Technology Tarena 达内科技

标准C++语言

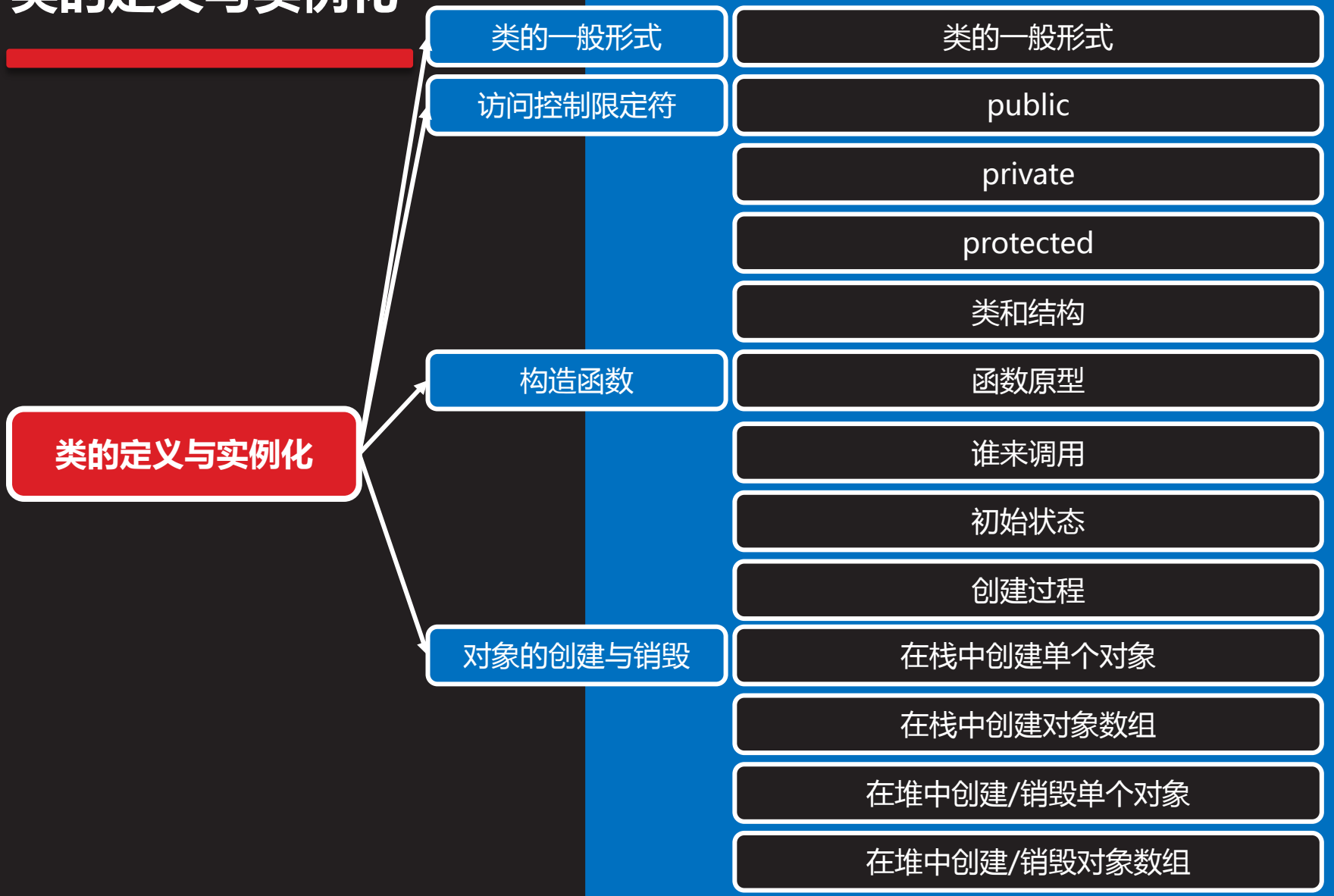
PART 1

DAY03

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	类的定义与实例化
	10:30 ~ 11:20	
	11:30 ~ 12:20	构造函数与初始化表
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	this指针与常函数
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑

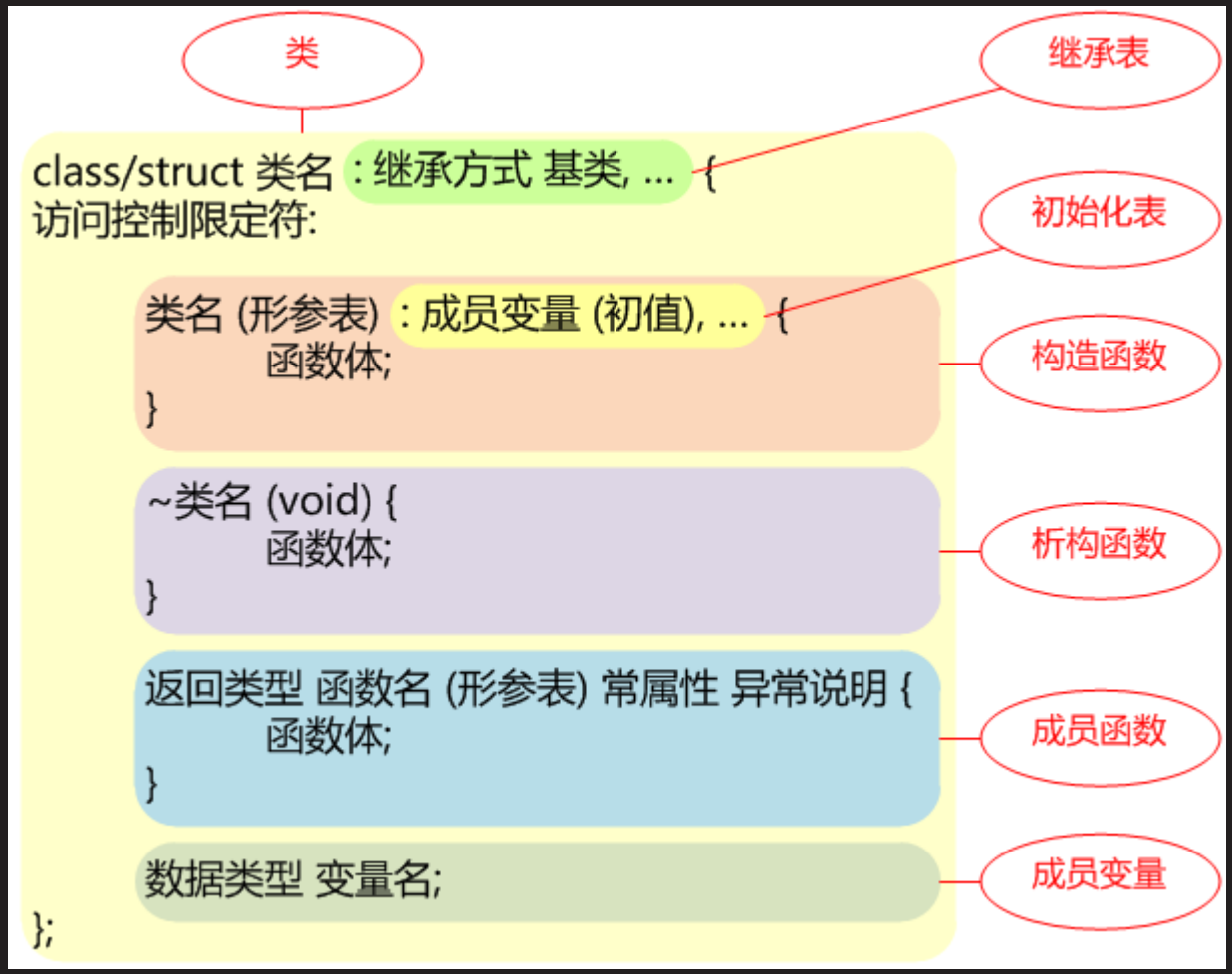
类的定义与实例化


类的定义与实例化
类的一般形式
访问控制限定符
构造函数
对象的创建与销毁
类的一般形式
public
private
protected
类和结构
函数原型
谁来调用
初始状态
创建过程
在栈中创建单个对象
在栈中创建对象数组
在堆中创建/销毁单个对象
在堆中创建/销毁对象数组

Technology Tarena 达内科技

类的一般形式

类的一般形式

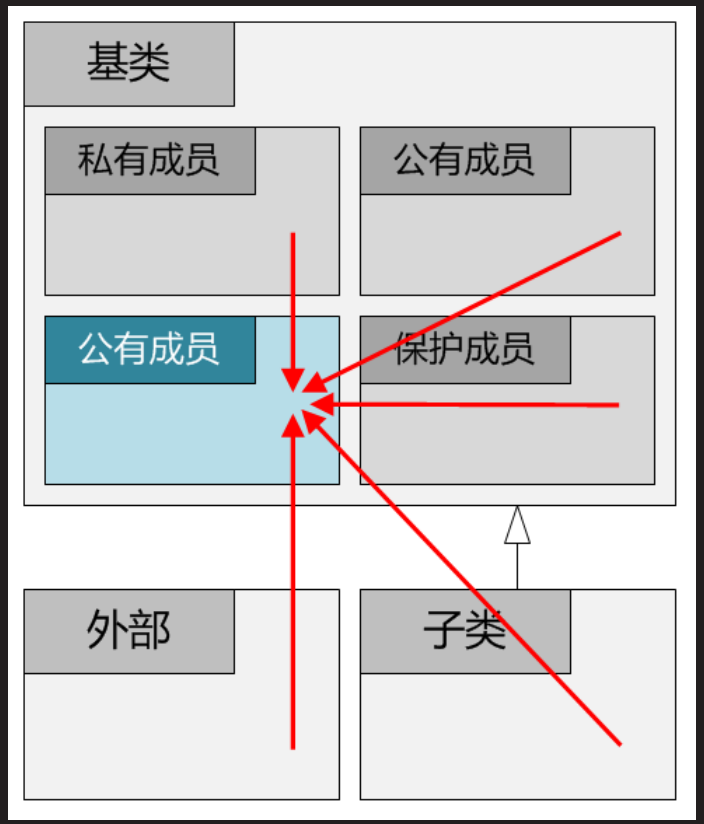


Technology Tarena 达内科技

访问控制限定符

public

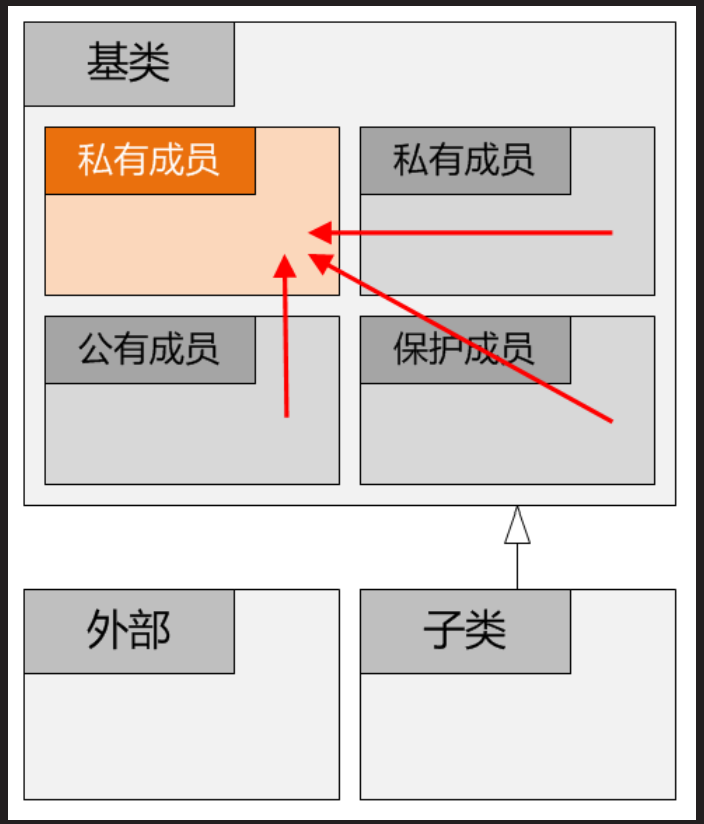
- 公有成员——谁都可以访问
  - class Dummy {
    public:
    int m_pub;
    void pub (void);
    };
  - Dummy dummy;
    dummy.m_pub = 123;
    dummy.pub ();



知识讲解

private

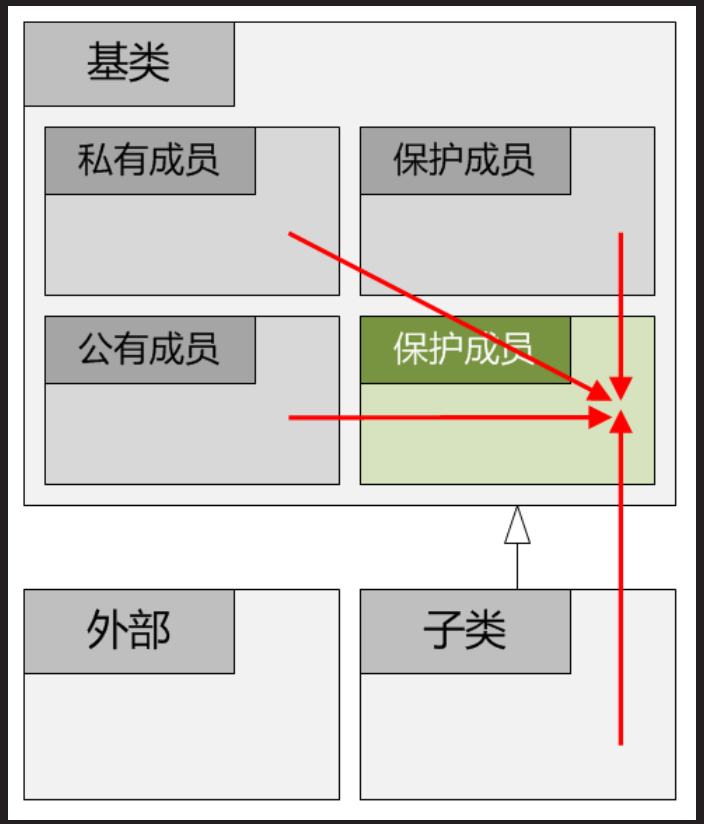
- 私有成员——只有自己可以访问
  - class Dummy {
    private:
      int m_pri;
      void pri (void);
    };
  - Dummy dummy;
    dummy.m_pri = 123; // 错误
    dummy.pri (); // 错误



知识讲解

protected

- 保护成员——只有自己和自己的子类可以访问
  - class Base {
    protected:
    int m_pro;
    void pro (void); };
  - class Derived : public Base {
    void foo (void) {
    m_pro = 123;
    pro (); } };
  - Base base;
    base.m_pro = 123; // 错误
    base.pro (); // 错误



知识讲解

类和结构

- 在C++中，类(class)和结构(struct)已没有本质性的差别，唯一的不同在于
  - 类的缺省访问控制属性为私有(private)

    class Dummy { int m_var; };

    等价于

    struct Dummy { private: int m_var; };
  - 结构的缺省访问控制属性为公有(public)

    struct Dummy { int m_var; };

    等价于

    class Dummy { public: int m_var; };

知识讲解

Technology Tarena 达内科技

构造函数

函数原型

- 函数名与类名相同，且没有返回类型
  - class 类名 {
    public:
      类名 (构造形参表) {
        构造函数体;
      }
    };
  - class User {
    public:
      User (string const& name, int age) {
        ...
      }
    };

知识讲解

谁来调用

- 构造函数在创建对象时被系统自动调用
  - 对象定义语句调用构造函数
    User user ("张飞", 25);
    User user = User ("张飞", 25);
  - new操作符调用构造函数
    User* user = new User ("张飞", 25);
- 构造函数在对象的整个生命周期内，一定会被调用，且仅被调用一次

知识讲解

初始状态

- 构造函数负责为类的成员变量赋初值，分配资源，设置对象的初始状态
  - User (string const& name, int age) {
    m_name = name;
    m_age = age;
    }
  - Array (size_t size) {
    m_array = new int[size];
    }
  - File (char const* filename, char const* mode) {
    m_fp = fopen (filename, mode);
    }

知识讲解

创建过程

- 创建一个对象要经历如下过程
  1. 用类似malloc的机制，为包括基类子对象、成员子对象等在内的整个对象分配内存空间
  2. 以构造实参调用构造函数，完成如下任务
     ① 依次调用各个基类的构造函数，初始化所有基类子对象
     ② 依次调用类类型成员的构造函数，初始化所有成员子对象
     ③ 执行构造函数体代码
- 注意，执行构造函数体代码是整个构造过程的最后一步，这保证了构造函数体代码所依赖的一切资源和先决条件，在该代码被执行时已经准备充分，并获得正确地初始化

知识讲解

Technology Tarena 达内科技

对象的创建与销毁

在栈中创建单个对象

- 类名 对象; // 注意不要加空括号
  类名 对象 (构造实参表);
  类名 对象 = 类名 (构造实参表);
  - User user;
    User user (); // ERROR
    User user ("张飞", 25);
    User user = User ("张飞", 25);

知识讲解

在栈中创建对象数组

- 类名 对象数组[元素个数];
  类名 对象数组[元素个数] = {类名 (构造实参表), ...};
  类名 对象数组[] = {类名 (构造实参表), ...};
  - User users[3];
    User users[4] = {User ("张飞", 25), User ("关羽", 30),
    User ("赵云", 20)};
    User users[] = {User ("张飞", 25), User ("关羽", 30),
    User ("赵云", 20)};
- 注意，凡是未指明构造方式的对象，无论是单个对象还是数组中的元素对象，一律通过无参构造函数初始化

知识讲解

在堆中创建/销毁单个对象

- 类名* 对象指针 = new 类名;
  类名* 对象指针 = new 类名 ();
  类名* 对象指针 = new 类名 (构造实参表);
  delete 对象指针;
  - User* user = new User;
    User* user = new User ();
    User* user = new User ("张飞", 25);
    delete user;
- 注意，new和malloc相比最大的区别就在于，它除了分配内存之外，还会调用构造函数初始化，malloc却不会
  - User* user = (User*)malloc (sizeof (User));
    user->User ("张飞", 25); // 形式代码

知识讲解

在堆中创建/销毁对象数组

- 类名* 对象指针 = new 类名[元素个数];
  类名* 对象指针 = new 类名[元素个数] {
  类名 (构造实参表), ...};
  - User* users = new User[3];
    User* users = new User[4] {User ("张飞", 25),
    User ("关羽", 30), User ("赵云", 20)}; // C++11
    delete[] users;
- 注意，以数组方式new的一定要以数组方式delete，即通过delete[]操作符销毁堆中的对象数组，以免发生异常

知识讲解

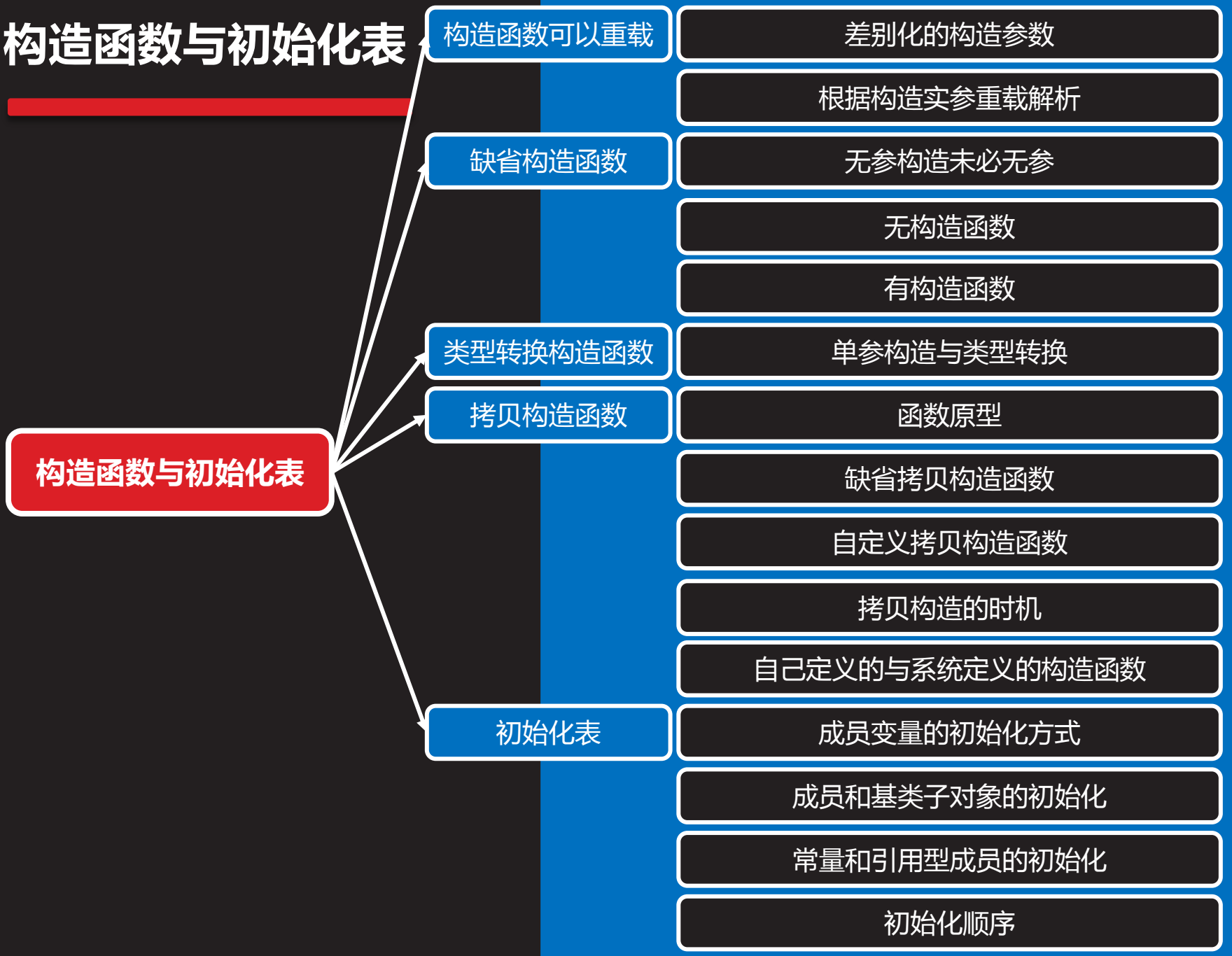
定义用户类并实例化为对象

【参见：TTS COOKBOOK 】

课堂练习

- 定义用户类并实例化为对象

构造函数与初始化表


构造函数与初始化表
构造函数可以重载
差别化的构造参数
根据构造实参重载解析
缺省构造函数
无参构造未必无参
无构造函数
有构造函数
类型转换构造函数
单参构造与类型转换
拷贝构造函数
函数原型
缺省拷贝构造函数
自定义拷贝构造函数
拷贝构造的时机
自己定义的与系统定义的构造函数
初始化表
成员变量的初始化方式
成员和基类子对象的初始化
常量和引用型成员的初始化
初始化顺序

构造函数可以重载

差别化的构造参数

- 构造函数也可以通过参数表的差别化形成重载
  - class User {
    public:
      User (void) { // ①
        m_name = "";
        m_age = 0;
      }
      User (string const& name, int age) { // ②
        m_name = name;
        m_age = age;
      }
    };

知识讲解

根据构造实参重载解析

- 重载的构造函数通过构造实参的类型选择匹配
  - User user; // ->①
    User user = User (); // ->①
    User user ("张飞", 25); // ->②
    User user = User ("张飞", 25); // ->②
  - User* user = new User; // ->①
    User* user = new User (); // ->①
    User* user = new User ("张飞", 25); // ->②

知识讲解

类的构造函数可以重载

【参见：TTS COOKBOOK 】

课堂练习

- 类的构造函数可以重载

Technology Tarena 达内科技

缺省构造函数

无参构造未必无参

- 缺省构造函数亦称无参构造函数，但其未必真的没有任何参数，为一个有参构造函数的每个参数都提供一个缺省值，同样可以达到无参构造函数的效果
  - Complex (void);
  - Complex (int r = 0, int i = 0);
- “缺省构造”着力表现，在不提供任何定制信息(即参数)的情况下，对象被初始化为何种缺省状态

知识讲解

无构造函数

- 如果一个类没有定义任何构造函数，那么编译器会为其提供一个缺省构造函数
  - 对基本类型的成员变量，不做初始化
  - 对类类型的成员变量和基类子对象，调用相应类型的缺省构造函数初始化
- 所谓“编译器提供的某某函数”其实并非语法意义上的函数，而是功能意义上的函数。编译器作为可执行指令的生成者，直接生成完成特定功能的机器(汇编)指令即可，完全没有必要再借助高级语言意义上的函数完成此任务

知识讲解

有构造函数

- 对于已经定义至少一个构造函数的类，无论其构造函数是否带有参数，编译器都不会再为其提供缺省构造函数
  - class Complex {
    public:
    Complex (int r, int i);
    };
- 这种情况下，若该类需要支持以缺省方式构造对象，则必须自己定义缺省构造函数，否则将导致编译错误
  - Complex c1 (123, 456);
    Complex c2; // 错误

知识讲解

Technology Tarena 达内科技

缺省构造函数

【参见：TTS COOKBOOK 】

课堂练习

- 缺省构造函数

Technology Tarena 达内科技

类型转换构造函数

单参构造与类型转换

- 在目标类型中，可以接受单个源类型对象实参的构造函数，支持从源类型到目标类型的隐式类型转换
  - class 目标类型 {
    目标类型 (源类型 const& src) { ... }
    };
- 通过explicit关键字，可以强制这种通过构造函数实现的类型转换必须显式地进行
  - class 目标类型 {
    explicit 目标类型 (源类型 const& src) { ... }
    };

知识讲解

支持自定义类型转换的构造函数

【参见：TTS COOKBOOK 】

课堂练习

- 支持自定义类型转换的构造函数

拷贝构造函数

函数原型

- 形如

```
class 类名 {
    类名 (类名 const& that) { // 注意这里必须是引用
        克隆源对象that的副本对象;
    }
};
```

的构造函数被称为拷贝构造函数，用于从一个已定义的对象构造其同类型的副本对象，即对象克隆

知识讲解

缺省拷贝构造函数

- 如果一个类没有定义拷贝构造函数，那么编译器会为其提供一个缺省拷贝构造函数
  - 对基本类型的成员变量，按字节复制
  - 对类类型的成员变量和基类子对象，调用相应类型的拷贝构造函数
  - class User {
    string m_name; // 调用string类的拷贝构造函数
    int m_age; // 按字节复制
    char m_mbox[256]; // 按字节复制
    };
  - User user1 { ... };
    User user2 = user1;

知识讲解

自定义拷贝构造函数

- 如果自己定义了拷贝构造函数，那么编译器将不再提供缺省拷贝构造函数
- 这种情况下，所有与对象复制有关的操作，都必须在自定义拷贝构造函数中通过编写代码完成
  - User (User const& that) {
    m_name = that.m_name;
    m_age = that.m_age;
    strcpy (m_mbox, that.m_mbox);
    }
- 除特殊情况外，编译器提供的缺省拷贝构造函数已足够适用，这时无需自己定义拷贝构造函数

知识讲解

拷贝构造的时机

- 拷贝构造过程通常出现在如下语境中
  - 用已定义的对象作为同类型对象的构造实参
    User user1 = user2;
  - 以对象(而非其指针或引用)的形式向函数传递参数
    void foo (User user) { … }
    foo (user);
  - 从函数中返回对象(而非其指针或引用)
    User foo (void) { … }
    foo ();
- 拷贝构造过程风险高而效率低，设计时应尽可能避免
- 编译器会通过必要的优化策略，减少拷贝构造的机会
  - 通过-fno-elide-constructors选项可关闭此优化特性

知识讲解

自己定义的与系统定义的构造函数

- 任何一个类，在任何情况下都不可能没有任何构造函数
- 一个类只有在拥有自定义拷贝构造函数的情况下，才有可能仅包含唯一的构造函数
- 所有系统定义的构造函数，其访问控制属性都是公有的

知识讲解

自定义构造函数	系统定义构造函数
无	缺省构造函数 缺省拷贝构造函数
除拷贝构造函数以外的任何构造函数	缺省拷贝构造函数
拷贝构造函数	无

拷贝构造函数

【参见：TTS COOKBOOK 】

课堂练习

- 拷贝构造函数

Technology Tarena 达内科技

初始化表

成员变量的初始化方式

- 通过在类的构造函数中使用初始化表，指明该类的成员变量如何被初始化
  - User (string const& name, int age) :
    m_name (name), m_age (age) {}
  - User (void) : m_name (""), m_age (0) {}

知识讲解

成员和基类子对象的初始化

- 类的类类型成员变量和基类子对象，要么在初始化表中显式初始化，要么通过相应类型的缺省构造函数初始化
  - class Date {
    Date (int year, int mon, int day) { ... }
    };
  - class User {
    User (string const& name,
    int year, int mon, int day) : m_name (name),
    m_bday (year, mon, day) {}
    string m_name;
    Date m_bday;
    };

知识讲解

常量和引用型成员的初始化

- 类的常量型和引用型成员变量，必须在初始化表中显式初始化，不能在构造函数体中赋初值
  - class Circle {
    public:
    Circle (double pi, double& r) : m_pi (pi), m_r (r) {}
    double perimeter (void) { return 2 * m_pi * m_r; }
    double area (void) { return m_pi * m_r * m_r; }
    private:
    double const m_pi; // 常量型成员变量
    double& m_r; // 引用型成员变量
    };

知识讲解

初始化顺序

- 类的成员变量按其在类中的声明顺序依次被初始化，而与其在初始化表中的排列顺序无关
  - class Message {
    public:
        Message (string const& str) :
            m_str (str), m_len (m_str.length ()) {} // BUG
    private:
        size_t m_len;
        string m_str;
    };

知识讲解

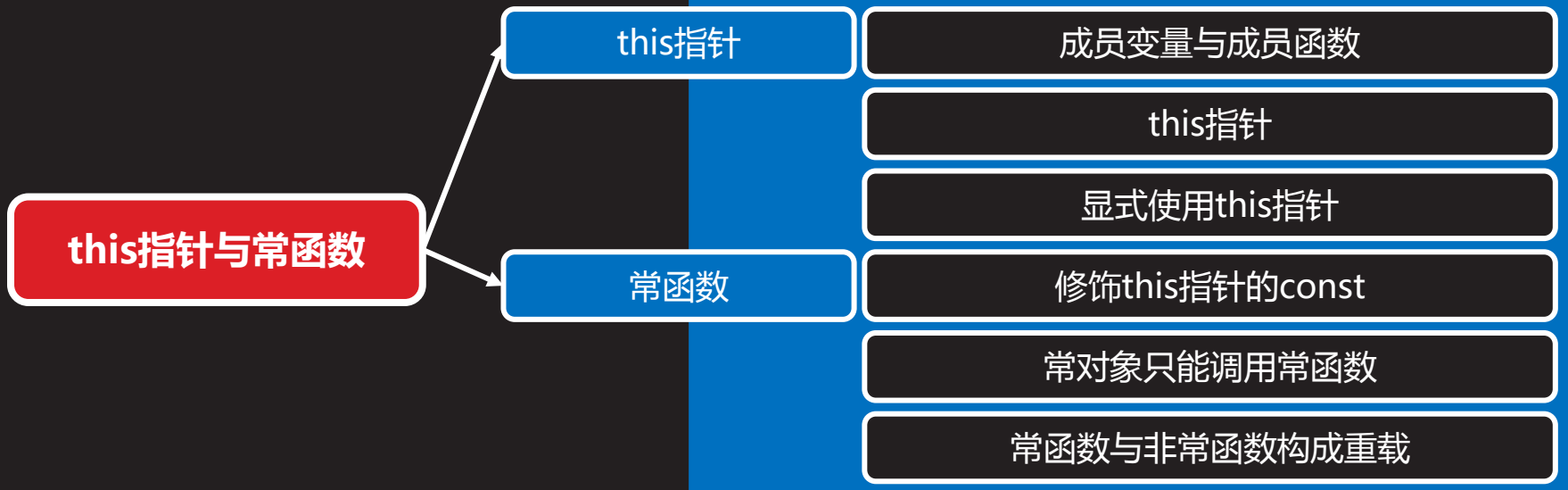
初始化表

【参见：TTS COOKBOOK 】

课堂练习

- 初始化表

this指针与常函数

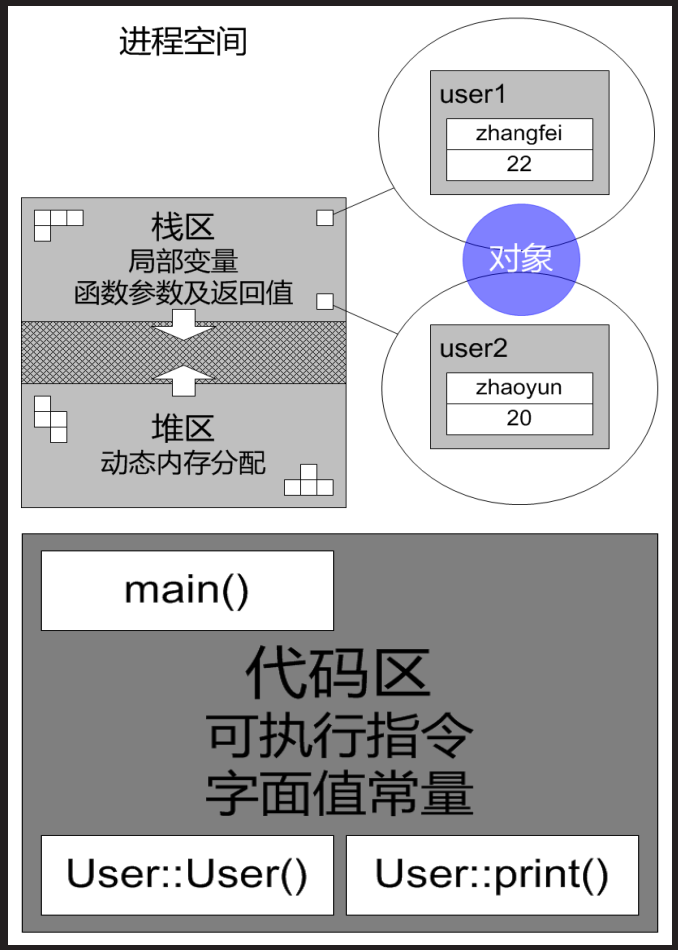

this指针与常函数
this指针
常函数
成员变量与成员函数
this指针
显式使用this指针
修饰this指针的const
常对象只能调用常函数
常函数与非常函数构成重载

Technology Tarena 达内科技

this指针

成员变量与成员函数

- 相同类型的不同对象各自拥有独立的成员变量实例
- 相同类型的不同对象彼此共享同一份成员函数代码
- 为相同类型的不同对象所共享的成员函数，如何区分所访问的成员变量隶属于哪个对象？
  - void User::print (void) {
    cout << m_name << ", "
    << m_age << endl; }
  - User user1 ("zhangfei", 22);
    User user2 ("zhaoyun", 20);
    user1.print (); // zhangfei, 22
    user2.print (); // zhaoyun, 20



知识讲解

this指针

- 类的每个成员函数都有一个隐藏的指针型参数this，指向调用该成员函数的对象，这就是this指针
  - void User::print (void) { ... }
    void _ZN4User5printEv (User* this) { ... }
  - user1.print ();
    _ZN4User5printEv (&user1);
- 类的构造和析构函数中同样有this指针，指向这个正在被构造或析构的对象
- 在类的成员函数、构造函数和析构函数中，对所有成员的访问，都是通过this指针进行的
- 事实上，在类的成员函数、构造函数和析构函数中，也可以显式地通过this指针访问该类的成员

知识讲解

显式使用this指针

- 多数情况下，程序并不需要显式地使用this指针，编译器通过名字解析，可以判断出所访问的标识符是否属于该类的成员，是则为其加上“this->”
- 在构造函数内部，除了使用初始化表外，还可以通过this指针，将同名的成员与非成员区分开来
- 在类的成员函数中，通过this指针向该成员函数的调用者，返回调用对象的自拷贝或自引用
- 将this指针作为函数的参数，从一个对象传递给另一个对象，可以实现对象间的交互

知识讲解

显式使用this指针

【参见：TTS COOKBOOK 】

课堂练习

- 显式使用this指针

Technology Tarena 达内科技

常函数

修饰this指针的const

- 在类成员函数的形参表之后，函数体之前加上const关键字，该成员函数的this指针即具有常属性，这样的成员函数被称为常函数
- 在常函数内部，因为this指针被声明为常量指针，所以无法修改成员变量的值，除非该成员变量被mutable关键字修饰
  - class User { ...; mutable unsigned int m_times; };
  - void User::print (void) const {

```
cout << m_name << ", " << m_age << endl;
m_name = ""; // 编译错误
m_age++; // 编译错误
++m_times; }
```

知识讲解

常对象只能调用常函数

- 通过常对象调用成员函数，传递给该成员函数this指针参数的实参是一个常量指针
  - User user (...);
    User const* cptr = &user;
    User const& cref = user;
  - void User::modify (void) { ... }
    void _ZN4User6modifyEv (User* this) { ... }
    void User::print (void) const { ... }
    void _ZNK4User5printEv (User const* this) { ... }
  - cptr->modify (); // 错误
    _ZN4User6modifyEv (cptr);
    cref.print ();
    _ZNK4User5printEv (&cref);

知识讲解

常函数和mutable关键字

【参见：TTS COOKBOOK 】

课堂练习

- 常函数和mutable关键字

常函数与非常函数构成重载

- 原型相同的成员函数，常版本和非常版本构成重载
  - class Array {
    int const& at (size_t) const;
    int& at (size_t i); };
  - Array array (...);
- 常对象选择常版本
  - Array const& cref;
    cout << cref.at (0) << endl;
- 非常对象选择非常版本
  - array.at (0) = 100;
- 如果没有非常版本，非常对象也能选择常版本
  - cout << array.at (0) << endl;

常函数与非常函数的重载匹配

【参见：TTS COOKBOOK 】

课堂练习

- 常函数与非常函数的重载匹配

Technology Tarena 达内科技

总结和答疑