ZLToolKit笔记13：日志

2025年1月9日 19:43

//初始化日志系统 [AUTO-TRANSLATED:25c549de]

```
// Initialize the logging system
Logger::Instance().add(std::make_shared<ConsoleChannel> ());
Logger::Instance().add(std::make_shared<FileChannel>());
Logger::Instance().setWriter(std::make_shared<AsyncLogWriter>());
```

从test_logger.cpp入手：

```
INSTANCE_IMP(Logger, exeName())

Logger::Logger(const string &loggerName) {
    _logger_name = loggerName;
    _last_log = std::make_shared<LogContext>();
    _default_channel = std::make_shared<ConsoleChannel>("default", LTrace);
}
```

其中INSTANACE_IMP是个宏定义：

```
#define INSTANCE_IMP(class_name,__VA_ARGS__...) class_name &class_name::Instance() { static
std::shared_ptr<class_name> s_instance(new class_name(__VA_ARGS__)); static class_name
&s_instance_ref = *s_instance; return s_instance_ref; }
```

扩展到:

```
Logger &Logger::Instance() { static std::shared_ptr<Logger> s_instance(new Logger(exeName()));
static Logger &s_instance_ref = *s_instance; return s_instance_ref; }
```

通过宏定义的方式构造了一个Logger对象的单例。

在Logger的构造函数中：

构造了一个LogContext对象和ConsoleChannel对象。作为默认的日志输出通道。

事实上项目不仅支持ConsoleChannel终端输出，还支持File Channel和广播输出EventChannel。

add函数负责将日志输出通道添加到日志中：

```
void Logger::add(const std::shared_ptr<LogChannel> &channel) {
    _channels[channel->name()] = channel;
}
```

删除日志通道

```
void Logger::del(const string &name) {
    _channels.erase(name);
}
```

获取日志通道

```
std::shared_ptr<LogChannel> Logger::get(const string &name) {
    auto it = _channels.find(name);
    if (it == _channels.end()) {
        return nullptr;
    }
    return it->second;
}
```

设置写日志器

```
void Logger::setWriter(const std::shared_ptr<LogWriter> &writer) {
    _writer = writer;
}
```

。

这个日志模块的设计主要分为以下几个关键部分：

1. **日志上下文（LogContext）**
   - 用于存储每条日志的详细信息，包括日志级别、文件名、函数名、行号、线程名、模块名等。
   - 提供了日志内容的缓存和格式化功能，确保日志输出的完整性和可用性。
2. **日志捕获器（LogContextCapture）**
   - 用于封装日志的生成过程，支持流式操作。
   - 在日志捕获完成时会自动将日志上下文交给日志器进行输出。
3. **日志通道（LogChannel）**
   - 负责日志的具体输出，包括终端输出（ConsoleChannel）、文件输出（FileChannel）和广播输出（EventChannel）。
   - 支持设置日志级别和格式化输出。
4. **日志写入器（LogWriter）**
   - 提供异步写入日志的功能，通过后台线程批量写日志，提升性能。
   - 设计了 AsyncLogWriter，将日志按需推送到不同的日志通道。
5. **日志器（Logger）**
   - 作为日志系统的核心管理类，提供添加、删除日志通道，以及设置日志级别等功能。
   - 支持多通道管理，日志可以同时输出到多个目标（例如控制台和文件）。
6. **文件管理（FileChannel）**
   - 提供了日志文件的自动切片和清理功能。
   - 可以根据时间、大小和数量限制管理日志文件的生命周期。

整体设计方案

该日志模块的设计采用了模块化、可扩展的架构。通过将日志的生成、格式化、写入、通道输出等功能解耦，每个模块独立完成自己的任务。这种设计方案具有以下特点：

- **高性能**：异步写入和多线程支持，降低对主线程的性能影响。
- **灵活性**：支持多种日志输出方式（终端、文件、广播等），可根据需要扩展新的通道类型。
- **易用性**：通过流式接口封装，用户可以方便地生成和记录日志。
- **可维护性**：清晰的模块职责分离，便于后续维护和扩展功能。

整个系统的核心目标是提供一个高效、灵活、易用的日志管理工具，满足不同场景下的日志记录需求。

之后：

```
InfoL << "测试std::cout风格打印： ";
   //ostream支持的数据类型都支持,可以通过友元的方式打印自定义类型数据 [AUTO-TRANSLATED:c857af94]
   // All data types supported by ostream are supported, and custom type data can be printed through friend methods
   TraceL << "object int"<< TestLog((int)1)  << endl;
   DebugL << "object short:"<<TestLog((short)2)  << endl;
   InfoL << "object float:" << TestLog((float)3.12345678)  << endl;
   WarnL << "object double:" << TestLog((double)4.12345678901234567)  << endl;
   ErrorL << "object void *:" << TestLog((void *)0x12345678) << endl;
   ErrorL << "object string:" << TestLog("test string") << endl;
```

上面其实是一系列的宏定义：

```
//可重置默认值 [AUTO-TRANSLATED:b1e0e8b9]
```

```
//Can reset default value
extern Logger *g_defaultLogger;    // 全局的日志指针

//用法: DebugL << 1 << "+" << 2 << '=' << 3;  [AUTO-TRANSLATED:e6efe6cb]
//Usage: DebugL << 1 << "+" << 2 << '=' << 3;
#define WriteL(level) ::toolkit::LogContextCapture(::toolkit::getLogger(), level, __FILE__,
__FUNCTION__, __LINE__)
#define TraceL WriteL(::toolkit::LTrace)
#define DebugL WriteL(::toolkit::LDebug)
#define InfoL WriteL(::toolkit::LInfo)
#define WarnL WriteL(::toolkit::LWarn)
#define ErrorL WriteL(::toolkit::LError)
```

其中:

```
Logger *g_defaultLogger = nullptr;

Logger &getLogger() {
   if (!g_defaultLogger) {
      g_defaultLogger = &Logger::Instance();
   }
   return *g_defaultLogger;
}
```

那么:

```
#define InfoL WriteL(::toolkit::LInfo)
```

扩展到:

```
::toolkit::LogContextCapture(::toolkit::getLogger(), ::toolkit::LInfo, "/home/fangfang/cpp/ZLToolKit-
master/tests/test_logger.cpp", __FUNCTION__, 42)
```

Util.cpp 390行

```
fangfang@fangfang-VM:~/cpp$ cd PicoRedis/
fangfang@fangfang-VM:~/cpp/PicoRedis$ ls
bin    db0  db10  db12  db14  db2  db4  db6  db8  Makefile  testnew
build  db1  db11  db13  db15  db3  db5  db7  db9  src       third_party
fangfang@fangfang-VM:~/cpp/PicoRedis$ cd bin/
fangfang@fangfang-VM:~/cpp/PicoRedis/bin$ ls
redisClient  redisServer
fangfang@fangfang-VM:~/cpp/PicoRedis/bin$ ./redisServer
2025-01-22 14:53:48.161 D [redisServer] [5092-stamp thread] util.cpp:390 operator() | Stamp thread started
2025-01-22 14:53:48.161 I [redisServer] [5092-redisServer] EventPoller.cpp:616 EventPollerPool | EventPoller created size
: 4
2025-01-22 14:53:48.233 D [redisServer] [5092-redisServer] CmdParserFactory.h:19 operator() | Data loaded from Disk !
2025-01-22 14:53:48.233 D [redisServer] [5092-redisServer] Timer.cpp:16 Timer | Timer created with interval: 10 seconds
2025-01-22 14:53:48.233 D [redisServer] [5092-redisServer] Timer.cpp:16 Timer | Timer created with interval: 10 seconds
    Last message repeated 3 times
2025-01-22 14:53:48.233 I [redisServer] [5092-redisServer] TcpServer.cpp:242 start_l | TCP server listening on [::]: 6380
```