[19]中华人民共和国国家知识产权局

[51]Int. Cl[7]
G06F 9/44

[12]发明专利申请公开说明书

[21]申请号 00810846.3

[43]公开日 2002年8月14日

[11]公开号 CN 1364260A

[22]申请日 2000.7.13 [21]申请号 00810846.3
[30]优先权
[32]1999.7.28 [33]US [31]09/362,615
[86]国际申请 PCT/GB00/02680 2000.7.13
[87]国际公布 WO01/08006 英 2001.2.1
[85]进入国家阶段日期 2002.1.24
[71]申请人 国际商业机器公司
地址 美国纽约
[72]发明人 山本启一 余健雄
大卫·布鲁斯·基米尔
斯坦福德·路易斯·耶兹

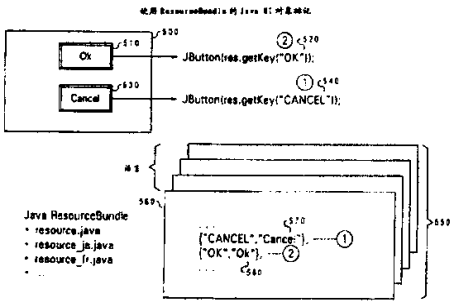
[74]专利代理机构 中国国际贸易促进委员会专利商标事务所
代理人 吴丽丽

权利要求书2页 说明书14页 附图页数9页

[54]发明名称 软件翻译的内省编辑器系统、程序和方法

[57]摘要

一种向语言翻译者提供要翻译文本的上下文关系信息的系统、方 法和程序。以基本语言向翻译者提供图形用户界面，随后翻译者可交 互式翻译屏幕上的各个文本标记。由于是在正确的上下文中对文本进 行翻译，因此可减少或消除翻译校验测试的时间和费用。通过向软件应用程序本身增加编辑器功能，获得在应用程序内编辑文本的能力。利用特定的资源束名称并依据关键字，可将应用程序中的各个文本标 记保存在本地化文件中。当激活编辑器时，内省文本对象，以查找其 源数据，翻译者可直接编辑文本。随后源数据被用于保存译文，供以后使用。



ISSN1008-4274

权 利 要 求 书

1. 一种编辑软件程序的文本数据的方法，包括：

通过调用包括与第一文本数据相关的源数据的软件对象，显示第一文本数据；

响应用户输入，检查软件对象，以识别源数据；

从用户输入接收对应于第一文本数据的第二文本数据；

按照源数据，以机器可读的形式保存第二文本数据；和

在计算机显示器上替代第一文本数据而显示第二文本数据。

2. 按照权利要求 1 所述的方法，其特征在于机器可读形式是计算机存储媒介上的本地化文件。

3. 按照权利要求 1 所述的方法，其特征在于第二文本数据是第一文本数据的译文。

4. 按照权利要求 1 所述的方法，其特征在于软件对象是 Java 组件。

5. 按照权利要求 1 所述的方法，其特征在于源数据包括对 Java 资源束的引用。

6. 按照权利要求 1 所述的方法，其特征在于源数据包括与第一文本数据相关的关键字。

7. 按照权利要求 1 所述的方法，其特征在于检查步骤中的用户输入是鼠标点击。

8. 具有至少一个处理器、可存取存储器及一个可访问显示器的计算机系统，所述计算机系统包括：

用于通过调用包括与第一文本数据相关的源数据的软件对象，显示第一文本数据的装置；

用于响应用户输入，检查软件对象，以识别源数据的装置；

用于从用户输入接收对应于第一文本数据的第二文本数据的装置；

用于按照源数据，以机器可读的形式保存第二文本数据的装置；

和

用于在计算机显示器上替代第一文本数据而显示第二文本数据的装置。

9. 具有在计算机可用媒介上的计算机可读程序代码的计算机程序产品，包括:

用于通过调用包括与第一文本数据相关的源数据的软件对象，显示第一文本数据的指令;

用于响应用户输入，检查软件对象，以识别源数据的指令;

用于从用户输入接收对应于第一文本数据的第二文本数据的指令;

用于按照源数据，以机器可读的形式保存第二文本数据的指令;和

用于在计算机显示器上替代第一文本数据而显示第二文本数据的指令。

02.01.24

说　　明　　书

软件翻译的内省编辑器系统、程序和方法

本发明涉及国际通用软件的开发工具，尤其涉及多语言软件开发。更具体地说，本发明涉及实现计算机软件的语言翻译的改进系统和方法。

随着计算机变得越来越普遍，软件开发商最好能够把他们的产品销售给不讲软件开发商本国语言的那些人们。具体地说，最好用英语开发的软件既适用于美国国内人民，又适用于全世界内不讲英语的其他人民。因此，以英语开发的许多软件应用程序稍后都要经过翻译，供非英语用户使用。

把软件包翻译成另一（或者一种以上其它）语言的过程既费时，费用又高。必须翻译各个文本消息、菜单和工具按钮，以使用户能够运用该程序。最直接的方式是搜索整个程序源代码，查找每个文本串，即查找将要显示给用户的每个字符串，并把各个字符串翻译成新的语言。

这种方法存在几个问题。一个问题是使用这种方法意味着必须相对于每种预期的语言，特别地翻译并编译软件。这本身是费用高昂的过程，并且意味着源代码中的任意变化都需要编辑和重新编译各种语言版本的代码。

图 2 中表示了 Java 应用程序中硬编码标记的例子。Java 是两种事物：编程语言和平台。一方面，Java 是高级编程语言，并且与从不同的是每个 Java 程序既被编译又被解释，从而使其能够利用多种平台。Java 也是运行于其它基于硬件的平台之上的纯软件平台。Java 平台具有两个组件：Java 虚拟机（Java VM）和 Java 应用程序编程接口（Java API）。

Java VM 是 Java 平台的基础，并且通向各种基于硬件的平台。Java API 是一大批提供多种有用能力的现成软件组件，例如图形用户界面

（GUI）窗口小部件。Java API 被分为相关组件的库（包）。

JavaBeans 为 Java 平台带来组件技术（component technology）。JavaBeans API 支持可重新使用的与平台无关的组件。通过使用服从 JavaBeans 的应用程序生成器工具，可把组件组合成为小程序、应用程序或复合组件。JavaBeans 组件被称为 Beans。

图 2 中，所示窗口 200 具有两个图形用户界面（GUI）“按钮”210 和 230。这些按钮是利用硬编码 Java “JButton” 调用产生的；例如，代码行 220 调用具有串变元“OK”的 Jbutton，产生具有文本标记“Ok”的按钮 210。类似地，代码行 240 调用具有串变元“Cancel”的 Jbutton，产生具有文本标记“Cancel”的按钮 230。要翻译使用如图 2 中的硬编码串的软件应用程序，必须分析每行代码，并且相对于各个目标语言，单独翻译各个显示串。

这种问题的一种解决方案是使用独立的本地化文件，要显示的文本串独立于其可执行代码单独存储在所述本地化文件中。当执行软件时，只是简单地以本地化文件中存储的任何语言从本地化文件中读出各个指定显示屏幕的文本。这样，可在不干扰可执行代码的情况下，翻译本地化文件中的文本，可在不干扰翻译文件的情况下，改变或替换可执行代码（当然，如果要显示的文本发生变化，则本地化文件中的相应条目也必须被改变的情况除外）。本地化文件可以采取任意多种格式，包括被编译信息目录、Java 资源束（resource bundle）、HTML 文件及其它多种格式。

本地化文件的使用便于进行多重翻译，但是带来的一个新问题是对于翻译者来说，是在不存在任何上下文关系的情况下，对单独文件中的孤立文本进行翻译。由于情况是这样，因此翻译通常总是含有错误，而如果翻译者能够按照文本出现的上下文进行翻译，就不会存在这样的错误。

不管用何种方法处理翻译，也必须校对使用中的程序的各个屏幕，以确保正确地显示翻译后的文本，并且正确地翻译显示环境中的文本。于是，通常的做法是雇用具有其它语言背景的人员来校对翻译后

的程序的各个屏幕，以便检测任意显示、语法或其它翻译错误。该第二阶段被称为翻译校验测试（TVT）。

图 3 中表示了按照常规方法的翻译和测试循环图。A 国的代表性公司 A 构建了打算供 B 国使用的应用程序（310）。接下来，公司 A 把应用程序打包并运输给位于 B 国的公司 B（320）。公司 B 将运行该应用程序，测试任何翻译或国际化错误（330）。随后公司 B 进行 TVT 测试，记录任何翻译错误，并进行任何必需的重新翻译（340）。随后公司 B 把改正内容返回公司 A，重复上述过程，直到不再发现错误为止。

按照这种常规方法，在 B 国每次校正错误或改变翻译，在重新测试产品之前，都必须在 A 国重构并重新打包该产品。因此，TVT 过程必然既费钱又费时。于是希望提供一种向翻译者提供辅助信息，尤其是上下文信息的装置，从而初始翻译能够尽可能准确。这样，可降低或消除进行 TVT 测试所需的费用和时间。

本发明的一个目的是提供一种改进的国际通用软件的开发工具。

本发明的另一目的是提供改进的多语言软件的开发工具。

本发明的又一目的是提供一种在计算机软件中进行语言翻译的改进系统和方法。

在下述详细的书面说明中，本发明的上述及其它目的、特征和优点将是显而易见的。

通过提供一种向语言翻译者提供要翻译文本的上下文关系信息的系统、方法和程序，可实现更高效的翻译过程。以基本语言向翻译者提供图形用户界面，随后翻译者可交互式翻译屏幕上的各个文本标记。由于是在正确的上下文中对文本进行翻译，因此可减少或消除翻译校验测试的时间和费用。通过向软件应用程序本身增加编辑器功能，获得在应用程序内编辑文本的能力。利用特定的资源束（resource bundle）名称并依据关键字，可将应用程序中的各个文本标记保存在本地化文件中。当激活编辑器时，内省文本对象，以查找其源数据，翻译者可直接编辑文本。随后源数据被用于保存译文，供以后使用。

因此，本发明提供了一种编辑软件程序的文本数据的方法，包括：通过调用包括与第一文本数据相关的源数据的软件对象，显示第一文本数据；响应用户输入，检查软件对象，识别源数据；从用户输入接收对应于第一文本数据的第二文本数据；按照源数据，以机器可读的形式保存第二文本数据；并在计算机显示器上替代第一文本数据而显示第二文本数据，其中机器可读形式最好是计算机存储媒介上的本地化文件。第二文本数据最好是第一文本数据的译文，软件对象最好是 Java 组件。源数据还可包括对 Java 资源束的引用，并且也可包括与第一文本数据相关的关键字。检查步骤中的用户输入是鼠标点击。

在另一个实施例中，本发明提供具有至少一个处理器、可存取存储器及可访问显示器的计算机系统，所述计算机系统包括：用于通过调用包括与第一文本数据相关的源数据的软件对象，显示第一文本数据的装置；用于响应用户输入，检查软件对象，以识别源数据的装置；用于从用户输入接收对应于第一文本数据的第二文本数据的装置；用于按照源数据，以机器可读的形式保存第二文本数据的装置；以及用于在计算机显示器上替代第一文本数据而显示第二文本数据的装置，其中机器可读形式最好是计算机存储媒介上的本地化文件。第二文本数据最好是第一文本数据的译文，软件对象最好是 Java 组件。源数据还可包括对 Java 资源束的引用，并且也可包括与第一文本数据相关的关键字。检查步骤中的用户输入是鼠标点击。

在又一个实施例中，本发明提供具有在计算机可用媒介上的计算机可读程序代码的计算机程序产品，包括：用于通过调用包括与第一文本数据相关的源数据的软件对象，显示第一文本数据的指令；用于响应用户输入，检查软件对象，以识别源数据的指令；用于从用户输入接收对应于第一文本数据的第二文本数据的指令；用于按照源数据，以机器可读的形式保存第二文本数据的指令；以及用于在计算机显示器上替代第一文本数据而显示第二文本数据的指令，其中机器可读形式最好是计算机存储媒介上的本地化文件。第二文本数据最好是第一文本数据的译文，软件对象最好是 Java 组件。源数据还可包括

对 Java 资源束的引用，并且另一方面可包括与第一文本数据相关的关键字。检查步骤中的用户输入是鼠标点击。

下面将参考附图，举例说明本发明的优选实施例，其中：

图 1 描述了根据本发明的优选实施例的系统可实现于其中的数据处理系统；

图 2 是具有硬编码文本的 GUI 显示画面的例子；

图 3 描述了按照常规方法的翻译和测试过程的方框图；

图 4 是根据本发明的优选实施例的翻译和测试过程的方框图；

图 5 描述了根据本发明的优选实施例，使用 Java 资源文件的 GUI 显示画面的例子；

图 6 是根据本发明的优选实施例的用于软件应用程序的内省工具的方框图；

图 7 描述了根据本发明的优选实施例的 Java 包装（wrapper）类别的方框图；

图 8 更详细地表示了根据本发明的优选实施例的 Java 包装类别；

图 9 描述了根据本发明的优选实施例的 Java façade 类别的方框图。

现在参考附图，尤其是参考图 1，图中描述了本发明的优选实施例可实现于其中的数据处理系统的方框图。数据处理系统 100 可以是，例如可从 New York，Armonk 的国际商用机器公司购买的计算机之一。数据处理系统 100 包括处理器 101 和 102，在例证实施例中，处理器 101 和 102 分别与二级（L2）高速缓冲存储器 103 和 104 相连，高速缓冲存储器 103 和 104 又与系统总线 106 相连。

与系统总线 106 相连的还有系统存储器 108 和初级主桥接器（PHB）122。PHB 122 使 I/O 总线 112 与系统总线 106 耦接，转发和/或变换从一个总线到另一总线的数据事务。在例证实施例中，数据处理系统 100 包括与 I/O 总线 112 相连，为显示器 120 接收用户界面信息的图形适配器 118。诸如非易失性存储器 114 和键盘/定点设备 116 之类的外围设备通过产业标准结构（ISA）桥接器 121 与 I/O 总

线 112 相连，非易失性存储器 114 可以是硬盘驱动器，键盘/定点设备 116 可包括传统鼠标、跟踪球等等。PHB 122 还通过 I/O 总线 112 与 PCI 插槽 124 相连。

图 1 中所示的例证实施例只用于说明本发明，本领域的技术人员将认识到形式及功能方面的各种各样变化都是可能的。例如，数据处理系统 100 还可包括光盘只读存储器（CD-ROM）或数字视频光盘（DVD）驱动器、音卡和音频扬声器、以及各种其它可选组件。所有这种变化都被认为在本发明的精神和范围之内。数据处理系统 100 和下面的例证图形只是作为说明性的例子提供的，并不意味着结构方面的任何限制。事实上，该方法和系统可容易地适于用在任何可编程的计算机系统或系统网络上，软件应用程序可在所述可编程计算机系统或系统网络上运行。

软件应用程序的典型翻译过程包括两个阶段。在第一阶段中，在文本文件中捕捉应用程序的人类文本单元的人类语言文本，并将其递送给翻译者进行翻译。这些翻译者不了解该应用程序，从而不具有和被翻译文本的实际应用有关的任何上下文信息。从而，该翻译过程不可避免易于出错。为了解决这些错误，并验证翻译后的文本，进行第二阶段，即翻译校验测试。

优选实施例通过允许开发人员合并翻译过程的这两个阶段，简化了翻译过程。这是通过向翻译者提供和被翻译内容相关的直接上下文信息实现的。从而，可简化或消除费用大并且耗时的 TVT 阶段。

根据优选实施例，提供了一个内省编辑器，所述内省编辑器允许翻译者直接编辑界面组件，将翻译结果保存在文本文件中，在优选实施例中，文本文件是 Java 资源束。这向翻译者提供进行准确翻译所必需的上下文关系，从而翻译可更为准确。

现在参考图 4，图中表示了根据优选实施例的改进的翻译和测试过程的方框图。A 国的公司 A 构建了打算供 B 国使用的应用程序（410）。接下来，公司 A 把应用程序打包并运输给位于 B 国的公司 B（420）。公司 B 将运行该应用程序，测试任何翻译或国际化错误

（430）。随后公司 B 将进行 TVT 测试，记录任何翻译错误，并进行任何必需的重新翻译（440）。和上面图 3 中描述的情况不同，由于本实施例允许参照上下文关系进行翻译，并本地保存翻译，因此公司 B 可立即对翻译进行再测试，而不必把产品送回公司 A 进行重构建。从而，和按照常规方法的翻译及测试阶段相比，该翻译及测试阶段的效率高得多。

当然，在 B 国的测试阶段可能会暴露出其它错误，例如不按翻译问题的方式纠正的编程问题。这种情况下，和常规过程一样，应用程序被送回公司 A 进行纠错。

根据这里说明的几个实施例，翻译者会实际运行他正在翻译的软件应用程序，公开的实施例向翻译者提供直接从屏幕选取文本，并实时编辑所选取文本的途径。由于翻译者如同最终用户那样实际看到该应用程序，因此翻译会准确得多。

使用中，以伴随要翻译的文本的“弹出式”编辑器窗口的形式呈现根据优选实施例的内省编辑器。当文本要被翻译时，内省（introspect）与该文本相关的 Java 组件，确定其源数据，例如资源束名称和关键字。当翻译者选择该文本时，显示编辑器弹出式窗口，翻译者将输入该按钮的翻译文本。编辑器将关闭，并且或者在新文件中，或者通过在初始文件中重写初始文本，把翻译后的文本存储在本地化文件中。从而，在按钮将出现于其中的最终应用程序的环境下，完成整个翻译。

在优选实施例中，“ctrl-alt-shift”键组合被用于表示显示的文本将被翻译。当显示 GUI 窗口时，通过在显示屏幕中的文本标记上按下“ctrl-alt-shit”和点击鼠标或其它定点设备的右键，翻译者指示该文本标记将被翻译。编辑器窗口打开，翻译者可把新文本直接输入编辑器窗口中。当翻译结束时，存储新文本供未来使用。

通过直接编辑软件应用程序的用户界面中的文本单元，并且通过向翻译者提供上下文关系，这种方法消除了翻译过程的 TVT 阶段，从而减少了时间和错误。

编辑器被称为“内省“编辑器，因为在本实施例中，它利用 Java 编程语言的内省特性，借助“setText”和“getText”方法查看并编辑控件。虽然这里说明了 Java、JavaScript、JavaBeans 及其它与 Java 相关工具的一些方面，可从太阳微系统公司得到其它材料，并且（从本申请的申请日期开始）可在 http: //java.sun.com 找到其它材料。

各个 Java Bean 组件具有其自身的特征，包括其属性、方法及事件。借助称为内省的过程，其它 Beans 可检查这些特征。Beans 以两种方式支持内省:

2 通过在命名 Bean 特征时遵守称为设计模式的特定规则。java.beans.Introspector 类别针对这些设计模式检查 Beans 以发现 Beans 特征。Introspector 类别依赖于核心反射 API; 和

3 通过将相关的 Bean Infermation 类别明确提供给性质、方法和事件信息。Bean Information 类别实现 BeanInfo 界面（interface）。BeanInfo 类别明确列出将暴露给应用程序构建工具的那些 Bean 特征。

性质是可在设计时候改变的 Bean 的外观和行为特征。构建工具内省 Bean，以发现其性质，并揭示这些性质以供操作。

Beans 揭示属性，从而可在设计时定制这些性质。以两种方式支持定制: 使用性质编辑器或使用更复杂的 Bean 定制。Beans 利用事件与其它 Beans 通信。希望接收事件的 Bean（接收 Bean）向发出该事件的 Bean（源 Bean）登记其兴趣。构建工具可检查 Bean 并确定该 Bean 可发出（发送）哪些事件，以及可处理（接收）哪些事件。

连续性使 Beans 能够保存并恢复它们的状态。一旦你改变了 Beans 性质，你可保存 Bean 的状态，并在以后恢复该 Bean，性质保持不变。JavaBeans 使用 Java 对象串行化支持连续性。

Bean 的方法与 Java 方法没有区别，并且可从其它 Bean 或脚本编写（scripting）环境调用 Bean 的方法。

虽然 Beans 被设计成能被被构建工具理解，但是包括对事件、性质及连续性的支持在内的所有关键 API 已被设计成也易于被程序员

阅读和理解。

Introspector 类别向工具提供学习由目标 Java Bean 支持的性质、事件及方法的标准方式。对于这三种信息中的每种信息，Introspector 将单独分析 bean 的类别及超类别，查找明确或含蓄信息，并使用该信息构建综合描述目标 bean 的 BeanInfo 对象。

用于翻译的内省编辑器

优选实施例提供了使用类别字段值，从而允许翻译软件文本的方法。软件显示屏幕中的每个文本标记都被存储为一个类别对象。

本实施例中，将在例如，GUI 按钮上显示的实际文本未被硬编码，而是存储在 Java ResourceBundle 本地化文件中。现在参考图 5，图中表示了利用 Java 资源束显示的按钮标记的一个例子。图 5 中，所显示的窗口 500 具有 GUI 按钮 510 和 530。注意该窗口对应于图 2 的窗口 200。但是，这里是利用 Java JButton 调用 520 产生按钮 510 的，Java JButton 调用 520 指的是 ResourceBundle。

本例中，命令“JButton（res.getString（“OK”））520 产生 GUI 按钮，并通过查找资源文件，寻找应对应于“OK”按钮的文本，标注该 GUI 按钮。Java 资源文件被共同描述为 550，并且英语参考文件被表示为 560。在文件 560 中，条目 580 指出对应于“OK”按钮的文本是“OK”。于是，命令 520 产生带有文本“OK”的 GUI 按钮。注意在本例中，为了清楚起见，用大写字母显示按钮名称，用大小写混合字母显示按钮文本标记。

类似地，命令“JButton（res.getString（“CANCEL”））540 产生 GUI 按钮，并通过查找资源文件，寻找应对应于“CANCEL”按钮的文本，标注该 GUI 按钮。在文件 560 中，条目 570 指出对应于“CANCEL”按钮的文本是“Cancel”。于是，命令 520 产生带有文本“Cancel”的 GUI 按钮 530。当然，资源文件中的实际对应文本可以是任意文本；条目 570 可为{“CANCEL”，“Stop”}，于是每次执行命令 540 时，会导致产生标注为“Stop”的按钮。当然，这种方法并不局限于 GUI 按钮，也可用于从定域资源文件中查找任意的

可显示文本。

由于资源文件把要显示的文本存储为 Java 对象，因此 Java 环境的内省特性可被用于允许翻译者利用弹出式编辑器，直接编辑应用程序文本。

当激活弹出式编辑器时，对目标对象进行 getText 调用，所述调用把当前文本返回给编辑器。该文本被显示给翻译者，翻译者随后再编辑显示的文本。在完成翻译/编辑之后，利用 setText 调用，把新文本保存在对象中。

最后，具有新文本的对象被存回 Java ResourceBundle 文件。之后，可利用修改后的对象执行软件应用程序，于是便将在显示器上显示翻译后的文本。

参考图 6，应指出的是在本实施例中，内省编辑器 610 是不同于正被翻译的应用程序 650 的独立软件程序。编辑器 610 含有内省应用程序编程接口（API）620、编辑器 GUI 630 及文件 I/O 系统 640 的代码。目标应用程序 650 把含有要翻译的文本串的 UI 对象 660 保存在 Java 资源文件中。

本实施例中，翻译者同时运行目标应用程序 650 和内省编辑器 610。当使用弹出式编辑器时，内省编辑器从目标应用程序 650 中检索对象，并保存检索出的对象。

Java Wrapper API

另一实现方法完成和上述相同的上下文翻译结果，但是使用的是 Java UI 类别的“wrapper”类别。Java wrapper 类别允许程序员把 Java 组件“包装”为新的组件，所述新组件具有原始组件的所有属性，外加一个或多个新的或扩展属性。

本实施例扩展诸如 JButton 之类的 Java Swing 组件，以便把资源束名称属性和关键字名称属性保存在实例对象中。扩展后的组件利用这些属性和 get/set 方法，实现构造器。这些属性可以显示在这些组件上的弹出式对话框中，并且运行时，用户可编辑该标记。在弹出式对话框中进行的修改可直接反映到应用程序 UI 组件上，并被保存在

记录文件中。

在大型应用程序中，可能在几个资源文件中定义同一文本（例如“Save”），不同的对话框可把不同的资源文件用于相同的标记。

如果翻译者发现必须用另一单词，例如“Store”替换指定对话框中的标记，例如“Save”，但是其它对话框应保留为“Save”，则翻译者必须知道实际上哪个资源文件被用于显示该标记。翻译者通常可以使用资源文件，但是不能使用程序源，而在程序源中提供了 GUI 组件和资源文件的联系。翻译者不得不推测哪个资源文件正被使用，而这是效率非常低、耗时并且易于产生错误的步骤。

本实施例提供了使翻译者了解哪个资源文件及哪个关键字被用于特定的 GUI 标记的方法。为了实现这一点，必须利用 Java wrapper 类别把资源文件名称和关键字名称添加到 GUI 组件实例以及标记自身中。一旦该信息被保存到 GUI 组件中，工具可向翻译者显示要编辑的资源名称及关键字，或者工具甚至可利用由工具所提供的编辑器界面，为翻译者编辑资源文件。

wrapper 以两种方式扩展 Java 类别。一种方式是增加两个串属性用于保存资源文件名称及关键字，另一种方式是提供构造器及方法以设置/获得(set/get)这些属性。wrapper 的构造器及方法非常类似于父类别的那些构造器及方法。

“setText/getText”被用于设置和获得基于文本的组件的标记，所述标记始终是文本条目。即使程序写下 setText（getKey（…）），所述标记也仅仅是解析的文本，并不包含资源文件名称及关键字。

和上面描述的只依赖于 getText/setText 方法的实施例相比，通过利用 Java wrapper 类别，本实施例具有两个优点：

1　wrapper 可保存资源文件名称及关键字，而 setText 只保存标记。

1　wrapper 的构造器设置资源文件名称和关键字，以及标记。在无 wrapper 的情况下，每当开始人员对带有标记的 GUI 组件实例化时，都需要调用另一种方法来设置资源文件名称、关键字及标记，

而在开发过程中，该步骤频繁地被忽略。

如图 7 中所示，该实施例扩展 Java 基础类别（Foundation Class）的文本组件，例如 JButton 710、Jlabel 720、JCheckBox 730 及 JMenu 740，以产生相应的组件 TButton 712、Tlabel 722、JCheckBox 732 和 JMenu 742；当然，也可类似扩展其它的 JFC 组件。新组件为每个组件的资源束名称（res）和关键字增加了一个获得/设置（get/set）方法。

现在参考图 8，图中更详细地表示了“JButton”类别的组件扩展。图 8 中，利用 Java wrapper API，JButton 类别 810 被扩展为 TButton 类别 820。TButton 类别具有与初始的 JButton 类别相同的“标记”属性 830，但是增加了用于资源束名称（res）840 和关键字 850 的属性。这些允许翻译者按照各个 TButton 的关键字，直接调出并编辑各个 TButton 的标记属性。

利用这种方法，翻译者可在这些组件上调出弹出式菜单，并在运行时直接编辑/翻译标记。对标记做出的改变按照类别和关键字被保存在记录文件中，并且还立即在 GUI 显示屏幕上做出相应变化。之后，当软件应用程序遇到该类别和关键字时，显示编辑/翻译后的文本。

应注意和上面的内省工具不同，本实施例使用作为正被翻译的软件应用程序的一部分的 wrapper 类别。如果要翻译的软件应用程序不支持扩展的 wrapper 类别，则该方法将无效。

Facade

另一实施例通过产生“facade”类别，把每个组件翻译信息附到 JComponent 上。在本实施例中，存取方法和编辑器用户界面逻辑被集中到单个“facade”类别中，而不是使它们位于各个 wrapper 类别中。

应用程序借助该 facade 类别设置 Java UI 组件的资源束名称和关键字名称。这些属性可显示在这些组件上的弹出式对话框中，并且用户在运行时编辑标记。在弹出式对话框中做出的改变直接被反映到应

用程序 UI 组件上，并被保存在记录文件中。

该方法与上面的包装（wrappering）方法解决相同的问题。和包装（wrappering）方法相比，这种方法的优点是:

（1）在包装方法中，亚分类具有维护开销（新的及更新的 JComponents 要求改变导出的一组组件）；

（2）包装招致开发开销，因为要求设计人员只从该组可用子类别中选择，或者实现他们自己的组件;

（3）包装还会招致运行时间开销，因为必须把编辑器属性加入到各个组件中；及

（4）包装实施例中使用的 setName（）涉及把所有的翻译属性组合到组件的“名称”属性中，而基于性质的方法提供对各个翻译属性的直接存取，更易于扩展（例如，对于工具提示文本翻译属性），并且留下组件的“名称”属性，以用于其它用途。

该实施例提供用作 JFC 文本组件（JButton、JLabel、JCheckBox、JRadioButton）的 facade、并实现资源束名称和关键字的 get/set 方法的类别“NLSUtility”。这些属性被保存在具有存取关键字的 clientProperty 散列表中。用户可在这些组件上显示弹出式菜单，并在运行时编辑标记。对标记做出的修改被保存在记录文件中。

图 9 表示了根据该 facade 类别实施例的方框图。这里，facade 类别 900 起 JFC 文本组件 910、920、930、940（及其它，图中未示出）的前端（front end）的作用，并使所有这些 JFC 文本组件中的每个文本组件与某个客户端性质(client property)属性相联系。随后通过使用 putClientProperty( )950 和 getClientProperty( )960, 可使用 facade 类别直接读取和编辑底层的 JFC 基础类别的性质属性。通过利用 façade 类别，获得和上面的包装（wrappering）实施例相同的结果，但是编辑器功能的开销被加入到 façade 类别中，而不是加入到各个单独组件中。

重要的是注意虽然在全功能数据处理系统和/或网络的环境下说明本发明，但是本领域的技术人员将认识到本发明的机制能够以具有多

种形式的指令的计算机可用媒介的形式散布，并且无论用于实际进行所述散布的信号承载媒介的特定类型如何，本发明的应用情况都相同。计算机可用媒介的例子包括：非易失性硬编码型媒介，例如只读存储器（ROM）或可擦电可编程只读存储器（EEPROM），可记录型媒介，例如软磁盘、硬盘驱动器和 CD-ROM，以及传输型媒介，例如数字及模拟通信链路。

虽然已参考几个优选实施例具体显示和说明了本发明，但是本领域的技术人员要明白，在不脱离本发明的精神和范围的情况下，在形式和细节上可做出各种变化。此外，要明白虽然已论述了具体的数据结构、函数和语言，在不脱离本发明的范围的情况下，所公开的技术可容易地适用于其它编程环境。

说　明　书　附　图

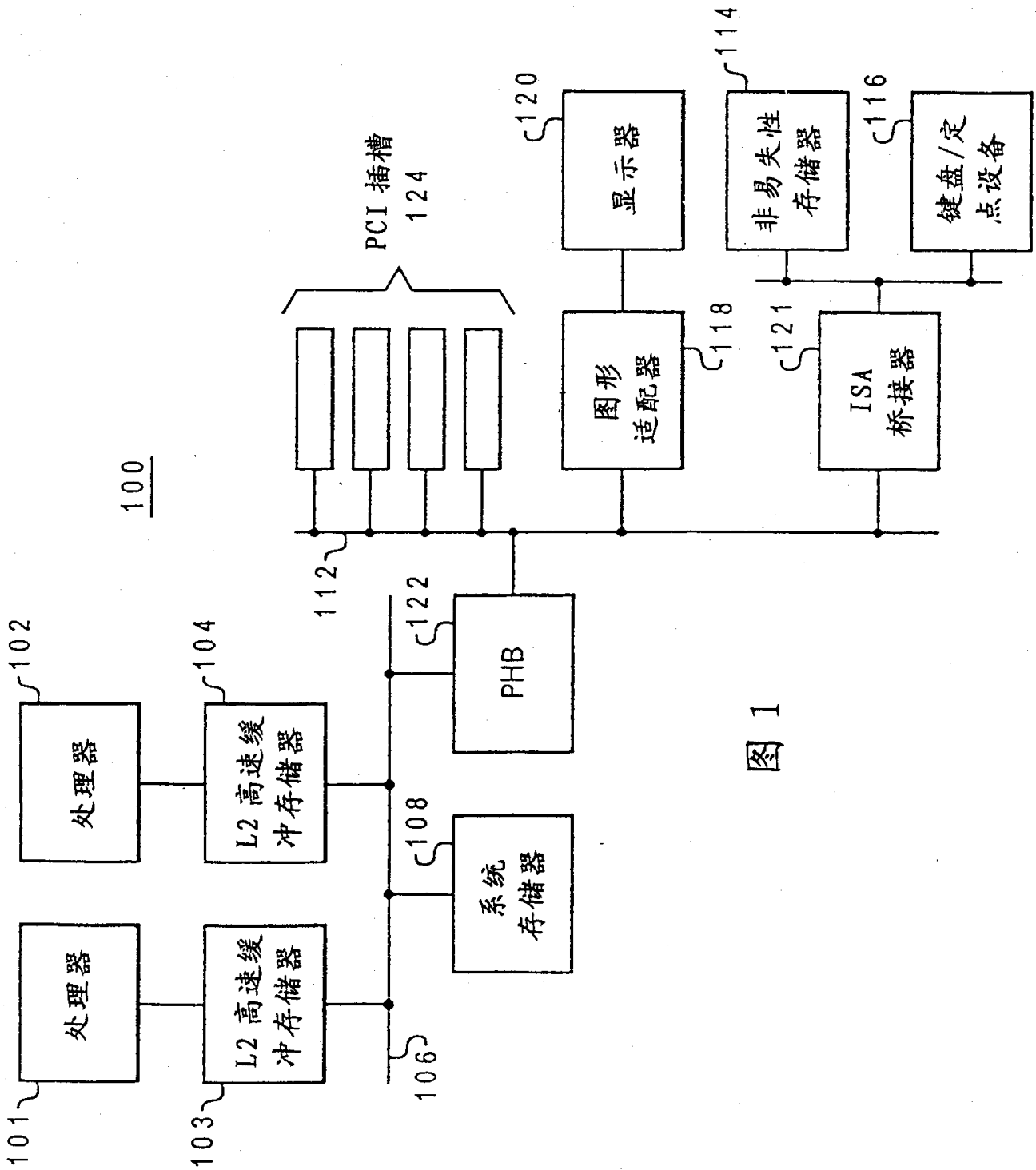

100
处理器
101
处理器
102
103
L2 高速缓冲存储器
104
L2 高速缓冲存储器
106
108
系统存储器
PHB
122
112
PCI 插槽
124
显示器
120
图形适配器
118
非易失性存储器
114
ISA 桥接器
121
键盘/定点设备
116

图 1

硬编码 Java UI 对象标记

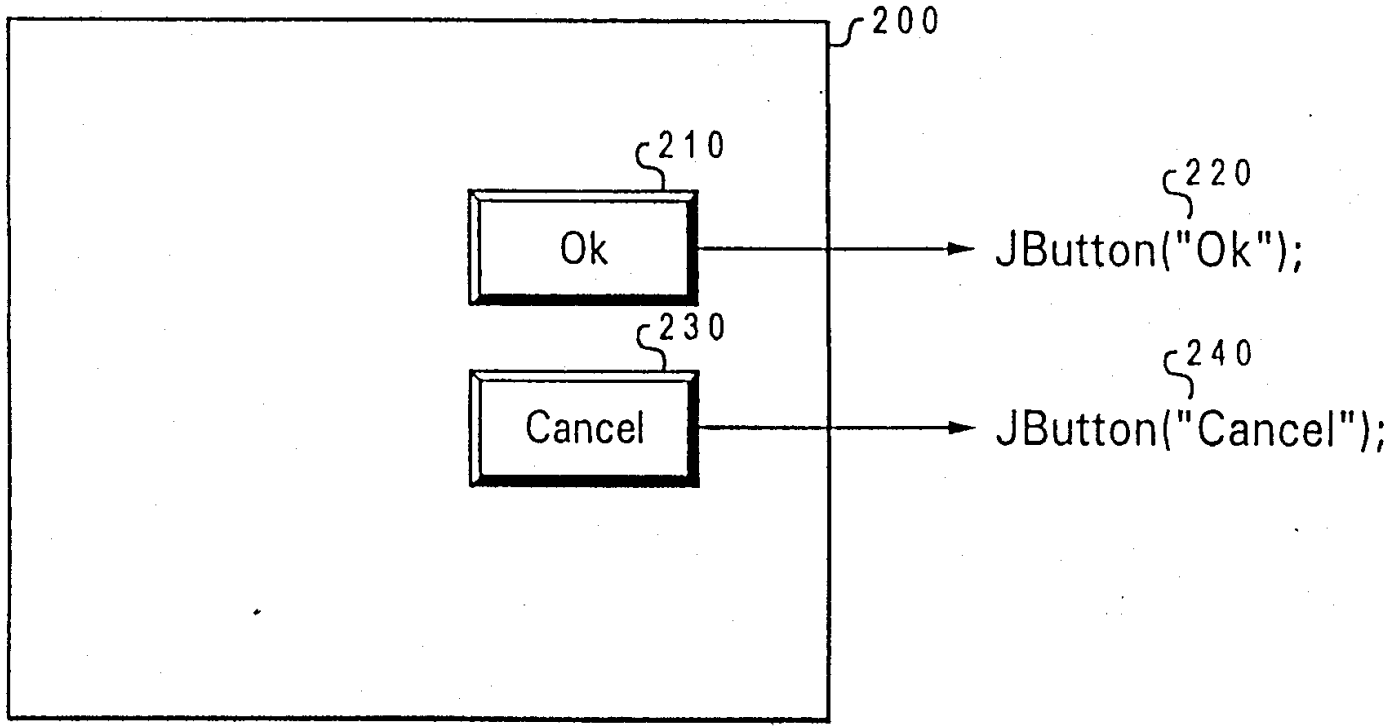


图 2

目前的翻译/测试循环

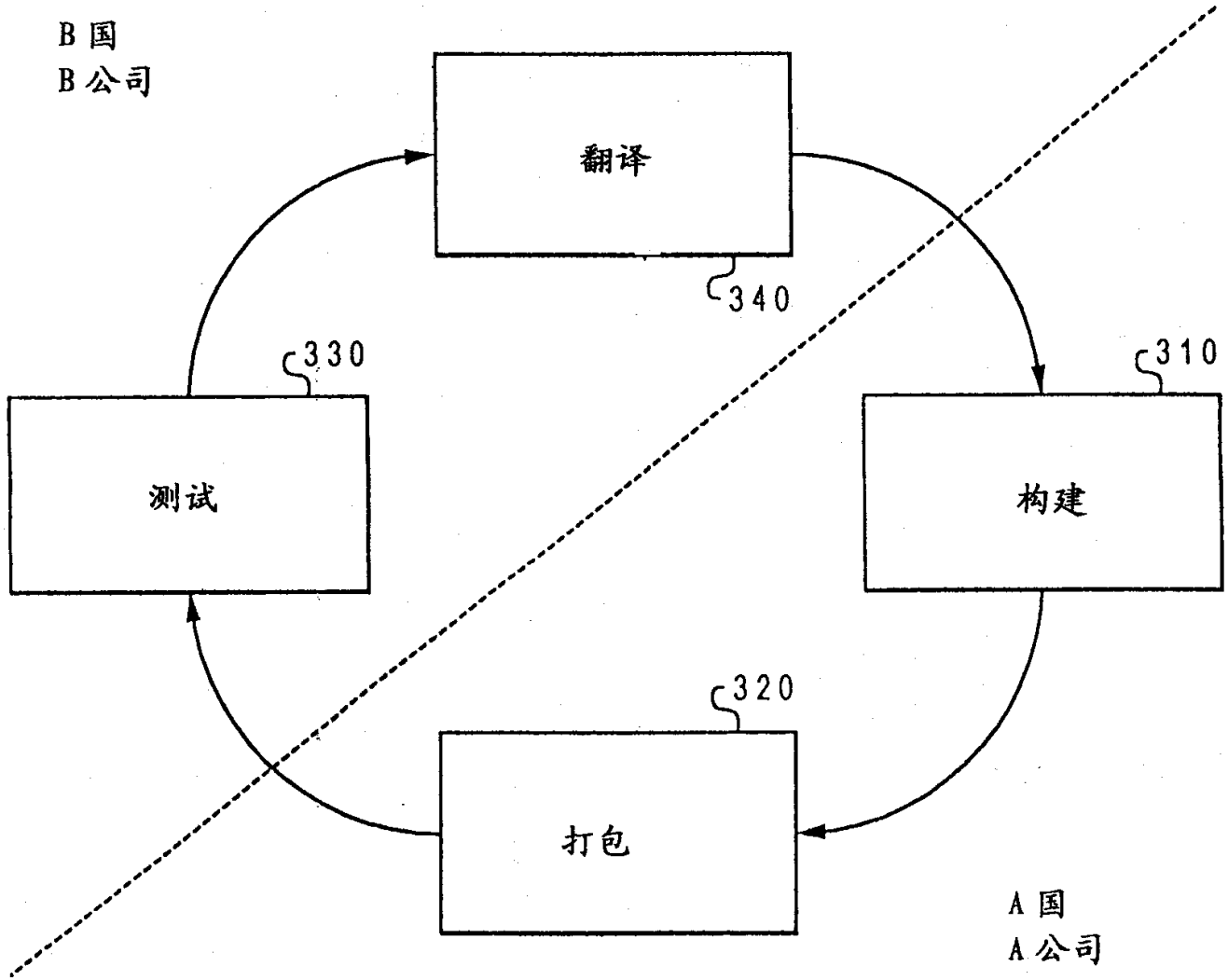


图 3

利用本发明工具的改进的翻译/测试循环

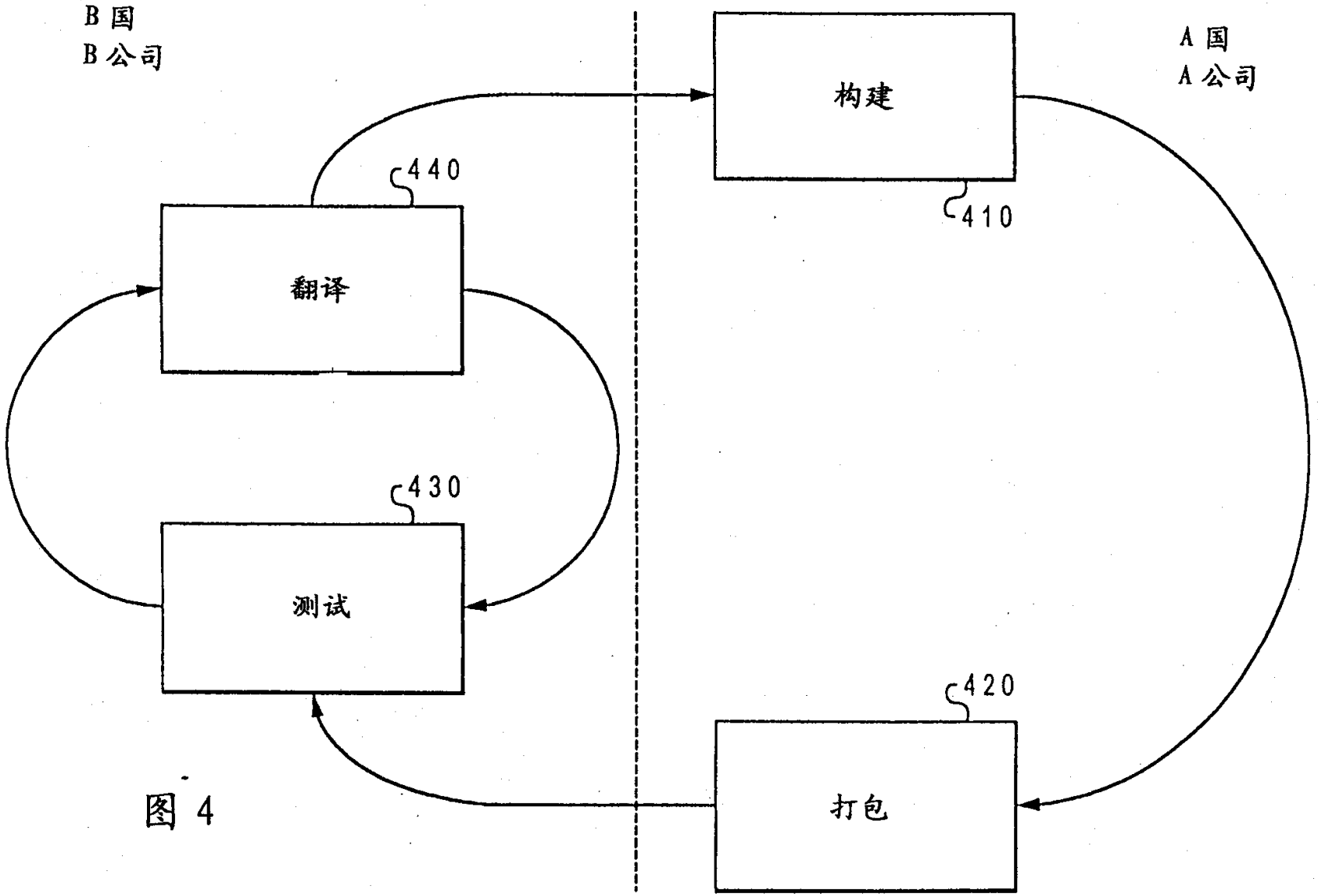


图 4

使用 ResourceBundle 的 Java UI 对象标记

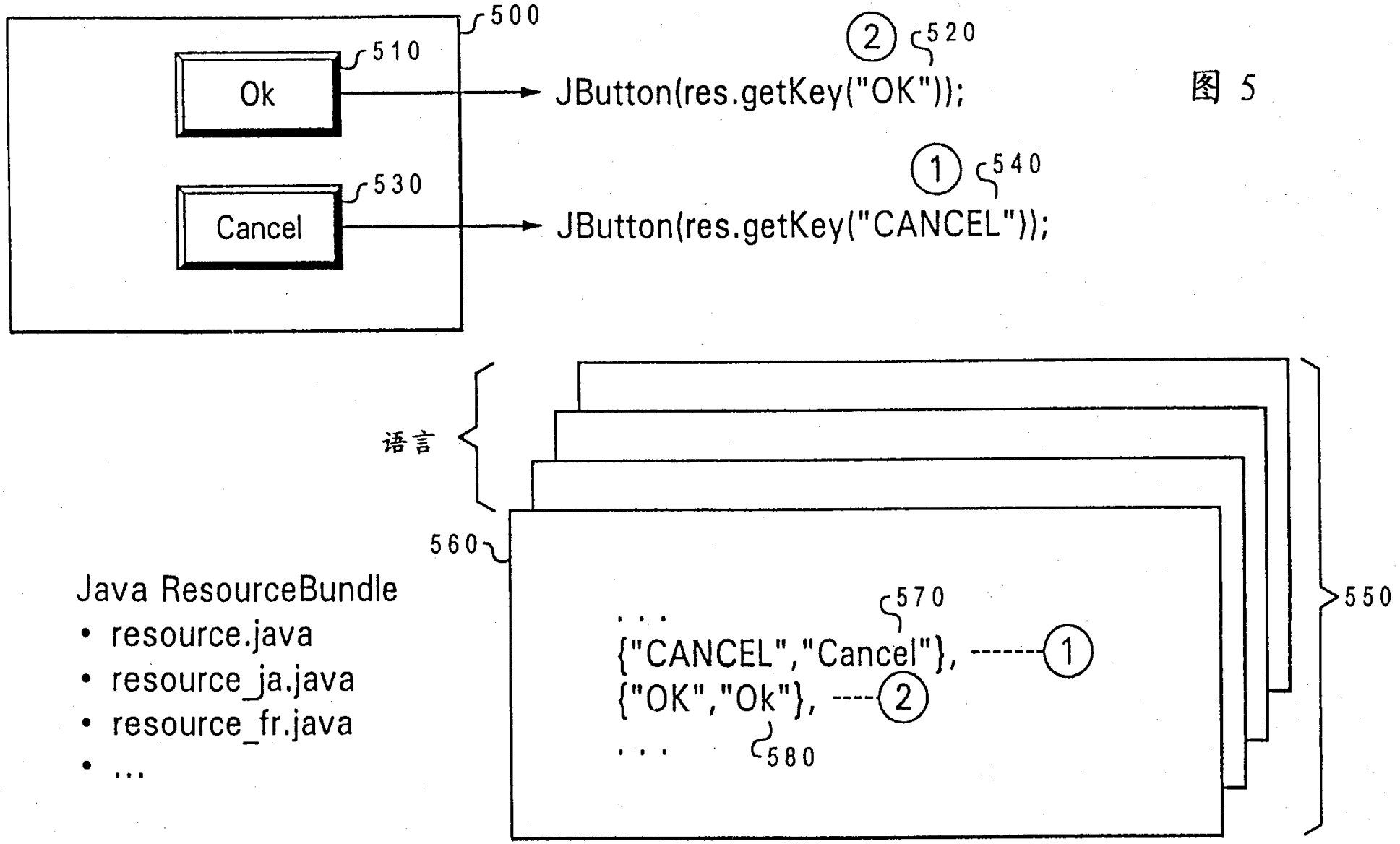


图 5

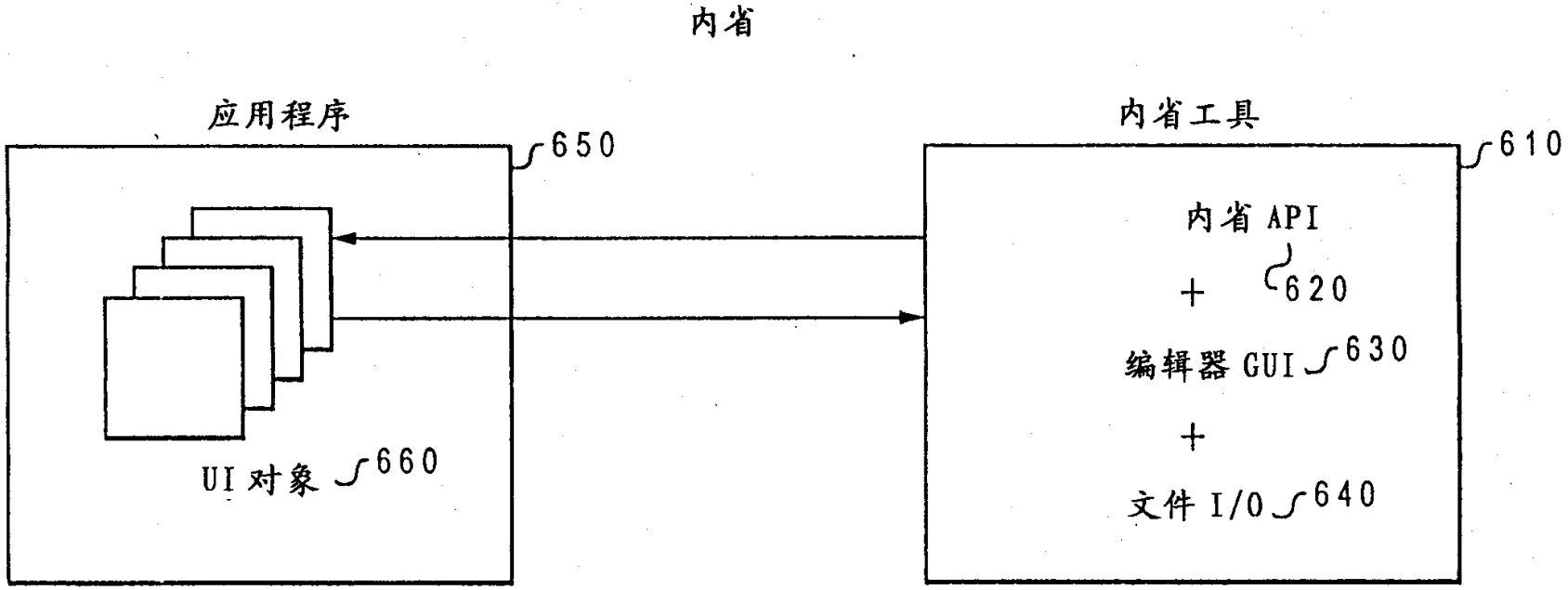


图 6

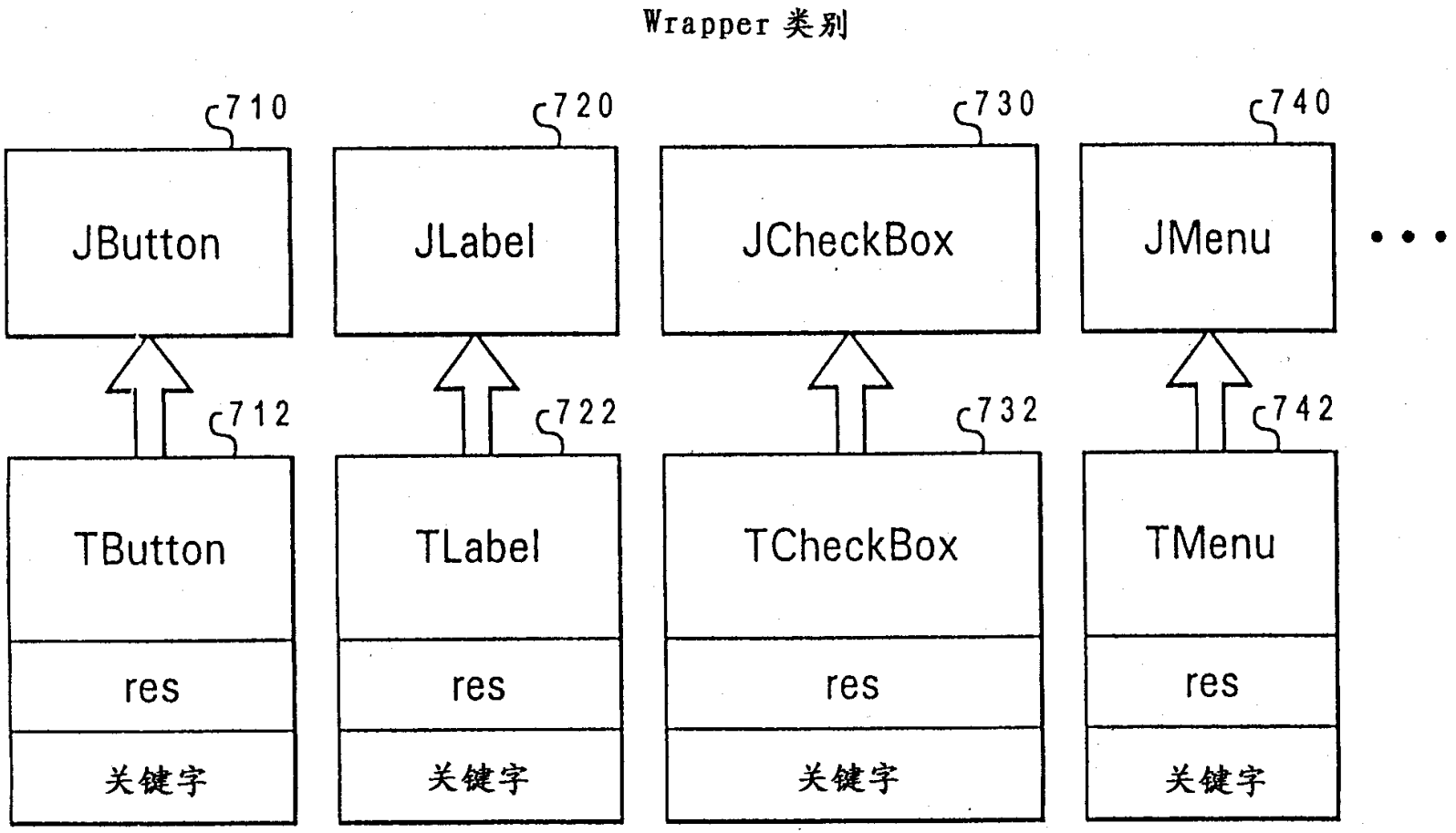

Wrapper 类别
710
720
730
740
JButton
JLabel
JCheckBox
JMenu
712
722
732
742
TButton
TLabel
TCheckBox
TMenu
res
res
res
res
关键字
关键字
关键字
关键字

图 7

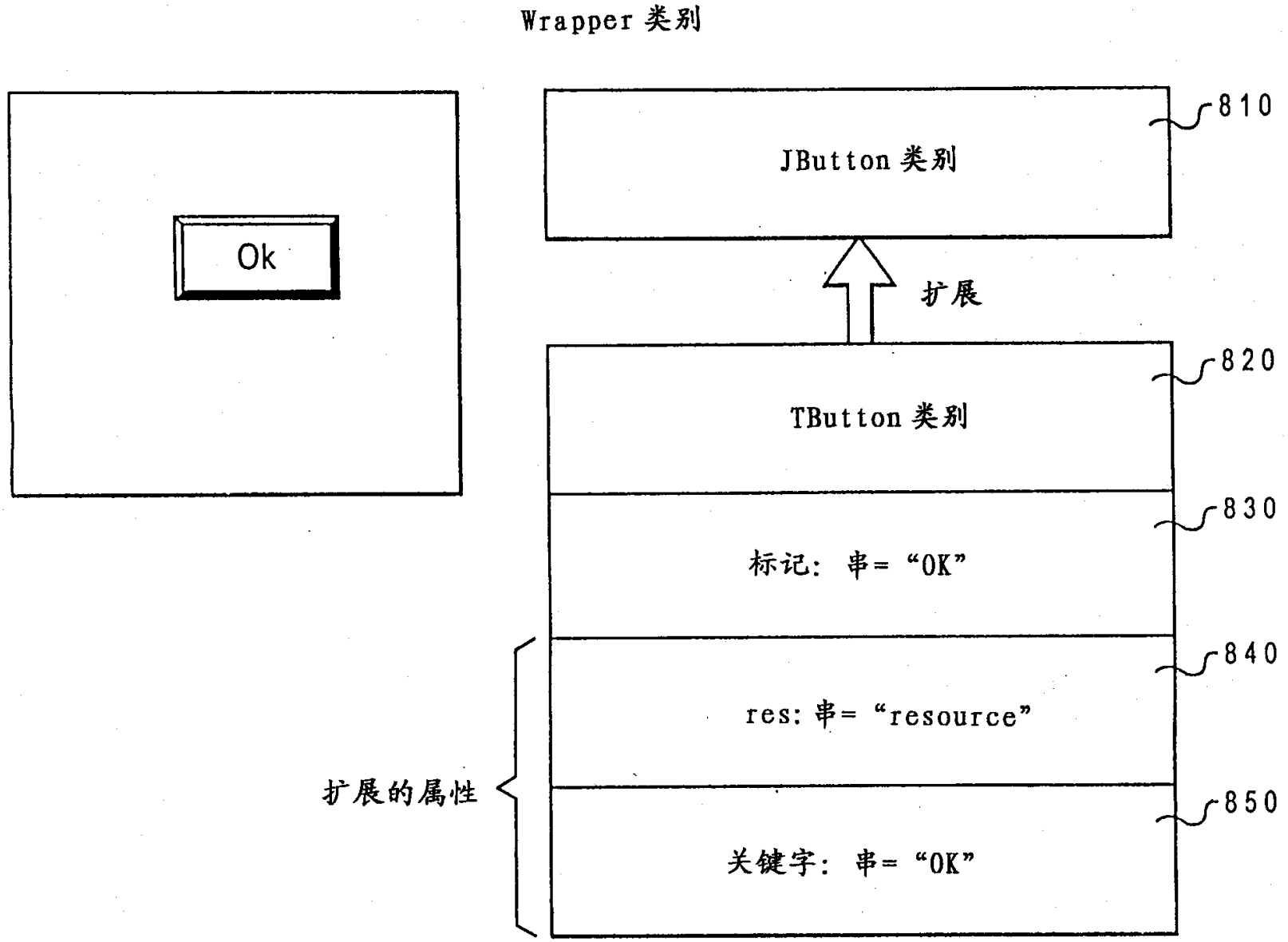

Wrapper 类别
Ok
JButton 类别
810
扩展
TButton 类别
820
标记: 串= “OK”
830
res: 串= “resource”
840
关键字: 串= “OK”
850
扩展的属性

图 8

知识产权出版社出版
ISBN 7-980008-04-9

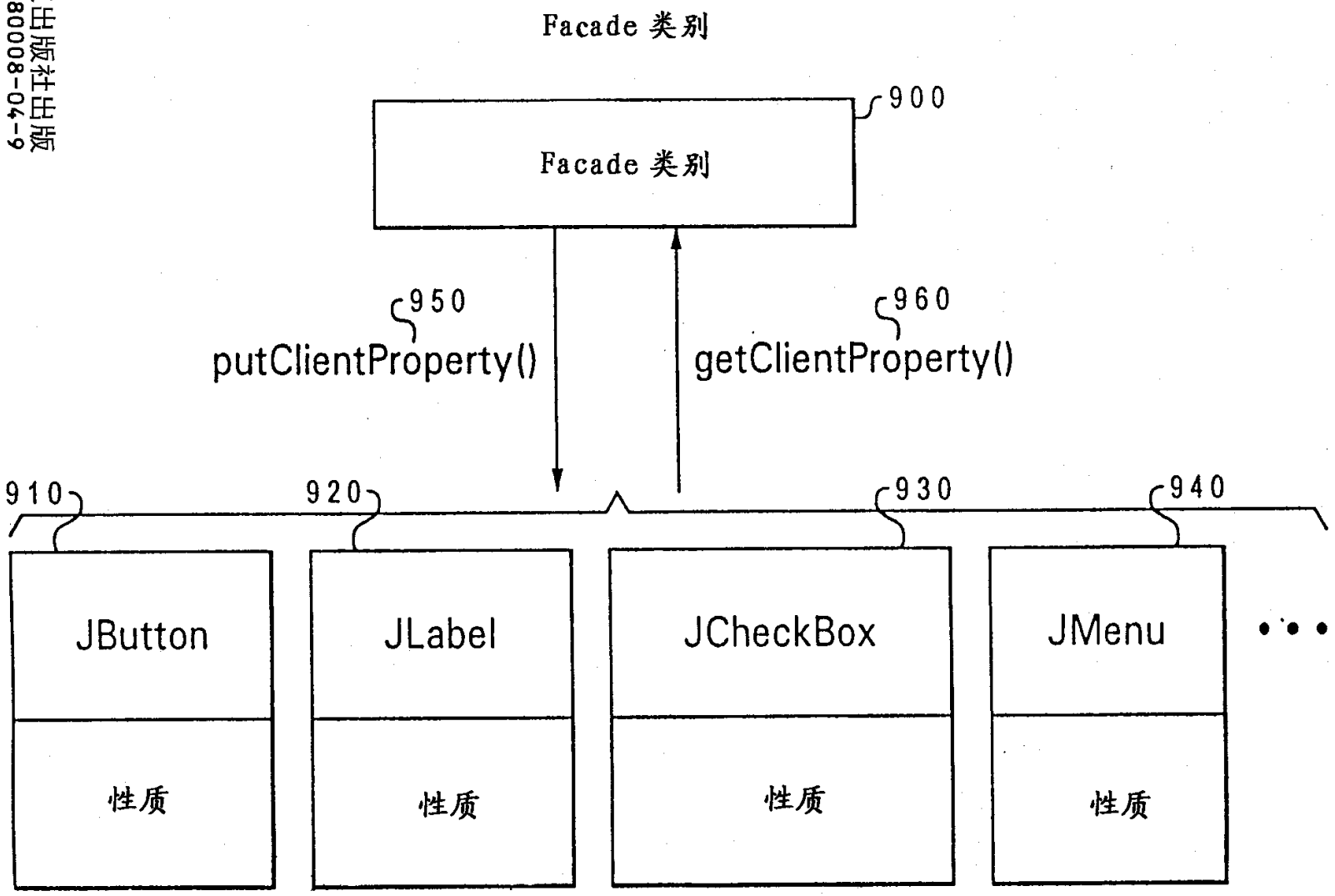


图 9