

Oracle数据库10g、Java、 JDBC和Web服务

Oracle白皮书
2004年6月

Oracle数据库10g、Java、JDBC 和Web服务

介绍.....	4
Oracle 数据库 10g 中的 JDBC.....	4
重新设计基础.....	5
统一驱动程序代码库.....	5
支持 Java、J2EE、Web 服务和 GRID 服务的最佳数据库连通性.....	5
全面的 JDBC 3.0 支持.....	5
可管理性 / 易用性 / 灵活性.....	6
全新和增强的数据库类型支持.....	6
兼容 type-2 和 type-4 JDBC 驱动程序.....	7
智能和可靠的 JDBC 连接缓存.....	8
隐含连接缓存.....	8
透明访问连接缓存.....	8
支持任何经过用户鉴权的连接.....	9
基于用户定义的属性的连接检索.....	9
向存储的连接应用连接属性.....	11
连接缓存管理器.....	12
快速连接故障切换.....	14
快速连接故障切换与 TAF 对比.....	15
Oracle 数据库 10g 中的Java.....	16
利用 Oracle 数据库 10g 的 Java 降低您的成本.....	16
削减成本.....	17
降低风险.....	17
扩展数据库的范围与功能.....	18
使用数据库 Java 特性构建一个定制集成框架——TECSIS 案例研究.....	18
关于公司.....	19
关于应用.....	19
业务和技术要求.....	19
集成框架体系结构.....	20
综合全部.....	22
Oracle 数据库 10g 的 Web 服务.....	23

数据库成为 Web 服务消费者程序	24
构建 “Database as Web Services Consumer” 应用	24
数据库作为 Web 服务提供程序	26
建立 OC4J	26
Web Services Assembler	27
构建 SQL Query 和 SQL DML Web 服务	28
构建 PL/SQL 或 Java Web 服务	32
结论	33
参考资料	33

Oracle数据库10g、Java、JDBC和Web服务

介绍

通过服务器整合和网格计算，数据库可满足新的要求，如通过服务器群来部署和管理集群数据库，以便有效地动态调整数千位并发用户对几百 TB 数据的访问。Oracle 数据库 10g JDBC 驱动程序引入了新一代驱动程序，能够更有效地处理基础设施服务，如数据库连接池，以及传统上由应用服务器或其容器处理的高可用性数据库连接。在大致掌握了 Oracle 数据库 10g JDBC 的新功能后，我们将介绍全新的连接池和快速连接故障切换机制（RAC 和 GRID 环境中）。

从 Oracle8i 起，Oracle 数据库就内嵌了 Java VM，其与 SQL 相结合，打开了通向传统 RDBMS 难以涉足的高级应用的大门。我们将重点介绍在关系数据库中集成 Java 所带来的能力和扩展功能，包括：实施复杂数据库程序，触发外部操作（Web 组件、EJB 组件、ERP 框架、外部数据库、以及外部 Web 服务），这些都取决于数据事件。

以前，数据库通常通过连接协议进行访问，在本部分我们将介绍非连接客户端如何通过 Web 服务协议触发数据库操作；以及您如何在数据库中使用以 Web 服务形式提供的外部动态数据。数据库 Web 服务与 Oracle 数据库（SQL、PL/SQL、Java-in-the-Database、JDBC、HTTPclient、SOAP 客户端）和 Oracle 应用服务器（Java、J2EE、JDBC、SOAP 服务器、XML、HTTP）的能力完美结合。JPublisher 可简化和阐明 Web 服务。

Oracle 数据库 10g 中的 JDBC

JDBC 为传统的 Java 对象（a.k.a.POJO）、以及 J2SE 应用、J2EE 组件、Web 服务、ERP 数据包、O/R 映射框架、以及 GRID 服务明确或隐含地提供了数据库连通性。Oracle 数据库 10g 旨在提供一流的 JDBC 驱动程序，它更加快速有效、更可靠、更具可用性，兼容大多数 J2EE/JDBC，并且支持 Web 服务和网格。该演示的第一部分概述了新特性，包括面向所有驱动程序类型的通用架构调整基础；主要性能增强；全面的标准 JDBC 3.0 特性支持；增强的可管理性、简单易用性和跟踪能力；功能兼容 type-2 和 type-4 驱动程序；新数据库类型和增强的 LOB 与 VARRAY 支持；面向 Web 服务的 JDBC Web RowSet（JSR-114）支持；全新隐含连接缓存和快速连接故障切换；以及未来支持网格计算的透明对话移植。第二部

分更详细地描述了全新隐含连接缓存，它提供了一种透明机制来使用和管理数据源连接缓存，消除了使用和集中数据源的复杂性、混乱，而且快速连接故障切换与 RAC 事件通知机制相结合，可提供早期数据源连接故障检测和更快速的故障切换。

重新设计基础

统一驱动程序代码库

自 Oracle8i 起，Oracle JDBC 开发与执行部门严格评估了 Oracle JDBC 驱动程序，旨在：为所有驱动程序提供一个统一的代码库以及最佳性能。为各类驱动程序提供统一的代码库，包括 **type-2** 客户端、**type-2** 服务器、**type-4** 客户端和 **type-4** 服务器，将允许在一个基础代码内对新特性进行更快速地实施，从而减少甚至消除各驱动程序的功能差距。

执行速度最快的 JDBC 驱动程序 – 更快速的 JDBC 应用执行

性能是重新设计基础的第二大主要目标。为了提供执行速度最快的驱动程序 – 更快的 SQL/XML 数据查询/检索、执行最快的定制

Java/J2SE/J2EE 应用，以及最佳标准 SPECJApp200x 性能指标评测结果，我们已经：监视并改进了代码路径长度，即执行的设备指令数量；通过使用能够消除许多 Java 中介和临时对象的创建，并能够存储和重复使用大量此类对象的算法，最大限度减少了满足一个请求所需调用的 Java 方法；我们还最大限度减少了来往于数据库的数据；并优化了 Java/SQL 数据转换操作。此外，诸如本机 IEEE 双精度浮点数据（Double and Float）支持和 XA 处理优化等众多新特性还将进一步提高性能。

支持 Java、J2EE、Web 服务和 GRID 服务的最佳数据库连通性

全面的 JDBC 3.0 支持

JDBC 3.0 最先在 Oracle9i R2 JDBC 驱动程序中提供支持，具备：事务保存点、本地与全球交易转换、重复使用 PreparedStatement、以及对客户端 JDBC 驱动程序的 JDK 1.4 支持。

在 Oracle 数据库 10g 中，JDBC 驱动程序提供了更多的全新 JDBC 3.0 特性：

命名参数：这一版本支持 CallableStatement 和 PreparedStatement 中的命名参数，使 JDBC 应用能够按名称传递参数，以及按名称登记和检索输出参数。以隐藏的方式，JDBC 驱动程序首先按名称收集和存储来自所有设置输入参数的信息（以及按名称收集和存储所有登记输出参数）；然后重写呼叫；以及在执行前按次序进行所有设置（和寄存）。

新参考接口和数据链路/URL：DATALINK 值参考了来自底层数据源、并由该数据源进行管理的文件。JDBC 3.0 规范可将此全新 JDBC 数据类型映射到 Java 类型 java.net.URL；并在 java.sql.Types J2EE 连接器架构资源适配器中添加新类型代码：

新程序包 `oracle.jdbc.connector` 实施了 J2EE 连接器规范 `javax.resource.spi` 的服务提供程序接口合约。Oracle JDBC 驱动程序可作为支持 Oracle 数据库的标准 J2EE 资源适配器；支持增强的插接能力（plug-ability）、封装和部署。

以及连接池和 *JDBC Web RowSet*，将在本文后面进行描述。

面向 Web 服务和网络服务的最佳数据库连通性

数据库 Web 服务：以隐藏的方式使用 JDBC 调用数据库操作，如 PL/SQL 程序包、Java 存储过程、SQL 查询、SQL DML 等，Web 服务使用 Java Wrapper，部署在中间层 Oracle 应用服务器上。

JDBC Web RowSet (JSR-114)：JSR-114 的早期实施，支持分散的应用（如 Web 服务客户端或 J2EE 组件），从数据库表（或其它数据源）读取一系列 XML 格式的行 — 以 XML 文件输出的 RowSet，以便数据源中没有活动的连接，以及浏览、更新、和与数据源同步数据行。

透明会话移植：在近期，JDBC 驱动程序将支持面向部署于网格环境中的 RAC 数据库的移植非会话状态和会话状态的数据库对话。

与 Oracle 数据库紧密集成。

此版本还提供了全套特性，包括可管理性、易用性、灵活性、新数据类型，以及对 LOB 和 VARAY 的增强，这使客户能够全面访问 Oracle 数据库功能。

可管理性 / 易用性 / 灵活性

端到端跟踪：JDBC 驱动程序向 RDBMS 引擎传播中间层 Web 客户端 id，以便跟踪从 Web 浏览器到 SQL 相应操作的资源消耗。

即时客户端：新封装支持 type-2 “OCI” 驱动程序的简单无忧 “homeless” 安装。

增强的 Oracle JDBC Datum 支持：Datum 类与其余 jdbc 的分离可以使 datum 用户下载容量较“小”。更好（fier-grain）的异常事件而不是到处存在的 SQLException。Datum 数据的算法和转换操作。该特性将为采用 `oracle.sql.Datum` 的 JDBC 应用提供更小的下载容量；新 datum 转换和算法操作及更详细的异常处理。

全新和增强的数据库类型支持

以下增强或新数据类型可增加 JDBC 驱动程序与 Oracle 数据库的集成，从而带来更快速的应用开发和执行。

Native IEEE DOUBLE 和 Native IEEE Float：全新 SQL 浮点数据类型在 JDBC 中可得到支持，从而使采用 JDBC 的 Java 和 J2EE 应用能够执行更快的算术计算，而不会丢失信息，并可减少存储。要将浮点数或双精度捆绑到一个 `binary_float` 或 `binary_double` 输入参数中，可将准备好的语句提供给 `oracle.jdbc.driver.OraclePreparedStatement`，并调用

`setBFloat()` 或 `SetBDouble()`。如果您转而调用 `setFloat()` 或 `setDouble()`，实际数据捆绑将针对 `NUMBER` 列完成，从而将会造成性能损失以及数据破坏。对于 `CallableStatement` 捆绑或来自结果集合的数据没有此类限制。

`binary_float` 和 `binary_double` 不可作为支持 Java 存储过程的参数，通常在服务器端内部驱动程序中。

增强的 VARRAY 支持：添加聚合功能（高频项设置计数）；使用集合列表（应用请求）；通用设置操作 – 交叉、联合、成员、相等，等 – 对于数据挖掘应用特别有帮助。

LONG 到 LOB 转换：针对 `CLOBs` 和 `BLOB` 添加了转换功能，因此它们可与 `LONG` 兼容；`RAW` 和 `LONG RAW`，简化了其在 JDBC 应用中的操作，可改进 JDBC 应用可移植性。

无限大小的 LOB：该特性消除了对于 `LOB` 的 4 GB 长度限制；JDBC 应用现在可以存储/检索非常大的二进制数据。

INTERVAL DAY TO SECOND：JDBC 应用现在可以使用数据库的 `INTERVAL DAY TO SECOND` 数据类型来改进时间管理。

兼容 type-2 和 type-4 JDBC 驱动程序

以下内部增强和特性允许灵活地使用 type-2 “OCI”或 type-4 “瘦” JDBC 驱动程序进行开发和部署：

PL/SQL 索引表：该特性使您可以发送和接收 type-4 “瘦” JDBC 驱动程序中的 PL/SQL 表。使用 `setPlsqlIndexTable()` 和 `getOraclePlsqlIndexT`，您可以交换 Java 集合和 PL/SQL 集合。

新加密算法：type-4 “瘦” JDBC 驱动程序现在支持 3DES112 和 3DES168 作为 JDBC 瘦驱动程序中连接属性 `SQLNET.ENCRYPTION_TYPES_CLIENT` 的值。

直接 XA：使用 type-4 “瘦” JDBC 驱动程序时对 JDBC XA 操作的优化。

代理鉴权：type-4 “瘦” JDBC 驱动程序现在可提供支持多个中间层用户/线程共享同一鉴权用户下的一个数据库连接的能力。

数据库启动/终止：使用 type-4 “瘦” JDBC 驱动程序支持 Oracle 的 EM 及第三方管理工具启动/终止数据库的能力。

RAC/HA 故障切换：type-4 “瘦”和 type-2 “OCI”现在均可提供对 RAC 的兼容支持。详见第二部分，了解有关快速连接故障切换的更多信息。

Miscellaneous

设置 `CHAR/NCHAR` 行为的系统属性。

停止发运 classes111 和 zip 文件: 由于 Sun Microsystems 不再支持 J2SE 1.1.x JDK, 因而从此版本开始将不再提供 JDBC classes111.jar。此外, Oracle JDBC 将只作为 JAR 文件提供, 将不再提供 ZIP 文件。

智能和可靠的 JDBC 连接缓存

在第二部分, 我们将更详细地介绍两个主要新特性: “隐含连接缓存”和“快速连接故障切换”。

隐含连接缓存

在 Oracle 数据库 10g 之前, JDBC 驱动程序可提供连接缓存支持, 然而在可扩展性、易用性和可管理性方面存在诸多限制。例如, 所有对象均在同一数据库中, 并作为同一用户进行鉴权, 而没有清除失效连接的机制; 此外, 由于所有对话均属于同一数据库, 并作为同一用户进行鉴权, 因此根本无法存储作为其它用户进行鉴权的连接。Pre-10g JDBC 驱动程序缓存管理不允许连接缓存在可能失效时调整大小或刷新; 不支持搜索连接或收回所放弃的连接。这些缺点促使了对一种更完善的新型连接缓存机制“隐含连接缓存”的需求。

隐含连接缓存可提供丰富的特性, 包括: 透明或隐含访问连接缓存, 支持存储任何经过鉴权的连接, 能够刷新或重复利用来自缓存的失效连接, 支持基于用户定义的属性的连接检索、基于属性和权重的连接检索。

透明访问连接缓存

通过缺省设置, DataSources 允许获得直接到数据库的物理连接。利用“隐含连接缓存”, 通过简单将 DataSource 属性 *ConnectionCachingEnabled* 指向 *true*, 便可从连接缓存获得连接, 始终使用同一标准 *getConnection()* API。这极大简化了 DataSource 和连接缓存访问。其它所有事项均可选, 并可根据需求来执行。例如, 可以选定一组定义缓存行为的连接缓存属性, 而不必使用缺省属性。

```
// Example to show binding of OracleDataSource to JNDI
// with relevant cache properties set on the DataSource.
...
// Set DataSource properties
ods.setUser("Scott");
...
ods.setConnectionCachingEnabled(True);
ods.setConnectionCacheName("MyCache");
ods.setConnectionCacheProperties(cp);
ctx.bind("MyDS", ods);
...
ods =(OracleDataSource) ctx. lookup("MyDS"); // lookup cache DataSource

//Transparent creation and retrieval of connection(s) from "MyCache"
conn = ods.getConnection();
...
```

```
conn.close(); // return connection to the cache
```

```
...
```

支持任何经过用户鉴权的连接

连接缓存保留到数据库的连接。虽然数据库不能对连接鉴权施加任何限制，但传统缓存可能会强加此类限制。隐含连接缓存可以处理任何经过用户鉴权的连接。例如，*joe.blow* 连接可在同一连接缓存中与 *sue.miller* 连接很好地共存。

基于用户定义的属性的连接检索

隐含连接缓存支持在将连接返回到缓存之前对其进行分割处理的理念，从而可以将用户定义的属性应用于注销连接（checked out connection）。这些属性之后可用于检索来自缓存的同一连接。

示例1：基于连接属性 **NLS_LANG** 的检索

```
// Look up the datasource object
javax.sql.DataSource ds = (javax.sql.DataSource)
ctx.lookup(MyOracleDataSource);

// get a connection from MyCache
java.util.Properties connAttr = null;
connAttr.setProperty("NLS_LANG", "ISO-LATIN-1");
conn = ds.getConnection(connAttr); // retrieve connection - NLS_LANG
...
conn.applyConnectionAttributes(connAttr); // apply attributes

Statement stmt = conn.createStatement();
stmt.execute("select empname from emp");
...
conn.close(); // release the connection back to MyCache
```

示例2：基于连接属性隔离等级的检索

```
...
java.util.Properties connAttr = null;
connAttr.setProperty("TRANSACTION_ISOLATION", "SERIALIZABLE");
conn = ds.getConnection(connAttr); // retrieve connection that matches
Transaction Isolation
...
conn.close(connAttr); // another way to apply attributes to the connection
```

示例3：基于连接属性、连接标记（预留连接）的检索

```
...
java.util.Properties connAttr = null;
connAttr.setProperty("CONNECTION_TAG", "JOE'S_CONNECTION");
conn = ds.getConnection(connAttr); // retrieve connection that matches Joe's
connection
...
conn.close(connAttr); // apply attributes to the connection
...
conn = ds.getConnection(connAttr); // This will retrieve Joe's connection
```

基于属性和权重的连接检索

可根据 *ConnectionAttributes* 和属性权重从连接缓存搜索连接。权重被指定给 *ConnectionAttribute* 中的各密钥，这是一次性的操作，是作为缓存属性对缓存进行的一项设置。缓存属性 *CacheAttributeWeights* 是 *java.util.Properties*，允许属性权重设置。各权重是一个整数值，用于定义密钥的重要程度。一旦指定了缓存的权重，便可在 *DataSource* 上调用 *getConnection(connectionAttributes)* 发出连接请求。这些 *connectionAttributes* 是密钥及其相关值。从缓存的连接检索涉及搜索满足以下组合的连接：

- 密钥/值匹配缓存的连接
- *connectionAttributes* 的所有密钥的最大总权重匹配此连接。

考虑以下示例。采用 **CacheAttributeWeights** 配置的缓存如下所示：

```
java.util.properties cacheProps = new Properties();
java.util.properties cacheWeights = null;

cacheWeights.setProperty("NLSLANG", "10");
cacheWeights.setProperty("SecurityGroup", "8");
cacheWeights.setProperty("Application", "4");
...
// set weights on the cache
cacheProps.put(CacheAttributeWeights, cacheWeights);
...
...

```

一旦设置了权重，便可提出连接请求，如下所示：

```
java.util.properties connAttr = null;
connAttr.setProperty("NLSLANG", "ISO-LATIN-1");
connAttr.setProperty("SecurityGroup", "1");
connAttr.setProperty("Application", "HR")

// Request connection
ds.setCacheName("MyCache");
// First retrieval of connection from myCache
conn = ds.getConnection(connAttr);
...
conn.close(connAttr); // apply attributes on the connection
...
// Next retrieval finds the connection in the cache
conn = ds.getConnection(connAttr);
...

```

getConnection() 请求尝试检索来自缓存的连接，*MyCache*。满足条件，并可从缓存中检索到的连接包括如下情况：

- 发现完全匹配。根据上述举例，完全匹配是满足相同属性值和所有密钥（*NLS_LANG*、*SecurityGroup* 和应用）的连接。
- 未发现完全匹配。在这种情况下，使用属性密钥/值及其相关权重最接近的连接（条件是设置了 *ClosestConnectionMatch* 属性）。例如，最

接近的匹配值可能是一个包含匹配 `NLS_LANG` 和 `APPLICATION` 的属性，但非 `SECURITY_GROUP` 的连接。还可能找到匹配原始列表中部分密钥，但其组合权重相同的连接。例如，连接 1 的相关权重为 10，与 `NLS_LANG` 匹配；而连接 2 的综合权重为 12，与 `SECURITY_GROUP` 和 `APPLICATION` 匹配。在本例中，要求返回连接 2。换言之，连接2 是最接近的匹配值，与连接1 相比权重建起来更昂贵（从调用者角度）。当没有 `connectionAttributes` 相匹配时，则返回一个新连接。新连接可使用 `DataSource` 上的用户和密码设置进行创建。

一旦连接被返回，用户便可调用连接对象上的

`getUnMatchedConnectionAttributes()` API 来返回一组不匹配此标准的属性（`java.util.Properties`）。这些不匹配的属性列表由调用者（或应用）用于在使用此连接前重新预置这些值。

向存储的连接应用连接属性

有两种方法可以向缓存中的连接应用连接属性。通过调用连接对象上的 `applyConnectionAttributes(java.util.properties connAttr)` API。这可简单设置连接对象上所提供的属性。可以使用该 API 以递增方式应用这些属性，从而允许用户应用连接属性到多个调用。例如，`NLS_LANG` 可通过从 模块 A 调用此 API 来应用。然后来自模块 B 的下一个请求可应用 `TXN_ISOLATION` 属性，以此类推。

通过在连接对象上调用 `close(java.util.properties connAttr)` API。该 API 可关闭此逻辑连接，然后在基本 `PooledConnection`（物理连接）上应用所提供的连接属性。如果可能，通过此 `close()` API 设置的属性将覆盖采用 `applyConnectionAttributes()` API 设置的属性。

以下举例显示了在连接对象上对 `close(connectionAttributes)` API 的调用，这使得缓存在将匹配的 `connectionAttributes` 返回到缓存之前，返回到 `PooledConnection`。从而可确保当以后提出具有相同连接属性的连接请求时，此缓存可找到匹配连接。

```
// Sample connection request and close
java.util.properties connAttr = null;
connAttr.setProperty("NLSLANG", "ISO-LATIN-1");
conn = ds.getConnection(connAttr); // request connection based on attributes
java.util.properties unmatchedAttr =
conn.getUnMatchedConnectionAttributes();
...
--- App Server code applies unmatched attributes to the connection, either by
calling PL/SQL procedures or SQL, before using the connection. ---
--- work ---
...
conn.close(connAttr); // apply attributes to connection
```

连接缓存管理器

连接缓存管理器可提供集中的方法来管理一个或多个连接缓存。每个虚拟机 (VM) 有一个连接缓存管理器例程，可管理所有连接缓存。在使用任何连接缓存管理器功能之前，必须获得此连接缓存管理器例程。要获得连接缓存管理器例程，必须调用连接缓存管理器类提供的静态 `getter` 方法。连接缓存管理器主要有两个作用：

缓存管理与维护： 连接缓存管理器负责创建缓存，维护缓存的功能，删除缓存。连接缓存管理器“了解”各缓存的存在。各缓存独立管理。提供丰富的 API 以执行连接缓存管理器任务。

绑定连接缓存与 DataSource： 连接缓存管理器可确保连接缓存与其 `DataSource` 对象间的关联。这可在每次创建、删除或重新初始化连接缓存时得到加强。从而可确保以有效的方式访问连接缓存和检索来自缓存的连接，以支持 `DataSource` 上的每个 `getConnection()` 请求。

相对于在访问 `DataSource` 时透明创建缓存，可以使用连接缓存管理器 API 明确地创建一个缓存。如果缓存已经存在，则进行第一次调用以获得相应 `DataSource` 上的连接，略过缓存创建过程，简单将连接请求路由至该缓存。一旦选定了缓存，可将连接检索程序交由缓存本身处理。该缓存可搜索此连接，无论是基于用户鉴权或者基于连接属性执行更详细的搜索。

多缓存支持

连接缓存管理器支持一个以上缓存共存。各缓存通过特有的名称进行识别，它们与 `DataSource` 紧密绑定。每一缓存或者当 `getConnection()` 请求在缓存支持的 `DataSource` 上发出时，透明创建，或通过 `Connection Cache Manager API` 在中间层直接创建。采用支持多个缓存的 `DataSources`，提供以下优势：

请求到一个以上 `DataSource` 的连接，特别是当每个 `DataSources` 都指向不同的基本数据库时。

通过连接缓存管理器管理配置的所有缓存。这使得管理缓存更加轻松，且无需在 `Application Server` 代码中保持不必要的内存块（使用 `Connection Cache Manager` 的代码）。

连接缓存管理器对于可创建的缓存数量没有限制。此数量受到 `JVM` 和 `Oracle` 数据库服务器上的可用资源的限制。

一旦创建了缓存，可以通过连接缓存管理器将其明确删除，或者在 `DataSource` 使用后关闭时通过 `DataSource close()` API 将其删除。

连接缓存属性

使用一组连接缓存属性，很容易限制和调整连接缓存。缓存属性可在初次创建缓存时在 `DataSource` 上作为连接缓存属性进行设置，或在重新初始化缓存时，通过连接缓存管理器设置。请参阅连接缓存管理器 API 部分，了解如何在缓存上设置缓存属性的详细信息。

MinLimit: 这可设置缓存可维护的 **PooledConnections** 的最低数量，确保此缓存不会减少到此最低限值以下。该属性不能利用最低数量的连接初始化缓存¹。请参阅 **InitialLimit** 属性部分了解详细的缓存信息。缺省值为 **0**。

MaxLimit: 这可设置缓存可容纳的最大数量的 **PooledConnections**。该缺省值未绑定，这意味着没有假定的 **MaxLimit**。

由于连接缓存不假定最大限值，因此它只受到为数据库配置的数据库对话的数量限制。换言之，缓存中的连接可达到数据库允许的最大范围。

InitialLimit: 这一参数在初次创建或重新初始化缓存时设置连接缓存的大小。当该属性被设置为大于 **0** 的值，则可以预选创建许多连接，并可随时使用。该属性通常用于在促进缓存达到其最佳大小时缩短“加速”时间。缺省值被设置为 **0**。

动态重新配置支持

隐含连接缓存允许通过专用的全新缓存属性重新初始化缓存，从而动态重新配置缓存属性。这些新属性可在新创建的所有 **PooledConnections** 上及未使用的 **PooledConnections** 上生效。对于使用中的连接，新属性仅在那些连接返回到缓存后生效。

放弃的连接支持

隐含连接缓存可处理放弃的连接。放弃的或孤立的连接是那些已在缓存外检查过，但从未返回到缓存的连接。连接被放弃的原因有很多。例如，用户线程在注销连接后，可能被中断。当一个连接被放弃，最好是能够自动检测和重新使这些连接返回到缓存，而这正是连接缓存上的

AbandonedConnectionTimeout 属性的作用。将该属性设置为有效的超时值，启动所检查的连接上的心跳监视程序。心跳是在所检查的连接上，针对数据库的任何 **SQL** 活动。当心跳未在指定时间内在连接上进行记录，该连接即被认为是被放弃，并自动宣称返回缓存。

语句缓存支持

MaxStatementsLimit 连接缓存属性将最大语句设置为始终为一个连接缓存开放。如果缓存利用此属性进行重新初始化，如果可以存储而不是指定更多指针，则可以关闭额外的指针。缺省值为 **0**。

连接重复利用支持

在一段时间内，连接缓存会积累失效连接。可以通过两种方式来重复利用或刷新缓存中的失效连接。(1) **REFRESH_INVALID_CONNECTIONS** (2) **REFRESH_ALL_CONNECTIONS**。当通过无效的连接模式进行调用，可检查缓存中的各 **PooledConnection** 是否有效。如果发现一个无效的 **PooledConnection**，该连接的资源将被删除并替换为新的 **PooledConnection**。有效性测试基本上是对 **dual table** 的一种简单查询：

¹ 采用 **OracleConnectionCacheImpl** 时是这种情况

select 1 from dual; 当采用所有连接模式进行调用时，缓存中的所有可用连接被关闭，并替换为新的有效物理连接。

监视连接缓存

这些 API 可在连接缓存管理器上被调用，以获得缓存信息，如被检查的连接数量，或缓存中可用连接的数量。

getNumberOfAvailableConnections

int getNumberOfAvailableConnections(String cacheName)

此 API 返回连接缓存中的连接数量，这些可用。所返回的值是在处理 API 时连接缓存中可用连接数量的快照，因此是一个统计值。

getNumberOfActiveConnections

int getNumberOfActiveConnections(String cacheName)

此 API 返回所检查的连接的数量。这些是使用中或繁忙的连接，因此当前不可用。所返回的值是在处理 API 时连接缓存中所检查的连接数量的快照，因此是一个统计值。

getCacheProperties

java.util.properties getCacheProperties(String cacheName)

这可检索面向指定缓存名称的缓存属性。

getCacheNameList

String[] getCacheNameList()

此 API 返回为连接缓存管理器所知的所有连接缓存名称。然后该缓存名称可被用于管理连接使用连接缓存管理器 API 的缓存。

快速连接故障切换

快速连接故障切换与隐含连接缓存机制和 RAC 数据库共同发挥作用。

当针对多例程 RAC 数据库对连接缓存进行了设置，数据库连接可在任何 RAC 例程上结束，具体根据数据库监听程序的指定。当一个例程结束时，同于例程或主机故障，它会使所有连接结束。例如，在一个双节点中，两个例程 RAC 集群，考虑这样一种环境：其中每个例程有 50 个连接，这样连接缓存在缓存中可达到 100 个连接。如果其中一个 RAC 例程结束，则立即缓存中会出现 50 个坏连接。这会潜在地导致多个坏连接从缓存处返回，这反过来又会导致应用或浏览器页面错误。

快速连接故障切换支持使自动检测和修复机制能够处理 RAC 环境中的任何例程或主机故障。通过简单地将 **DataSource property *FastConnectionFailoverEnabled*** 指向 **true**，可在支持缓存的 **DataSource** 上启用该机制。

// Example to show binding of OracleDataSource to JNDI

```

// with relevant cache properties set on the DataSource.

import oracle.jdbc.pool.*; // import the pool package

Context ctx = new InitialContext(ht);
OracleDataSource ods = new OracleDataSource();

// Set DataSource properties
ods.setUser("Scott");
ods.setPassword("tiger");
ods.setConnectionCachingEnabled(True);
ods.setConnectionCacheName("MyCache");
ods.setConnectionCacheProperties(cp);
setURL("jdbc:oracle:thin:@(DESCRIPTION=
    (LOAD_BALANCE=on)
    (ADDRESS=(PROTOCOL=TCP)(HOST=host1) (PORT=1521))
    (ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))
    (CONNECT_DATA=(SERVICE_NAME=service_name)))");
ods.setFastConnectionFailoverEnabled(true); // Enable fast connection
failover
ctx.bind("MyDS", ods);
...
ds = lookup("MyDS"); // lookup DataSource from the cache
// implicitly create connection cache, that is set up for fast connection failover
conn = ds.getConnection();
...
conn.close(); // return connection to the cache
...
ods.close() // close datasource and cleanup the cache

```

快速连接故障切换机制通过处理 Service DOWN 或 UP 事件和 Host DOWN 事件来发挥作用。DOWN 事件处理总是清除缓存中的坏连接。UP 事件处理在缓存中进行连接负载平衡。

快速连接故障切换功能可提供以下优势：

*当检测到 RAC 例程/节点故障，快速关闭连接缓存中的连接：*这可避免将坏连接或无效连接提供给应用连接请求。对于依赖于隐含连接缓存进行总体连接管理的应用，快速连接故障切换机制可提供最大连接可用性。

*当生成 RAC UP 事件时，对连接进行负载平衡：*在这种模式下，可针对所有有效 RAC 例程，建立连接并对其进行负载平衡，而无需等待应用连接重试/请求。注意，对于 RAC 设置，该服务几乎总是即时可用，除了当全部服务失效时。

快速连接故障切换与 TAF 对比

OCI/网络层提供透明应用故障切换（TAF）支持，建立到第二例程的连接，无论主例程是否失败。使用此连接的应用可看到 SQL Exception，并在需

要时处理对话状态重新初始化。因此，连接重试和连接预置可由 OCI/网络层来完成。

快速连接故障切换与隐含连接缓存和 RAC 数据库一同发挥作用，这极大提高了连接缓存中有效和实用连接的可用性。如果应用正在交易，它将会收到一个适当的 SQL exception，此交易将会退回重来。由应用或容器负责重试此连接请求，并重建对话状态。

这两种机制的主要区别在于连接重试出现的方式。利用快速连接故障切换，重试由应用控制，可以决定是重新向请求者提供 SQL Exception 还是重试。

注：目前，快速连接故障切换不应与 TAF 一起使用。原因在于 TAF 的工作等级低于 JDBC。因此，当这些特性一起使用时，快速连接故障切换机制可能会删除某些 TAF-Failover'd 连接。

Oracle 数据库 10g 中的 Java

自 Oracle8i 以来，Oracle 数据库 提供了一个内置的可扩充 Java VM (OracleJVM)，以运行 Java 存储过程，以及标准和第三方 J2SE、JDBC、JMS 和 JAI 库。它们直接在数据库中与数据临近的位置实施数据逻辑和/或持久性逻辑。借助 Oracle 数据库 10g，我们进一步改进了这一功能。

本文的第一部分概要介绍了 Oracle 数据库 10g 的全新 Java 特性。这些特性将可以帮助您降低成本与风险，同时有效扩充 Oracle 数据库的功能。第二部分将介绍 TECSIS 案例研究：该公司如何通过实施一款易于使用的框架来降低其成本，以集成其包含 PL/SQL 存储过程、SAP R3、Natural/Adabas、RPG/DB400、COBOL Tandem、COM 组件和使用 OracleJVM 的非 Oracle 数据库 (Adabas-D 和 MSSQL Server) 的平台，进而充分利用其功能。

利用 Oracle 数据库 10g 的 Java 降低您的成本

在这一全新的数据库版本中，我们将 Java 引擎升级到 J2SE 1.4.x，显著改善了 Java 与 SQL、PL/SQL、XML、J2EE 和 Web 服务的集成。例如，一个纯 Java SOAP 客户端库可将您的数据库转变成为一个 Web 服务消费者。我们优化了 Java 内存管理、Java 类加载和 Java 池调整；我们还重新设计了服务器端 JDBC 驱动程序，以加快会话期间的 SQL 数据访问。您通过这些新特性可以享受到哪些优势？

削减成本

我们将简单阐述数据库中的 **Java** 如何能够帮助降低应用开发、部署与执行成本。

速度更快的低成本 Java 应用开发

升级 **OracleJVM** 到 **J2SE 1.4.x** 支持根据需要在数据库中重复使用大型最新的 **Java** 库，以降低开发成本和周期。重复使用现有 **Java** 技能构建定制库的能力，或重复使用库的能力，也可以帮助加快数据库应用开发速度，同时降低成本。

此外，全新 **Native Java Interface** 支持进行直接的 **Java** 到 **Java** 调用，而无需采用 **PL/SQL** 包装。而用于为 **J2EE** 和 **Web** 服务提供更多数据库功能的 **JPublisher** 程序扩展，诸如 **SQL Query**、**SQL DML**、以及数据库中的 **Java** 类等，简化了数据库与这些环境的集成，从而可显著降低总体应用开发时间和成本。

更快的应用配置与执行

LoadJava 程序中的全新字节码验证器加快了数据库的应用加载和部署时间。

当使用专用服务器时，全新服务器端 **JDBC** 驱动程序的整个结构调整，以及 **Java** 内存管理优化可显著加快会话间 **SQL** 数据访问——调用相同内存空间中的函数，同时也可加快数据库中 **J2SE**、**JDBC**、**SQLJ**、**JMS** 和 **JAI** 应用执行。

低成本平台集成

请参阅第二部分的 **TECSIS** 案例研究，了解使用 **OracleJVM** 的定制的、低成本集成框架。

降低风险

选择

凭借数据库 **Java** 特性，**Oracle** 客户可以选择数据库编程语言——**Java** 或 **PL/SQL**。其优缺点不再本文的讨论范围之内。在本版本中，与 **PL/SQL** 类似，**Java** 存储过程现在可以将所有 **SQL** 异常事件返回至调用程序，从而使得 **Java** 成为一个理想的选择。对最新 **Java** 标准，包括 **J2SE 1.4.x**、**JDBC 3.0**、**JAI 1.0**、**JMS 1.x** 和 **JAX-RPC** 客户端等的支持，允许在中档/**J2EE** 和数据库一级进行 **J2SE**、**JDBC**、**JMS** 和 **JAI** 应用，从而使得客户在部署体系结构时可以进行全新的选择。此外，全新 **Native Java Interface** 支持为数据库中的 **Java** 特性选择替代创新模式：或直接通过 **Java/J2EE**，或通过传统 **PL/SQL** 包装。

安全和可靠的 Java 执行

除了强大的安全机制², OracleJVM 还支持最新的标准, 和所需的 J2SE 1.4.x 安全特性, 包括 Java GSS-API、JAAS、JCE 和 JSSE 等, 可以创建一个安全的 Java 执行环境, 减少与传统 JDK 相比可以受到的恶意 Java 代码的攻击。请参见 OTN 数据库技术中心³ Java 中的客户案例研究《*JSSE support with HTTP Call-out using Oracle8i and Oracle9i*》。

扩展数据库的范围与功能

扩展您的数据库的范围

这一版本提供了数据库 Web 服务的交付 (参见相应章节)。通过 Web 服务机制 (SOAP 和 WSDL)⁴调用数据库操作的能力, 将您的 Oracle 数据库客户端群扩展未连接的客户端, 以及异构环境。同样, 消耗、联合和使用 HTTP 调用功能将 J2EE 和 Web 服务的数据整合至 JSP/Servlet、直接 RMI/IIOP EJB 调用、以及支持 Web 服务调用的 SOAP 客户端的能力, 将数据库的范围延伸到了动态/按需数据。

开放和安全的数据功能可扩展性

加载和运行标准、第三方或定制 Java 库的能力, 诸如将纯 Java RPC 库加载到外部 ERP/传统系统, 以及将纯 Java JDBC 驱动程序加载到非 Oracle 数据库的能力, 提供了一个扩展数据库能力的开放机制, 如 TECSIS 案例研究所示 (第二部分)。加载一个纯 Java SOAP 库可将您的数据库转变为一个 Web 服务使用者; 通常, 加载标准 JSSE 库支持使用 Oracle8i 和 Oracle9i 的安全 cHTTP Call-out——JSSE 现在需要 J2SE 1.4.x, 因此是 Oracle 数据库 10g 中的 OracleJVM 的不可分割的一部分。

Java 与 SQL 引擎的集成允许将全新能力应用到 SQL 和 PL/SQL。针对安全性和数据完整性, OracleJVM 安全机制允许定义、限制和调试已加载库的范围、优先级和可见性。

使用数据库 Java 特性构建一个定制集成框架——TECSIS 案例研究

在这一部分中, 您将了解到 TECSIS 如何通过实施一个易于使用的企业级框架降低其成本, 以集成不同的平台, 包括 Oracle PL/SQL 存储过程、SAP R3、Natural/Adabas、RPG/DB400、COBOL Tandem、COM 组件、和使用 OracleJVM 的非 Oracle 数据库 (Adabas-D 和 MSSQL Server), 进而充分利用其功能。

² <http://otn.oracle.com/oramag/oracle/03-jul/o43devjvm.html>

³ http://otn.oracle.com/tech/java/java_db/content.html

⁴ <http://www.sys-con.com/webservices/articleprint.cfm?id=515>

关于公司

全球管道技术的领先厂商 Tenaris，代表了 8 家享誉全球的钢管生产商：AlgomaTubes、Confab、Dalmine、NKKTubes、Siat、Siderca、Tamsa 和 Tavsa。凭借 300 万吨无缝钢管和 85 万吨焊接钢管的生产能力，以及 30 亿美元的年销售额，和分布于五大洲的 13,000 名员工，Tenaris 是向全球能源和机械行业提供管道产品与服务的领先厂商。我们的市场份额在 OCTG 无缝产品中占据 30%，在全球无缝钢管销量中占据 13%。Tenaris 的系统技术分部——Teccsis 的主要目标是验证和推广在 Tenaris 集团内部技术。这一部分介绍了我们在使用 Oracle 数据库的 Java 特性的过程中所积累的经验，以及如何解决我们的集成需求。

关于应用

我们公司不仅仅将 Oracle 数据库用做一个数据库，同时还将其作为集成基础设施。我们使用 Oracle 数据库已有三年的历史。我们开始时使用 PL/SQL 存储过程实施了业务规则，并获得了诸多优势。借助数据库中内嵌的 Java 虚拟机（OracleJVM），我们进一步延伸了数据库的功能，使其称为一个数据集成中枢。

业务和技术要求

业务要求

我们的业务要求需要集成来自不同平台的在线信息，包括：SAP、AS400、ADABAS/NATURAL 和 COBOL Tandem。我们基于 PL/SQL 的业务规则需要通过这些平台收发数据。现有传统系统，以及基于内联网/互联网的全新应用开发，需要跨平台集成。我们的主要目标是通过重复使用软件、系统和技能降低成本。

技术要求

我们需要集成以下平台：Oracle PL/SQL 存储过程、SAP R3、Natural/Adabas、RPG/DB400、COBOL Tandem、COM 组件、和非 Oracle 数据库（Adabas-D 和 MSSQL Server）。不同的 RPC 技术可被用于集成传统系统，但我们对其集成度并不满意。当时，在其它平台上利用现有 PL/SQL 数据包提供在线信息的需要变得非常重要。

总体而言，我们最重要的要求为：

1. 简化跨平台集成工作。
2. **成本节省：**与添加一个全新集成层相反，我们决定充分利用现有组件，并将这些组件发挥到最大限度。
3. 此外，我们希望能够避免由点对点集成所产生的通信流量爆炸。

设计和编程选择

我们选择充分利用 OracleJVM 及其能力来在数据库中运行 Java 库，因为所有现有的 ERP 系统和非 Oracle 数据库采用了基于 Java 的远程程序调用 (RPC) 软件或可被加载到 OracleJVM 的纯 Java JDBC 驱动程序。PL/SQL 包装使得这些机制可被应用于以 Java 存储过程为主的 SQL 环境。基于 PL/SQL 的现有或全新业务规则可轻松与其它系统进行交互。通过将我们的业务规则以及转换规则集中于 Oracle 数据库，同时使一切事务均可通过 Web 客户端和批处理工作访问，Oracle 数据库成为了我们的集成引擎。

我们所有的新系统均构建于 Web 页之上，由它调用存储过程，进而访问业务规则。现有批处理工作，以及客户端应用，共享相同的业务规则。我们还使用基于 XML 的 IN 和 OUT 参数，对我们调用参数的方法实施了标准化。这些 XML 参数使用 Oracle XDK 进行解析或生成。

系统在短短几天内即开始运行，而无需花费大量资金重新培训我们的 PL/SQL 程序员的事实，有力地证明了这一解决方案的简单性。此外，批处理工作的集成还通过三行 SQL*Plus 脚本得以实现。

使用一个传统的企业应用集成 (EAI) 产品将更加复杂与昂贵。与之相反的是，在数据库中使用 Java 不仅可以简化我们的集成流程，同时也可以帮助我们节省大量资金。

集成框架体系结构

典型使用情况场景

我们有三种使用情况。第一种，代码验证，系统 A (COBOL Tandem) 需要检查系统 B (Oracle 数据库) 中是否存在特定的代码值。第二种，Pop-Up 列表，系统 A (AS400 screen) 需要使用来自系统 B (Natural Adabas) 的内容显示一个数值列表。第三种，跨平台变动，一款新产品被添加至系统 A (Oracle 数据库)，相同的产品也必须被添加至系统 B (Natural Adabas)。

Java 存储过程调用外部系统

我们选择 Software AG EntireX Communicator (远程 EntireX broker) 来远程调用 Natural/Adabas 程序 (OS/390)、RPG 程序 (AS400)、和 Tandem COBOL 程序。

虽然 SAP Java connector (SAP JCO) 作为一个 JAR 文件进行分配，但它并非完全基于 Java，因为它使用了几个 .sl (库)。出于安全的原因，OracleJVM 不允许 Java 类使用外部 .sl 库 (JNI 调用)。围绕这一限制，我们将 SAP JCO 作为一个外部的 RMI 服务器进行运行。这一方法使得我们能够在 Java 存储过程内部向 SAP JCO 发出 RMI 呼叫。

我们在数据库中加载了第三方纯 JDBC 驱动程序，以在 Java 存储过程和 Oracle 数据库间进行交互。如果我们需要与远程 Oracle 数据库进行交互，我们可以为数据库加载 Oracle 的纯 Java JDBC 驱动程序，亦称作瘦 JDBC。

之后我们每一加载的模块创建了标准 PL/SQL 包装(称作 EAI_PKG)，以根据基于 PL/SQL 的业务规则进行一致的调用。

最后，我们在内部向所有 PL/SQL 提供了一份应用集成指南。在短短的几天内，他们便能够编写出可与其它平台进行交互的程序。

通过统一业务规则，以及添加集成和转换规则，我们创建出了一个完整的数据集成框架。这些业务、集成和转换框架均通过使用 EAI_PKG 的 Java 存储过程与外部系统进行交互。我们的新系统包含了一个基于 Web 的展示层，以及批处理任务(SQL*PLUS 脚本)。我们使用 Oracle Object for OLE (OO4O) 执行来自我们的展示层(.asp)的 Oracle 程序。展示层和批处理工作使用了这一相同的集成框架。

图 1 显示了这种体系结构。

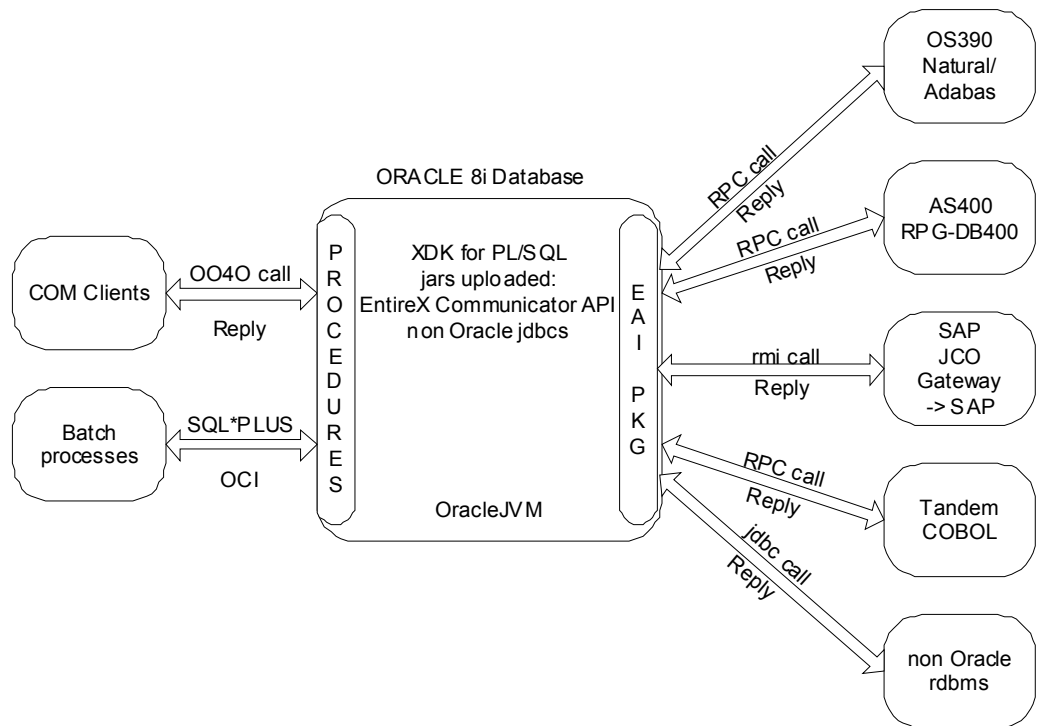
尽管 Oracle Advanced Queuing 系统的概念并未在图 1 中显示，但 EAI_PKG 为每一个远程系统均提供了一个队列。如果一个远程系统停止或不可访问，EAI_PKG 数据库将自动将消息加入相应的队列中。一个内部 DBMS_JOB 工作与每一个队列相关，并以特定的频率运行。这一工作尝试将待处理信息取出，并加以处理，直至目标系统恢复可用。与 Java 存储过程相比，传统 EAI 产品未能提供相同的易于使用性。

外部系统调用 Java 存储过程

当我们能够通过 Java 存储过程调用外部系统时，下一步便是支持外部系统调用 Java 存储过程。为了实现第二步，我们重复使用了 EntireX Communication 和 SAP Java connector。Natural Adabas、AS400 和 COBOL Tandem 将对 EntireX 发出调用命令，之后由 EntireX 调用 Java 存储过程。Java 存储过程的响应被返回至传统系统。

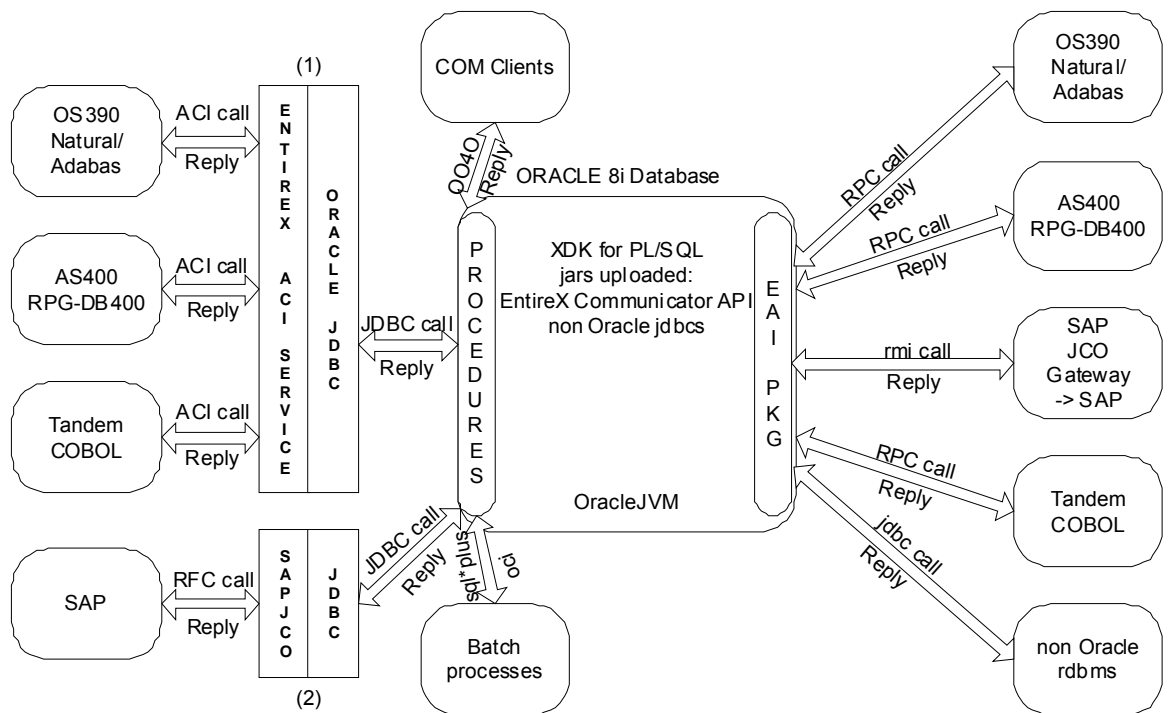
同样，SAP/ABAP 应用调用 SAP Java connector，并由它来调用 Oracle 存储过程。存储过程的响应被返回至 ABAP 应用。

图 1



综合全部

如图 2 所示，通过将 Oracle 数据库作为一个中央通信点，我们使得任意系统都能够与其它系统进行交互，同时避免了点对点的通信。整个框架被 Oracle 存储过程以预先规定的频率进行监视，并向每一目标系统、以及 EntireX 和 SAP connector 进行调用。如果其中一个监视系统返回错误，程序将发出一封通知电子邮件。我们的下一步骤将是向中央控制台发送 SNMP 陷阱。我们当前正在着手解决这一问题。



Oracle 数据库 10g 的 Web 服务

Web 服务是使用 URI 标志的软件应用或组件，为其接口提供了描述，并与 XML 捆绑，同时还可通过基于互联网的协议与采用 XML 的其它 Web 服务进行直接交互。

Web 服务采用了标准格式与协议，从而可以摆脱对特定实施语言

(Java、Managed C++、JScript、Perl、VB.NET、C# 和 J#)、对象模型 (EJB 和 COM 等)、以及平台 (J2EE 和 .NET 等) 的依赖。在其中心，Web 服务通过不同传输机制 (诸如 HTTP、FTP、SMTP 和 JMS 等) 传输采用 XML 格式的信息。当前，大多数 Web 服务都接受支持防火墙的 HTTP 或 HTTPS 格式。XML 本身依据 XML Schema Definition [3] 确定格式。SOAP 消息发送协议[4] 为在 Web 服务之间传递的消息规定了一般形状和处理规则。对于指定的 Web 服务终点 (或端口)，传输和可以接受的消息格式在其 Web 服务描述语言 (WSDL [5]) 文档中进行了规定。具体而言，这包括允许操作，包括参数和返回类型，以及如何展示 XML 格式中捆绑的编程语言的指令。通常，您使用支持特定 Web 服务的 WSDL，以通过您的编程平台的动态调用能力，或通过一个支持您编程语言的通用静态客户端代理，与该服务进行交互。当基于编程语言人造程序 (如 Java 类或 PL/SQL 数据包) 创建 Web 服务时，通常将会生成 WSDL。最后，UDDI 标准[6]

为寻找 Web 服务及其描述提供了一条标准的注册途径。在第一级，您可以将 Web 服务作为一个特定的 XML 消息。然而，考虑到 Web 服务实践和标准的复杂性与不断发展的特性，您通常需要依赖于 Web 服务技术堆栈和工具来发布 Web 服务及其客户端程序。这使得您可以最大限度地减少自身代码的变化，同时最大限度地改进与其它 Web 服务的互操作性。

数据库成为 Web 服务消费者程序

为了使在数据库中运行的代码能够访问其它 Web 服务，诸如股价、天气信息、Web 搜索结果、科学数据或通过 Web 服务可用的企业数据等，我们需要能够调用这些服务。

构建“Database as Web Services Consumer”应用

要使用或调用数据库内的外部 Web 服务，您可以部署您自己的服务（如本文第一部分所述），或使用现有的服务。

第1步 建立数据库

在 Oracle 数据库 10g 中，提供了来自 [Oracle Home]/sqlj/lib 的库 sqljutil.jar 和 utl_dbws_jserver.jar，以及 SYS.UTL_DBWS 数据包。您需要将 `${ORACLE_HOME}/sqlj/lib/sqljutil.jar` 和 `utl_dbws_jserver.jar` 加载到数据库中。确定是否加载了 sqljutil.jar 的一种方法为在 SYS 下描述程序“utl_dbws_validate”，将可以获得如下响应：

```
SQL> conn / as sysdba
Connected.
SQL> describe utl_dbws_validate;
PROCEDURE utl_dbws_validate
```

要将 jar 文件加载到文件，执行以下步骤。确保数据库在 java_pool_size 和 shared_pool_size 方面分别至少拥有 96M 和 80M。如果没有，修改用于两个目录的数据库简档，例如 `init.ora`，然后重启数据库。

运行脚本 SYS 下的 `sql/initdbws.sql` 或 `sql/initdbws9.sql`：

- 对于 9i 数据库，运行 `initdbws9.sql`
SQL>conn / as sysdba
SQL>@initdbws9
- 对于 10g Beta 数据库，运行 `initdbws.sql`
SQL>conn / as sysdba
SQL>@initdbws
- 运行 SYS 下的脚本 `[ORACLE_HOME]/sqlj/lib/sqljutil.sql`，即
SQL> conn / as sysdba
SQL> @sqljutil.sql

如果您正使用 Java，您可能希望使用 JAX-RPC 动态调用接口 (DII) API 直接进行编程。在本示例中，您已将您的 Web 服务客户端代码放入服务端 JavaVM，并完成。或者，您可能希望通过特定的静态客户端代理代码调用外部 Web 服务。

第2步 生成和加载客户端代理与包装

JPublisher [10] 将生成客户端代理代码、编译、打包和将其加载到数据库的任务实现了自动化。例如，提供以下命令行就已足够了。

```
jpub -proxywsdl=URL_of_Web_Service_WSDL  
-user=username/password
```

此外，如果这与 WSDL 提供的不同，您可以指定 `-endpoint=external_Web_Services_URL`，如果不同于通过 OCI 驱动程序可以访问的缺省数据库例程，则指定 `-url=JDBC_database_URL`。

```
% jar xvf dist/javacallout.jar META-INF/HelloServiceEJB.wsdl
```

```
% jpub -proxywsdl=META-INF/HelloServiceEJB.wsdl -dir=genproxy  
-package=javacallout -user=scott/scott  
-endpoint=http://localhost:8888/javacallout/javacallout
```

JPublisher 将在子目录 `genproxy` 下生成静态 Java 代理类和 PL/SQL 脚本，并自动将下划线文件加载到数据库。

- `genproxy/javacallout`: 本目录下的 Java 类由 JPublisher 调用的 `wscmpile` 工具生成。
- `genproxy/HelloServiceEJBJPub.java`: `HelloServiceEJBJPub.java` 类定义了 `sayHello` 方法，作为一种静态 Java 方法，以通过 PL/SQL 包装进行调用。
- `genproxy/plsql_wrapper.sql`: 面向 Web 服务客户端的 PL/SQL 包装 (即 `HelloServiceEJBJPub.java`.)
- `genproxy/plsql_dropper.sql`: 移除 `plsql_wrapper.sql` 定义的 PL/SQL 类型和数据包。
- `genproxy/plsql_grant.sql`: 当在 SYS 下运行时，赋予 Java 客户端代理必要的优先级，以调用 Web 服务。
- `genproxy/plsql_revoke.sql`: 取消 `plsql_grant.sql` 赋予的优先级。
- `genproxy/jpub_proxyload.log`: 加载这些生成的 Java 文件和安装 `plsql_wrapper.sql` 脚本的日志文件。

为了避免 JPublisher 将生成的代码加载到数据库，您可以指定 `-proxyopts=noload`。使用静态客户端代理的优势是您可以在您的 Java 代码中简单参考端口类型例程方法，而无需担心如何对 SOAP 消息的单独变量进行打包或解包。

第3步 授予优先级

脚本 `plsql_grant.sql` 面向 SYS, 用于赋予 SCOTT 必要的优先级, 以执行已加载的客户端代理。

```
SQL> conn / as sysdba
SQL> @genproxy/plsql_grant.sql
```

第4步 调用外部 Web 服务

要调用数据库中的外部 Web 服务, 声明和运行 `sql/run-plsql-proxy.sql` 脚本或运行以下的 PL/SQL 数据块。

```
SQL> set serveroutput on
SQL> declare
x varchar2(100);
begin
x:=JPUB_PLSQL_WRAPPER.sayHello('Hello from database');
dbms_output.put_line(x);
end;
/
```

第5步 清除

以下措施将允许您返回到之前的状态。

移除 PL/SQL 包装, 使用 SQLPLUS 撤销授予的优先级。

```
SQL> conn scott/tiger
SQL> @genproxy/plsql_dropper.sql
SQL> conn / as sysdba
SQL> @genproxy/plsql_revoke.sql
```

使用 `dropjava` 工具撤销数据库中已加载的 Java 类。

```
dropjava -u scott/tiger genproxy/wsdGenerated.jar
```

数据库作为 Web 服务提供程序

Oracle 数据库提供了一系列资源和能力, 从 SQL 到 PL/SQL, 以及 Java 存储过程、XML 数据库 (XDB) 的 XML 能力、和高级队列 (AQ) 和 Streams。为了向 Web 服务展现这些资源, 我们生成了相应的 Java 包装, 并将这些包装部署于 Oracle 应用服务器 10g 之中, 从而提供了一个带有 J2EE Web 服务框架的标准、可扩展的 J2EE 容器。

建立 OC4J

正如所述, 我们利用了 Oracle 应用服务器的 J2EE、Web 服务堆栈 和 OC4J, 即使一个独立的 OC4J 就已足够。如果数据库没有本地运行, 修改 `J2EE_HOME/data-sources.xml` 文件, 以便数据源 "jdbc/OracleDS" 指向运行数据库。例如,

```

<data-source
class="com.evermind.sql.DriverManagerDataSource"
name="OracleDS"
location="jdbc/OracleCoreDS"
xa-location="jdbc/xa/OracleXADS"
ejb-location="jdbc/OracleDS"
connection-driver="oracle.jdbc.driver.OracleDriver"
username="scott"
password="tiger"
url="jdbc:oracle:thin:@<host>:<port>:<sid>"
inactivity-timeout="30"
/>

```

转至 J2EE_HOME 目录，单独启动 OC4J 例程。

```
% java -jar oc4j.jar
```

Web Services Assembler

Oracle 应用服务器允许您使用 J2EE 组件创建支持 J2EE 的 Web 服务，以及在 Java 类和数据库资源上创建，包括 PL/SQL 数据包、SQL 查询、DML 声明和 Java 存储过程。10g Oracle 应用服务器的 Web Services Assembler 工具用于创建 Web 服务，作为部署于 OC4J 之上的兼容 J2EE 的 EAR 文件。它直接调用 JPublisher 程序来生成与各种 Web 服务对应的 Java 代码。

Web Services Assembler 配置文件

Web Service Assembler 的输入通过基于 XML 的配置文件提供：service-config.xml 和 client-config.xml。

Service Configuration File

无论合适您希望展现数据库资源，您将需要提供配置文件中的如下信息。

```
<db-port>
```

```
<!-- The uri to be used within the servlet context -->
```

```
<uri>/statelessSP </uri>
```

```
<!-- The database schema for which to generate a Web Service -->
```

```
<schema>scott/tiger</schema>
```

```
<!-- The database connection string to generate a Web Service -->
```

```
<db-conn>jdbc:oracle:thin:@localhost:5521:sqlj</db-conn>
```

```
<!-- The datasource used during Web Service runtime -->
```

```
<datasource-location>jdbc/OracleDS</datasource-location>
```

```
<jpub-input>
jpub.omit_schema_names
jpub.tostring=true
</jpub-input>
```

The database resource that is published as a Web Service ...

```
</db-port>
```

客户端配置文件

与 Web 服务配置文件类似，客户端配置文件被用于生成 Java 客户株（client stub）。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services>
<web-service>
<wsdl-input url="http://localhost:8888/query/query?WSDL"/>
<jaxrpc-mapping-file>mapping.xml</jaxrpc-mapping-file>
    <source-output-dir>build/src/client</source-output-dir>
    <proxy-gen>

        <package-name>oracle.demo.db.query.stub</package-name>
        <output>build/classes/client</output>
    </proxy-gen>
</web-service>
</web-services>
```

构建 SQL Query 和 SQL DML Web 服务

动机

空间是此处的限制。SQL 为获取、更新和向数据库存储任意数据提供了无限的可能性，包括关系、文本、空间、多媒体和 XML 数据。考虑亚马逊 Web 服务（www.amazon.com/webservices）或 Google Web 服务（www.google.com/apis/index.html）。这些能力使得客户端应用可以使用标准 Web 服务协议（WSDL 和 SOAP）发现并与其目录或搜索引擎进行交互。您应仅在优势超出 SOAP 时使用 SQL 声明实施 coarse-grain 服务。例如，SQL 数据库 Web 服务将允许客户端应用和轻型 SOAP 支持的应用查询目录，或根据 ZIP 代码检索位置地图。

第 1-a 步 汇集查询 Web 服务

要创建一个基于 SQL 查询声明的 Web 服务，您需要将 SQL 代码放入配置文件。以下 service-config.xml 片段描述了问题和数据库信息中的查询。

```

<output>./build/query.ear</output>

<uri>/query</uri>
<datasource-location>jdbc/OracleDS</datasource-location>
<schema>scott/tiger</schema>
<port-name></port-name>
<prefix>oracle.demo.db.query</prefix>
<db-conn>jdbc:oracle:thin:@OW-pc.us.oracle.com:1521:odb</db-conn>

<sql-statement>
<operation>
<!-- Name for the Web Service operation -->
<name>getEmp</name>
<!-- SQL code for a query -->
<statement>select ename from emp
                where ename={myname
VARCHAR}</statement>
</operation>

</sql-statement>

```

第 1-b 步 汇集 DML Web 服务

与 SQL Query Web 服务相同的是,要基于 SQL DML 声明 (INSERT、UPDATE 和 DELETE) 创建 Web 服务,您需要将 SQL 代码放入服务配置文件。以下 `service-config.xml` 片段描述了有问题的 DML 操作。

```

<operation>
<name>updateEmp</name>
<statement>update emp SET sal=sal+500
                where ename={myname
VARCHAR}</statement>
</operation>

```

Web 服务,尤其是数据库 Web 服务,目前通常没有状态,生成的 JPublisher 代码必须自动进行一次成功的 DML 操作,或在发生故障时将其返回。

多个 DML 操作的批处理执行可在 `service-config.xml` 中指定。单个或多个操作返回成功或失败的行数量。

所有查询和 DML web 服务操作可在相同的服务配置文件中指定。

```

<sql-statement>
<operation>

```

```

<!-- Name for the Web Service operation -->
<name>getEmp</name>
<!-- SQL code for a query -->
<statement>select ename from emp
            where ename={myname
VARCHAR}</statement>
</operation>
<operation>
<name>updateEmp</name>
<statement>update emp SET sal=sal+500
            where  ename={myname
VARCHAR}</statement>
</operation>
</sql-statement>

```

第 2 步 生成 Web 服务 ear 文件

以下命令将生成一个包含与数据库操作对应的 Java 包装的 ear 文件。

```
%java-jar<OC4J_HOME>/webservices/lib/wsa.jar -config
service-config.xml
```

以下文件将生成。

- dist/query.ear, webservices 应用
- build/src, 容纳客户端代理源代码的子目录
- build/classes, 容纳客户端代理 .class 文件的子目录。

注意: SQL 声明中的 bind 变量 {myname VARCHAR} 将转换为相应的 Web 服务操作总的字符串参数。注意查询以结果集合的形式返回, 它将形成结构行值阵列, 或不同的通用 XML 格式。

第 3 步 将生成的 .EAR 文件部署到 OC4J

以下命令将把上一步生成的 ear 文件部署到 OC4J。

```
% java -jar <OC4J_HOME>/j2ee/home/admin.jar
orimi://<hostname>:<port> admin <admin-password> -deploy -file
build/query.ear -deploymentName query
```

hostname: 是 OC4J 运行的主机

port 是 RMI 端口

第 4 步 捆绑已部署的应用

OC4J 拥有 http-web-site 的概念, servlets 和 web 应用围绕这一概念进行集成。以下命令将应用与 http-web-site 捆绑起来。

```
% java -jar <OC4J_HOME>/j2ee/home/admin.jar  
ormi://<hostname>:<port> admin <admin-password> -bindWebApp  
query query-web http-web-site query
```

在这一阶段，Web 服务已被创建；然而，为了使用它或让其发挥作用，您需要一个客户端应用。基于 J2EE 的 Web 服务框架，诸如 Oracle Application Server 等，通过生成基于 WSDL 的客户端代理，简化了 Web 服务编程。这一代理被客户端应用使用。

第 5 步 生成客户端代理

客户端配置文件

与 Web 服务配置文件类似，客户端配置文件被用来生成 Java 客户端。

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-services>  
<web-service>  
<wsdl-input url="http://localhost:8888/query/query?WSDL"/>  
<jaxrpc-mapping-file>mapping.xml</jaxrpc-mapping-file>  
    <source-output-dir>build/src/client</source-output-dir>  
    <proxy-gen>  
  
        <package-name>oracle.demo.db.query.stub</package-name>  
        <output>build/classes/client</output>  
    </proxy-gen>  
    </web-service>  
</web-services>
```

以下 WebServices 汇编器命令，用于在子目录 build/classes/client 中生成客户端代理

```
% java -jar $ORACLE_HOME/webservices/wsa.jar -config  
client-config.xml
```

第 6 步 编译和运行客户端应用

在 src/client/oracle/demo/db/query/QueryClient.java 中编译和运行范例客户端程序

using ANT commands:

```
% ant compile-client
```

```
% ant run-client
```

这一客户端程序使用在第 4 步中生成的客户端代理，来调用 WebServices 操作。目标“run-client”将提供以下输出：

```
[java] *** Query Emp Rows by ID returns 1 rows  
[java] *** Query Emp Rows by ID returns <7900,JAMES>
```

第 7 步 通过Web 浏览器访问服务

为测试目的，服务也可以通过浏览器<http://localhost:8888/query/query> 进行访问

- [getEmpBeans](#) 服务根据姓名返回员工相关信息。输入 “SCOTT”，按下调用察看返回的信息。
- [getEmpByNoBeans](#) 服务根据员工 ID 返回员工相关信息。输入 “7900”，按下调用察看返回的信息。
- [getEmpBySalBeans](#) 服务根据指定金额的薪资返回员工相关信息。输入 “3500”，按下调用察看返回的信息。
- [getEmpCountBeans](#) 服务返回表中记录的员工数量。请调用。根据 emp 表的技术，返回信息将显示 14。

构建 PL/SQL 或 Java Web 服务

动机

存储过程为重要的数据库变成模型，允许将运行于数据库中，以数据为中心的逻辑与运行于中间层的业务逻辑隔离开来。存储过程可以使用特定数据库语言，如 PL/SQL 进行编程，或使用独立于数据库之上的语言，如 Java 进行编程。ANSI SQLJ 第 1 部分指定了从 SQL 中调用静态 Java 方法的能力，作为程序或函数。Web 服务的优势在于您可以充分利用您在存储过程领域的投资，并将其展现为 Web 服务，而无需考虑存储过程采用的语言。

第 1-A 步 集合 PL/SQL Web 服务

要使 PL/SQL 数据库成为 Web 服务，我们需要待发布的数据库数据包的名称。或者，我们可以提供一个方法名称列表，并仅包含其函数和程序的子集。

```
<plsql-package>
<!-- The database package to be exposed. -->
<name>Company</name>
<!-- A list of methods to expose (optional). -->
<method>method1</method>
<method>method2</method>
</plsql-package>
```

第 1-B 步 集合 DB-Java Web 服务

通常您将您的 Web 服务构建于运行于 Application Server 中间层中的 Java 类之上。如果您希望将您的服务构建于运行于 Oracle 数据库 JavaVM 中的 Java 代码之上，该采取何种措施呢？虽然您可以通过 PL/SQL 调用规范展现您的代码，并将其作为 PL/SQL Web 服务进行发布，但直接将 Java 类作为 Web 服务更加方便。在本例中，您可以在您的 Web Services Assembler 配置文件指定一个类似于数据库资源的标签。

```
<db-java>
<!-- Server-side Java class to be exposed -->
```

```
<name>foo.bar.Baz</name>
<!-- List of methods to be exposed (optional) -->
<method>method1</method>
<method>method2</method>
</db-java>
```

Jpublisher, 作为 Web Services Assembler, 仅能发布满足以下标准的 Java 方法。它们必须为公共静态方法。参数或异常事件中的 Java 类型必须串行化。此外, 所有这些类型必须存在, 并在客户端和服务端 Java 环境中以相同的方法定义。充分利用本地 Java 调用的优势可以简化 Java 阵列和 Java Bean 类型在方法签名中的使用——无需提供 PL/SQL 呼叫规范, 或使用特殊 SQL 对象或 VARRAY 类型来展示此类值。

第 2-7 步

与 SQL query Web 服务相同。

结论

在本文中, 我们向 Java 开发人员全方位地介绍了如何使用 Java、JDBC 和 Web 服务进行编程。读者将意识到它们能够最大限度地利用其在 Oracle 数据库 10g 方面的投资。JDBC、Java VM 运行时、HTTP 调用、RMI 调用、EJB 调用和 Web 服务调用随数据库免费提供, 从而可以节省大量成本。

参考资料

- [1] 《数据库 Web 服务, Oracle 白皮书 (*Database Web Services, An Oracle White Paper*) 》, 2002 年 11 月。网址为:
http://otn.oracle.com/tech/webservices/htdocs/dbwebservices/Database_Web_Services.pdf
- [2] 《Web 服务期刊专题文章: Web 服务支持您的数据库 (*Web Services Journal Feature: Web Services Enable Your Database*) 》; Kuassi Mensah; Web Services Journal, 第 4 期第 3 卷。网址为:
<http://www.sys-con.com/webservices/articleprint.cfm?id=515>
- [3] W3C XML Schema Specification.
<http://www.w3.org/TR/xmlschema-1/>
- [4] 《W3C 建议: SOAP 1.2 版 (W3C Recommendation: SOAP Version 1.2) 》。<http://www.w3.org/TR/soap12-part1/>
- [5] 《W3C 注: Web 服务描述语言 (WSDL) 1.1 (W3C Note: Web Services Description Language (WSDL) 1.1) 》。
<http://www.w3.org/TR/wsdl>
- [6] Oasis UDDI 3.0 版。http://uddi.org/pubs/uddi_v3.htm

- [7] 《GGF: 开放式网格服务基础设施 (GGF: Open Grid Services Infrastructure)》。
<http://www.gridforum.org/documents/GFD/GFD-R-P.15.pdf>
- [8] JAX-RPC 规范。<http://java.sun.com/xml/jaxrpc/>
- [9] 《技术专有知识: 加载 SOAP 库 (Technical How To: Loading SOAP Libraries)》。
http://otn.oracle.com/sample_code/tech/java/jsp/loadjars.html
- [10] Oracle9i Jpublisher。
http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/java.920/a96658/toc.htm
- [11] 《技术专有知识: 使用表函数 (Technical How To: Using Table Functions)》。
http://otn.oracle.com/sample_code/tech/java/jsp/dbwebservices.html
- [12] 《JSR 114: JDBC Rowset 实施 (JSR 114: JDBC Rowset Implementations)》。<http://www.jcp.org/en/jsr/detail?id=114>
- [13] INCITS H2.3 任务组。<http://www.sqlx.org>
- [14] Oracle XML SQL Utility – XSU。
http://otn.oracle.com/tech/xml/oracle_xsu/content.html
- [15] OTN Web 服务中心。<http://otn.oracle.com/tech/webservices>
- [16] 数据库 Web 服务。
<http://otn.oracle.com/tech/webservices/database.html>

ORACLE

Oracle 数据库 10g, Java, JDBC and Web 服务
2004年6月
作者: Kuassi Mensah
Contributing Authors: Ekkehard Rohwedder

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

全球咨询:
电话: +1.650.506.7000
传真: +1.650.506.7200
www.oracle.com

版权所有 © 2003 Oracle 公司。保留所有权利。
本文仅用于提供信息，此处所含信息可随时变更，恕不另行通知。
本文并不保证没有错误，也不作任意明确或隐含的担保或声明，
包括对于适销性或适用于特定用途的隐含担保。我们对本文不承
担任何责任，且本文不代表任意直接或间接的合约责任。未经事
先书面许可，不得以电子或物理方式复制或传播本文内容。

Oracle 是 Oracle 公司和/或其子公司的注册商标。其它名称
为其各自所有者的商标。