### 记录时间2019.11.18-2019.11.25（一周）

**开发**： **BUG**： **分析**： **辨析研究**： **重点标记**：

2019.11.18

insert、splice、merge三个函数之间的关系

splice、merge函数都会改变源链表的结构。 splice直接剥夺，merge则是来源链表和自己本身进行重新排序构造。

insert会新建一个结点，对来源链表不会造成任何改动。

基于splice、merge都会改变来源链表，所以可以提供一个辅助函数实现”迁移操作”。

void transfer(iterator position, iterator first, iterator last)

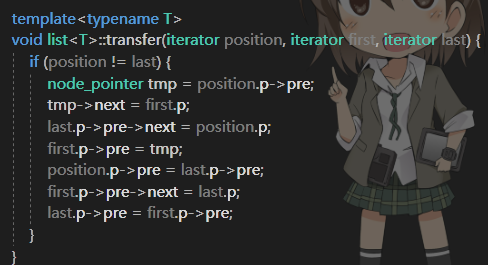
要不要在这个函数里面考虑边界问题

position 链表头、尾部、中间

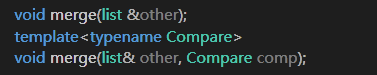
[first, last)是区间位置.

太复杂了吧。。

**！！！首先这个函数是我自己调用的，我自己肯定会提前避开各种边界条件吧！！！**



merge函数思路：



merge也就两种方式，一是升序，二是降序，默认是升序。

我居然妄想一段代码只因为一个判断的不同就能实现即可升序排列又可降序排列..实际情况是，当你需要升序排列时，你的链表得是升序的；当你需要降序排列时，你的链表得是降序排列的！

merge就是合并两个链表，思路还是很简单的。

remove函数：

使用到了lambda表达式。

2019.11.19

1迭代器相关问题

**从iterator 到const iterator 无法直接转换。**

vector的迭代器之所以能够直接强制转换，是因为vector迭代器是指针，指针是可以直接转换的，但是list的迭代器是一个类对象，所以需要进行特殊处理。

**区分：顶层和底层const的概念**

顶层const 表示 指针本身似乎个常量 底层const 表示 指针所指对象是一个常量

顶层const可以表示任意的对象是常量，这一点对任何数据类型都适用，如类、指针等

底层const则与指针、引用等符合类型的基本类型部分有关。

用于声明引用的都是底层const

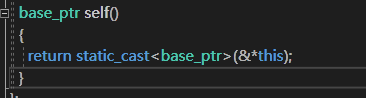
整个设计很混乱，const、非const这种设定，乱起八糟的！！！ 重新捋一捋设计。

2 node\_pointer 和iterator 使用没有定好规则，使用起来太随意了。

3 之前的设计是采用的双向链表，加个空白结点！ 原来是为了满足 **前闭后开** 的规范。

参照MyTinySTL的设计吧~，他把数据域和指针域分成了两个部分。list\_node\_base 和list\_node两个类，list\_node 继承list\_node\_base，然后增加数据域。

整个链表创建过程中，都是在处理指针关系，没看到数据域的使用诶！



把当前结点的地址转换成base\_ptr 然后赋值给prev、next

设计思路是这样：

一个首元结点只有prev、next指针，该结点用来满足前闭后开的条件，同时，方便插入结点时直接挂载在这个首元结点上，不像之前的双向链表，还什么需要先建立一个结点然后又给它赋值，最后又得新建一个结点作为结尾。

4 开始重构吧

重构之前，先搞明白“存在继承关系的类型之间关于指针转换的问题”。这个问题可以写一篇博客了。

2019.11.20

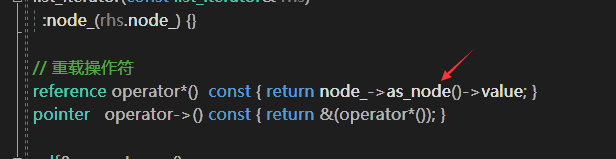
1

想明白了关于基类和派生类指针转换的问题。 主要一点还是在于，如果基类指针指向了派生类对象，那么将基类指针转换成派生类指针是可行的，会激活改指针所有功能，能够访问派生类对象的所有成员和方法。

list容器的标志结点成员 node\_ 主要就是为了满足 [前闭，后开)的原则，例如begin可以是 返回 node\_->next; end返回node\_ 本身就行。

因为这个前闭后开的原则，我们使用end()获取迭代器后，对其进行解引用操作肯定是不合法的，不能通过编译的。

但是this对象问题



as\_node涉及到this对象，是node\_ 这个类型的对象，也就是list\_node\_base的对象

**辨析研究1**：设计成list\_node 和list\_node\_base 这样有什么好处呢？ 优势、便利性在哪？

迭代器的成员变量是list\_node\_base 指针，基类是可以指向派生类的，同时还可以从基类指针转换成派生类指针。

也没有特别突出的便利性，只是好像可以让我们专心处理指针链接的问题，关于值域不用操心太多。

2

了解一下snippet语法格式，方便写自己的代码片段

3

重构了结点和迭代器

2019.11.21

思考设计问题，如果让我自己设计list容器的内部构造，我应该从哪下手？

①首先明确一点，这些数据结构都是之前的大佬想出来的，我现在只要知道这个结构是什么样子的就行了。比如list就是个双向链表咩~，然后想想双向链表怎么插入、删除的？

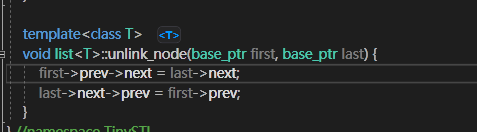
②通过代码尝试容器的每一个成员函数的作用，自己思考大概的实现方式

③参考其他代码

STL源码剖析、GCC版本实现

捋一捋list整个框架 在草稿本上的一些图

编写helper function



函数只需要做好自己应该做的事，边界判断条件不应该是函数所要考虑的，不能说我函数还需要自己判断first是不是在头部或是尾部等。这种工作应该由更高层的函数去考虑。

2019.11.22

**BUG1**：list<T> test(10,1)这种情况是就会跑出来。**☑**

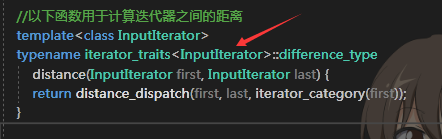


**分析**：

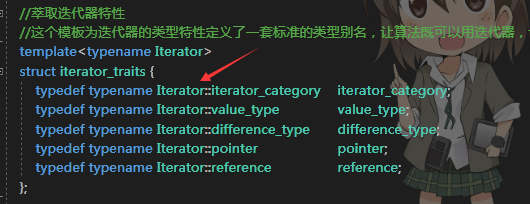
list<T> test(10,1)这种情况调依然是如下函数



这个构造函数内部又是调用copy\_init的，copy又使用了TinySTL::distance函数

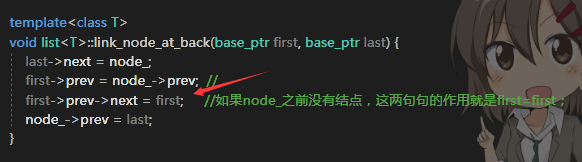


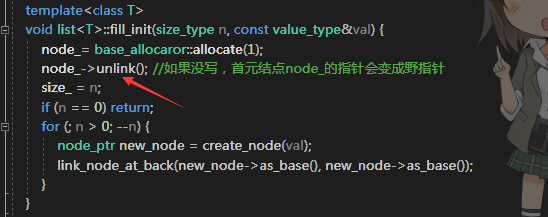
可以看到，distance对InputIterator进行了一次迭代器萃取！但是我们的本意这两个参数并不是迭代器！所以会导致如下发生错误！



解决办法还是之前一样，加一个判断 \_\_is\_integer

**BUG 2**：如果不初始化首元结点node\_的指针，会导致link系列函数出问题...**☑**

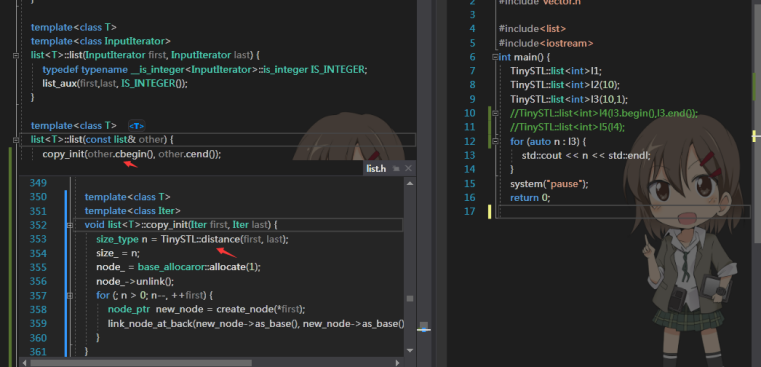




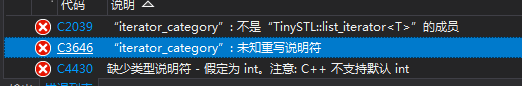
2019.11.23

**开发**：析构函数

**BUG1**：由于distance函数导致copy\_init函数出错，无法构建对象**☑**



**分析**：

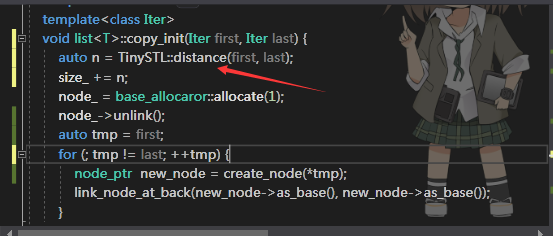


最开始我推测是begin()、end()这些函数出了问题，当时我以为这些函数返回的压根不是一个迭代器，从而导致了distance里面萃取出问题。

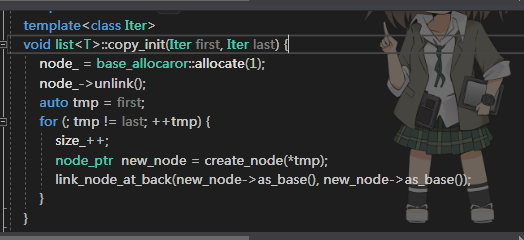
但是经过一番学习，和对begin()、end()的测试发现，这两个函数不存在问题。

|  |
| --- |
| 首先看函数返回类型和return语句：  “**当函数返回类型不是void时，则return语句必须返回一个值。返回值要么与函数的返回类型一样，要么返回值能够进行隐式转换成函数的返回类型**。”  iterator begin() {....} iterator end() {....}  返回类型是iterator或是const\_iterator。iterator和const\_iterator类都有成员变量 base\_ptr node\_；容器list也有一个base\_ptr node\_;  可以用list容器的node\_ 来初始化一个iterator的对象！也就是iterator需要有copy构造函数。  如iterator end() { return node\_;} node\_是base\_ptr类型，肯定不满足函数返回类型的，所以需要转换。 |

然后抱着尝试的心态，去掉了distance函数，发现copy\_init就能正常运行了，所以我暂时确认为distance函数有问题。



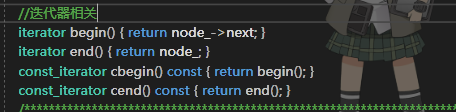
经过如下修改改能够进行正常的对象构造了



为了确认是不是distance的问题，我又计算vector的迭代器之间距离，又发现行得通。

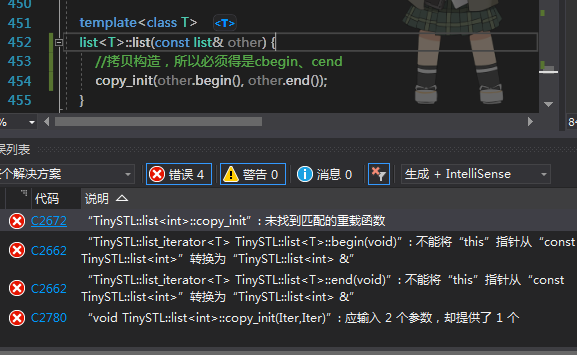
**BUG2**：迭代器问题（说到底还是const和非const问题）**🗵**

1





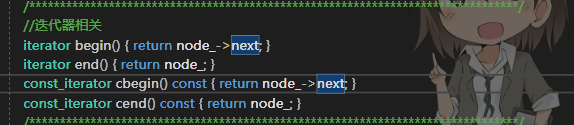
2



**分析**：

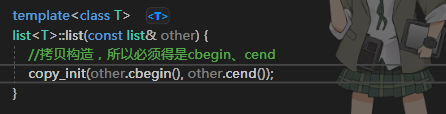
1

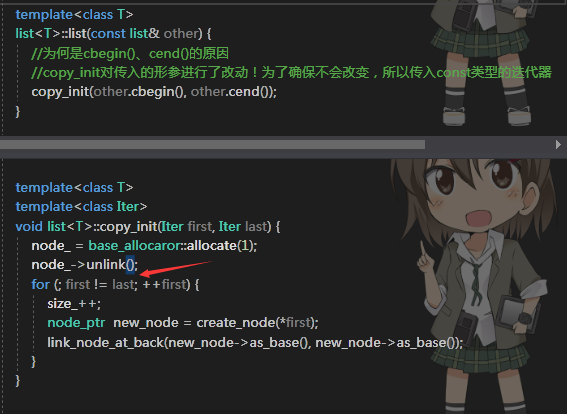
这样改动就行得通

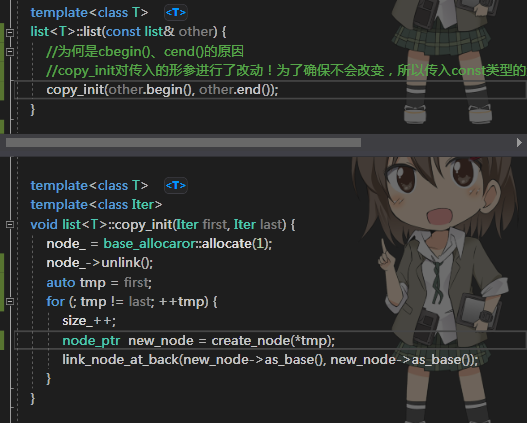


问题应该是函数返回类型是类类型，然后函数又返回了一个对象

2







前提原因是因为other是const类型（确保传入对象不会被改动）

如果copy\_init没有对参数进行改动，则传入begin()、end()即可，如果copy\_init对参数进行了修改，则参数必须是cbegin()、cend()

**这个改动主要是赋值操作！ 对迭代器进行移动什么的无关紧要其实**。

2019.11.24

**知识点**：





首先，data\_allocator具体类型的一个别名，之所以可以使用node\_allocator调用函数！是因为allocator类里面的函数是static函数！ static函数可以直接通过“类名：：xxx”这种形式访问。

**辨析研究**：list容器框架总结

list容器的一个具体对象的成员只有两个：①结点指针node\_ ②当前结点个数size。

要获取容器的迭代器，则只有调用begin()、end()等相关函数，然后以node\_为值，进行迭代器对象的构造！

要创建结点，通过一系列的辅助成员函数进行结点创建，然后再连接到一个首元结点上！同时，指针成员node\_还指向这个首元结点。

一个具体容器可以分成是三部分来看：

1. **迭代器部分**

因为迭代器最后还是指向容器的元素的，所以迭代器肯定有一个指针型的成员变量，通过构造一个迭代器对象，这个指针的指向和容器的指针成员指向同一个地方。

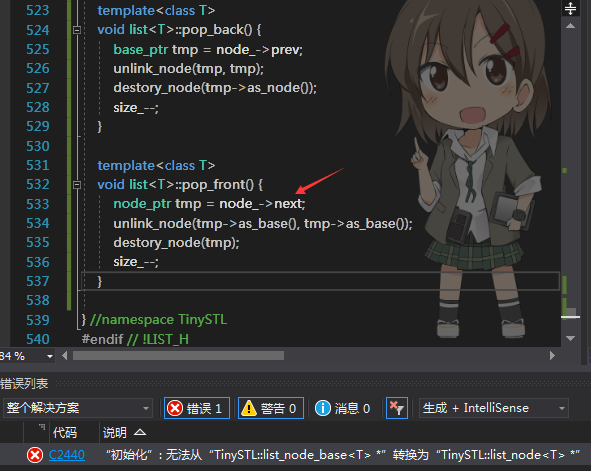
1. **基础数据结构部分**

进行数据存储的结构~

1. **成员变量以及相关成员函数部分**

成员变量肯定有一个指向容器元素的指针，这个指针用来被初始化迭代器对象的！

**BUG1**:pop\_front、pop\_back问题☑



**分析**：

push\_back、push\_front时，我们创建新的结点，自己可以保证一定是node\_ptr类型的。node\_->next 或prev都是base\_ptr类型的...

我们创建结点时按照node\_ptr类型创建，但是连接不是呀！！连接时只用到了prev、next两个指针，所以把结点转换成list\_node\_base类型。

**同时还需要注意tmp->unlink() 操作！**

**BUG2:**resize()出了点小问题**☑**

没注意resize()调用pop\_back其实也会进行结点数目减一。

**开发**：

①为每个函数加上注释，参数、作用、返回值等，解释清楚。

2019.11.25

**BUG1**: 当链表为空时，使用insert插入，都会缺少一个元素！**☑**

insert的底层实现就是调用这两个函数

|  |
| --- |
| template<class T>  typename list<T>::iterator list<T>::fill\_insert(const\_iterator pos, size\_type n, const value\_type& val) {  size\_ += n;  iterator r;  for (; n > 0; n--) {  node\_ptr new\_node = create\_node(val);  r=link\_one\_node(pos, new\_node->as\_base());  }  return r;  } |
| template<class T>  template<class Iter>  typename list<T>::iterator  list<T>::copy\_insert(const\_iterator pos, size\_type n, Iter first) {  size\_type add\_size = n;  size\_ += add\_size;  iterator r;  for (; n > 0; n--,++first) {  node\_ptr new\_node = create\_node(\*first);  r=link\_one\_node(pos, new\_node->as\_base());  }  return r;  } |

**分析**：

经过一番调试发现问题出在 link\_one\_node内部调用的link\_node\_at\_front函数中！

且看下图分析：

|  |
| --- |
| fullsizerender(2) |

进行连接的时候必须考虑假设链表有结点，如果没有，虚线部分可以直接贯穿的。