C++创建对象时区分圆括号( )和大括号{ }

emin 关注他

> 内容来自Effective Modern C++ 条款7，学习总结笔记

C++11的对象初始化的语法选择是不堪和混乱的。总的来说，初始值可以借助**大括号**'{ }'，**等号**'='，**圆括号**'( )'：

```
int x(0);    // 初始值在圆括号内

int y = 0;   // 初始值在等号后面

int z{0};    // 初始值在大括号内
```

使用**等号初始化**经常会让C++初学者认为会进行一次**赋值**，但不是那样的。对于内置类型，例如 `int` ，初始化和赋值操作的**差别是模糊的。**但是对于用户定义的类，区分初始化和赋值操作是很重要的，因为这会导致**不同**的函数调用：

```
Widget w1;       // 调用默认构造函数

Widget w2 = w1;  // 不是赋值操作，调用拷贝构造函数

w1 = w2;         // 赋值操作，调用operator=函数
```

因为初始化的语法很混乱，而且有些情况无法实现，所以C++11提出了**统一初始化**语法：一种至少在概念上可以用于表达任何值的语法。它的实现基于大括号，所以我称之为**大括号初始化。**

使用大括号可以更容易的初始化容器**列表初始化**： `std::vector<int> v{1, 3, 5};`

大括号也可以用于**类内成员的默认初始值**，在C++11中，等号" ="也可以实现，但是圆括号 '( )' 则**不可以**：

```
class Widget {
  ...
private:
  int x{ 0 };   // x的默认初始值为0
  int y = 0;    // 同上
  int z( 0 );   // 报错
}
```

另一方面，**不可拷贝对象**(例如， `std::atomic`)可以用**大括号**和**圆括号**初始化，但不能用等号：

```
std::atomic<int> ai1{ 0 };  // 可以

std::atomic<int> ai2( 0 );  // 可以

std::atomic<int> ai3 = 0;   // 报错
```

注意：当大括号初始化用于内置类型的变量时，如果我们初始值存在**丢失信息的风险**，则编译器将报错：

```
doubel ld = 3.14;
int a {ld};    // 报错，存在信息丢失风险
int b (ld);    // 正确
```

大括号初始化的另一个值得注意的特性是它会免疫C++中的**最让人头痛的歧义。**当开发者想要一个**默认构造的对象**时，程序会不经意地**声明个函数**而不是**构造对象。**

```
Widget w1(10);  // 调用Widget的带参构造函数
```

但当你尝试用类似的语法调用**无参构造**时，你**声明了个函数**，而不是创建对象：

```
Widget w2();   // 最让人头痛的歧义，声明了一个名为w2，不接受任何参数，返回Widget类型的函数!
Widget w2;     // 正确：w2是个默认初始化的对象
```

使用大括号包含参数是无法声明为函数的，所以使用大括号默认构造对象不会出现这个问题：

```
Widget w2{};   // 无歧义
```

我们讲了很多大括号初始化的内容，这种语法可以用于多种场景，还可以避免隐式范围窄化转换，又免疫C++的最让人头痛的歧义问题。一举多得，那么为什么这条款不起名为“用大括号初始化语法替代其他”呢？

大括号初始化的缺点是它有时会显现**令人惊讶**的的行为。这些行为的出现是因为**与 `std::initializer_list` 混淆了。**在**构造函数中**，只要形参**不带有** `std::initializer_list` ，圆括号和大括号行为一致：

```
class Widget {
public:
  Widget(int i, bool b);
  Widget(int i, double d);
  ...
};

Widget w1(10, true);  // 调用第一个构造函数

Widget w2{10, true};  // 调用第一个构造函数

Widget w3(10, 5.0);   // 调用第二个构造函数

Widget w4{10, 5.0};   // 调用第二个构造函数
```

但是，如果**构造函数的形参带有** `std::initializer_list` ，调用构造函数时大括号初始化语法会**强制使用**带 `std::initializer_list` 参数的重载构造函数：

```
class Widget {
public:
  Widget(int i, bool b);
  Widget(int i, double d);
  Widget(std::initializer_list<long double> il);
  ...
};
Widget w1(10, true);   // 使用圆括号，调用第一个构造函数

Widget w2{10, true};   // 使用大括号，强制调用第三个构造函数，10和true被转换为long double

Widget w3(10, 5.0);    // 使用圆括号，调用第二个构造函数

Widget w4{10, 5.0};    // 使用大括号，强制调用第三个构造函数，10和5.0被转换为long double
```

就算是正常的**拷贝构造**和**赋值构造**也可以被带有 `std::initializer_list` 的构造函数**劫持**：

```
class Widget {
public:
  Widget(int i, bool b);
  Widget(int i, double d);
  Widget(std::initializer_list<long double> il);
  operator float() const;   // 支持隐式转换为float类型
  ...
};

Widget w5(w4);    // 使用圆括号，调用拷贝构造函数

Widget w6{w4};    // 使用大括号，调用第三个构造函数
                  // 原因是先把w4转换为float，再把float转换为long dobule

Widget w7(std::move(m4));  // 使用圆括号，调用移动构造函数

Widget w8{std::move(m4)};  // 使用大括号，调用第三个构造函数，理由同w6
```

编译器用带有 `std::initializer_list` 构造函数匹配大括号初始值的决心是如此的**坚定**，即使带有 `std::initializer_list` 的构造函数是无法调用的，编译器也会**忽略**另外两个构造函数(第二个还是**参数精确匹配**的)：

```
class Widget {
public:
  Widget(int i, bool b);
  Widget(int i, double d);
  Widget(std::initializer_list<bool> il);  // long double 改为 bool
  ...
};

Widget w{10, 5.0};  // 报错，因为发生范围窄化转换
                    // 编译器会忽略另外两个构造函数(第二个还是参数精确匹配的！)
```

只有当大括号内的值**无法转换**为 `std::initializer_list` 元素的类型时，编译器才会使用正常的重载选择方法：

```
class Widget {
public:
  Widget(int i, bool b);
  Widget(int i, double d);
  Widget(std::initializer_list<std::string> il);  // bool 改为 std::string
  ...
};

Widget w1(10, true);  // 使用圆括号，调用第一个构造函数

Widget w2{10, true};  // 使用大括号，不过调用第一个构造函数，因为无法转换为string

Widget w3(10, 5.0);   // 使用圆括号，调用第二个构造函数

Widget w4{10, 5.0};   // 使用大括号，  不过调用第二个构造函数，因为无法转换为string
```

不过这里有一个有趣的边缘情况。一个大括号内**无参**的构造函数，不仅可以表示默认构造，还可以表示带 `std::initializer_list` 的构造函数。你的空括号是表示哪一种情况呢？

正确答案是你将使用**默认构造**，一个空的大括号表示的是没有参数，而不是一个空的 `std::initializer_list` ：

```
class Widget {
public:
  Widget();
  Widget(std::initializer_list<int> il);
  ...
};

Widget w1;     // 调用默认构造函数

Widget w2{};   // 调用默认构造函数
```

如果你想要用一个空的 `std::initializer_list` 参数来调用带 `std::initializer_list` 构造函数，那么你需要**把大括号作为参数**，即把空的大括号放在圆括号内或者大括号内：

```
Widget w4({});   // 用了一个空的list来调用带std::initializer_list构造函数
```

此时此刻，大括号初始化， `std::initializer_list` ，构造函数重载之间的复杂关系在你的大脑中冒泡，你可能想要知道这些信息会在多大程度上关系到你的日常编程。可能比你想象中要多，因为 `std::vector` 就是一个**被它们直接影响的类。** `std::vector` 中有一个可以**指定**容器的**大小**和容器内**元素的初始值**的不带 `std::initializer_list` 构造函数，但它也有一个可以指定容器中元素值的带 **`std::initializer_list` 函数。**

```
std::vector<int> v1(10, 20);   // 使用不带std::initializer_list的构造函数
                               // 创建10个元素的vector，每个元素的初始值为20

std::vector<int> v2{10, 20};   // 使用带std::initializer_list的构造函数
                               // 创建2个元素的vector，元素值为10和20
```

编辑于 2020-11-18 20:57

C++ 编程 C++

写下你的评论...

NightSky 2020-11-14

无参结构结果声明了个函数这段迷之好笑哈哈哈哈

8

善俗 2020-11-18

C++ 简直太牛逼了！！！ 哈哈哈

3

tch0 2020-11-09

简单来说，小括号有窄化转换，也不能使用小括号无参构造函数。大括号又有优先匹配初值列表版本的特性。
我的习惯是通常都用小括号(清晰的表明我在调用构造函数），定义了初始化列表版本才用大括号。

3

我爱中华 03-13

哈哈，一个初始化就整这么多。这还没将默认初始化，指定初始化 哈哈哈c++

赞

灵格余以嗟来耶 02-19

我去，我一直以为()就是初始化列表

赞

神奇的22号 2021-12-13

牛皮

赞

梦霸 2021-05-22

好家伙，看懵了

赞

AlexFan 2020-11-10

回字有四种写法😂

赞