 1 Java与数据库应用，JDBC Java发明以来，在短短的几年之间，迅速占领了从桌面应用（J2SE）到服务器（J2EE），再到小型设备嵌入式系统（J2ME）的应用开发市场，其语言吸取了SmallTalk的一切皆对象的理念，摆脱了C 的历史累赘，简洁、自由的风格赢得了很多开发者的喜爱。从JDK1.1开始，Java成为实用的语言，而不是被人观望的新品秀；再经过JDK1.2的大量增强（尤其是Collection Framework），JDK1.3的虚拟机效率提升（HotSpot），JDK1.4的融合百家之长（Logging、RegExp、NewIO等），现在已经是成熟稳重，颇显大家风范。 在企业级市场上，大部分的应用建立在数据库基础上，数据是企业的生命，传统开发语言，包括面向过程的C、面向对象的C、变种Pascal的Delphi（非常棒的语言，我用过四年），面向数据的PowerBuilder等等，先后在数据库开发的舞台上展现风姿。Java当然不会放过这些，于是，出现了JDBC。在JDBC的帮助下，Java也迅速渗入数据库开发的市场，尤其是面向企业服务器的应用开发。 今天要谈的JDO，与JDBC有非常密切的关系，尽管JDO并不是只面向JDBC的数据对象包装规范。下面先简单地介绍一下JDBC。 1.1 关系数据库之百家争鸣，ODBC 关系数据库的历史一言难尽，我只能从我的接触经历和所见所闻，简单地叙述一下。最早的时候，计算机还只在一些大型的研究机关露面，并不是普罗大众可以涉及的。苹果电脑将个人电脑引入民间，再随着IBM的PC标准开放，

个人电脑逐步普及开来，加上微软的DOS操作系统，以及Borland的 Turbo系列语言开发环境，老百姓发现原来电脑可以做这么多事！后来，出现了DBASE，一个简单的关系数据库系统，和SQL语言。后来，Borland看到了数据库的市场前景，推出了Paradox（也是当今Delphi和C Builder中仍然使用的Paradox），一举占领了民用数据库的大部分江山，之后，Borland干脆收购了Dbase，后来又购买了InterBase，将数据库市场的领先优势一直保持到Windows3.0出现。这时候，微软在Windows1.0 和2.0被人痛骂之后顽强地推出3.0，以及更稳定的3.1和Win32API，造就了个人电脑桌面操作系统的霸主地位，在Borland未警觉的情况下，购买了同样具有类Dbase数据库技术的Fox公司，并迅速将其易用化，形成了FoxBase，后来演变成FoxPro，逐渐超过了Borland，成为个人电脑数据库的大户。微软再接再励，为简单易用而低负荷要求的数据库应用开发了Access，赢得了广大开发人员的心。当然，同期的Oracle、 Sybase、Informix等商用数据库凭专注于企业级数据库技术成为高端的几位领军人物。微软当然也想成为高端数据库供应商之一，于是自行开发一套面向企业级应用的数据库，不过很快项目夭折，微软不甘心，购买了Sybase的底层TDS技术，包装成了SQL Server，凭微软的高度易用性的特点，也占领了不少市场。 当市场上出现众多的数据库产品之后，Borland和微软都发现自己拥有的数据库产品挺多，市场也不小，不同的产品给用户带来不同的配置任务，不利于所有产品的推广，于是，两者纷纷开始制定数据库访问的规范，微软推出了ODBC，其面向开发人员的亲和性，逐步获得了认可，同时，Borland纠集了IBM和Novell也推出了IDAPI数

据库接口规范，也就是今天BDE的核心，不过后来Novell和IBM先后退出，只剩Borland独力支撑。不过Borland是一个技术实力雄厚的公司，其技术一向领先于微软，BDE的性能比初期的ODBC不知道要好多少倍，后来微软偷师学艺，把连接池等技术加到ODBC中，在Delphi3.0及其BDE在市场上风光无限的时候，逐步赶了上来并有超过。直到今天，BDE仍是Borland的产品线上的数据库访问标准，而微软如果不是将ODBC和多数数据库的客户端内嵌进Windows的话，估计BDE仍是市场的赢家。不过，微软是玩弄市场的老手，通过对操作系统的垄断，其数据库产品和ODBC标准终究占据了多数开发市场。 1.2 从optional pack到JDK的标准API Java开始涉及数据库应用后，Sun就极力制定Java的数据库规范，JDBC API就是类似ODBC一样，对数据库访问的底层协议进行最基本的包装，然后形成一套统一的数据访问接口，数据库连接、SQL语句句柄、结果集，都带有ODBC的影子。以方便配置为目的，Sun极力推荐完全瘦客户端的TYPE 4型JDBC驱动，这是一个不需要安装数据库客户端的驱动规范，是现在使用最多的。当然，为了保持与旧的数据库兼容，JDBC规范中包括了专用于连接 ODBC的TYPE 1驱动和需要安装数据库客户端的TYPE 2驱动，以及可以由厂商在数据库服务端专门提供面向JDBC的服务的TYPE 3驱动。 JDBC最早出现时，还不属于标准JDK的一部分，而是作为一个额外包提供下载。后来，随着Java编写的数据库应用的的增多，和JDBC规范本身的逐渐成熟，JDBC终于成为JDK1.1的一部分。 JDBC目前最新的是3.0版本，还有正在讨论中的4.0版本。实际上，在开发中使用得最多的还是1.0中的API，2.0中主要

增加了可双向滚动的结果集、更新批处理等提高可用性和性能的API，3.0主要增加了连接池、可更新的结果集等特性。4.0将在可管理性、连接池规范化等方面再做改进。 2 面向对象与数据库 现在的程序员，没有不知道面向对象的。作为接近真实客观世界的开发概念，面向对象使程序代码更易读、设计更合理。在普遍存在的数据库应用领域，开发人员对面向对象的追求从未停止过。从八十年代开始，就有很多公司和研究机构在进行着面向对象与数据库结合的研究。 2.1 SmallTalk、C与C 、Delphi-Object Pascal、Java 面向对象的语言最早有好几种雏形，IBM的SmallTalk是其中最为流行的，在SmallTalk中，一切都是对象，一切都是类，它将面向对象的概念发挥到了极致。面向对象的编程比起传统的面向过程的方式挺进了一大步，使人们认识到：原来软件可以这样写。不过，由于计算机基本结构与底层硬件体系和系统软件的限制，SmallTalk还不能在理想的性能前提下推广到普通的应用上，这一点暂时限制了SmallTalk的发展，接着，C语言的面向对象版C 出现了，由于使用C语言的人很多，C 很快成为面向对象编程的主流语言。不过，为了保证与C的兼容，C 保留了很多面向过程的痕迹，比如恶心的指针、全局变量等等。Pascal的改进版Object Pascal相对来说安全许多，后来Borland干脆将Object Pascal换了个名字，叫Delphi，从此开创了一片面向对象编程的新世界， Delphi的严谨语法和快速编译吸引了众多的应用开发者，加上Borland的完美的VCL组件体系，比起MFC来方便而容易，另外，Delphi完整的数据库组件，也将数据库开发变得简单而容易，Delphi再次成为成熟的面向对象开发语言。微软当然不会放过这些，通过

将MFC内置到操作系统中，微软的VC 也抢回一些市场。这也是为什么Delphi开发的应用程序编译后会比VC、VB开发的程序大的原因。 1995年，Sun的一个开发小组本来为了小型嵌入式系统开发OAK语言，结果无心插柳柳成荫，发展出了Java语言，它是一个完全摆脱了传统语言的各种负担的面向对象的语言，当然，也保留了一些非面向对象的核心（原始类型）以保证速度。现在Java也为最流行的面向对象语言之一。当然，微软同样不会放过它，擅于模仿的微软立即弄出一个C#来与之竞争，并在C#中保留了一些变种的指针（指代）以吸引传统的C开发者。关于这些语言的各自特点，这里就不一一赘述了。 2.2 数据库与数据对象化 数据库是企业级应用不可缺少的，因此，在面向对象流行的时候，数据库厂商也在进行着数据对象化的研究。这些研究在上个世纪八十年代就初现端倪。 数据库的对象化一般有两个方向：一个是在主流的关系数据库的基础上加入对象化特征，使之提供面向对象的服务，但访问语言还是基于SQL；另一个方向就是彻底抛弃关系数据库，用全新的面向对象的概念来设计数据库，这就是对象数据库ODBMS。 2.2.1 关系数据库对象化、SQL99与JDBC3.0 随着许多关系数据库厂商开始提供对象化服务，各自的接口开始互不兼容，在经历一些麻烦之后，关系数据库厂商感觉到规范化的必要，因为当初关系数据库雄霸天下时SQL92标准起了很大作用，大家可以按照统一的编程方式来访问高性能的商用数据库。 关系数据库厂商集中起来，重新将对象化服务规范起来，形成了SQL99规范，将其中的对象结构等内容规范起来，开始一个崭新的面向对象的关系数据库（ORDBMS）的历程。 JDBC3.0就是在这种情况

下出台的，它将对关系数据库中的对象服务的访问API规范起来，为Java平台提供了访问ORDBMS的标准方式。当然，JDBC3.0对传统的SQL操作也进行了很多功能增强。 Oracle是一个传统的关系数据库厂商，在对象化的道路上，Oracle当然采取追加对象化特征的道路，以侵入数据对象化的市场，保持Oracle在数据库领域的领导地位。如果说Oracle7.4使Oracle走向全盛的话，从Oracle8开始，Oracle就成为关系数据库加对象类型的先驱。在Oracle8 中，我们可以定义一些数据结构（Record），将普通的类型包装在其中成为数据元素，然后可以在客户端按Record结构进行访问，初步提供了面向对象的数据库服务。 2.2.2 对象数据库 对象数据库就是采用全新的面向对象概念来设计数据库的全新数据库类型。在这方面，主要以一些大学研究机构进行设计和开发，有些也形成了产品，不过由于市场方面的原因（主要是关系数据库的容易上手和市场绝对领导地位）和ODBMS先天的一些弱点（比如查询引擎很难优化），使ODBMS没有象关系数据库那样流行起来。 不过对象数据库的对象化特点还是令人割舍不下，目前还是有一些很好的产品在市场上，从商用的到免费的都用。目前在ODBMS领域占据领导地位的是Versant、FastObjects和ObjectStore等几大厂商，并且，市场份额也在逐步扩展。免费的产品包括C 编写的Ozone、纯 Java的db4o等等。还有一些研究机构开发一些底层的面向对象数据库引擎，但只提供一些底层的API，不提供管理方面的功能，以及一些算法提供开放式接口，让厂商去选择和实现。比如美国威斯康新大学计算机系数据库组的SHORE引擎，就是一个非常出色的面向对象数据库引擎，现在还在积极的更新中，一些

其它研究机构和数据库厂商采用它完成了自己的特别的对象数据库，比如专用于地理信息的数据库、专用于宇宙空间数据研究的数据库等等。 目前对象数据库最大的障碍是缺乏统一的规范，各个数据库厂商有各自的访问接口。对象数据库比起关系数据库来，不只是基本的几种数据类型那么简单，它还涉及继承处理、多态等一大堆面向对象特征的实现，规范化道路当然困难重重。这也是对象数据库无法普及的一个重要原因。 也有一些机构提出了一些建议的规范，比如制定Corba标准的OMG小组的一个分组ODMG提出的ODMG规范，目前已经是3.0版本，其中的 OQL对象查询语言相当具有吸引力。还有一些中立的机构提出了其它的一些标准化的对象访问API，也可算是面向对象数据库的规范之一。象前面提到的 FastObjects和Ozone就是符合ODMG3.0规范的。 3 Java对象映射 话说回来，在一般的开发人员眼中，数据库就是指关系数据库，因此，很多应用还是采用简单的JDBC来访问数据库。在开发的过程中，大家逐渐感觉到JDBC的局限性，比如调用复杂、容易产生资源泄漏等等，与面向对象的Java语言有一段距离，因此，很多开发小组开始思考如何将应用中的数据进行对象化建模，然后再想办法与JDBC结合起来，这就是Java数据库开发中的层出不穷的对象包装技术。 3.1 对象包装技术 3.1.1 传统包装与演变 传统包装顾名思义，就是最初出现的包装方式，很多公司都经历过这一步，产生了很多风格各异的包装方法。当然，笔者也有过还算丰富的尝试过程。举例来说，如果我们有一个用户类： public class User { public int userId. public String name. public java.util.Date birthday. } 我们可以将其当作一个简单的数据类，然后写一些工具方法来实

现与JDBC的交互。这些方法，我们可以放到一个另外的工具类中，也可以放到User类中作为静态方法。这些方法包括简单的增、删、改、查（以Oracle为例）： public class User { public int userId. public String name. public java.util.Date birthday. public static User addUser(String name, Date birthday) throws SQLException { Connection conn = ….//获取一个JDBC连接 PreparedStatement ps = conn.prepareStatement("…"). // 获取一个序列值来作为用户标识 ResultSet rs = ps.executeQuery(). rs.next(). User user = new User(). user.userId = rs.getInt(1). //读取序列值为新用户标识 user.name = name. user.birthday = birthday. ps = conn.prepareStatement("insert into …."). //插入用户数据记录的SQL ps.setInt(1,user.id). ps.setString(2,user.name). ps.setDate(3,user.birthday). ps.executeUpdate(). rs.close(). ps.close(). conn.close(). return user. } public static void 0deleteUser(int userId) throws SQLException { Connection conn = …..//… } public static User getById(int userId) throws SQLException { //… } //… } 以上就是一个简单的数据包装的基本雏形，我们可以看到，这是一个非常简单的JDBC包装，一些代码可以模块化，以实现重用。另外，这段代码还有很大隐患，就是中途如果出现异常的话，就会使系统出现JDBC资源漏洞，因为JDBC分配的资源（conn,ps,rs等）是不能被Java虚拟机的垃圾回收机制回收的。因此，我们的addUser方法就需要改成下面的样子： public static User addUser(String name, Date birthday) throws SQLException { Connection conn = null. PreparedStatement ps = null. ResultSet rs = null. User user = new User(). 100Test 下载频道开通，各类考试题目直接下载。详细