API 设计的小手册

诺基亚公司 Jasmin Blanchette Trolltech

2008 年 6 月 19 日

内容

3.

第一章

介绍

应用程序编程接口(application programming interface，简称 API)是一组导出的符号，供库的用户编写应用程序时使用。api 的设计可以说是库设计中最关键的部分，因为它会影响构建在它们之上的应用程序的设计。引用 Daniel Jackson[1] 的话:

> 软件是建立在抽象之上的。选择正确的抽象，编程就会自然而然地从设计中产生;模块会有小而简单的界面;新的功能将更有可能适应，而无需进行广泛的重组。选错了，编程将会是一系列令人讨厌的惊喜。

本手册收集了在 Trolltech(现在是诺基亚的一部分)的 Qt 应用程序开发框架上，通过多年的软件开发发现的 API 设计的关键见解。在设计和实现库时，除了纯粹的 API 考虑因素外，你还应该记住其他因素，例如效率和实现的易用性。而且，虽然重点是公共 api，但在编写应用程序代码或内部库代码时，应用这里描述的原则并没有什么坏处。

Qt ! Qt 历史中的例子以这样的块形式呈现。如果你不熟悉 Qt，你可能会发现这些例子有些晦涩。你可以毫不犹豫地向你的同事询问细节。此外，许多例子来自我工作过的课程，原因很简单，我最了解那些课程。其他课程也可以提供同样好的例子。

鸣谢 首先我要感谢 Lars Knoll，他鼓励我写这本手册，并在写作过程中给予反馈。我还要感谢 Frans Englich, Andreas Aardal Hanssen, Simon

1Daniel Jackson, Software Abstractions，麻省理工学院出版社，2006 年。

Hausmann、Mark Summerfield、Jan-Arve Sæther 和 Trenton W. Schulz 审阅了本手册的手稿，以及与我进行过卓有成效的 API 讨论的所有 Trolltech 开发人员。

第二章

优秀 api 的特征

什么是好的 API?虽然这个概念本质上是主观的，但大多数库开发者似乎都同意 API 的主要可取特性。以下列表的灵感来自 Joshua Bloch[1] and 的一次演讲和 Matthias Ettrich[2] 的一篇文章:

- **易学易记**

  代码可读性强

- **不易误用**
- **易于扩展**
- **完成**

注意“简约”和“一致”没有出现在列表上。他们不需要;臃肿、不一致的 API 会很难学习和记忆，也很难扩展。我们应该努力做到极简和一致，但仅限于它们有助于满足上面列出的标准。

“一致性”与“概念完整性”大致一致，这一原则是一个复杂系统应该有一个连贯的设计，反映了一个建筑师的愿景。几十年前，弗雷德里克·布鲁克(Frederick Brooke)写道 [3]:

> 我认为概念完整性是系统设计中最重要的考虑因素。最好是让一个系统忽略某些反常的特征和改进，但反映一套设计思想，而不是让一个系统包含许多好的但独立的、不协调的思想。

1Joshua Bloch，“如何设计一个好的 API 以及为什么它很重要”，谷歌 Tech Talk，2007 年。可在 http://video.google.com/videoplay?docid=-3733345136856180693 获取。

2Matthias Ettrich，《设计 Qt 风格的 c++ api》，Qt 季刊第 13 期，2004 年。可在 http://doc.trolltech.com/qq/qq13-apis.html 下载。

3 小弗雷德里克·p·布鲁克斯，《人月神话:软件工程论文集》，Addison-Wesley，1975 年。1995 年第 2 版。

顺便说一句，如果你足够幸运，正在扩展一个已经展示了上面列出的优点的库，模仿是你通往成功的快车道。

2.1 易学易记

易学的 API 具有一致的命名约定和模式，概念的经济性和可预测性。它对相同的概念使用相同的名称，对不同的概念使用不同的名称。

最小的 API 很容易记住，因为要记住的东西很少。一致的 API 很容易记住，因为你可以在使用不同的部分时，重新应用你在 API 的一个部分中学到的东西。

一个 API 不仅仅是组成它的类和方法的名称，还包括它们预期的语义。语义应该简单明了，并遵循最少意外的原则。

有些 api 很难学习，因为它们需要用户编写大量的样板代码才刚刚开始使用。对于新手用户来说，没有什么比创建他们的第一个 widget、3D 场景或数据模型更令人生畏的了;也没有什么比这方面的失败更能破坏他们的自信了。一个易于学习的 API 使我们可以只用几行简单的代码就写出“hello world”示例，并逐步扩展它以获得更复杂的程序。

Qt !QPushButton 是一个低门槛类的例子。你只需实例化它，并对其调用 show()，瞧，你就有了一个拥有自己的窗口框架和任务栏条目的 QPushButton。回想起来，如果可以省略对 show()的调用(例如，在事件循环的下一次迭代中显示 widget)，并且 QApplication 对象是由 Qt 隐式创建的，那么“Hello world”将会是

```
# include < QtGui >

Int main() {

    QPushButton 按钮(“Hello
    world”);返回 QApplication:运行();

}
```

便利方法——用其他公共方法实现(或可以实现)的方法，以使某些习语更短，便于编写——导致 API 膨胀。然而，只要它们被清楚地记录为“方便”并与 API 的其他部分很好地配合，它们是可以接受的，并且确实被鼓励使用的。例如，如果你提供了一个 insertItem(int, Item)方法，该方法接受一个索引和一个 Item，那么通常也提供 addItem(Item)作为方便也是有意义的。

2.2 使代码更易读

编程语言或库可能很容易学习和记忆，但却会导致完全不透明的代码。APL 语言是一个极端的例子;表达式(~*R*∈*R*∘ .×*R*)/*R*←1↓⍳*R*返回到 R 的素数列表。更 [4] 接近我们的是，Perl 因其神秘的语法而获得了“只写语言”的声誉。

应用程序代码只写一次，但在应用程序的生命周期中，不同的开发人员会反复阅读。可读性强的代码更容易记录和维护。它也不太可能包含 bug，因为代码的可读性使 bug 更明显。

Qt !在 Qt 3 中，QSlider 的构造函数允许你指定各种属性:

```
slider = new QSlider(8,128, 1,6, Qt::Vertical, 0 ,
                                “volume” );
```

在 Qt 4 中，你会写

```
slider = new QSlider(Qt::Vertical);
滑块- > setRange (128);

滑块- > setValue
(6);

滑块- >setObjectName( “卷” );
```

这更容易阅读，甚至更容易编写，没有文档或自动补全。也更容易看出，6 的值位于范围[8,128]之外。

可读的代码可以简洁，也可以冗长。无论哪种方式，它总是处于正确的抽象级别——既不隐藏重要的东西，也不强迫程序员指定无关的信息。

Qt !Qt 占比代码

```
QGridLayout 布局=新 QGridLayout;布
局。addWidget(滑块， 0,0);布局。
addWidget(spinBox, 0,1);布局。
addWidget(resetButton, 2,1);布局。
setRowStretch (1,1);

setLayout(布局);
```

比等价的 Swing 代码可读性更高:

```
GridBagLayout layout = new GridBagLayout();

GridBagConstraints 约束=新的 GridBagConstraints();约束。
fill = GridBagConstraints.HORIZONTAL;

约束。insets =新 insets (10,10,0);

约束。权重 x= 1;
```

[4] 维基百科，“APL 编程语言”。可在 http://en.wikipedia.org/wiki/APL_programming_language 下载。

```
布局。setConstraints(滑块、约束);

约束。gridwidth = GridBagConstraints.REMAINDER;
约束。插图=新插图(10,5,10,10);约束。Weightx = 0;
布局。setConstraints(转轮、约束);

约束。锚= gridbagconstraints .东南;约束。fill =
GridBagConstraints.REMAINDER;约束。插图=新插
图(10,10,10,10);约束。分量= 1;
布局。setConstraints (resetButton、约束);

JPanel 面板=新 JPanel(布
局);panel.add(滑块);
panel.add(转子);
panel.add (resetButton);
```

2.3 不易误用

设计良好的 API 使编写正确的代码比错误的代码更容易，并鼓励良好的编程实践。它不会不必要地强迫用户严格按顺序调用方法，或者意识到隐式副作用或语义上的奇怪之处。

为了说明这一点，让我们暂时离开 api 的世界，来考虑一下 HTML、纯 TEX 和 LATEX 的语法。下表展示了一个使用这三种标记语言的简短示例:

```
结果:    goto 标签 语句
HTML:    <b>goto <u>label</u></b>语句{\bf goto
Plain TEX:  \underline{label}}语句\textbf{goto
LATEX:   \underline{label}}语句
```

因为 HTML 迫使我们在结束标签中重复标签的名称，它鼓励我们写这样的东西

```
<b>goto <u>label</b></u>语句
```

这是严重的嵌套和非法的。Plain TEX 没有这个问题，但它也有自己的缺陷。很多 TEX 的使用者，包括本文档的作者，经常会发现自己键入类似这样的东西

```
\bf{goto \underline{label}}语句
```

结果是“the **goto**__label__ **statement**”，即“statement”这个词，以及接下来直到下一个右大括号的所有内容，都是粗体。(这是因为\bf 命令适用于当前作用域，而\underline 的

Effect 受限于它以{}分隔的参数。)TEX 的另一个奇怪之处是粗体和斜体是互斥的。因此,

```
{\bf goto {\it label\/}}语句
```

生成“the **goto** *label* 语句”。第三个怪癖是，我们必须记得在斜体文本之后插入斜体校正字偶距(\/)。LATEX 解决了所有这些问题，代价是引入了更长的命令名(\textbf、\textit 等)。

API 设计，在很多方面，就像标记语言设计。API 是一种语言，或者更确切地说，是编程语言的扩展。下面的 c++代码反映了 HTML 示例及其缺陷:

```
流。
writeCharacters( “ ” );stream.wri
teStartElement (" b ");流。
writeCharacters( “ goto ” );stream
.writeStartElement( “ 我 ” );strea
m.writeCharacters( “ 标
签 ” );stream.writeEndElement(
“ 我 ” );
stream.writeEndElement (" b
");
流。 writeCharacters( “ 声 明 ” );
```

如果我们能让 c++编译器检测出不好的嵌套，不是很好吗?利用表达式的力量，可以这样做:

```
流。 写(文 本( “ 的 ” )
        +元素( “ b ” ,
                文本
                ( “ goto ” )
                +元 素( “ u ” ， “ 标
签 ” ))+文 本( “ 声 明 ” ));
```

虽然上面的代码片段关注的是标记文本的生成，但同样的问题也会发生在其他上下文中，比如配置文件，设置可能会嵌套在组中:

```
QSettings 设置;
settings.beginGroup( “ 主窗口 ” );
设置。 setValue( “ 大 小 ” ,赢 得- ＞大
小());
设 置。 setValue( “ 全 屏 ” ,赢 得- ＞ isFullScreen
());settings.endGroup ();
```

还有一个教训可以从标记语言中借鉴，特别是 HTML。HTML 的作者希望我们使用(强调)

一位 5A 读者评论道:“这个例子很奇怪。实际上我觉得 HTML 比 TEX/LTEXA 更容易阅读。”也许是这样。但这一节的标题是“难以误用”，而不是“导致可读代码”。

代替<i>和代替<b>。不过，皱眉头的<i>和<b>标签，打字更方便，语义也更清晰(强调是指斜体还是粗体?)如果他们是认真的，他们会用<font style="italic">替换<i>，粗体也同样替换，或者干脆省略这两个标签。

Qt !Qt 3中一个常见的陷阱是在创建对象时交换初始文本和 QPushButton、QLabel或 QLineEdit 的父元素。例如:

```
button = new QPushButton(这个，“Hello world”);
```

代码编译完成，但 widget 并没有显示任何文本，因为“Hello world”被当成了对象的名称。在 Qt 4 中，这是在编译时捕获的。如果你想指定一个对象名，你必须写

```
button = new QPushButton(this);
按钮->setObjectName(“Hello world”);
```

非常明确地表达了你的意图。

最后，我们通常可以通过消除冗余来防止 API 误用。例如，一个addItem(Item)方法，让用户写 obj.addItem(yksi);

```
obj.addItem (kaksi);
obj.addItem (kolme);
```

比更通用的 insertItem(int,Item)方法更难误用，后者鼓励差一错误:

```
//错误
obj 。 insertItem (0, yksi);obj 。 kaksi insertItem(1 曰 );obj 。 kolme insertItem (3);
```

2.4 易于扩展

图书馆随着时间的推移而增长。新的类出现。现有类获得新方法，方法获得新参数，枚举获得新枚举值。api 设计时应该考虑到这一点。API 越大，新概念和现有概念之间的冲突就越可能发生。此外，在最初的设计阶段，必须考虑二进制兼容性问题。

Qt !QStyle 是一个很好的例子，在 Qt 2 中很难以二进制兼容的方式扩展类。它有一整套用于绘制 Qt 中不同部件的虚函数，这使得在不破坏二进制兼容性的情况下无法为新的部件(例如，QToolBox)设置样式。在 Qt 3 中，

QStyle 的 API 被重新设计为基于枚举的，这样新的部件就可以通过添加枚举值来支持。

2.5 完成

理想情况下，一个 API 应该是完整的，让用户做他们想做的一切。几乎不可能(如果有的话)为任何事情提供一个完整的 API，但至少应该允许用户扩展或自定义现有的 API(例如，通过子类化)。

完整性也是可以随着时间的推移而出现的东西，通过逐步向现有的 api 添加功能。然而，即使在这些情况下，对未来的发展方向有一个清晰的想法通常也是有帮助的，这样每一个添加都是朝着正确方向迈出的一步。

第三章

设计过程

api 通常是跨越数年、涉及许多人的设计过程的结果。这个过程中的每一步都提供了改进 api 的机会——或者破坏它。本章提供了一些设计新 API 或扩展现有 API 的指导。通过遵循这些指导方针，你可以增加 API 成功的机会，而不是成为负担。

3.1 了解需求

在开始设计或实现 API 之前，你应该对需求有一个很好的了解。有时，需求是相当明确的(例如，“实现 POSIX”)，但在大多数情况下，你将需要进行某种需求分析。一个好的起点是询问尽可能多的人(特别是你的老板、同事和潜在用户)他们希望看到什么功能。

Qt !在为 Qt 4.3 设计新的 MDI 类时，主要实现者给内部开发人员的邮件列表写了一封电子邮件，询问关于之前的 MDI 框架已知的问题以及对新框架的任何愿望。除了帮助定义 API 和功能集之外，这确保了其他开发人员有他们的发言权，并防止了以后的纠纷。

3.2 在编写任何其他代码之前编写用例

在编写库组件时，一个常见的错误是先实现功能，然后设计 API，最后发布。这样设计的 API 往往反映的是底层代码的结构，而不是将要使用 API 的应用程序程序员想要的

写作。实现应该适应用户，而不是反过来。

在你编码一个实现或者甚至设计一个 API之前，首先要根据你的需求列表编写一些典型的应用程序代码片段。在这一阶段不要担心在实现 API 时可能出现的困难。当你编写代码片段时，API 就成型了。代码片段应该反映 Perl 的座右铭“使简单的事情变得容易，使困难的事情成为可能”。

Qt !对于 QWizard，确定的两个主要场景是线性向导和复杂向导。在线性向导中，页面是顺序显示的，从来没有跳过任何页面;在一个复杂的向导中，根据用户的输入，不同的遍历路径是可能的，描述了一个有向无环图。虽然线性的情况可以看作是复杂情况的一种特殊情况，但它足够常见，值得特别考虑。

3.3 在同一个库中寻找类似的 api

在设计一个名为 XmlQuery 的类时，查看同一个类库中已有的 SqlQuery。它们很可能有相似的概念——你执行或预编译一个查询，导航结果集，将值绑定到占位符，等等。已经知道 SqlQuery 的用户就可以毫不费力地学习 XmlQuery。此外，通过模仿现有的 API，您可以隐式地受益于进入该 API 的所有增量设计工作及其收到的反馈。

当然，仅仅因为已有的 API 就盲目地遵循它是愚蠢的。大多数现有的 api 都可以做一些改进，完全与它们保持一致可能会变成一把双刃剑。话虽如此，要想在一个糟糕的 API 上实现一致性，一个极好的方法就是一劳永逸地修复那个 API，然后模仿它。

在大型库中找到类似的 api 并不总是像上面的 XmlQuery/SqlQuery 案例那样容易。它需要耐心、好奇心，以及 grep 等强大的搜索工具。

如果你正在编写一个新版本的 API，你应该对你正在替换的 API 了如指掌，否则，你最终将以新的设计缺陷代替旧的缺陷。你可能会很容易相信旧的 API 是“完全的垃圾”，不值得你花时间，但不太可能是完全错误的，所以尝试采纳任何好的想法。此外，由于用户将需要移植为旧 API 编写的代码，避免不必要的更改，并支持旧功能的超集。

Qt !Qt 4.0 中可疑变化的例子:QDockWindow 被重命名为 QDock- Widget(即使每个人都说“dock window”)和 QTextEdit:: setOverwriteMode()被删除，只在 Qt 4.1 中重新出现。

3.4 在实现之前定义 API

同样的原因，你应该在定义 API 之前针对 API 编写代码片段，你应该在实现它之前指定 API 及其语义。对于一个拥有数千用户的库来说，如果实现比较棘手，API 比较直接，那要比反过来好得多。

Qt ! Qt 4 允许用户在任何时候通过调用 setParent()来指定 QWidget 的父组件。在 Qt 4.0 和 4.1 中，创建一个无父组件会导致在后台创建一个窗口，这是非常昂贵的。在 Qt 4.2 中引入的延迟窗口创建，是 api 驱动的实现改变的一个漂亮例子。

相比之下，Qt 3 中的方法 QWidget:: recreage()则体现了实现驱动 api 的弊端。诚然，在大多数平台上，QWidget 都创建了一个原生的 widget 或窗口并将其包装起来，但这个实现细节从来都不属于 API。

API 及其语义是库交付的主要产品。经验一次又一次地表明，api 比它们的实现(例如，UNIX/POSIX, OpenGL 和 QFileDialog)更持久。

当你实现 API 或为你的实现编写单元测试时，你很可能会在原始设计中发现缺陷或未定义的边角情况。利用这个机会来完善你的设计，但不要让实现方面的考虑泄露到 API 中，除了在例外情况下(例如优化选项)。

Qt ! QGraphicsScene::setBspTreeDepth()可能构成了这样的例外。这种方法让用户控制用于存储图形场景项目的二叉空间分割(BSP)树的深度。QGraphics-Scene 试图推断出一个合适的树的深度，但如果场景频繁变化，固定树的深度可能会更有效。这个方法的名字对用户来说足够吓人(他们可能甚至不知道或不关心 QGraphicsView 内部使用的是 BSP 树)，说明这是一种先进的方法。

3.5 让你的同行审查你的 API

寻求反馈。寻求反馈。寻求反馈。向你的同行展示你的 api，收集你得到的所有反馈。试着暂时忘记实现所请求的更改需要多少工作。

当收到负面反馈时，要相信所有反馈都是有用的这一原则。任何意见(例如，“这整件事糟透了”，来自 Joe Blow)都是一条新的信息，你可以把它重铸成一个事实，并添加到你脑海中的事实列表中(例如，“Joe Blow，一个 Linux 狂热分子，非常不喜欢这个

COM-inspired 架构”)。你拥有的事实越多，你就越有机会设计出好的 API。

针对 API 写几个例子

在你设计完一个 API 之后，你应该写一些使用 API 的例子。通常，你可以简单地通过充实前面定义的用例来获得示例。如果其他人使用 API 编写示例并给你他们的反馈，这也会有所帮助。

Qt !最初在尝试使用 QWizardAPI 时编写的代码片段最终成为与 Qt 提供的类向导和许可证向导示例，其中涵盖了向导的线性和复杂情况。还有一个琐碎的向导示例，它不涉及子类化，有一种“hello world”的味道。

3.7 为扩展做准备

期望你的 API 以两种方式被扩展:由 API 的维护者，他们将添加到它(偶尔会弃用它的一部分)，以及由用户，他们将编写子类来定制其组件的行为。

规划可扩展性需要仔细考虑现实情况并考虑可能的解决方案。对于每个声明虚拟方法的类，你应该至少编写三个子类(作为示例或作为 API 中的公开类)，以确保它足够强大，以支持广泛的用户需求。这有时被称为“三法则”。

Qt !Qt 4.0 引入了 QAbstractSocket，以及具体的子类 QTcpSocket 和 QUdpSocket。当在 Qt 4.3 中添加 QSslSocket 时，我们被迫在 QAbstractSocket 的一些方法中添加了 downcast，这些方法本应是虚拟的，但并没有。“手动多态”在这种情况下是可行的，因为 QSslSocket 和 QAbstractSocket 生活在同一个库中，但它对于第三方组件是行不通的。

3.8 未经审查，不要发布内部 api

有些 api 在公开之前，就开始了作为内部 api 的辉煌生活。一个常见的错误是在采取这一步之前忘记对它们进行适当的审查。Raymond chen 的故事说明[2]了这一点:

---

1 Will Tracz，“软件重用”，1995 年。下载地址:http://webcem01.cem.itesm.mx:8005/web/200111/cb00894/software_reuse.html。

2 Raymond Chen，“为什么函数 SHStripMneumonic 拼写错误?”，《旧新事物》博客，2008 年。可在 http://blogs.msdn.com/oldnewthing/archive/2008/ 05/19/8518565.aspx 下载。

如果你在 MSDN 上闲逛，你可能会遇到 SHStripMneumonic 函数。正确的拼写是助记符。为什么函数名拼错了？

“我来的时候就是这样的。”

这个函数最初只供内部使用，写函数的人把这个词拼错了。尽管如此，由于这是一个内部函数，所以并没有真正的紧迫性去修复它。

[……]

2001 年，命令下来记录所有符合特定标准的功能(具体细节我就不讲了，如果别人不尝试的话，我会很感激)，SHStrip-Mneumonic 功能在它还没来得及梳头发、确保牙齿上没有食物卡之前，就突然被推到了公共舞台上。这个函数必须被记录下来，毫无瑕疵，拼写错误就是其中一个瑕疵。

当然，现在这个函数已经发布了，它的名字是锁定的，无法更改，因为那样会破坏所有使用原始拼写错误的程序。

3.9 有疑问的时候，就把它删掉

如果你对某个 API 或功能有疑问，你可以把它排除在外，或者把它标记为内部，稍后再重新考虑。

等待用户的反馈通常会有所帮助。另一方面，要把用户建议的所有功能都加进去，实际上是不可能的。一个很好的经验法则是，等到至少有三个用户独立要求某个功能后再去实现它。

第四章

设计指导方针

本章介绍的设计准则旨在帮助你正确使用 api。你可以在前一章中强调的过程的不同阶段参考它们。这些指导方针是在 Trolltech 或其他地方开发的实际 api 中发现的。

对于许多指南，我们可以制定一个同样正确(但较少被违反)的反指南。例如，从准则 4.8“避免让你的 api 过于聪明”中，我们得到了“避免让你的 api 过于愚蠢”的反准则，从准则 4.9“注意边缘情况”中，我们得到了“关注一般情况”的反准则。需要考虑许多相互冲突的需求，正是 API 设计如此有价值的原因。说到底，指导方针并不能代替动脑筋。

命名

4.1 选择易于理解的名字和签名

为了方便阅读和编写代码，选择自解释的名称和签名。名字应该读起来像英文。

Qt !QPainterPath 的主要作者推荐在文档中写“矢量路径”而不是“画家路径”，因为“其他人都叫它矢量路径”。那么 Qt 为什么不叫它 QVectorPath 呢?

在 Qt 4.2 之前，另一个例子是 QWorkspace，它实现了 MDI(多文档接口)。为什么 MDI 没有出现在名字里?值得庆幸的是，取代它的新类叫做 QMdiArea。

一个方法的参数的含义应该从调用点的上下文中看得很清楚。注意布尔型参数，这通常会导致代码不可读。

Qt !QWidget::repaint()接受一个可选 bool 参数，控制 widget 在重绘之前是否应该被擦除。repaint(false)这样的代码让一些读者怀疑这个方法到底有没有做任何事情(“不要重绘!”)。至少，这个参数应该是一个枚举。

保持命名的一致性。如果你交替使用“部件”和“控件”这两个术语，用户就会被误导，以为它们是有区别的。而如果你对两个不同的概念使用相同的术语，用户会更加困惑，直到他们阅读文档。在确定参数的顺序时，一致性尤为重要。如果矩形在 API 的某个部分被描述为(*x, y, width, height*)，那么它们不应该在其他地方变成(*x, width, y, height*)。

Qt !在 Qt 4.0 的预备阶段，QStackArray 实现了一个存储在调用栈上的向量，很像 C99 变长数组(VLAs)。它到底出了什么问题?Qt 已经有了一个 QStack 容器类，实现了 FIFO (first in, first out，先进先出)列表。所以 QStackArray 听起来就像一个 FIFO 列表的数组，任何了解 QStack 的人都知道。为了避免“stack”这个词重载，我们将这个类重命名为 QVarLengthArray。

好的命名还需要了解受众。如果你为 XML 定义了一个强大的 API，那么你肯定需要在 API 名称中使用 XML 术语。(你也应该在文档中定义它，并提供相关标准的链接。)相反，如果你正在为 XML 设计高级 API，不要指望你的读者知道 IDREFs 和 NCNames 是什么。

提供好的参数名也是设计 API 的一部分，即使在像 c++这样的语言中，它们并不是签名的一部分。参数名在文档和一些开发环境提供的自动补全工具提示中都有体现。重命名命名不佳的参数是乏味且容易出错的，所以要像关注 API 的其他方面一样关注参数命名。尤其要避免使用所有的单字母参数名。

4.2 为相关事物选择明确的名称

如果需要区分两个或两个以上的相似概念，选择与它们所表示的概念无歧义映射的名称。更具体地说，给定一组名称{n_1，…， n_k}和概念集合{c_1，…， c_k}，任何人都应该很容易将这些名称与正确的概念联系起来，而无需在文档中查找。

1 例外的是，x、y 和 z 作为平面坐标的名字总是优于它们学究气的表亲——横坐标、纵坐标和应用。

假设你有两种事件交付机制，一种用于立即(同步)交付，另一种用于延迟(异步)交付。sendEventNow()和 sendEventLater()的名字暗示了自己。如果期望受众知道“同步”的术语，你也可以叫他们 sendeventsynchronousdy()和 sendEvent- asynchronous()。

现在，如果你想鼓励你的用户使用同步交付(例如，因为它更轻量级)，你可以将同步方法命名为 sendEvent()，并保留 sendEventLater()用于异步情况。如果你更愿意让他们使用异步交付(例如，因为它合并了相同类型的连续事件)，你可以将同步方法命名为 sendEventNow()，将异步方法命名为 sendEvent()或 postEvent()。

Qt !在 Qt 中，sendEvent()是同步的，postEvent()是异步的。保持它们分离的一种方法是观察到发送电子邮件通常比使用普通邮件发送信件快得多。不过，邮件的名称本可以更清楚。

类似的原则也适用于 c++中复制构造函数或赋值运算符的形参命名。因为它与隐含的 this 参数提供了良好的对比，所以 other 是一个不错的名字。一个较差的选择是在下课后调用它:

```
//不佳
Car &Car::operator=(const Car &Car) {
    M_model =
    car.m_model;M_year =
    car.m_year;M_mileage =
    car.m_mileage;…
    返回*;
}
```

这有什么不对?这个方法处理的是两辆车:* This 和 car。在只有一辆车的情况下，car 这个名字是合适的，但当有两辆或更多的车时，它就没有多大帮助了。

4.3 谨防虚假的一致性

api 和优秀的文字一样，应该体现对称性。在他的《美式英语风格》[2]一书中，小威廉·斯特伦克(William Strunk Jr.)告诫我们要“用

[2] 小威廉·斯特伦克，《风格的要素》，1918 年。bartleby.com/141/。 可在 http://www 上在线购买。

类似形式"

> 这一原则，即平行构造原则，要求内容相似、功能相似的表达形式在外观上要相似。形式的相似使读者更容易识别内容和功能的相似。《圣经》中常见的例子有十诫、福度和主祷文的请求。

这一规则的另一面是，功能的不对称应该通过形式的不对称来反映。因此，如果你习惯在 setter 方法前加上“set”(例如 setText())，那么就应该避免为非 setter 方法加上这个前缀。

> **Qt !**在 Qt 的历史上，这一准则不时被违反。Qt 3 方法 QStatusBar::message(text, msecs)在状态栏中显示一条消息，时间为指定的毫秒数。名称看起来像 getter，即使这个方法是非 const 并且返回 void。对于 Qt 4，为了“一致性”，我们考虑将其重命名为 setMessage()。然而，由于有两个参数，这个方法感觉不太像 setter。最后，我们选择了 showMessage()这个名字，这个名字并不意味着错误的类比。

前面指南中的事件传递例子在这里也可以作为说明。在 c++中，由于同步机制立即交付事件，事件通常可以在栈上创建并作为常量引用传递，如下所示:

```
KeyEvent 事件(KeyPress, key,
state);sendEventNow(事件);
```

相比之下，异步机制需要使用 new 创建对象，对象在交付后被删除:

```
KeyEvent *event = new KeyEvent(KeyPress, key,
state);sendEventLater(事件);
```

多亏了静态类型，编译器会捕捉到以下不好的惯用法:

```
//无法编译
KeyEvent *event = new KeyEvent(KeyPress, key,
state);sendEventNow(事件);

//不会编译
KeyEvent 事件(KeyPress, key,
state);sendEventLater(事件);
```

False consistency 将建议使两个方法的签名相同，尽管函数是不对称的(即 sendEventLater()获得事件的所有权，而 sendEventNow()则没有)。

Qt !在 Qt 中，我们直接走进了陷阱:sendEvent()和 postEvent()都带指针，这使得泄漏内存或 double- delete 事件非常容易。

从 API 的角度来看，更好的解决方案应该是通过让 sendEventNow()获取事件的所有权或者 sendEventLater()复制事件来恢复对称性。然而，这样做的效率较低。为了吸引神灵的青睐，我们必须在性能和易于实现的祭坛上例行牺牲纯 API 设计原则。

4.4 避免使用缩写

在 API 中，缩写意味着用户必须记住哪些单词被缩写了，在什么上下文中。因此，应该尽量避免使用缩写。对于非常常见的、没有歧义的形式，如" min "、" max "、" dir "、" rect "和" prev "，通常会有例外，但这个约定必须一致地应用;否则，用户会无意中调用 setMaximumCost()而不是 setMaxCost()，之后的一天他们会尝试调用 setMaxWidth()而不是 setMaximumWidth()。虽然这样的错误会被编译器捕捉到，但却会让用户更加恼火。

Qt !上面的例子直接来自 Qt 4 (QCache 和 QWidget)。同样，Qt 4 有 QDir 和 QFileInfo::isDir()，还有 QFileDialog::set- Directory()和 QProcess::workingDirectory()。

这个指南适用于缩写，而不是首字母缩略词。要区分这两者，在走廊里听一听:XML 叫 XML，不是可扩展标记语言，所以够格做缩略语;但 JavaScript 是 JavaScript，不是 JS。

另外，由于用户不需要输入参数名，所以在这种情况下缩写是可以接受的，只要意思清楚就行。

4.5 更喜欢具体的名称而不是通用的名称

喜欢专用名称而不是通用名称，即使它们有时看起来太受限制。记住，名字占据着 API 的不动产:一旦一个名字被取走，它就不能用于其他用途。这对于类名来说尤其是个问题，因为一个库可能包含数百甚至数千个需要不同名称的类。偏好更具体名称的另一个原因是，它们更容易与用户联系起来。

Qt !QRegExp 类提供了对正则表达式的支持，同时也支持通配符和固定字符串。它也可以被称为 QStringPattern，但这个更通用的名称隐藏了这个类在绝大多数情况下的用途，即正则表达式。出于这个原因，QRegExp 这个名字可以说是最适合这个类的。

如果需要 SQL 的错误报告机制，可以调用抽象基类 SqlErrorHandler，而不仅仅是 ErrorHandler，即使实现没有绑定到 SQL。可以说，调用 ErrorHandler 类可以促进它在 API 的其他部分的重用，但在实践中，这种重用是不可能的，因为很难预见未来的需求。有一天，你可能需要一种与 SQL 错误处理稍有不同的 XML 错误处理机制，并将其称为 XmlErrorHandler。十年之后，你可能最终会发现一种统一所有错误处理程序的方法——然后，你会感谢 ErrorHandler 这个名字仍然可用，并将它作为库的下一个版本中所有特定处理程序类的基类，那时你可以打破二进制兼容性。

Qt !多年来，Qt 为 XML 提供了两种 api: SAX 和 DOM。一个好的、对称的命名约定应该是使用 QSax 和 QDom 作为类前缀，但不幸的是，我们为 SAX 和 DOM 解决了 QXml。这种在美学上令人讨厌的不对称，在我们在 Qt 4.3 和 4.4 中引入新的 XML 类(同样使用 QXml 前缀)时变得令人困惑。看看类名，没有人告诉 QXmlReader 和 QXmlSimpleReader 是现在半废弃的 SAX API 的一部分，而 QXmlStreamReader 是 Qt 解析 XML 文档的首选方法。

4.6 不要成为底层 API 命名实践的奴隶

一个常见的需求是包装现有的 API 或实现一个文件格式或协议。这些通常都有自己的名称，可能选择得不好或与你的库中的名称冲突。一般来说，不要犹豫发明你自己的术语。例如，如果你想把 OpenGL pbuffer 扩展封装在一个类中，就叫它 GLPixelBuffer，而不是 GLPbuffer。

类似地，在将 API 移植到新的编程语言时，你应该尝试将它与你的库很好地合并，而不是受制于外来的约定。少数了解其他语言中现有 API 的用户不会介意这些改进，而剩下的用户将会从这些改进中大大受益。

Qt !Qt 中的 SAX 类是模仿 Java 事实上的标准 SAX API。为了适应 c++和 Qt 做了一些更改，但总体上 c++架构与 Java 非常相似——也非常相似

其实差不多。例如，XML 数据由一个名为 QXmlInputSource 的类提供，可以用 QIODevice 进行初始化;如果与 Qt 的集成更紧密，就会直接使用 QIODevice，而淘汰 QXmlInputSource。

语义

4.7 选择好的默认值

选择合适的默认值，这样用户就不需要编写或复制粘贴模板代码。例如，在用户界面设计中，一个好的默认值是尊重平台原生的外观和感觉。在 Mac OS X 上使用 Qt，我们简单的写

```
QPushButton *button = new QPushButton(text, parent);
```

并且我们获得了一个具有原生外观的 Aqua push button，带有指定的文本和合适的首选尺寸(它的“尺寸提示”)，随时可以插入布局管理器中。对比一下下面这段代码，使用苹果的 Cocoa 框架编写:

```
static const NSRect dummyRect = {
    {0.0, 0.0}, {0.0, 0.0}};
NSButton *button = [[NSButton alloc]
                                initWithFrame
dummyRect):;(按钮 setButtonType:
NSMomentaryLightButton);(按钮 setBezelStyle:
NSRoundedBezelStyle);
[按钮 setTitle:[NSString stringWithUTF8String:text]];(父
addSubview:按钮);
```

为什么我们需要设置按钮类型和边框样式才能得到一个普通的 Aqua 按钮?如果与 NeXTstep 和 OpenStep 的兼容性这么重要，为什么不提供一个具有正确默认值的便利类(NSPushButton 或 NSRounded- Button 或 NSAquaButton，随便什么)呢?为什么我们需要为初始化方法指定位置和大小，而我们可能稍后需要根据窗口大小和按钮报告的首选大小设置它们?

但是选择好的默认值并不仅仅是为了消除样板代码。它还有助于使 API 简单和可预测。这尤其适用于布尔选项。一般来说，将它们命名为默认为 false 并由用户启用。在某些情况下，这需要在它们前面加上 No、don 或 Anti。这是可以接受的，尽管应该寻求替代方法

在可能的情况下(例如，区分大小写而不是区分大小写)。在任何情况下，都要抵制为了炫耀而默认启用各种花哨功能的诱惑。

最后，记住，选择好的默认值并不能免除你记录它们的责任。

4.8 避免让你的 api 过于聪明

前面我们说过，API 的语义应该简单明了，并遵循最小意外原则。如果一个 API 太聪明了(例如，它有微妙的副作用)，用户就会无法很好地理解它，并且会发现自己的应用程序中有微妙的 bug。聪明还会导致代码的可读性和可维护性降低，并增加对文档的需求。

Qt !Qt 3 中的 QLabel 就提供了一个聪明的漂亮例子。setText()方法接受一个字符串，可以是纯文本或 HTML。Qt 通过检查字符串自动检测格式。这有时有点太聪明了，偶尔会导致错误的格式被选择(如果字符串来自最终用户，这是一个真正的问题)。另一方面，自动检测通常是有效的，非常方便，并且可以通过调用 setTextFormat()来覆盖，所以有人可能会说这是可以接受的。

另一方面，在 Qt 3 中用 HTML 字符串调用 setText()也有将 wordWrap 属性设置为 true(默认为 false)的副作用。反过来，wordwrap 启用了“高度换宽度”(height-for-width)，这是一种影响整个对话框的布局特性。结果，在一个标签中将“Name”更改为“Name”的用户发现自己有着完全不同的对话框大小调整行为，并且无法理解为什么。

4.9 关注边缘情况

在库编程的背景下，正确处理边缘情况的重要性怎么强调都不为过。如果你在走廊里听到同事说“没关系，这只是一个边角情况”，请给他们一记耳光。

边缘情况很关键，因为它们往往会在 API 中产生涟漪效应。如果你的基本字符串搜索算法具有糟糕的边界情况语义，这可能会导致正则表达式引擎出现 bug，而这又会导致使用正则表达式的应用程序出现 bug。但边界情况本身也很重要:没有人会对计算 0 的阶乘函数感兴趣!=0。

Qt !Qt 3 静态方法 QStringList::split(separator, string, allow- EmptyEntries = false)在每次出现分隔符时拆分字符串。因此,分离(':,” b: c”)三种列表返回(“a”、“b”、“c”)。人们可以预期，如果分隔符在 string 中出现 *n* 次，则

得到的列表将包含 *n* + 1 个元素。然而，split()在两方面违反了这一规则。首先，allowEmptyEntries 参数默认为 false。其次，作为一个特殊情况，如果 string 为空，该方法总是返回空列表，而不是单元素列表[""]。这两个问题都会导致应用程序中出现微妙的 bug，并向 Trolltech 报告 bug。

在 Qt 4 中，这个方法已经被 QString::split(separator, behavior = KeepEmptyParts, casessensitivity = Qt:: casessensitive)所取代，它并没有受到这些缺陷的影响。

如果你构建的抽象可以优雅地处理边缘情况，那么你的代码很可能也会这样做。为了处理不好的边界情况，你经常需要额外的代码。考虑下面的 product()函数:

```
double product(const QList<double> &factors) {
    //错误
    如果(factors.isEmpty
    ())
        返回 0.0;

    Double result = 1.0;
    For (int I = 0;I < factors.count();++i)
        result *= factors[i];
    返回结果;
}
```

如果传入一个空列表，该函数返回 0，这在数学上是不合理的:乘法的中性元素是 1，而不是 0。我们当然可以修正函数开头的检查:

```
if (factors.isEmpty())
    返回 1.0;
```

但是，这种检查首先是必要的吗?事实证明并非如此。函数的其余部分依赖于 c++的 for 循环和乘法运算符，它们可以优雅地处理边缘情况。空性检查是一个错误的优化——它勉强优化了已经快得让人眼花缭乱的琐碎情况，同时减慢了所有其他情况。

在实现 API 时，先处理一般情况，不用担心边缘情况，然后再测试边缘情况。通常情况下，你会发现你不需要任何额外的代码来处理边缘情况。(当你这样做的时候，这通常是一个迹象，表明底层的编程结构之一是罪魁祸首。)不过，要确保单元测试正确地覆盖了边缘情况，这样它们的语义就不会因为优化或代码的其他更改而在发布之间发生变化。

4.10 在定义虚拟 api 时要小心

众所周知，虚拟 api 很难正确使用，并且很容易在发布之间出错。这通常被称为“脆弱的基类问题”。在设计虚拟 API 时，有两个主要的错误需要避免。

虚方法定义过少的第一个错误。对于图书馆作者来说，很难预见图书馆将被使用的所有方式。通过省略方法声明前的 virtual 关键字，我们经常会降低整个类的可重用性和可定制性。

第二个错误是把每个方法都设为虚的。在 c++这样的语言中，这是低效的，但更重要的是，它传递了一个虚假的 promise。通常情况下，重新实现一些方法即使没有危险，也是没用的，因为类的实现并没有调用它们，也没有期望它们具有特定的语义。

正确的方法需要思考 API 的哪些部分应该是虚拟的，哪些部分应该是非虚拟的，并在文档中明确指定类如何使用方法本身。

Qt！官方 Qt 3book3 中的电子表格示例重新实现了虚拟方法 QTable- Item::text()。这个例子在 Qt 3.2.0 上运行得很好，但在后来的 Qt 3 版本中被破坏了，因为在 QTable 中对 text()的一些调用被替换为直接访问在默认的 text()实现中存储文本的成员变量，绕过了任何子类。

通常情况下，将类的公共 API(由应用程序直接使用)和虚拟 API(可以重新实现以自定义类的行为)干净地分开是很有效果的。在 c++中，这意味着让所有的虚拟方法都受到保护。

Qt！在 Qt 4 中，QIODevice 成功地遵循了这种方法。类的直接用户调用 read()和 write()，而子类作者则重新实现 read- Data()和 writeData()。在 QIODevice 的早期实现中，公共的、非虚拟的方法简单地称为受保护的、虚拟的方法。此后，我们添加了缓冲、换行转换、“unget”缓冲区和一次读取一个字符的快速代码路径，所有这些都在非虚拟的 read()和 write()方法中。

在某种程度上，QWidget 的事件传递机制也遵循了这种模式。类的直接用户调用 show()、resize()或 repaint()，而子类的作者则重新实现 showEvent()、resizeEvent()或 paintEvent()。

c++的一个恼人的限制是，我们不能在不破坏二进制兼容性的情况下向现有类添加新的虚拟方法。作为一个丑陋的

3 Jasmin Blanchette 和 Mark Summerfield, c++ GUI Programming with Qt 3, Prentice Hall, 2004 年。可在 http://www.informit.com/content/images/0131240722/下载/blanchette_book.pdf 下载。

变通一下，养成提供一个通用虚拟方法的习惯，以后如果需要扩展这个类，可以使用这个方法。例如:

```
Virtual void virtual_hook(int id, void *data);
```

结构

4.11 争取基于属性的 api

一些 api 强制用户在创建时预先指定对象的所有属性。许多底层平台特定的 api 都是这样工作的。下面的例子直接来自一个 Win32 应用程序:

```
m_hWindow =
  ::CreateWindow("AppWindow", /*类名*/ m_pszTitle, /*窗口
                 标题*/ WS _OVERLAPPEDWINDOW, /*样
                 式*/ CW_USEDEFAULT, /*开始 pos x */
                 CW_USEDEFAULT, /*开始 pos y */
                 m_nWidth, /* width */ m_nHeight, /* height
                 */ NULL, /*父节点 HWND */ NULL, /*菜
                 单句柄*/ hInstance, /* */
                 零);/* creatstruct param */
```

需要在单独的一行中显示每个参数，并伴随注释以增加可读性，通常表明过程有太多的参数。

相比之下，基于属性的方法允许用户创建对象，而无需向构造函数指定任何参数。相反，用户可以以任意顺序逐个设置定义对象的属性。例如:

```
window =新窗口;
窗口->setClassName(“AppWindow”);
窗口->setWindowTitle (winTitle);窗口->
setStyle(窗口::重叠);窗口->setSize(宽度、
高度);窗口->setModuleHandle
(moduleHandle);
```

对于用户来说，这种方式有很多优点:

- 它提供了一个非常直观的模型。

用户不需要记住他们提供属性或选项的顺序。

用户代码更具可读性，不需要额外的注释(可能与代码不同步)。

- 由于属性有默认值，用户只需要设置那些他们明确想要改变的(例如，在上面的窗口示例中，他们不必设置 *x* 和 *y* 坐标，父窗口句柄，菜单句柄，或“creatstruct”-无论那是什么)。
- 用户可以在任何时候更改属性的值，而不必在他们想要修改的时候用一个新的对象替换它。
- 通过调用 getter，用户可以查询回他们设置的所有内容，这有助于调试，在一些应用程序中很有用。

该方法与图形属性编辑器兼容，可以让用户立即看到设置属性的结果。

对于库开发者来说，设计基于属性的 api 需要更多的思考。由于属性可以按任何顺序设置，因此通常需要在实现中使用延迟初始化等技术，以避免每次属性改变时重新计算整个对象。

Qt !QRegExp。如果用户愿意，可以一次性设置好:

```
QRegExp regExp(“*.wk?”，Qt::不区分大小写，
                QRegExp::通配符);
```

但他们也可以一步一步初始化它:

```
QRegExp 正则表
达式;

regExp.setPattern(“*.wk？
”);

regExp.setCaseSensitivity (Qt::
CaseInsensitive);regExp.setPatternSyntax
(QRegExp:通配符);
```

QRegExp 实现将正则表达式或通配符模式的编译延迟到实际使用 QRegExp 对象来匹配某些文本的时候。

4.12 最好的 API 是没有 API

电影里，最好的特效是那些你没有注意到的。类似的原则也适用于 API 设计:理想的功能是那些不需要(或很少需要)应用程序编写人员额外编写代码的功能。

Qt !在 Qt 3中，widget在大多数平台上被限制为 32 767 × 32 767像素。要渲染比 QScrollView(例如，大型网页)更大的场景，唯一的解决办法是将 QScrollView 子类化，并重新实现 drawcontents()、contentsMousePressEvents()等。Qt 4 在内部围绕所有 QWidgets 的窗口组件大小限制工作。因此，我们现在可以在 QWidget 的 paintEvent()中以标准方式绘制大型场景，并将 widget 直接插入到 QScrollArea 中，从而无需使用 drawContents()和 contentsMousePressEvents()等方法。

其他具有少量 API 的重要 Qt 4 功能的例子包括 PDF 支持(通过在 QPrinter 上调用 setOutputFormat(QPrinter:: PDF - Format)来启用)，样式表(通过调用 setStyle- Sheet()来设置)，以及在 OpenGL 部件上使用 QPainter 进行 2D绘制(刚刚工作)。