linux系統編程：setjmp和longjmp函數用法

```
#include <stdio.h>

#include <setjmp.h>

// jmp_buf：數組，保存棧信息即運行環境

jmp_buf buf;

double Divide( double a, double b)

{

if (b == 0.0 )

{

longjmp(buf, 1 ); // throw

}

else

return a / b;

}

// setjmp保存當前棧信息，成功返回0，當執行到longjmp時，

// 恢復棧信息即跳轉到setjmp位置重新執行setjmp

// 且此次返回值根據longjmp函數參數給定

int main( void )

{

int ret;

ret = setjmp(buf);

if (ret == 0 ) // try

{

printf( " division ...\n " );

printf( " %f\n " , Divide( 5.0 , 0.0 ));

}
```

```
else  if (ret == 1 ) // catch

{

printf( " divisiong by zero\n " );

}

return  0 ;

}
```

運行結果：

division ...

divisiong by zero

其實上次錯誤處理模式已經是c++異常處理雛形

上述相當於：

```
#include <iostream>

using  namespace std;

double Divide( double a, double b)

{

if (b == 0.0 )

{

throw  1 ; // throw

}

else

return a / b;

}

int main( void )

{

try  // try

{

cout << " division ... " << endl;
```

```
cout <<Divide( 3.0 , 0.0 )<< endl;

cout <<Divide( 5.0 , 0.0 )<< endl;

}

catch ( int ) // catch

{

cout << " divisiong by zero " << endl;

}

return  0 ;

}
```

運行結果同上
