# **游戏服务器之性能统计**

性能统计分析器用来统计多个函数在一定执行次数下的执行时间，并输出函数执行时间较长的函数名和时间。

（1）性能统计分析器

1

class Performance\_Statistics

2

｛

3

private:

4

struct Performance

5

｛

6

Performance():\_times(0),\_total\_time(0) ｛｝

7

/\*\*

8

\* \author: cjy

9

\* \description: 执行的次数

10

\*/

11

unsigned long \_times;

12

/\*\*

13

\* \author: cjy

14

\* \description:执行的总时间

15

\*/

16

unsigned long \_total\_time;

17

｝;

18

std::map&lt;std::string, Performance&gt; \_times;

19

timer \_log\_timer;

20

public:

21

Performance\_Statistics(const int how\_long) : \_log\_timer(how\_long) ｛｝

22

~Performance\_Statistics() ｛｝

23

void inc(const std::string &amp;func, const unsigned long total)

24

｛

25

Performance &amp;mt = \_times[func];

26

if (mt.\_times)

27

｛

28

++mt.\_times;

29

mt.\_total\_time += total;

30

｝

31

else

32

｛

33

++mt.\_times;

34

mt.\_total\_time = total;

35

｝

36

｝

37

void reset(const realtime &amp;ct, const bool force\_print)

38

｛

39

if (force\_print || \_log\_timer(ct))

40

｛

41

g\_log-&gt;debug("[分时统计]：%s, %lu", force\_print ? "force" : "timer", \_times.size());

42

for(std::map&lt;std::string, Performance&gt;::iterator it = \_times.begin(); it != \_times.end(); ++it)

43

｛

44

if (it-&gt;second.\_times &amp;&amp; (it-&gt;second.\_total\_time / 1000000L) &gt; 5)//输出5ms以上的

45

｛

46

g\_log-&gt;debug("[分时统计]：%s, %lu毫秒, %lu次, %lu毫秒/次",it-&gt;first.c\_str(), it-&gt;second.\_total\_time /1000000L, it-&gt;second.\_times, (it-&gt;second.\_total\_time / 1000000L) / it-&gt;second.\_times);

47

｝

48

｝

49

｝

50

\_times.clear();

51

｝

52

｝;

（2）统计方便函数，在构造函数和析构函数里执行时间统计

1

class Function\_Exe\_Time

2

｛

3

private:

4

const std::string \_func;

5

struct timespec \_tv\_1;

6

public:

7

static Performance\_Statistics my\_func;

8

Function\_Exe\_Time(const std::string &amp;func) : \_func(func)

9

｛

10

clock\_gettime(CLOCK\_REALTIME, &amp;\_tv\_1);

11

｝

12

~Function\_Exe\_Time()

13

｛

14

struct timespec \_tv\_2;

15

clock\_gettime(CLOCK\_REALTIME, &amp;\_tv\_2);

16

unsigned long end=(unsigned long)\_tv\_2.tv\_sec\*1000000000L + \_tv\_2.tv\_nsec;

17

unsigned long begin=(unsigned long)\_tv\_1.tv\_sec\*1000000000L + \_tv\_1.tv\_nsec;

18

my\_func.inc(\_func, end-begin);

19

｝

20

｝;

（3）定时器

1

/\*\*

2

\* \author cjy

3

\* \description 定时器

4

\* 固定时间间隔的定时器，方便对于时间间隔的判断，精确到毫秒级

5

\*/

6

class timer

7

｛

8

public:

9

/\*\*

10

\* \author cjy

11

\* \description构造函数

12

\* \param how\_long 定时器的时间，单位：毫秒

13

\* \param first 有些定时器可能希望在启动时就可以执行一次,所以不能直接addDelay哦

14

\*/

15

explicit timer(const int64 how\_long, bool first=false, const int64 delay=0) : \_long(how\_long), \_timer()

16

｛

17

if(!first)

18

\_timer.addDelay(\_long+delay);

19

｝

20

/\*\*

21

\* \author cjy

22

\* \description构造函数

23

\* \param how\_long 定时器的时间，单位：毫秒

24

\* \param first 有些定时器可能希望在启动时就可以执行一次,所以不能直接addDelay哦

25

\* \param ctv 当前时间

26

\*/

27

explicit timer(const int64 how\_long, bool first , realtime &amp;ctv) : \_long(how\_long), \_timer(ctv)

28

｛

29

if(!first)

30

\_timer.addDelay(\_long);

31

｝

32

/\*\*

33

\* \author cjy

34

\* \description重新设置定时器的精度和开始计时时间

35

\* \param how\_long 定时器的时间，单位：毫秒

36

\* \param ctv 当前时间

37

\*/

38

void reset(const uint64 how\_long, const realtime &amp;cur)

39

｛

40

\_long = how\_long;

41

\_timer = cur;

42

\_timer.addDelay(\_long);

43

｝

44

/\*\*

45

\* \author cjy

46

\* \description重新设置定时器的时间

47

\* \param cur 指定定时器启动的时间

48

\*/

49

void current(const realtime &amp;cur)

50

｛

51

\_timer = cur;

52

｝

53

/\*\*

54

\* \author cjy

55

\* \description延时定时器时间

56

\* \param cur 指定定时器启动的时间

57

\* \param delay 延时时间

58

\*/

59

void next(const realtime &amp;cur, const uint32 delay)

60

｛

61

\_timer = cur;

62

\_timer.addDelay(delay);

63

｝

64

/\*\*

65

\* \author cjy

66

\* \description重新设置定时器的时间

67

\* \param cur 指定定时器启动的时间

68

\*/

69

void next(const realtime &amp;cur)

70

｛

71

\_timer = cur;

72

\_timer.addDelay(\_long);

73

｝

74

/\*\*

75

\* \author cjy

76

\* \description倒计时剩余秒数.不受时间调整影响.

77

\* \param cur 当前时间

78

\* return 剩余描述

79

\*/

80

inline uint32 leftSec(const realtime &amp;cur)

81

｛

82

return (\_timer.sec() &gt; cur.sec()) ? (\_timer.sec() - cur.sec()) : 0;

83

｝

84

/\*\*

85

\* \author cjy

86

\* \description倒计时剩余毫秒数.受时间调整影响

87

\* \param cur 当前时间

88

\* return 剩余值

89

\*/

90

inline uint64 leftMSec(const realtime &amp;cur)

91

｛

92

return (\_timer.\_msec &gt; cur.\_msec) ? (\_timer.\_msec - cur.\_msec) : 0;

93

｝

94

/\*\*

95

\* \author cjy

96

\* \description定时器检查

97

\* \param cur 检查定时器的时间

98

\* return 是否到达指定时间

99

\*/

100

inline bool operator() (const realtime &amp;cur)

101

｛

102

if (\_timer.\_msec &gt; cur.\_msec)

103

｛

104

return false;

105

｝

106

addDelay(cur);

107

return true;

108

｝

109

inline void addDelay(uint64 addLong)//添加延迟

110

｛

111

\_timer.addDelay(addLong);

112

｝

113

private:

114

/\*\*

115

\* \author cjy

116

\* \description定时器时间间隔

117

\*/

118

uint64 \_long;

119

/\*\*

120

\* \author cjy

121

\* \description上次检查定时器的时间

122

\*/

123

realtime \_timer;

124

private:

125

/\*\*

126

\* \author cjy

127

\* \description非严格检测，存在积累误差

128

\* 从效率方面考虑,严格监测定时器的需求并不是必须的，去除对严格监测的支持

129

\*/

130

inline void addDelay(const realtime&amp; cur)

131

｛

132

\_timer = cur;

133

\_timer.addDelay(\_long);

134

｝

135

｝;

（4）测试实例

测试ai攻击处理花费时间

1

｛

2

Function\_Exe\_Time func("NPC\_AI\_ATTACK"STR(\_\_LINE\_\_));

3

return doAttackAI();

4

｝

来自：https://blog.csdn.net/chenjiayi\_yun/article/details/35276861