Cpp 11 / 万能引用、引用折叠和完美转发

一、万能引用

1、英文：Universal Reference 。

2、诞生的原因

因为 C++ 中存在左值引用和右值引用，导致若想同时实现既可传入左值又可传入右值的功能，需要对相同函数进行重载，导致代码 冗余。

3、解决办法

为了解决上述问题，就诞生了万能引用，具体用法如下：

```
template<typename T>
void func(T&& param) {
    ;
}
```

主要实现方法是依靠 C++ 强大的模板推导能力，在编译期间确定 param 是左值还是右值。

4、扩展

&&，在模板形参列表中代表万能引用，在其他地方就代表右值引用。

5、栗子

```
#include <iostream>

template <typename T>
void func(T &&param)
{
    std::cout << param << std::endl;
}
int main()
{
    int num = 0;
    func(num);
    func(100);
    return 0;
}
```

```
0
100
```

二、引用折叠

1、英文：Reference Collapse 。

2、诞生的原因

模板函数：

```
template <typename T>
void func(T &param)
{
    std::cout << param << std::endl;
}

template <typename T>
void func(T &&param)
{
    std::cout << param << std::endl;
}
```

函数形参由于有左值引用和右值引用之分，传入函数的数据也有左值引用和右值引用的区分，这就分出了 4 种情况。

但是 C++ 是不能对引用进行引用的，故需要一种方案，判定上述 4 种情况下最终的结果是左值引用还是右值引用。

3、解决办法

为了解决上述问题，诞生了引用折叠这个概念，也就是判定上述 4 种情况下最终的结果是左值引用还是右值引用的方案，如下：

形参	传入数据	结果
&	&	&
&	&&	&
&&	&	&
&&	&&	&&

可以发现，形参和传入数据只要有一个是左值引用，其结果就是左值引用；只有全部都是右值引用的情况下其结果才能是右值引用。

4、注意

引用折叠只能发生在模板函数中，即：编译期间。

5、栗子

```
#include <iostream>

template <typename T>
void func_L(T &param)
{
    std::cout << param << std::endl;
}

template <typename T>
void func_R(T &&param)
{
    std::cout << param << std::endl;
}

int getvalue()
{
    return 100;
}
int main()
{
    int num = 0;
    int &&num_r = getvalue();
    func_L(num);
    func_L(num_r);
    func_R(num);
    func_R(200);
    return 0;
}
```

```
0
100
0
200
```

三、完美转发

1、英文：Perfect Forwarding 。

2、诞生的原因

经过上述引用折叠之后，传之前的数据的引用类型和传入之后的引用类型可能发生变化，怎么才能保持引用类型呢？如下：

```
#include <iostream>

template <typename T>
void func(T &param)
{
    std::cout << "左值" << std::endl;
}
template <typename T>
void func(T &&param)
{
    std::cout << "右值" << std::endl;
}

template <typename T>
void warp(T &&param)
{
    func(param);
}

int main()
{
    int num = 0;
    warp(num);
    warp(100);
    return 0;
}
```

```
左值
左值
```

3、解决办法

为了解决上述问题，增加了完美转发的概念，即：经过转发之后，数据的引用类型恢复到之前的类型。

4、栗子

```
#include <iostream>

template <typename T>
void func(T &param)
{
    std::cout << "左值" << std::endl;
}
template <typename T>
void func(T &&param)
{
    std::cout << "右值" << std::endl;
}

template <typename T>
void warp(T &&param)
{
    func(std::forward<T>(param));
}

int main()
{
    int num = 0;
    warp(num);
    warp(100);
    return 0;
}
```

```
左值
右值
```

(SAW：Game Over！)