C++ 的社会繁衍 - 文章

把 C++ 想象成人类社会。

访问权限、继承、友元将无比真实的反应人类社会中的种种关系。

一、类内部访问权限

```
1
2
3
4
5
6
7
8
9
+---------+
|class       | --> 人
+---------+
|public      | --> 你能干啥
|protected| --> 你留下啥
|private     | --> 你藏了啥
+---------+
|friend      | --> 你的圈子
+---------+
```

几乎所有人都知道你能干啥，这某种程度上是你在这个社会的价值体现。但这些不相干的人，并不知道你留下了啥，是万贯家财？还是诗书传承？谁知道？你的家人知道（类内部），你的后代知道（子类），你的朋友知道（友元）。至于你藏了些啥，除了你的家人（类内部），可能也只有几位密友（友元）了解。

二、继承时访问权限

```
1
2
3
4
```

5
6
7
+---------+
|Inherits | --> 繁衍后代
+---------+
|public | --> 嫡出
|protected| --> 庶出
|private | --> 私生
+---------+

C++ 的社会没有计划生育，也似乎没有限制一夫一妻。所以就存在正房和偏房的问题，嫡长子光明正大（public）的继承了你的财产和光环（public, protected）。其余庶出（protected）的就没那么好运了，仅仅能够分到一点财产（protected, protected）。而风流快活的产物——私生子（private）更是可怜，分到的东西谁也说不得，是私密。（private，private）。

C++
```
class Base {
protected:
        int prot_mem;
};
class Sneaky : public Base {
        friend void clobber(Sneaky &s) { s.j = s.prot_mem = 0; }
        friend void clobber(Base &b) { b.prot_mem = 0; } // error
        int j;
};
```
`prot_mem` 是 `Base` 的 `protected` 成员。那么对于 以 `public` 的姿势继承 `Base` 的子类 `Sneaky` 来说，它可以取得该成员（嫡长子）。

而第一个 `clobber` 作为 `Sneaky` 的密友，自然也可以取得 `Sneaky` 的成员，如其自己的 `j`，以及继承自父类的 `prot_mem`。

第二个 `clobber` 作为 `Sneaky` 的密友，却妄想去直接拿其父亲留下的 `prot_mem`，这显然是不合理的。 注意这里与上面那个的区别，该密友越过了 `Sneaky`，直接去拿其父亲的遗物，这是违背了社会法规的。

再来看三代同堂：

```
// 父
class Base {
public:
        int pub_mem;
protected:
        int prot_mem;
private:
```

```
        int pri_mem;
};
// 子
struct Pub_Derv : public Base {
        int f() { return prot_mem; }
        int g() { return pri_mem; } // error
};
struct Priv_Derv : private Base {
        int f1() const { return prot_mem; }
};
// 孙
struct Derived_from_Public : public Pub_Derv {
        int use_base() { return prot_mem; }
};
struct Derived_from_Private : public Priv_Derv {
        int use_base() { return prot_mem; } // error
};
```

- 注意第一个 error 处，是由于嫡长子去触及父亲藏起来的东西导致的。
- 再看第二个 error 处，是由于私生子的儿子想去拿爷爷留下的东西导致的。

细分析，儿子辈，一个嫡长子一个私生子，`prot_mem` 是祖辈留下之物，虽然两个儿子都继承了，但对于嫡长子来说，`prot_mem` 仍旧是可以遗留之物(`protected`权限)；而对于私生子来说，`prot_mem` 却成了需藏起来之物(`private`权限)。那么，到了孙子辈，长房长孙继续拿到祖辈继承之物，而反观私房长孙，却和祖辈几乎毫无瓜葛了。

再来看一个朋友乱入的：

C++

```
class Base {
        friend class Pal;
protected:
        int prot_mem;
};
class Sneaky : public Base {
        int j;
};
class Pal {
public:
        int f(Base b) { return b.prot_mem; }
        int f2(Sneaky s) { return s.j; } // error
        int f3(Sneaky s) { return s.prot_mem; }
};
class D2 : public Pal {
```

```
public:
        int mem(Base b) { return b.prot_mem; } // error
};
```

`Pal` 是父亲的密友，拿到父亲所留之物理所当然。然而直接跑去拿儿子的私物，却不合情理。但如果只是拿儿子所继承的父亲遗留之物呢？这是可能的，在很多情况下，也是合情合理的。（可能所留本来就有密友的份）。

好了，如果父亲的密友也有了嫡长子，他去拿该父亲的所留之物呢？这就有点不讲规矩了，密友关系仅存在于父辈两者之间，继承者无论如何也无法去拿上一辈朋友的东西的。

四、私生子的逆袭

从上面的一些例子，可以很明显的看到私生子的惨状，只要私生，祖上的一切接变成 `private`，几乎没法再传承下去。

幸好 C++ 的社会里倒也公平，提供了一个 `using` 关键字，让私生子也有了逆袭的机会。如下例：

```
C++#include <cstddef>
class Base {
public:
        std::size_t size() const { return n; }
protected:
        std::size_t n;
};
class Derived : private Base {
public:
        using Base::size;
protected:
        using Base::n;
};
```

所谓私生子的 `Derived`，原本 `size` 和 `n` 都是私有成员，经过 `using` 声明后，前者为 `public`，后者为 `protected`。逆袭成功。

以上皆是对 C++ 面向对象体系中，类内部及继承时成员可见度规则的一些调侃。

祝愿那些还在这个社会体系中混的程序员们，让自己的家族开枝散叶，生生不息~

（更高端的齐家之道，估计还要学习设计模式才好。）

合作联系

Email：bd@Jobbole.com

QQ： 2302462408 （加好友请注明来意）

更多频道

小组 - 好的话题、有启发的回复、值得信赖的圈子
头条 - 分享和发现有价值的内容与观点
相亲 - 为IT单身男女服务的征婚传播平台
资源 - 优秀的工具资源导航
翻译 - 翻译传播优秀的外文文章
文章 - 国内外的精选文章
设计 - UI,网页，交互和用户体验
iOS - 专注iOS技术分享
安卓 - 专注Android技术分享
前端 - JavaScript, HTML5, CSS
Java - 专注Java技术分享
Python - 专注Python技术分享