


JAVA持久性

Allen Long

allen@huihoo.com

http://www.huihoo.com

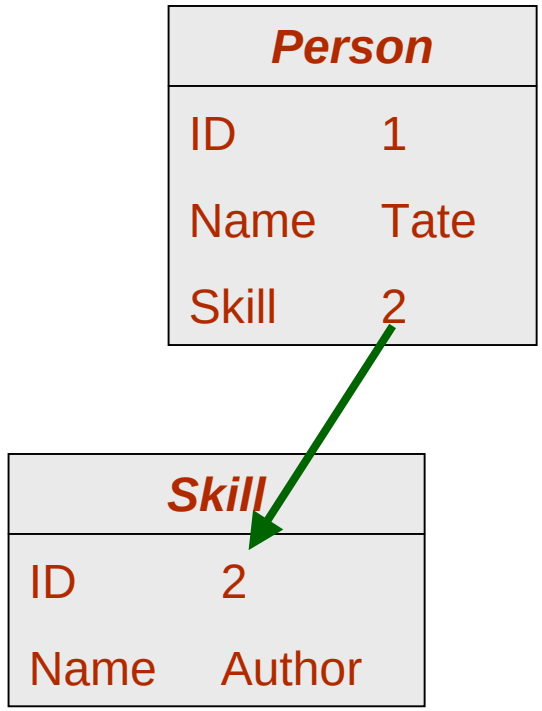
2004-04

内容安排

- 解决方案一: BMP/CMP
- 解决方案二: JDO

对象模型

- 对象: Behavior plus data
- 对象拥有字段
- 对象是唯一的
- 关联是显式的
  - 参考
- Advanced notions
  - 继承Inheritance
  - 抽象类Abstract classes



解决方案一: BMP/CMP

- 组件模型
- Default choice for EJB
- Troubled history
  - But has seen some improvement

历史: EJB 1.x

- EJB 1.0 was introduced in October 1998
  - EJB 1.1 followed in December, 1999
- 意图: Nirvana
  - Fully distributed persistent entities
  - Transparent distribution as needed
- 现实
  - Not ready
    - Deployment, security, persistence stories incomplete
    - Portability problems
  - Massive performance problems
    - Lacked local interfaces
    - Lacked relationship management
  - Most applications developed with BMP

当前: EJB 2.x

- EJB 2.0 was introduced in October 2001
- Fixed some of the major problems
  - Local interfaces allow better performance
  - Container-managed relationships reduce application joins
- Most applications still use a façade
- Many problems still exist
  - Container overhead. Both façade and entity support
    - Transactions, security, potentially distribution
  - Model is limited
    - No component inheritance, abstract classes
    - Artificial restrictions forced by EJB spec

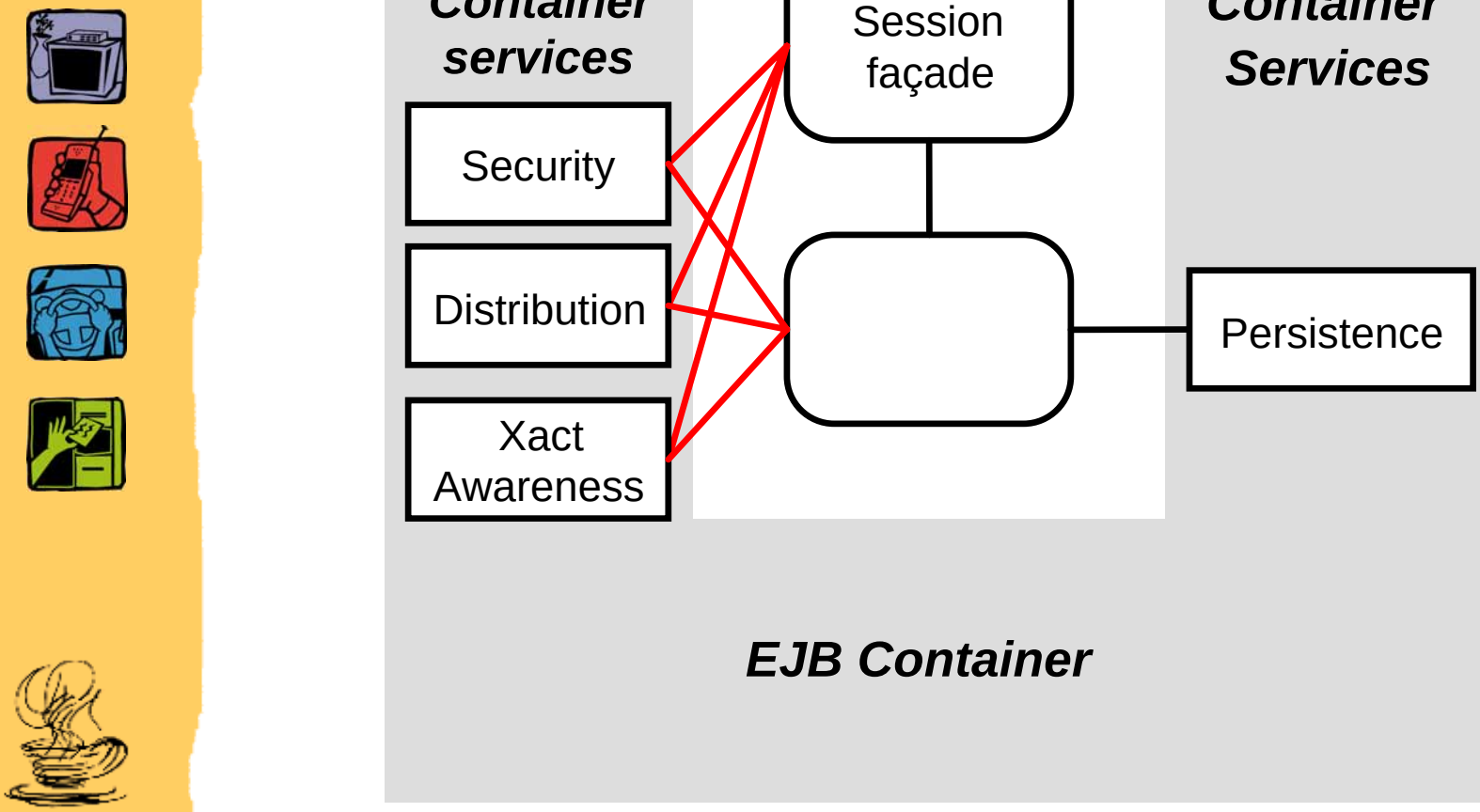
EJB 2.x 关心(Concerns)

- EJB 2.0 changes torpedo philosophical advantages
- Local interfaces
  - Require deployment of all beans with local interfaces
  - Now persistent components have different interfaces
- CMR: It's no longer a component model.
  - *EJB QL: Deploy time binding!*
  - Meta-component comprised of other components? No
    - Can't rely on the EJB component deployment

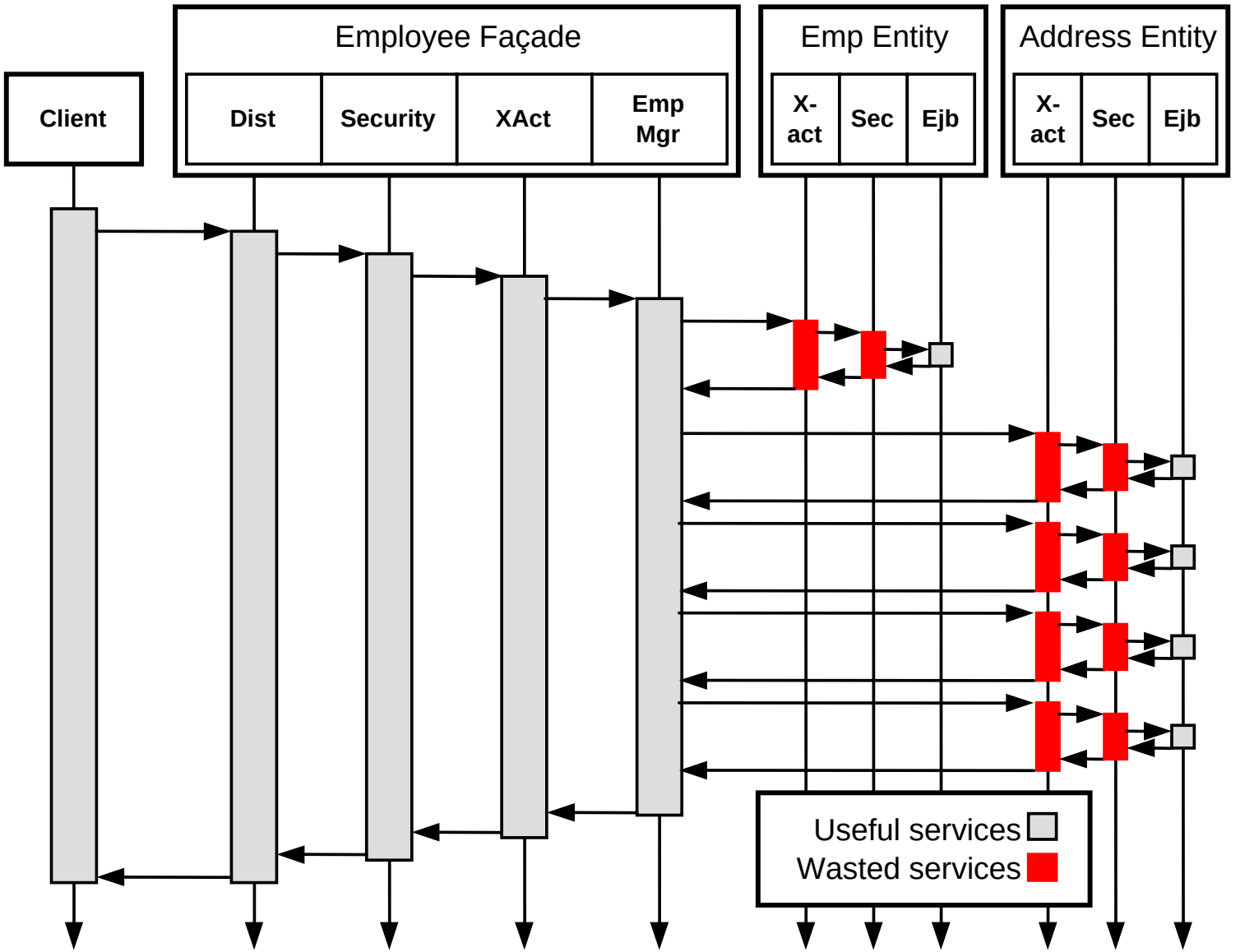
基本问题

- 模型灵活
  - EJB definition of reentrant class is not on method level
  - *No inheritance (component inheritance)*
- 透明
  - Entities are too different from Java objects
- 绑定灵活
  - *Queries are bound at deploy time*
- Coarse-grained architecture for fine-grained problem
  - Performance
  - Clarity of model

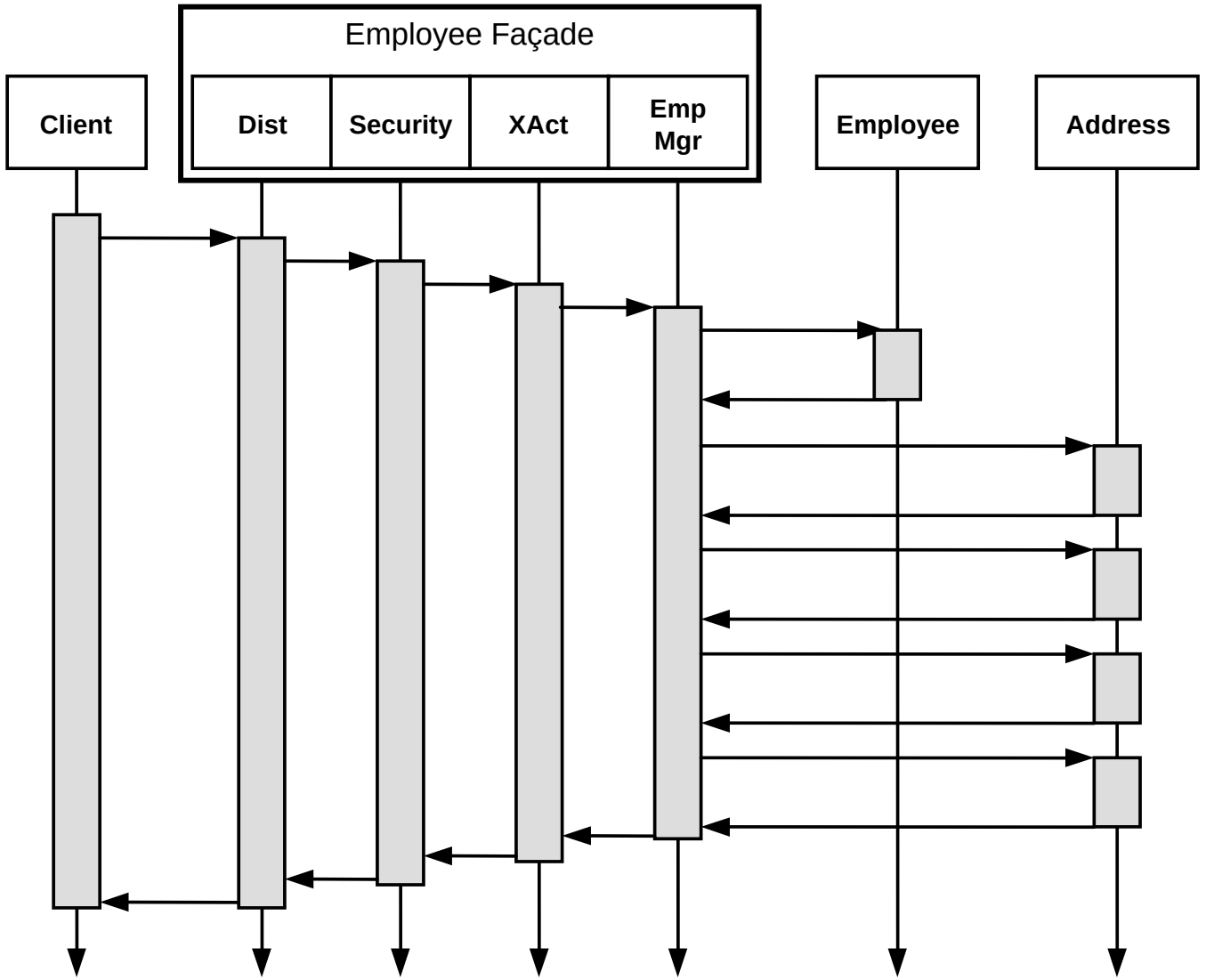
复制服务


Container services
Security
Distribution
Xact Awareness
Session façade
Container Services
Persistence
EJB Container

无用services


Client
Employee Façade
Dist
Security
XAct
Emp Mgr
Emp Entity
X-act
Sec
Ejb
Address Entity
X-act
Sec
Ejb
Useful services
Wasted services

正确的方式


Employee Façade
Client
Dist
Security
XAct
Emp Mgr
Employee
Address

BMP/CMP好处

- Political support
- Standard support
- Solutions for some problems
- Ongoing investment
- Sharing the model

The bottom line




- Model is much too limited
  - Not transparent
  - No inheritance, abstract classes
  - Reentrance, threading limitations
- Service is poorly packaged
  - Coarse-grained service
- So many look elsewhere

解决方案二: JDO




目标: 定义对象级的持久化

--完全支持对象模型，包括引用、集合、接口、继承

--完全透明持久化：这使得业务对象完全独立于任何数据库技术，这使用了一个字节码增强机制（a byte-code enhancement mechanism）。

--缩短开发周期（不再需要映射）

--在开发小组中清晰地划分业务人员和数据库人员。

--通用持久性

JDBC限于RDBMS，JDO潜在地可以处理任何类型的数据源，包括RDBMS，ODBMS，TP监控处理，ASCII无格式文件，xml文件，properties文件，大机上的Cobol数据库，等

JDO是面向大型信息系统的一个完全的解决方案，在这样的系统中，信息存储于多种异质数据源

--涵盖J2EE、J2SE、J2ME的广泛实现

--强壮的事务模型

--同时支持C/S和多层体系结构

JDO 包

PersistentCapable：拥有持久实例的类必须实现这个接口。管理对象生命周期。

PersistenceManager ：代表了到数据源的连接。一个应用可以打开一个或多个PersistenceManagers。

PersistenceManagerFactory：允许从数据源中得到一个PersistenceManager的实例，这个工厂也可以作为一个连接池。

Transaction：允许界定事务

Query ：允许明确地声明性地使用JDO查询语言从数据源中获取数据。NB：也可以通过引用之间的基本的定位，隐含地、透明地从数据源中获取对象。

InstanceCallback：在数据库操作中（比如before/after read, before/after write，等），定义一些钩子，以做特殊处理（像暂时属性的初始化）。

JDOException：JDO操作中抛出的例外。

JDO也定义了帮助类，对象标识（由应用或数据源管理）

JDO实现可以支持或不支持兼容的PersistenceManager（当PersistenceManager是兼容的时，你可以得到存储于不同数据库的对象引用）。

NB：在JDO的第一个发布版本中，并没有严格地定义锁和锁策略。

JDO对象模型

--JDO对象模型基本上是Java的对象模型,包括所有的基本类型,引用,集合和事件接口。

--除了系统定义类(system-defined classes),支持所有的字段类型(包括简单型、可变和不变的对象类型(immutable and mutable object types)、用户定义类、数组、集合、接口)。

--支持所有的成员变量修饰符(private, public, protected, static, transient, abstract, final,

synchronized, volatile)

--除了对象状态依赖于不可访问的或远程对象,即继承于java.net.SocketImpl、本地方法等,所有的用户定义类都可以是PersistentCapable。

JDO对象生命周期




为了能够在数据源中访问、存储对象，应用必须首先得到一个或几个数据源的连接。一个JDO PersistenceManager对象就代表了这样一个连接。它可以通过PersistenceManagerFactory类得到。持久化对象必须是实现了PersistentCapable接口的类的实例。这样的类可能同时拥有持久化的或临时的（transient）实例。

为了使一个实例持久化，编程者必须调用PersistentManager的makePersistent方法。通知JDO对象为持久化或临时的很重要，即使它们可以从JDO的行为中得到它是临时的，比如事务管理和对象标识。对象标识可以由应用管理，或者由数据源代理（这大多是在使用ODBMS实例时，因为概念ObjectID本身就是ODMG模型的一部分）。

JDO支持一种持久化模型，这这种模型中持久性自动传播到引用的对象。这种机制经常称为“延伸持久（persistence by reachability）”或者“传递持久（transitive persistence）”。这意味着一旦一个已经持久化的对象引用了一个临时对象，这个临时对象自动变成持久化的。对于JDBC编程者，这个模型可能很奇怪，但是他们会发现这是大多数情况下编程者希望从持久化框架中得到的支持。

JDO code: Employee.java


```
public class Employee
    extends Person
{
    private float salary;
    private Company company;
    private Set projects = new HashSet ();

    public Employee (String firstName, String lastName)
    {
          super (firstName, lastName);
    }

    public void giveRaise (float percent)
    {
          salary *= 1 + percent;
    }

    public Collection getProjects ()
    {
          return projects;
    }
```

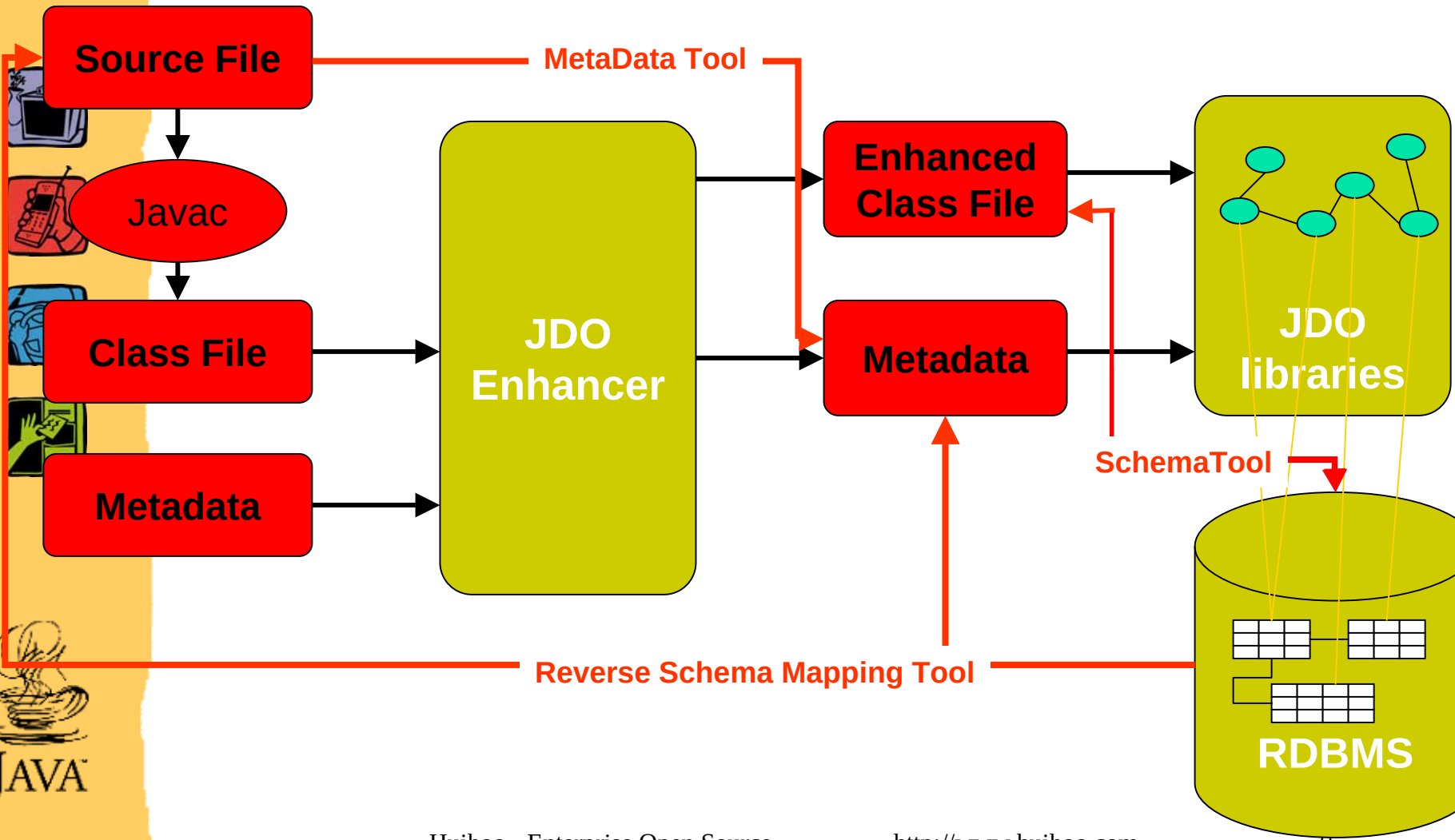
Employee.jdo


```
<?xml version="1.0" encoding="UTF-8"?>
<jdo>
  <package name="com.solarmetric.example">
    <class name="Employee" persistence-capable-
    superclass="Person">
      <extension vendor-name="kodo" key="table" value="EMPLOY_T"
    />
      <field name="projects">
        <collection element-type="Project" />
      </field>
    </class>
  </package>
</jdo>
```

JDO增强

为了达到上面提到的完全透明的持久化，JDO定义了一个称为“增强（Enhancement）”的字节码工具机制（byte-code instrumentation mechanism）。它的思想是从业务类中剔除所有的显式的数据库依赖代码。和已存在的或新的数据源的映射通过外部的元数据（metadata）XML文件定义。JDO增强器读取编译的java文件（.class文件），并且应用元数据文件中定义的持久化规则。

JDO增强


Source File
MetaData Tool
Javac
Class File
Metadata
JDO Enhancer
Enhanced Class File
Metadata
JDO libraries
SchemaTool
RDBMS
Reverse Schema Mapping Tool

JDO 公共接口

- PersistenceManagerFactory: Factory for obtaining configured PersistenceManagers
- PersistenceManager: Persistence controller and factory for Transaction, Query
- Transaction: Replaced by sessions with EJB
- Query, JDOHelper, PersistenceCapable, InstanceCallbacks: Other types of support

JDO 查询语言

- 基本查询:

```
String filter = "salary > 30000";
Query q = pm.newQuery (Employee.class, filter);
Collection emps = (Collection) q.execute ();
```

- 带Ordering的基本查询:

```
String filter = "salary > 30000";
Query q = pm.newQuery (Employee.class, filter);
q.setOrdering ("salary ascending");
Collection emps = (Collection) q.execute ();
```

JDO好处

- 不需要写持久性基础设施
  - No hand-coded SQL
- Standard means of persisting objects
- Portability between data stores
- 轻量级(Light weight)

JDO好处



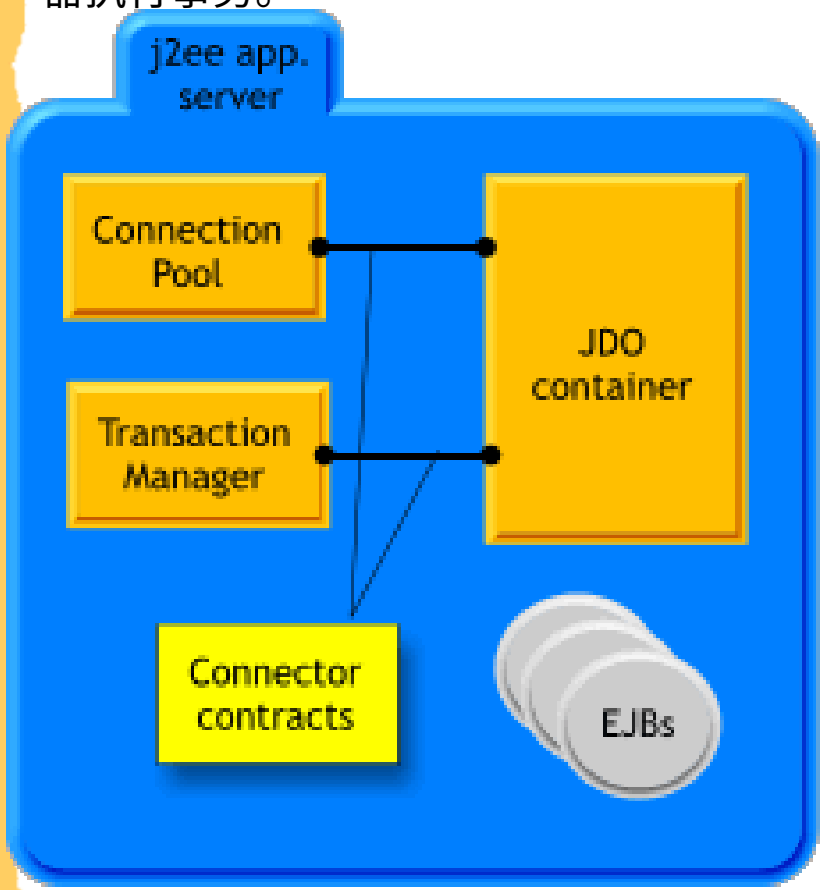

- Does not limit the object model
- 全面支持面向对象概念
  - 抽象类(Abstract classes)
  - 继承(Inheritance)
  - Loops in calling graph
- Reports of a 20 - 40% decrease in persistence coding time

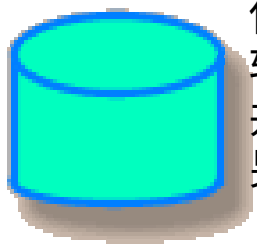
扩大的附加值

- Example: SolarMetric's Kodo
- Caching features provide > 10x boost
  - Compared to standard JDO
  - Distributed cache allows high scalability
- Supports most JDO specification optional features
- Supports many databases, application servers

JDO和J2EE

JDO已经设计为整合入J2EE体系。JDO依靠一个新的Connector规范来管理一个J2EE应用和JDO容器之间的交互。在JDO规范中这个描述为管理环境（managed environments，意思是事务和连接由应用服务器自己管理）。JDO容器和J2EE应用服务器交互，得到数据源的连接（因为应用有它自己的连接池），并根据应用服务器的JTA兼容的事务管理器执行事务。

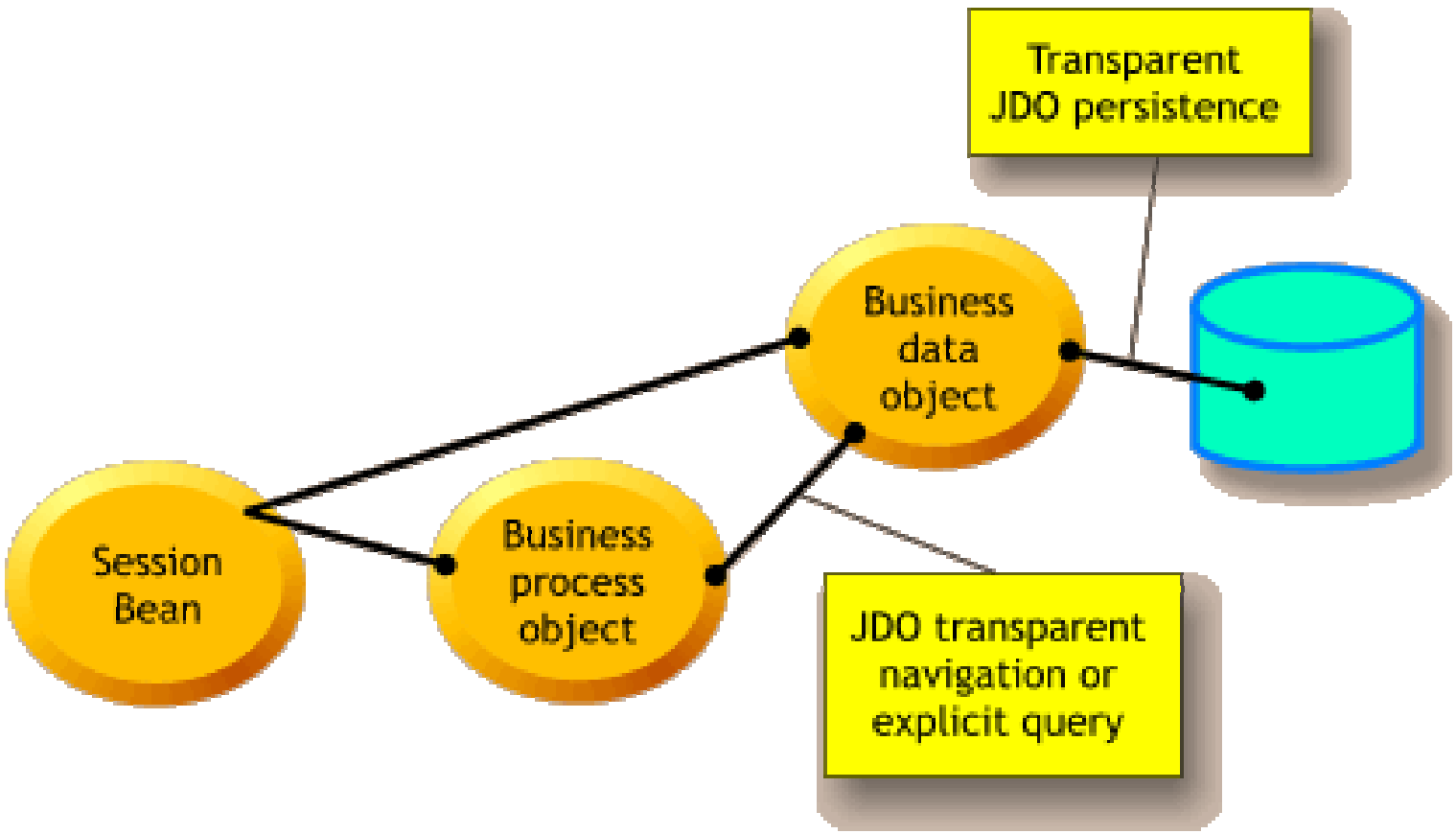




JDO明确的解决了J2EE体系的一个主要缺陷，就是EJB部件模型将持久性和分布性结合起来。这两个概念是不相关的，但是EJB规范试图简化，迫使应用对持久性和分布性使用相同的粒度级别。这在软件工程的视角并不清晰，并且也不具可扩展性。
另外一点，EJB持久模型（CMP和BMP）过分简化了，只能应付有限的情况，不适合现实的应用需要。

JDO和J2EE

使用JDO，你可以使用一种非常优雅、通用的设计：Session Bean（访问业务处理对象）自己访问业务数据对象。这样做，应用仍然拥有J2EE体系所有的优点（分布式、事务、连接，...），并且持久化由JDO透明、高效地管理。



JDO和J2EE




JDO可以支持在异种数据源中非常复杂的映射机制，然而，EJB/CMP模型仅适用于简单的JDBC模型。

使用JDO你没有业务对象的复杂性的限制（然而EJB/CMP不支持继承）。

使用JDO在你的业务数据对象中完全没有数据库代码（然而使用EJB/BMP 你的业务代码会掺杂入JDBC代码）

业务处理对象提供了处理多个业务数据对象的方法。

业务处理对象通常是非持久化的，它们通常通过混合的JDO查询和导航得到业务数据对象。

从SessionBean中剥离业务处理方法仍然很重要，因为这样你的业务模型可以把众多应用中任何的基础构架应用到J2EE应用中。

JDO开发步骤

1.定义类

2.构建包含有关类持久性信息的元数据文件

3.运行使用元数据生成已增强的类字节码的类增强器

4.运行使用元数据和已增强的类字节码在数据库中构建表定义的模式定义工具

5.构建利用JDO接口使类具有持久性的应用程序

JDO小结




- JDO具有所有必须的数据存储功能：增、删、改、事务、数据唯一性、缓冲
- JDO APIs (PersistenceManager, Query, Transaction)
- JDO提供了一个称为JDOQL的查询语言
- JDO可以支持在异种数据源中非常复杂的映射机制，然而，EJB/CMP模型仅适用于简单的JDBC模型。
- 使用JDO你没有业务对象的复杂性的限制（然而EJB/CMP不支持继承）。
- 使用JDO在你的业务数据对象中完全没有数据库代码（然而使用EJB/BMP 你的业务代码会掺杂入JDBC代码）
- JDBC只是面向关系数据库（RDBMS），而JDO更通用，提供到任何数据底层的存储功能，比如关系数据库、文件、XML以及对象数据库（ODBMS）等等，使得应用可移植性更强。

内容回顾

- 解决方案一: BMP
- 解决方案二: CMP
- 解决方案三: JDO

参考资料

- http://java.sun.com/products/jdo/

sun公司的jdo站点

- http://access1.sun.com/jdo/
- http://www.huihoo.com
  国内一个关于中间件的专业站点

结束

谢谢大家！

Allen@huihoo.com
http://www.huihoo.com