openFOAM学习笔记（四）—— openFOAM中的List

原创 ycwang125 2020-06-27 23:13:55 291 收藏 1 版权

分类专栏： openFOAM 文章标签： C++ openfoam

又是一个很底层的部分，但是也非常重要，我们在进行数据写入的时候就会使用到List。这里介绍他的基本结构，以及在openFOAM被如何使用

首先它的路径为 `src/OpenFOAM/containers/Lists/`

这里主要介绍 `UList` `List` 两个类，其中 `List` 为 `UList` 的子类

UList

首先我们看下代码中的注释：

```
Description
    A 1D vector of objects of type \<T\>, where the size of the vector is
    known and can be used for subscript bounds checking, etc.

    Storage is not allocated during construction or use but is supplied to
    the constructor as an argument.  This type of list is particularly useful
    for lists that refer to parts of existing lists such as SubList.

他是一个T类型的一维向量，向量的size已知，并且size可以被用来在下标边界检查
在构造期间并没有allocate新的内存，而是将内存作为构造函数的输入参数给定
```

它定义了最基础的类 `UList` 。他只有两个 `private` 的成员变量

```
template<class T>
class UList
{
//- Number of elements in UList
label size_;
//- Vector of values of type T，以T为类型形成的List，这里是初始的指针
T* __restrict__ v_;
...
}
```

第一个变量用来存储 `UList` 的尺寸，第二个变量用来存储首地址。

我们这里介绍一下基础的功能

1. 通过内置的类 `less` 和 `greater` 规定大小的判断。
2. 通过构造函数进行赋值
3. 返回迭代器的收尾位置
4. 通过检索范围检查当前的 `List` 的正确性
5. 深度和前度copy，区别在于只复制指针还是连带整个数据一起复制

6. 基本的索引`[] =`等的重定义
7. 迭代器
8. 返回当前`List`尺寸，判断为空，交换的成员函数
9. 和IO相关的函数
10. 类外定义的相关排序等功能。

List

我们首先来看注释

```
Description
    A 1D array of objects of type \<T\>, where the size of the vector
    is known and used for subscript bounds checking, etc.

    Storage is allocated on free-store during construction.
 同样是size已知的T类型一维数组，size同样可以用来做下标边界检查
 但是内存是在构造函数中allocate的（这里是区别）
```

他继承了`UList`，如下：

```
template<class T>
class List
: public UList<T>
{  没有新的变量
    ...
}
```

其所实现的基础功能如下：

1. 创建内存空间
2. 返回size，以及resize
3. clear和append操作
4. =的操作符重定义

List中的forAll等宏定义

这里非常常用，另外写在这里
https://blog.csdn.net/qq_40583925/article/details/106989038

用List和各种基础类创建的类型

这里的类型`T`，会在代码中给定为不同的类，从而形成最终使用的类，比如：

```
typedef List<label> labelList;
typedef UList<scalar> scalarUList;
typedef List<scalar> scalarList;
```

List中的文件读取

前面大体的实现并不难理解，常常实现部分就一句话，基本上看到函数名就了解大概的功能。而其中比较需要关注的就是和IO相关的部分，真正用户使用过程中，给定一系列文件，就要用到这个类进行读取。但是 `List` 的读取，在 `UList` 中并没有给出，而是写在了 `List` 中，它是从 `UList` 继承来的子类。其他部分对 `UList` 进行了一些功能的补充，但是总体功能并没有变化。主要是添加了如下的函数

```
template<class T>
Foam::List<T> Foam::readList(Istream& is)
{
    List<T> L;
    token firstToken(is);
    is.putBack(firstToken);

    if (firstToken.isPunctuation())
    {
        if (firstToken.pToken() != token::BEGIN_LIST)
        {
            FatalIOErrorInFunction(is)
                << "incorrect first token, expected '(', found "
                << firstToken.info()
                << exit(FatalIOError);
        }

        // Read via a singly-linked list
        L = SLList<T>(is);
    }
    else
    {
        // Create list with a single item
        L.setSize(1);

        is >> L[0];
    }

    return L;
}
```

首先创建了 `List L` ，然后判断是否为单个量，如果是就用

```
is >> L[0];
```

如果不是，就用

```
// Read via a singly-linked list
L = SLList<T>(is);
```

而 `SLList` 我们后续会继续给出。