# C++数据类型转换

### 栈内数据类型转换

栈转换规则遵循了严格的数据类型转换规则，即没有关系的数据无法进行转换，继承关系中，子类可以向父类转换（数据丢失），父类无法向子类转换。

### 指针转换（堆数据转换）

指针的本质存储的是地址信息，具体地址中的数据模型由指针的类型决定，而指针的类型决定了编译器对其指向内存空间的解释方式

C++中，指针的转换是没有约束的，即任何指针都可以做类型转换。C++认为指针的转换只影响了空间中数据的操作方式，并不指指针本身。所以指针间的互相转换是被允许的，转换后地址值不变。

### 继承关系中的指针转换

多继承中需要注意指针对齐方式，对象模型的构建由继承层次关系决定。但是为指针进行转换时，如果转换到第二继承位类型时，则指针的地址会做偏移操作，偏移长度为继承位第一位基类的长度，关系以此类推

**C++的指针类型转换是不安全的（旧有C的转换规则），由于编译器不会处理类型转换后是否合规，经常会导致数据读取错误，所以我们需要进行基本的数据转换类型检查。**

### 万用指针void\*（无类型指针）

可以指向任意地址，存储任意指针的值。只保留了地址信息，编译器无法进行解地址操作，因为它是不完整类型。不确定数据长度，一般用来进行参数传递操作中，不约束指针内容传入，方便进行更广泛适应性的逻辑编写。

### C风格的类型转换

直接借助括号加上转换目标即可完成数据转换，操作方便，但是转换时不考虑合理性，需要由程序员保证转换的合理性。属于转换中暴力的转换方式。

### C++指针的转换方式

1. reinterpret\_cast基本和C转换没有区别，踏实简单的二进制转换，转换存在危险
2. static\_cast 有理性转换（有关系的可以互转），基本数据类型可以互转（但是不能是堆对象），同一继承体系中可以互转，任意类型与void\*之间转换。这种转换方式是编译时态转换。
3. dynamic\_cast 最具有C++特性的转换，转为继承关系中指针转换进行操作，dc在编译层约束了非指针类型无法进行转换，只能操作指针。Dc语序无关系类型指针进行转换，但是会返回空。Dc转换只存在于继承关系中，并且基类需要有虚函数（多态操作）

注意：被转的目标必须具备多态的特性（虚函数），接受转换的目标可以不具备多态的特性

A\* a = dynamic\_cast<A\*>(b);//b必须具备多态的特性，否则无法当作参数被dynamic\_cast使用

1. const\_cast 为变量增加或是删除常态属性，编译器可以通过。

const int\* pa = new int(10);

int\* pm = const\_cast<int\*>(pa);

\*pm = 100;

*cout* << \*pa << *endl*;

### 为什么会有类型转换

一般类型转换会出现在内容交互过程中，例如函数需要的参数是整形，而目前我们手里只有浮点型，那么调用函数时将出现类型转换操作（对象数据类型一样）。但是我们必须记住！类型转换之间应该遵从有理转换，即有关系的进行转换，无关系的不要转换！这可以有效的防止因转换类型导致数据操作异常的问题。