

Introduction to signal0x

Wang Feng

May 11, 2011

1 Tutorial

1.1 Motivation

This is another wheel invented as an implementation of Observer Pattern written in C++.

This is also a demonstration of the upcoming c++0x.

Features:

1. Thread-safe
2. Faster
3. Non-intrusive connection tracking

1.2 Compatibility note

Signal0x library compiles currently only compatible with g++4.6 and g++4.7, as it employs many c++0x features such as:

1. variadic template
2. range-based for loop
3. rvalue reference
4. lambda
5. mutex
6. ...

The header file of signal0x is

```
#include <signal.hpp>
```

The namespace of signal0x is

```
using signal0x::signal;
```

A typical commandline compilation is

```
g++ -o test test.cc -std=c++0x -O2 -I PATH/TO/HEAD/FILE
```

1.3 Connecting to signals

1.3.1 First example – hello world

```
1 // an ordinary function
2 void hello_world()
3 {
4     std::cout << "Hello World.\n";
5 }
6
7 //(1) a signal with no arguments and a void return value
8 signal0x::signal<void> sig;
9
10 //(2) connect function to signal
11 sig.connect( hello_world );
12
13 //(3) signal triggered
14 sig();
```

This example writes "Hello World." using signals and slots in the following way:

1. create a signal `sig`, a signal that takes no arguments and has a void return value
2. connect a slot, the `hello_world` function, to the signal using the `connect` method
3. the signal is triggered, which in turns invokes `hello_world()` to print "Hello World."

1.3.2 Slots

Several kinds of slots can be connected to a signal

1. a function

```
1 #include <signal.hpp>
2 using signal0x::signal;
3
4 void f(){}
5 signal<void> sig;
6 sig.connect( f );
7 sig();
```

2. a functor

```
1 #include <signal.hpp>
2 using signal0x::signal;
3
```

```

4 struct f{ void operator()() const {} };
5 f f_;
6 signal<void> sig;
7 sig.connect( f() );
8 sig.connect( f_ );
9 sig();

```

3. a lambda object

```

1 #include <signal.hpp>
2 using signal0x::signal;
3
4 signal<void> sig;
5 sig.connect( []{} );//an anonymous lambda object
6 sig();

```

4. a pointer to a member function

```

1 #include <signal.hpp>
2 #include <functional>//for std::bind
3 using signal0x::signal;
4
5 struct f
6 {
7     void ff() {}
8 };
9 f f_;
10 signal<void> sig;
11 sig.connect( std::bind( &f::ff, &f_ ) );
12 sig();

```

1.3.3 Multiple slots

Multiple slots can be connected to a same signal one by one, and if the signal is triggered, all slots connected to this signal will be called.

Another "hello world" example

```

1 #include <signal.hpp>
2 #include <iostream>
3
4 void hello{ std::cout << "Hello "; }
5 void world{ std::cout << "World!"; }
6
7 signal0x::signal<void> sig;
8 sig.connect( hello );
9 sig.connect( world );

```

```
10 sig(); //print "Hello World!"
```

Simply, multiple slots can be connected to a same signal at the same time

for example:

```
1 #include <signal.hpp>
2 #include <iostream>
3
4 void hello{ std::cout << "Hello "; }
5 void world{ std::cout << "World!"; }
6
7 signal0x::signal<void> sig;
8 sig.connect( hello, world );
9 sig(); //print "Hello World!"
```

1.3.4 Ordering multiple slots call priorities

Connection could be created with priority argument, which means slots can be called in a specified order we set.

an example:

```
1 #include <signal.hpp>
2 #include <iostream>
3
4 void hello{ std::cout << "Hello "; }
5 void world{ std::cout << "World!"; }
6 struct good
7 {
8     void operator()() const
9     { std::cout << " Good "; }
10 };
11
12 signal0x::signal<void> sig;
13
14 sig.connect( 1, world );
15 sig.connect( hello, 0 );
16 sig.connect( 3, [](){std::cout << "Morning!";} );
17 sig.connect( good(), 2 );
18
19 sig(); //print "Hello World! Good Morning!"
```

if several slots connected to a same signal at the same time, they share same priority

```
1 #include <signal.hpp>
2 #include <iostream>
3
```

```

4 void hello{ std::cout << "Hello "; }
5 void world{ std::cout << "World!"; }
6 struct good
7 {
8     void operator()() const
9     { std::cout << " Good "; }
10 };
11
12 signal0x::signal<void> sig;
13
14 sig.connect( 0, hello, world );
15 sig.connect( 1, good(), []() {std::cout << "Morning!";} );
16
17 sig(); //print "Hello World! Good Morning!"

```

1.4 Passing values to and from slots

Signals can propagate arguments to each of the slots they call.

most code of the example below is borrowed from boost::signals

```

1 #include <signal.hpp>
2 #include <iostream>
3
4 void print_sum(float x, float y)
5 {
6     std::cout << "The sum is " << x+y << std::endl;
7 }
8
9 void print_product(float x, float y)
10 {
11     std::cout << "The product is " << x*y << std::endl;
12 }
13
14 void print_difference(float x, float y)
15 {
16     std::cout << "The difference is " << x-y << std::endl;
17 }
18
19 void print_quotient(float x, float y)
20 {
21     std::cout << "The quotient is " << x/y << std::endl;
22 }
23
24 int main()
25 {
26     signal0x::signal<void, float, float> sig;
27     sig.connect( print_sum, print_product, print_difference, print_quotient
28 );
29     sig( 3, 4 );

```

```

29
30     return 0;
31 }
32

```

This programme will print the following

```

The sum is 7
The product is 12
The difference is -1
The quotient is 0.75

```

This means any values that are given to sig when it is called like a function are passed to each of the slots.

1.5 Collecting slots return values

Just as slots can receive arguments, they can also return values. Following STL's patterns, In signal0x, we collect return values by forwarding them into an output iterator

Slightly modify previous example, we redirect the return values to an ostream iterator

```

1 #include <signal.hpp>
2 #include <iostream>
3 #include <iterator>
4
5 float sum(float x, float y) { return x+y; }
6 float product(float x, float y) { return x*y; }
7 float difference(float x, float y) { return x-y; }
8 float quotient(float x, float y) { return x/y; }
9
10 int main()
11 {
12     signal0x::signal<float, float, float> sig;
13     sig.connect( sum );
14     sig.connect( product );
15     sig.connect( difference );
16     sig.connect( quotient );
17     sig( std::ostream_iterator<float>(std::cout, "\n"), 3.0,
4.0 );
18
19     return 0;
20 }
21

```

This programme will print the following

```
7
12
-1
0.75
```

Please note here, if we connect all slots at the same time, they will be combined, and only returning the last function's return value; This means if the connection is written as

```
1    sig.connect( sum, product, difference, quotient );
```

only 0.75 will be printed.

1.6 Disconnecting slots

Some times slots are only used to receive a few events and are then disconnected, so the programmer needs control to decide when a slot should no longer be connected.

an example

```
1 #include <signal.hpp>
2 #include <iostream>
3 #include <iterator>
4
5 void hello(){ std::cout << "Hello "; }
6 void world(){ std::cout << "World!"; }
7
8 int main()
9 {
10     signal0x::signal<void> sig;
11
12     auto con = sig.connect( hello, world );
13     sig(); // this will print "Hello World!"
14
15     sig.disconnect( con ); // disconnect the connection
16     sig(); // nothing happens here
17
18     return 0;
19 }
20
```

1.7 Blocking slot

temp empty

1.8 Scoped connections

The `signal0x::scope_connection` class references a signal/slot connection that will be disconnected when the `scoped_connection` class goes

out of scope.

an example

```
1 #include <signal.hpp>
2 #include <iostream>
3 #include <cstdint>
4
5 template< std::size_t N >
6 struct f
7 {
8     void operator()() const
9     { std::cout << "\nfunctor f<" << N << "> called."; }
10 };
11
12 int main()
13 {
14     signal0x::signal<void> sig;
15     sig.connect( f<0>() );
16     std::cout << "\nsignal triggered out of scope.";
17     sig();
18     {
19         signal0x::scope_connection<void> s( sig, f<1>() );
20         std::cout << "\nsignal triggered in scope.";
21         sig();
22     }
23     std::cout << "\nsignal triggered out of scope.";
24     sig();
25
26     return 0;
27 }
```

this program will produce

```
signal triggered out of scope.
functor f<0> called.
signal triggered in scope.
functor f<0> called.
functor f<1> called.
signal triggered out of scope.
functor f<0> called
```


1.9 Examples

2 Comparison With Boost::signals and sigc::signal

2.1 Grammar Comparison

2.2 Benchmark

3 Design and Implementation