11_新型的类型转换

C语言方式的强制类型转换

```
(Type)(Expression);
Type(Expression);
```

`lesson_11-1.c`

```
#include <stdio.h>

typedef void PF(int);

struct Pointer
{
    int x;
    int y;
};

int main(void)
{
    int v = 0x12345;
    PF* pf = (PF*)v;
    char c = char(v);
    Pointer* p = (Pointer*)v;

    pf(2);
    printf("p->x = %d\n", p->x);
    printf("p->y = %d\n", p->y);

    return 0;
}
```

> 编译时以上源码使用 `g++ -m32 lesson_11-1.c`
> 编译时候需要 `-m32` 参数，指定生成32位程序才行，否则有警告，因为 `int` 只有4字节，64位程序中指针的大小是8字节，将4字节的值强转为64位指针提示：
> `warning: cast to pointer from integer of different size`

输出

```
root@ubuntu:~/exp/DT_CPP/part01# ./a.out
Segmentation fault (core dumped)
```

C方式强制类型转换存在的问题

- 过于粗暴

  任意类型之间都可以进行转换，编译器很难判断其正确性

- 难于定位

  在源码中无法快速定位所有使用强制类型转换的语句

C++方式的强制类型转换

语法

```
xxxx_cast<Type>(Expression)
```

0x1 static_cast强制类型转换

- 用于 `基本类型` 间的转换
- `不能` 用于 `基本类型指针` 间的转换
- 用于 `有继承关系类对象` 之间的转换和 `类指针` 之间的转换

0x2 const_cast强制类型转换

- 用于 `去除` 变量的 `只读属性`
- 强制转换的 `目标类型` 必须是 `指针` 或 `引用`

0x3 reinterpret_cast强制类型转换

- 用于 `指针类型` 与 `指针类型` 之间的强制转换
- 用于 `整数` 和 `指针类型` 之间的强制转换

0x4 dynamic_cast强制类型转换

- 用于 `有继承关系的类指针` 间的转换
- 用于 `有交叉关系的类指针` 间的转换
- 具有 `类型检查` 的功能
- **需要虚函数的支持**

实例分析

`lesson_11-2.cpp`

```
1 #include <stdio.h>
2
```

```
void static_cast_demo()
{
    int i = 0x12345;
    char c = 'c';
    int* pi = &i;
    char* pc = &c;

    c = static_cast<char>(i);
    //pc = static_cast<char*>(pi);  // error: invalid static_cast from type ‘int*’ to type ‘char*’

}

void const_cast_demo()
{
    const int& j = 1;
    int& k = const_cast<int&>(j);

    const int x = 2;
    int& y = const_cast<int&>(x);

    //int z = const_cast<int>(x);  //error: invalid use of const_cast with type ‘int’, which is not a pointer, reference, nor a pointer-to-data-member type

    k = 5;

    printf("k = %d\n", k);
    printf("j = %d\n", j);

    y = 8;

    //printf("*(&x) = %d\n", *(&x));  // correct

    printf("x = %d\n", x);
    printf("y = %d\n", y);
    printf("&x = %p\n", &x);
    printf("&y = %p\n", &y);
    printf("x = %d\n", x);
    printf("y = %d\n", y);
}

void reinterpret_cast_demo()
{
    int i = 0;
    char c = 'c';
    int* pi = &i;
    char* pc = &c;

    pc = reinterpret_cast<char*>(pi);
    pi = reinterpret_cast<int*>(pc);
    pi = reinterpret_cast<int*>(i);
    //c = reinterpret_cast<char>(i);  //error: invalid cast from type ‘int’ to type ‘char’
```

```
53  }
54
55  void dynamic_cast_demo()
56  {
57      int i = 0;
58      int* pi = &i;
59      //char* pc = dynamic_cast<char*>(pi);  //error: cannot dynamic_cast ‘pi’ (of type
    ‘int*’) to type ‘char*’ (target is not pointer or reference to class)
60  }
61
62  int main(void)
63  {
64      static_cast_demo();
65
66      const_cast_demo();
67
68      dynamic_cast_demo();
69
70      return 0;
71  }
```

小结

- C方式的强制类型转换
  - 过于粗暴
  - 潜在的问题不易被发现
  - 不易在代码中定位
- 新式类型转换以C++关键字的方式出现
  - 编译器能够帮助检查潜在的问题
  - 非常方便的在代码中定位
  - 支持动态类型识别(dynamic_cast)