请求一个有状态会话Bean实例时，客户端将会得到一个会话实例，该Bean的状态只为该客户端维持。
  - ✓ 例如在线购物篮应用
  - ✓ 不同的客户端调用，都将产生新的有状态的会话Bean实例。
  - ✓ 会话Bean是暂时的，因为该状态在会话结束，系统崩溃或者网络失败时都不会被保留。
- @Stateful来标注

# 消息驱动Bean

- Message Driven Bean，MDB提供了一个实现异步通信的方法，该方法比直接使用Java消息服务（JMS）更容易。
- 可以通过创建MDB接收异步JMS消息。当一个业务执行的时间很长，而执行结果无需实时向用户反馈时，很适合使用消息驱动Bean。
  - ✓ 如订单成功后给用户发送一封电子邮件或发送一条短信等。
- 对客户机来说，消息驱动bean是一个在服务器上实现某些业务逻辑的JMS消息使用者。

- 消息驱动bean(MDB)通常要实现MessageListener接口，实现该接口的onMessage(Message message)方法处理接收的jms消息。
- 可以通过@MessageDriven标注来指定一个Bean是消息驱动Bean。

# The New Enterprise JavaBean

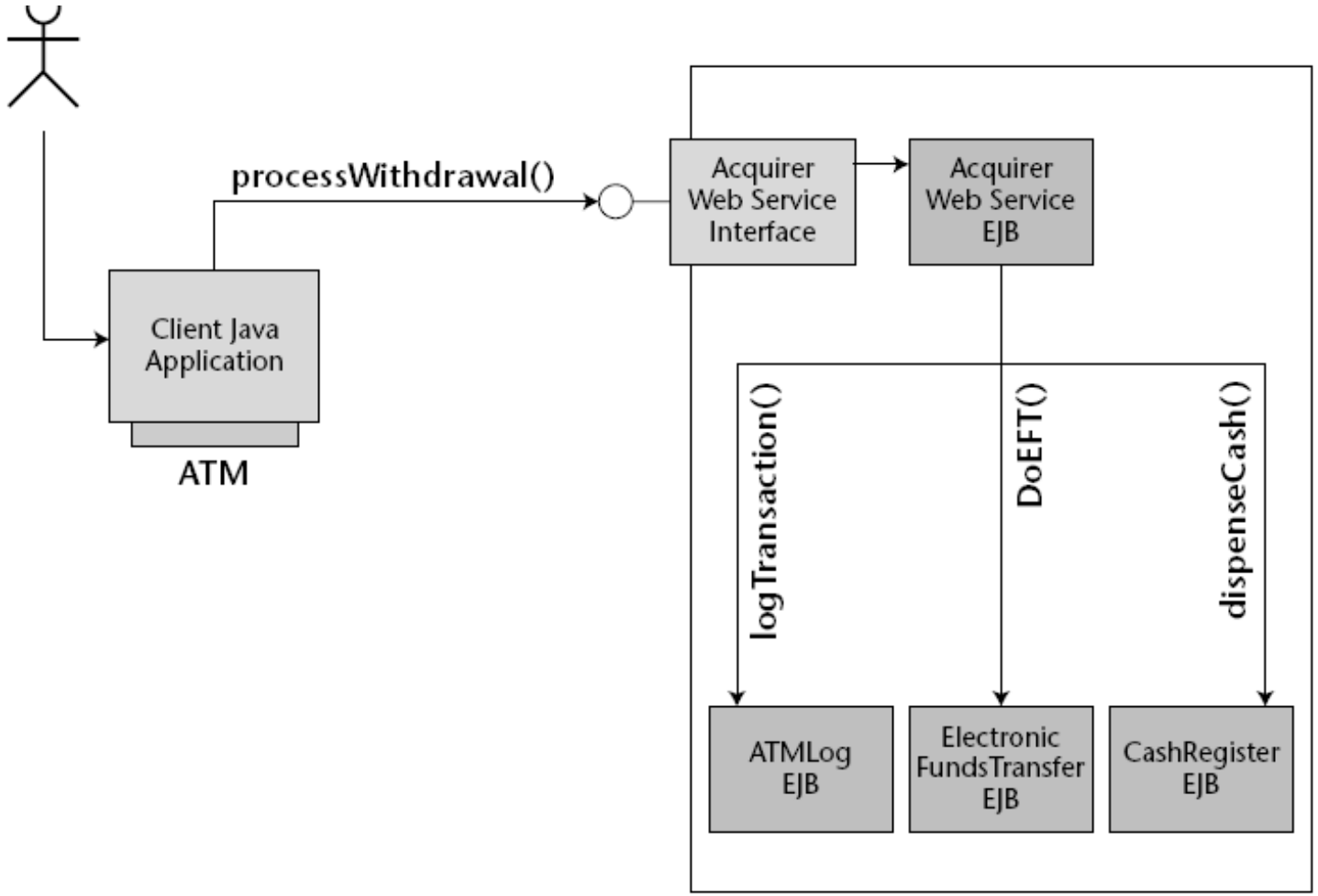



**Figure 3.1** ATM cash withdrawal scenario accomplished using EJBs.

# The New Enterprise JavaBean

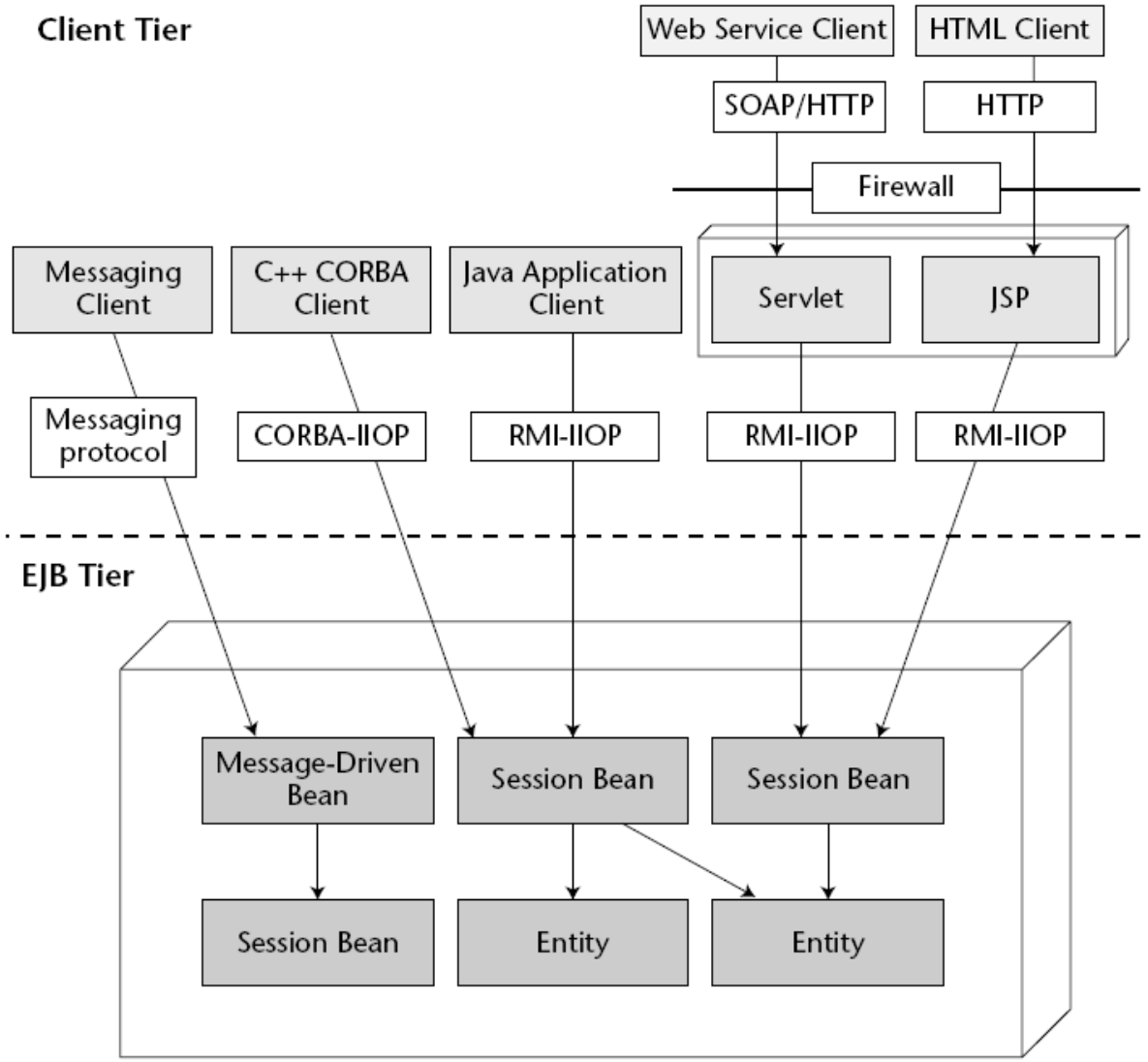



**Figure 3.2** EJB sub-system: The various clients and beans.

# The New Enterprise JavaBean

- RMI-IIOP: The Protocol of Bean
- EJB and Location Transparency
  - ✓ JNDI is an enabler of location transparency
    - ✓ Provides a standard API to access different kinds of naming and directory services within a java program
    - ✓ Provides two API
      - Naming API
      - Directory API

# The New Enterprise JavaBean

- The EJB 3.0 Simplified API
  - ✓ No home and object interfaces
  - ✓ No component interface
  - ✓ Use of Java metadata annotations
  - ✓ Simplification of APIs for accessing bean's environment

# The New Enterprise JavaBean

- Elimination of Home and Object Interface
  - Pre EJB 3.0
    - The Home interface served as a factory for creating reference to the EJB Object.
    - The EJB Object Interfaces was to provide client view for an EJB
- Elimination of Component Interface
  - Pre EJB 3.0
    - Components need to implement javax.ejb.SessionBean or javax.ejb.MessageDrivenBean
      - Carried the various life cycle method

# The New Enterprise JavaBean

- Use of Annotations
  - ✓ Annotations are used to provide the additional context to program.
  - ✓ Can be applied to various elements in Java
    - ✓ Methods, variables, constructors, package declarations

```
@Failsafe
public interface SomeRMIExample extends java.rmi.Remote
{
      ...
}
```

# The New Enterprise JavaBean

- ✓ Annotations and Bean Development
  - ✓ EJB 3.0 is a mix of metadata tags and code structs

```
@Stateful
public class exampleBean implements BeanBusinessInterface
{
      @Remove
      public void removeBean()
      {
        // Close any resources that were opened to service requests.
      }
}
```

# The New Enterprise JavaBean

- Annotations and Deployment Descriptions
  - EJB 3.0 annotations can be used in lieu of deployment descriptors.
- Container Specific Deployment Description
  - Describe the value-added services of container

# The New Enterprise JavaBean

- The Good, Bad and Ugly of Deployment Annotation
  - ✓ Convenient
    - ✓ Place configuration along with the bean's logic.
  - ✓ Tricky
    - ✓ When the bean provider and deployer is not the same person.
- Simplified Access to environment

# Package and Deployment of the "New" Bean

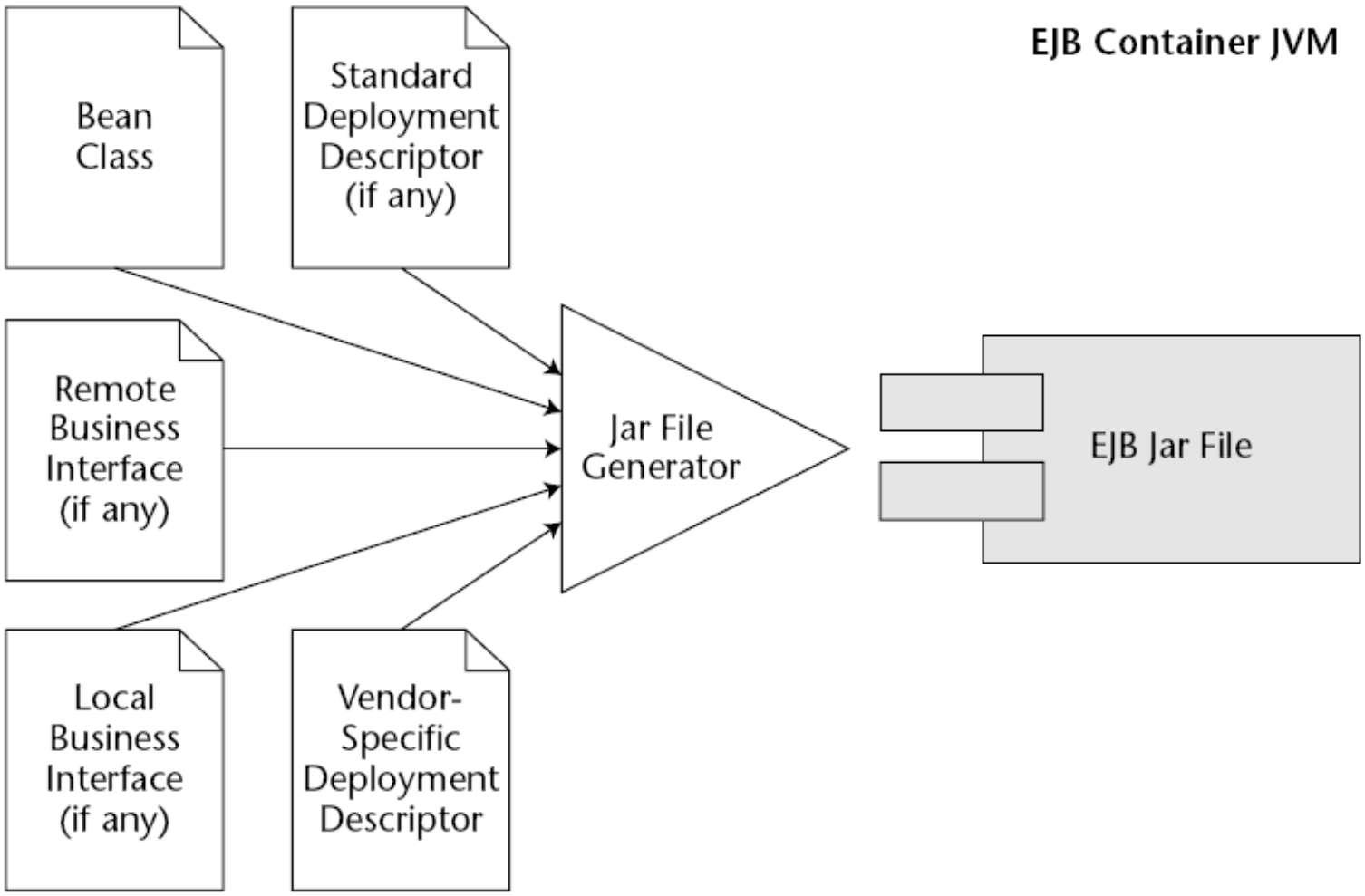



**Figure 3.5** Creating an Ejb-jar file.

# The New Enterprise JavaBean

- Example of EJB 3.0 Bean
  - ✓ The Business Interface

```
package examples.session.stateless;

/**
 * This is the Hello business interface.
 */

public interface Hello {
    public String hello();
}
```

**Source 3.1** Hello.java.

# The New Enterprise JavaBean

## The Bean Class

```
package examples.session.stateless;

import javax.ejb.Remote;
import javax.ejb.Stateless;

/**
 * Stateless session bean.
 */
@Stateless
@Remote(Hello.class)
public class HelloBean implements Hello {
     public String hello() {
        System.out.println("hello()");
        return "Hello, World!";
    }
}
```

**Source 3.2** HelloBean.java.

# The New Enterprise JavaBean

- The Deployment Descriptor

```
<?xml version="1.0" encoding="UTF-8" ?>
 <ejb-jar  xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_3_0.xsd"
version="3.0">
 <enterprise-beans>
 </enterprise-beans>
</ejb-jar>
```

**Source 3.3** Ejb-jar.xml.

# The New Enterprise JavaBean

## The Client

```
package examples.session.stateless;

import javax.naming.Context;
import javax.naming.InitialContext;

/**
 * This class is an example of client code which invokes
 * methods on a simple, remote stateless session bean.
 */
public class HelloClient {

     public static void main(String[] args) throws Exception {
          /*
           * Obtain the JNDI initial context.
           *
           * The initial context is a starting point for
           * connecting to a JNDI tree.
           */
          Context ctx = new InitialContext();
          Hello hello = (Hello)
ctx.lookup("examples.session.stateless.Hello");

          /*
           * Call the hello() method on the bean.
           * We then print the result to the screen.
           */
          System.out.println(hello.hello());
     }
}
```

**Source 3.4** HelloClient.java.

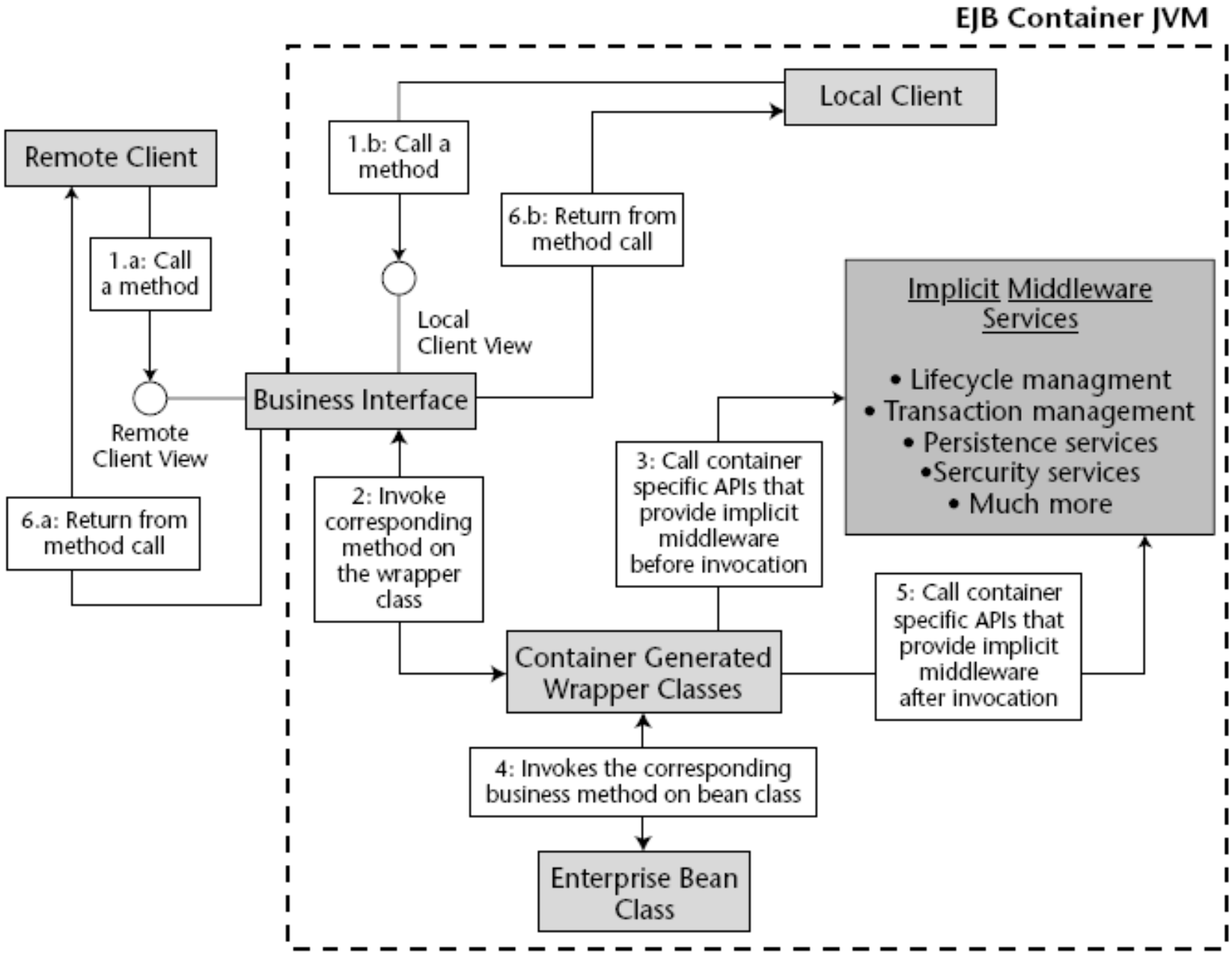



**Figure 3.4** EJB 3.0 programming model.

- EJB例子：ejb-remote

# 本章小结

- 概念简介
- Java EE和Spring框架介绍
- 相关技术介绍
  - ✓ 反射
  - ✓ 注解
  - ✓ 依赖注入
  - ✓ AOP编程
- EJB编程