开源 mooon 开发规范说明书

一见

目录

1. 前言

谨记：代码是程序员的脸面。严格严谨编写每一行代码，命名每一个名称。网址：http://code.google.com/p/mooon/ :

2. SVN 规范

SVN 上只存放必须的文件，其它文件禁止存放在 SVN 上。其它文件包括：

1) 临时文件：如 Make 时生成的 object 文件；
2) 结果文件：如 Make 时生成的库文件；
3) 可通过其它文件生成的文件，如：Makefile.in 文件；
4) 不直接存放第三方库的源码文件和库文件，而只存放第三方库的源码包文件和二进制包文件。

3. 命令规范

3.1. 文件名和目录名

文件名和目录名均采用 linux 风格命名，即“小写+下划线”方式，如：thread.cpp 和 util_config.h 等。

头文件名的后缀为“.h”，不能为“.hpp”等；实现文件名的后缀为“.cpp”，不能为“.cc”和“.cxx”等。

3.2. 类名和接口名

3.2.1. 类名

类名以大写字母“C”打头，“C”代表 **C**lass，如：class **C**Thread;。

3.2.2. 接口名

接口以大写字母“I”打头，“I”代表 **I**nterface，如：class **I**Logger;。

3.3. 结构体和枚举类型名

结构体和枚举类型的命名规则相同，均带后缀“_t”。

3.3.1. 结构体类型名

示例：

```
/***
  * Agent消息结构头，专用于Agent和Center间通讯
  */
typedef struct
{
    uint32_t byte_order:1;   /** 字节序，0为小字节序，1为大字节序 */
    uint32_t body_length:31; /** 消息体长度 */
    uint16_t version;        /** 消息版本号 */
    uint16_t command;        /** 消息类型 */
    uint32_t check_sum;      /** 校验和，为version、command和body_length三者之和
*/
}
agent_message_t;
```

3.3.2. 枚举类型名

示例：

```
/***
  * 处理结果类型
  */
typedef enum
{
    handle_error,    /** 处理出错 */
    handle_finish,   /** 处理成功完成 */
    handle_continue /** 处理未完成，需要继续 */
}
handle_result_t;
```

3.4. 变量名

所有变量采用“小写+下划线”风格， 除普通变量（包括类成员变量和全局变量）外，其它变量均有相应的前缀。

3.4.1. 类成员变量

类成员变量总是以前缀下划线“_”打头，如：

```
class CThread
{
private:
    pthread_t _thread; // 成员变量
};
```

3.4.2. 全局变量

1) 全局非静态变量总是以前缀“g_”打头，如：extern ILogger* **g_**logger;
2) 全局静态变量总是以前缀“gs_”打头，如：static IAgent* **gs_**agent;

3.4.3. 局部变量

类成员变量和全局变量之外的局部变量，除没有前缀外，其它规则是相同的。

3.4.4. 静态变量

1) 类静态成员变量命名和类成员变量相同，但使用时必须加上类限定符；
2) 局部静态变量的命名和局部谜题相同。

3.4.5. 枚举类型成员名

1) 名枚举成员采用“大写+下划线”风格命名，如：

```
/***
  * 常量定义
  */
enum
{
    /***
      * 内置命令取值范围：[0, MAX_BUILTIN_AGENT_COMMAND]
      * 非内置命令取值范围：
            [MAX_BUILTIN_AGENT_COMMAND+1,
             MAX_NON_BUILTIN_AGENT_COMMAND]
      */
    MAX_BUILTIN_AGENT_COMMAND     = 127, /** 最大的内置命令 */
    MAX_NON_BUILTIN_AGENT_COMMAND = 511  /** 最大的非内置命令 */
};
```

2) 有名枚举成员采用“小写前缀+下划线”命名风格，如：

```
/***
  * 处理结果类型
  */
typedef enum
{
    handle_error,    /** 处理出错 */
    handle_finish,   /** 处理成功完成 */
    handle_continue /** 处理未完成，需要继续 */
}handle_result_t;
```

4. 目录结构

采用扁平化目录结构，尽量减少目录层次数，结构如下：

主目录名
- 分层名称
  - 头文件目录
    - 模块头文件目录
  - 实现文件目录
    - 模块实现文件目录
  - 文档目录
  - 测试代码目录
    - 模块测试文件目录
  - 第三方库目录
    - 二进制包目录
    - 源码包目录

对应的目录层次结构如下：

mooon
- **common_library**
  - include
    - util
    - sys
    - net
  - src
    - util
    - sys
    - net
    - shell
  - doc
    - net
  - test
    - util

```
                        |_______sys
                        |_______net
            |_______add-ones
                        |_______bin
                        |_______src
|____common_component
            |_______include
                        |_______observer
                        |_______agent
                        |_______server
                        |_______dispatcher
            |_______src
                        |_______conf
                        |_______observer
                        |_______agent
                        |_______server
                        |_______dispatcher
            |_______doc
            |_______add-ones // 第三方库
                        |_______bin
                        |_______src
```

请注意 include 下只包含外部需要直接或间接使用到的头文件，其它头文件放在相应的 src 目录下。所有的 Make 文件也只放在相应的 src 目录下。include 目录下不能包含任何实现文件。**只能在 src 下的叶子目录下存放实现文件**。

5. 编译规范

1) 不允许存在任何编译警告，编译时必须将所有警告打开，即使用-Wall 编译开关；
2) mooon 采用 automake 编译方式。请注意 SVN 上不保留 configure 和 make 生成的中间文件和结果文件。

6. 缩略语规范

禁止使用一切非众所周知的缩略语，如：common_library 不能缩写成 cl，common_compoent 也不能缩写成 cc，而应当使用全称。

7. public/protected/private 规范

可以为 protected，则不能声明为 public；可以为 private 的，则不能声明为 protected 或

public。

8. 换行规范

1) **头文件和实现文件必须以空行结尾**；
2) **一个函数体和另一个函数体之间必须有一个空行**；
3) 类成员访问修饰符 public/protected/private，第一个的前一行必须为左大括号{，其它的前一行必须为空行，如：

```
class CMyClass
{
public: // 第一个

private: // 非第一个

private: // 非第一个
};
```

4) **一个类定义与另一个类定义之间必须有一个空行，结构体和枚举类型也是如此**，如：

```
class ClassA
{
};

class ClassB
{
};
```

5) **大括号{和}总是独占一行**，如：

```
while (true)
{
}
```

6) **每个变量声明和定义独占一行，不在同一行声明或定义多个变量**，如：int i,j;是禁止的。
7) **模板标识志符总是独占一行**，如：

```
template <typename DataType>
void template_function();
```

9. 空格规范

下列情况不保留任何空格：

1) **语句结束符分号“;”前不能有空格**，如：int i;
2) **函数名后必须紧跟左小括号“(”**，如：hello()
3) **空的 for 语句内部不含任何空格**，如：for (;;)
4) **for 语句中的赋值和比较操作符前后**，如：for (int i=0; i<10; ++i)

5) **模板<后和>前**，如：vector<int>
6) **作用域“::”前后**，如：std::vector<int>
7) **指针符*和类型间**，如：int* p，注意*是紧跟着类型，而不是变量名

下列情况下必须保留且仅保留一个空格：
1) **if/for/while 之后**，如：if (true)、for (;;)、while (true)
2) **非 for 语句内的赋值和比较操作符前后**，如：int i = 0、if (0 == i)
3) **参数分隔符逗号后紧跟一个空格**，如：fopen(filename, "r")

10. 比较规范

1) 如果是和常量比较相等“==”，则常量必须放在“==”前，非常量放在“==”后，如：if (NULL == thread)
这样做，是为防止犯 if (thread = NULL)类错误。
2) 如果是和常量比较不相等“!=”、“>”和“<”等，则常量放在比较操作符后面，如：if (thread != NULL)
这样做是为了避免头重脚轻，影响代码的协调性。

11. 名字空间 namespace 规范

1) **禁止在头文件中以 using 方式引入名字空间**，而应当直接使用名字空间名方式，如：std::vector<int>
2) **禁止在全局作用域名以 using 方式引入名字空间**；
3) **不直接使用 namespace mooon {和}**，而是替代使用 MOOON_NAMESPACE_BEGIN 和 MOOON_NAMESPACE_END；
4) 在头文件中，NAMESPACE_BEGIN 总是紧跟在#include 之后，而 NAMESPACE_END 总是在#endif 之前，两者间不包含空行，如：

```
#ifdef SYS_UTIL_H
#define SYS_UTIL_H
#include "sys/sys_config.h"
SYS_NAMESPACE_BEGIN

class CSysUtil
{
};

SYS_NAMESPACE_END
#endif // SYS_UTIL_H
```

12. #include 规范

1) #include <>放在#include ""之前；
2) #inlcude 应当包含对应的短目录名，如：#include <util/util_config.h>；
3) 不#include 未使用到的头文件；
4) 不直接和间接地重复#include 相同的头文件；
5) 同类间的#include，按字母顺序，如：

```
#include <assert.h>
#include <stdio.h>
#include <unistd.h>
```

13. 类向前声明规范

1) **可以不使用类向前声明的，杜绝使用，而应当直接使用#include 将类引用进来；**
2) **总是在次类中对主类使用向前声明，而在主类中直接#include 次类**，如：

```
#include "b.h" // 直接 include 头文件
class ClassA // ClassA 是主类
{
private:
    ClassB* _b;
};

class ClassA; // 使用向前声明
class ClassB // ClassB 是次类
{
private:
    ClassA* _a;
};
```

14. 分组规范

1) **#include 按<>和""分组，两者之间不能交错**，如：

```
#include <stdio.h>
#include <stdlib.h>
#include "net/net_util.h"
#include "net/tcp_client.h"
#include "net/data_channel.h"
```

如果#include 很多，则#include 间还可以采用空行进行二级分组，以让#include 显得更为清楚。

2) **利用 public/protected/private 对类成员进行分组**，如：

```
class CMyClass
{
public:
    CMyClass();

public: // virtual
    virtual void foo();

private: // 分组一定义区间
    int _a;
    int _b;

private: // 分组二定义区间
    ClassA* _a;
    ClassB* _b;
};
```

3) **使用空行对函数类定义的变量进行分组**，如：

```
// 分组一定义区间
int a;
int b;

// 分组二定义区间
ClassA* a;
ClassB* b;
```

15. 对齐规范

1) **#include 按照从短到长的方式对齐**，如：

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
```

2) **类成员变量、全局变量和普通变量的定义和声明按照从短到长的方式对齐**，如：

```
int a;
int bc;
char* str;
```

3) **参数过长时，按参数分隔符逗号对齐**，如：

```
DISPATCHER_LOG_ERROR("Sender %d:%s:%d got unknown events %d.\n"
    , _route_id, get_peer_ip().to_string().c_str(), get_peer_port()
    , events);
```

4) **单个字符串过长时，按"对齐**，如：

```
if (sscanf(line_p
                    ,"%s"
                     "%lu%lu%lu%lu%lu"
                     "%lu%lu%lu%lu%lu"
                     "%lu%lu%lu%lu%lu"
                     "%lu"
        /** 01 */ , net_traffic.interface_name
        /** 02 */ ,&net_traffic.receive_bytes
        /** 03 */ ,&net_traffic.receive_packets
        /** 04 */ ,&net_traffic.receive_errors
        /** 05 */ ,&net_traffic.receive_dropped
        /** 06 */ ,&net_traffic.receive_fifo_errors
        /** 07 */ ,&net_traffic.receive_frame
        /** 08 */ ,&net_traffic.receive_compressed
        /** 09 */ ,&net_traffic.receive_multicast
        /** 10 */ ,&net_traffic.transmit_bytes
        /** 11 */ ,&net_traffic.transmit_packets
        /** 12 */ ,&net_traffic.transmit_errors
        /** 13 */ ,&net_traffic.transmit_dropped
        /** 14 */ ,&net_traffic.transmit_fifo_errors
        /** 15 */ ,&net_traffic.transmit_collisions
        /** 16 */ ,&net_traffic.transmit_carrier
        /** 17 */ ,&net_traffic.transmit_compressed
                    ) != filed_number)
```

16. 第三方库规范

1) 严格禁止直接使用第三方库的源代码，而应当总是只使用它的头文件和库文件；
2) 严格禁止在核心功能上使用第三方库，除非凭自己能力写不出或需要花费的时间不能接受时，方可在核心功能上使用第三方库；
3) 使用第三方库时，应当考虑可替换性，即在不改变现有实现的情况下，可以将更换成其它的第三方库，或自己实现第三方库的替代品；
4) 在非关键路径优先考虑开源的第三方库，但当自己实现相同功能的工作量较小时则仍不使用第三方库。

17. 头重脚轻规范

如不影响性能，轻块的代码段放在重块的代码段前，如：

```
if (x >= y)
{
```

```
    // 这里是轻块代码段，行数比重块代码段明显少
    x = y * 2;
}
else
{
    // 这里是重块代码段，行数明显比轻块代码段多
    y = x + y;
    x = y * 2;
    foo(x, y);
}
```

18. 类成员组合和聚合规范

可以组合的，不使用聚合，如：

```
class ClassA
{
private:
    ClassB _b; // 不使用 ClassB* _b;
};
```

这样做的好处有两个：一个减少内存碎片，二是连续内存区别更方便调试和定位问题。

19. 宏规范

1) 尽量减少宏的使用，优先使用枚举和内联函数替代；
2) **单行宏体，使用小括号包含起来**，除非语法上不允许小括号包含起来，如：

```
#define INT8_MIN    (-128)
```

而下面不能使用小括号，否则会产生语法错误：

```
#define MOOON_NAMESPACE_BEGIN namespace mooon {
```

3) **多行宏体，使用 do { ... } while(false)包含起来**，除非语法上不允许；
4) **如果在 if/for/while 块内使用宏，则 if/for/while 块必须使用{}包含起来**，即使只有宏一行代码，如：

```
if (!choose_center(center_ip, center_port))
{
    MYLOG_ERROR("Not found valid center.\n");
}
```

上面的写法不能为：

```
if (!choose_center(center_ip, center_port))
    MYLOG_ERROR("Not found valid center.\n");
```

原因是一旦宏体为多行代码，就会造成错误。