分布计算环境

北京邮电大学计算机学院

主要内容

- 基于构件的软件体系结构
- J2EE/Java EE
- EJB
- 轻量级框架和EJB3.0

轻量级框架主要内容

- 概述
- Spring
- EJB 3.0

重量级框架

- 典型代表EJB 1.X、EJB2.X
- 非POJO实现类（ Plain Ordinary Java Objects ）,如EJB：
  - 三个构件：Bean类，Home接口和Remote接口
  - Bean类要实现相应的生命周期方法（回调函数）
- 开发的系统基本需要放置在一个容器系统中进行运行，并需要实现容器要求的接口
- 容器在实例化业务对象后，传给业务对象上下文，而业务对象本身要通过JNDI手段来定位或者pull出其他资源或者业务对象
- 这些容器因为基本针对大型企业应用，所以体积庞大，占用资源，内在服务多，启动比较慢
- 开发需要遵从的规则比较多，开发效率也比较低，很大一部分时间都用在了Deploy、Run这样的过程上，调试和测试比较困难

轻量级框架

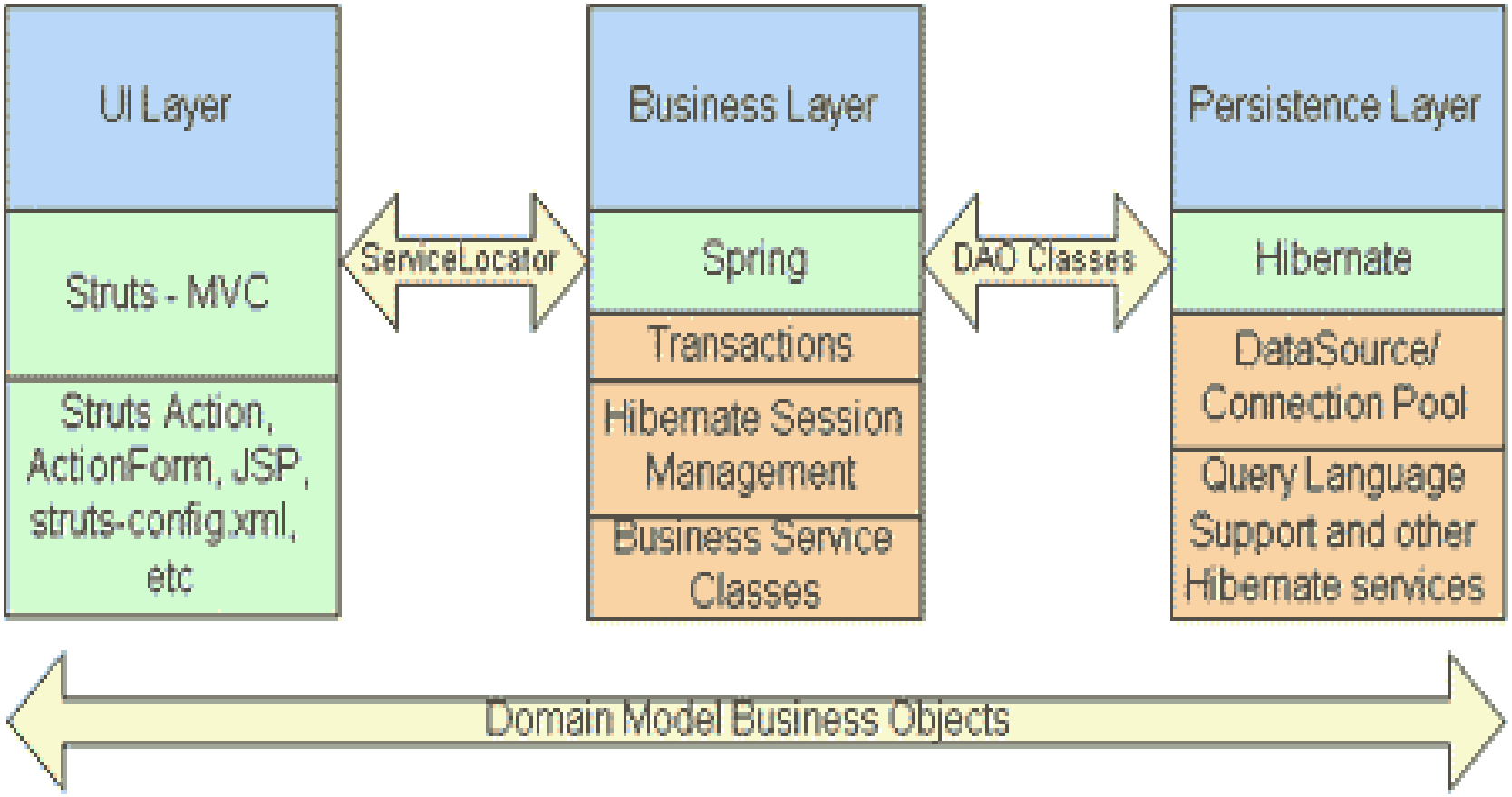
- 典型代表：Spring
- 以依赖注入（Dependency Injection）为代表的解耦合模式，可以让组件不去依赖容器（运行环境）的API
- 轻量级容器通过反向控制（Inversion of Control）让容器具有主动权，去管理插进来的组件，只要组件是符合标准的，就可以被轻量级容器管理
- 组件以POJO（Plain Old Java Object）的形式存在，只要你有Java.exe就可以运行它，不需要容器就可以实现测试行为
- 相对于重量级框架，基于轻量级框架进行系统开发的时候非常迅速

轻量、重量：量级主要由构件对容器的依赖性所决定，依赖性越小，越轻量

经典和轻量级Java EE

- Java EE平台已经成为电信、金融、电子商务、保险、证券等各行业的大型应用系统的首选开发平台。
- Java EE的开发大致分为两种方式
  - 以EJB 3.x+JPA为核心的经典Java EE开发平台
    - ➔在要求高度伸缩性、高度稳定性的企业应用（银行、保险）中，有一定的占有率
  - 以Spring为核心的轻量级Java企业开发平台
    - ➔在保留经典Java EE应用架构、高度可扩展性、高度可维护性的基础上，降低了JavaEE应用的开发、部署成本，对于大部分中小型企业应用是首选
    - ➔SSH：Struts2 + Spring + Hibernate
    - ➔SSM：Spring-mvc + Spring + MyBatis

SSH架构


UI Layer
Struts - MVC
Struts Action, ActionForm, JSP, struts-config.xml, etc
ServiceLocator
Business Layer
Spring
Transactions
Hibernate Session Management
Business Service Classes
DAO Classes
Persistence Layer
Hibernate
DataSource/ Connection Pool
Query Language Support and other Hibernate services
Domain Model Business Objects

SSM

- SpringMVC：Spring实现的一个Web框架，相当于Struts的框架，但是比Struts更加灵活和强大
- Spring framework
- Mybatis： Java持久层框架，在使用上相比Hibernate更加灵活，可以控制SQL的编写，使用 XML或注解（标注）进行相关的配置

轻量级框架主要内容

- 概述
- Spring
- EJB 3.0

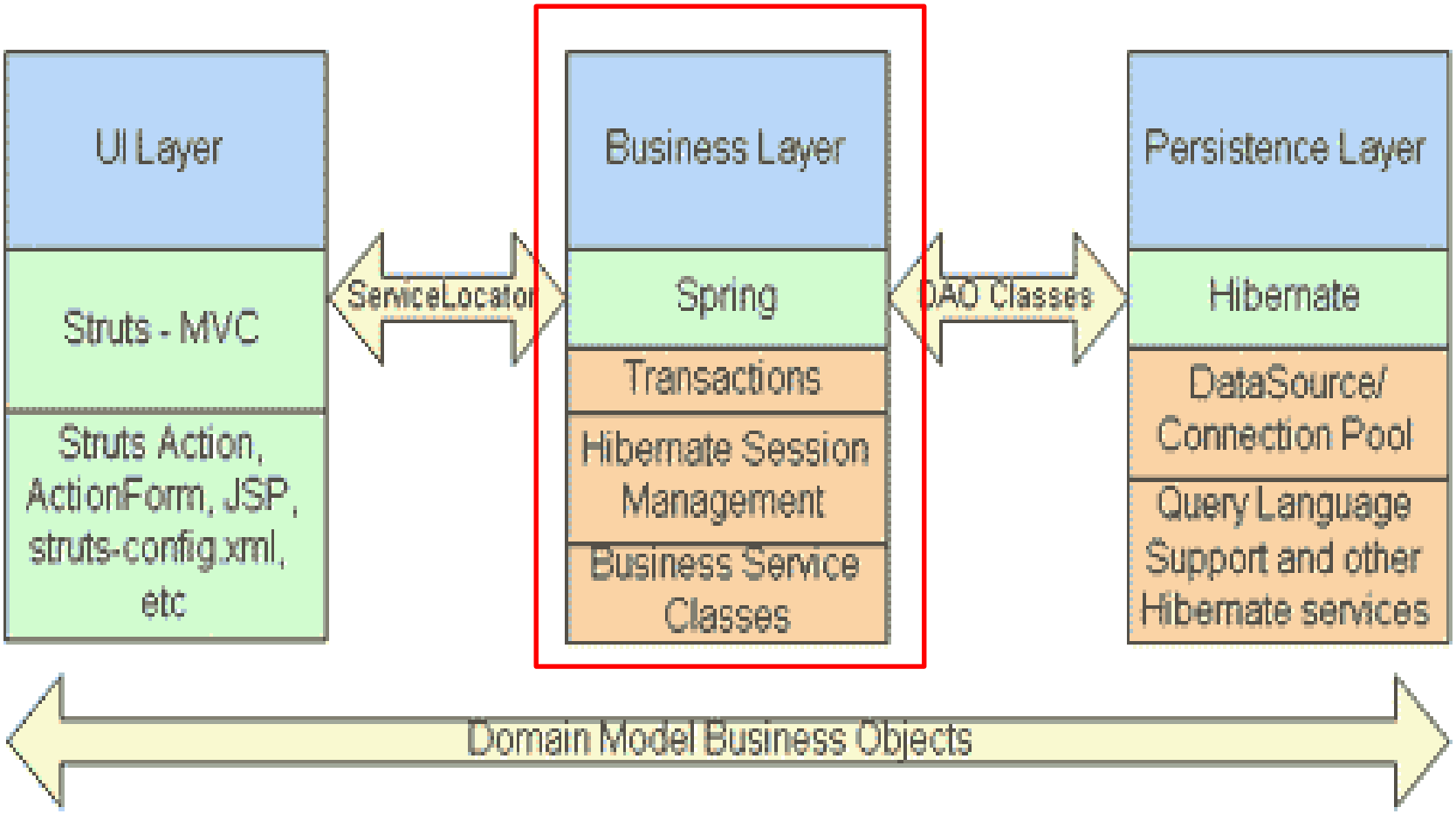
Spring主要内容

- 概述
- 控制反转与依赖注入
- 面向切面的编程AOP
- 服务抽象层

什么是Spring框架？

- Spring framework是一个轻量级的应用框架
  - 主要面向服务端应用，也可以面向普通应用
  - 由Rod Johnson开发，已发展成为Java EE开发中最重要的框架之一
    - ➔1.0（2004）、2.0（2006）、3.0（2009）、4.0（2013）、5.0（2017）、5.3（2020）
  - Spring面向应用逻辑
  - Struts、SpringMVC面向Web框架
  - Hibernate、MyBatis面向数据库映射（ORM）
- Spring面向组件（应用逻辑）本身，并且将组件和其他框架等粘合（glue）起来

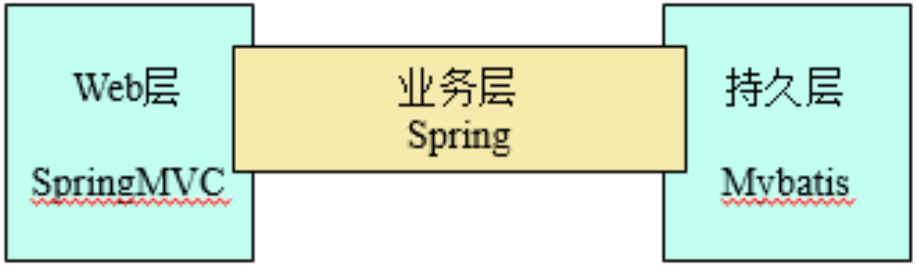
Spring 在SSH中的位置



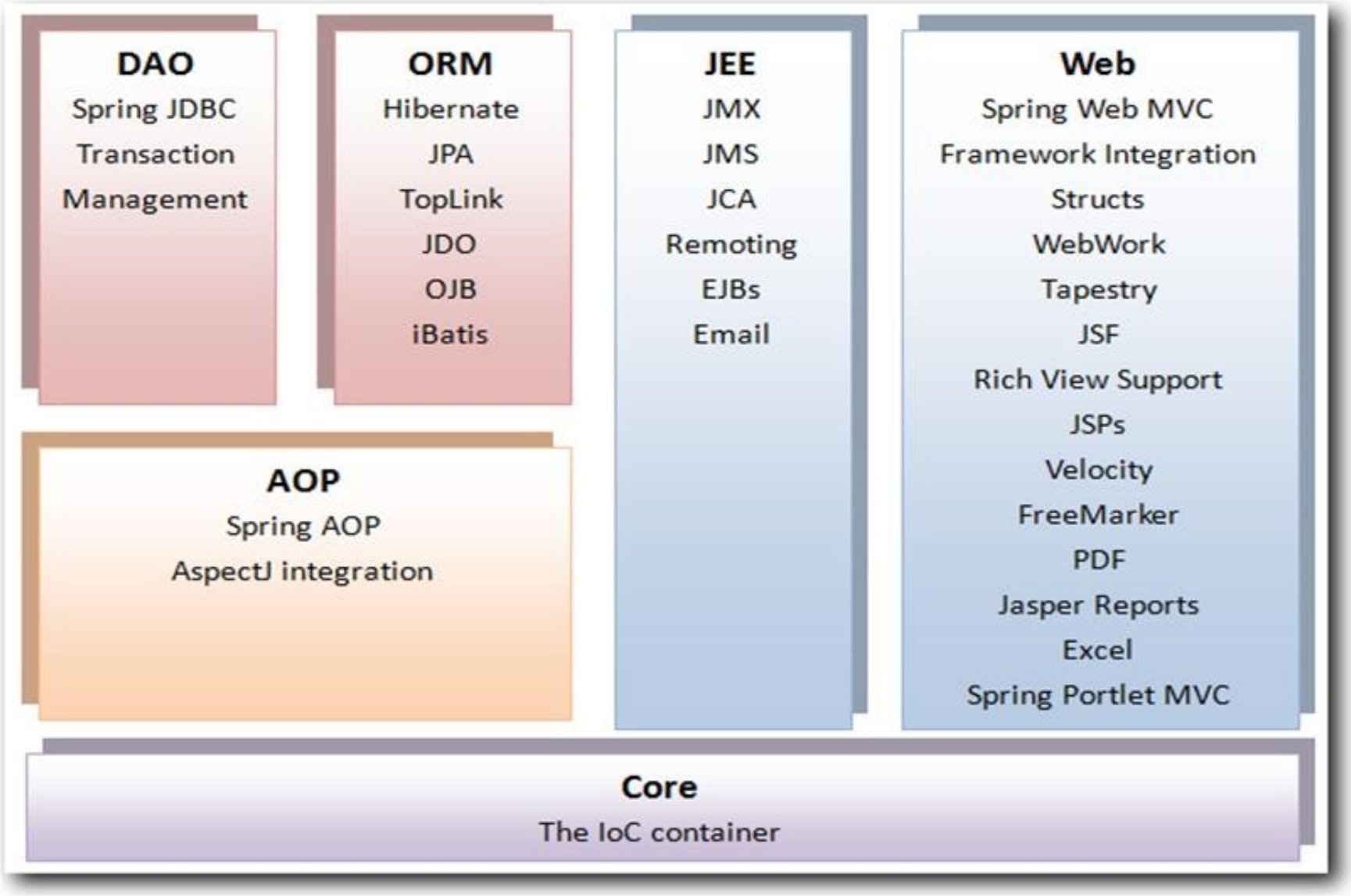
Spring 在SSM中的位置

Spring的理解

- Spring 抽象了大量Java EE应用中常用的代码，将它们抽象成一个框架，通过使用Spring 可以大幅度地提高开发效率、并保证整个应用具有良好的设计
- Spring 框架中整合了各种设计模式的应用：单例、工厂、代理等
- Spring 框架号称轻量级Java EE应用的一站式解决方案。但实际上Spring并未提供完整的持久层框架，但它可以与大部分持久层框架无缝整合
- Spring 更像一种中间层容器，向上可以与MVC框架无缝整合，向下可以与各种持久层框架无缝整合



Spring架构


DAO
Spring JDBC
Transaction
Management
ORM
Hibernate
JPA
TopLink
JDO
OJB
iBatis
JEE
JMX
JMS
JCA
Remoting
EJBs
Email
Web
Spring Web MVC
Framework Integration
Structs
WebWork
Tapestry
JSF
Rich View Support
JSPs
Velocity
FreeMarker
PDF
Jasper Reports
Excel
Spring Portlet MVC
AOP
Spring AOP
AspectJ integration
Core
The IoC container

Spring 核心的主要功能

- Spring核心提供了以下支持：
  - 支持“控制反转/依赖注入”的容器
    - 控制反转： Inversion of Control
    - 依赖注入：Dependency Injection， by Martin Fowler
  - AOP框架
    - AOP：Aspect-Oriented Programming
  - 服务抽象层
    - 以统一的方式来集成不同的标准接口和第三方API

这些功能使程序员可以使用POJO来编写功能强大并且伸缩性好的应用

Spring主要内容

- 概述
- 控制反转与依赖注入
- 面向切面的编程AOP
- 服务抽象层

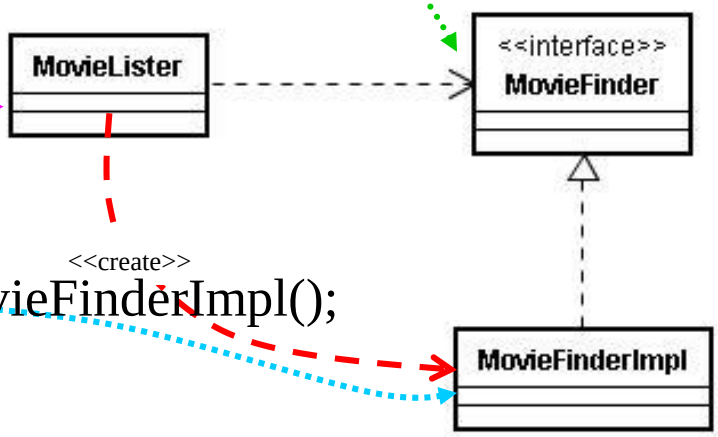
依赖性(Dependency)

- 当某个类继承另一个类、实现另一个接口或者调用其他类，我们就说该类依赖（depends-on）另一个类

```
public class MySessionBean implements javax.ejb.SessionBean {

  private javax.ejb.SessionContext ctx;

  public void ejbCreate() {}
  public void ejbRemove() {}
  public void ejbActivate() {}
  public void ejbPassivate() {}
  public void setSessionContext(SessionContext ctx) {
   this.ctx = ctx;
  }
    …(略)…
}
```

依赖性对程序造成负面影响

- 依赖性(Dependency)会对源代码的可重用性造成负面影响
  - 依赖性越低，程序的可重用性越高
- Liskov替换原则也可以看成是降低依赖性的方法
  - 将Client与实现类之间用接口分离，就可以解除Client与实现类之间的依赖性間的依赖关系



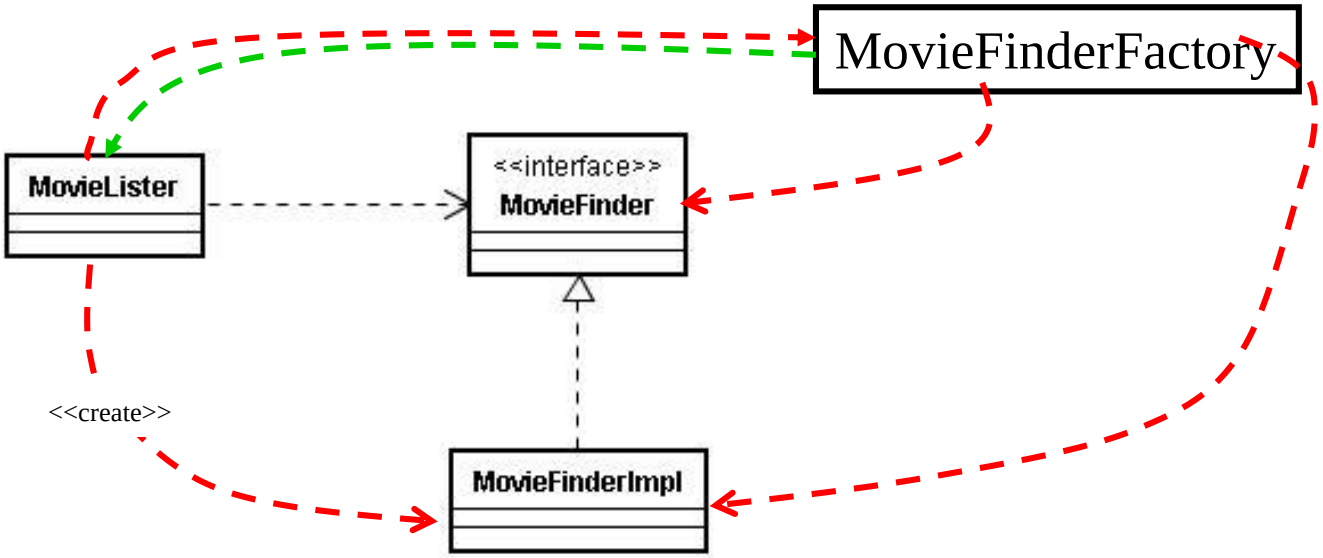

```
public MovieLister () {
    movieFinder = ( MovieFinder) new MovieFinderImpl();
}
```

控制反转与依赖注入

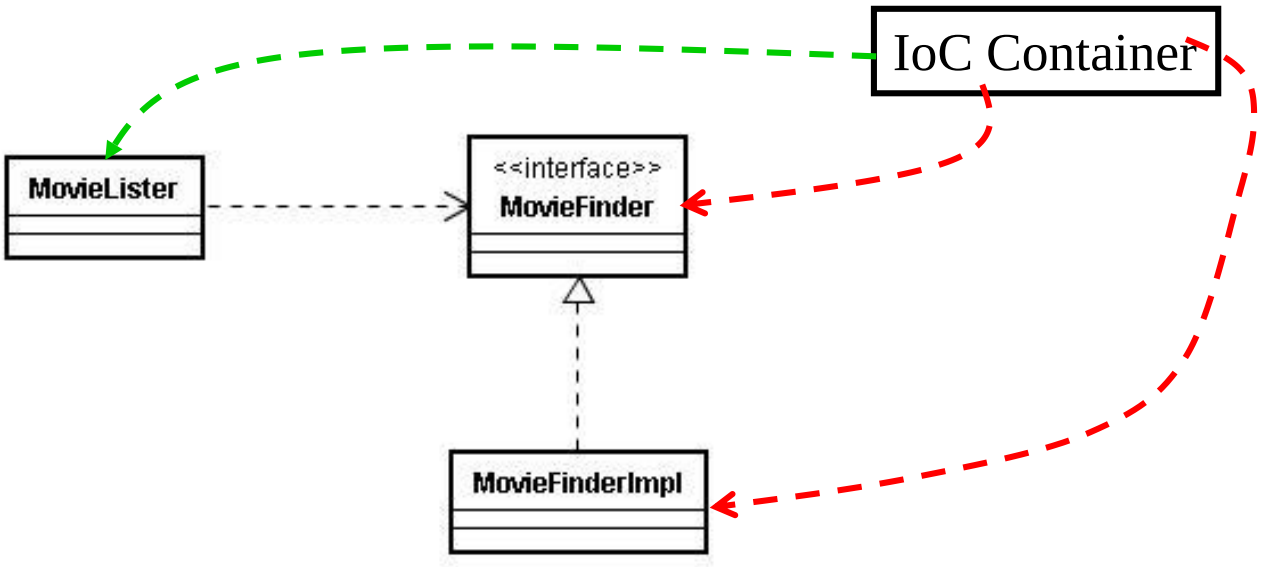
- 传统的程序开发，在一个对象中，如果要使用另外的对象，就必须得到它（自己new一个，或者从JNDI中查询一个），使用完之后还要将对象销毁（比如Connection等），对象始终会和其他的接口或类藕合起来。
- 在轻量级容器里，创建被调用者实例的工作不再由调用者来完成，而通常由容器来完成，因此称为控制反转（Inversion of Control，IoC），容器将所创建的实例注入调用者，因此称为依赖注入(Dependency Injection，DI)
- Martin Fowler建议使用依赖注入来描述“对象生成控制权”的倒置
- 颠覆了“使用对象之前必须创建”的基本Java语言定律，降低了模块之间的耦合度，提高了应用的灵活性和代码重用度

对象生成控制权的倒置

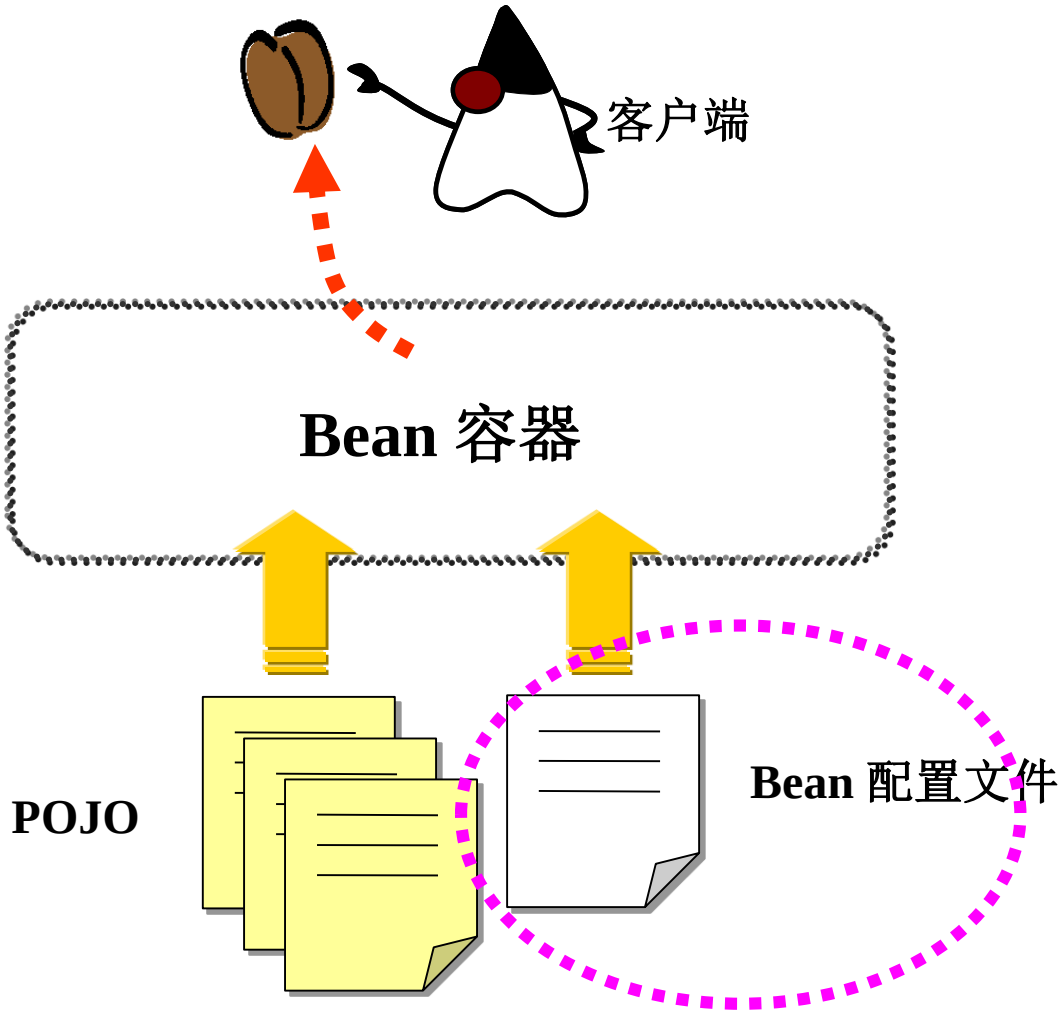
```
public MovieFinder create() {
    …
    return new MovieFinderImpl();
}
```



依赖注入（DI）

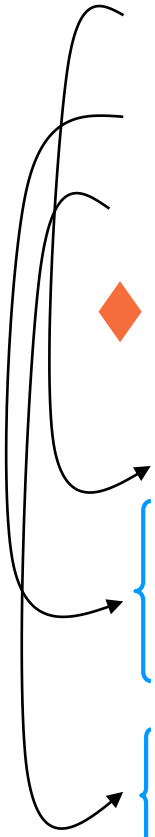

IoC Container
MovieLister
<<interface>>
MovieFinder
MovieFinderImpl

Spring Bean容器执行方式



Bean配置文件的基本结构

- Spring的Bean配置文件由三部分组成：
  - XML文件头
  - 标头定义段
  - Bean定义段
- DTD定义方式



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                       "http://www.springframework.org/dtd/spring-
beans-2.0.dtd">
<beans>
…
</beans>
```

XML Schema的定义方式

```
<? xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans“
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=“
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
   <bean id="…" class="…">
        <property name="…" value="…"/>
        …
   </bean>
   <bean id="…" class="…">
        …
   </bean>
   …
</beans>
```

也可使用注解，各有优缺点

DI例：通过setter

```
public class ExampleBean {

    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;

    public void setBeanOne(AnotherBean b) { beanOne = b; }
    public void setBeanTwo(YetAnotherBean b) { beanTwo = b; }
}
```

```
<bean id="exampleBean" class="eg.ExampleBean">
    <property name="beanOne"><ref bean="anotherExampleBean"/></pro
    <property name="beanTwo"><ref bean="yetAnotherBean"/></propert
</bean>

<bean id="anotherExampleBean" class="eg.AnotherBean"/>
<bean id="yetAnotherBean" class="eg.YetAnotherBean"/>
```

DI：通过构造函数

```
public class ExampleBean {
    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;
    private int i;
    public ExampleBean(AnotherBean b1, YetAnotherBean b2, int i) {
        this.beanOne = b1;
        this.beanTwo = b2;
        this.i = i;
    }
}
```

```
<bean id="exampleBean" class="eg.ExampleBean">
    <constructor-arg><ref bean="anotherExampleBean"/></constructor-arg>
    <constructor-arg><ref bean="yetAnotherBean"/></constructor-arg>
    <constructor-arg><value>1</value></constructor-arg>
</bean>

<bean id="anotherExampleBean" class="eg.AnotherBean"/>
<bean id="yetAnotherBean" class="eg.YetAnotherBean"/>
```

如何在代码中主动获取一个Bean?

- 例：

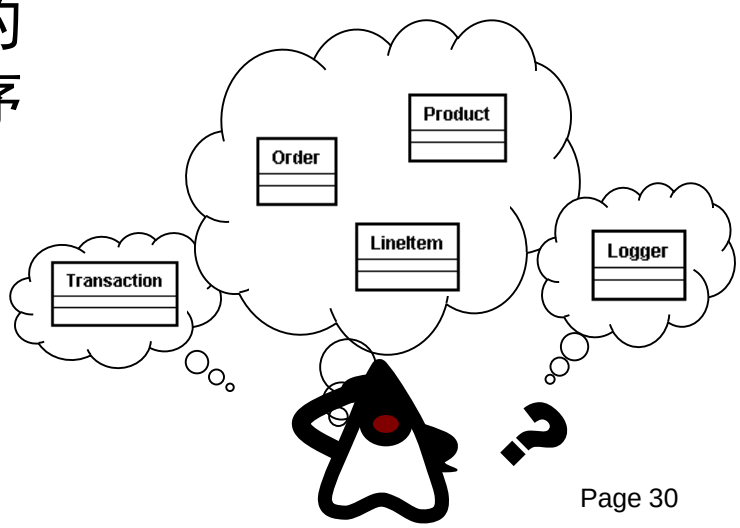
```
ApplicationContext ac = new FileSystemXmlApplicationContext("applicationContext.xml");
ac.getBean("userService");

//在application.xml中配置：
<bean id="userService" class="com.service.UserServiceImpl"></bean>
```

Spring主要内容

- 概述
- 控制反转与依赖注入
- 面向切面的编程AOP
- 服务抽象层

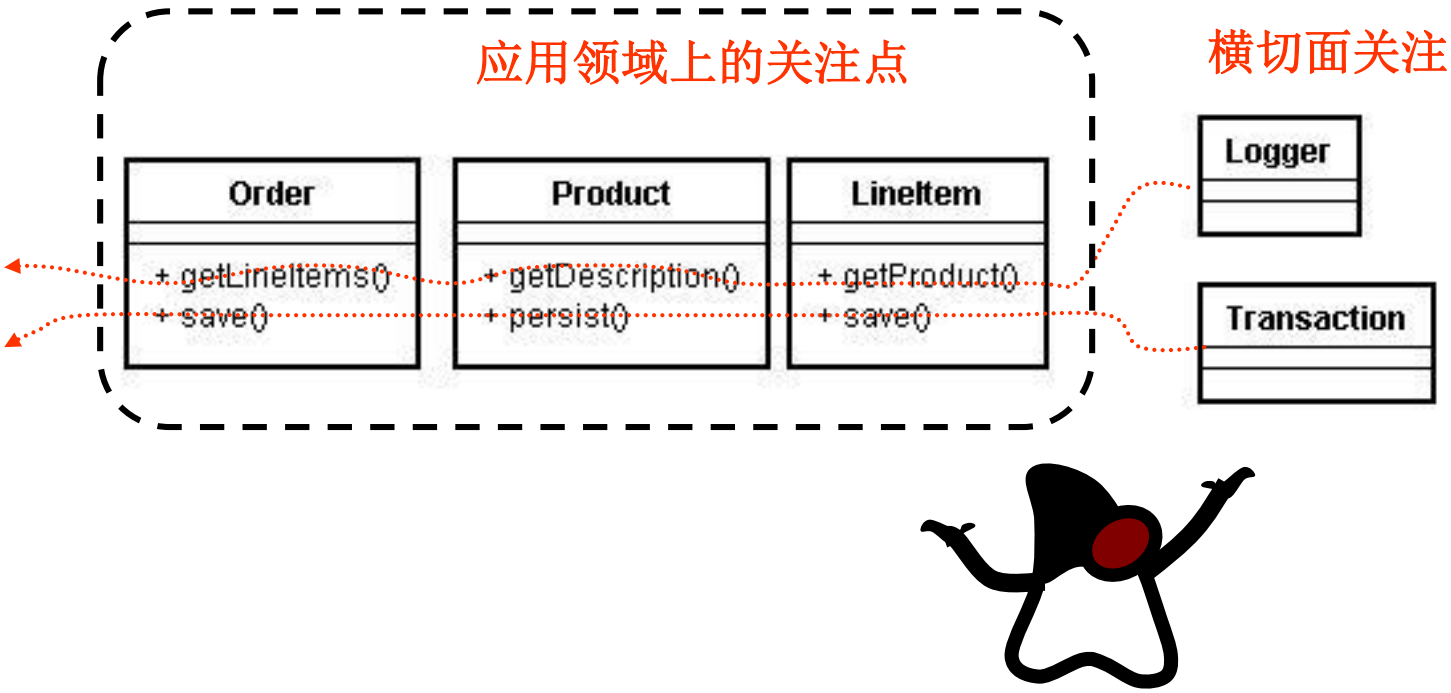
功能逻辑和横切面的逻辑混杂问题

- 系统开发过程中往往必须考虑许多与主要流程无关的横切面关注点（Cross-Cutting Concerns）
  - 事务（Transaction）
  - 安全性
  - 异常处理
  - 日志（Logging）
- 横切面关注点容易与源代码中的主要流程交杂在一起，造成程序难以维护



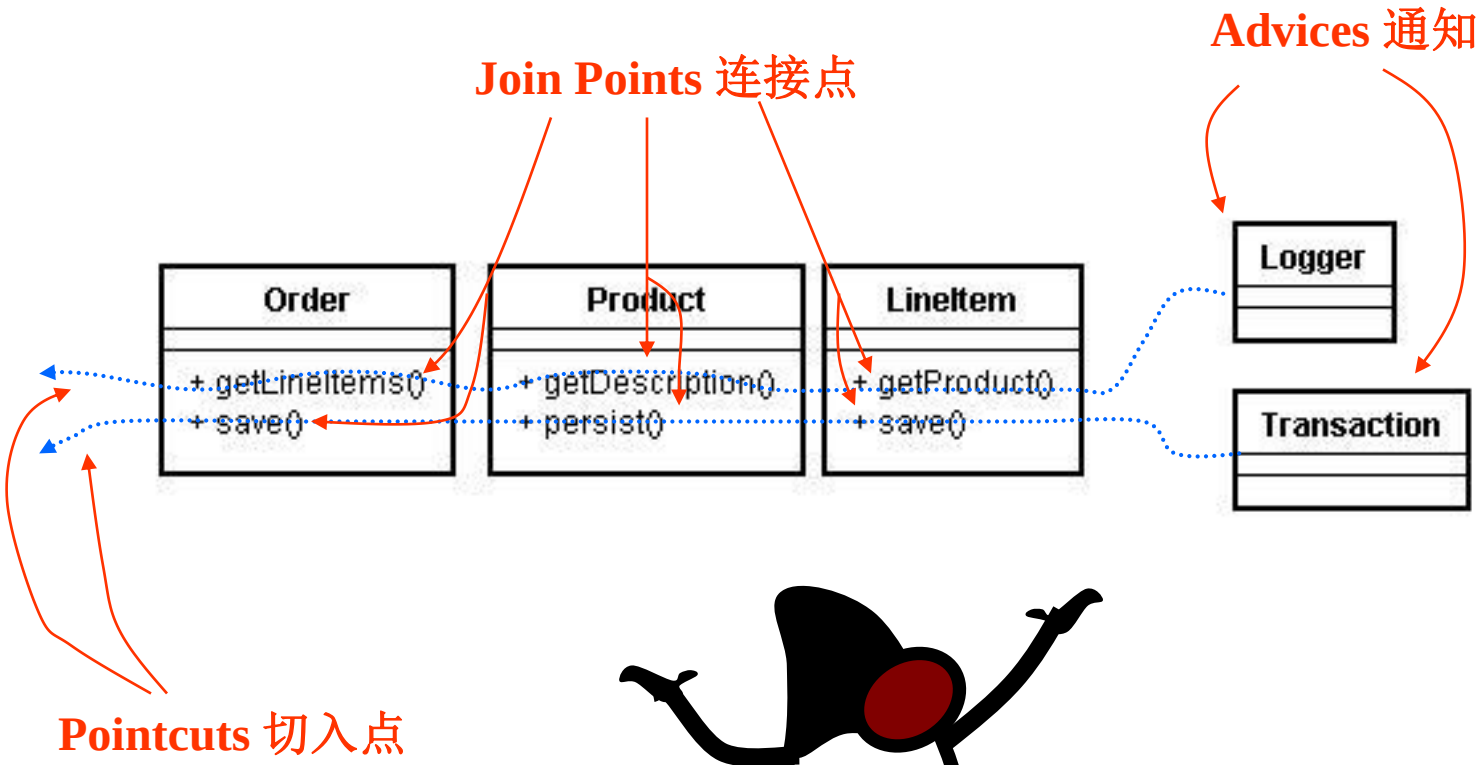
AOP：将横切面 (Aspect)独立考虑

- 基于重用的原则，可以将这些“横切面”的处理逻辑抽象出来，作为一个“Aspect”单独开发
  - Aspect可以类比为一个特殊的“类（Class）”
- 开发完成后可以再次应用在所有具有此需求的流程上



Pointcuts、Join Points与Advices

- Join Point：可被拦截到的点
- Pointcut：对哪些连接点进行拦截的定义
- Advice：拦截到的JoinPoint之后要做的事情
- Apspect（切面）：横切性关注点的抽象。包括切入点和通知的描述。



两个普通的Java类

```
package com.longthsoft.learn.spring.models;
public class A {
    public void sayHello() {
        System.out.println("Hello, I'm a");
    }
}


public class B {
    public void sayHi() {
        System.out.println("Hi, I'm b");
    }
}
```

Aspect

```
    .......
@Aspect
     public class SimpleAspect {

         @Pointcut("execution(* com.longthsoft.learn.spring.models.
*.say*())")
         public void simplePointcut() { } //声明一个切入点

         @AfterReturning(pointcut="simplePointcut()")
         public void simpleAdvice() {
             System.out.println("Merry Christmas");
        }
    }
```

配置文件

```
.....
 <aop:aspectj-autoproxy />

      <bean id="a" class="com.longthsoft.learn.spring.models.A" />
      <bean id="b" class="com.longthsoft.learn.spring.models.B" />

      <bean id="simpleAspect" class="com.longthsoft.learn.spring.Si
mpleAspect" />

</beans>
```

主程序

```
……..
public final class Boot {

      public static void main(String[] args) {
            ApplicationContext ctx = new ClassPathXmlApplicati
onContext("applicationContext.xml");
            A a = (A) ctx.getBean("a");
            a.sayHello();

            B b = (B) ctx.getBean("b");
            b.sayHi();
        }
}
```

运行结果

```
Hello, I'm a
Merry Christmas
Hi, I'm b
Merry Christmas
```

Spring中的AOP

- Spring通过AOP技术，向普通Java对象（POJO）中提供服务
  - 使用XML
  - 使用标注/注解
- Spring预定义的一些AOP服务
  - 事务管理
  - 安全
  - 日志
- 程序员可以自定义一些AOP服务
  - 审计（Auditing）
  - 缓存（Caching）
  - 定制化的安全管理

Spring主要内容

- 概述
- 控制反转与依赖注入
- 面向切面的编程AOP
- 服务抽象层

服务抽象层例

- 事务管理
  - JTA, JDBC, others
- 数据访问
  - JDBC, Hibernate, JDO, TopLink, iBatis，MyBatis
- Email
- 远程调用
  - RMI, Web Services, Hessian, HTTP Invoker……

Spring 的优点

- 低侵入式设计，代码的污染极低
- DI容器降低了业务对象替换的复杂性，提高了组件之间的解耦
- AOP支持允许将一些通用任务，如安全、事务、日志等进行集中式处理，从而提供了更好的复用
- ORM和DAO提供了与第三方持久层框架的良好整合，并简化了底层的数据库访问
- 具有高度开放性，并不强调应用完全依赖于Spring，开发者可自由选用Spring框架的全部或部分

SpringBoot

- SpringBoot是由Pivotal团队在2013年开始研发、2014年4月发布第一个版本的开源轻量级框架
- 基于Spring4.0设计，不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用的整个搭建和开发过程
  - 可轻松创建独立的Spring应用程序
  - 内嵌Tomcat、jetty等web容器，不需要部署WAR文件
  - 提供一系列的“starter”来简化Maven配置，不需要添加很多依赖
    - ➔maven是一个文件的仓储管理器
  - 开箱即用，尽可能自动配置Spring
- 它并不是什么新的框架，而是整合了很多框架，并默认配置了它们的使用方式
  - 简单、快速、方便地搭建项目；对主流开发框架的无配置集成；极大提高了开发、部署效率

轻量级框架主要内容

- 概述
- Spring
- EJB 3.0

EJB 2.X的问题

- 非POJO实现类（ Plain Ordinary Java Objects ）
  - 三个构件：Bean类，Home接口和Remote接口
  - Bean类要实现相应的生命周期方法（回调函数）
- 调用过程复杂
- 繁琐的XML配置文件
- 实体Bean有一定的局限
  - BMP由于需要用户去维护持久化工作和维护EJB状态，开发者需要做很多繁琐工作。
  - CMP不是POJO无法实现容器外测试，且EJBQL能力比较弱，对库表间关系复杂的情况支持不够。

EJB 3.0：简化EJB

- EJB3.0规范中主要涉及两个方面的改变：
  - 一套以标注/注解（Annotations）为基础的EJB编程模型
    - 取消或最小化了很多（以前这些是必须实现）回调方法的实现
    - 支持依赖注入
  - 新的实体Bean持久化模型
    - JPA
    - 降低了实体Bean及O/R映射模型的复杂性

EJB 注解/标注Annotations

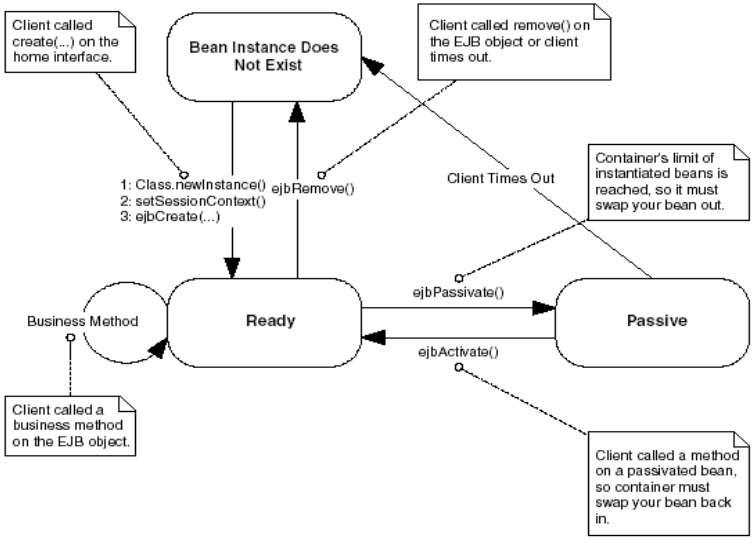
- 通过使用标注，可以在不改变原有逻辑的情况下，在源文件嵌入一些补充的信息。代码分析工具、开发工具和部署工具可以通过这些补充信息进行验证或者进行部署。
  - @Stateless, @Stateful, @MessageDriven
  - @PostConstruct, @PreDestroy,@PostActivate,@PrePassivate
  - …..
- EJB只是一个加了适当标注的简单Java对象(POJO)
  - Container interface requirements removed
  - Bean type specified by annotation or XML
- 在EJB3.0中部署描述文件不再是必须的，home接口也没有了
- 标注可以用于定义bean的业务接口、O/R映射信息、资源引用信息，效果与在EJB2.1中定义部署描述文件和接口是一样的

无状态会话Bean举例

```
import javax.ejb.*;
   /**
   * A stateless session bean requesting that a remote
   * business interface be generated for it.
   */
   @Stateless
   @Remote
   public class HelloWorldBean {
         public String sayHello() {
               return "Hello World";
         }
   }
```

有状态会话Bean

- 除了几个SFSB的特别说明之外，有状态会话bean(SFSB)和SLSB一样精简：
  - 一个SFSB应该至少有一个方法来初始化自己
    - ➔比如设置成员变量的初始值
  - Bean的提供者可以用@Remove注释来标记任何SFSB的方法，以说明这个方法被调用之后bean的实例将被移除
  - 如果需要，其他一些用于状态保持、恢复的方法
    - ➔作用等同于EJB2.X的生命周期方法（回调方法）



有状态会话Bean举例

- CounterBean.java

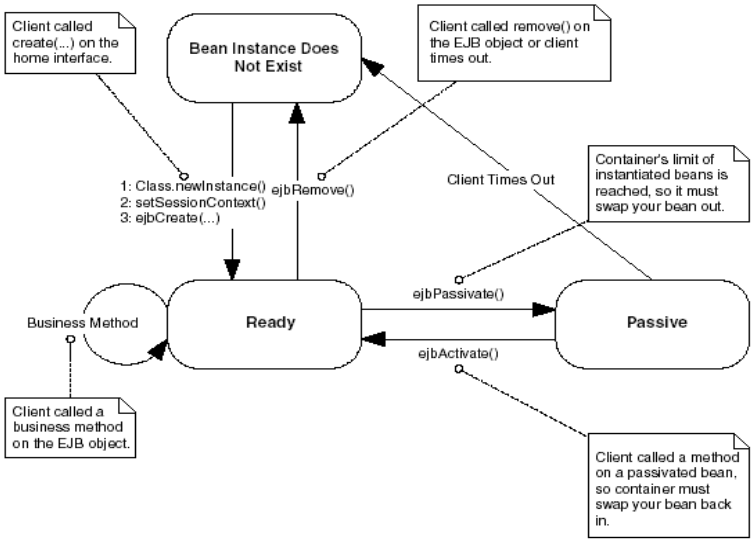
```
……
import javax.ejb.Stateful;
```

-

```
@Stateful
public class CounterBean implements Counter{
    ……
    @Remove
    public void clean(){
        System.out.println("我，被删除了!");
    }
}
```

有状态会话Bean举例2

```
@Stateful public class AccountManagementBean
    implements AccountManagement {
        Socket cs;
       @PostConstruct
       @PostActivate
        public void initRemoteConnectionToAccountSystem { ... }

      @PreDestroy
      @PrePassivate
      public void closeRemoteConnectionToAccountSystem { ... }
      ...
}
```



事务举例

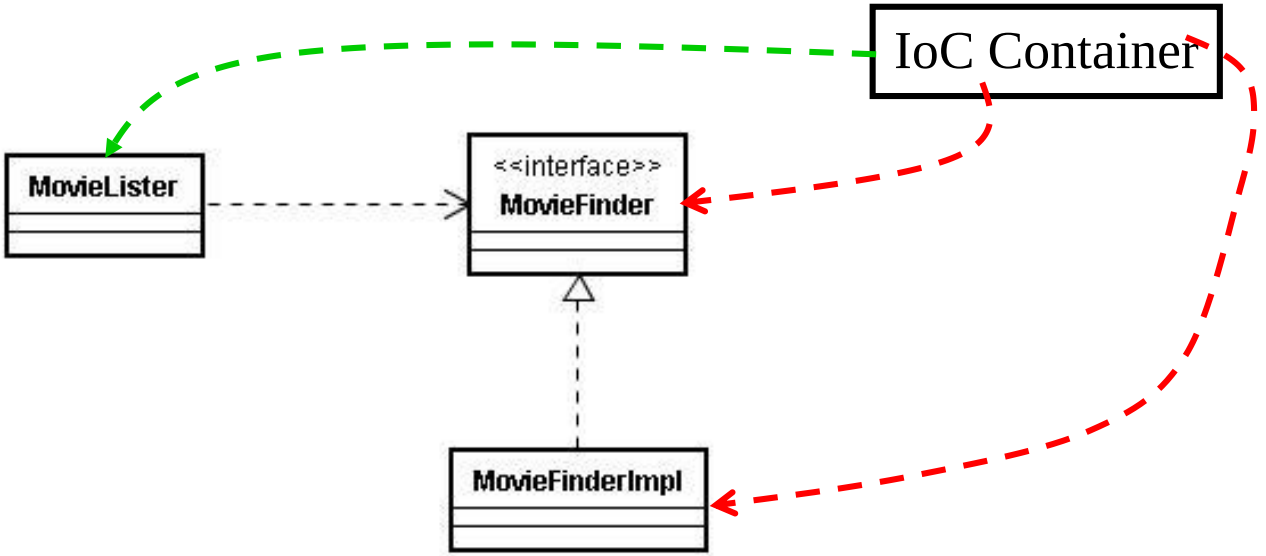
```
@Stateless public PayrollBean implements Payroll {
    @TransactionAttribute(MANDATORY)
    public void setBenefitsDeduction(int empId, double  deduction) {...}

    public double getBenefitsDeduction(int empId) {...}
    public double getSalary(int empid) {...}

    @TransactionAttribute(MANDATORY)
    public void setSalary(int empId, double salary) {...}
}
```

事务属性	客户端事务	业务方法的事务
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Mandatory	None	Error
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1
Never	None	None
	T1	Error

安全举例

```
// EJB 3.0: Security View

@RolesAllowed(HR_Manager)
@Stateless public class PayrollBean implements Payroll {

    public void setSalary(int empId, double salary) {
        ...
    }

    @RolesAllowed({HR_Manager, HR_Admin})
    public int getSalary(int empId){
        ...
    }
}
```

EJB的DI



对于EJB3.0来说，依赖注入就是由容器负责查找被依赖的对象，并注入到依赖bean中，而bean本身不再需要进行JNDI或者context查询。此外，依赖注入发生在任何业务方法被调用之前，而且支持setter方法注入和域注入两种方式

依赖注入举例

```
@Stateless
public class ServiceBean implements Service {
    private javax.sql.DataSource myDS;

    @Resource(mappedName="LocalDataSource")
    public void setMyDS(javax.sql.DataSource ds)
    {this.myDS = ds; }

    @EJB(beanName="AccountBean")
    private Account account;
}
```

EJB3.0中客户端调用Bean

```
public static void main(String[] args){
    InitialContext ctx;
   try{
        ctx = new InitialContext(); //JBOSS中得到上下文
        Counter counter = (Counter)
                        ctx.lookup(Counter.class.getName());
        counter.add(10);
        ……
    }
    ……….
}
```

EJB 3.0 Programming model

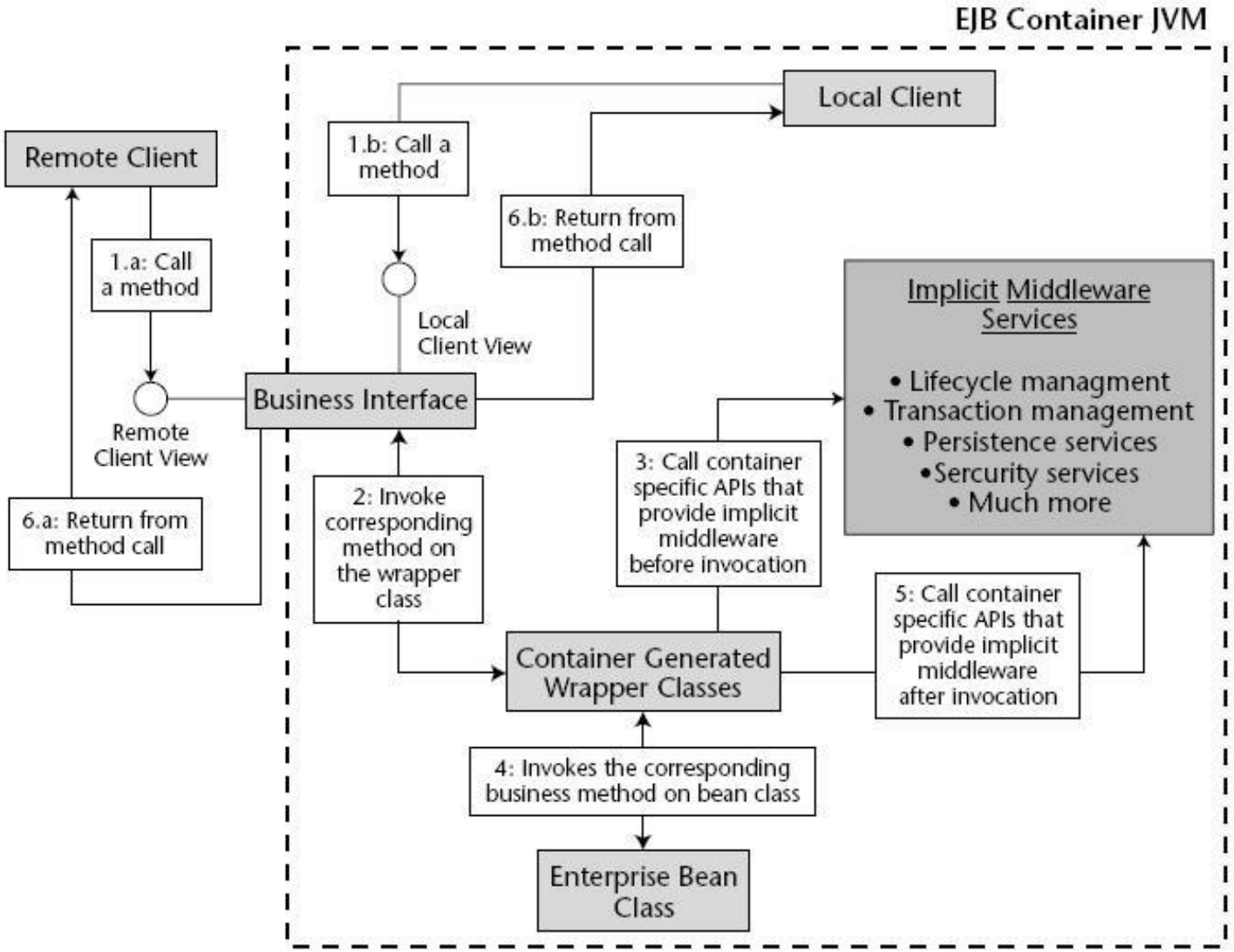


Figure 3.4 EJB 3.0 programming model.

轻量级框架小结

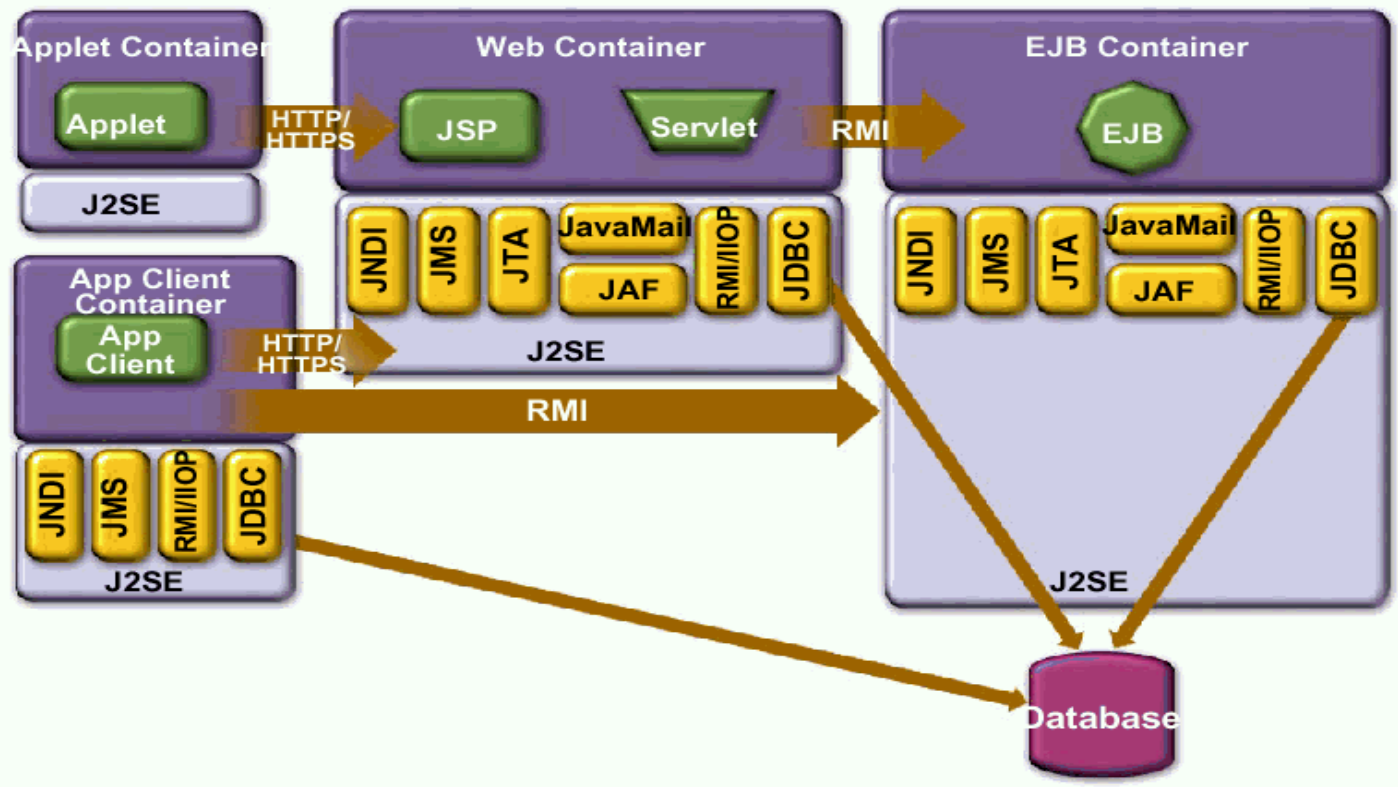
- 基本技术：依赖注入和控制反转、AOP
  - 标注
  - 配置文件（部署描述符）
- 松耦合基于构件的应用与容器的关系，在需要的地方整合需要的服务
  - 安全、事务
  - 用户自定义的支撑功能
- 松耦合基于构件的应用中构件间的关系
  - 如Spring中通过setter或构造函数注入

面向构件的分布计算环境小结

- 基于构件的软件体系结构：框架+构件+对象总线
  - 在分布计算领域，框架提供通信支持，从而涵盖对象总线功能。
- 构件模型：定义构件和容器的基本结构和接口，制定构件与构件之间、构件与容器之间交互的机制
  - 构件模型=构件+容器
  - 在分布计算领域，容器= 框架+对象总线
- 框架一词在不同的上下文中含义有一定的区别，如
  - 基于构件的软件体系结构中的框架
  - 轻量级框架中的框架：构件模型
  - 已经实现好的框架代码包

面向构件的分布计算环境小结

- Java EE /J2EE：提供了一套基于构件的方法来设计、开发、装配及部署N层结构的、面向Web的，以服务器为中心的企业级应用



面向构件的分布计算环境小结

- EJB：用于开发和部署多层结构的、分布式的、面向对象的 Java 应用系统的跨平台的构件体系结构。
  - 构件模型：Beans + 容器
  - EJB 2.0 的开发方式
  - EJB的种类：会话、实体和消息驱动 Bean
    - 生命周期方法
    - Bean的状态迁移
  - 容器：事务服务的使用方式
    - 容器管理的
    - Bean管理的

面向构件的分布计算环境小结

- 轻量级框架：以依赖注入（Dependency Injection）为代表的解耦合模式，可以让构件不去依赖容器（运行环境）的API。组件以POJO（Plain Old Java Object）的形式存在
- Spring：轻量级框架：
  - 支持“控制反转/依赖注入”的容器
  - AOP框架
  - 服务抽象层
- EJB 3.0
  - EJB的标注
  - 生命周期方法
  - 容器服务的使用方法

面向构件的分布计算环境小结

- 技术的重点不在于如何进行远程调用
  - RPC或者RMI技术足以支持，目前也有很多支持远程调用的框架
- 技术重点：如何简化应用的开发
  - 由容器提供相应的各种服务：事务、安全、持久化等
- 进一步地：轻量级框架技术把框架服务传递给松散耦合的简单旧式Java对象（POJO）。这些框架服务组件通过在运行时截取执行内容或向POJO注入服务对象，把服务与POJO捆绑在一起。POJO本身不关心捆绑的过程，并且对框架组件几乎没有依赖。
  - 松耦合基于构件的应用与容器的关系，只整合需要的服务
  - 松耦合构件之间的关系

关于框架：多说两句

- 框架：功能越强越好？
  - 事实证明：NO！
- 现在的一个趋势是框架只做最关键的东西，不需要封装太多功能
  - 很多互联网公司的产品小而精，要求开发效率高、性能好，而对事务等功能的要求不高
  - 而且大多程序员更愿意把学习的精力放在底层如servlet、sql上，而不是struts2、 Hibernate等框架上
- 因此，相较SSH，近年来SSM更受欢迎
  - springMVC、Spring、 MyBatis
  - SpringBoot