GObject

维基百科，自由的百科全书

GLib对象系统，或者说**GObject**，是一个在LGPL下发布的自由软件库，它提供了一个轻便的对象系统并支持透明的多语言互通。GObject被设计为可以直接使用在 C 程序中，也封装至其他语言。

```
g_object_class_install_property
  (object_class, FILE_SIZE,
   (g_param_spec_uint64
    ("file-size", _("File Size"),
     _("The size of the file "
       "in bytes, or zero."),
     0, G_MAXUINT64, 0,
     G_PARAM_READWRITE)));
```

一个典型的GObject类的初始化代码。这个例子显示了一个属性 `file-size` 被加到一些类中。

目录

历史

GObject仅依赖于GLib和libc。它是GNOME的基石并且在GTK+， Pango，Accessibility Toolkit和大多数GNOME的高级库和应用程序中被广泛使用 。在GTK+ 2.0之前，GObject代码是GTK+的一部分。（现在GObject这个名字已经不在GTK+中了——取代它的基本类型叫做`GtkObject`。）

由于对象系统适用的范围更广，较有一般性，所以在GTK+2.0发布时，对象系统被分离了出来，改放到了另一个函数库。GtkObject 在 Gtk 演进的过程里，大部分与 GUI 不相关的部份都移到了 GObject 里，造就了新的共用基础类型 GObject 的诞生。从2002 年 3 月 11 日(也就是 GTK+ 2.0 发布的日子)开始，GObject 就一直以一个分离的函数库的形式存在。GObject 函数库现在被许多非 GUI 的程序，像命令行应用程序、服务器应用程序等，所使用。

与GLib的关系

虽然 GObject 有属于它自己的文件 [1] (http://developer.gnome.org/doc/API/2.0/gobject/) 与独立的库，但源代码却是在 Glib 的源码树里，并且与 GLib 一起发布。因此，GObject 使用与 GLib 一样的版本号码，而且一般 Linux Distro 的作法也是把GObject包在 GLib 里（举例来说，Debian 把 GObject 放在 libglib2.0 包家族里）。

类型系统

GObject 框架的基层主要依靠泛型与动态类型，称作 GType。GType 系统保留所有对象的运行时期描述以让 glue code 能方便地与其他语言作链接。类型系统可以处理任何单一继承的类型架构以及非类型的类型，如 不透明指针、字符串跟各种长度的整数与浮点数。

类型系统知道如何复制、指派和释放属于任何已注册类型的值。这对像整数这种不是参考计数(reference-counted)的类型来说是很琐碎的事情，但是对于其他是参考计数的复杂对象来说，是必要的。当类型系统复制了一个参考计数的对象，它仅仅只是增加该对象的参考计数，对复制一个复杂、不是参考计数的对象时，则是以配置存储器的方式创建副本。

这项基本的能力被实现在 GValue，GValue 是一个泛型容器的类型，里面可以用来保存注册在类型系统里的类型的值。这样的容器在与动态类型语言沟通时，特别地有用。

基础类型

没有任何关系类型的类型被称作非类型的。这些类型相当于根类型，也就是基础类型，可以被其他类型继承。

在 GLib 2.9.2 [2] (http://developer.gnome.org/doc/API/2.0/gobject/gobject-Type-Information.html) 里，非类型的自带基础类型有：

- 空类型，对应到 C 的 void (G_TYPE_NONE);
- 对应到 C 的有号、无号char、int、long 和64位整数(long long)(G_TYPE_CHAR, G_TYPE_UCHAR, G_TYPE_INT, G_TYPE_UINT, G_TYPE_LONG, G_TYPE_ULONG, G_TYPE_INT64, and G_TYPE_UINT64);
- 布林类型(G_TYPE_BOOLEAN);
- 枚举类型和"旗标"类型，都对应到 C 的枚举类型，但后者只使用在比特字段上(G_TYPE_ENUM and G_TYPE_FLAGS);
- 单精度与双精度的 IEEE 浮点数，对应到 C 的 float 跟 double(G_TYPE_FLOAT and G_TYPE_DOUBLE);
- 字符串类型，对应到 C 的 char * (G_TYPE_STRING);
- 不透明指针，对应到 C 的 void *(G_TYPE_POINTER)。

类型自带的基础类型有：

- GObject实体的基底类型类型，标准类型继承树的根 (G_TYPE_OBJECT)
- 基底接口类型，跟基底类型类型很相似，但却是标准接口继承树的根 (G_TYPE_INTERFACE)
- 包装了结构的类型，被用来包装简单的值对象或外来对象在参考计数的"盒子"里 (G_TYPE_BOXED)
- "参数规格对象的类型，用在 GObject 里作为描述对象属性的元数据(G_TYPE_PARAM)。

可以被系统自动实体化的类型被称作可实体化(instantiable)。这些类型的一个重要特色就是实体的第一个字节永远包含一个指针，指向链接到该实体类型的类型结构(虚拟表格的窗体)。为此，任何可被实体化的类型应该是类型。相对地，任何非类型类型（像整数或字符串）必须是不可实体化。另外，大部分类型类型是可实体化的，但某些类型，像接口类型，就不是。

派生类型

从自带 GObject 基础类型派生下来的，主要有四种类型：

枚举类型和 "旗标" 类型
: 一般来说，枚举类型或旗标其实都是整数，以相对口语的单字来表示特定的数值，一般都作为对象属性的类型。GLib 提供了 `glib-mkenums` [3] (http://developer.gnome.org/doc/API/2.0/gobject/glib-mkenums.html) 工具来自动化产生的过程，并产生必要的代码。

Boxed 类型
: 有些数据结构很简单，并不需要是一个类型。举例来说，我们可能有个类型，里面需要加个 `background-color` 属性，他的值应该是某个结构的实体，所以代码看起来像是这样：`struct color { int r,g,b; }`。要避免继承 `GObject` 的话，我们可以创建一个 boxed 类型来呈现这样的结构，并且提供复制和释放的处理函数给 GType。GObject 提供了简便的方法，可以让你为 GLib 数据类型作包

装。

不透明的指针类型(Opaque pointer types)

有时候，对象既不需要复制也不需要作参考计数或释放。这样的对象在 GObject 里，可以当作是不透明指针 (G_TYPE_POINTER)。通常被用来参考到特定种类的对象。

类型与接口类型

GObject 应用程序里的大部分类型都是类型，直接或间接地继承自根类型：GObject。是的，GObject 里也可以使用类似Java 的接口 (interface)，虽然很少被使用到。很少使用到的原因可能是因为接口是在 GLib 2.4 (在 2004 年 3 月 16 日发布)才被加进去。GIMP 就有使用到 GObject 的接口。

信息系统

GObject 信息系统由两个互补的部份所组成：closures 与信号。

Closures

GObject closure 是 callback (回呼)的一般化版本。支持现存已经用 C/C++ 或其他语言写好的 closure（当提供绑定时）。这允许以例如 Python 或 Java 等写好的代码被 GObject closure 调用。

信号

信号(Signal) 是 closure 被调用的主要机制。对象向类型系统注册信号 listener，在给定的信号与给定的 closure 间指定对应关系。当注册的信号被发出时，对应的 closure 就会被调用。在 GTK+ 里，所有内定的 GUI 事件（像鼠标移动和键盘动作）都会为 listeners 产生 GObject 信号以进行运作。

类型实现

每个 GObject 类型必须包含至少两个结构：类型结构与实体结构。

类型结构

类型结构相当于 C++ 类型的 vtable。第一个元素必须是父类型的类型结构。里面包含一组函数指针，也就是类型的虚拟方法。放在这里的变量，就像是 C++ 类型里的 const 或类型层级的成员。

实体结构

每个对象实体都将会是这个实体结构的副本，同样地，第一个元素，必须是实体结构的父类型 (这可以确保每个实体都有个指针可以指向类型结构，所有的基类也同样如此)，在归入父类型之后，可以在结构内放其他的变量，这就相当于 C++ 的数据成员。

C 结构没有像 "public", "protected" 或 "private" 等的访问层级修饰，一般的方法是借着在实体结构里提供一个指向私有数据的指针，照惯例称作 _priv。私有的结构可以声明在公有的头文件里，然后把实体的定义写在实现的文件中，这样作，对用户来说，他并不知道私有数据是什么，但对于实现者来说，却可以很清楚的知道。如果私有结构也注册在 GType 里，那么对象系统将会自动帮它配置空间。

GObject 框架最主要的不利点在于太过于冗长。像是手动定义的类型转换宏和难解的类型注册咒语等的大量模板代码都是创建新类型所必要的。GObject Builder 或 GOB2 这些工具试图以提供样板语法来解决这个问题。以 GOB2 写的代码必须事先处理过才能编译。另外，Vala 可以将 c# 的语法转换成 C，并编译出独立的二进制档。

用途

C 与 GObject 的组合被使用在许多成功的自由软件专案上，像是 GNOME 桌面、GTK 与 GIMP 图像处理软件。

尽管许多的 GObject 应用程序完全以 C 来撰写，但 GObject 系统可以很好地对应到许多语言，像C++、

Java、Ruby、Python和 .NET/Mono等的原生对象系统。所以在为已经使用 GObject 框架写好的库创建语言绑定时，通常比较不会那么痛苦。

以 C 来撰写 GObject 代码时，却是相对痛苦。学习曲线十分陡峭，有高级面向对象语言经验的开发者可能会发现以 C 撰写 GObject 代码相当的乏味。举例来说，要继承一个类型(即使是继承 GObject)可能就需要撰写和(或)复制上百行的代码。虽然如此，不可否认地，GObject 可以为 C 的代码提供面向对象的功能。

GObject 应用程序在运行时期为类型和接口所创建的元对象提供了互相操作的良好支持。可互相操作的能力被使用在语言绑定上，还有用户界面设计应用程序(如Glade)上。Glade 允许加载提供放了Widget(组件，派生自GObject)的库，并且取得类型的属性列表、类型信息以及文件字符串。

与其他对象系统比较

GObject 为 C 提供了近乎完整的对象系统，所以使用C语言配合GOject，可以做为使用其他从 C 分支出去的语言，像 C++ 和 Objective-C ，的替代方案。(不过 C++和Objective-C 其实各自有很多其他特色，而不是只有差在对象系统。) 最明显也最简单的不同，是 GObject 跟 Java 一样，不支持多重继承。

另一个重要的不同，GObject 仅仅只是一个库，而 C++ 和 Objective-C 的编译器还额外提供了新的语法或特性。

就目前的 C++ 编译器来说，并没有标准的 ABI 可以在所有的 C++ 编译器运行(除了 Windows，Windows 上有 COM 可以处理)，以 A 这个 C++ 编译器所编译出来的库，并不一定能被以 B C++ 编译器所编译的程序调用。如果需要这样的兼容性，C++ 的方法必须要输出为 C 的函数，这样就无法享受 C++ 带来的好处了。这主要是因为不同的 C++ 编译器使用了不同的名称特殊处理(Name mangling)以确保输出符号的独一性。(这是必要的，举例来说，不同的类型可能会有一样名称的成员函数、被覆载许多次的函数名称，或者出现在多个命名空间但同名的函数，但在输出为目标文件时，这些代码都是独立的，如果名称都一样，会被视为同一份代码，因此需要对名称作特殊处理。)对照 C 来看，C 不支持任何形式的覆载或名称特殊处理，C 库的作者永远只能使用明确的前置名以确保输出名称的全局独一性。因此，以 C 撰写的 GObject 基底的库将不会有名称特殊处理的问题，也不会受到编译器的影响。

最后，"信号"(signal) 可以说是更容易被发现的相异点了 (在其他语言被称作事件)。这当然是因为 GObject 最早被设计用在 GUI 工具箱上。的确，许多面向对象语言都已经有现成的信号库，但 GObject 是被自带在对象系统里的。因此，GObject 应用程序与其他非 GObject 应用程序比起来，会倾向于去使用信号来实现。这使得 GObject 组件，相较于纯 C ++或Java写的组件，更易于封装，也容易被重复使用。不过该注意的是，在几乎所有的平台都可以使用信号，但有时其实需要额外的库支持，例如可用于 C ++的Boost.Signals2。

外部链接

- The GObject Reference Manual (and tutorial) (http://library.gnome.org/devel/gobject/stable/)
- GObject Tutorial Aug 2004 (http://docs.programmers.ch/index.php/HOWTO_gobject)
- GOB2 — the GObject Builder (http://www.jirka.org/gob.html)
- Vala Homepage (http://live.gnome.org/Vala)

来自“http://zh.wikipedia.org/w/index.php?title=GObject&oldid=22625844”

维基媒体基金会是在美国佛罗里达州登记的501(c)(3)免税、非营利、慈善机构。