



D2DS | COURSES | 2024

C++ 基础: 范围 for 循环

代码展开和自定义类型支持

- for循环对比-std::vector

- 范围for代码展开分析

- 模拟Python中的range

- dslings - py_range代码实操

- python

```
for i in range(0, 10):  
    print(i)
```

- C++

```
for (auto i : d2ds::py_range(0, 10)) {  
    cout << i << endl;  
}
```

For循环对比 - std::vector

遍历 - 迭代器

```
#include <vector>

int main() {
    std::vector<int> vec { 1, 2, 3, 4 };
    int val;
    for (auto it = vec.begin(); it != vec.end(); it++) {
        val = *it;
        //...
    }

    return 0;
}
```

遍历 - 基于范围for

```
#include <vector>

int main() {
    std::vector<int> vec { 1, 2, 3, 4 };
    for (int val : vec) {
        // ...
    }

    return 0;
}
```

范围for比直接使用迭代器的for要更简洁



范围for代码展开分析

这里引用一下cppreference上对它的解释

```
// https://en.cppreference.com/w/cpp/language/range-for
{ // until C++17
    auto && __range = range-expression ;
    for (auto __begin = begin-expr, __end = end-expr; __begin != __end; ++__begin)
    {
        range-declaration = *__begin;
        loop-statement
    }
}
```



范围for代码展开分析 - 化简

```
{ // 没有展开的形式
    for (int val : vec) {
        // ...
    }
}
{ // 编译器代码展开的可能实现
    auto && __range = vec;
    for (auto __begin = __range.begin(), __end = __range.end(); __begin != __end; ++__be
        auto && val = *__begin;
        // ...
    }
}
{ // 编译器代码展开的可能实现 -- 易读版
    auto __end = vec.end();
    for (auto it = vec.begin(); it != __end; ++it) {
        int val = *it;
        // ...
    }
}
```

范围for代码展开分析 - 自定义类型支持



```
// 编译器代码展开的可能实现 -- 易读版
auto __end = vec.end();
for (auto it = vec.begin(); it != __end; ++it) {
    int val = *it;
    // ...
}
```

范围for是一个
基于C++迭代器模型的语法糖

- 类型需要实现begin/end返回迭代器
- 对应迭代器类型要有类指针行为

模拟Python中的range

Python - range

- `range(start, stop)`
- `range(start, stop, step)`

range对象表示不可变的数字序列
通常用于在for循环中

```
speak@speak-pc:~/workspace/github/d2ds$ python3
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC
Type "help", "copyright", "credits" or "license"
>>> for i in range(0, 10):
...     print(i)
...
0
1
2
3
4
5
6
7
8
9
>>> for i in range(0, 50, 5):
...     print(i)
...
0
5
10
15
20
25
30
35
40
45
```



模拟Python中的range - py_range接口设计

```
for (int val : d2ds::py_range(0, 10)) {  
    d2ds_assert_eq(val, index);  
    index++;  
}
```

生成 0 1 2 3 4 5 6 7 8 9

生成 0 5 10 15 20 25 30 35 40 45

```
int index = 0, step = 5;  
for (auto val : d2ds::py_range(0, 50, step)) {  
    d2ds_assert_eq(val, index);  
    index += step;  
}
```

上述代码中, 在接口的使用上为了更像Python中的range, py_range也遵从了如下设计

接口	简介
py_range(start, stop, step = 1)	step为值变化步长默认为1.数据生成遵从左闭右开原则



模拟Python中的range - py_range类型定义

```
d2ds::py_range(0, 10);  
d2ds::py_range(0, 5, 200);
```

py_range的构造函数为了简单, 使用了三个int作为输入参数, 并且为了支持上面两种使用模式最后一个参数step使用了默认参数为1的设置

```
class py_range {  
public:  
    py_range(int start, int stop, int step = 1) {  
    }  
};
```



模拟Python中的range - begin/end结构

```
d2ds::py_range range(2, 10);
auto __begin = range.begin();
auto __end = range.end();
```

给py_range实现两个无参数的成员函数begin和end, 搭出基本结构

```
class py_range {
public:
    py_range(int start, int stop, int step = 1) {
        }
    void * begin() const {
        return nullptr;
    }

    void * end() const {
        return nullptr;
    }
};
```

using IteratorType = XXX;

模拟Python中的range - 迭代器实现



实现对step=1的情况支持

- const int *: 当做迭代类型
- __mArr: 存储生成的数据

这里为了简单直接使用 const int *
作为py_range的迭代器类型, 它的好
处是 — 天然支持 * 和 ++ 操作

迭代器不是指针, 但指针天然是迭代器

<https://sunrisepeak.github.io/d2ds-courses>

```
class py_range {  
public:  
    py_range(int start, int stop, int step = 1) {  
        __mLen = stop - start;  
        for (int i = 0; i < __mLen; i++) {  
            __mArr[i] = i + start;  
        }  
    }  
  
public:  
    const int * begin() const {  
        return __mArr;  
    }  
  
    const int * end() const {  
        return __mArr + __mLen;  
    }  
  
private:  
    int __mLen;  
    int __mArr[100];  
};
```

模拟Python中的range - 完整实现



- 实现对step=x的情况支持
- py_range当前的限制

```
d2ds_assert(start < stop);
d2ds_assert(step > 0);
d2ds_assert(__mLen <= 100);
```

- py_range实现不优雅?

迭代器设计模式章节将会让py_range更优雅一点

<https://sunrisepeak.github.io/d2ds-courses>

```
class py_range {
public:
    py_range(int start, int stop, int step = 1) {
        __mLen = (stop - start) / step;
        d2ds_assert(start < stop);
        d2ds_assert(step > 0);
        d2ds_assert(__mLen <= 100);
        for (int i = 0; i < __mLen; i++) {
            __mArr[i] = start;
            start = start + step;
        }
    }

public:
    const int * begin() const {
        return __mArr;
    }

    const int * end() const {
        return __mArr + __mLen;
    }

private:
    int __mLen;
    int __mArr[100];
};
```

总结 - C++范围for分析



- 用法 – 对比普通for和范围for的使用
- 用处 – 通过分析器代码生成来介绍自定义数据结构如何实现
- 示例 – python - range | C++ - py_range

dslings - py_range代码练习



动手写py_range

https://sunrisepeak.github.io/d2ds-courses