

翻书

博客园 首页 新随笔 联系 订阅 管理

C++11 带来的新特性 （2）—— 统一初始化（Uniform Initialization）

1 统一初始化（Uniform Initialization）

在C++ 11之前，所有对象的初始化方式是不同的，经常让写代码的我们感到困惑。C++ 11努力创建一个统一的初始化方式。其语法是使用{}和std::initializer_list，先看示例。

```
int values[] { 1, 2, 3 };
std::vector<int> v { 2, 3, 6, 7 };
std::vector<std::string> cities {
    "Berlin", "New York", "London", "Braunschweig"
};
std::complex<double> c {4.0, 3.0}; //等价于 c(4.0, 3.0)
auto ar = { 1, 2, 3 }; // ar 是一个std::initializer_list<int>类型

std::vector<int> v = { 1, 2, 3 };
std::list<int> l = { 1, 2, 3 };
std::set<int> s = { 1, 2, 3 };
std::map<int, std::string> m = { {1, "a"}, {2, "b"} };
```

2 原理

针对形如"{ 1, 2, 3 }"的参数列表，系统会首先自动调用参数初始化(value initialization)，将其转换成一个std::initializer_list，它将用于对变量（可能是简单类型，或者对象类型）初始化。比如"{ 1, 2, 3 }"会首先生成一个std::initializer_list，然后用于生成一个int数组values。c{4.0, 3.0}会生成一个std::initializer_list,然后调用std::complex类的构造函数std::complex(double,double)。

我们通过一个例子来分析具体细节：

```
std::vector<int> v { 2, 3, 6, 7 };
```

- 首先，将参数列表{ 2, 3, 6, 7 }转换成std::initializer_list。从stl源码中可以看出initializer_list的带参数构造函数是个私有函数，它只能由编译器调用。

```
private:
    iterator      _M_array;
    size_type     _M_len;

    // The compiler can call a private constructor.
    constexpr initializer_list(const_iterator __a, size_type __l)
        : _M_array(__a), _M_len(__l) { }
```

- 其次，使用std::initializer_list对象来初始化std::vector类的构造函数。下面是构造函数源码。

```
vector(initializer_list<value_type> __l,
        const allocator_type& __a = allocator_type())
    : _Base(__a)
    {
        _M_range_initialize(__l.begin(), __l.end(),
                            random_access_iterator_tag());
    }
```

3 未赋值的初始化

如果使用了std::initializer_list，但是没有指定参数值，结果会怎样？直接看示例。

```
int i; //i值未定义
int j{}; //j=0
int *p; //p值未定义
int *q{}; //q=nullptr
```

4 在构造函数中显示使用std::initializer_list

我们可以在构造函数中主动使用std::initializer_list，这时外部调用{}初始化时，会优先调用包含std::initializer_list参数的构造函数。请看下例子。

```
class P
{
public:
    P(int, int){std::cout<<"call P:P(int,int)"<<std::endl;}
    P(std::initializer_list<int>){
        std::cout<<"call P:P(initializer_list)"<<std::endl;
    }
};

P p(77,5); // call P:P(int,int)
P q{77,5}; // call P:P(initializer_list)
P r{77,5,42}; // call P:P(initializer_list)
P s = {77, 5}; // call P:P(initializer_list)
```

5 拒绝隐式调用构造函数

我们知道，C++会先使用{}中的参数生成一个std::initializer_list对象，然后调用赋值对象的构造函数。有一种特殊情形，我们如果希望对象的某个构造函数必须要被显示调用，如何做到呢？向其中添加一个explicit关键字。在stl库中出现大量的这种使用方式。请看下例：

```
class P
{
public:
    P(int a, int b){...}
    explicit P(int a, int b, int c){...}
};

P x(77,5); //OK
P y{77,5}; //OK
P z{77,5,42}; //OK
P v = {77,5}; //OK, (implicit type conversion allowed)
P w = {77,5,42}; //Error,必须要显示调用该构造函数

void fp(const P&);

fp({47,11}); //OK
fp({47,11,3}); //Error
fp(P{47,11}); //OK
fp(P{47,11,3}); //OK
```

6 局限 —— Narrowing Initializations

统一初始化用起来很舒爽，那它有什么局限呢？有，在一种场景下无法使用，那就是Narrowing Initializations。Narrowing Initializations，我翻译为“精度截断”。比如float转换为int，double转换为float。统一初始化，完全不允许精度阶段的发生，更进一步，要求参数列表中的所有参数的精度一样。请看以下示例。

```
int x1{5.3}; //OK, x1 = 5.3
int x3{5.0}; //Error
int x4 = {5.3}; //Error
char c1{7}; //OK
char c2{99999}; //Error
std::vector<int> v1{ 1, 2, 4, 5}; //OK
std::vector<int> v2{ 1, 2,3, 4, 5.6}; //Error
```

但是如果实际工程允许精度截断的发生，那么我们应该怎么完成初始化。可以使用()来完成初始化，它会调用赋值操作或者相应的构造函数。

```
int x3{5.0}; //Error
int x2{5.2}; //OK, x2 = 5
```

分类: C && C++

好文要顶 关注我 收藏该文

翻书

关注 - 3
粉丝 - 26

推荐博客

+加关注

« 上一篇: C++11 带来的新特性 (1)
» 下一篇: C++11 带来的新特性 (3) —— 关键字noexcept

1 推荐

0 反对

随笔 - 55 文章 - 0 评论 - 28 阅读 - 18万

公告

昵称： 翻书
园龄： 12年9个月
荣誉： 推荐博客
粉丝： 26
关注： 3
+加关注

盖楼抽奖

#她的梦想在发光#
HWD科技女性故事有奖征集

分享最打动你的科技女性故事

活动时间: 2022年3月8日-3月18日

马上参与

<	2022年3月						>
日	一	二	三	四	五	六	
27	28	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	1	2	
3	4	5	6	7	8	9	

搜索



常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

C && C++(7)
Live without IDE(3)
Python(3)
silverlight(1)
百科(7)
编译原理(10)
操作系统(3)
繁枝细节(3)
区块链(6)
软件人生(3)
系统工具(4)
写作工具(4)

随笔档案

2019年10月(1)
2019年5月(1)
2019年3月(1)
2018年12月(2)
2018年11月(7)
2018年10月(3)
2018年9月(1)
2018年8月(1)
2018年7月(13)
2018年4月(1)
2013年12月(5)
2013年9月(4)
2011年3月(1)
2011年2月(1)
2010年12月(3)
更多

阅读排行榜

1. C++11 带来的新特性 (3) —— 关...
2. 儿子和女儿——解释器和编译器的区...
3. Telnet和SSH(11260)
4. 语法分析器自动生成工具一览(8641)
5. 在Ubuntu18.04下安装Java 11(763...

评论排行榜

1. 一只小小麻雀——基于语法分析工具G...
2. 儿子和女儿——解释器和编译器的区...
3. C++11 带来的新特性 (3) —— 关...
4. 安装版本管理软件-SVN(2)
5. Telnet和SSH(1)

推荐排行榜

1. 儿子和女儿——解释器和编译器的区...
2. C++11 带来的新特性 (3) —— 关...
3. 一只小小麻雀——基于语法分析工具G...
4. Telnet和SSH(2)
5. C++11 带来的新特性 (2) —— 统...

最新评论

1. Re:Telnet和SSH
“虽然前者比较古老，但是后者用了不太长的时间就占据了主流。”——这句话怎么感觉语法有点怪性的
--sfljk

2. Re:儿子和女儿——解释器和编译器的...
看的明明白白，还有其他文章吗，想看
--Captain_Li

3. Re:C++11 带来的新特性 (3) ——...
noexcept is particularly valuable for the move operations, swap, memory deallocation functions, and ...
--kite97

4. Re:C++11 带来的新特性 (3) ——...
最近看了一下关于异常这边，effective modern C++中的意思是，noexcept还更倾向于一种接口上的设计保证，一个函数如果不抛异常，那么就是noexcept的。我觉得考虑清楚的话，应该...
--薛定谔的三味

5. Re:C++11 带来的新特性 (3) ——...
@ JoeChenzzz 移动分配函数，英文名move assignment，又称为移动赋值函数（微软官方翻译），它是c++11之后才引入的新特性，与早期版本中的复制赋值函数对应。以 a = f() ...
--翻书

编辑推荐：
· 平时的的工作如何体现一个人的技术深度？
· 革命性创新，动画杀手铜 @scroll-timeline
· 戏说领域驱动设计（十二） —— 服务
· ASP.NET Core 6框架揭秘实例演示[16]：内存缓存与分布式缓存的使用
· .Net Core 中无处不在的 Async/Await 是如何提升性能的？

#她的梦想在发光#
HWD科技女性故事有奖征集

活动时间: 2022年3月8日-3月18日

最新新闻：
· 日本CG大神又整活了！3D建模软件拿来搞面部实时捕捉
· 科学家利用模型描绘第二个“地球”可能的样子
· 人造神经元成功模拟植物，让捕蝇草强行闭合！Nature子刊
· 京东“吃下”德邦
· 独立添加好友功能终于来了！Windows端微信发布3.6内测版
» 更多新闻...