UML 类图中的关联、聚合和组合

李　云
Blog: yunli.blog.51cto.com

摘要

本文基于 UML 规范，介绍了关联的三种形式。此外，通过给出例子和相应的程序源代码帮助读者加深理解。

关键词

UML　　类图　　关联　　聚合　　组合

缩略语

UIS	UML Infrastructure Specification	UML 基础结构规范
UML	Unifed Modeling Language	统一建模语言
USS	UML Superstructure Specification	UML 上层结构规范

参考资料

《UML Infrastructure Specification, v2.2》
《UML Superstructure Specification, v2.2》
《UML 类图中的类》

1　类图中的关联

关联（association，请参见 USS 的 7.3.3 节）表示两个或多个类实例之间所存在的一种语义关系（sematic relationship）。一个关联至少有两个用属性（property，请参见 USS 的 7.3.44 节）表达的终端（end）。一个关联关系表明了多个所关联类实例（instance）之间的连接（link），也就是说关联是连接的集合。一个连接是一个包含两个关联终端的值的元组，每一个关联终端的值表示一个末端类型的实例。图 1 中，连接类 Car 和类 Window 的直线就表示一个关联关系，这个关联关系只有一个连接，因为只有两个类。连接的两个末端分别是 car_和 windows_，car_是终端类 Car 的实例（名），而 windows_是终端类型 Window 的实例（名）。在 1.3 节讨论关联的元数时，我们会进一步讨论连接与关联的关系。一个关联可以包含多个终端（或说多个类），且关联终端可以是相同的类型（或相同的类）。

需要指出的是，对于关联的终端其拥有关系有可能不同，这与关联是否是可导航的有关。当关联关系是可导航的，则关联终端的拥有者就是终端反方向所连接的类，否则终端的拥有者是关联自己。在 1.1 节讨论可导航性时，我们再回过头来看一看对于关联终端的拥有问题。

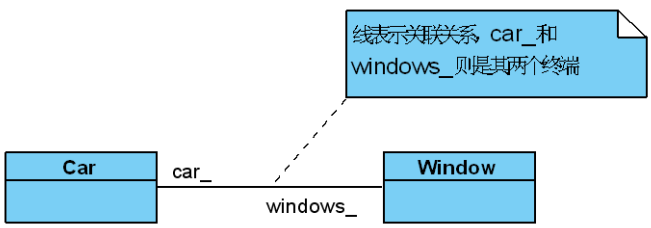

图 1

关联在我们的语言中的表现形式是什么样子的呢？下面先看看用 Visual Paradigm for UML 生成图 1 中的 C++代码（你可以生成自己所熟悉的程序语言来看一看关联在其中的表达形式）是怎么样的，在 Visual Paradigm for UML 中选择相应的 C++代码生成菜单，如图 2 所示。

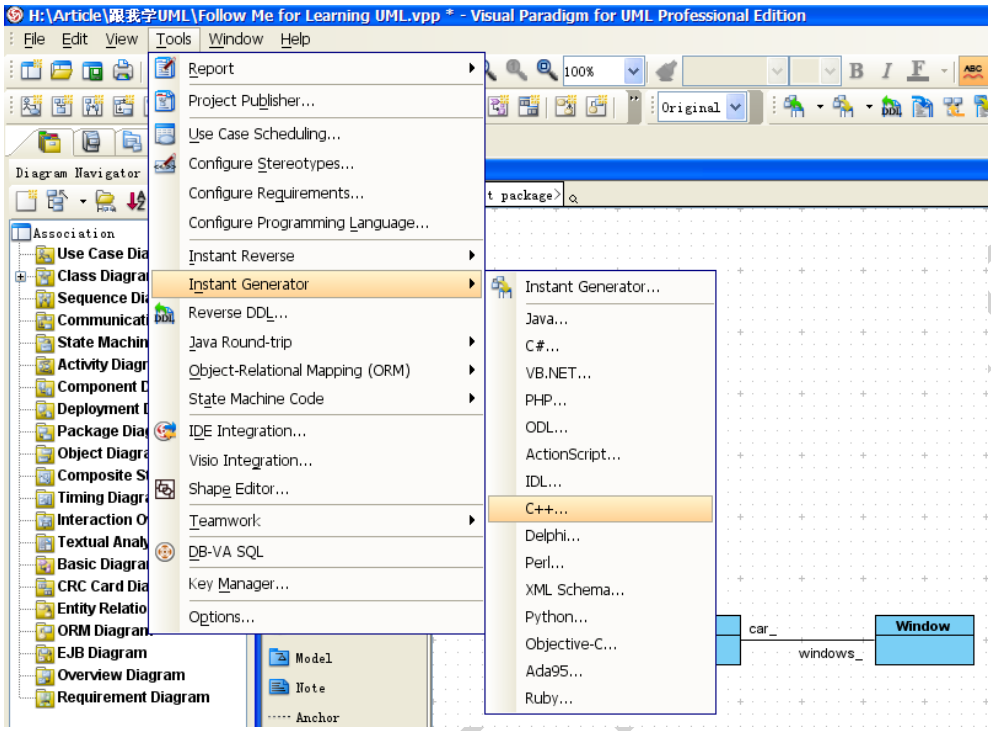

图 2

此时，将出现如图 3 所示的对话框，选择所需生成代码的元素和被生成代码的存放路径后，点击“Generate”按钮。之后，在相应的目录中将生成四个文件，分别是 Car.h、Car.cpp、Window.h 和 Window.cpp。

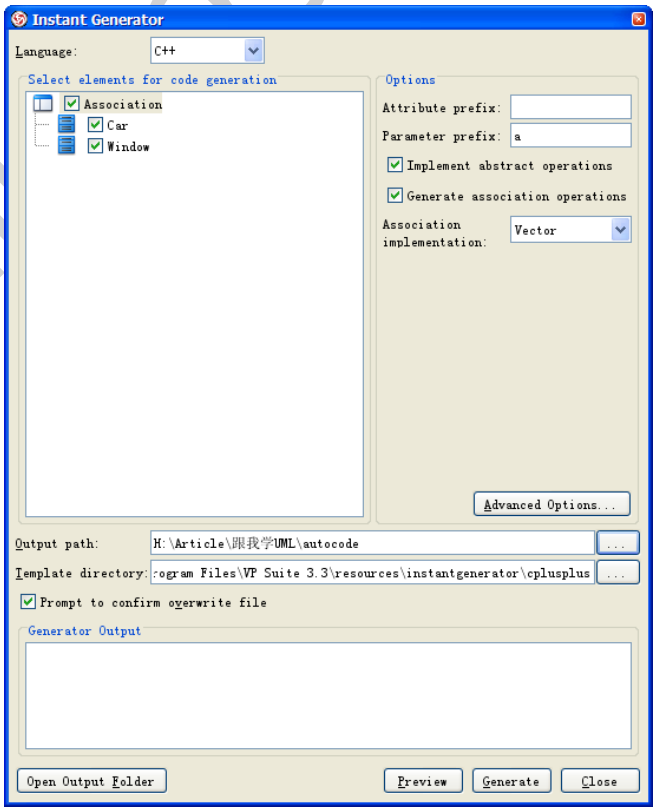

图 3

为了看一看所生成的代码中关联是如何表达的，我们需要查看一下 Car.h 和 Window.h。其代码如图 4 所示。从图 4 中可以看出关联关系在类中表现为一个成员变量或说是属性。

Car.h

```
00014: class Car
00015: {
00016: private: Window* windows_;
00017: };
```

Window.h

```
00014: class Window
00015: {
00016: private: Car* car_;
00017: };
```

图 4

1.1 关联的可导航性

可导航性（navigability）表示一个关联中连接的一端在运行时是否能被另一端类有效的存取。或者通俗的说可导航性的意思就是一个类能否通过所关联的类实例来调用类的方法。对于图 1，如果没有指明可导航性，则默认是双向都可导航的，这也是为什么生成的代码会互相拥有一个对方类型的指针变量的原因。对于图 4，从导航性角度来说，由于类 Car 和类 Window 之间的关系是相互可导航的，所以类 Car 可以通过 windows_变量调用类 Window 的方法（或说是函数）。反之，类 Window 也可以通过 car_变量来调用类 Car 的方法。

在 UML 中采用在关联线的末端放置箭头来表示是否是可导航的。当两端都不放置箭头时，表示双向都可导航的，此时，我们说两个末端的拥有者是反方向的类。回到图 1 由于两端都没有箭头，因此，我们说这个关联是双向可导航的，在这种情况下，car_的拥有者是其对端类，即类 Window 拥有 car_，与此类似，我们说类 Car 拥有 windows_。

对于图 1 中的类图，我们看一看是不是有些东西可以更加的精确。比如，车（用类 Car 表示）可能需要调用窗户（用类 Window 表示）的成员函数以实现窗户的开关，但窗户却不需要调用车的任何函数去实现特定的功能，从 UML 角度来说，从类 Car 到类 Window 是可导航的，但从类 Window 不需要导航到类 Car。为此，我们将图 1 修订成图 5。对于这种指定了可导航性的关联，其关联终端的被拥有关系将有所不同。我们说由于在 windows_侧存在关联箭头，所以 windows_的拥有都仍然是类 Car，但在 car_侧，我们说其不具导航性，因此，car_的拥有者并不是类 Window 而是关联自己。

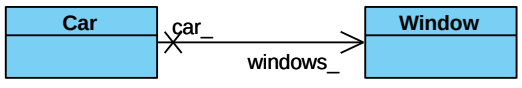

图 5

在 1.2 节讨论关联的多重性时，我们看一看一个不可导航的关联所生成的代码与可导航的关联有什么不同。

1.2 关联的多重性

前面说到关联的终端是类的实例，但实例的个数可能是需要指定的，即可以根据应用的需要进行指定。比如，通常情况下，一辆车有四个窗户，即窗户可以有四个实例；而一个窗户只能是属于一辆车。这在 UML 中如何在关联关系上进行表达呢？在 UML 中对于关联，我们可以用多重性（multiplicity）来限定一个末端（或属性）的数量。

多重性是采用如下的格式来表示的：

<lower-bound> '..' <upper-bound>

其中，<lower-bound>和<upper-bound>都是一个数字，分别表示实例个数的下限和上限。此外，“*”表示任意多个，还有就是当上限和下限值相同时，即必须是一个固定值，我们可以采用简写形式。比如，对于“1..1”我们可以简写成“1”，对于“*..*”我们可以简写成“*”。当没有指定多重性时，隐含的值为 1。

加入多重性后的图如图 6 所示，在后面谈到聚合和组合时，我们会使该图更加的精确。图 6 所生成的 C++代码列出于图 7 中。

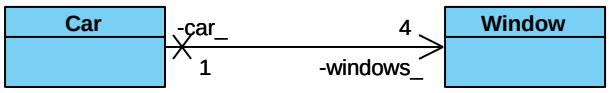

图 6

Car.h

```
00014: class Car
00015: {
00016: private: std::vector<Window*> windows_;
00017: };
```

Window.h

```
00011: class Window
00012: {
00013: };
```

图 7

图 7 与图 4 代码的区别有两处：

- 由于类 Window 到类 Car 不具导航性，因此，在类 Window 中不再有成员变量 car_。
- 由于在图 6 中我们增加了多重性，因此，图 7 中类 Car 的变量 windows_从类型 Window*变成了 std::vector<Window*>，即从一个指针变成了一个标量。在 Visual Paradigm for UML 中我们可以设置对于多重性的实现是采用 STL（Standard Template Library）中的 vector 或还是 C 中的数组。回到图 3 你可以看出对于“Association Implementation”我们选择的是 vector。

1.3 关联的元数

关联中类的个数表示关联的元数（N-ary），比如图 1 中所示的关联只有两个类，因此我们说它是二元关联（binary association）。图 8 是从 UML 规范中提取出来的一个包含三元关联（ternary association）的类图。

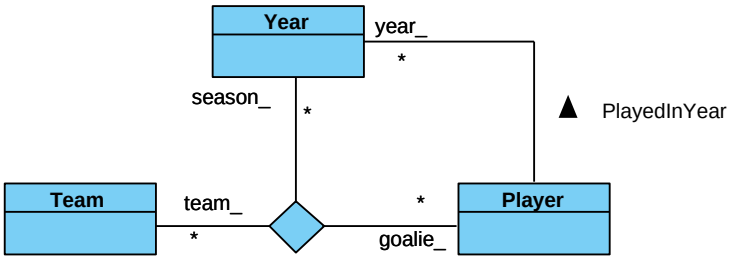

图 8

与二元关联所不同的是，三元及以上关联需采用一个菱形来连接所有的关联类。前面我们谈到了关联与连接的关系，对于图 8 中的三元关联，其中存在三个连接，即每二个关联类组成一个连接。我们说这一个三元关联是由三个连接组成的。

此外，图 8 中还存在一个二元关联，而且这一二元关联边上还有一个实心的小三角形。这一小三角形在图中的作用是用于指示关联关系的阅读顺序。比如，对于图中的二元关联，我们应当这么读“Player 在 year_年参加过比赛”（对应的英文是：Player PlayedInYear year_）。

1.4 关联的可见性

关联的可见性（visibility，请参见 USS 的 7.3.55 节）是指一个类所拥有的关联对于这一类的外部是否可见。可见性分为私有（private）、保护（protected）、包（package）和公共（public）四种。各种可见性在 UML 中的表示方法如下：

- “+”表示 public；
- “-”表示 private；
- “#”表示 protected；
- “~”表示 package；

从 C++的角度来看，我们对于 private、protected 和 public 都已非常熟悉，至于 package，它可以被理解为命名空间（namespace）。

采用 Visual Paradigm for UML 我们可以表示出关联的可见性。可见性被放在关联终端的属性名之前，在图 1 中加入关联的可见性后就变成了图 9。在图 9 中你可以看到我们将两个关联末端的可见性都设置成了 private，也就是说，windows_对于类 Car 的外部是不可见的，而 car_对于类 Window 的外部也是不可见的。

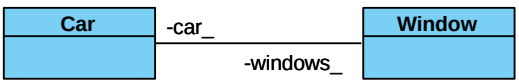

图 9

1.5 关联的属性字符串

属性字符串（property string）是采用花括号将其括起来，这些属性字符串可以应用于关联的终端。在 UML 中目前定义了以下几个属性字符串。

- *{subsets <property-name>}*：用于指示关联终端是另一个关联属性<property-name>的子集，子集的概念与集合中的子集概念是相同的。
- *{redefines <end-name>}*：用于指示关联终端是对另一个终端<end-name>的重新定义。
- *{union}*：用于指示关联终端是从其子集导出的。

- *{ordered}*：用于指示关联终端是一个有序集。
- *{nonunique}*：用于指示关联终端是一个允许存在多个相同元素的一个集合。
- *{sequence}*或*{seq}*：用于表示关联终端是一个有序的集合。

图 10 是一存在属性字符串的类图，从图中我们可以看出类 A 包括类 B 的实例 b_。从 b_的多重性我们可以看出，其中可以有任意多个类 B 的实例，且其属性字符串告诉我们 b_中的实例是一个有序的集合。同样的，类 C 有类 D 的实例 d_，且从其多重性我们可以看出，它最多只能有一个类 D 的实例，或是一个也没有。此外，从其属性字符串可以看出，d_是 b_的一个子集。

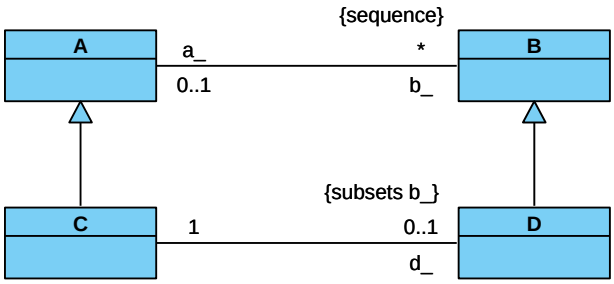

图 10

1.6 关联的聚合类型

聚合类型（aggregation kind，请参见 USS 的 7.3.2 节）是用于表明一个属性的聚合类型。在 UML 中定义了以下三种：

- *none*：不存在聚合；
- *shared*：共享聚合；
- *composite*：组合聚合；

以上三种聚合类型在 UML 中的表达方式也是不同的。对于本章节到目前为止，所有的类图都只表达了 none 类型的聚合，下面我们着重看一看共享聚合与组合聚合。

通常我们简称不存在聚合（none）的关联为***关联***，共享聚合（shared）关联为***聚合***，组合聚合（composite）关联为***组合***，后面我们也将采用这种简称。首先需要明确的一点是，聚合和组合都是关联的一种形式，那么为什么要提出这些概念呢？其中很重要的一点是我们在对应用建模时，我们有时仅仅表达存在关系是不够的，而是还要去表达在这些关系中的主和从、整体和局部的关系，在这种情况下我们就需要用到聚合和组合了。可以说聚合和组合概念的提出完全是为了使 UML 所表达的内容更加的精确。

图 11 是一个使用了组合的一个类图，从图中我们可以得知类 Window 是主，而其它的类是从。组合的表达是在关联上主方（整体方）加上一个实心的菱形。与组合的表示方法不同的是，聚合采用的是一个空心的菱形。图 11 所传达的信息是，GUI 窗口是由滑动条、标题头和面板组成的。可能有人会问，那我能不能用聚合来表示图 11 中的类之间的关系呢？要给出这个答案，有一个问题我们必需先得到答案，这个问题是：多个窗口是否会共用同一个滑动条、标题头和面板？如果答案是“不存在”，那么就显然要用组合。反之，如果答案是“存在”那么就应当考虑用聚合。当我们存在不知道应当是用聚合还是组合时，问自己一个类似“多个窗口是否会共用同一个滑动条、标题头和面板？”这样的问题，往往能帮助我们找到答案。

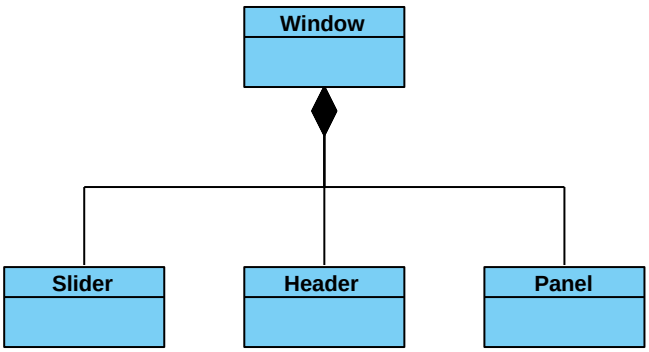

图 11

此外，除了前面提到的通过问问题的形式来帮助我们确定到底应当用聚合还是给合外，我们还可以通过其它的一种形式。这种形式就是：如果主对象的消亡会造成从对象的消亡那么应当是组合关系，否则应当是聚合关系。

图 12 是一个采用了聚合关联的类图，从图中我们可以看出，Club 与 Person 是聚合关系。我们可以得到以下信息：

- 一个 Person 可以同时属于多个 Club;
- 一个 Club 可以有多个 Person；
- 当一个 Club 消亡时并不会导致其所属 Person 的消亡。

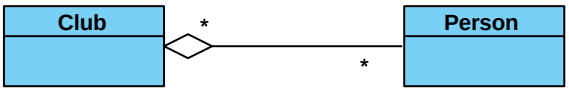

图 12

对于聚合和组合，有一点我想强调的是：对于不少情况，采用聚合和组合从语义或我们平常的生活逻辑来看，都是行得通的，但是，我们应当从设计的角度来看我们到底应当选择哪一种，这与我们在设计中所想表达的内容有关。

回到图 6 的例子，如果此时我们想表达 Car 是整体而 Window 是局部这一思想，那么我们可以将其修订为如图 13 所示。可能有人会问：我觉得用聚合也可以的啊？因为一块玻璃可以从一个车上折下来，然后装到另一个车上啊。是的，从生活常识是行得通的。对于这一问题，我的解答是：如果我们的重点是为了设计一个车的模型，我们不在乎车上的玻璃是否可以拆下来，也就是说应当用组合；反过来，如果我们设计的重点是一个汽车配件模型，我们可能就需要考虑一块玻璃是可以被折下来装到另一个车上去这种情况，在这种情形下聚合就显得更为恰当了。

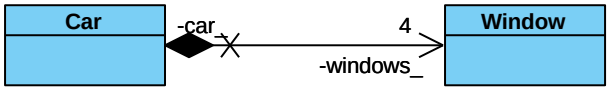

图 13

1.7 关联的另一种表达形式

除了采用图形的方式来表达关联关系外，我们还可以通过另一种方式进行表达。从前面生成的C++代码可以看出，对于类来说关联关系就是定义了一个变量。因此，很自然的我们会相到另一种表达关系。图 14 示例了这种采用定义变量（在 UML 中称之为 attribute）的方式来表达关联关系。

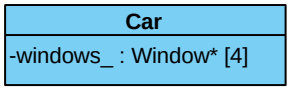

图 14

2 结束语

UML 中的关联可以分为三种，其中聚合和组合是为了进一步表达主和从、整体与局部的关系。关联关系从类的角度来看就是定义了一个类（型）的变量。而可否导航说明，能否通过所定义的变量去存取变量所对应类的方法。此外，即然关联可以表达成一个变量，那么这一变量是私有的、保护的还是公有的，等等，也是可以选择的，对于这种情形 UML 采用可见性来表达。

致读者

如果你觉得本文的哪些地方需要改进或是存在一些不明白的地方，请点击这里并留言。

修订历史

日　期	修　订　说　明
2009-07-07	新文档
2009-07-08	1)　修订了 1.2 节，增加了在没有指定多重性的情况下其隐含值为 1 的说明。 2)　引入了 UUS 和 UIS 的缩略语以方便表达。 3)　增加了对于可导航性的说明。
2009-07-11	增加了一小段用于描述关联终端的拥有关系。