平台非依赖性建议及使用抽象隔离变化 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022__E5_B9_B3_E5_8F_B0_E9_9D_9E_E4_c104_145436.htm 要使程序具有不依赖于平台的特性，在设计的初始就必须考虑如何使程序不依赖于特定平台，并在开发的整个过程中不断对程序的平台依赖性进行审视。一． 前言：本文的目的是给出一些建议，这些建议可以帮助开发人员以比较小的代价，使程序运行在多个目标平台之上。如果程序只运行在特定的平台之上，则不需要使之具有平台非依赖性。但是：1.用户使用多种平台；2.技术平台本身不断升级；3.为满足性能或其他方面的要求，软件转移到另一个平台使得程序可能会在多个平台上运行，或者迁移到新的平台上，因此，不得不考虑这种迁移的代价。本文的受众是公司内部的全体开发人员，因而所有的环境都是针对公司目前的开发环境的。由于作者本身的限制，提出的建议不可能具有完善的体系，希望尽我所能对大家有所帮助，如有疏漏错误，还望补充指正。二． 依赖的平台：程序运行依赖于：硬件平台、操作系统平台、数据库平台、网络通讯平台、语言平台、java虚拟机平台。上述平台中，语言平台可能是其他平台的组成部分，但是在考虑平台依赖性时，单独的提出来。理论上，开发java程序只与java虚拟机平台有关，但是：1． 程序编制时不加注意，暴露了本来已经被抽象出的对象。例如：不使用File. separator，而使用特定于平台的文件分隔符“ \\ ”。这样，程序不仅依赖与虚拟机，还与特定的其他平台相关。2． Java虚拟机本身就不能对所有的平台进行抽象。比如，现在已知Sun JRE 和 Symantec JIT（Just

in Time Library）与Intel P4处理器不兼容，即jvm与硬件相关；java的线程使用操作系统提供的线程服务，即jvm与操作系统相关；JDBC接口由各个数据库厂商自己实现，通常都不会被完全支持，即java平台与数据库相关；java对字符的处理与本地语言和字符编码相关（java已经尽量减少了这种相关性，详细说明请见第六节），如果语言平台不是国际化的，那么java的国际化和本地化处理就会受到阻碍，即java平台与语言平台相关；我们的应用程序基本上只使用TCP/IP的有线网络，但是应用层的不同（例如http和https）使我们会依赖于网络通讯平台。第一种情况可以避免，但第二种情况是必须接受的现象，由于java平台本身就与其他平台相关，所以在它之上的应用程序就不可避免的与平台相关了。同时java平台的各个版本有许多不同，在java平台升级时不得不考虑这些不同点。在这里，特别提出数据库平台需要引起注意。ANSI先后发布了多个关系数据库的标准，众所周知的是SQL92，并于1999年再次更新成SQL99标准。虽然各个开发商都声明支持SQL标准，但是没有一个开发商实现了SQL标准中的全部特性。事实上，各个开发商的实现都与标准相差甚远，如果按照SQL92或SQL99来编写数据库程序，根本就无法在任何一个现实世界中存在的系统中运行。在这种参差不齐的基础上，sun制定了JDBC标准，各个厂商也纷纷宣布支持，但是每个厂商的手册中都会详细说明自己的实现与JDBC标准的差异，因此，你不可能指望在所有的数据库上使用JDBC的大多数特性。为了使程序具有较好的平台非依赖性，开发人员就必须对各个平台的特性和不同之处有所了解，并在开发过程中不断作出努力。三． 平台非依赖性的权衡：当程序从一个平

台转移到另一个平台时，程序所具有的平台非依赖性就会产生好处，但是在获得这种好处之前，必须为此付出代价。有时，程序的平台非依赖性会和程序要达到的其他要求产生矛盾。例如，你可能需要在数据库中执行这样一条语句：0select * from table1 where substr(field1, 2, 2) = ‘ aa ’但是并不是所有的数据库都会支持substr函数，有的数据库可能不支持自定义函数，因此，为了获得在不同数据库平台上的可移植性，你只能不使用函数，而只使用数据库的基本特性：0select * from table1 ，然后在程序中判断每条记录是否符合substr(field1, 2, 2) = ‘ aa ’的条件。但这样做将可能增加大量的IO，并使程序更加复杂。因此，你必须在程序的平台非依赖性和其他环境要求之间作出权衡。为此，需要先确定程序可能运行的目标和范围，在约束条件下考虑程序的平台非依赖性，并尽量考虑未来约束条件可能发生的变化。四． 处理变化：当你必须在程序中处理平台非依赖性问题的时候，事情归根结底就变成如何处理变化的问题。我的意思是：正是由于不同平台有所区别，才使得程序可能依赖于某个特定的平台，而要使程序能适应多种平台，你就必须处理不同平台之间的变化。Java平台已经处理了这些变化的大部分，这使得对于java应用程序而言，这些变化中的大部分都是不可见的。然而正如前面所展示的，即便如此，仍然有许多变化需要我们处理，并且只有付出努力，才能使应用程序利用到java平台的优点。1． 规避变化：处理变化的一个方法是避开它。如果你的程序代码不涉及到在不同平台上发生变化的部分，那么它就不会受到平台变化的影响。例如，你可能想编制一个系统管理程序，使用图形用户界面。该程序只有一个对话框，对话框中包含

两个按钮，一个是红色，一个是绿色。对话框上还有一行提示：按绿色按钮启动服务，按红色按钮停止服务。但是由于该程序属于后台的系统程序，可能运行该程序的计算机只配备单色显示器，这样的话用户无法正常的使用，因此这时最好的选择是不使用颜色而使用文字区分不同功能的两个按钮。另外，该程序还可能运行在只有字符输入设备的计算机上，为了不依赖于可变的输入设备（鼠标），最好将该程序编制成控制台形式：输入“1. run”启动服务，输入“2. stop”停止服务。通常，为了回避可能发生的变化，程序应当只使用通用的特性，即不同平台上都具有的一些相同的特性。例如，为了回避数据库平台的变化，就仅使用关系数据库和JDBC的基本能力，例如仅使用简单的语句，尽量不使用较复杂的连接、子查询、集合运算，以及存储过程、扩展函数、触发器等，只使用简单的ResultSet，而不使用可滚动、可更新的结果集。但是回避引起变化的部分，就会受到各种限制，通常在需要实现相对复杂的功能和相对高的性能时，就无法使用这种方式了。2. 隔离变化：现在，我们先回顾一下在前言中提出的本文的目的。隔离变化是把变化的部分和不变的部分隔离起来，这样，开发人员就仅需要在程序的一小部分里去适应变化，而大部分代码都不随平台变化而变化，从而降低了实现平台非依赖性的成本。隔离变化的方法是使用抽象。例如File.separator是对文件分隔符的抽象，在不同的平台上具有不同的值。在Windows操作系统上File.separator=”\\”；在UNIX操作系统上File.separator=”/”。这样，File.separator隔离了变化，实现了平台非依赖性。（File.separator是一个static final的类成员，作者认为对它的处

理属于编译器，因而是否需要将源代码在目标平台上再编译一遍是个问题，详细请参考附录。）再举一个例子，假如需要更新数据库中的一条记录，被更新的字段是日期类型，在SQL Server中，可以以字符串形式表示：insert into table1 (date1) values(2001-9-11 10:10:10)而在Oracle数据库中日期不能用字符串表示，因此，需要写成：insert into table1 (date1) values(to_date(2001-9-11 10:10:10, YYYY-MM-DD HH24:MI:SS))如果使用Statement.executeUpdate(String sql)，则必须处理这种变化，对每一种可能的数据库平台使用不同的SQL语句。但是如果我们把更新日期类型的数据库操作抽象成PreparedStatement. setDate，则可以隔离出这种变化，由各个数据库的JDBC驱动来处理变化，从而使程序变得不依赖于特定的数据库平台。下面以数据库的“ 0select top ”语句为例，说明如何在程序中使用抽象来隔离变化。在SQL Server中，0select语句可以使用top子句，但是top不是标准的SQL，不被其他数据库所支持，要在其他数据库中实现top子句的功能，就必须使用各个数据库独特的方式。因此，将0select top语句抽象出来：public class SelectCode{public static String getSelectCode(String 0select, int top){if (top return 0select.switch (DBType){case SQLServer:return SQLServerSelectCode.case Oracle:return OracleSelectCode.case MySQL:return MySQLSelectCode.}}}假如需要使用这样的SQL语句：0select top 10 * from table1无论使用哪一种数据库，都只需要这样调用：String s = SelectCode.getSelectCode(“ 0select * from table1 ”, 10).ResultSet rs = stmt.executeQuery(s). 