db4o | 开放源代码的面向对象数据库 | Java and .NET

意想不到的数据库

作者 Rick Grehan 译 韩巍

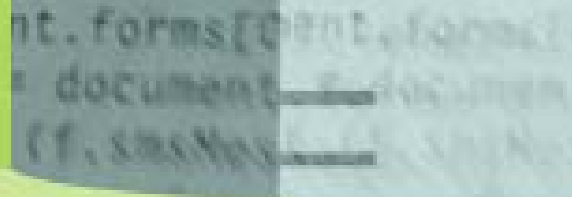
Rick Grehan 是 Compuware/Numega 实验室在 Java 和.NET 领域的一位 QA 工程师。
他也是 infoWorld 杂志一位有贡献的编辑.他的工作范围是嵌入式系统编程，EDN，微处理器报告，计算机设计 。
在来 Compuware 之前，Rick 在 Metrowerks 有限公司的 Discover DSP Project 工作 。
早期， Rick 是 BYTE 杂志的一位高级编辑，作为实验室主管，和 BYTE 的 JavaTalk 栏目作者。

移动设备，信息装置，智能控制系统。不需要深入的调查就可以揭示计算机周边应用程序的传播。这些系统通常被称为“嵌入式系统”，但这个称呼并不能表达这些应用不断增长的复杂性。

处理器时钟周期提高，内存密度增长，硬盘尺寸缩小，价格下降，系统需要运行的任务越来越复杂。和从前相比嵌入式系统被要求用来做更多的事情——变得比以前更加智能。用另外一种确切的方式表达就是嵌入式系统需要能存储，取得和操作越来越多的数据；这些情况的一个回应做法，嵌入式系统的用户期待在处理器性能方面有一个全球性增长,和在通讯的吞吐能力方面有所增加。

从另一方面看，智能设备的智能化不紧紧依赖于它所执行的算法，还依赖于它输入这些算法的数据。所以，由于设备复杂度的增长，随之而来的是对于相对复杂数据的组织、存入、取出方面软件的增长，以满足置入嵌入式应用程序的需求。

免费下载和评估 db4o 请浏览 www.db4o.com



需要什么？

对于我们需要什么这个问题，答案相当简单：“一个小的，快速的，强力并且易于使用的数据库”。

这么说有些太简要了，而且从答案本身告诉我们的也很少，我们需要更进一步的检验一下：

- **最小限度的资源消耗** 虽然现时科技的进步允许系统设计者购买更多的内存，而优化应用程序的内存消耗却永远不会过时。这种权衡很简单：由于我们的数据库消耗更少的内存，所以会有更多的内存供其他应用程序组件使用，并且设计者可以添加更多的特性（或者改进现有程序的性能）。

  注意我们上面用到的“优化”一词，我们不是说简单的减小数据库本身的大小－－这能通过减少特性的方式很容易的做到。但是，为了容量而牺牲性能是危险的：你当然可以开发一个更小的产品，但同时它也缺少了很多实用性。

- **高输入输出性能** 这种需求是不言而喻的。只有非常有限的应用能忍受低速的数据库访问。换句话说，我们还不知道有人会抱怨数据库返回记录太快了

- **易于实现** 尽管在开发环境上的所有改进，尽管在面向对象编程方面的提升，框架的内置库仍然充斥着预先写好的算法和数据结构，开发者仍然必须设计和实现应用程序的业务逻辑－－一件琐碎繁重、困难的事情。由此，一个数据库的内置库不应该表现出一种学习上的挑战，引用一种更精炼的表达，就是：它应该“尽可能的简单，但不简陋”，这创造了一条最低限度的学习曲线；开发者能集中于应用程序本身，而不是学习数据库的内置库技巧。

  另外，把数据库的内置库合并入正在开发的项目应该是一个统一的过程。理想的情况，内置库应该是一个单独的文件（对于 JAVA，后缀为.JAR 的文件；对于.NET,后缀为.dll 的文件）。基于命令行的工程，添加内置库应该包括一行可改变的脚本；对于使用集成开发环境的工程，这种添加应该是一种拖拽操作。

- **便移植的** 一款数据库的内置库应该是“平台无关”的，以最大化它的运行时目标－－从而最大化应用程序的潜在用户群。广泛的便移植性给程序员在主流操作系统平台开发

免费下载和评估 db4o 请浏览 www.db4o.com

带来了很多豪华特性，这样充分利用艺术级开发和除错工具，可以确保平台无关性的结果。

- **可靠的** 可靠性是另外一个公认的需求——并且可能是最重要的需求。一款不可靠的数据库产品简单的说就是无用的产品。对于大多数嵌入式应用程序，尤其是那些应用在实时系统中的，可靠性是所有组件属性的不二法则。

此外，数据库必须符合工业认可的标准。数据库必须明确的实现 ACID 特性（见右侧注释）

ACID

ACID 是一款数据库在被认为是有用的数据库系统之前必须遵守的四种特性的英文字母的缩写， 它们分别是：

原子性 – 数据库的事务处理组件必须以要么全有要么全无的方式执行.例如，如果一个数据库的事务处理包括删除 4 个对象,虽然它们各自是独立的对象但也要一起被删除.三个被删除了,但是最后一个有些原因没有被删除是不允许的

一致性 – 数据库上的操作从一个定义好的状态到下一个状态,中间状态是不可见. 例如，如果用户向数据库里添加一个对象A,然后对用户来说好像没有办法得到一个部分的对象 A .数据库不应该出现一种操作半完成的状态.

独立性 – 数据库的多事务处理彼此间是无法察觉的. 所以如果两个用户想同时修改同一个对象，数据库必须有一种机制使他们可以顺序访问这个对象,以便于既妨碍不到用户工作,也让其他用户看不到对方

耐久性 – 一旦一个事务处理应用到数据库,所做的工作便不会丢失,即使遇到硬件或者软件错误. 所以在数据库上执行一个删除 3 个对象的操作,在执行到删除第二个对象的时候系统崩溃了,当系统重启后,数据库不仅可以自我恢复,连未完成的事务,也将得到恢复

相关可能性

关系型数据库是众所周知的，并且很可能是使用最广泛的数据库存储范例。它的流行使关系型数据库成了任何数据库应用程序的一个适当选择。然而，尽管这些优势，整合一个关系型数据库和面向对象的应用程序却不是很流畅的。关于在面向对象环境使用关系型数据库系统的解释经常提及一个词“阻抗失调”。这个术语借用自电子领域，用来表示在关系和面向对象模型中的不一致性

这种不一直源于关系型数据库管理系统（RDBMS）以表格中的行的形式存储信息这个事实。因此，把对象的内容存储进这样的数据库需要把对象分割开——拆成一个一个部分，把这些部分存入数据库表格的不同字段。想要重新得到这个对象，这些分离的部分需要聚集起来并重新装配。

免费下载和评估 db4o 请浏览 www.db4o.com

举例（java 语言）：智能自动售货机

通过举例便于阐明成果

假设我们已经设计出了一款自动售货机：一种跟踪零食的种类和数量，分配、监视在它的铁壳子里面可用的零钱，记录自己的销售额—也许还配备有连接 internet 的无线连接，当需要补充食品的时候可以向总公司发送邮件。（相信我，这不象听起拉那么遥远）

从 RDBMS 得到一个 Snack 对象需要如下的代码：

```
ResultSet results = statement.executeQuery("SELECT ID, Name, Cost," +
  " Retail, Supplier FROM Snack WHERE ID = " + searchID);
 if(results.next())
 {  snack = new Snack();
    snack.ID = results.getLong("ID");
    snack.name = results.getString("Name");
    snack.cost = results.getLong("Cost");
    snack.retail = results.getLong("Retail");
    snack.manufacturer = result.getString("Supplier");
    //  ... Do something with the Snack object ...
 }
}
```

列表 1 从 RDBMS 取得一个 snack 对象的数据.这部分代码得到 snack 对象的 ID 属性为 searchID 的所有信息

查询的结果首先存入 results 记录集。然后执行操作拷贝各个独立字段的内容从 results 对象到 Snack 对象,另外，每个字段的执行操作还包括数据库中所存储值的格式转换为 JAVA 语言的内部表现形式（虽然这种转换发生在特殊的方法内，对于那些大且复杂的对象结构你也可以想象的到这些代码将会是什么样子）

注意这个过程也是以执行字符串形式表示的 SQL 声明为开始的，如果这个声明没有预先编译，还要消耗更多的 CPU 时钟周期来解析和执行 SQL 语句来完成后端数据库的实际查询。

免费下载和评估 db4o 请浏览 www.db4o.com

“阻抗失调”现象在对象/关系型数据库系统中有一些减弱。一款对象/关系型数据库系统（ORDBMS）隐式的处理对象和关系数据的转换。（关系型数据库仍然存在，它只是在后端被隐藏了起来）。 结果，如上所示的代码分解和重新装配便不再需要。

但是，即使这些代码是含糊不清的，但它们还是存在。它典型地隐藏在一个“映射层”、类的集合和对象关系型数据库内置库的方法内。这个映射层完成面向对象应用程序和关系型数据库内表格的行、列之间的转换

这样，当程序员可以以对象方式操作对象时，因此只需要移植少量的代码，CPU 时钟周期仍然被在关系型数据库和对象数据间转换所消耗。并且，在数据库内置库的某个位置，SQL 必须执行确切的行数从存储行转换到数据库表格。

总而言之，一款 RDBMS 在应用程序之外增加了空间和时间，空间被对象到 RDBMS 的转换代码消耗；时间被执行这些代码消耗。一款 ORDBMS 做了一些改良，程序员至少从编写转化代码中解放了出来。但是，虽然隐藏了，这些转换代码仍然存在，占据内存容量和 CPU 时钟周期。

解决方案

上面引用问题的解决方案是使用纯面向对象数据库。Db4o（来自 db4objects 公司）作为一款独特的，实用的面向对象数据库，同时满足第一部分大纲所列举的需求,并且规避了在第二部分所描述的缺点。如果我们探究它是如何演绎我们刚才讨论的问题，Db4o 的实力就是最好的说明

- **资源的最少消耗** db4o 的内置库紧紧使用了大约 400K，正如你将看到的，它的节省内存特性并没有预示性能上的不足
- **高输入输出** db4o 的执行等同于最好的数据库系统,下面的基准测试结果显示 db4o 的性能表现与典型的 SQL 数据库的比较：

免费下载和评估 db4o 请浏览 www.db4o.com

barcelona benchmarks	read	write	query	delete
db4o/4.5.200	1.0	1.0	1.0	1.0
Hibernate / MySQL	20.8	32.2	6.7	17.3
Hibernate / HSQLDB	10.4	5.4	536.0	3.9
JDBC / MySQL	10.8	14.6	1.7	6.5
JDBC / HSQLDB	0.4	1.7	677.8	0.7
JDBC / Derby	3,696.0	12.9	1,299.7	7.1
JDO/VOA/MySQL	4.4	14.8	3.0	2.4

fastest
slowest

db4o 基准测试　上表显示 db4o 的性能同典型的 SQL 数据库在读、写、查询、删除操作方面的比较。复杂的操作包括在一个包含 5 层继承的树型结构对象上的操作

- **易于实现**　JAVA 版本的 db4o 是一个单独的 JAR 文件；.NET 版本的是一个单独的 DLL 文件。你可以在应用程序里引入这个库通过把它放置在你的 CLASSPATH 中（如果使用基于命令行的模式开发）或者把文件拖拽到你的工程里（如果使用 JAVA 或.NET 集成开发环境）

  db4o 的应用程序接口没有过于烦琐的描述、复杂的类和方法。举例：一位 db4o 程序开发人员主要使用的是 db4o 的 ObjectContainer 类（数据库本身的表示）。ObjectContainer 的接口只定义了 10 个方法；这 10 个方法提供了数据库的大部分操作——添加、查找和删除数据等。

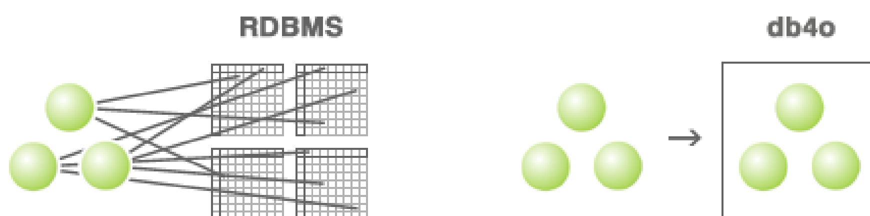
  所以，如果我们假定一个 ObjectContainer 叫做 vendingmachineDB，它已经被打开了，并且我们想存入一个 Snack 对象到这个数据库，JAVA 可以很简单的表示：

  ```
  vendingmachineDB.set(snack);
  ```

  这段代码还可以用来更新一个已经存储在数据库中的 Snack 对象。(注意：如果上面的代码在应用程序中是最后的数据库操作，还应该调用 ObjectContainer 的 Commit()和 close()方法以便数据库正确的执行结束操作。在 RDBMS 或者 ORDBMS 中类似的操作也是必须的)

免费下载和评估 db4o 请浏览 www.db4o.com

- **便移植的** 正如已经声明的，db4o 有 Java 和.NET 两种版本。Java 版能运行在所有从 Java 1.1.x 后的所有 Java 平台（包括 PersonalJava 和 J2ME CDC 配置）。.NET 版本兼容.NET 1.0 ，.NET 1.1 ，和未来的 CompactFramework.
  .NET 版可以使用所有的.NET 语言，而且还可以运行在开源的 Mono 框架下。
- **可靠的** 最后，db4o 支持所有的 ACID 特性。对于多用户同时访问一个 db4o 数据库做了适当的隔离，他们的操作被 db4o 内置库隐式的连续执行。事务处理由 ObjectContainer 类的 commit()方法和 rollback()方法完成。假使在数据库更新期间产生一个冲突，这时 db4o 的 ObjectContainer 会重新打开，任何被中断会正确的完成
- **无“阻抗失调”**
  db4o 是一款完全面向对象的数据库。对象以“as-is”关系存储；没有对象到关系的转换层，也不含糊或者不可见。另外，db4o 能任意处理复杂的对象结构。你不用建立对象到关系数据库表格映射的模式定义。你的应用程序类的层次和类本身的关系定义了数据库的模式：



无“阻抗失调” 你的应用程序类的层次和类本身的关系定义了数据库的模式

使用 db4o 的简单易用可以通过一系列例子最好的体现出来。回到刚才的自动售货机数据库例子（上面用 SQL 所展示），假设我们在 db4o 里建立了一个一样的数据库。打开并添加一个新的 Snack 记录的代码如下：

```
// Open an ObjectContainer
// (openFile creates it if it does not exist)
ObjectContainer    vendingmachingDB    =
Db4o.opernFile("vmachine.YAP");
```


免费下载和评估 db4o 请浏览 www.db4o.com

```
// Create a new Snack object and populate
// its fields.
// Constructor fields are:
//  ID code
//  Product name
//  Cost in pennies
//  Retail in pennies
//  Supplier's Name
snack = new Snack(100,
  "Cheeze Zaps",
  1500,  // Cost is 0.15
  5000,  // Retail is 0.50
  "Sooper Cheeze Inc.");

// Put the snack into the database
vendingmachineDB.set(snack);

// A transaction is automatically started when
//  the ObjectContainer is opened. Before closing,
//  we should commit() the transaction that included
//  the set() operation
vendingmachineDB.commit();
vendingmachineDB.close();
```

列表 2 存储一个 Snack 对象到 db4o 数据库

所以，存储一个对象只要简单的调用 ObjectContainer 的 set()方法，传递一个要存储的对象的引用到这个方法。Commit()方法确保从数据库打开后的任何更改（或者从上一个 commit()方法后开始）可以写入数据库。（它还确保如果遇到一个系统错误的话操作不会丢失）

注意这种简单性。程序员不需要操作成员对象来获得数据库内的数据。对象被很好的处理。程序员所要做的所有事情就是告诉 db4o：“请把这个对象存入数据库”。Db4o 隐式的解决具体细节。

免费下载和评估 db4o 请浏览 www.db4o.com

得到 Snack 对象也一样的简单。Db4o 用户使用 Query By Example（QBE）技术来定位数据库对象。我们为 db4o 提供了一个“模板”对象，db4o 用户可以用来定位要查找的目标。这个模板对象同要查找的目标对象是相同的，用指定搜索条件的数据填充所有元素。

假设我们的 vendingmachineDB ObjectContainer 已经被打开，下面的代码得到刚刚输入的 Snack：

```
// Create a template object.
// Retrieve the snack by ID number.
// Other fields are null or 0, and will be
//  ignored by the query
snackTemplate = new Snack(100,null,0,0,null);


// Issue the query
ObjectSet result =
  vendingmachineDB.get(snackTemplate);


// Fetch the retrieved snack
if(result.hasNext())
{  Snack snack = (Snack)result.next();
   // Do something with the snack.
   . . .
}
```

列表 **3.** 从一个 db4o 数据库得到一个 Snack 对象

与给定的 RDBMS 示例相比，使用 db4o，Snack 对象可以完整地得到；程序员不需要编写一系列赋值声明来组装对象字段。(这是一种真正的无需考虑要得到对象的字段数量的方式)。也不需要 SQL 命令符传递给 executeQuery()方法来解析和处理。

删除一个对象也是相当直截了当的。一旦一个物体已经从数据库里获得，你只要简单的传递它的引用到 ObjectContainer 的 delete（）方法。代码如下：

免费下载和评估 db4o 请浏览 www.db4o.com

```
// Create a template object
// This time, query the snack by name
snackTemplate = new Snack(
  0, "Cheeze Zaps", 0, 0, null);

// Issue the query
ObjectSet result =
  vendingmachineDB.get(snackTemplate);

// Get the retrieved object
if(result.hasNext())
{ Snack snack = (Snack)result.next());
  // Delete the snack
  vendingmachineDB.delete(snack);
}
```

列表 **4.** 从数据库里删除一个 Snack 对象

再一次，对于对象的操作是整体的。对象被作为一个不可见的整体。

有一点可能没有引起读者的注意，给定的这个 db4o 的例子已经包含了单独的 Snack 对象。

我们给 Snack 对象定义如下:

```
public class Snack {
  private int id;
  private String name;
  private long cost;
  private long retail;
  private String supplier;
  // Constructors and accessors follow
  ...
}
```

列表 5. Snack 类

免费下载和评估 db4o 请浏览 www.db4o.com

Snack 对象的第二个和最后一个属性是字符串对象。但是，Java 字符串作为对象实现，而不是作为原始的字符串。所以，存储一个 Snack 对象同时存储了两个字符串对象，删除一个 Snack 对象也同时删除了与它的名字和供给者关联的字符串对象。在后台 db4o 执行这些存储和删除不需要我们的请求。（正如所表现出来的，db4o 处理字符串对象和其他简单的类型作为第二类对象；它们没有自己本身的一致性，删除和更新与它们的父对象相关联）

然而，db4o 可以简单的处理任意复杂的对象层次。假设我们深入自动售货机数据库的结构。我们定义一个类叫做 SnackSlot，在实际存放零食的自动售货机的众多投币口中的一个作为模型。（每个投币口可以持有一个给定 Snack 对象的多个实例），这个类如下：

```
public class SnackSlot {
  int slotnum;      // Slot number
  Snack thisSnack;  // Snack in this slot
  int original;     // Original number stocked
  int current;      // Number of snacks left
  // Constructors and accessors follow
...
}
```

列表 **6**.SnackSlot 类

这样，一个 SnackSlot 持有一个由投币口分配的 Snack 对象的引用，我们也会追踪一个给定的投币口的初始数量,还有零食的剩余数量。这可以让一台自动售货机决定一个指定的投币口已经卖出去了多少零食。

假设我们扩大自动售货机的容量,添加新的零食和新的投币口，有人会想需要两个分开的 set()操作，而实际上不是这样的。

当一个对象第一次被添加到数据库，db4o 探测所有的对象引用，并且存储所有的对象

免费下载和评估 db4o 请浏览 www.db4o.com

引用到数据库，这一切全部自动完成。所以，代码同时添加一个新 SnackSlot 并且关联新的 Snack，如下：

```
// Create the Snack object
snack = new Snack(101,
  "Nacho Novas",
  1200,  // Cost is 0.12
  7500,  // Retail is 0.75
  "Sooper Cheeze Inc.");

// Create the SnackSlot object
snackslot = new SnackSlot(
  10,     // Slot number 10
  snack,  // Connect the snack to the slot
  10,     // 10 bags on this slot...
  10);    // ...none sold yet

// Put the new SnackSlot object in the database
//  The Snack is stored as well.
vendingmachineDB.set(snackslot);
```

列表 7. 添加更复杂的对象到数据库.

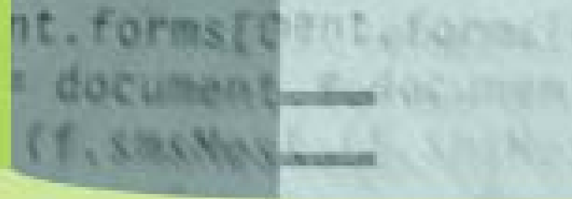
取出，删除 Snack 和 SnackSlot 对象有一点点复杂，但那只是因为 db4o 让您调整对象一旦在数据库中是如何管理的。

要得到一个混合的对象，当我们要取得一个对象时我们不得不告诉 db4o 在数据库的引用树型结构内要达到多大的深度（当调用 get()方法)。这被叫做“激活深度”。激活深度为 0 将只得到根对象，所以，如果我们取得一个 SnackSlot 对象的激活深度为 1，db4o 会取得 SnackSlot 和 Snack 对象。

因为 db4o 的默认激活深度被设置成 5，我们能得到一个 SnackSlot 和与它关联的 Snack 对象用在**列表** 3 里所示相同的代码。（接下来的部分我们将示范如何使用 db4o 的激活深度）

db4o 通过使用根对象同样能删除所有对象引用。然而，我们不得不明确得说明——当 db4o 删除一个对象，它也会同时删除它所引用的所有对象，不然的话，它将只删除根对象。

免费下载和评估 db4o 请浏览 www.db4o.com



这样，如果当我们想删除一个 SnackSlot 对象时，同时删除 Snack 对象，我们必须为 SnackSlot 对象关联一个“层叠删除”标志。我们需要在 db4o 的“全局配置对象”内做这些。代码如下：

```
Configuration config = Db4o.configure();
ObjectClass oc = config.objectClass("<package>.SnackSlot");
oc.cascadeOnDelete(true);
…
```

列表 8．设置层叠删除标志

这部分代码告诉 db4o 何时删除一个 SnackSlot 对象，然后所有的成员对象（在这个例子里，这些被 Snack 对象成员所引用的）也会被删除。把列表 8 的这部分代码附加在前面列表 4 的前面，会完全的删除 SnackSlot 对象和与它关联的 Snack 对象。

显然，层叠的删除在所有的案例里是不适用的（所以 db4o 使这个功能为可选的）在我们假想自动售货机例子里，出于一些原因删除 SnackSlot 对象是合理的，但是关联 Snack 是由不同的投币口提供的。在那样的情况下，我们不想使用层叠删除，因为我们想要 Snack 对象继续留在数据库内。

原生查询

db4o 的 QBE 查询的核心多少与“WHERE … IS EQUALS TO …”这样的在一条语句内的筛选相近。这对于在目标对象层次上定位根对象来说足够了。一旦根对象被放入内存，db4o 允许你使用普通对象引用来导航任何所需的子或同属对象。在 db4o 数据库内通过对象树本身导航贯穿对象树型结构是很自然的。这样的结果是强力的、简单的查询机制，能完全充分的满足一系列的查询需求。

然而，有很多次当需要更复杂的查询时；当所要选择的项需要从基于标准的持久化存储而不是同等的选择出来。你会想到更复杂的查询会需要一种更复杂的查询语法。对于很多数据库系统，确实是的，但对于 db4o 则不是。

免费下载和评估 db4o 请浏览 www.db4o.com

Db4o 的原生查询 API 很容易处理那些超越 QBE 性能的查询。而且，在保持 db4o 简单易用原则下，原生查询不需要开发者掌握一门独立的语言，不需要在迷宫似的 API 内徘徊。实际上，使用原生查询，开发者仅仅象使用应用程序的其他部分那样使用。

回想一下我们在**列表** 3 里的查询。在那段代码片断里，我们取出 ID 等于 1000 的 Snack 对象。相同的功能，用原生查询语言表示，代码如下：

```
List<Snack> snacks =
    vendingmachineDB.query<Snack>(new Predicate()
  {
    public boolean match(Snack snack)
    {  return( snack.id == 100); }
  })

if(snacks.hasNext())
{
     Snack snack = snacks.next();
     // Do something with snack
     ...
}
```

列表 9. 等价于列表 3 例子的原生语言查询.

在这里，我们不仅使用了 db4o 的原生查询 API，还使用了 Java 最近添加的 Generics 特性使我们的查询为完全类型安全的。(Generics 是 JDK5 添加的新特性，但是 db4o 的原生查询可以追溯到 Java1.1 版本来使用)

剥离开上面的查询代码，我们可以发现 db4o 原生查询 API 的特征。首先，原生查询定义 Predicate 类，作为当前查询引擎的实例化。在这个类内部，有一个独立的抽象方法 -- match() –扩展了必须实现的 Predicate 类。Match()方法带有一个单独的类参数。这个参数指定了数据库内要查询的目标类。在**列表** 9 里， Snack 类是要参与查询的目标类

Match()方法返回一个布尔值，很容易看出 match()方法的基本目的是筛选目标类。简

免费下载和评估 db4o 请浏览 www.db4o.com

单说，如果一个类满足搜索条件 match()方法返回真值；否则的话返回假。所以，当查询被执行时，数据库内的每个 Snack 对象都传递给 match()方法，并且使用返回的真值来判定是否一个给定的对象应该被返回给 List 集合。

原生查询有很多吸引人的地方，不只是应用程序本身的语言作为查询语言这个事实。举例推想，我们要统计哪种零食卖的最好（5 种或者更多）

使用原生查询 API，是很简单的，如下：

```
// Fetch list of snacks that have sold
//  5 items or more.
// The list is returned in
//  popularSnacks ArrayList

List<SnackSlot> slots =
   vendingmachineDB.query<SnackSlot>(new Predicate()
{
   public boolean match(SnackSlot snackslot)
   { if(snackslot.original == 0)
        return false;
     return((snackslot.original -
        snackslot.current) > 4);
   }
}
while(slots.hasNext())
{
     SnackSlot goodSlot = slots.next();

     // Read the snack object in, too
     vendingmachineDB.activate(goodSlot,2);
     popularSnacks.add(goodSlot.thisSnack);
}
... process popularSnacks...
```

列表 **10**. 复杂的 db4o 原生查询

免费下载和评估 db4o 请浏览 www.db4o.com

使用原生查询，我们可以在候选类的属性内执行数学运算并且决定哪个 SnackSlot 卖出 5 或者 5 个以上的零食。在 while 循环内，我们检查匹配项，然后使用 db4o 的 activate() 方法来读取 Snack 类到内存中。（如我们先前提示的，activate()方法告诉 db4o 从数据库内取出类成员到内存中，直到一个指定的“深度”。这样，深度为 2 时确定不只是 SnackSlot 类，连同 Snack 类也被读入到内存中）

查询被执行以后，应用程序可以处理 popularSnacks ArrayList，也许提醒自动售货机的所有者用不了多久就该添加商品了。

当然，我们可以使查询的对比部分任意复杂化。我们可以用 activate()方法读入其他成员类到内存中。原生查询之美是我们可以使用所需要的任意 java 代码来实现查询。（有一些次要的限制，所有这些需要避免不必要的副作用。你可以从 db4o 网站的在线文档得到更多相关信息）

最大的优点，我们的查询是完全类型安全的，完全可重构的。如果我们已经使用了象 SQL 这样基于字符串的查询语言，我们在对象属性引用内的任何错误（拼写错误，不正确的字段名等）直到运行时才会被捕获。使用原生查询，编译器可以保持这种一致。另外，如果我们继续开发我们的应用程序 ，当我们需要改变类的名字，类成员，属性等等时，我们完整的开发环境重新构建能力可以安全的使名字批量的改变。而在基于 SQL 的关系型系统，我们会被强制手动定位这些查询字符串的改变

表层描述

这紧紧是表层描述。而我们只是展示了 db4o 实际能力的一个很小的子集，我们至少证明了它满足一个小型的，可靠的，可响应的数据库需求。此外，我们也展示了 RDBMS 与 ORDBMS “阻抗失调”问题的解决方案在面向对象环境下一旦使用 db4o 的完全不适用性。Db4o 原生查询性能使复杂查询变得可以很容易的建立， 易于维护，回避很多开发中导致程序员使用 RDBMS 查询中所陷入混乱的很多缺陷。

还有很多我们仍然没有展示出来。

免费下载和评估 db4o 请浏览 www.db4o.com

- **对象版本改变**　例如：我们没有展示 db4o 在处理对象版本改变方面是多么简单。假使我们想要改变我们 Snack 类的结构，增加一个 SnackType 成员，允许我们区别块糖和散装糖。我们可以在已经建立的 db4o 数据库里应用这个改变并且可以重新组装旧的 Snack 类。我们可以无需中断在使用中的数据库应用。Db4o 允许我们使用一个单独的方法调用，来重新命名一个已经存储在数据库的类。（这样，旧的 Snack 对象可以被重命名为 OldSnack 对象）。对比之下，应用在关系型数据库后台的解决方案需要在数据库表格内成批的进行改变，还包括查询代码的改变。
- **简单的对象复制和同步**　软件智能的运行，但是间歇性的连接设备（比如：手持采集器或者扫描器）必须拥有从主数据库接受持久对象的子集的方法，允许用户应用程序工作在导出的数据库（从主数据库断开的）上，然后重新连接主数据库并且同步所做的改变。Db4o拥有这种正确的插入数据的能力。叫做db4o的复制系统(dRS)[2]
- **零管理**　另外，我们不得不说一下关于 db4o 得管理特性－db4o 实际上不需要管理，事实证明确实是这样。因此，对于处理移动设备应用程序，信息应用程序，智能医疗系统，所有对于用户不可见的数据库应用，它是理想的选择。
- **开放源代码并且遵循 GPL 协议**　我们把也许是最吸引人的 db4o 品质特性留在最后：所有 Java 和.Net 版本都是开放源代码的。如果你想把 db4o 集成进你下一版本的商业应用程序，低廉的所有权花费使 db4o 本身可以作为一个类来应用。

但是如果不是基于 db4o 的本质所有这些特性将一无是处：它是完全可依赖的，强力的，无废话的 Java 和.NET 版面向对象数据库。

[2] Db4o的复制系统的更多信息，请访问: http://www.db4o.com/about/productinformation/features/drs.aspx

免费下载和评估 db4o 请浏览 www.db4o.com