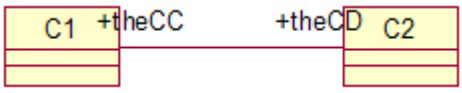
UML 之 C++类图关系全面剖析

UML 的类图关系分为： 关联、聚合/组合、依赖、泛化（继承）。而其中关联又分为双向关联、单向关联、自身关联；下面就让我们一起来看看这些关系究竟是什么，以及它们的区别在哪里。

1、关联



双向关联：
C1-C2：指双方都知道对方的存在，都可以调用对方的公共属性和方法。

在 GOF 的设计模式书上是这样描述的：虽然在分析阶段这种关系是适用的，但我们觉得它对于描述设计模式内的类关系来说显得太抽象了，因为在设计阶段关联关系 必须被映射为对象引用或指针。对象引用本身就是有向的，更适合表达我们所讨论的那种关系。所以这种关系在设计的时候比较少用到，关联一般都是有向的。

使用 ROSE 生成的代码是这样的：

```
 class C1
{
public:
    C2* theC2;
};

 class C2
{
public:
    C1* theC1;
};
```

双向关联在代码的表现为双方都拥有对方的一个指针，当然也可以是引用或者是值。

单向关联:
C3->C4：表示相识关系，指 C3 知道 C4，C3 可以调用 C4 的公共属性和方法。没有生命期的依赖。一般是表示为一种引用。

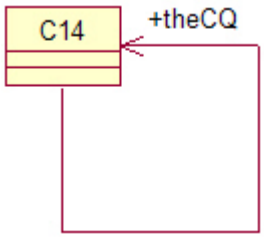
生成代码如下：

```
 class C3
{
```

```
public:
    C4* theC4;
};

 class C4
{
};
```

单向关联的代码就表现为 C3 有 C4 的指针，而 C4 对 C3 一无所知。



自身关联（反身关联）：
自己引用自己，带着一个自己的引用。

代码如下：

```
 class C14
{
public:
    C14* theC14;
};
```

就是在自己的内部有着一个自身的引用。

2、聚合/组合

当类之间有整体-部分关系的时候，我们就可以使用组合或者聚合。

聚合：表示 C9 聚合 C10，但是 C10 可以离开 C9 而独立存在（独立存在的意思是在某个应用的问题域中这个类的存在有意义。这句话怎么解，请看下面组合里的解释）。

代码如下：

```
 class C9
{
public:
    C10 theC10;
```

```
};

class C10
{
};
```

组合（也有人称为包容）：一般是实心菱形加实线箭头表示，如上图所示，表示的是 C8 被 C7 包容，而且 C8 不能离开 C7 而独立存在。但这是视问题域而定的，例 如在关心汽车的领域里，轮胎是一定要组合在汽车类中的,因为它离开了汽车就没有意义了。但是在卖轮胎的店铺业务里，就算轮胎离开了汽车，它也是有意义的， 这就可以用聚合了。在《敏捷开发》中还说到，A 组合 B，则 A 需要知道 B 的生存周期，即可能 A 负责生成或者释放 B，或者 A 通过某种途径知道 B 的生成和释放。

他们的代码如下：

```
class C7
{
public:
    C8 theC8;
};

class C8
{
};
```

可以看到，代码和聚合是一样的。具体如何区别，可能就只能用语义来区分了。

3、依赖

依赖:
指 C5 可能要用到 C6 的一些方法，也可以这样说，要完成 C5 里的所有功能，一定要有 C6 的方法协助才行。C5 依赖于 C6 的定义，一般是在 C5 类的头文件中包含了 C6 的头文件。ROSE 对依赖关系不产生属性。

注意，要避免双向依赖。一般来说，不应该存在双向依赖。

ROSE 生成的代码如下：

```
// C5.h
#include "C6.h"
class C5
{
};

// C6.h
#include "C5.h"
class C6
{
};
```

虽然 ROSE 不生成属性，但在形式上一般是 A 中的某个方法把 B 的对象作为参数使用(假设 A 依赖于 B)。如下：

```
#include "B.h"
class A
{
    void Func(B &b);
}
```

那依赖和聚合\组合、关联等有什么不同呢？

关联是类之间的一种关系，例如老师教学生，老公和老婆，水壶装水等就是一种关系。这种关系是非常明显的，在问题领域中通过分析直接就能得出。

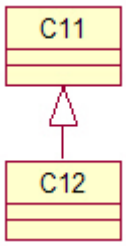
依 赖是一种弱关联，只要一个类用到另一个类，但是和另一个类的关系不是太明显的时候（可以说是“uses”了那个类），就可以把这种关系看成是依赖，依赖也 可说是一种偶然的关系，而不是必然的关系，就是“我在某个方法中偶然用到了它，但在现实中我和它并没多大关系”。例如我和锤子，我和锤子本来是没关系的， 但在有一次要钉钉子的时候，我用到了它，这就是一种依赖，依赖锤子完成钉钉子这件事情。
组合是一种整体-部分的关系，在问题域中这种关系很明显，直接分析就可以得出的。例如轮胎是车的一部分，树叶是树的一部分，手脚是身体的一部分这种的关系，非常明显的整体-部分关系。

上述的几种关系（关联、聚合/组合、依赖）在代码中可能以指针、引用、值等的方式在另一个类中出现，不拘于形式，但在逻辑上他们就有以上的区别。

这 里还要说明一下，所谓的这些关系只是在某个问题域才有效，离开了这个问题域，可能这些关系就不成立了，例如可能在某个问题域中，我是一个木匠，需要拿着锤 子去干活，可能整个问题的描述就是我拿着锤子怎么钉桌子，钉椅子，钉柜子；既然整个问题就是描述这个，我和锤子就不仅是偶然的依赖关系了，我和锤子的关系 变得非常的紧密，可能就上升为组合关系（让

我突然想起武侠小说的剑不离身，剑亡人亡...）。这个例子可能有点荒谬，但也是为了说明一个道理，就是关系和 类一样，它们都是在一个问题领域中才成立的，离开了这个问题域，他们可能就不复存在了。

4、泛化（继承）

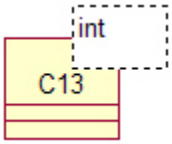


泛化关系：如果两个类存在泛化的关系时就使用，例如父和子，动物和老虎，植物和花等。
ROSE 生成的代码很简单，如下：

```
#include "C11.h"

class C12 : public C11
{
};
```

5、这里顺便提一下模板



上面的图对应的代码如下：

```
template<int>
class C13
{
};
```

这里再说一下重复度，其实看完了上面的描述之后，我们应该清楚了各个关系间的关系以及具体对应到代码是怎么样的，所谓的重复度，也只不过是上面的扩展，例如 A 和 B 有着“1 对多”的重复度，那在 A 中就有一个列表，保存着 B 对象的 N 个引用，就是这样而已。