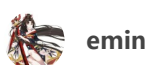


C++ 创建对象时区分圆括号 () 和大括号 { }



emin

+ 关注他

69 人赞同了该文章

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退和混乱的。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型，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> a1{ 0 }; // 可以

std::atomic<int> a2{ 0 }; // 可以

std::atomic<int> a3 = 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**丢失信号的风险**，则编译器将报错：

```
double ld = 3.14;
int a {14}; // 报错，存在信息丢失风险
int b {14}; // 正确
```

大括号初始化的另一个值得注意的特性是它会避免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**无参构造函数**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，又避免C++的最让人头痛的歧义问题。一语多得，那么为什么这条款不赶名为“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**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

<  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
```

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，再让float转换为long double

Widget w7(std::move(w4)); // 使用圆括号，调用移动构造函数

Widget w8(std::move(w4)); // 使用大括号，调用第三个构造函数，理由同w6
```

编译器调用带有 std::initializer_list 构造函数匹配大括号初始值的决心是知道的**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<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
```

此刻此刻，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嗡嗡，你可能想知道这些信息会在多大程度上关系到你的日常编程，可能比你想象中更多，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**指定容器的尺寸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20和20
```

编辑于 2020-11-18 20:57

[C++ 编程](#) [C++](#)

8 条评论

[🔼 切换为时间顺序](#)

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

Ich0

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。大括号又有优先级和初始化列表逆本的特性。
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)，定义了初始化列表原本才用大括号。

3

我爱中华

03-13

哈哈，一个初始化就整这么多，这还没得默认为初始化，指定初始化 哈哈哈c++

赞

灵格余以堪来耶

02-19

我去，我一直以为0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，看错了

赞

AlexFan

2020-11-10

回字有四种写法

赞