

73 Dynamic Casting in C++

前情提要：69 Casting in C++

Casting（强制转换）是我们在C++中使用的类型之间转换的一种方法，而类型系统是C++提供给我们的一种保护代码的方法，不是我们必须坚持使用的东西，因为我们愿意的话可以在类型之间自由地转换。

1. 认识动态类型转换

而**dynamic casting**是当我们想要做特定类型的类型准换时，作为一种安全机制提供给我们的。**dynamic_cast**是C++风格的类型转换，仅适用于C++，不能在C语言中使用。它做了额外的工作来确保我们实际的类型转换 *is valid*（有效）的。

是否使用 **dynamic_cast** 完全取决于你自己，这里只给出一些要点，让你知道什么情况下你应该使用它。

认识到这一点很重要：**dynamic_cast** 更像一个函数，它不像编译时进行的类型转换，而是在运行时计算，因此它有相关的运行成本。

dynamic_cast 是专门用于**沿继承层次结构**进行的强制类型转换，比如我的一个游戏里有一个实体类，它派生出了玩家类和敌人类，如果我想将玩家转换为实体是很简单的，因为玩家本身就是实体对象，可以隐式转换。但如果我想将一个实体类型转换为玩家，编译器会相信我们，如果它并不是一个玩家的话我们就相当于在尝试玩家独有的数据，程序可能会崩溃。因为这个原因，**dynamic_cast** 常被用来做验证，如果我们尝试使用它将一个敌人转化为玩家，这个转化会失败，**dynamic_cast** 会返回一个NULL指针，也就是0。我们可以用它来检查一个对象是否是给定类型，

2. 示例

```
C++

class Entity // 基类
{
public:
};

class Player : public Entity
{
public:
};

class Enemy : public Entity
{
public:
};

int main()
{
    Player* player = new Player();
    Entity* e = player; // 这里直接就隐式转换了

    Entity* e1 = new Enemy();
    Player* p = (Player*)e1; // 报错，我们需要向编译器保证这是一个Player
}
```

但这样强制转换是很危险的，因为e1其实是Enemy，如果强制转换为Player，除非Player和Entity都有Enemy的所有成员和函数，程序都会出现问题。

`dynamic_cast` 只用于多态类型:

```
dynamic_cast<Player*>(e);  
Cannot cast from Entity* to Player* via dynamic_cast: expression type is not polymorphic
```

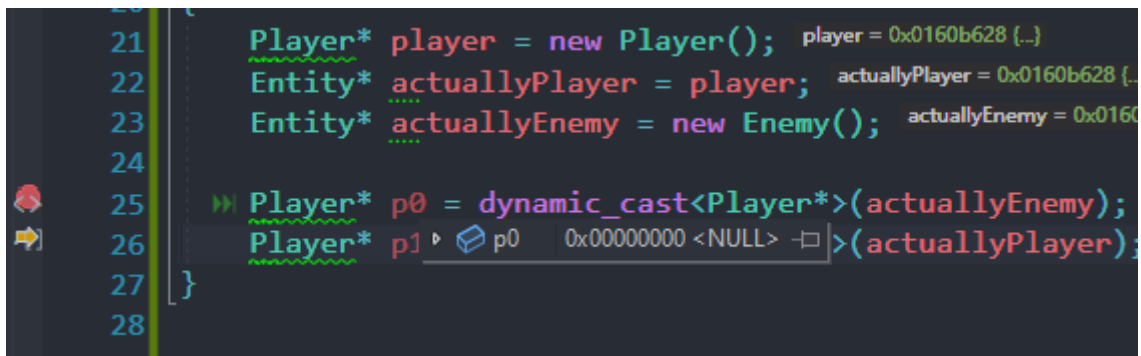
所以我们需要一个虚函数表, 来告诉我们这实际上是一个多态类类型:

```
class Entity  
{  
public:  
    virtual void PrintName(){} // 见 28 虚函数  
};
```

测试运行:

```
Player* player = new Player();  
Entity* actuallyPlayer = player;  
Entity* actuallyEnemy = new Enemy();  
  
Player* p0 = dynamic_cast<Player*>(actuallyEnemy);  
Player* p1 = dynamic_cast<Player*>(actuallyPlayer);
```

可以看出p0转换失败:



```
21 Player* player = new Player(); player = 0x0160b628 {...}  
22 Entity* actuallyPlayer = player; actuallyPlayer = 0x0160b628 {...}  
23 Entity* actuallyEnemy = new Enemy(); actuallyEnemy = 0x0160b628 {...}  
24  
25 Player* p0 = dynamic_cast<Player*>(actuallyEnemy);  
26 Player* p1 = dynamic_cast<Player*>(actuallyPlayer);  
27  
28
```

而p1是一个有效的Entity:

p0	0x00000000 <NULL>
p1	0x0160b628 {...}

这就是动态类型转换做的事: 如果强制转换有效, 就返回你想要的指针, 如果它不是你声明的给定类型, 转换无效就返回 NULL。

3. 它是做到判断的

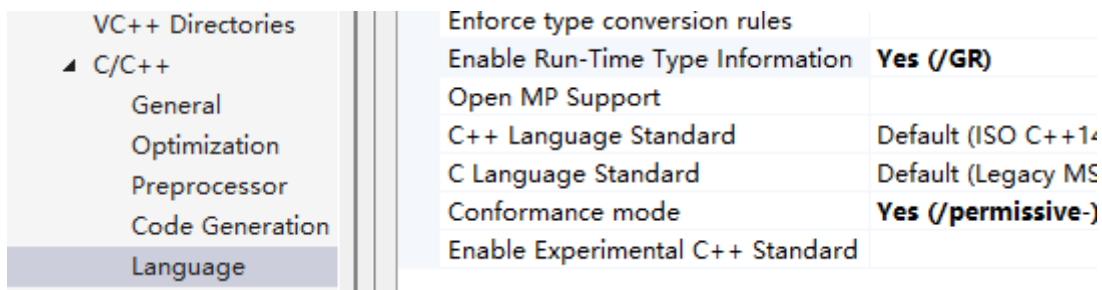
能做到这一点是因为它储存了 *runtime type information* (运行时类型信息, RTTI), 这存储着我们的所有类型的运行时类型信息, 这会增加一些开销, 但它可以让你做到动态类型转换之类的事情。

这里有两件事情需要考虑:

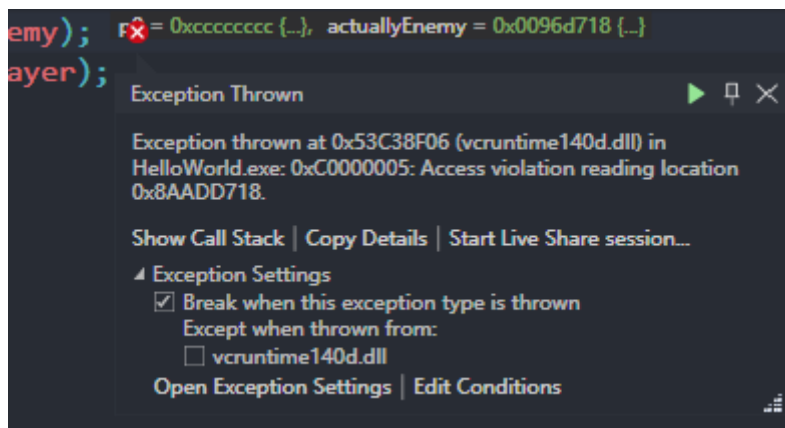
1、RTTI增加了开销, 因为类型需要存储更多关于自己的信息;

2、其次，`dynamic_cast`也需要时间，因为我们需要检查类型信息是否匹配，这个实体是敌人还是玩家，是什么类型的？当我们使用它的时候，必须在运行时进行验证，它确实增加了开销。

可以在VS中关掉RTTI：



运行代码：



Output报错：

```
Access violation reading location 0x8AADD718.
```

可以看到有访问冲突，因为它没有得到类型信息，所以没法返回NULL。

所以一定要了解动态类型转换的实际含义，因为它们会做一些额外的事情，并且大多数情况下需要开启RTTI（隐式转换就不会程序崩溃）

除此之外，你还可以像C#或Java一样可以做检查之类的事：

```
Player* p0 = dynamic_cast<Player*>(actuallyEnemy);  
if(p0) // 为了防止以后再转换，一般就不在条件里写完整的转换了  
{  
  
}
```