提高应用程序接口可用性探究①

Improving the Usability of Application Programming Interface

叶青青　(杭州职业技术学院 浙江 杭州 310018)

摘　要： 尽管没有图形界面，作为信息产品的应用程序接口也存在可用性的问题，应用程序接口的可用性直接影响到应用软件人员的开发效率和质量。基于可用性工程研究者所提出的设计具有良好的可用性时需要遵循的普遍原则，再结合 API 本身的特殊性和 API 用户(应用软件开发人员)的特殊性，本文对设计、评估和提高 API 的可用性时要考虑的主要要素进行初步的探究。

关键词： 可用性 应用程序接口 软件开发

1　引言

随着计算机的普及和计算机用户的急剧增长，计算机系统的可用性问题得到了越来越广泛的重视。衡量一个计算机系统的可用性主要是看它的用户能否使用该系统有效地达成他的工作目的，系统使用时的工作效率如何，用户的主观感受又怎样。评估可用性的国际标准有 ISO 9241 10-17，它规定了五个评估标准：效率性、易学性、记忆性、容错性和满意度。

由于可用性是交互式计算机系统和产品的重要质量指标，体现了该产品的核心竞争力，同时也可以减少后期维护，降低开发成本，提高用户接受度和提高企业信誉度，它不仅吸引了无数研究人员对这一问题进行深入的探讨，也促使各大软件公司投入大量的人力物力提高计算机系统和工具的可用性。

目前对可用性的研究主要集中在用户界面的设计上，特别是对具有图形界面的设计。事实上，可用性的问题涉及到所有的信息产品，无论它们是否有图形界面。其中有一个尚未引起普遍重视的是应用程序接口(API)的可用性问题。

尽管没有一个图形界面让用户直接操作，API 也是一种信息产品，它的用户是应用软件开发人员。应用软件开发人员利用 API 来达成他们的编程目的，一个具有良好可用性的 API 会提高软件开发效率，减少程序错误。

多年来，可用性的研究主要集中在普通用户上，由于软件开发人员是专业的 IT 人士，他们使用的计算机系统和信息产品的可用性并没有引起足够的重视。近来，由于下列原因 API 的可用性问题开始得到了一定的重视。首先是由于 API 数量的急剧增加。比如说 Java 1.5 的标准 API 有 3279 个类,.Net 的 API 有 2686 个类。其次是编程人员的扩大，以前 API 只是由一些专业的软件开发人士使用，如今越来越多的人加入了编程的行列，尽管他们的专业不是软件开发，但他们却要频繁地编制一些程序。第三，随着 web service 的概念的推广，越来越多的系统开始提供 API，比如说 Google，Yahoo，Flickr 等等，因此，API 设计不仅仅是少数的编程语言设计者所要考虑的问题，它正在逐步成为各种计算机系统开发人员都要关心的问题。

那么 APIs 的可用性涉及到那些具体的因素呢？基于可用性工程研究者所提出的设计具有良好的可用性时需要遵循的普遍原则[1]，再结合 API 本身的特殊性和 API 用户(应用软件开发人员)的特殊性[2]，本文对设计、评估和提高 API 的可用性时要考虑的主要要素进行初步的探究。

2　API的可用性

API 是应用软件人员的工具，衡量其可用性要看作为 API 用户的应用软件人员是否能容易地学会、记

① 收稿时间:2008-11-19

住，并在编程时有效快速地使用。具体地说，一个具有良好可用性的 API 应有下列几个特点：

易学：应用软件人员应该不需要花费太多的时间学会使用 API；

易记：一旦学会了以后不容易忘记；

易懂：应用软件人员在阅读或维护软件时能迅速地理解它的含义和功能；

高效：应用软件人员在编程时应该能用少量的代码完成大多数的编程任务；

出错率低：使用时不容易犯如写错参量的顺序或用错参量的类型等文法或语义上的错误。

为了达到具有以上特性的可用的 API,设计和评估 API 时要着重注意在抽象度、基本组合、透明度、一致性、自然对应等五个方面的考虑。下面结合一些具体的例子对这五个方面进行具体地阐述。

2.1 抽象度

API 是对某一个领域的各种操作功能的一种抽象。抽象本身是一个双刃剑，有利也有弊。高度的抽象可以减少 API 的个数，减少需要学习和记忆的 API 的个数，提高 API 的易学性和易记忆性。

但是高度抽象的 API 也会降低可用性。当应用软件人员把一个高抽象度的 API 应用于特定程序时，他们往往要作一些特殊处理，增加编程的工作量，从而减少应用 API 时的效率性。高度的抽象也会带来理解 API 的困难，从另一个方面降低 API 的易学性，提高应用时出错的可能性。

在设计和评估 API 的可用性时必须要考察它所提供的抽象程度是否与应用软件开发者在实际开发中所需要的一致，是否符合应用软件开发者的思维习惯。比如说我们要提供画圆的 API。最直接的抽象可以是

```
drawCircle(int x, int y, int radius)
```

但我们也可以提供更高一层的抽象，只提供画圆弧的功能

```
drawArc(int x, int y, int radius,
        int startAngle, int arcAngle)
```

如果进行进一步的抽象，我们也可以考虑只提供画椭圆弧线的功能

```
drawElliptiArc(int x, int y, int width, int height,
        int startAngle, int arcAngle)
```

如果要在坐标(1, 2)画一个半径为 5 的圆，我们可以用以上的任何一个 API 来完成：

```
drawCircle(1, 2, 5)
drawArc(1, 2, 5, 0, 360)
drawEllipticArc(1, 2, 2*5, 2*5, 0, 360)
```

从画圆的功能上来说，以上 API 完全等价。但利用 drawArc 不仅可以画圆，也可以画任意角度的圆弧，因为圆是圆弧的一个特殊形式。利用 drawEllipticArc 不仅可以画圆和圆弧，也可以画各种离心率的椭圆弧线，因此应用软件开发者只要学习并记住 drawEllipticArc 一个就可以解决多种问题。仅从抽象度来分析，drawEllipticArc 是最好的选择。但是这也同时要求应用软件开发者有扎实的数学知识，知道圆，圆弧，椭圆弧线之间的抽象关系，否则他们就要花更多的时间理解并记住这一个功能。再说，如果应用软件人员要从 API 库中查询画圆的功能，他们可能只会在他们的检索需求中使用 circle 这个词，从而无法简单的找到 drawEllipticArc 或 drawArc。

2.2 基本组合

大多数的 API 提供一个基本功能，应用软件人员在编程时为完成一个任务往往要把相关的 API 进行组合。设计和评估 API 时要考虑的问题是，为完成比较典型的任务，应用软件人员必须使用怎样的 API 的组合呢？这些基本组合是否简单易用？Steven Clark 在[3]中举了下面一个程序例子，该程序段的目的时为了给用户 to@you.com 发一个电子邮件。

```
IPAddress h=IPAddress.Parse("10.0.1.1");
IPAddress a=new IPAddress(h);
CEmailServer s=new CEmailServer(a);
if (s.Connect()) {
  CMailbox mb=s.GetMailbox("joe");
  mb.Open("joe", true);
  CEmailMessage m=mb.NewMessage();
  m.Build("message",
      "subject", "foo@ppp.com");
  mb.Send(m, true);
```

```
    mb.Close();
} else
    Console.WriteLine("Connect error");
```

在上述程序段中，为了完成发一个电子邮件这样基本的任务，软件开发人员必须要完成 9 个步骤：解析 IP 地址、获取电子邮件服务器、连接电子邮件服务器、获取发信用户的邮箱、打开邮箱、创建新邮件、填入邮件内容、发送邮件、最后关闭邮件。利用这样一组 API 的设计编程不仅费时也很容易出错。考虑到获取电子邮件服务器的唯一目的是取得发信用户的电子邮箱，我们至少可以考虑提供直接从 IP 地址获取一个用户邮箱的基本功能。

下面再来看看 Java 开发平台 API 的一个例子。从标准输入读入是程序中常用的功能，而在 Java 中，一个程序员必须要用下列的基本组合。

```
InputStreamReader isr=
    new InputStreamReader(System.in);
BufferedReader in=
    new BufferedReader(isr);
String str = "";
while (str != null) {
    str = in.readLine();
    //对 str 进行处理
```

考虑到读入输入是初学 Java 的程序员一开始就要用到的典型任务，这一组的 API 设计的可用性很有提高的余地。当然，Java 文件处理 API 对各种的输入设备进行了统一和抽象，从而使 Java 的输入处理程序不再依存于输入的物理设备，提高了 Java 的可移植性，但是它在可用性方面存在的缺陷也是很明显的。

2.3 透明度

如果应用软件开发者凭借 API 的名字和使用方式就能准确地理解该 API 的功能和意义，那么该 API 就具有良好的透明度。良好的透明度可以增加 API 的易学、易记、易懂的程度。

为提高 API 的透明度，设计时应尽量减少模式(mode)的使用。在一个界面中，如果用户执行同样的动作但得到不同的结果，那么这样的界面就存在模式现象。由于模式现象的存在，用户执行的结果依赖于系统当时的状态，在不同的状态下，会得到不同的结果，很容易造成误解和不解。同样的，如果 API 的执行结果依赖于对象所处的状态，它就有了不透明的模式，增加了学习和记忆的内容。比如说，Java API 中的 FileWriter 有下列的构造方法

```
public FileWriter(String fileName)
```

当 fileName 指定的文件不存在时，它就创建一个新的文件，如果文件已存在，它就重写该文件。如果程序员在编程时不仔细考虑，在没有测试文件是否存在时就调用 FileWriter("test")的功能，那么如果执行该程序的用户恰巧有一个同名的文件，他的文件就会被重写了。

在同一个类中还有另一个构造方法

```
public FileWriter(String fileName, boolean append)
```

即如果在文件名后再加上一个 true 的值，该方法就在文件的后面添加内容。这一构造方法的执行同样地依赖于系统的当前状态：如果文件不存在，不论 append 的值是 true 还是 false，它都会创建一个新文件；如果文件存在，那么当 append 的值为 true 时，它就在文件的末尾添加新的内容，而当 append 的值为 false 时，它就重写该文件。这一个构造方法除了上述的模式问题以外，还有一个影响透明度的问题：降低了它的易读性。当一个对 FileWriter 不熟悉的程序员在程序中读到这样的语句

```
out = FileWriter("test", true)
```

时就不能马上理解这一语句的意义。

我们可以考虑采用这样的方式提高这一个 API 的透明度。首先定义一组常量

```
public static final int NEW
public static final int OVERWRITE
public static final int APPEND
```

然后把 FileWriter 改写为

```
public FileWriter(String fileName, int openMode)
```

那么该 API 的使用方法就变成了

```
FileWriter("test", NEW)
FileWriter("test", OVERWRITE)
```

FileWriter("test", APPEND)

由于透明度的增加，这样的 API 可减少出错的机会，并增加其易读性。

2.4 一致性

API 的设计应遵循相同的逻辑和概念，保持高度的一致性。一致性可直接提高知识的普遍性，学了一组即可举一反三地用于推测其他 API 的功能和使用方法。

一致性首先要求在命名方式上的统一。比如说，所有的遵循工厂模式(factory pattern)设计的 API 都应有 factory 做词缀，让人一目了然。类似的操作要有类似的命名方式。如果一个 API 里同时有名为 fileOpen, closeFile 的操作混在，应用软件开发者就很容易出错。在 Java 开发平台的 API 中，下列所有的方法都是返回元素的个数，却用了不同的命名方式：

Java.net.DatagramPacket int getLength()

Java.io.File long length()

Java.awt.Choice int countItems()

Java.awt.Container int countComponents()

Java.awt.List int countItems()

Java.awt.Menu int countItems()

Java.awt.MenuBarint countMenus()

Java.awt.Polygon int npoints

这些名字的混在增加了应用软件人员要记忆的内容，降低了 API 的可用性。

另外需要注意的是 API 的变量名的顺序要一致。比如说在 Java 开发平台的 API 中有下列两个方法

public void setCharAt(int index, char ch)

public void setElementAt(Object obj, int index)

两者都是对 index 指定的位置赋值，但它们的参量的位置却不同，很容易造成混乱。当相类似功能的 API 的参量的个数不一样时要特别注意。比较

File.write(File buffer, int writeSize);

File.read(int offset, File buffer, int readSize);

和

File.write(File buffer, int writeSize);

File.read(File buffer, int readSize, int offset);

就不难发现前者更难记忆、更易出错。

2.5 自然对应

为提高 API 的可用性，设计时必须要从应用软件开发人员的角度考察问题，因为他们是 API 的用户。尽管 API 的设计人员和应用软件开发人员都是软件开发人员，但两者的知识结构不一样，对问题的看法和术语的使用也会各有不同。设计 API 时提供的功能要和应用领域的功能之间有自然的对应关系，API 的命名和描述要用应用软件人员熟知的语言和概念，尽量提高 API 本身的示能性(affordance)。示能性是可用性研究中的一个重要概念，一个界面或物体的示能性指界面或物体本身的外观和形态所传递的功能和使用方法。比如说，如果一个门把手是圆的，大多数人会不假思索地去旋转把手开门。但是如果用了一个圆把手，却要往上扳才能开门，这个门把手的示能性就很不好，因为它的外表所传递的使用方法和实际的使用方法之间没有自然的对应关系。

具有良好示能性的 API 易学，易懂，易记。应用软件人员在很多情况下可以不用查询文档就可理解 API 的功能和用法。API 设计人员最容易忽略的一点是把 API 设计和系统设计当作同样的问题，直接把系统设计的类和方法当作 API。API 设计和系统设计有不同的要求。在设计系统时，设计者关心的是系统开发的时间和设计本身的工程上的合理性。比如说设计者经常会利用 refactoring(设计重构)把重复的代码抽出，形成一个可复用的模块。这些抽出的模块往往并不是问题领域里的一个基本功能，和应用软件人员所关心和面临的开发工作没有自然的对应。

软件设计人员在进行系统设计时经常使用设计模式(design pattern)。各种各样的设计模式，特别是在{Gamma, 1994 #151}书中提出的各种模式已成了面向系统设计的基本词汇和技巧。这些模式也经常直接反映在 API 中。比如说 Java 1.5 版有 61 个工厂模式类和界面，.Net 的 API 也有 13 个类使用了工厂模式。但是，这些从使用 API 的应用软件开发人员的角度来看，直接基于设计模式的 API 不一定会有良好的可用性，因为创建这些设计模式时只考虑了其对系统的设计和编码方面的优点。以工厂模式为例，Brian

Elllis[4]等人的研究发现，应用软件人员发现在 API 设计中广泛使用的工厂模式(factory pattern)难学难用，因为工厂模型过度抽象，在实际的应用领域里没有自然对应的概念。利用 Smalltalk 引入的类簇(class cluster)的技巧[5]同样可以实现大多数的工厂模式，可用性比较实验表明应用软件人员发现类簇更容易理解学习。

因此，编程设计和 API 设计应分开考虑，不能把模块设计时得到的类结构直接地转成 API。

3 提高API可用性的方法

和所有的可用性问题一样，提高 API 可用性没有灵丹妙药，只能采取一个渐进反复的方法，同时兼顾设计和评估。提高可用性不能在设计完成之后才加以考虑，期望通过用户测试的手段发现并纠正问题。大多数可用性的问题不是表面上的，而是构造性的，如上文中提到的工厂模式的可用性问题，这些问题无法通过对界面的调整而得到纠正，必须要在设计的初级阶段就要进行系统的考虑。

3.1 设计可用的 API

可用性不是一个绝对的概念，而是一个相对于用户的概念，所以设计可用的 API 首先要界定目标的用户群，即决定谁将是 API 的主要用户。

然后通过观察、访谈、问卷等方式选定或构件典型的用户模型。如果可能的话，用对用户已开发的程序进行分析，了解他们现有的知识结构和概念结构。比如说如果这些应用程序员的程序中已经大量的使用了工厂模式，那么工厂模式的可用性就不再是一个主要问题。

列出 API 需要支持的主要任务，并对任务进行分析，大致估算各个任务的出现频率。根据这些估算对任务进行分解成为基本任务和基本组合任务，尽可能使常见的任务可以用最简短的方法实现。

另一个需要考虑的问题是，应用软件开发人员将以什么样的频度来使用这些 API：他们会在每天的编程中使用，还是偶尔调用？比如说，大多数的 Java API，特别是 java.lang 中的 API 是每一个 Java 程序员几乎天天都要使用的东西，那么易学易记可能不如有效性更重要。但是如果应用软件人员只是偶尔使用一个 API，如 google 的 API，那么易学易记就是更重要的可用性目标。

3.2 可用性的评估

在完成 API 的初步设计后，可以采用三种方法对 API 的可用性进行评估：API 审评、API 使用经验审评和用户测试。

API 审评和常用的代码审评类似，由可用性专家审查所有的 API 设计，根据 API 可用性的各个要素对现有的设计进行考察分析，提出改进的地方。这一个形式特别有助于发现 API 设计中不一致的地方。

API 使用经验审评需召集一组软件开发人员，让他们利用设计好的 API 编制一些简单的程序，然后再开会一起讨论各自的开发经验。这些参加评估的软件开发人员既可以是与 API 设计开发无关的人员，也可以是 API 设计和开发的人员。让 API 设计和开发人员参与这一评估活动可加深他们对可用性的感性认识。API 使用经验审评应在 API 的开发完成之前进行。这一活动时编制的简单程序还可以兼任测试程序的功能，一举两得。

用户测试要在可用性实验室里进行，通过观察用户执行预定任务来评估产品的可用性。API 的可用性测试需要分成两类：编程测试和理解测试。进行编程测试，参与的应用软件开发人员需要完成一些预先指定的编程任务。用户观察可以通过录像、在场观察并纪录、屏幕纪录等方式完成，再辅以实验结束后的用户访谈和问卷调查。结合对观察到的现象的分析、访谈和问卷的结果、以及用户编制的程序的正确性和编制时所用的时间等种种要素的综合考虑来评估 API 的可用性，提出修改的建议。参与理解测试的软件开发人员需阅读 API 编制的程序，或者说出其功能，或者对其进行修改维护。这一测试方式最好要求用户采用有声思维(think aloud)即在思考时自言自语把思考的内容说出来。

用户测试不仅可以发现 API 设计中存在的可用性问题，也可以用来帮助决定选择多个设计方案中的一个。比如说，类的初始化可以用两种方式来实现。第一种方式是先不用任何参量调用类的构造方法，然后再给对象的各个属性赋值，如

```
FooClass foo = new FooClass();
foo.Bar = barValue;
foo.Use()
```

第二种方法是利用参量直接初始化对象， 如

```
FooClass foo = new FooClass(barValue);
foo.Use()
```

在微软开发人员中进行的用户测试表明，大多数开发人员认为第一种方法的可用性更好一些，更愿意使用第一种方法[6]。这一用户测试结果提醒 API 设计人员在设计 API 时应该考虑提供无变量的初始化方式。

4 结语

应用软件开发人员是 API 的用户，因此 API 也存在可用性的问题。考察 API 的可用性要分析它是否有合适的抽象度、合理的基本组合能力、良好的透明度、高度的一致性以及与用户概念和思维方式的自然对应。提高 API 的可用性要采取设计和评估齐头并进，在了解用户后完成初步设计，并通过 API 审评、使用经验审评和用户测试等三种评估手段对设计进行评估反馈，提高 API 的可用性。

参考文献

1 Nielsen J. Usability Engineering. 1993, San Francisco, CA:Morgan Kaufmann.

2 Green TRG. Instructions and Descriptions: Some Cognitive Aspects of Programming and Similar Activities. Proceedings of Working Group on Advanced Visual Interface (AVI2000), Di Gesù V, Levialdi S, Tarantino L. Editors. 2000, ACM Press: New York:21－28.

3 Clarke S. Measuring API Usability. Dr. Dobb's Journal, 2004,(5):S6－S9.

4 Ellis B, Stylos J, Myers B. The Factory Pattern in API Design: A Usability Evaluation. Proceedings of 2007 International Conference on Software Engineering, 2007:302－312.

5 Cocoa Fundamental Guide: Class Clusters.

6 Stylos J, Clarke S. Usability Implications of Requiring Parameters in Objects' Constructors. Proceedings of 2007 International Conference on Software Engineering, 2007:529－539.