

# 条款7\_1. 区别 { } 与 { } 创建对象

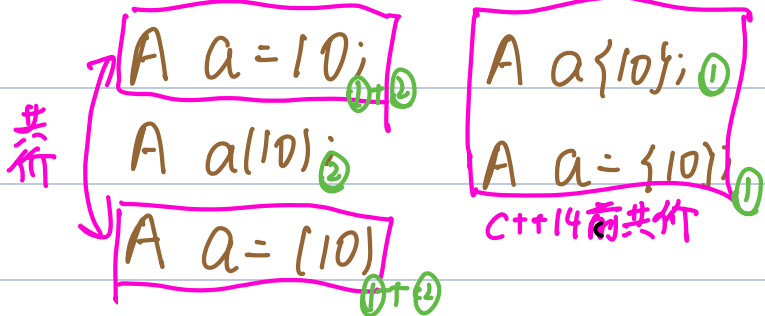
① { } 的引入

`struct A { A (int a) {} }; ①`

`A (const A&a) {} ; ②`

`struct B { B (int b, int c) {} ; ①`

`B (const B&b) {} ; ②`



1. `A a = 10;` 问题: 仅能传递1个参数 问题: 额外进行一次拷贝

2. `A a(10);` 问题: 被作用参数或返回值时仍会拷贝

3. `A a{10};` 的优势:

{ } 能完美解决上述问题, 但不允许缩窄转换 (避免精度丢失)

可使用 `static_cast<int>(x)` 强制转换

大大简化了聚合类的初始化

☆ { } 对 C++ 最令人头痛的解析问题天生免疫 (类内初始化无法使用 ( ), 被解析为函数)

② most vexing parse:

a. 对象参数的创建  $\xleftrightarrow{\text{解析错误!}}$  b. 函数声明

例1: `void f(double value) {`

`int i ( int (value) ); }` a. 新建 - [int 类型变量 (将 value 强转 为 int)]

此时编译器只会把 此时新建临时对象 b. 声明 - [函数 `int i (int value);`

i 认成函数, 而非变量 而非类型强转 C语言允许参数被综合变量包裹! ]

例2: `struct Timer{};`

a. `struct Timersave{ Timersave(Timer t){}};`

b. `Timersave time_saver(Timer t);`

a. 创建 Timersave 构造函数, 传入参数为 Timer, 创建 Timersave 对象

b. 声明了 time\_saver 函数, 拥有匿名参数, 该参数为

Timer 的指针 (由函数退化)

### ③ 聚合类定义与不同标准的区别

定义: 可通过 "聚合初始化" (列表, {} ) 直接初始化所有成员

C++14 a. 所有成员均为 public c. 没有类内初始化 C++17 取消  
b. 没有定义任何构造函数 <sup>但可以使用 default</sup> d. 无基类, 无虚函数

问题: 初始化任务交给用户

C++11 可有基类, 但必须为公有继承, 且必须是非虚继承

```
class MyString: public std::string{
```

```
public: int index = 0; }
```

```
MyString s = {"Hi", 0};
```

可省略

### ④ 奇怪的 array (C++ 中数组的更优选择)

```
int a[3]{1, 2, 3};
```

共价

```
std::array<int, 3> arr{1, 2, 3};
```

```
Point{ int arr[2]; }
```

Point p1[3]{ {1,2}, {1,4}, {5,6} }; ✓

std::array<Point, 3> p2 { {1,2}, {5,6}, {8,9} }; ✗

原因: template <typename T, size\_t n>

struct Myarr { T arr[n]; }

底层仍为原生数组

### ⑤ 如何让A支持窄式转换

C++11 §11.3.1.2 initializer\_list<> emCP84 C++P P147

注意: 1. 禁止隐式缩窄转换

2. 优先级最高 (除非不得已, 否则优先使用

initializer\_list<>, 即使会报错)

size 为 0

3. 空的 {}, 不会调用 std::ini... 构造, 但 {{}}, ({})

仍会调用 std::init...

↓  
size 为 1  
元素 为 0