C++11 advanced

Contents

1. Move semantic
2. Functor
3. Lamda expression
4. Smart Pointer

VC IVI Development Center Vietnam

Hanoi, Jan 2018

Move Semantic

- Let ‘s Analyze example below :

```
#include <stdio.h>

int main (int argc, char* argv[]) {

    int b = 5;

    int c = 5 + 6 - (9 * b);

    b - c;

    return 0;
}
```

Directly assigning

how it run?

No assignee ??

Move Semantic

- Lvalue: what we can put it on the left of “=“ operator and compiler does not notice error.
  Example:

  int b = 5;

- Rvalue: what we can put it on the left of “=“ operator and compiler notice error.

  Example:

  (b + d) = 4;

Move Semantic

- Lvalue reference : is normal reference, it ‘s used to refer to normal lvalue.

  Example:

  ```
  int x = 4;
  int &b = x;
  ```

- Rvalue reference : refer to rvalue.

  Example:

  ```
  int &&b = 5 + 6;
  ```

Move Semantic

- Copy sematic: copy data of object to new object, at the end, all of object have same data.

Example:

```
#include <stdio.h>

class A {
 public:
  A() {};
  A(std::string _name) {
    m_name = _name;
  }

  std::string get_name() {
    return m_name;
  }

 private:
  std::string m_name;
};

int main (int argc, char* argv[]) {

  A a("tienlam");

  std::string get_name = a.get_name();

  return 0;
}
```

Move Semantic

- move sematic: move data of an object to new object, at the end, new object hava data of old object and old data of old object is released.

Example:

```
const Data& Data(Data && value)
{
        this->_pData = value._pData;
        value._pData = nullptr;
        this->_n = value._n;
        return *this;
}
```

Move Semantic

- std::move:

```
 1 #include <stdio.h>
 2 #include <string>
 3
 4 class A {
 5  public:
 6   A() {};
 7
 8   A(std::string _name) {
 9     m_name = _name;
10   }
11
12   std::string get_name() {
13     return m_name;
14   }
15
16  private:
17   std::string m_name;
18 };
19
20 int main (int argc, char* argv[]) {
21
22   A a("tienlam");
23
24   A b = std::move(a);
25
26   printf("a is %s and b is %s \n", a.get_name().c_str(), b.get_name().c_str());
27
28   return 0;
29 }
```

Functor

- Concept: **Functors** are objects that can be treated as though they are a function or function pointer

Example:

```
 1 #include <bits/stdc++.h>
 2 class increment {
 3  private:
 4   int num;
 5  public:
 6   increment(int n) : num(n) { }
 7   int operator () (int arr_num) const {
 8     return num + arr_num;
 9   }
10 };
11
12 int main() {
13     int arr[] = {1, 2, 3, 4, 5};
14     int n = sizeof(arr) / sizeof(arr[0]);
15     int to_add = 5;
16
17     std::transform(arr, arr+n, arr, increment(to_add));
18
19     for (int i=0; i<n; i++)
20       std::cout << arr[i] << " ";
21 }
```

Lamda expression

- Concept : anonymous function, as a normal function but it have no name.

Syntax:

[capture-list](param){to do}

```
1  #include<iostream>
2
3 int main() {
4       int a = 0, b = 0;
5      [a, &b]() mutable { a = 1; b = 1; } ();
6      std::cout << a << std::endl; // in 0
7      std::cout << b << std::endl; // in 1
8 }
```

Smart pointer

- Concept: is a normal pointer

auto delete if it ‘s out of scope

```
 1 #include<iostream>
 2
 3 class SmtPointer {
 4  public:
 5   SmtPointer(int size) {
 6     data = new int(size);
 7   }
 8   ~SmtPointer() {
 9     std::cout << "audo delete data " << std::endl;
10     delete[] data;
11   }
12  private:
13   int * data;
14 }
15
16 int main() {
17   {
18     SmtPointer x(500);
19   }
20
21   return 0;
22 }
```

References

- https://www.stdio.vn/articles/rvalue-references-va-move-semantics-28
- http://devnt.org/modern-c-functors/
- https://kipalog.com/posts/C---lambda
- https://mbevin.wordpress.com/2012/11/18/smart-pointers/

thank you!