

封装

胡船长

初航我带你，远航靠自己

一、类的基本封装

1. 初次相识：类与对象
2. 编码安全：访问控制权限
3. 生命周期：构造函数与析构函数
4. 何以为我：属性和方法

二、运算符重载

1. 函数重载
2. 类外运算符重载基础
3. 类内运算符重载基础
4. 函数对象、数组对象与指针对象
5. 总结：运算符重载

三、附加内容：返回值优化

1. RVO 返回值优化
2. NRVO 命名返回值优化
3. 优化前后的效果对比

一、类的基本封装

1. 初次相识：类与对象
2. 编码安全：访问控制权限
3. 生命周期：构造函数与析构函数
4. 何以为我：属性和方法

类型与变量

类型	变量	最小位数
int	a	16
long long	b	64
char	c	8
double	d	N/A
float	e	N/A

```
int a;  
long long b;  
char c;  
double d;  
float e;
```

类型与变量

类型 = **类型数据** + **类型操作**

类与对象

类	对象
Cat	garfield
Dog	odie
People	hug

```
class Cat {  
};  
  
class Dog {  
};  
  
class People {  
};
```

```
Cat garfield;  
Dog odie;  
People hug;
```


成员属性与方法

```
class People {  
    string name;  
    Day birthday;  
    double height;  
    double weight;  
  
    void say(string word);  
    void run(Location &loc);  
};
```

一、类的基本封装

1. 初次相识：类与对象
2. **编码安全：访问控制权限**
3. 生命周期：构造函数与析构函数
4. 何以为我：属性和方法

访问权限

```
class People {  
public :  
    string name();  
    string birthday();  
    double height();  
    double weight();  
  
    void say(string word);  
    void run(Location &loc);  
private :  
    string __name;  
    Day __birthday;  
    double __height;  
    double __weight;  
};
```

public	公共访问权限
private	私有访问权限
protected	受保护的访问权限
friend	定义友元 [函数/类]

C++中的结构体与类

struct 访问权限默认为 **public**

class 访问权限默认为 **private**

一、类的基本封装

1. 初次相识：类与对象
2. 编码安全：访问控制权限
3. **生命周期：构造函数与析构函数**
4. 何以为我：属性和方法

3-1. 一个『对象』的生命周期

一个『对象』的生命周期

一个『对象』的生命周期



一个『对象』的生命周期



一个『对象』的生命周期



一个『对象』的生命周期



构造函数与析构函数

构造/析构函数	使用方式
默认构造函数	People a;
People(string name);	People a("hug");
People(const People &a)	拷贝构造, 与 = 不等价
People(People &&a)	移动构造, C++重回神坛的关键
~People();	无

初始化列表：基础使用

初始化列表： 注意事项

为什么要使用初始化列表

3-2. 左值与右值

『值』与『引用』

代码1

```
int a = 123;  
int b = a;
```

代码2

```
int a = 123;  
int &b = a;
```

『值』与『引用』

代码1

```
int a = 123;  
int b = a;
```

a 123

代码2

```
int a = 123;  
int &b = a;
```

『值』与『引用』

代码1

```
int a = 123;  
int b = a;
```

a 123

b

代码2

```
int a = 123;  
int &b = a;
```

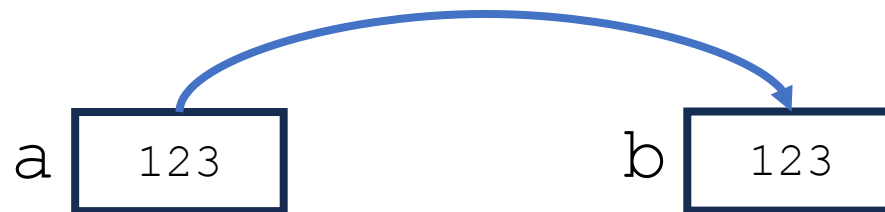
『值』与『引用』

代码1

```
int a = 123;  
int b = a;
```

代码2

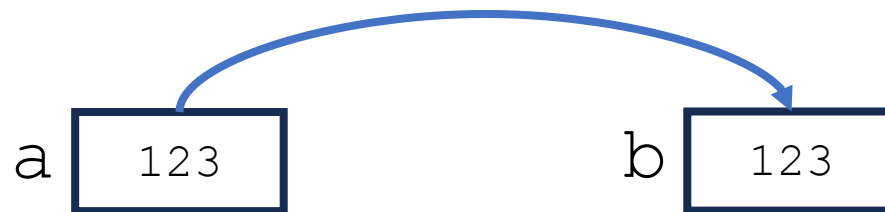
```
int a = 123;  
int &b = a;
```



『值』与『引用』

代码1

```
int a = 123;  
int b = a;
```



代码2

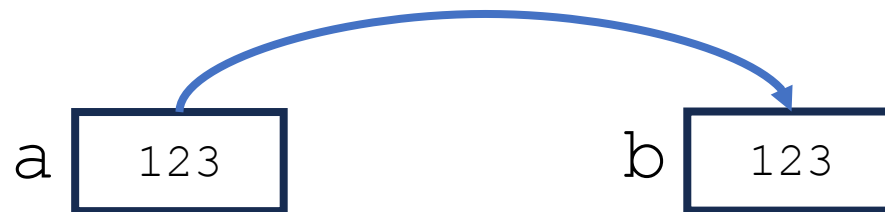
```
int a = 123;  
int &b = a;
```



『值』与『引用』

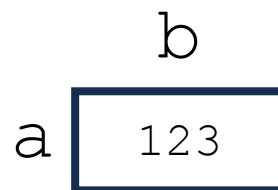
代码1

```
int a = 123;  
int b = a;
```



代码2

```
int a = 123;  
int &b = a;
```



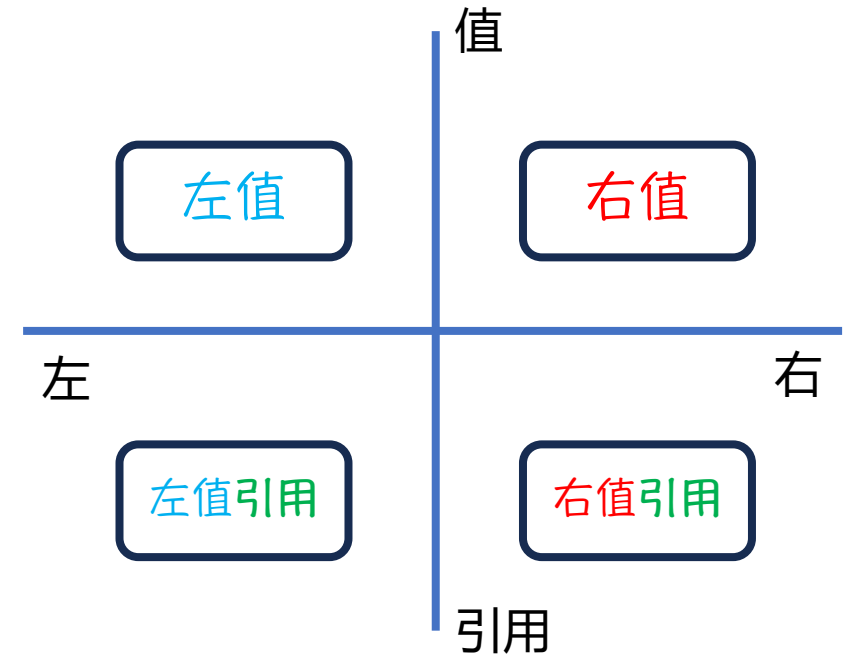
『左值』与『右值』

1. 左值：持续存在的，可以取地址，可以出现在赋值语句左侧
2. 右值：字面量或匿名对象，不能出现在赋值语句左侧
3. 学习诀窍1：字面量一定是右值
4. 学习诀窍2：是否可以通过单一变量访问

『左值』与『右值』：函数传参问题

C++ 引用：总结

1. 区分三个概念：实参、形参、引用参数
2. 引用需要在定义时，完成『绑定』
3. 左值、右值、左值引用，右值引用



3-3. 默认构造函数

3-4-1. 有参构造函数

3-4-2. 有参构造函数：转换构造

3-5. 拷贝构造函数

思考题：拷贝构造函数为什么传入引用？

『深拷贝』与『浅拷贝』问题

3-5. 移动构造函数

3-6. 析构函数

『构造函数』与『析构函数』的执行顺序

执行顺序总结

1. 对象之间：先构造的，后析构
2. 对象属性：先于对象构造，晚于对象析构

3-7. delete 与 default 的作用

3-8. new 、 delete 与 emplace new

随堂练习1

设计一个不能被自动创建的对象

随堂练习2

设计一个只能在『特定函数』中被自动创建的对象

一、类的基本封装

1. 初次相识：类与对象
2. 编码安全：访问控制权限
3. 生命周期：构造函数与析构函数
4. 何以为我：属性和方法

成员属性与方法

```
class People {  
    string name;  
    Day birthday;  
    double height;  
    double weight;  
  
    void say(string word);  
    void run(Location &loc);  
};
```

类属性与方法

```
class People {
public :
    void say(string word);
    void run(Location &loc);

    static void is_valid_height(double height);

private :
    static int total_num;

    string __name;
    Day __birthday;
    double __height;
    double __weight;
};
```

const 方法

```
class People {  
public :  
    void say(string word);  
    void run(Location &loc);  
    -----  
    string &name() const;  
    -----  
    static void is_valid_height(double height);  
  
private :  
    static int total_num;  
  
    string __name;  
    Day __birthday;  
    double __height;  
    double __weight;  
};
```

二、运算符重载

1. 函数重载
2. 类外运算符重载基础
3. 类内运算符重载基础
4. 函数对象、数组对象与指针对象
5. 总结：运算符重载

函数重载

如果一个作用域内几个函数名字相同但是参数列表不同，称为函数重载

与返回值没有关系

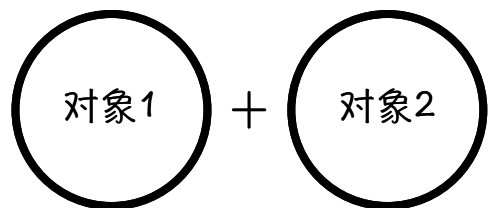
函数重载

1. 通过函数名对函数功能进行提示
2. 通过函数参数列表对函数的用法进行提示
3. 扩展已有的功能

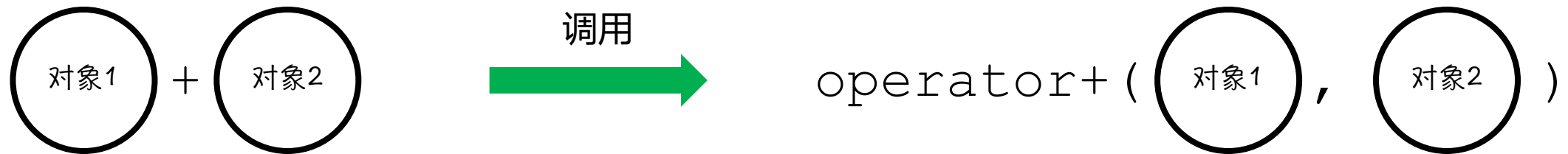
二、运算符重载

1. 函数重载
2. 类外运算符重载基础
3. 类内运算符重载基础
4. 函数对象、数组对象与指针对象
5. 总结：运算符重载

类外运算符重载



类外运算符重载



什么运算符能重载?

+	-	*	/	%	^
&		~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	
+=	-=	*=	/=	^=	&=
=	%=	<<=	>>=	[]	()
->	->*	new	new[]	delete	delete[]

:: **.*** **.** **? :** **sizeof**

随堂练习3

让 cout 输出用户自定义数据类型

二、运算符重载

1. 函数重载
2. 类外运算符重载基础
- 3. 类内运算符重载基础**
4. 函数对象、数组对象与指针对象
5. 总结：运算符重载

类内运算符重载

`operator+ ((对象1), (对象2))`

类外重载

类内运算符重载

`operator+ ((对象1), (对象2))`

类外重载



类内重载

二、运算符重载

1. 函数重载
2. 类外运算符重载基础
3. 类内运算符重载基础
4. 函数对象、数组对象与指针对象
5. 总结：运算符重载

4-1. 函数对象

4-2. 数组对象

4-3. 指针对象

二、运算符重载

1. 类外运算符重载基础
2. 拓展 cout 的输出行为
3. 类内运算符重载基础
4. 函数对象、数组对象与指针对象
5. **总结：运算符重载**

总结：运算符重载

1. 『运算符重载』基于『函数重载』的语言特性
2. 可以在类内重载，也可以在类外重载
3. ()为『函数对象』、[]为『数组对象』、->为『指针对象』
4. ()、[]、->、= 四个运算符只能在类内重载
5. 不能重载的5个运算符：
 1. :: 作用域操作符
 2. . 成员引用运算符
 3. .* 成员指针引用运算符
 4. ?: 唯一的三目运算符
 5. sizeof 运算符

随堂练习4

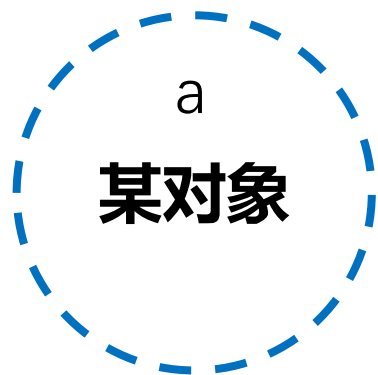
实现一个存储整型的 `vector` 类型

三、附加内容：返回值优化

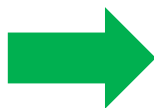
1. RVO 返回值优化
2. NRVO 命名返回值优化
3. 优化前后的效果对比

对象的初始化

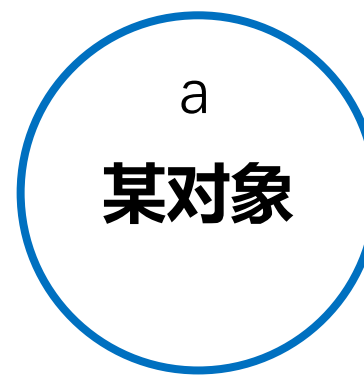
```
SomeClass a;
```



开辟对象数据区



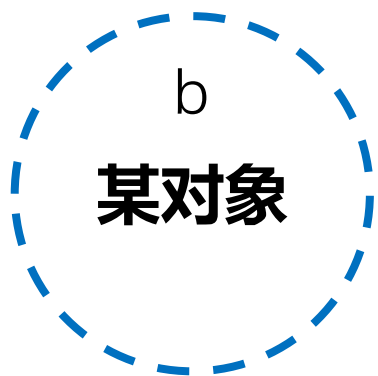
匹配构造函数



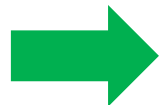
完成构造

对象的初始化

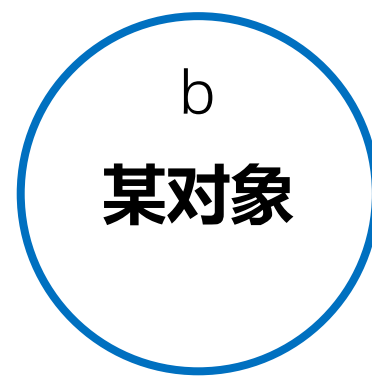
```
SomeClass a;  
SomeClass b = a;
```



开辟对象数据区



匹配拷贝构造函数

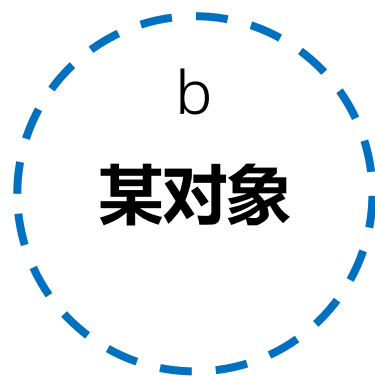


完成构造

函数形参的初始化

```
void func(SomeClass b);
```

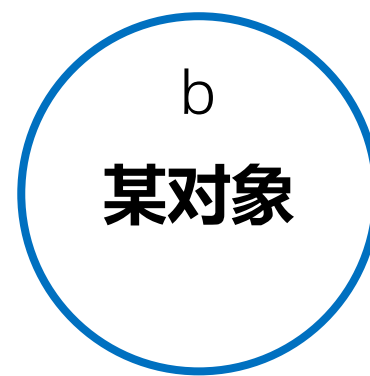
```
SomeClass a;  
func(a);
```



开辟对象数据区



匹配拷贝构造函数



完成构造

正常的拷贝过程

```
People func() {  
    People temp_a("temp name");  
    return temp_a;  
}  
  
int main() {  
    People a = func();  
    return 0;  
}
```

Step 1 : 开辟 a 对象数据区

Step 2 : 调用函数 func

Step 3 : 开辟对象 temp_a 数据区

Step 4 : 调用 temp_a 对象的构造函数

Step 5 : 使用 temp_a 调用临时匿名变量的拷贝构造函数

Step 6 : 销毁 temp_a 对象

Step 7 : 使用临时匿名变量调用 a 的拷贝构造函数

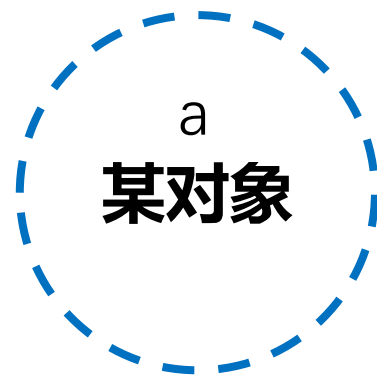
Step 8 : 销毁临时匿名变量

Step 9 : 销毁 a 对象

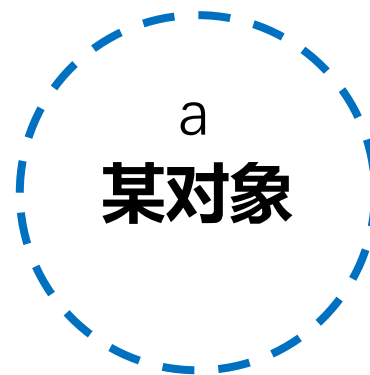
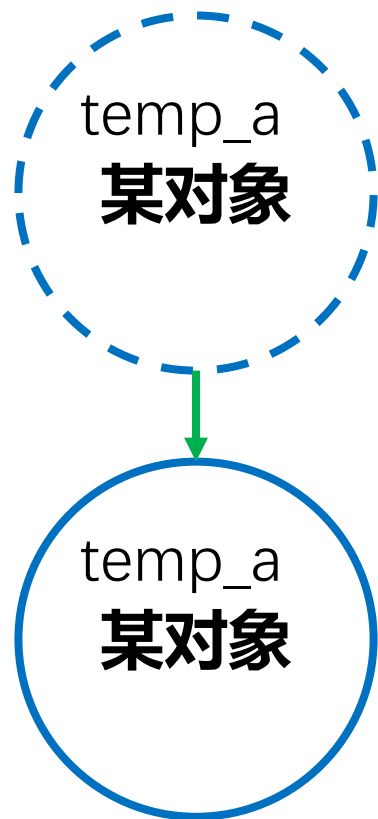
正常的拷贝过程



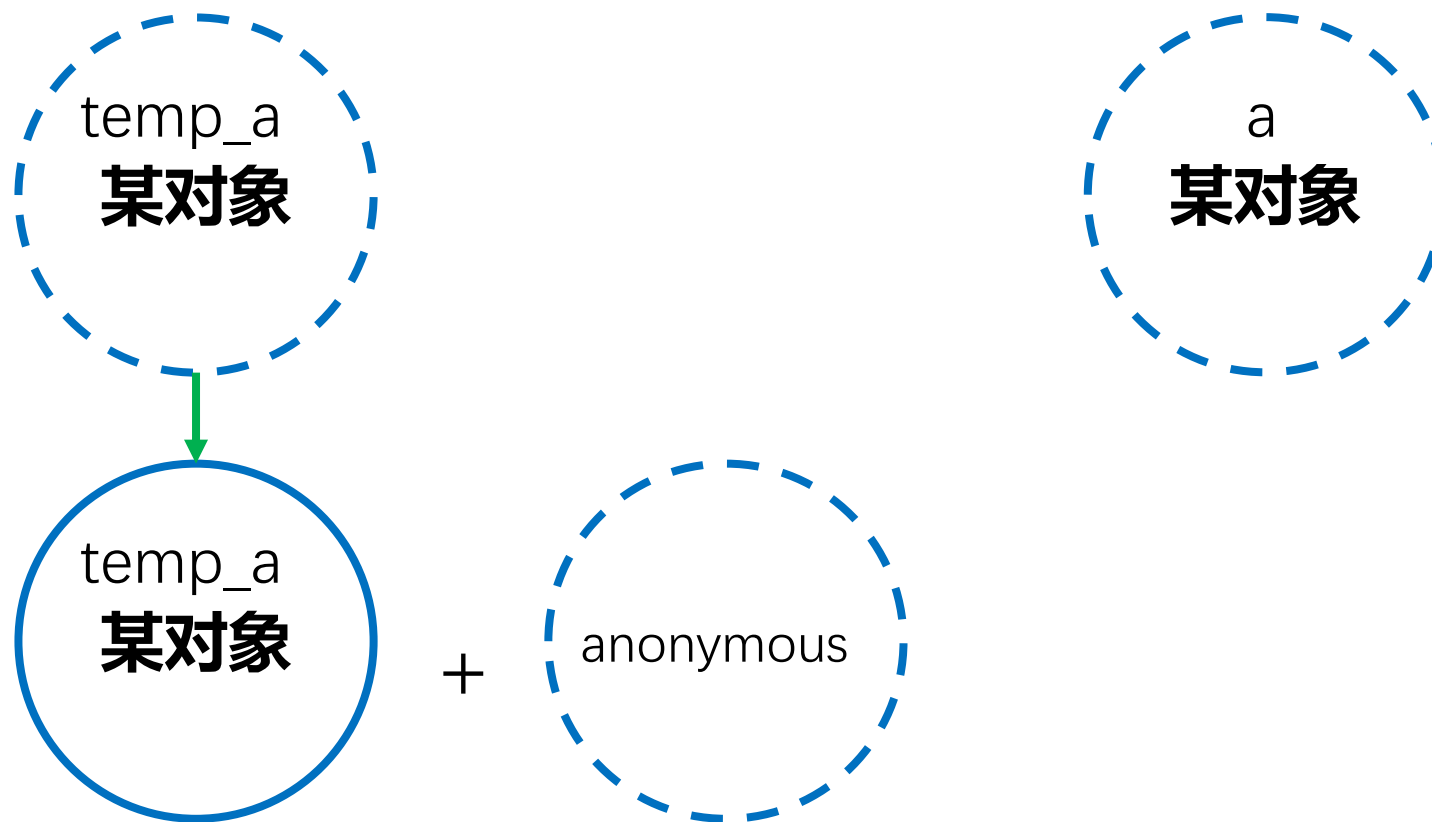
正常的拷贝过程



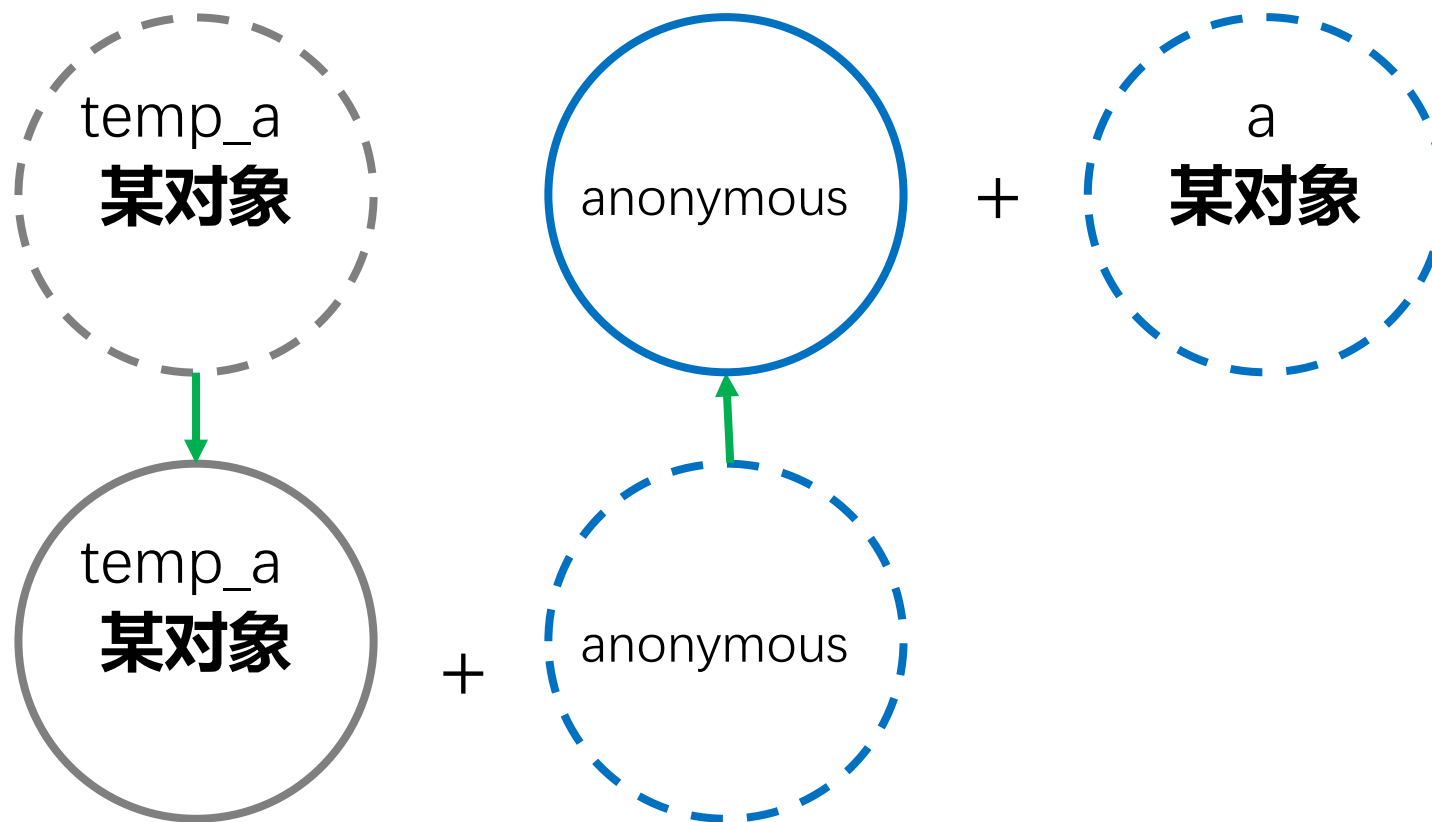
正常的拷贝过程



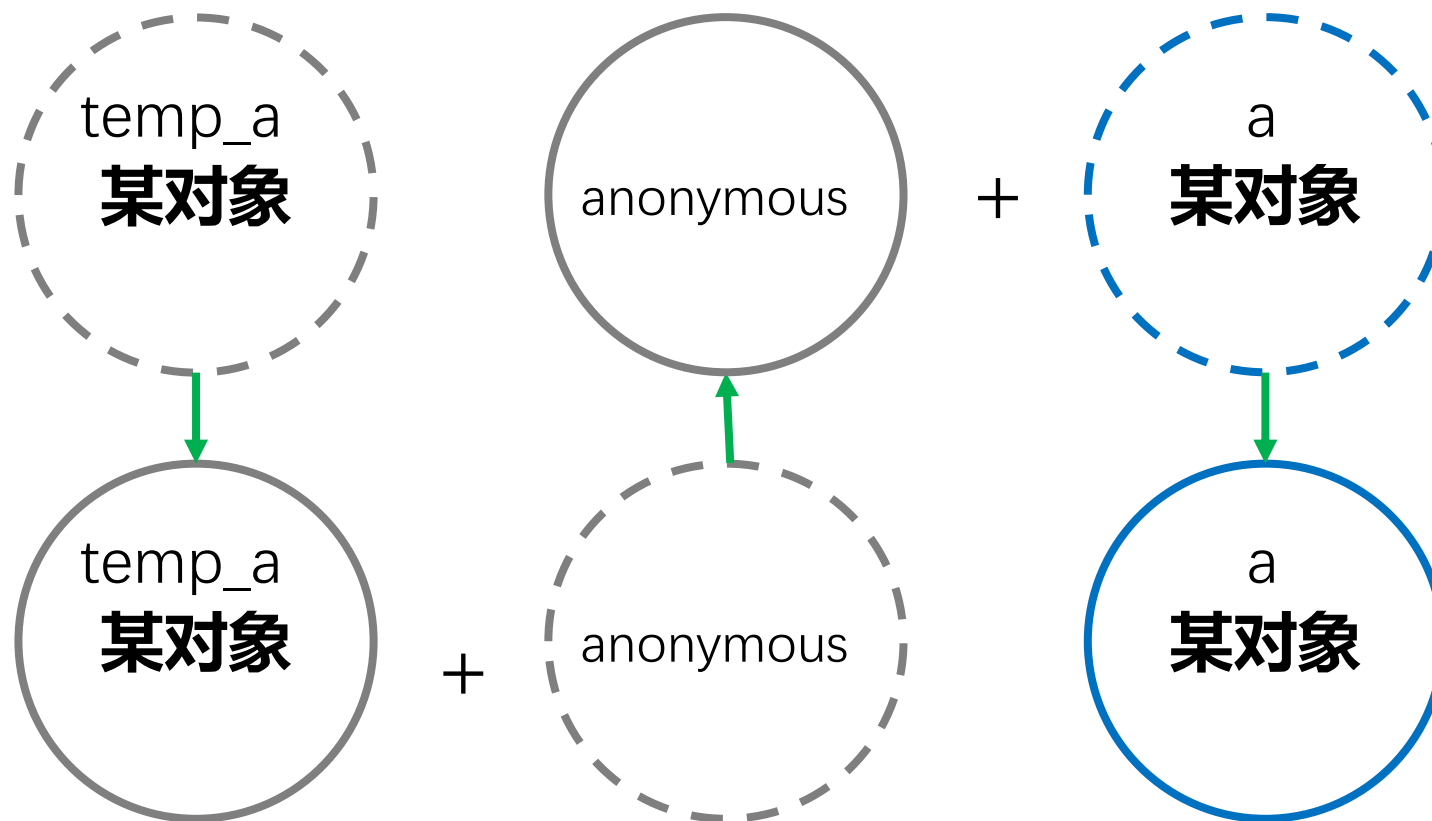
正常的拷贝过程



正常的拷贝过程



正常的拷贝过程



返回值优化 (RVO)

```
People func() {  
    People temp_a("temp name");  
    return temp_a;  
}  
  
int main() {  
    People a = func();  
    return 0;  
}
```

Step 1 : 开辟 a 对象数据区

Step 2 : 调用函数 func

Step 3 : 开辟对象 temp_a 数据区

Step 4 : 调用 temp_a 对象的构造函数

Step 5 : 使用 temp_a 调用临时匿名变量的拷贝构造函数

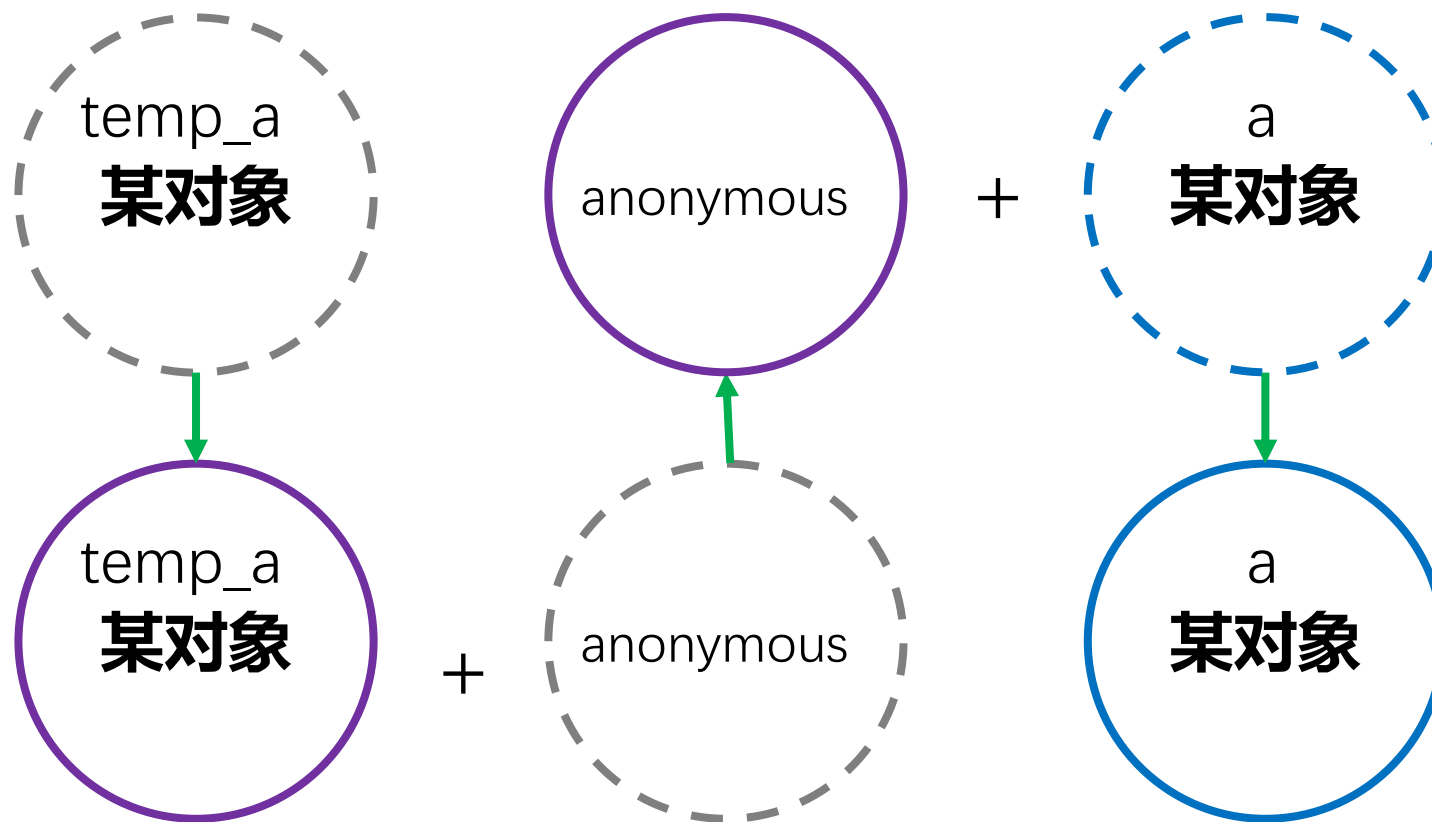
Step 6 : 销毁 temp_a 对象

Step 7 : 使用临时匿名变量调用 a 的拷贝构造函数

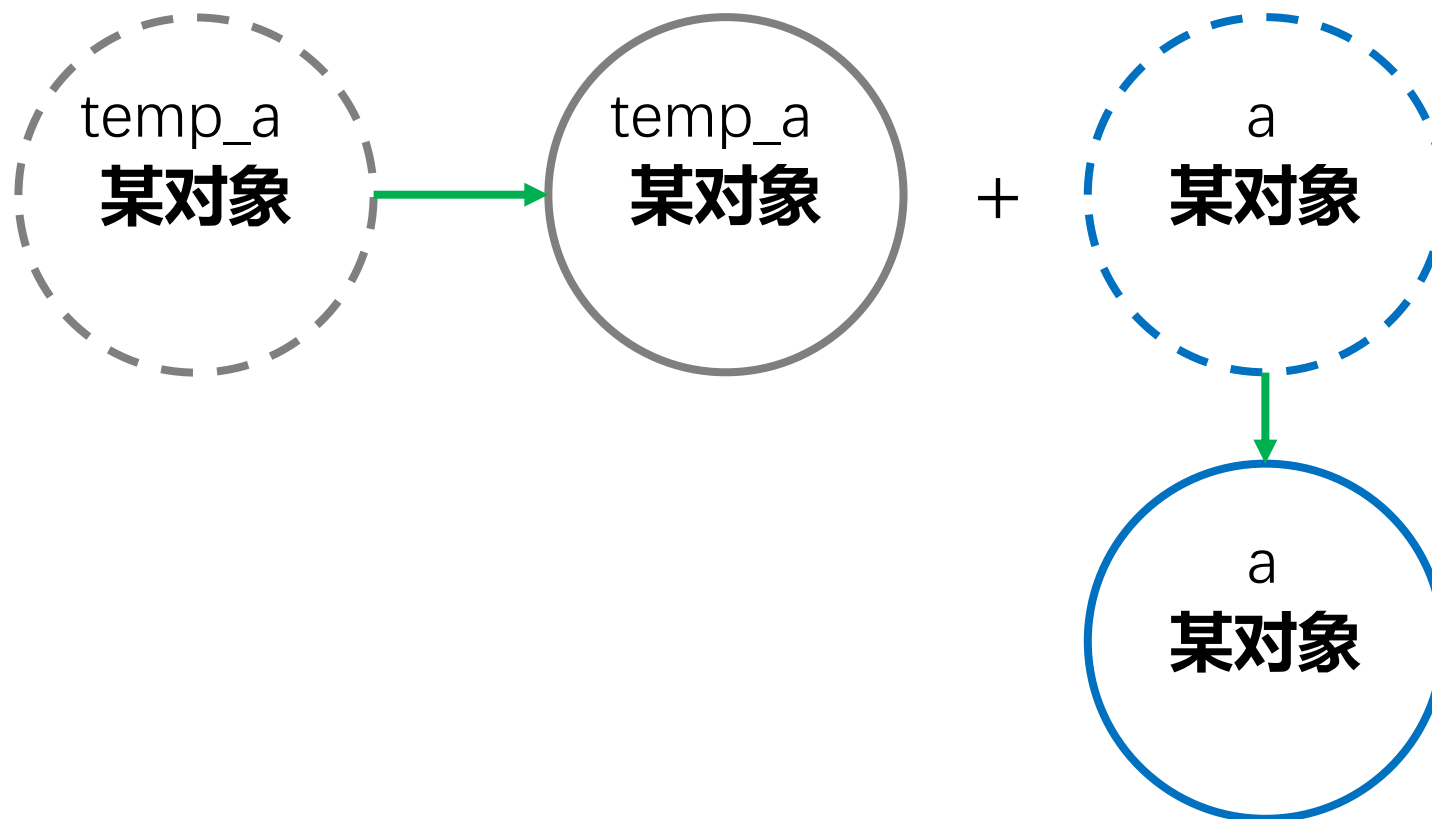
Step 8 : 销毁临时匿名变量

Step 9 : 销毁 a 对象

返回值优化 (RVO)



返回值优化 (RVO)



三、附加内容：返回值优化

1. RVO 返回值优化
2. **NRVO 命名返回值优化**
3. 优化前后的效果对比

返回值优化 (RVO)

```
People func() {  
    People temp_a("temp name");  
    return temp_a;  
}  
  
int main() {  
    People a = func();  
    return 0;  
}
```

Step 1 : 开辟 a 对象数据区

Step 2 : 调用函数 func

Step 3 : 开辟对象 temp_a 数据区

Step 4 : 调用 temp_a 对象的构造函数

Step 5 : 使用 temp_a 对象调用 a 的拷贝构造函数

Step 6 : 销毁 temp_a 对象

Step 7 : 销毁 a 对象

返回值优化 (RVO)

```
People func() {  
    People temp_a("temp name");  
    return temp_a;  
}  
  
int main() {  
    People a = func();  
    return 0;  
}
```

Step 1 : 开辟 a 对象数据区

Step 2 : 调用函数 func

Step 3 : 开辟对象 temp_a 数据区

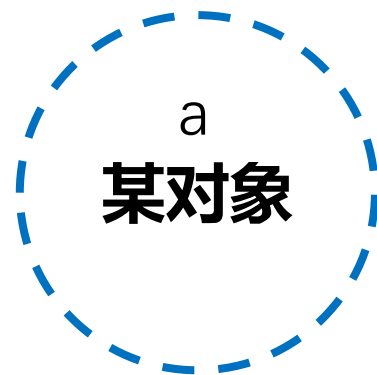
Step 4 : 调用 temp_a 对象的构造函数

Step 5 : 使用 temp_a 对象调用 a 的拷贝构造函数

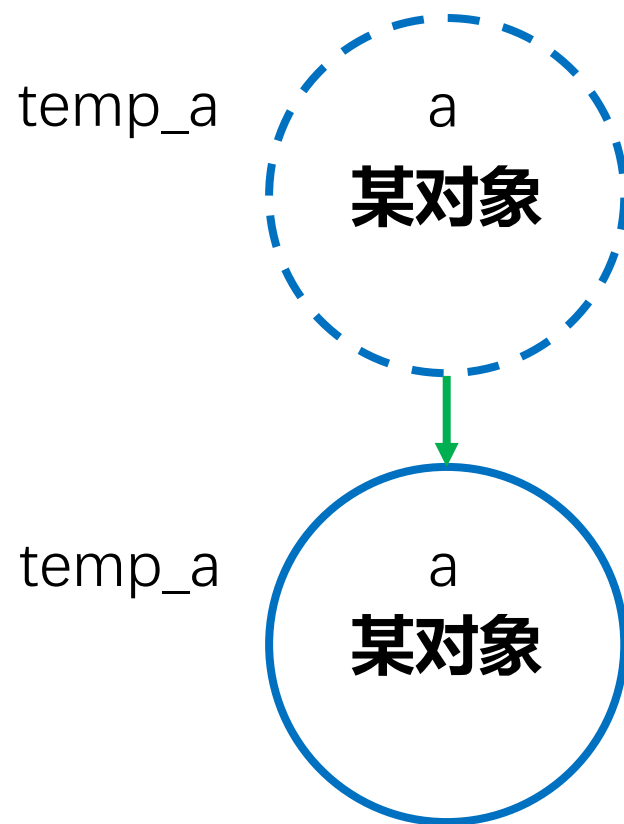
Step 6 : 销毁 temp_a 对象

Step 7 : 销毁 a 对象

命名返回值优化 (NRVO)



命名返回值优化 (NRVO)



返回值优化 (RVO)

```
People func() {  
    People temp_a("temp name");  
    return temp_a;  
}  
  
int main() {  
    People a = func();  
    return 0;  
}
```

Step 1 : 开辟 a 对象数据区

Step 2 : 调用函数 func

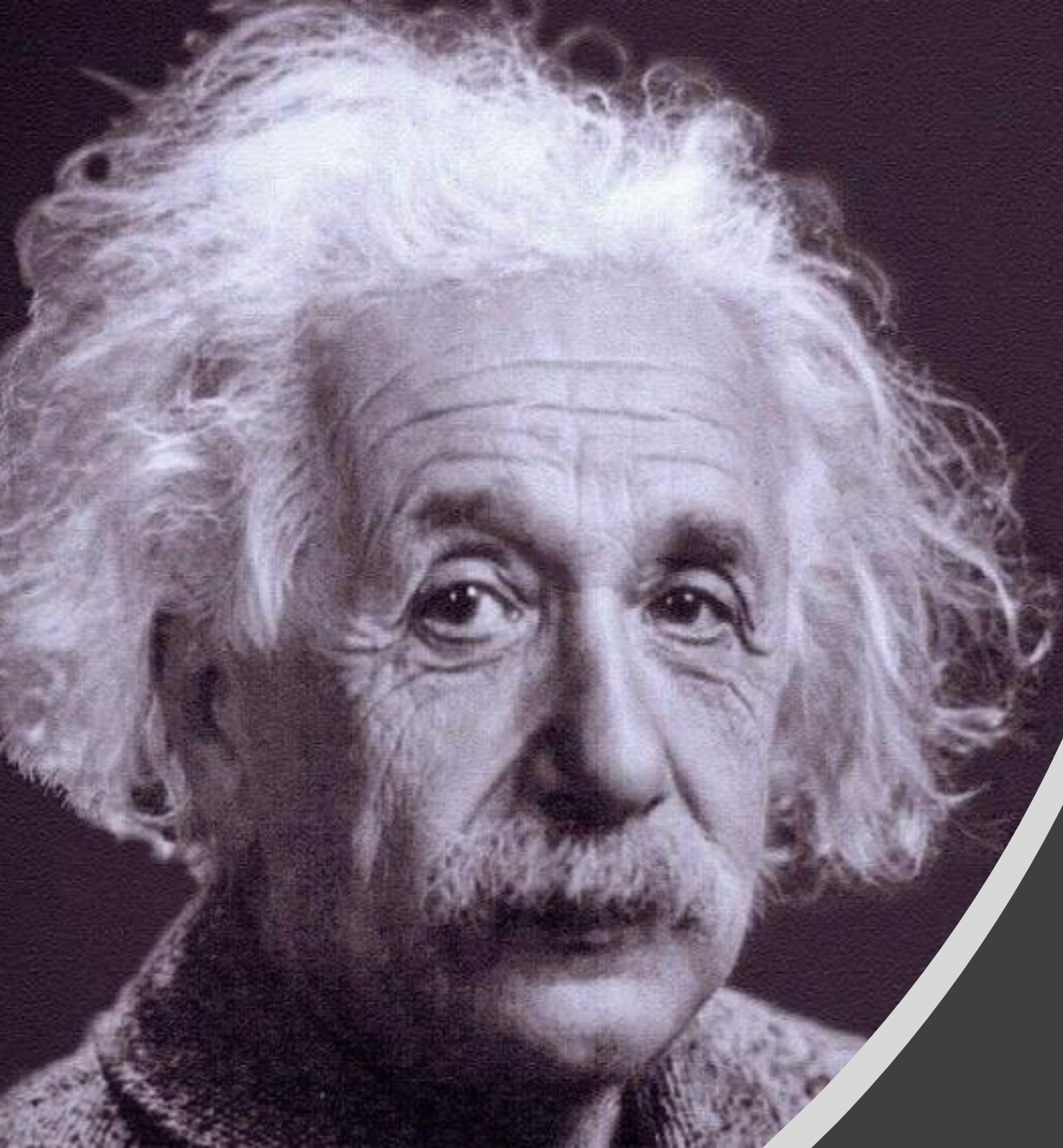
Step 3 : 将 temp_a 设置为 a 对象的替身

Step 4 : 调用 temp_a 对象的构造函数

Step 5 : 销毁 a 对象

三、附加内容：返回值优化

1. RVO 返回值优化
2. NRVO 命名返回值优化
3. 优化前后的效果对比



为什么
会出一样的题目？