Technology Tarena 达内科技

标准C++语言

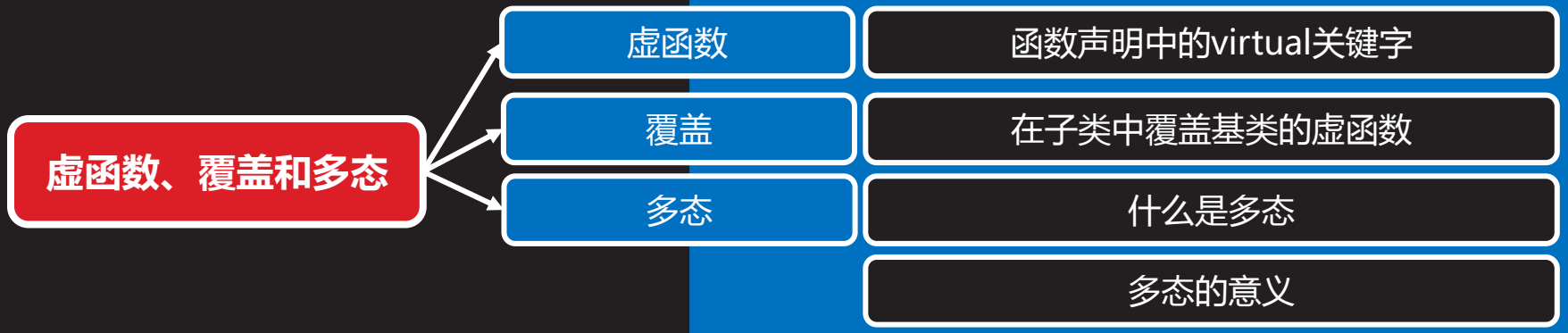
PART 2

DAY03

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	虚函数、覆盖和多态
	10:30 ~ 11:20	
	11:30 ~ 12:20	覆盖和多态的条件
下午	14:00 ~ 14:50	纯虚函数与抽象类
	15:00 ~ 15:50	虚函数表与动态绑定
	16:00 ~ 16:50	运行时类型信息
	17:00 ~ 17:30	总结和答疑

虚函数、覆盖和多态


虚函数、覆盖和多态
虚函数
覆盖
多态
函数声明中的virtual关键字
在子类中覆盖基类的虚函数
什么是多态
多态的意义

Technology Tarena 达内科技

虚函数

函数声明中的virtual关键字

- 形如

  class 类名 {

  virtual 返回类型 函数名 (形参表) { … }

  };

  的成员函数，称为虚函数或方法

  - class Shape {

    public:

    virtual void draw (void) const { … }

    };

知识讲解

Technology Tarena 达内科技

覆盖

在子类中覆盖基类的虚函数

- 如果子类的成员函数和基类的虚函数具有相同的函数原型，那么该成员函数就也是虚函数，无论其是否带有virtual关键字，且对基类的虚函数构成覆盖
  - class Rectangle : public Shape {
    public:
    virtual void draw (void) const { … }
    };
  - class Circle : public Shape {
    public:
    void draw (void) const { … }
    };

知识讲解

Technology Tarena 达内科技

多态

什么是多态

- 如果子类提供了对基类虚函数的有效覆盖，那么通过一个指向子类对象的基类指针，或者引用子类对象的基类引用，调用该虚函数，实际被调用的将是子类中的覆盖版本，而非基类中的原始版本，这种语法现象称为多态
  - Rectangle r (...);
    Circle c (...);
    Shape* ps = &r;
    ps->draw (); // 调用Rectangle::draw
    Shape& rs = c;
    rs.draw (); // 调用Circle::draw

知识讲解

多态的意义

- 多态的重要意义在于，一般情况下，调用哪个类的成员函数是由调用者指针或引用本身的类型决定的，而当多态发生时，调用哪个类的成员函数则完全由调用者指针或引用的实际目标对象的类型决定。这样一来，源自同一种类型的同一种激励，竟然可以产生多种不同的响应，谓之多态
  - void drawAny (Shape const& shape) {
    shape.draw ();
    }
  - void drawAll (Shape const* shapes[]) {
    for (size_t i = 0; shapes[i]; ++i)
    shapes[i]->draw ();
    }

知识讲解

Technology Tarena 达内科技

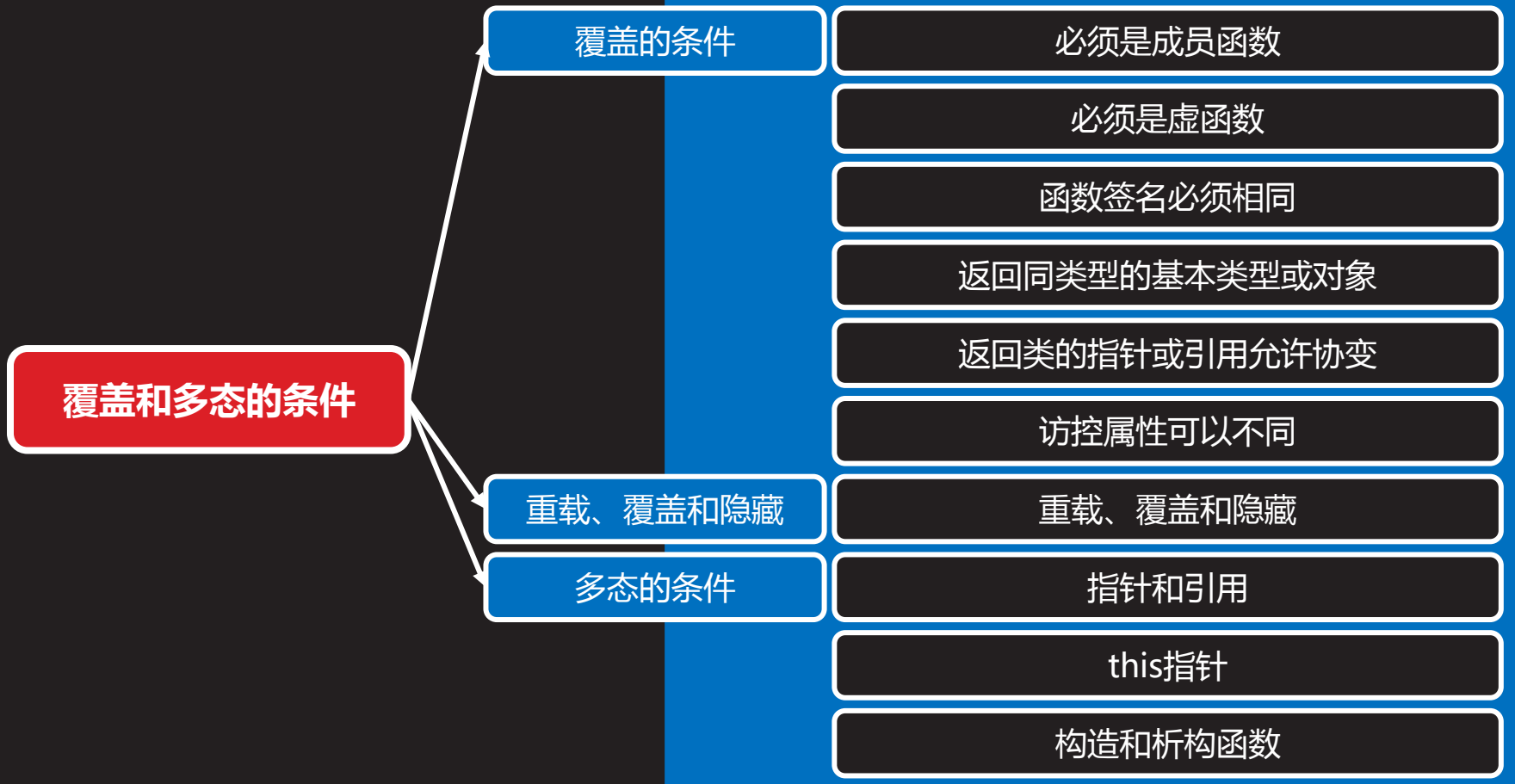
电子文档阅读器

【参见：TTS COOKBOOK 】

课堂练习

- 电子文档阅读器

覆盖和多态的条件


覆盖和多态的条件
覆盖的条件
必须是成员函数
必须是虚函数
函数签名必须相同
返回同类型的基本类型或对象
返回类的指针或引用允许协变
访控属性可以不同
重载、覆盖和隐藏
重载、覆盖和隐藏
多态的条件
指针和引用
this指针
构造和析构函数

Technology Tarena 达内科技

覆盖的条件

必须是成员函数

- 只有类的成员函数才能被声明为虚函数，全局函数和类的静态成员函数都不能被声明为虚函数
  - virtual void global (void); // 错误
  - class A {
    static virtual void staticNumb (void); // 错误
    };

知识讲解

必须是虚函数

- 只有基类中被virtual关键字声明为虚函数的成员函数才能被子类覆盖
  - class A { void foo (void); };
  - class B : public A {
    virtual void foo (void); // 隐藏A::foo
    };
  - class C : public B {
    void foo (void); // 覆盖B::foo
    };
  - class D : public C {
    void foo (void); // 覆盖C::foo
    };

知识讲解

函数签名必须相同

- 虚函数在子类中的覆盖版必须和该函数的基类版本拥有完全相同的签名，即函数名、形参表和常属性严格一致
  - class A {
    virtual void foo (void);
    }
  - class B : public A {
    virtual void bar (void); // 函数名不一致
    virtual void foo (int); // 形参表不一致
    virtual void foo (void) const; // 常属性不一致
    void foo (void); // 覆盖A::foo
    };

返回同类型的基本类型或对象

- 如果基类中的虚函数返回基本类型的数据或类类型的对象，那么该函数在子类中的覆盖版本必须返回相同类型的数据或对象，否则将引发编译错误
  - class A {
    virtual void foo (void);
    virtual int bar (void);
    virtual X hum (void);
    };
  - class B : public A {
    void foo (void);
    int bar (void);
    X hum (void);
    };

知识讲解

返回类的指针或引用允许协变

- 如果基类中的虚函数返回类类型的指针或引用，那么该函数在子类中的覆盖版本可以返回其基类版本返回类型的公有子类的指针或引用——类型协变
  - class X { ... };
  - class Y : public X { ... };
  - class A {
    virtual X* foo (void);
    virtual X& bar (void);
    };
  - class B : public A {
    Y* foo (void);
    Y& bar (void);
    };

知识讲解

访控属性可以不同

- 无论基类中的虚函数位于该类的公有、私有还是保护部分，该函数在子类中的覆盖版本都可以出现在该类包括公有、私有和保护在内的任何部分
  - class A {
    public:
    virtual void foo (void);
    };
  - class B : public A {
    private:
    void foo (void);
    };
  - A* a = new B;
    a->foo (); // 调用B::foo，虽然其为私有成员

知识讲解

覆盖的条件

【参见：TTS COOKBOOK 】

课堂练习

- 覆盖的条件

重载、覆盖和隐藏

重载、覆盖和隐藏

- 重载必须在同一个作用域中，包括通过using声明引入的
- 覆盖要满足一系列特殊条件
- 子类与基类的同名成员函数，不满足重载和覆盖的条件，且能正常通过编译，则必然构成隐藏
  - class Base {
    ① virtual void foo (void);
    ② virtual void foo (void) const;
    };
  - class Derived : public Base {
    ③ virtual void foo (void);
    ④ virtual char foo (void) const;
    };

①和②构成重载
③和④构成重载
③隐藏②覆盖①
④隐藏①覆盖②出错

知识讲解

Technology Tarena 达内科技

多态的条件

指针和引用

- 多态特性除了需要在基类中声明虚函数以外，还必须借助指针或者引用调用该虚函数，才能表现出来
  - Rectangle r (...);
    Circle c (...);
    Shape sr = r;
    sr.draw (); // 调用Shape::draw
    Shape sc = c;
    sc.draw (); // 调用Shape ::draw

知识讲解

this指针

- 调用虚函数的指针也可能是成员函数中的this指针，只要它是一个指向子类对象的基类指针，同样可以产生多态
  - class A {
    virtual void foo (void) { ... };
    void bar (void) {
    foo (); // this->foo ()，调用B::foo
    }
    };
  - class B : public A {
    void foo (void) { ... };
    };
  - B b;
    b.bar ();

知识讲解

构造和析构函数

- 当基类的构造函数被子类的构造函数调用时，子类对象尚不能说是子类类型的，它只表现出基类类型的外观和行为。这时调用虚函数，它只能被绑定到基类版本
  - class A {
    A (void) {
    foo (); // 调用A::foo
    }
    virtual void foo (void) { ... };
    };
  - class B : public A {
    void foo (void) { ... };
    };
  - B b;

知识讲解

构造和析构函数（续1）

- 当基类的析构函数被子类的析构函数调用时，子类对象已不再是子类类型的了，它只表现出基类类型的外观和行为。这时调用虚函数，它只能被绑定到基类版本
  - class A {
    ~A (void) {
    foo (); // 调用A::foo
    }
    virtual void foo (void) { … };
    };
  - class B : public A {
    void foo (void) { … };
    };
  - B b;

知识讲解

构造和析构函数（续2）

- 在基类的构造和析构函数中调用虚函数，绝不可能表现出多态性。实际被调用的一定是基类的原始版本，而非子类的覆盖版本
- 在构造或析构函数中通过已构造完毕或尚未析构的对象调用虚函数，其多态性不受任何影响

知识讲解

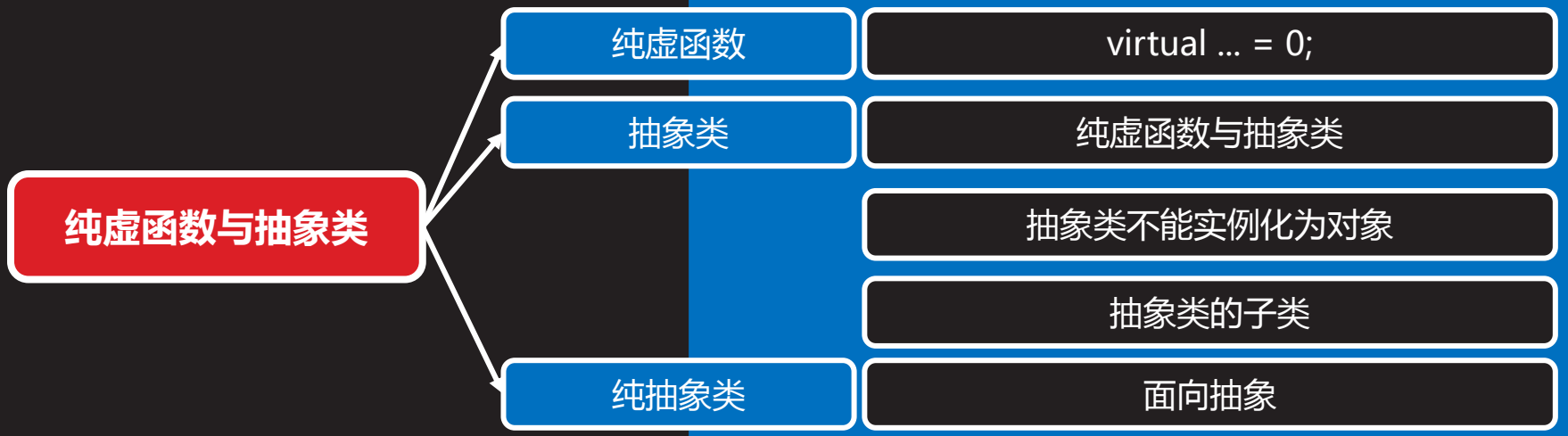
在构造和析构函数中调用虚函数

【参见：TTS COOKBOOK 】

课堂练习

- 在构造和析构函数中调用虚函数

纯虚函数与抽象类


纯虚函数与抽象类
纯虚函数
virtual ... = 0;
抽象类
纯虚函数与抽象类
抽象类不能实例化为对象
抽象类的子类
纯抽象类
面向抽象

Technology Tarena 达内科技

纯虚函数

virtual ... = 0;

- 形如
  class 类名 {
  virtual 返回类型 函数名 (形参表) = 0; };
  的虚函数，称为纯虚函数或抽象方法
  - class Shape {
    public:
    virtual void draw (void) const = 0; };
- 纯虚函数可以不定义，但如果定义，必须写在类的外部
  - void Shape::draw (void) const { ... }
- 在基类的构造和析构函数中调用纯虚函数，结果将是未定义的。通常会在链接阶段报告失败。如果该纯虚函数有定义，编译器将在给出警告之后，选择调用基类版本

Technology Tarena 达内科技

抽象类

纯虚函数与抽象类

- 至少包含一个纯虚函数的类称为抽象类
  - class Abstract {
    void foo (void) { … }
    virtual void bar (void) { … }
    virtual void hum (void) = 0;
    };
- 纯虚函数因其所代表的抽象行为而无需或无法实现，包含此种函数的类亦因其所具有的一般性而表现出抽象的特征
- 抽象类往往用来表示在对问题进行分析、设计的过程中所得出的抽象概念，是对一系列看上去不同，但本质上相同的具体概念的抽象

知识讲解

抽象类不能实例化为对象

- 无论是直接定义，还是通过new运算符，抽象类永远不能实例化为对象
  - Shape shape (...); // 错误
  - Shape* shape = new Shape (...); // 错误
  - void show (Shape shape); // 错误
  - Shape make (void); // 错误

知识讲解

抽象类的子类

- 抽象类的子类如果不对基类中的全部纯虚函数提供有效的覆盖，那么该子类就也是抽象类
  - class A { // 抽象类
    virtual void foo (void) = 0;
    virtual void bar (void) = 0;
    };
  - class B : public A { // 抽象类
    void foo (void) { ... }
    void bar (int x, int y) { ... } // 没有覆盖A::bar，继承之
    };
  - class C : public B { // 具体类
    void bar (void) { ... } // 覆盖B中从A继承的bar
    };

知识讲解

纯抽象类

面向抽象

- 全部由纯虚函数构成的抽象类称为纯抽象类或接口
- 面向抽象编程，使得所有基于接口编写的代码，在子类被更替后，无需做任何修改或只需做很少的修改，就能在新子类上正确运行
  - class Animal {
    virtual void eat (void) = 0;
    virtual void run (void) = 0;
    virtual void cry (void) = 0;
    };
  - Animal* animal = new Cat (...); // Dog, Ox, Sheep, ...
    animal->eat ();
    animal->run ();
    animal->cry ();

知识讲解

Technology Tarena 达内科技

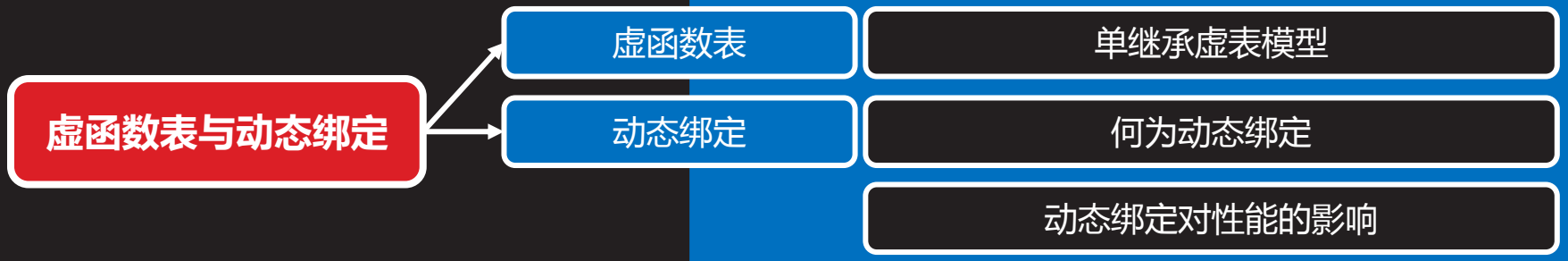
抽象类

【参见：TTS COOKBOOK 】

课堂练习

- 抽象类

虚函数表与动态绑定



Technology Tarena 达内科技

虚函数表

单继承虚表模型

- 包含虚函数的类
  - class B {
    virtual int f1 (void);
    virtual void f2 (int);
    virtual int f3 (int);
    };
- 编译器会为每个包含虚函数的类生成一张虚函数表，即存放每个虚函数地址的函数指针数组，简称虚表(vtbl)，每个虚函数对应一个虚函数表中的索引号
  - 0 -> B::f1
    1 -> B::f2
    2 -> B::f3

单继承虚表模型（续1）

- 除了为包含虚函数的类生成虚函数表以外，编译器还会为该类增加一个隐式的成员变量，通常在该类实例化对象的起始位置，用于存放虚函数表的首地址，该变量被称为虚函数表指针，简称虚指针(vptr)
- 代码：B* pb = new B; pb->f3 (12);
  将被编译为：pb->vptr[2] (pb, 12); // B::f3
  调用对象的地址被做为this指针，传递给成员函数的第一个(看不见的)形参
- 虚表是一个类一张，而不是一个对象一张，同一个类的多个对象，通过各自的虚指针，共享同一张虚表

知识讲解

单继承虚表模型（续2）

- 继承自B的子类
  - class D : public B {
    int f1 (void);
    int f3 (int);
    virtual void f4 (void);
    };
- 子类覆盖了基类的f1和f3，继承了基类的f2，增加了自己的f4，编译器同样会为子类生成一张专属于它的虚表
  - 0 -> D::f1
    1 -> B::f2
    2 -> D::f3
    3 -> D::f4

单继承虚表模型（续3）

- 指向子类虚表的虚指针就存放在子类对象的基类子对象中，通常在起始位置
- 代码：

```
B* pb = new D;
pb->f3 (12);
```

被编译为：

```
pb->vptr[2] (pb, 12); // D::f3
```

而这就是所谓的多态

知识讲解

单继承虚表模型

【参见：TTS COOKBOOK 】

课堂练习

- 单继承虚表模型

Technology Tarena 达内科技

动态绑定

何为动态绑定

- 当编译器看到通过指针或引用调用虚函数的语句时，并不急于生成有关函数调用的指令，相反它会用一段代码替代该语句，这段代码在运行时被执行，完成如下操作
  1. 确定调用指针或引用的目标对象的真实类型
  2. 从调用指针或引用的目标对象中找到虚函数表，并从虚函数表中获取所调用虚函数的入口地址
  3. 根据入口地址，调用该函数

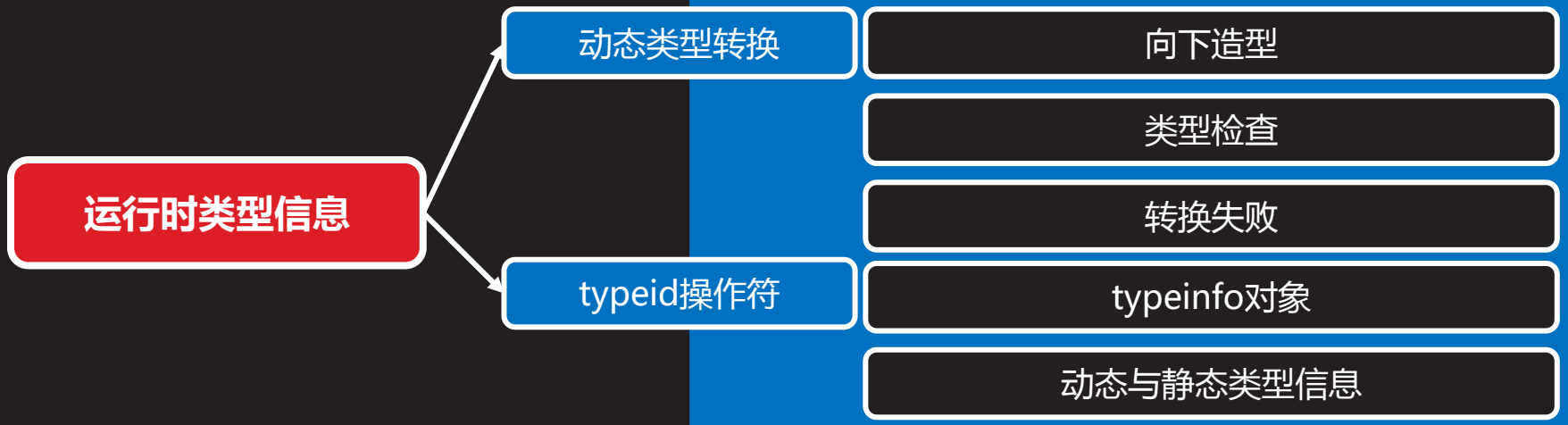
知识讲解

动态绑定对性能的影响

- 虚函数表本身会增加内存空间的开销
- 与普通函数调用相比，虚函数调用要多出几个步骤，增加运行时间的开销
- 动态绑定会妨碍编译器通过内联来优化代码
- 只有在确实需要多态特性的场合才使用虚函数，否则尽量使用普通函数

知识讲解

运行时类型信息


运行时类型信息
动态类型转换
向下造型
类型检查
转换失败
typeid操作符
typeinfo对象
动态与静态类型信息

Technology Tarena 达内科技

动态类型转换

向下造型

- 动态类型转换(dynamic_cast)用于将基类类型的指针或引用转换为其子类类型的指针或引用，前提是子类必须从基类多态继承，即基类包含至少一个虚函数
  - class A {
    virtual void foo (void) = 0;
    };
  - class B : public A { ... };
  - B b;
    A* pa = &b;
    B* pb = dynamic_cast<B*> (pa);
    A& ra = b;
    B& rb = dynamic_cast<B&> (ra);

知识讲解

类型检查

- 动态类型转换会对所需转换的基类指针或引用做检查，如果其目标确实为期望得到的子类类型的对象，则转换成功，否则转换失败
  - 不是对指针或引用做类型转换，编译错误
  - 转换目标和源不具多态继承性，编译错误
  - 转换源的目标对象非目标类型，运行错误

知识讲解

转换失败

- 针对指针的动态类型转换，以返回空指针(NULL)表示失败，针对引用的动态类型转换，以抛出bad_cast异常表示失败
  - A* pa = ...;
    B* pb = dynamic_cast<B*> (pa);
    if (pb == NULL)
        cout << "转换失败！" << endl;
  - A& ra = ...;
    try {
        B& rb = dynamic_cast<B&> (ra); }
    catch (bad_cast& ex) {
        cout << "转换失败！" << endl; }

知识讲解

动态类型转换

【参见：TTS COOKBOOK 】

课堂练习

- 动态类型转换

Technology Tarena 达内科技

typeid操作符

typeinfo对象

- typeid操作符既可用于类型也可用于对象
  - int x;
    typeid (int); typeid (x);
- typeid操作符返回typeinfo对象的常引用
  - #include <typeinfo>
  - typeinfo类的成员函数name()，返回空字符结尾的类型名
    cout << typeid (
    char (*(*[5]) (short (*) (int, long))) (float, double)
).name () << endl; // A5_PFPFcfdEPFsilEE
  - typeinfo类支持“==”和“!=”操作符，可直接用于类型相同与否的判断
    if (typeid (*human) == typeid (Student)) { … }

知识讲解

动态与静态类型信息

Technology Tarena 达内科技

- 当typeid作用于基类类型的指针或引用的目标时，若基类包含至少一个虚函数，即存在多态继承，该操作符所返回的类型信息将由该指针或引用的实际目标对象的类型决定，否则由该指针或引用本身的类型决定
  - class A {};
    class B : public A { virtual void foo (void) {} }; // 继承
    class C : public B {}; // 多态继承
  - C c;
    A& a = c;
    cout << typeid (a).name () << endl; // 1A - 静态类型
    B& b = c;
    cout << typeid (b).name () << endl; // 1C - 动态类型

知识讲解

类型信息

【参见：TTS COOKBOOK 】

课堂练习

- 类型信息

Technology Tarena 达内科技

总结和答疑