第20章 结构化绑定

《现代C++语言核心特性解析》 谢丙堃

接收多个返回值

- 例子

```
std::tuple<int, int> return_multiple_values()
{
    return std::make_tuple(11, 7);
}

int main()
{
    int x = 0, y = 0;
    std::tie(x, y) = return_multiple_values();
    std::cout << "x=" << x << " y=" << y << std::endl;
}
```

使用结构化绑定

- 基本语法

```
auto return_multiple_values()
{
    return std::make_tuple(11, 7);
}

int main()
{
    auto[x, y] = return_multiple_values();
    std::cout << "x=" << x << " y=" << y << std::endl;
}
```

使用结构化绑定

- 并非只适用于函数返回结果

```
struct BindTest {
    int a = 42;
    std::string b = "hello structured binding";
};

int main()
{
    BindTest bt;
    auto[x, y] = bt;
    std::cout << "x=" << x << " y=" << y << std::endl;
}
```

结构化绑定的原理

- 在结构化绑定中编译器会根据限定符生成一个等号右边对象的匿名副本，而绑定的对象正是这个副本而非原对象本身。另外这里的别名就真的是单纯的别名，别名的类型和绑定目标对象的子对象类型相同。

```
BindTest bt;
const auto [x, y] = bt;
// 编译器为其生成的代码大概是这样的:
BindTest bt;
const auto _anonymous = bt;
aliasname x = _anonymous.a
aliasname y = _anonymous.b
```

结构化绑定的3种类型

- 绑定到原生数组

```
int main()
{
    int a[3]{ 1, 3, 5 };
    auto[x, y, z] = a;
    std::cout << "[x, y, z]=["
        << x << ", "
        << y << ", "
        << z << "]" << std::endl;
}
```

结构化绑定的3种类型

- 绑定到结构体和类对象

```
class BindBase {
public:
    int a = 42;
    double b = 11.7;
};
class BindTest : public BindBase {};

int main()
{
    BindTest bt;
    auto[x, y] = bt;    // 编译成功
}
```

结构化绑定的3种类型

- 绑定到元组和类元组的对象
  - 需要满足条件：
    1. 首先需要满足std::tuple_size<T>::value是一个符合语法的表达式，并且该表达式获得的整数值与标识符列表中的别名个数相同；
    2. 其次，类型T还需要保证std::tuple_element<i, T>::type也是一个符合语法的表达式，其中i是小于std::tuple_size<T>::value的整数，表达式代表了类型T中第i个元素的类型；
    3. 最后，类型T必须存在合法的成员函数模板get<i>()或者函数模板get<i>(t)，其中i是小于std::tuple_size<T>::value的整数，t是类型T的实例，get<i>()和get<i>(t)返回的是实例t中第i个元素的值。

绑定的访问权限问题

- 例子

```
struct A {
    friend void foo();
private:
    int i;
};

void foo() {
    A a{};
    auto x = a.i; // 编译成功
    auto [y] = a; // 编译失败
}
```

感谢您的观看

欢迎关注