Modern C++风格类型转换

C++类型转换分为显式类型转换和隐式类型转换，隐式类型转换由编译器自动完成，这里我们讨论显示类型转换

旧风格的类型转换

C 风格转换不进行类型检查，因此可能导致未定义行为，在安全性上存在一些问题

旧风格的类型转换 C++

```
type(expr); //函数形式的强制类型转换
(type)expr; //C语言风格的强制类型转换
```

Modern C++风格的类型转换

Modern C++风格的类型转换 C++

```
cast-name<type>(expression)
```

type：转换的目标类型

expression：被转换的值

cast-name：表示转换的方式，有 `static_cast` `dynamic_cast` `const_cast` 和 `reinterpret_cast` 四种

static_cast

static_cast C++

```
static_cast<type>(expression)
```

任何编写程序时能够明确的类型转换都可以使用 `static_cast` （但是不能转换掉底层 `const` `volatile` 和 `__unaligned` 属性。由于不提供运行时的检查，所以叫 `static_cast` ，因此需要在编写

程序时确认转换的安全性

主要的应用场景：

1. 类层次结构中，父类和子类之间指针和引用的转换
2. 用于基本数据类型之间的转换
3. 把void指针转换成目标类型的指针（极不安全）

dynamic_cast

dynamic_cast C++

```
dynamic_cast<type>(expression)
```

相比于 `static_cast` ， `dynamic_cast` 会在运行时检查类型转换是否合法，具有一定的安全性，但同时也会额外消耗一些性能。 `dynamic_cast` 使用的场景与 `static_cast` 类似，在类层次结构中使用时，上行转换和static_cast没有区别，都是安全的；下行转换时， `dynamic_cast` 会检查转换的类型，相比于 `static_cast` 更为安全。

dynamic_cast转换仅适用于指针或者引用

若指针转换失败，则返回空指针；若引用转换失败，则抛出异常

const_cast

const_cast C++

```
const_cast<type>(expression)
```

`const_cast` 用于移除类型的 `const` `volatile` 和 `__unaligned` 属性

常量指针被转换成非常量指针，并且仍然指向原来的对象；常量引用被转换成非常量引用，并且仍然引用原来的对象

示例代码 C++

```
const char *pc;
char *p = const_cast<char *>(pc);
```

reinterpret_cast

reinterpret_cast C++

```
reinterpret_cast<type>(expression)
```

非常激进的指针转换类型，在编译期完成，可以转换任何类型的指针，所以极不安全。

非极端情况下，不建议使用。

示例代码 C++

```
int *ip;
char *pc = reinterpret_cast<char *>(ip);
```