基础5 多态与RTTI

① C++运行时的多态 详情见目录(2)

```
struct test1{ virtual void vfunc(){…}};
struct test2:public test1{
        virtual void vfunc() override{…}};
Test2 b      ; Test1 a=b;   非多态
             Test1 *p=b;    多态
```

原因：两者类型转换方式不同，上面是对象之间的类型转换 (虚表已变)

下面为指针的重新解析 (虚表未变)

被继承的根节点类通常定义一个虚析构函数

② RTTI (Run Time Type Identification)

运行时类型判别 使用std::type_info*获取类型信息

```
std::type_info * base_type =(std::type_info*)(*((int_64*)
(*((int_64*)(xxx))));
base_type->name();
```

得到其类型

RTTI提供了2个有用的操作符

a. typeid操作符返回指针的引用所指的实际类型

```
typeid(A).name()   typeid(B).name()
typeid(a).name()   typeid(&a).name()
```

注：① typeid删除了std::type_info的拷贝构造，

因此只能用引用\指针接收

② typeid会忽略CV限定符

b. dynamic_cast<type*&>(e)，可将某类的指针或引用安全地转为其它派生类的指针或引用

```
struct Base{存在虚函数};
struct A: public Base{ ... };
Base *b_ptr = dynamic_cast<Base *>(&a);
```

✓

子类可安全转为父类指针

```
A* a_ptr = dynamic_cast<A*>(&b);
```

✗

父类无法转为子类指针，操作了未知内存 返回空指针nullptr

```
A* a_ptr = dynamic_cast<A*>(b_ptr);
```