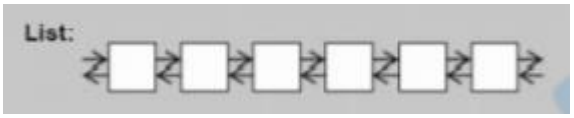
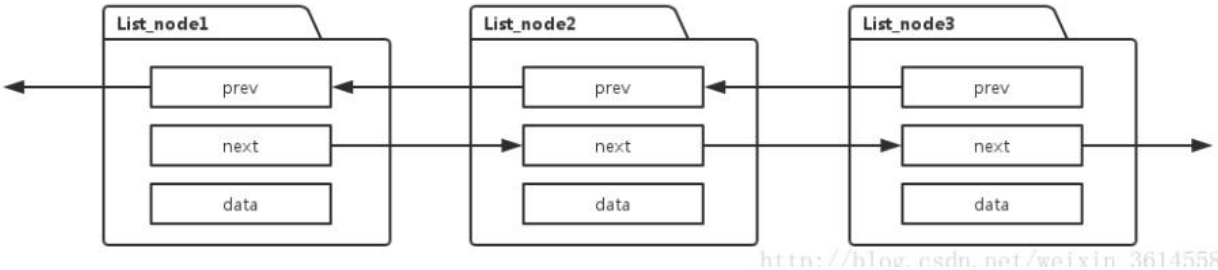
https://blog.csdn.net/weixin_36145588/article/details/76576700

list：序列容器，内存是不连续的，任意元素的访问、修改时间复杂度是O(n)，插入、删除操作是常数时间复杂度，可以在任何位置插入新元素；不支持随机访问、比vector占用更多的存储空间；内部实现采用双向链表，如下图；





List有一个重要的性质就是插入和接合操作都不会造成原有的List迭代器失效。而且，再删除一个节点时，也仅有指向被删除元素的那个迭代器失效，其他迭代器不受任何影响；

原因猜测：list的迭代器，底层就是指针，每个元素都是单独的内存节点，插入和接合都不会影响到其他元素的内存，所以迭代器不失效；删除的时候，内存释放，所以指向该元素的迭代器失效，其他元素的迭代器不受影响；

特性：

有序：序列容器中的元素以严格的线性顺序排列。单个元素按其顺序通过其位置访问；

双向链表：每个元素都保存了关于如何定位下一个和前一个元素的信息，允许在特定元素之前或之后执行插入和删除操作，但不允许直接通过位置访问。

自动存储：容器使用allocator对象动态处理其存储需求；

头文件：

```
#include<list>
```

基本接口：

```
list.size();              //返回容器中，元素的数量
list.max_size(); //容器能够存储的元素最大数量，受限于内存
```

//调整list容器中的元素个数到n个；如果n比容器中现有元素数量少，容器中只保留前n个元素，其他元素全部弹出list，并销毁，list容器占用的内存大小一起缩小调整；如果n比容器中现有元素数量多，则在容器尾部扩展元素直到满足n个元素的要求，扩展容器占用内存大小；

```
list.resize(size_type n, value_type val = value_type());
```

//返回的均为引用，可以通过引用修改该元素的值

```
list.front();                    //返回容器中第一个元素的引用;
list.back();                     //返回容器中最后一个元素的引用;

//可以通过迭代器修改指向元素的值：*it = xxx;
list.begin();      //返回指向第一个元素的迭代器
list.end();        //返回指向最后一个元素下一个位置的迭代器

list.push_back(const value_type& val);          //插入元素到队列尾部
list.push_front(const value_type& val);         //插入元素到队首
list.pop_back();                                //将队列尾部的元素移出队列
list.pop_front();                               //将队列首部的元素移出队列

//插入元素到指定位置的前面；返回值为迭代器，该迭代器指向新插入的第一个元素;
list.insert(iterator position, const value_type& val);
list.insert(iterator position, size_type n, const value_type& val);
list.insert(iterator position, InputIterator first, InputIterator last);

//删除容器中元素，返回值为一个迭代器，该迭代器指向最后一个删除元素(position)后面的一个元素的位置；所以可以通过erase的返回值安全的迭代删除所有元素;
list.erase(iterator position);
list.erase(iterator first, iterator last);

//use iterator to delete all element safely
for (list<int>::iterator iter =list.begin(); iter != list.end();){
      iter = list.erase(iter);
}

//删除容器中得所有元素，释放容器中对应元素占用的内存
list.clear();

//将list x中得元素转移到当前容器中，转移的元素不会再出现在x中
list.splice(iterator position, list& x);
list.splice(iterator position, list& x, iterator i);
list.splice(iterator position, list& x, iterator first, iterator last);

//删除链表中元素值为val的全部元素
list.remove(const value_type& val);
```

```
//删除链表中相同的元素，相同元素只留下第一个出现该值的元素；
list.unique();

//将两个有序链表合并，合并后的链表也是有序的
list.merge();

//排序，默认按升序排列
list.sort();

//反转链表
list.reverse();
```