[quote]

[align=center][b][color=orangered][size=200%]' 免 责 声 明 '[/size][/color][/b][/align]

======

&#8195; [b][size=150%]该[/size]教材的出版时间是2009年，由于游戏与代码的迭代特性，本书的内容已经与现在你实际接触到的游戏与代码内容有所差异，有些内容可能已经过时，甚至是被淘汰“过期的”，请大家自己斟酌参考。若出现由于原文与本帖翻译有所不同而导致的错误或损失，概不负责。[/b]

[/quote]

[quote]

[align=center][b][color=orangered][size=200%]' 扇 贝 碎 碎 念 '[/size][/color][/b][/align]

======

[del]本来该帖子应该是发布在插件区的插件开发板块，但我在申请声望的帖子中等待了两天，发现版主[b][iomect][/b]在[b][tid=19010187][公告] WOW插件开发者招募[/tid][/b]中已经消失好久了，就干脆直接在主板块发布了。如果有其他版主看到，可以帮忙发点声望并把该帖移到插件开发板块。我也方便发布新的翻译内容到插件开发区。[/del]

[b][size=150%]这[/size][/b]是我翻译《Beginning Lua with World of Warcraft Addons》的第一个章节，但并不是书本目录的第一章。因为这章是我比较感兴趣的一章，我想先翻译我想了解的内容。全书共14个章节和2个附录，其他章节的翻译内容我会在以后陆续发帖分享。[b]也希望阅读的人不要吝啬你们的点赞&#128077;，只要点一下便能给予我莫大的支持和鼓励。[/b]我已经出了到目前为止所翻译的几个章节的合集，有兴趣的可以自行浏览[tid=24502396][color=orangered][合集]Lua与魔兽世界插件——《Beginning Lua with World of Warcraft Addons》[/color][/tid]

该教材的出版时间是2009年，资料片内容是巫妖王之怒。如果在那段时间玩过游戏的玩家可能会有所疑惑，巫妖王之怒不是2010年才上线吗？其实美服(外服)早在2008年11月13日上线了巫妖王之怒的资料片，但由于当时代理权续约和变更问题，想上新资料片得等到次年6月份，也就是2009年的6月份。后来续约问题僵持不下，代理权变更。现在，我们也知道了结果。虽然资料片最终上线了，但也已经迟到了近2年的时间。最终于2010年8月31日与大家见面。[b]本书已经与即将到来的9.0已经相隔11年之久，跨越了7个大版本之多，大部分接口已经过时，是极不推荐作为新版本插件开发的。[/b]就算是作为怀旧服的插件指南，其中的有些接口或方法也是“超前的”。只能当作入门或业余读物，先当作一个简单的、整体的了解。该教材虽然过时，但这依然是一本相当优秀的插件开发学习指南。

我虽然是软件科班出身，但工作经验并不丰富，翻译上难免有些出入。信达雅无法全面顾及，只能先保证按原意翻译。有些比较在意地方，我虽然做了翻译但依然会放上英文原版用词，避免误人子弟。这是初次翻译和初次排版的第一个章节，内容和排版比较杂乱，有错误的地方请多多包含。

[b]如果有人在阅读过程中，如果发现哪里翻译错误，或错别字需要更正，请发评论或发私信告知，谢谢大家。[/b]

接下来让我们正式开始这次的阅读吧。

[/quote]

[quote]

[align=center][b][color=orangered][size=200%]' 目 录 '[/size][/color][/b][/align]

======

[list]

[\*][size=180%][b]第九章 使用战斗日志建立冷却监视器(Using the Combat Log to Build a Cooldown Monitor)[/b][/size]

[list]

[\*][b][size=130%]使用战斗日志事件(Working with Combat Log Events)[/size][/b]

[list]

[\*][b]战斗日志参数(Combat Log Arguments)[/b]

[\*][b]创建一个事件处理器(Building an Event Handler)[/b]

[/list]

[\*][b][size=130%]使用GUID和单位(Working with GUIDs and Unit)[/size][/b]

[list]

[\*][b]NPC和载具GUIDs(NPC and Vehicle GUIDs)[/b]

[\*][b]玩家和宠物的GUID(Player and Pet GUIDs)[/b]

[\*][b]GUID的例子(GUID Examples)[/b]

[\*][b]单位ID(Unit IDs)[/b]

[/list]

[\*][b][size=130%]单位标记和法术类型(Unit Flags and Spell Schools)[/size][/b]

[list]

[\*][b]位域(Bit fields)[/b]

[list]

[\*]阅读比特位(Reading bit)

[\*]设置比特位(Setting Bits)

[\*]重置比特位(Restting Bits)

[\*]切换比特位(Toggling Bits)

[\*]位域限制(Limits of Bit Fields)

[\*]位域和布尔值(Bit Fields vs. Booleans)

[/list]

[\*][b]法术类型(Spell Schools)[/b]

[\*][b]单位标志位(Unit Flags)[/b]

[/list]

[\*][b][size=130%]建立一个冷却监视器(Building a Cooldown Monitor)[/size][/b]

[list]

[\*][b]检测法术(Detecting Spells)[/b]

[\*][b]在聊天框中使用转义序列(Using Escape Sequences in Chat Frame)[/b]

[list]

[\*]彩色文本(Colored Texts)

[\*]超链接(Hyperlink)

[\*]纹理(Texture)

[\*]语法转义序列(Grammatical Escape Sequence)

[/list]

[\*][b]建立状态栏计时器(Building Status Bar Timer)[/b]

[list]

[\*]建立一个模板(Building a Templat)

[\*]处理定时器(Handing the Timers)

[\*]状态栏计时器的构造(The Status Bar Timer Constructor)

[\*]定位(Positioning)

[\*]更新计时器(Updating Timers)

[\*]取消计时器(Canceling Timers)

[\*]移动计时器(Moving the Timers)

[\*]修复上一个错误(Fixing the Last Bug)

[/list]

[/list]

[\*][b][size=130%]总结(Summary)[/size][/b]

[/list]

[/list]

[/quote]

======

[align=center][b][color=orangered][size=200%]&#9632; 第九章 使用战斗日志建立冷却监视器[/size]

[size=130%](Using the Combat Log to Build a Cooldown Monitor)[/size][/color][/b][/align]

[b][size=150%]我[/size][/b]们已经使用了很多的事件，但有两个特别有趣的事件我们还没有提及：战斗日志事件(COMBAT\_LOG\_EVENTS)和未过滤的战斗日志事件(COMBAT\_LOG\_EVENT\_UNFILTERED)。两种事件都与战斗日志有关，并且能接收大量的参数(arguments)(高达20个)。我们将在本章中看到这些参数的意思。所有与战斗相关的事件都会触发战斗日志事件，例如当你附近的任何人攻击一个目标或任何人被攻击。这些事件参数会告诉你到底发生了什么。

在本章中，我们将写一个被称为冷却监视器(Cooldown Monitor)的插件，这是一个显示来自你团队成员的重要技能冷却的插件，例如，英勇、嗜血或重生。这意味着当你的团队中有人施放了这样的法术时，我们必须找到相应的战斗日志事件类型。

======

[color=blue][b][size=160%]&#9675; 使 用 战 斗 日 志 事 件[/size]

[size=130%](Working with Combat Log Events)[/size][/b][/color]

我提到过两个战斗日志事件，分别是：战斗日志事件(COMBAT\_LOG\_EVENTS)和未过滤的战斗日志事件(COMBAT\_LOG\_EVENT\_UNFILTERED)，两者接收相同的参数。唯一的区别是所有战斗日志事件都会触发未过滤事件。战斗日志事件使用过滤器过滤某些事件。这个过滤器是你的战斗日志设置，并且每个战斗日志事件都被显示在你聊天框中的战斗日志里。修改战斗日志的聊天框设置会更改此过滤器。这也意味着你的大多数时间都是在使用事件的未过滤版本。在你的插件中，你不会希望让你的事件使用用户定义的过滤器。被过滤的版本只有在你想用你的插件去替代默认的战斗日志聊天框的时候才有用。

[size=130%][b]&#9675; 战 斗 日 志 参 数(Combat Log Arguments)[/b][/size]

对于所有类型的战斗日志事件来说，一个战斗日志事件的前八个参数总是相同的。表9-1列出了它们。

[align=center][size=80%]表9-1 前八个战斗日志参数[/size][/align]

[table]

[tr]

[td50][align=center]参数[/align][/td]

[td50][align=center]描述[/align][/td]

[/tr]

[tr]

[td][align=center]时间戳(timestamp)[/align][/td]

[td][align=center]事件发生的确切时间的时间戳。它能被用在例如函数date(fmt，time)的第二个参数。[/align][/td]

[/tr]

[tr]

[td][align=center]事件(event)[/align][/td]

[td][align=center]子事件(The subevent)[/align][/td]

[/tr]

[tr]

[td][align=center]源GUID(sourceGUID)[/align][/td]

[td][align=center]生成事件的实体的GUID(globally unique identifier，全局唯一标识符)。我们将在后面介绍如何从GUID中获取信息。[/align][/td]

[/tr]

[tr]

[td][align=center]源名称(sourceName)[/align][/td]

[td][align=center]发生该事件的实体的名称。例如，如果你施放了一个法术，则该参数就是你的名称。[/align][/td]

[/tr]

[tr]

[td][align=center]源标志位(sourceFlags)[/align][/td]

[td][align=center]该标志位是包含了生成此事件的实体的额外信息。这是一个位域(bit field)，它在第六章中被提到过，之后我们将看到如何从这个位域提取额外信息。[/align][/td]

[/tr]

[tr]

[td][align=center]目标GUID(destGUID)[/align][/td]

[td][align=center]目标的GUID[/align][/td]

[/tr]

[tr]

[td][align=center]目标名称(destName)[/align][/td]

[td][align=center]目标的名称[/align][/td]

[/tr]

[tr]

[td][align=center]目标标志位(destFlags)[/align][/td]

[td][align=center]目标的标志位[/align][/td]

[/tr]

[/table]

最有趣的参数是第二个。不要把这个和真实的战斗日志事件混淆了。这个事件参数被充当识别战斗日志动作的子事件。例如，如果你施放了一个法术，事件(event)将是SPELL\_CAST\_START；如果你完成了施法，它将是SPELL\_CAST\_SUCCESS.如果该法术击中了你的目标，事件是SPELL\_DAMAGE。你可以在附录B中找到所有战斗日志事件的子事件。

这个子事件也决定了所有事件处理程序接收的额外参数。如果子事件是以SPELL或RANGE开头，像前面举例的事件一样，则接下去的三个参数将会像表9-2所示那样。

[align=center][size=80%]表9-2 SPELL和RANGE事件的额外参数[/size][/align]

[table]

[tr]

[td50][align=center]参数[/align][/td]

[td50][align=center]描述[/align][/td]

[/tr]

[tr]

[td][align=center]法术ID(spellID)[/align][/td]

[td][align=center]法术的ID。它的ID与许多魔兽世界数据库网站所使用的ID相同。例如Wowhead。你可以使用以下方法构建一个URL，例如从以下网站搜索关于法术ID的详细信息：[url]http://www.wowhead.com/?spell=<spellID>[/url][/align][/td]

[/tr]

[tr]

[td][align=center]法术名称(spellName)[/align][/td]

[td][align=center]法术的名称[/align][/td]

[/tr]

[tr]

[td][align=center]法术类型(spellSchool)[/align][/td]

[td][align=center]该法术的类型。这是一个位域(bit field)，我们将在稍后详细讨论这个。[/align][/td]

[/tr]

[/table]

如果这个事件以ENVIRONMENTAL开头(例如ENVIRONMENTAL\_DAMAGE)，则下一个参数是环境类型(environmental type)，它是标识环境类型的字符串(例如岩浆“LAVA”)。

表9-3中所示的参数取决于战斗日志事件的后缀。这些后缀不会再添加额外的参数，例如在前面例子中的SPELL\_CAST\_START和SPELL\_CAST\_SUCCESS。这两个事件接收八个标准参数和额外三个法术参数。这些参数提供了所有你需要的信息：谁对谁施放了何种法术。但这个事件SPELL\_DAMAGE明显需要更多的参数，因为你还需要知道这个法术到底造成了多少的伤害，它是不是爆击(critical hit)，等等。表9-3展示了所有例如SPELL\_DAMAGE或SWING\_DAMAGE(一个白色字体的近战攻击)以DAMAGE结尾的事件的额外参数。

[align=center][size=80%]表9-3 以DAMAGE为后缀的事件的额外参数[/size][/align]

[table]

[tr]

[td50][align=center]参数[/align][/td]

[td][align=center]描述[/align][/td]

[/tr]

[tr]

[td][align=center]数值(amount)[/align][/td]

[td][align=center]攻击造成的伤害[/align][/td]

[/tr]

[tr]

[td][align=center]溢出伤害(overkill)[/align][/td]

[td][align=center]如果目标死于攻击，则为超出部分的伤害[/align][/td]

[/tr]

[tr]

[td][align=center]类型(school)[/align][/td]

[td][align=center]伤害的类型。这是一个位域，就像法术的类型。我们将在后面学习如何从中获取信息。[/align][/td]

[/tr]

[tr]

[td][align=center]被反击(resisted)[/align][/td]

[td][align=center]伤害被反击的数值[/align][/td]

[/tr]

[tr]

[td][align=center]被格挡(blocked)[/align][/td]

[td][align=center]伤害被阻挡的数值[/align][/td]

[/tr]

[tr]

[td][align=center]被吸收(absorbed)[/align][/td]

[td][align=center]伤害被吸收的数值[/align][/td]

[/tr]

[tr]

[td][align=center]爆击(critical)[/align][/td]

[td][align=center]如果这次攻击是爆击的则为1，否则为0[/align][/td]

[/tr]

[tr]

[td][align=center]偏斜(glancing)[/align][/td]

[td][align=center]如果是一次偏斜攻击则为1，否则为0[/align][/td]

[/tr]

[tr]

[td][align=center]致命(crushing)[/align][/td]

[td][align=center]如果它是一次致命攻击则为1，否则为0[/align][/td]

[/tr]

[/table]

附录B包含一个列出了所有战斗日志事件参数的表，那里有更多的后缀和更多的参数。

现在，让我们看看这些事件的实际用途。

[size=130%][b]&#9675; 创建一个事件处理器(Building an Event Handler)[/b][/size]

处理子事件基本上与处理一个普通事件没什么不同，所以你只需要使用普通事件处理程序。但有时候把子事件当作真实事件来处理可能会有用。下面的代码展示了执行此操作的事件处理程序。

[code=lua]

MyMod = {}

local function onEvent(self, event, ...)

if event == “COMBAT\_LOG\_EVENT\_UNFILTERED” then

return onEvent(self, select(2, ...), ...)

elseif MyMod[event] then

return MyMod[event](...)

end

end

local frame = CreateFrame(“Frame”)

frame:RegisterEvent(“COMBAT\_LOG\_EVENT\_UNFILTERED”)

frame:SetScript(“onEvent”, onEvent)

[/code]

这个事件处理程序的技巧是，如果事件是战斗日志事件，它只调用它自己。它在尾部的调用使用了子事件替代战斗日志事件。例如，我们现在可以使用下面的函数去打印输出所有SPELL\_CAST\_START子事件到聊天框：

[code=lua]

function MyMod.SPELL\_CAST\_START(...)

print(string.join(“,”, tostringall(...)))

end

[/code]

tostringall是一个由默认UI创建的辅助函数；它的工作原理与tostring类似，但有很多参数，string.join函数常被用于连接所有参数，用逗号隔开。每当你附近的人开始施放一个法术时，它将输出子事件的所有参数到你的聊天框。所以在一座主城或副本里使用该函数时，将发送大量的事件信息到聊天框。下面这个输出例子是在我施放一个快速治疗的时候产生的：

[quote]

12346219351.848，SPELL\_CAST\_START，0x00000000001517FB，Tandanu，1297，0x0000000000000000,nil,-2147483648,48071,Flash Heal,2

[/quote]

如果你曾经读过原始的战斗日志，那么这个输出的格式可能看起来很熟悉。你可以通过输入/combatlog来创建战斗日志；这会将所有COMBAT\_LOG\_EVENT\_UNFILTERED事件转储到\World of Warcraft\Logs\WoWCombatLog.txt文件夹中。此文件中的每一行保存一个事件，使用逗号分隔所有的参数。

第一个参数是当前时间，第二个是子事件，比较有意思的是第三个参数，它是我的GUID。需要注意，这是一个字符串，因为Lua的数字不能表示如此大的数据。一个GUID在服务器上总是唯一的，这意味着没有其他玩家或NPC的GUID与我的相同。注意，即使是两个在同一个生成位置的同类型的怪物(mobs)，也始终具有不同的GUID。这意味着你在战斗日志插件中可以使用这个ID跟踪一个NPC或玩家。例如像DamageMeters这样的伤害统计插件可以使用这个ID来区分同名的两个怪物。

第四个参数是我的名字。第五个参数是存储有关施法者(自己)的额外信息。在下一节中，你将看到我们如何从这个位域中提取信息。接下来三个参数，被称为destGUID，destName和destFlags，它们均指向动作的目标。事件SPELL\_CAST\_START不提供目标信息，因此你不能预测一个法术是对谁施放。这意味着为GUID为0，名字为空(nil)，标志位看起来是一堆毫无用处的数字。当法术击中目标时，这个信息将会被显示出来。

接下来的三个参数是spellId，spellName和spellSchool。最高级的快速治疗(Flash Heal)的ID是48071，你可以通过Wowhead来查找法术([url]http://www.wowhead.com/?spell=48071)[/url]。这个名字明显是快速治疗，伤害类型(school)是2，这表示这个法术是神圣法术(holy spell)；我们将在本章后面看到关于spellSchool的其他值。

======

[color=blue][b][size=160%]&#9679; 使 用 GUID 和 单 位[/size]

[size=130%](Working with GUIDs and Unit)[/size][/b][/color]

游戏中的每个单位都有一个唯一的GUID来标识对象。单位(unit)这个词是指诸如怪物、玩家、NPC或载具之类的对象。服务器上的每个单独的NPC或玩家都有一个唯一的GUID。

玩家或宠物的GUID只有在你重命名或转移到其他服务器时才会改变。NPC的GUID被用到死亡；所有重生的NPC们都有会有一个全新的不同的GUID。

GUID总是以十六进制的字符串表示。你不能把它赋值给一个Lua数值。因为这个数字太大了。这个数值对其本身来说包含有具有价值的信息。例如，这里是一个怪物的GUID：

0xF1300070BB00004A

前两位数值“F”和“1”的用途未知(注意这里“0”和“x”不作为数字计算在内，它只是表示这是一个十六进制数)。第三位数值“3”表示该单位的类型，3代表NPC。其他可能的值为：“4”是宠物，“5”是载具，“0”是玩家。所有其他的数值取决于该单位的类型。

[b][size=130%]&#9675; NPC和载具GUIDs(NPC and Vehicle GUIDs)[/size][/b]

接下来的三位数字的用途是未知的，但它们总是0。再接下来的四位数字(在我们的例子中是70BB)包含我们可以能从GUID中提取的最有价值的信息。这部分是NPC或载具的ID。我们可以使用下面的函数来从GUID提取NPC的ID：

[code=lua]

local function GetNPCId(guid)

return tonumber(guid:sub(9,12),16)

end

print(GetNPCId(“0xF1300070BB00004A”)) -->28859

[/code]

这里我们使用tonumber(str， base)的第二个参数，这可以用来告诉tonumber给定字符串在底层系统中的位数。需要注意的是，对于前缀为0x的十六进制数字来说这不是必须的。

在例子中，我们调用函数来提取GUID，并且得到的结果是28859。这个生物的ID就像法术ID一样，也被几乎所有的魔兽世界数据库网站所使用。我们可以通过打开URL使用Wowhead去获取关于NPC的名字和额外信息([url]http://www.wowhead.com/?npc=28859[/url])。这网页告诉我们例子中的GUID取自玛里苟斯。

我们也可以查找名字，它也是作为一个参数进行传递的，即使这样看起来有点毫无意义。这样做的真正目的在于，NPC们不会取决于本地。如果你编写了一个查找某些NPC的ID的插件，那么它将在任何客户端语言下运行。在本地化魔兽世界客户端怪物的名字肯定会有所不同。这同样适用于法术ID。

[quote]

小贴士：使用ID来识别一个生物或法术，或使用sourceName，destName和spellName去获取输出它的本地名称。

[/quote]

剩下的6位数字是一个“衍生计数器”(spawn-counter)，为每个相同类型的新怪物增加1，去保证GUID的唯一性。当服务器重启的时候，这个计数器被重置，所以NPC(非玩家角色)和载具的GUID只有当周是自己的。

[b][size=130%]&#9675; 玩家和宠物的GUID(Player and Pet GUIDs)[/size][/b]

整个玩家的GUID只是一个计数器，每当有人创建一个新角色时就会增加。这意味着你可以通过GUID来比较两个角色的创建时间。注意，服务器转移或改名也会给你一个新的GUID。

宠物和玩家非常相似，它们也有一个计数器，每当有人驯服了一只新宠物时，这个计数器就会增加。这个计数器是以七位数的形式存储，从第四位开始，后面六位是这个特定的宠物衍生计数器，每次你重新召唤你的宠物时它就会增加。

[b][size=130%]&#9675; GUID的例子(GUID Examples)[/size][/b]

一个一直使用GUID的插件是Deadly Boss Mods，它不依赖于客户端语言。下面这段代码稍微展示了一部分DBM的UNIT\_DIED处理程序的简化版本。此程序的目的是检测你对抗的boss的死亡。每当一名玩家或NPC死亡，该程序就会被调用；destGUID是死亡单位的GUID。该方法检测它是否是一个NPC或载具，并且提取NPC的ID，传递给OnMobKill方法：

[code=lua]

function DBM:UNIT\_DIED(\_,\_,\_,\_,\_, destGUID)

if destGUID:sub(5, 5) == ”3” or destGUID:sub(5, 5) == “5”then

self:OnMobKill(tonumber(destGUID:sub(9, 12),16))

end

end

[/code]

OnMobKill做些繁杂困难的工作。它使用一个循环来遍历所有当前战斗中的boss模块(处理boss战的DBM小模块)。这些模块(mods)被存储在局部变量inCombat中。例如一个boss模块对象包含了文件combatInfo，这是一个表，包含了所有关于如何去检测拉怪(pull)并击杀boss的所有信息。这个mob文件在表中包含了boss的NPC ID，它也被用于拉怪检测(pull detection)。下面的代码段展示了一个关于OnMobKill处理程序的稍微简化的版本。完整的方法有很多额外的代码，boss有多个怪物需要被杀死。

[code=lua]

function DBM：OnMobkill(cId, synced)

for i = #inCombat, 1,-1, do

local v inCombat[i]

if cId == v.combatInfo.mob then

if not synced then

sendSync(“DBMv4-Kill”, cId)

end

v:EndCombat()

end

end

end

[/code]

该功能还将此事件同步到副本组(raid group)，以告诉玩家们谁超出了射程，或战斗已经结束。该同步会再次调用OnMobKill，第二个参数synced = true用来防止同步循环。

代码调用给定NPC ID的所有boss模块的EndCombat方法。这个方法打印输出胜利消息到聊天框，结束所有计时器，并保存一些统计数据。

GUID的另一个重要用途是非常明显的：像识别码一样使用它们。毕竟它们是全局唯一的标识符。这种用法的一个例子是玛里苟斯的Boss模块。在战斗的第三阶段，所有玩家都使用了一种载具(一条龙)，并在载具身上施放类似电涌(Power Surge)的法术，而不是在玩家身上。然而，一个Boss模块仍然需要去确定玛里苟斯的法术目标。但它得到的唯一信息是这个受影响载具的名称和GUID。通过使用名字取获到龙的主人是不可能的，因为所有的龙都具有相同的名字。为此我们需要去使用GUID。

这个点子是使用GUID作为表中的键值，并存储载具所有者的名称作为值。下面代码展示了函数buildGuidTable()，在第三阶段被调用几秒后。它使用函数GetNumRaidMembers()，返回在副本中的玩家数量。它建立了一个识别副本成员和它们的宠物的字符串。在下一节中，我们将看到更多关于单位ID的内容。UnitGUID常被用于提取单位的GUID，而UnitName被用于提取名字。

[code=lua]

local guids = {}

local function buildGuidTable()

for i = 1, GetNumRaidMembers() do

guids[UnitGUID(“raid”..i..”pet”) or “”] = UnitName(“raid”..i)

end

end

[/code]

这个函数创建了表guids，允许我们去使用龙的GUID取获取它的拥有者。下面的例子展示了一个代码段，它负责在玛里苟斯对你施放一个电涌(Power Surge)时，显示一个巨大的警告文本。specWarnSurge是一个警告对象，它的方法Show()显示了这个警告。

[code=lua]

local target = guids[destGUID]

if target == UnitNme(“player”) then

specWarnSurge:Show()

end

[/code]

这个例子从大量的单位ID中循环建立了表guids。让我们看看它们是如何工作的。

[b][size=130%]&#9675; 单位ID(Unit IDs)[/size][/b]

单位ID是定义了一个例如玩家或NPC的短小的字符串。我们只是看到了一些复杂的单位ID，例如raid1target。但我们在这本书的前面提到了一个更简单的单位ID：player。player总是指你，它可以像所有的单位ID一样，传递给API函数，例如UnitName。有许多这样的函数可以获得一个单位的各种属性，比如它的当前法力值(UnitMana)，生命值(UnitHealth)，或GUID(UnitGUID)。附录B包含所有单位相关函数的引用。

单位ID总是由一个前缀组成，后面可以跟着很多个后缀。表9-4列出了单位ID的可能前缀。

[align=center][size=80%]9-4 单位ID前缀[/size][/align]

[table]

[tr]

[td50]前缀[/td]

[td]描述[/td]

[/tr]

[tr]

[td][align=center]焦点(focus)[/align][/td]

[td][align=center]你的焦点目标[/align][/td]

[/tr]

[tr]

[td][align=center]玩家(player)[/align][/td]

[td][align=center]你[/align][/td]

[/tr]

[tr]

[td][align=center]宠物(pet)[/align][/td]

[td][align=center]你的宠物[/align][/td]

[/tr]

[tr]

[td][align=center]队伍(partyn)[/align][/td]

[td][align=center]第n个队伍成员。注意，你不是你自己队伍的成员，意味着你不能使用单位ID引用你自己，n介于0到5之间。[/align][/td]

[/tr]

[tr]

[td][align=center]团队(raidn)[/align][/td]

[td][align=center]第几个团队成员。对你来说不像队伍partyn那样。[/align][/td]

[/tr]

[tr]

[td][align=center]目标(target)[/align][/td]

[td][align=center]你的当前目标[/align][/td]

[/tr]

[tr]

[td][align=center]鼠标目标(mouseover)[/align][/td]

[td][align=center]你当前鼠标指向单位[/align][/td]

[/tr]

[tr]

[td][align=center]空(none)[/align][/td]

[td][align=center]未指向单位[/align][/td]

[/tr]

[tr]

[td][align=center]NPC(npc)[/align][/td]

[td][align=center]你当前正在打交道的NPC。打交道意味着你有一个打开的窗口，例如飞行点地图或NPC任务对话框。[/align][/td]

[/tr]

[/table]

这个前缀后面可以跟着后缀。只有两个可用的后缀：target和pet。前者总是指着被前缀所标识的目标单位。后者为他的宠物。一个单位ID的例子是raid1targetpettarget，它指向第一个团队成员的目标的宠物的目标。第一个副本成员是组建团队的玩家，第二个玩家是第一个加入的玩家，以此类推。下面循环遍历所有副本成员并打印输出它们的名字和当前目标。它是从前面构建玛里苟斯检查表的例子中一个非常简单的循环遍历。

[code=lua]

for i=1, GetNumRaidMembers() do

print(UnitName(“raid”...i), UnitName(“raid”)...i...”target”))

end

[/code]

注意在永远不会有一个副本单位ID上有空缺(hole)，而且没有一个玩家将得到一个单位ID超过GetNumRaidMembers()，因为当有一名玩家离开团队组时，副本ID会下移。

其他战斗日志事件的重要参数是单位标志位(the unit flags)和法术类型(school)，我们已经看过了法术类型2(school2)，它告诉我们那是一个神圣法术。但这个数值的意思是什么呢？

======

[color=blue][b][size=160%]&#9679; 单 位 标 记 和 法 术类 型[/size]

[size=130%](Unit Flags and Spell Schools)[/size][/b][/color]

单位标志位和法术类型是位域，我在第六章提到过一些比特库(bit library)。现在让我们仔细看看这些位域。一个位域基本上就是个保存一系列布尔值的数字。我们不得不看一下一个数字的二进制表示去理解它是如何保存布尔值的。

[b][size=130%]&#9675; 位域(Bit fields)[/size][/b]

以数字20为例子。它的二进制表示是10100。每个二进制位都可以是“1”或者“0”，而“1”对应true，“0”对应false。这意味我们在带有n位的二进制数中，最多可以存储n位布尔值。这些布尔值也被称为flags(标志位)。例子中20的值存储了5个布尔值。我们将最右边的比特位称为第0位，在它左边的位称为第1位，以此类推。这意味着我们在例子中的数字20的第0位布尔值所存储的意思为false。第1位也是false，第3位为true，第4位为false，第5位是true。

如你所见，一个相对较小的数字能存储大量的布尔值，所以使用位域是为了节省空间。在本节最后的单位标志位(unit flags)，我们将看见只需一个数字就能存储25位布尔值的位域。想象这所有的25位布尔值作为额外参数传递给战斗日志事件。这当然是可能的，但使用代码会变得非常痛苦。

[b]· 阅读比特位(Reading bit)[/b]

我们现在将使用比特库(bit library)来提取这些信息。如果你想在一个普通的Lua解释器中测试下面这个例子，请使用下面代码去装载这个函数库。

[code=lua]

require(“bit”)

[/code]

这在魔兽世界中并不是必要的，因为这个函数库已经被装载到了里面。require()函数在游戏中甚至不是可用的。

逐位操作(The bitwise operation)是需要去对读取一个特殊二进制比特操作AND，它可以在函数bit.band中运行。表9-5列出了该操作的真值表(truth table)，它展示了所有可能的操作组合结果。你可以通过在左列中查找第一个操作数并在上一行中查找第二个操作数来读取表，结果然后在此行和列中。

[align=center][size=80%]表9-5 二进制AND的真值表[/size][/align]

[table]

[tr]

[td20][align=center]AND[/align][/td]

[td][align=center]1[/align][/td]

[td][align=center]0[/align][/td]

[/tr]

[tr]

[td][align=center]1[/align][/td]

[td][align=center]1[/align][/td]

[td][align=center]0[/align][/td]

[/tr]

[tr]

[td][align=center]0[/align][/td]

[td][align=center]0[/align][/td]

[td][align=center]0[/align][/td]

[/tr]

[/table]

回想一下这些按位操作是如何工作的。它们使用二进制AND来对两个数的每个比特位进行操作，从最右边的比特位开始。根据表格中的结果中设置相应的比特。这意味着只有当两个比特位都为“1”时，它才为1；其他情况都为0。

例如，如果我们想去检查在前面例子中的第4个比特位(请记住，从最右边的最低有效位开始以0计数)是否被置1，则我们可以使用二进制数10000(十进制的16)作为第二个操作数给二进制AND。第二个操作数也被称为掩码(bit mask)，并且运算的结果将清除我们在测试的数字中所有被设置的比特位，除了已经被置1的比特位。在下面的代码中，20是我们的位域，16是我们的掩码。

[code=lua]

print(bit.band(20, 16))

[/code]

它打印输出了16，因为10100 band 10000等于10000。如果第4个比特位没被置1，该函数返回0，例如，在数字15(二进制位01111)。在给定位域下，你可以使用下面表达式去检查第n位比特位是否在被设置。

[code=lua]

local isBitSet = bit.band(bf, 2^n) ~= 0

[/code]

用要检查的第几个比特位替换n。别忘记0是最不重要的比特。如果该比特位被置1，则表达式的值位true，其他情况则为0。

注意，表达式 2^n总是计算成一个二进制表达式100...00来表示。另外可能的方法，是使用函数bit.lshift(bitfield，n)去创建一个掩码。这是左移操作，它将位域中所有的位左移n位。新数字的右边将由0填充。下面的代码依然可以检查第n位是否被设置。

[code=lua]

local isBitSet = bit.band(20, bit.lshift(1, n)) ~= 0

[/code]

[quote]

小贴士：熟悉十六进制数和位域的人通常直接使用创建十六进制数来创建位域。在十六进制中一个数经常用四位来表示，这使得从二进制与十六进制之间的转化变得非常容易。在位域中最重要的十六进制数是0，2，4和8。他们的二进制位表示为0000，0010，0100和1000。

[/quote]

也可以使用具有多个设置位的掩码。如果未设置任何比特位，则此类掩码将返回0。这意味着它可用于检查一组比特位中至少由一位是否被设置。

[b]· 设置比特位(Setting Bits)[/b]

你也需要在位域中去存储信息。我们需要进行逐位OR操作。表9-6展示了二元操作OR的真值表。如果两者其中之一是1，则它返回1，如果两者都是0，则返回0。

[align=center][size=80%]表9-6 二元操作OR的真值表[/size][/align]

[table]

[tr]

[td20][align=center]OR[/align][/td]

[td][align=center]1[/align][/td]

[td][align=center]0[/align][/td]

[/tr]

[tr]

[td][align=center]1[/align][/td]

[td][align=center]1[/align][/td]

[td][align=center]1[/align][/td]

[/tr]

[tr]

[td][align=center]0[/align][/td]

[td][align=center]1[/align][/td]

[td][align=center]0[/align][/td]

[/tr]

[/table]

这个被用来设置的掩码与被用来读取比特位的掩码相同。着意味着我们可以使用下列代码去设置字段中的第三个比特位设置为20(10100)。

[code=lua]

local field = 20 --10100

local mask = 2^3 --01000

print(bit.band(field, mask) ~= 0) --> false, bit 3 not set

field = bit.bor(field, mask) -- set bit 3

print(bit.band(field, mask) ~= 0) --> true,bit 3 is now set

print(field) --> 28 (binary:11100)

[/code]

在此例子中，逐位OR操作成功的设置了第三位比特。注意，一个真实的插件你可能想在一个变量中存储掩码。所以，你可以让变量在读和设置操作中带有一个具有意义的名字。在下一章中，你将看到真正的插件是如何处理位域的。

我们现在可以在一个位域中，将比特位设为1。但我们如何设置一个比特位为0呢？

[b]· 重置比特位(Restting Bits)[/b]

重置比特我们还需要逐位AND。但是这里需要的掩码在我们要重置的位的位置还需要为0，并且所有的其他比特位需要是1。因为我们不想去改变任何其他的比特位。我们需要另一种操作来从原始掩码创建新的掩码：逐位NOT。

二进制操作NOT是个一元操作，意味着它只需要一个操作数。如果一个操作数是1则它返回0，如果是0则返回1。也就是说，它反转了比特位。在一个位域中逐位NOT，反转了比特位。所以我们可以在原掩码中使用函数bit.bnot(bitfield)。该函数被用于读取或设置比特，去获取能被重置比特的掩码。

下面的代码显示了如何使用这些操作来重置我们代码中的第4位，示例位域位20(10100)：

[code=lua]

local field = 20 -- 10100

local mask = 2^4 10000

print(bit.band(field, mask) ~= 0) --> true, bit 4 is set

field = bit.band(field, bit.bnot(mask)) -- reset bit 4

print(bit.band(field, mask) ~= 0) --> false, the bit is no longer set

print(field) --> 4(binary 00100)

[/code]

代码按预期操作。所以我们现在可以在位域中读取比特位，设置比特位和重置比特位。另一个可以派得上的操作是切换特定的比特位。

[b]· 切换比特位(Toggling Bits)[/b]

我们这里需要的操作是逐位XOR。此操作是异OR(异或)，意味着如果其中一个操作数是1，它就返回1，如果两个操作数都是1或都是0，它就是返回0，表9-7显示二进制异或的真值表。

[align=center][size=80%]表9-7 二进制异或的真值表[/size][/align]

[table]

[tr]

[td20][align=center]XOR[/align][/td]

[td][align=center]1[/align][/td]

[td][align=center]0[/align][/td]

[/tr]

[tr]

[td][align=center]1[/align][/td]

[td][align=center]0[/align][/td]

[td][align=center]1[/align][/td]

[/tr]

[tr]

[td][align=center]0[/align][/td]

[td][align=center]1[/align][/td]

[td][align=center]0[/align][/td]

[/tr]

[/table]

我们可以使用这个操作去切换在带有掩码的位域中的一个比特。该掩码中为0的所有字段保持不变，位1的字段被切换。下面的代码将切换所示位域的第四位，20：

[code=lua]

local field = 20 -- 10100

local mask = 2^4 -- 10000

print(bit.band(field, mask) ~= 0) --> true,bit 4 is set

field = bit.bxor(field, mask) --> toggle bit 4

print(bit.band(field, mask) ~= 0) --> false,the bit is no longer set

field = bit.bxor(field, mask) -- toggle bit 4

print(bit.band(field, mask) ~= 0) --> true, bit 4 is set again

[/code]

我们现在可以对字段执行所有必要的操作来提取信息，或者在其中存储信息。单一个位域中可以包含多少位呢？

[b]· 位域限制(Limits of Bit Fields)[/b]

在带有比特库(bitlib)的Lua中，位域的大小被限制为32位。所有较大的数字将被解释为：-2147483648(二进制：1后面跟着31个0；十六进制：0x80000000)。这也是我们在前面例子中获取的destFlags的值，在那里我们的战斗日志事件还没有一个目标。

[b]· 位域和布尔值(Bit Fields vs. Booleans)[/b]

你应该在只具有真正意义的情况下才使用位域。例如，如果你只有几个布尔值，在大多数情况下最好使用变量。当有很多布尔值(四个或更多)属于一起，并且经常出现在你的代码中时，应该考虑使用字段。

但是请记住，你最重要的目标应该始终是编写可读代码。使用位域的代码通常比使用普通变量的代码更难以理解。你至少应该使用有意义的变量名为你的位掩码(bit masks)命名，而不仅仅是在你代码中使用魔法数字(无任何注释命名修饰的数字)。你将在下一节看到，当使用战斗日志时，暴雪为所有你需要的重要掩码提供了有意义的变量名。

[b][size=130%]&#9675; 法术类型(Spell Schools)[/size][/b]

你在战斗日志事件中看到的位域之一是法术类型。同样的位域也被用于伤害类型，但不一定和法术属于相同类型。例如，这两个值因施法者的魔杖攻击而不同。这里的法术通常是物理法术“射击(Shoot)”，但伤害类型取决于施法者正在使用的魔杖。

前面的例子中展示了一个快速治疗(Flash Heal)的SPELL\_CAST\_START事件，它的法术类型是2(二进制为：10)。所以这个字段中的这个比特位代表着神圣法术。在全局变量中有可以用于针对法术类型的掩码。表9-8中列出了所有这些掩码及它们的值。变量的名称是不言而喻的(self explanator)。

[align=center][size=80%]表9-8 法术和伤害类型的位掩码[/size][/align]

[table]

[tr]

[td50][align=center]变量[/align][/td]

[td][align=center]位掩码[/align][/td]

[/tr]

[tr]

[td][align=center]空类型(SCHOOL\_MASK\_NONE)[/align][/td]

[td][align=center]00000000[/align][/td]

[/tr]

[tr]

[td][align=center]物理类型(SCHOOL\_MASK\_PHYSICAL)[/align][/td]

[td][align=center]00000001[/align][/td]

[/tr]

[tr]

[td][align=center]神圣类型(SCHOOL\_MASK\_HOLY)[/align][/td]

[td][align=center]00000010[/align][/td]

[/tr]

[tr]

[td][align=center]火焰类型(SCHOOL\_MASK\_FIRE)[/align][/td]

[td][align=center]00000100[/align][/td]

[/tr]

[tr]

[td][align=center]自然类型(SCHOOL\_MASK\_NATURE)[/align][/td]

[td][align=center]00001000[/align][/td]

[/tr]

[tr]

[td][align=center]冰霜类型(SCHOOL\_MASK\_FROST)[/align][/td]

[td][align=center]00010000[/align][/td]

[/tr]

[tr]

[td][align=center]暗影类型(SCHOOL\_MASK\_SHADOW)[/align][/td]

[td][align=center]00100000[/align][/td]

[/tr]

[tr]

[td][align=center]魔法类型(SCHOOL\_MASK\_ARCANE)[/align][/td]

[td][align=center]01000000[/align][/td]

[/tr]

[/table]

请注意法术类型并不是相互排斥的；有些法术属于不止一个类型。例如这样一个例子是“冰火球”(Frostfire Bolt)，它既是冰系也是火系，因此它的法术和伤害类型是冰系和火系。当某人使用冰火球时，在战斗记录中出现的位域为：0000010100(十进制位20)。

我们现在可以使用这些掩码和我们的逐位操作(bitwise operations)来创建一个小插件。每当你的目标开始施放一个神圣法术时候，这个插件会在你的聊天框中显示消息。

[code=lua]

MyMod = {}

local function onEvent(self, event, ...)

if event == “COMBAT\_LOG\_EVENT\_UNFILTERED” then

return onEvent(self, select(2, ...), ...)

elseif MyMod[event] then

return MyMod[event](...)

end

end

local frame = CreateFrame(“Frame”)

frame:RegisterEvent(“COMBAT\_LOG\_EVENT\_UNFILTERED”)

frame:SetScript(“OnEvent”, onEvent)

function MyMod.SPELL\_CAST\_START(\_, \_, srcGUID, srcName, \_, \_, \_, \_, \_, spellName, spellSchool)

if srcGUID == UnitGUID(“target”) then

if bit.band(spellSchool, SCHOOL\_MASK\_HOLY) ~= 0 then

print(scrName..” begins to cast ”..sellName..”!”)

end

end

end

[/code]

它使用函数UnitGUID(unitID)去获取当前目标的GUID，并将其与SPELL\_CAST\_START事件中的scrGUID进行比较。单位ID——target，总是指向你当前的目标，这里另一个可能的单位ID是focus，它总是指向你的焦点目标。附录B列出了所有单位ID和所有单位相关的API函数。

在法术类型的位掩码中，你可以使用bit.bor同时检查多个法术类型：

[code=lua]

bit.band(spellSchool, bit.bor(SCHOOL\_MASK\_HOLY, SCHOOL\_MASK\_SHADOW)) ~=0

[/code]

前面我还提到了另一个位域参数——单位标志位(the unit flags)。让我们看一下这个位域。

[b][size=130%]&#9675; 单位标志位(Unit Flags)[/size][/b]

这个位域包含关于单位的阵营(affiliation)、反应(reaction)、所有权(ownership)、类型(type)和副本角色(raid role)。还有一些包含掩码的全局变量，可以从给定的字段读取所有标志位。表9-9以十六进制表示列出了这些变量和每个变量的值。需要注意，这些位域几乎使用了所有的32位，这使得二进制表示有些模糊和冗长。位掩码的含义由变量的名称来解释。

[align=center][size=80%]表9-9 单位标志位的掩码[/size][/align]

[table]

[tr]

[td50][align=center]变量[/align][/td]

[td][align=center]掩码[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_AFFILIATION\_MINE[/align][/td]

[td][align=center]0x00000001[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_AFFILIATION\_PARTY[/align][/td]

[td][align=center]0x00000002[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_AFFILIATION\_RAID[/align][/td]

[td][align=center]0x00000004[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_AFFILIATION\_OUTSIDER[/align][/td]

[td][align=center]0x00000008[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_REACTION\_FRIENDLY[/align][/td]

[td][align=center]0x00000010[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_REACTION\_NEUTRAL[/align][/td]

[td][align=center]0x00000020[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_REACTION\_HOSTILE[/align][/td]

[td][align=center]0x00000040[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_CONTROL\_PLAYER[/align][/td]

[td][align=center]0x00000100[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_CONTROL\_NPC[/align][/td]

[td][align=center]0x00000200[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_TYPE\_PLAYER[/align][/td]

[td][align=center]0x00000400[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_TYPE\_NPC[/align][/td]

[td][align=center]0x00000800[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_TYPE\_PET[/align][/td]

[td][align=center]0x00001000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_TYPE\_GUARDIAN[/align][/td]

[td][align=center]0x00002000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_TYPE\_OBJECT[/align][/td]

[td][align=center]0x00004000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_TARGET[/align][/td]

[td][align=center]0x00010000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_FOCUS[/align][/td]

[td][align=center]0x00020000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_MAINTANK[/align][/td]

[td][align=center]0x00040000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_MAINASSIST[/align][/td]

[td][align=center]0x00080000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_RAIDTARGET1[/align][/td]

[td][align=center]0x00100000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_RAIDTARGET2[/align][/td]

[td][align=center]0x00200000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_RAIDTARGET3[/align][/td]

[td][align=center]0x00400000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_RAIDTARGET4[/align][/td]

[td][align=center]0x00800000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_RAIDTARGET5[/align][/td]

[td][align=center]0x01000000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_RAIDTARGET6[/align][/td]

[td][align=center]0x02000000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_RAIDTARGET7[/align][/td]

[td][align=center]0x04000000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_RAIDTARGET8[/align][/td]

[td][align=center]0x08000000[/align][/td]

[/tr]

[tr]

[td][align=center]COMBATLOG\_OBJECT\_NONE[/align][/td]

[td][align=center]0x80000000[/align][/td]

[/tr]

[/table]

这些位掩码可以分为五类：阵营(affiliation)、反应(reaction)、控制(或称位所有权，the ownership)、类型(type)、副本角色(raid role)。前四个类别的标志位是相互排斥的，在一个单位标志位中，绝不会有两个相同类别的比特位。

[quote]

注意：在团队中，与你在同一团队中的成员没有设置其团队的从属位。他们只有队伍的比特位被设置。此外，你自己的单位标志位将始终在此类别中被设置，而永远不会设置队伍或团队标志位。

[/quote]

我们现在可以使用掩码来更新函数MyMod.SPELL\_CAST\_START。我们通过更改这个函数来提醒我们敌对玩家施放的神圣施法。掩码COMBATLOG\_OBJECT\_REACTION\_HOSTILE和COMBATLOG\_OBJECT\_TPYE\_PLAYER可以被用在这里。

[code=lua]

function MyMod.SPELL\_CAST\_START(\_, \_, srcGUID, srcName, srcFlags, \_, \_, \_, \_, spellName, spellSchool)

if bit.band(spellSchool, SCHOOL\_MASK\_HOLY) ~= 0

and bit.band(srcFlags, COMBATLOG\_OBJECT\_REACTION\_HOSTILE) ~= 0

and bit.band(srcFlags, COMBATLOG\_OBJECT\_TPYE\_HOSTILE) ~= 0 then

print(srcName..” begins to cast “..spellName..”!”)

end

end

[/code]

与副本角色相关的掩码可以用来检查目标是你的主坦克(main tank)，主协助(main assist)还是被标记的目标，它们也提供了有趣的东西。编写一个只处理与标记目标或主坦克相关的事件函数是非常容易的。让我们在此改变这个函数，让他提醒你每一个被标记的单位所施放的神圣法术。这意味着我们需要检查是否设置了团队目标位。我们也可以用8位bit.band调用(每个比特调用1位)或使用掩码来设置所有的团队目标。构建一个掩码显然是更好的解决方案，实际上我们甚至不需要构建它。文件Interface\FrameXML\Constants.lua已经定义了可以使用的变量COMBATLOG\_OBJECT\_RAIDTARGET\_MASK。

[code=lua]

COMBATLOG\_OBJECT\_RAIDTARGET\_MASK = bit.bor(

COMBATLOG\_OBJECT\_RAIDTARGET1,

COMBATLOG\_OBJECT\_RAIDTARGET2,

COMBATLOG\_OBJECT\_RAIDTARGET3,

COMBATLOG\_OBJECT\_RAIDTARGET4,

COMBATLOG\_OBJECT\_RAIDTARGET5,

COMBATLOG\_OBJECT\_RAIDTARGET6,

COMBATLOG\_OBJECT\_RAIDTARGET7,

COMBATLOG\_OBJECT\_RAIDTARGET8

)

[/code]

我们可以在if语句中使用这个掩码来检查施法者是否标记位副本目标。

[code=lua]

function MyMod.SPELL\_CAST\_START(\_, \_, srcGUID, srcName, srcFlags, \_, \_, \_, \_, spellName, spellSchool)

if bit.band(spellSchool, SCHOOL\_MASK\_HOLY) ~= 0

and bit.band(srcFlags, COMBATLOG\_OBJECT\_RAIDTARGET\_MASK) ~= 0 then

print(srcName..” begins to cast ”..spellName..”!”)

end

end

[/code]

有许多插件类型可以明智地使用这些标志，并从中获益。例如，PVP(玩家vs.玩家)插件可以使用单位标志位来识别敌对和友好的玩家，而副本插件可以使用它们来识别主坦克和标记的怪物。

我告诉过你，我们会写一个插件例子，但是你在这一节看到的一切都是关于战斗日志事件的高度理论性的描述。让我们做一些联系，并构建一个真正的插件。