11.1 result_of

result_of 是一个很小但很有用的组件，可以帮助程序员确定一个调用表达式的返回类型，主要用于泛型编程和其他 Boost 库组件，它已被收入 C++11 标准。

result_of 位于名字空间 boost，为了使用 result_of 组件，需要包含头文件<boost/utility/result_of.hpp>，即：

```
#include <boost/utility/result_of.hpp>
using namespace boost;
```

11.1.1 原理

所谓“调用表达式”，是指一个含有 operator()的表达式，函数调用或函数对象调用都可以称为调用表达式，而 result_of 以模板元函数的方式确定这个表达式所返回的类型。[①]

result_of 模板类的示意代码如下：

```
template<typename F>
struct result_of
{
  typedef some_define type;                    //使用::type 返回元计算得到的类型
};
```


给定一个调用表达式 F，可以通过内部类型定义 result_of<F>::type 获得返回值的类型。

假设我们有一个类型 F，它可以是函数指针、函数引用或者成员函数指针，当然也可以是函数对象类型，它的一个实例是 f。F 有一个 operator()，参数是(T1 t1, T2 t2)，这里 T1、T2 是两个模板类型，那么

```
result_of<F(T1,T2)>::type
```

[②]

就是 f (t1, t2)的返回值类型。

result_of 虽然小，但它用到了很多 C++的高级特性，如模板偏特化和 SFINAE，并且部分依赖于编译器。如果编译器支持 C++11 的 decltype 关键字，那么它就会利用 decltype 的能力。

---

① 单从这一点来理解，result_of 的功能有些类似 C++11 的 auto。auto 可以确定一个表达式的类型，但不具备推演调用表达式的能力。

② 这种在模板中使用类似函数声明的形式还会在本书讲述 function 库时遇到。

11.1.2 用法

因为 result_of 使用了模板元编程技术，对元编程不熟悉的读者来说其工作原理好像有些抽象难以理解，现在我们用实例解说一下。

```
typedef double (*Func)(double d);
```

这行代码定义了一个函数指针类型 Func，它的调用式接受一个 double 类型，返回类型为 double。

```
Func func = sqrt;
```

这行代码声明了 Func 的一个实例（变量）func，一个具体的函数指针，并把它赋值为 sqrt——C 标准库中的开平方数学函数。那么

```
result_of<Func(double)>::type x = func(5.0);
```

这行代码必然可以正确通过编译，x 的类型将被推导为 double。

完整的程序如下：

```
#include <boost/utility/result_of.hpp>
using namespace boost;

int main()
{
  typedef double (*Func)(double d);                    //函数指针类型定义
  Func func = sqrt;                                    //函数指针赋值

  result_of<Func(double)>::type x = func(5.0);   //类型推导
  cout << typeid(x).name();
}
```

上面的实例演示了 result_of 的基本用法，不是很具有吸引力，用 auto 也可以完成同样的功能。但是，当处于一个泛型上下文之中，周围没有真实的类型，而且没有表达式的时候，auto 就无能为力了，例如这样的一个简单的泛型函数：

```
template<typename T, typename T1>
??? call_func(T t, T1 t1)                          //T 是个可调用的类型
{ return t(t1);}
```

无论如何，auto 都派不上用场，这里不存在任何赋值表达式，只有函数调用式。

这正是 result_of 发挥威力的机会，它可以正确推导出返回类型，像这样：

```
template<typename T, typename T1>
typename result_of<T(T1)>::type call_func(T t, T1 t1)
```

```
{ return t(t1);}
```

这里必须在 result_of<>::type 前加上关键字 typename，否则编译器会认为 type 是 result_of 的成员变量，从而产生找不到声明的编译错误。

仍然使用刚才定义的函数指针，使用 result_of 的完整程序如下：

```
#include <boost/utility/result_of.hpp>
using namespace boost;

template<typename T, typename T1>
typename result_of<T(T1)>::type call_func(T t, T1 t1)
{ return t(t1);}

int main()
{
  typedef double (*Func)(double d);
  Func func = sqrt;

  auto x =  call_func(func, 5.0);     //赋值表达式，可以用 auto
  cout << typeid(x).name();
}
```

如果读者暂时还不甚理解 result_of 的用法也不要紧，可以阅读推荐书目[3]，熟悉了模板元编程的基本概念再来学习它。