[quote]

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

======

[b][size=150%]该[/size]教材的出版时间是2009年，由于游戏与软件的迭代特性，本书的内容已经与现在你实际接触到的游戏与软件内容有所差异，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]

======

[b][size=150%]这[/size][/b]是国外教材《Beginning Lua with World of Warcraft Addons》的第十二个章节。全书共14个章节和2个附录。这是我翻译的第三个章节，其余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]

虽然教材在更新如此频繁的游戏中已经算得上是个“老大爷”了，但依然可以给我们提供很多有用的“建议”和思路。我极不推荐把这本书作为新版本的插件开发指南。但若从入门或业余角度来看，这本书依然是本相当优秀的教程，可以对魔兽和插件有个整体的了解。

我已经出了我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]

[size=110%][b]最后，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][/size]

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]处理游戏事件(Working with Game Events)[/b][/size]

[list]

[\*][b][size=130%]使用框体作为事件监听器(Using Frames as Event Listeners)[/size][/b]

[\*][b][size=130%]创建聊天链接悬停提示(Creating ChatlinkTooltips)[/size][/b]

[list]

[\*][b]聊天框及其脚本处理程序(Chat Frames and Their Script Handler)[/b]

[\*][b]物品链接(Item Links)[/b]

[\*][b]使用鼠标提示工具(Using Tooltips)[/b]

[/list]

[\*][b][size=130%]OnEvent[/size][/b]

[list]

[\*][b]事件处理程序基础(Event Handler Basics)[/b]

[\*][b]用于多个事件的事件处理程序(Event Handlers for Multiple Events)[/b]

[/list]

[\*][b][size=130%]使用OnUpdate构建计时库(Using OnUpdate to Build a Timing Library)[/size][/b]

[list]

[\*][b]调度器(The Scheduler)[/b]

[\*][b]The Unscheduler[/b]

[\*][b]OnUpdate和性能(OnUpdate and Performance )[/b]

[/list]

[\*][b][size=130%]为DKP模块使用计时库(Using the Timing Library for a DKP Mo)[/size][/b]

[list]

[\*][b]变量和选项(Variables and Options)[/b]

[\*][b]局部函数及其作用域(Local Functions and Their Scope)[/b]

[\*][b]开始拍卖(Starting Auctions)[/b]

[\*][b]结束拍卖(Ending Auctions)[/b]

[\*][b]出价(Placing bids)[/b]

[\*][b]创建斜杠命令(Creating Slash Commands)[/b]

[\*][b]远程控制(Remote Control)[/b]

[\*][b]隐藏聊天信息(Hiding Chat Messages)[/b]

[/list]

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

[/list]

[/list]

[/quote]

======

[align=center][b][color=orangered][size=200%]&#9632; 第四章 处理游戏事件[/size]

[size=130%](Working with Game Events)[/size][/color][/b][/align]

本章关于事件处理程序和如何去使用它们。事件处理程序(event handler)是一个游戏每次在发生特定事件时需要调用的函数。《魔兽世界》提供了各种各样的事件，从点击UI元素(UI elements)到战斗相关事件。因此我们需要做的是告诉游戏我们对某个事件感兴趣，并提供一个事件处理函数，该函数应该在事件发生时被调用。这样的函数也被称为回调函数(callback function)。

我们将在本章中构建三个示例模块，第一个模块将在聊天框中为物品、任务和法术链接提供鼠标悬停提示(tooltips)。图4-1显示了我们将要构建的模块。

[align=center][img]./mon\_202011/22/-10er0nQ5-c7xcZmT3cSk0-a8.png[/img]

[size=80%]图4-1，聊天链接提示模块的运行[/size][/align]

下一个示例模块将是一些更抽象的东西，一个计时库(timing library)。然而，它提供了我们接下来的章节中将要使用到其他例子中去的重要函数。这个计时库允许我们准备一个函数在给定的时间之后被调用。

第三个模块我们将在本章建立一个完整的功能齐全的DKP拍卖师(DKP auctioneer)，它可以在你的副本团队之间拍卖物品。他将用到计时库。

对于这些模块，我们需要处理某些事件，例如当你将鼠标悬停在一个框体链接上时发生的事件。另一个我们需要处理的事件是，当有人向你发送聊天信息。但是，在开始处理事件之前，我们需要先讨论框体(frames)。什么是框体，它是如何与事件相关联的？在我们开始创建模块之前，你需要学习一些关于框体和事件的知识。

======

[color=blue][b][size=160%]&#9679; 使 用 框 体 作 为 事 件 监 听 器[/size]

[size=130%](Using Frames as Event Listeners)[/size][/b][/color]

框体是一个UI元素，比如按钮或文本框，但是我们还没有讨论图形用户界面元素的创建。因此，在接下里的实例中，我们将使用非常简单的框体类型(frame type)，简称为框体(Frame)。这样的框体在默认情况下是不可见的，我们将仅使用该框体的一个功能：它可以被用作事件监听器，这意味着它可以注册事件和相应的处理程序来处理这个事件。下面的代码显示了如何使用API函数CreateFrame来创建这样的框体：

[code=lua]

local myFrame = CreateFrame(“Frame”)

[/code]

你在上一章已经见过这个函数，它创建并返回请求类型的框体。但是我们现在能用这个框体做什么呢？被创建的myFrame表(也称为对象——object)基本上只是一个可以被传递给其他API函数的标识符。这些函数(也称为方法)在被创建的表中始终可用，我们必须使用冒号操作符来调用它们。

不同的框体类型有不同的方法，例如，一个按钮有一个方法来设置按钮上的文本。我们在这里使用的基本框体没有这样的方法。在附录A中可以找到所有框体类型的所有可用方法的列表。这里我们将只使用几个方法：SetScript、GetScript、HookScript和HasScript。这些处理事件处理程序(event handlers)的方法，在《魔兽世界》中也称为脚本处理程序(script handler，事件也称为脚本——scripts)。从现在开始，我将使用术语“脚本(script)”来指代这样的事件，而使用“脚本处理程序(script handler)”来指代事件处理程序的回调函数(callback function)。

不同的框体类型可能具有不同的脚本。例如，一个按钮具有脚本OnClick。而其他类型的框体则没有此功能，因为在一个按钮上进行单击是非常常见的，而不是在一个普通框体上。还有其他的方法可以在框体中与鼠标进行交互，例如，当用户试图拖动框体时，会调用脚本处理程序OnDragStart。我们需要以下方法：

[code=lua]

frame : SetScript(script, func)

[/code]

此函数注册了一个脚本处理程序，该处理程序将在调用脚本时被调用。脚本总是由一个以“On”开头的字符串标识。第二个参数可以是函数，也可以是用于删除脚本处理程序的空值“nil”。SetScript将覆盖给定脚本的任何现有脚本处理程序。

当脚本发生时，传递给脚本处理程序(只是一个函数)的第一个参数始终是负责脚本的框体。这允许我们将一个函数分配给多个框体。以下所有参数都取决于脚本类型。

[code=lua]

frame : GetScirpt(script)

[/code]

上面的函数用于获取分配给特定脚本的函数。如果此脚本没有处理程序，则返回nil。

[code=lua]

frame : HookScript(script, func)

[/code]

仅当框体已具有该脚本的脚本处理程序时，此功能才起作用。它将添加func作为额外的回调函数给脚本。删除原始脚本处理程序还将删除所有钩子(hooks)。

[code=lua]

frame : HasScript(script)

[/code]

如果script是框体类型的有效脚本，则该函数返回1，否则返回nil。注意，这个函数不能用来检查是否设置了脚本处理程序。你必须使用GetScript。

现在，我们可以使用哪些脚本进行操作呢？如果你查看附录A中提供的常见框体的可用脚本列表，你会注意到几乎所有的脚本都与图形用户界面元素有关。例如，有像OnShow、OnHide、OnDragStart和OnDragStop这样的脚本。当将框体用作窗口打开、关闭或拖动它时，就会出现这些脚本。

但是如果回到本章开始时，我提到的那些示例模块。我们不需要在这里创建一个框体，因为我们可以使用已经存在的聊天框。聊天框的类型是ScrollingMessageFrame。它提供了脚本处理程序OnHyperlinkEnter和OnHyperlinkLeave，当鼠标悬停或离开一个在该框体中的链接时，将调用它们。链接是消息框体中任何可点击的文本块。所以，除了一些明显的链接，比如物品、聊天频道，消息前面的玩家名字也是链接。现在让我们开始构建这个插件吧。

======

[color=blue][b][size=160%]&#9679; 创 建 聊 天 链 接 悬 停 提 示[/size]

[size=130%](Creating ChatlinkTooltips)[/size][/b][/color]

我们需要的第一件事是在文件夹Interface\AddOns中创建一个名为ChatlinkTooltips的文件夹。创建插件的第二步始终是创建名为ChatlinkTooltips的TOC文件。这个插件的TOC文件应该像这样：

[code]

## Interface : 30100

## Title : Chatlink Tooltips

ChatlinkTooltips.lua

[/code]

可以使用属性(attribute)随意添加其他元数据(metadata)，比如描述。下一个任务是创建完成这项艰巨工作的文件ChatlinkTooltips.lua。

[b][size=130%]&#9675; 聊天框及其脚本处理程序(Chat Frames and Their Script Handler)[/size][/b]

我们需要获得对所有聊天框的引用(references)。我们需要使用它们的脚本处理程序，因为我们需要当用户将鼠标悬停在聊天框中的链接上时出现脚本。我们可以使用全局变量ChatFrameX。而X是我们想要的聊天框编号，所以你的第一个聊天框是ChatFrame1、第二个是ChatFrame2，以此类推。你的默认聊天框(带有用于发送消息的编辑栏)也存储在全局变量DEFAULT\_CHAT\_FRAME中。

对于此类插件，我们需要创建一个循环来遍历所有的聊天框体，并设置例如链接相关脚本处理程序一样的显示或隐藏提示的函数。这个示例中出现的一个复杂之处是，我们需要动态地、按顺序地访问全局变量ChatFrameX(换句话说就是，我们需要在循环迭代中先访问ChatFrame1，之后再访问ChatFrame2，以此类推)。《魔兽世界》提供了一个函数，它返回一个给定名称的全局变量的值：getglobal(name)。这里的参数name是一个字符串，因此我们可以在每次迭代期间动态地构建name。

下面的代码显示了可以被用作脚本处理程序的一个循环和两个伪函数(showTooltip和hideTooltip)。这些函数会将它们的参数打印到聊天框中，这样你就可以看到那里发生了什么。我还创建了一个辅助函数(helper function)，试图去钩住一个如果存在的脚本处理程序，否则，它将设置一个新的脚本处理程序。将此功能转移到一个小函数中是很有帮助的，因此我们需要为两个脚本处理程序OnHyperLinkEnter和OnHpyerLinkLeave去做这项工作。下面是代码：

[code=lua]

local function showTooltip(...)

print(...)

end

local function hideTooltip(...)

print(...)

end

local function setOrHookHandler(frame, script, func)

if frame:GetScript(script) then -- check if already has a script handler...

frame:HookScript(script, func) -- ...and hook it

else

frame:SetScript(script, func) -- set our function as script handler otherwise

end

end

for i = 1, NUM\_CHAT\_WINDOWS do

local frame = getglobal(“ChatFreme”..i) -- copy a reference

if frame then -- make sure that the frame exists

setOrHookandler(frame, “OnHyperLinkEnter”, showTooltip)

setOrHookandler(frame, “OnHyperLinkLeave”, hideTooltip)

end

end

[/code]

让我们来看看循环的主体。第一行只是将这个迭代处理的框体的引用复制到名为frame的局部变量中，所以我们不必一直写getglobal(“ChatFrame”..i)。之后代码检查框架是否确实存在，这个检查应该永远不会失败，但是你永远不知道你的用户安装了哪些可能会破坏聊天框的插件。之后它调用辅助函数setOrhookHandler，如果框体还没有用于该脚本的脚本处理程序，它将设置我们的脚本处理程序。否则，它会钩住(hook)我们的脚本处理程序去保护任何现有的脚本。注意，默认用户界面不使用这两个脚本处理程序。这个检查是为了维护与可能拥有类似功能的聊天插件的兼容性。

[b][size=130%]&#9675; 物品链接(Item Links)[/size][/b]

你现在可以在游戏中加载这个插件，但是不要忘记在创建文件后重启游戏。然后尝试将鼠标悬停在聊天框中的某个链接上，当鼠标进入或离开某个链接时，你将看到一条如下所示的信息。

[quote]

table:168BAC00 item:40449:3832:3487:3472:0:0:0:0:80 [Valorous Robe of Faith]

[/quote]

第一个参数是负责脚本处理程序的框体：

[quote]

table:168BAC00

[/quote]

稍后添加鼠标提示时，你将看到我们可以如何处理此框架。

第二个参数是链接的数据：

[quote]

item:40449:3832:3487:3472:0:0:0:0:80

[/quote]

链接的数据可以是任何字符串，默认用户界面使用的链接总是由冒号分隔的独立小字符串组成。我们将在第九章中创建和使用我们自己的链接。

第一个冒号之前的子字符串是链接的类型，因此这个是一个item(物品)链接。下面的数字是物品ID(40449)，下一个数字是物品上的附魔ID(3832)，接下来的四个数字(3487,3472,0,0)是已镶嵌珠宝的物品ID。下一个数字(0)是稀有(uncommon)物品的后缀ID。后面是物品的唯一ID(0)，但并非所有物品都有唯一ID，并且你无法从该ID中获取任何有用的信息。最后一个(80)是发送链接的玩家的等级，该物品与等级相匹配。

第三个参数是在聊天框中显示的整个可点击链接。

[quote]

[Valorous Robe of Faith]

[/quote]

[quote]

小贴士：大多数物品数据库网站也可以使用物品ID，比如[url=][url]http://www.wowhead.com[/url][/url]。这样允许你用物品链接构建url，例如，你可以使用上面链接中的一个珠宝ID来获取链接[url]http://www.wowhead.com/?item=3487[/url]。

[/quote]

[b][size=130%]&#9675; 使用鼠标提示工具(Using Tooltips)[/size][/b]

现在我们有了所需的脚本和数据。剩下的唯一任务就是显示鼠标提示。我们将使用框体GameTooltip来实现这一点，因为它是游戏使用的默认鼠标提示。它提供了SetHpyerlink(linkData)方法，该方法从一个链接(长字符串中有很多数字用冒号分隔)获取数据，并将鼠标提示的内容设置为相应的物品、法术、或成就。如果你提供的链接没有相关的鼠标提示(如玩家或频道链接)，则此方法将生成错误的消息。因此，我们需要在将链接传递给此方法之前检查它。下面的代码显示了我们的模块所需的脚本处理程序。这两个函数将取代我们的伪函数。这些函数需要放在循环之前，因为保存这些函数的局部变量需要在循环中可见。

[code]

loacal function showTooltip(self, linkData)

local linkType = string.splint(“:”, linkData)

if linkType == “item”

or linkType == “spell”

or linkType == “enchant”

or linkType == “quest”

or linkType == “talent”

or linkType == “glyph”

or linkType == “unit”

or linkType == “achievement” then

GameTooltip:SetOwner(self, “ANCHOR\_CURSOR”)

GameTooltip:SetHyperlink(linkData)

GameTooltip:Show()

end

end

[/code]

让我们逐行地看下这些代码。函数showTooltip使用string.split来获取第一部分，这表示链接的类型。if语句中的长表达式检查它是否是可以传递到SetHpyerlink的链接。但在调试SetHpyerlink之前，我们需要调试方法SetOwner。鼠标提示总是绑定到拥有该鼠标提示的框体上，隐藏所有者也会隐藏该鼠标提示。我们将鼠标提示(tooltip)的所有者设置为当前聊天框体。回想一下，传递给脚本处理程序的第一个参数(在本例中为self)是发起调用的框体，因此self是当前的聊天框。这个方法的第二个参数是鼠标提示的锚点，我们在这里使用了ANCHOR\_CURSOR，它使鼠标提示粘附在你的鼠标上。当我们在下一章讨论框体的创建时，你会学到更多关于锚点和框体定位的知识。之后调用Show方法，该方法将显示鼠标提示。隐藏鼠标提示的方法非常简单。它只是调用工具提示中的Hide方法。

现在模块已经准备好投入使用。注意，战斗日志中的法术和名称也同样适用，因为战斗日志也是一个聊天框。

现在你知道了如何使用脚本处理程序，但到目前为止，我们使用的所有脚本处理程序都与用户界面的图形部分有关。最有意思的是，脚本处理程序与GUI无关，而是与处理OnEvent脚本相关。我前面提到过脚本是事件，所有这听起来可能很奇怪——用于其它事件的脚本处理程序？是的，这个脚本处理程序有它自己的事件，所有与游戏相关的事件都将调用OnEvent脚本处理程序，并将一个事件作为字符串传递给它。这值得我们仔细研究。

======

[color=blue][b][size=160%]&#9679; OnEvent[/size][/b][/color]

我们使用OnEvent脚本处理程序之前，我们需要先讨论框体对象的另一种方法：frame:RegisterEvent(event)。此方法被该框体用于注册游戏相关的事件。只有在此框体中注册的事件才能被传递给它的OnEvent脚本处理程序。

因此，我将使用术语“脚本(script)”和“脚本处理程序(script handler)”来指代与GUI相关的事件和事件处理程序，就像我们在上一节中看到的那样。

术语“脚本(script)”和“脚本处理程序(script handler)”指的是OnEvent脚本处理程序。这些事件处理程序处理游戏事件，如聊天信息、施法(由你或其他玩家)或进入副本(instance)。

[b][size=130%]&#9675; 事件处理程序基础(Event Handler Basics)[/size][/b]

与GUI相关的事件和与游戏相关的事件之间的区别目前可能令人感到困惑，但你会很快习惯它。让我们看一个使用事件CHAT\_MSG\_WHISPER去演示事件处理程序的例子。此事件发生在每次你收到一个私聊消息(whisper message)的时候。注意，为了达到测试的目的，你可以对你自己发送私聊消息。

[code=lua]

local frame = CreateFrame(“Frame”)

local function myEventHandler(self, event, msg, sender)

print(event, sender, msg)

end

frame:RegisterEvent(“CHAT\_MSG\_WHISPER”)

frame:SetScript(“OnEvent”, myEventHandler)

[/code]

每当你收到一条私聊消息的时候，CHAT\_MSG\_WHISPER将显示，在聊天栏中显示发送私聊的玩家的名称以及该消息的文本。

事件处理程序接收到的第一个参数始终是被附加了处理程序的框体(在本例中为self)。第二个参数是发生的事件的名称(此处为event)。以下所有参数均取决于事件。

事件处理程序接收CHAT\_MSG\_WHISPER事件上的13个参数。第三个参数经常被称为arg1，或“事件的第一个参数”，因为实际的前两个参数是固定。事件最经常用的参数是第一个消息(the message)，第二个发送者(the sender)和第六个发送者状态(离开或免扰，“AFK”或“DND”)。最后(第13个)参数是此会话中收到的所有聊天消息的计数器。这四个参数的含义对于所有聊天事件都是相同的，包括团队聊天(CHAT\_MSG\_RAID)或公会聊天(CHAT\_MSG\_GUILD)中的消息。

但其他9个参数呢？其中一些用于其他聊天事件，如CHAT\_MSG\_CHANNEL。它们指的是聊天频道或受频道动作影响的玩家，如移除(kick)或禁言(ban)。除了发送消息的玩家和消息文本之外，你很少需要使用其他参数，因此你不必担心这所有这些参数。

有些参数总是空字符串或0。它们的确切用途尚不清楚，但是将它们从列表中删除会使参数向下移动，从而破坏现有事件处理程序的兼容性。所以不要认为你遇到的参数似乎没有意义。在下一节中，你将看到如何使用可变参数(varargs)快速提取其中一个参数。

[b][size=130%]&#9675; 用于多个事件的事件处理程序(Event Handlers for Multiple Events)[/size][/b]

大多数插件只有一个事件处理函数和一个框体，用于注册所有事件并将它们传递给回调函数。因此只有一个函数来处理你的插件所有需要被告知的事件。这个函数可能类似于下面的例子：

[code=lua]

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

if event == “CHAT\_MSG\_WHISPER” then

-- we received a whisper, do something with it here

local msg, sender = ...

print(sender..” wrote ”..msg)

elseif event == “ZONE\_CHANGED\_NEW\_AREA” then

-- we are in a new zone, do something different

elseif event == ”PARTY\_MEMBERS\_CHANGED” then

-- someone joined or left our group, deal with it

elseif event == ”...” then

-- to be continued...

end

end

[/code]

回想一下这个函数头部的三个点的意思：“含有未知的额外参数的数量。”。正如第二章中所讨论的，这三个点被称为可变参数(vararg)，你可以在Lua需要值列表的任何地方使用它们。通过编写示例中的第4行代码，我们可以从这个可变参数中获得常规变量(normal variables)。

[code=lua]

local msg, sender = ...

[/code]

还可以使用select去更改可变参数的开头。例如，如果你希望从可变参数中获得第六个参数CHAT\_MSG\_WHISPER事件(消息发送方的状态)，则不必创建五个无用变量来获得它。你可以只使用以下代码：

[code=lua]

local status = select(6,...)

[/code]

但是使用这种方法，整个功能可能会变得庞大而且模糊。编写监听20多个或更多事件的插件并不少见。因此，你可能采取的下一步是通过将if块的每个部分分为单个函数调用来分割这个函数。于是就有了许多小函数，每个函数都专门用于一个事件。它可以是这样的：

[code=lua]

local function onWhisper(msg, sender)

-- deal with whispers

end

local function onNewZone()

-- we are in a new zone

end

-- ... etc

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

if event == “CHAT\_MSG\_WHISPER” then

onWhisper(...)

elseif event == “ZONE\_CHANGED\_NEW\_AREA” then

onNewZone(...)

elseif event == “...” then

-- etc

end

end

[/code]

但还有一个更聪明的方法。你可以创建一个表，并使用事件作为键(key)，并将相关处理程序存储在此键(key)下。这使得分离事件非常容易，并且你不需要巨大的if结构。我们的事件处理程序就像这样：

[code=lua]

local eventHandlers = {}

function eventHandlers.CHAT\_MSG\_WHISPER(msg, sender)

end

function eventHandlers.ZONE\_CHANGED\_NEW\_AREA()

end

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

returen eventHandlers[event](...)

end

[/code]

该解决方案的另一个优点是，在任何时候用新的事件处理程序注册新的事件都非常容易。只需调用框架的RegisterEvent方法并将函数添加到表中。如果有很多事件，这种解决方案也可以更快。你将在第13章中看到这些示例以及更多关于性能的内容。

另一个值得仔细研究的脚本处理程序是OnUpdate处理程序。

======

[color=blue][b][size=160%]&#9679; 使 用 OnUpdate 构 建 计 时 库[/size]

[size=130%](Using OnUpdate to Build a Timing Library)[/size][/b][/color]

OnUpdate脚本处理程序总是在游戏呈现用户界面之前执行。这意味着如果你的游戏每秒运行25帧，则这个脚本处理程序将每秒执行25次。该函数接收一个额外的参数，即自从最后一次调用它以来所经过的时间。因此，我们将使用这个脚本处理程序来构建一个插件，该插件提供一个功能，该功能安排在特定时间后调用另一个功能。这次由你来创建TOC文件。

[b][size=130%]&#9675; 调度器(The Scheduler)[/size][/b]

我们需要的是一个花费一定时间(以秒为单位)的函数、另一个函数和一个参数列表。之后，插件将在指定的时间后调用使用了这些参数的函数。这个功能是经常需要使用到的，我们将在后面的例子中使用这个插件。下面的代码显示了这个调度函数和另一个存储所有调度任务的列表。

[code=lua]

local tasks = {}

function SimpleTimingLib\_Schedule(time, func, ...)

local t = {...}

t.func = func

t.time = GetTime() + time

table.insert(tasks, t)

end

[/code]

可变参数可以在Lua期望值列表的任何地方使用，因此{...}将创建一个新表，并用存储在可变参数中的值填充其数组部分。因此，数组部分将存储所有被传递给函数的参数。下一行存储了在哈希表中的func键值下的函数。函数执行的确切时间存储在键time下。API函数GetTime()以毫秒的精度返回当前系统的正常运行时间。向这个值添加时间，我们就得到了一个执行任务的时间。OnUpdate处理程序将根据GetTime()的当前值检查该条目。函数的最后一行将我们的新任务插入到表tasks中，该表存储所有任务。

我们现在需要编写OnUpdate处理程序，它在任务结束时执行任务。该函数遍历所有任务，并检查特定任务的时间是否结束。我们将使用函数unpack从表中获取参数。这个函数返回表的数组部分中存储的所有值。下面是处理程序的代码：

[code=lua]

local function onUpdate()

for i = #tasks, 1, -1 do

local val = tasks[i]

if val.time <= GetTime() then

table.remove(tasks, i)

val.func(unpack(val))

end

end

end

local frame = CreateFrame(“Frame”)

frame:SetScript(“OnUpdate”, onUpdate)

[/code]

注意，代码没有使用ipairs遍历表。相反，它使用数值for循环向后遍历表，起始值为#tasks，结束值为1，步长为-1。然后，我们将表中的当前条目存储在循环中的局部变量中。在这里，向前遍历表可能会导致问题，因为我们在遍历表时使用的是table.remove删除元素。这个函数在删除元素之后，元素向下移动一次。因此，如果在使用ipairs遍历表时删除了当前元素，则将跳过下一个元素。

循环保存对当前元素的引用，并检查是否应该现在执行它。如果是这种情况，则从列表中删除该任务并执行。在执行之前删除它是很重要的，因为执行可能会导致错误。一个错误会取消正在运行的Lua脚本。因此，如果你在执行后删除了它，并且发生了错误，那么下一次调用OnUpdate处理程序时，该条目(entry)仍然在表中。在这种情况下，脚本将再次尝试执行它，该条目仍将在表中。在这种情况下，脚本将再次尝试执行它，它会再次失败，你会一次又一次地得到同样的错误。

另一个可能出现的问题是如果你在参数中有nil值会发生什么？思考一下下面的调用：

[code=lua]

SimpleTimingLib\_Schedule(1, print, 1, nil, 3)

[/code]

你可以使用/script在游戏中执行。你得到的结果是1秒后将“1”打印到聊天框中。我们的代码忽略了调度任务(the scheduled task)的第二个和第三个参数，因为unpack只对表的数组部分有效，而第三个条目将在哈希表部分。这个操作实际上要更复杂一些，因为数组部分也可以包含nil值，但只在特定的情况下。这个话题超出了本章范围。我们将在第13章中进一步研究表。现在不要在调度函数中使用nil。

对此类库有用的另一个函数是取消任务的函数，让我们调用SimpleTimingLib\_Unschedule。

[b][size=130%]&#9675; The Unscheduler[/size][/b]

我们现在要做的是遍历并删除一个或多个任务。除了时间之外，该函数将接收与调度器(scheduler)相同的参数。然后，他将删除所有与给定条件相匹配的任务：

[code=lua]

function SimpleTimingLib\_Unschedule(func, ...)

for i = #tasks, 1, -1 do

local val = tasks[i]

if val.func == func then

local matches = true

for i = 1, select(“#”, ...) do

if select(i, ...) ~= val[i] then

matches = fasle

break

end

end

if matches then

table.remove(tasks, i)

end

end

end

end

[/code]

回想一下select(“#”, ...)返回变量中存储的参数数量。

该函数遍历指定表，并使用第二个循环来确定任务的参数是否与我们的参数匹配。注意，这个循环还使用了尚未传递给SimpleTimingLib\_Unschedule的额外参数匹配任务。这允许我们一次移除多个调度任务。让我们用下面的代码来测试它：

[code=lua]

SimpleTimingLib\_Schedule(1, print, “Foo”, 1, 2, 3)

SimpleTimingLib\_Schedule(1, print, “Foo”, 4, 5, 6)

SimpleTimingLib\_Schedule(1, print, “Bar”, 7, 8, 9)

SimpleTimingLib\_Unschedule(print, “Foo”)

[/code]

这将在一秒钟后打印“Bar 7 8 9”，因为对Unschedule函数的调用与对Schedule函数的前两次调用相匹配。

[quote]

小贴士：你可以使用TinyPad在游戏中执行这段代码。

[/quote]

但是这个非调度器(unscheduler)与调度器(scheduler)结合在一起有个bug。你发现了吗？请看下面的代码：

[code=lua]

local function buggy()

SimpleTimingLib\_Unschedule(pring, “Bar”)

end

SimpleTimingLib\_Schedule(2, pring, “Bar”)

SimpleTimingLib\_Schedule(1, buggy)

[/code]

执行此代码后，表tasks将有两个条目(entry)。第二个是调用buggy的任务。一秒过后，调度器(scheduler)执行buggy并删除其他条目。这个表现在是空的，因为执行任务之前， buggy的调用任务已经被OnUpdate脚本处理程序删除了。Lua现在返回OnUpdate处理程序中的循环，循环体将在i = 1的情况下执行第二次。它不知道现在表是空的，它将执行:

[code=lua]

local val = tasks[i]

[/code]

把val设置为nil。之后，下一行将尝试获取var.time并导致一个错误消息，因为尝试去索引一个nil值是不可能的。

修复这个问题很简单：我们只需要将访问var.time的代码改成这个：

[code=lua]

if val and val.time <= GetTimee() then

[/code]

这说明了检查变量是否成为你期望的类型是多么重要。我们希望val在这里是一个表，但在前面的情况下，它是nil，因为Unschedule函数是一个复杂的调用。

[b][size=130%]&#9675; OnUpdate和性能(OnUpdate and Performance )[/size][/b]

OnUpdate可以是一个非常有用的脚本，但他也可以是一个真正的性能杀手。所以在这使用的时候一定要小心。始终记住，它在每一帧(frame)中被执行。

你还应该问问自己是否真的有必要在每一帧(frame)中执行某个任务。如果你认为某件事必须尽可能频繁地执行，那么每隔几帧或0.5秒执行一次就足够了。限制OnUpdate处理程序的调用频率的一种常用方法是用下面将介绍的函数包装(wrap)它。

如前面所述，传递给OnUpdate脚本处理程序的第二个参数是自上次调用以来所经过的时间。我们可以用它来限制每秒的调用次数：

[code=lua]

local function onUpdate(self, elapsed)

-- the update task is here

end

local frame = CreateFrame(“Frame”)

local e = 0

frame:SetScript(“OnUpdate”, function(self, elapsed)

e = e + elased

if e >= 0.5 then

e = 0

returen onUpdate(self, elapsed)

end

end)

[/code]

这将每秒调用两次onUpdate函数。如果你想每隔几帧调用它，你可以使用以下包装函数(wrapper function)：

[code=lua]

local frame = CreateFrame(“Frame”)

local counter = 0

frame:SetScript(“OnUpdate”, function()

counter = counter + 1

if counter % 5 == 0 then

return onUpdate(self, elapsed)

end

end

[/code]

这将每隔5帧调用一次onUpdate函数。

======

[color=blue][b][size=160%]&#9679; 为 DKP 模 块 使 用 计 时 库[/size]

[size=130%](Using the Timing Library for a DKP Mod)[/size][/b][/color]

本章的最后一个例子将使用我们刚刚构建的计时库和事件处理程序来构建一个DKP拍卖师(DKP auctioneer)。这个插件将通过一个简单的命令行界面(斜杠命令)来控制。另一个有用的功能是允许团队(raid)或公会(guild)中的其他玩家，比如你的官员(officers)或团长(raid leaders)，开始或停止一个物品(item)的拍卖。

该模块将遵循我的公会使用的DKP系统规则。拍卖中所有的出价都是隐藏的，在最后出价最高的玩家必须支付第二高的出价上加上1DKP。如果只有一个玩家对一个物品出价，他只需支付在DKP规则中所定义的最低出价。如果一个以上玩家的出价相同且这个金额为最高出价，则它们必须通过掷骰子(roll)来决定这个物品。你可以调整插件来匹配你的DKP规则。

[b][size=130%]&#9675; 变量和选项(Variables and Options)[/size][/b]

拍卖开始后，这个插件将发布物品和剩余的时间到一个可配置(configurable)的聊天频道。这个插件可以通过私聊接受出价，并在时间到的时候将出价最高者发布到聊天频道。

所以让我们开始建立这个模块吧。构建TOC文件对你来说应该是一项简单的任务。注意，你应该将以下的代码行添加到TOC文件中。

[code]

## Dependencies: SimpleTimingLib

[/code]

这确保SimpleTimingLib在DKP插件加载之前被加载，因为我们将需要这个库。我们将从Lua代码开始，定义一些例如选项(options)一样的保存当前状态的变量。

[code=lua]

local currentItem -- the current item or nil if no auction is running

local bids = {} -- bids on the current item

local prefix = “[SimpleDKP] ” -- prefix for chat message

-- default values for saved variables/options

SimpleDKP\_Channel = “GUILD” -- the chat channel to use

SimpleDKP\_AuctionTime = 30 -- the time (in seconds) for an auction

SimpleDKP\_MinBid = 15 -- the minmum amount of DKP you have to bid

SimpleDKP\_ACL = {} -- the access control list

[/code]

访问控制列表(ACL，the access control list) 是一个允许玩家通过聊天命令控制插件的列表。它一开始是空的，这意味着不允许任何人控制你的插件。

将变量SimpleDKP\_Channel、SimpleDKP\_AuctionTime、SimpleDKP\_MinBid和SimpleDKP\_ACL添加到TOC文件保存的变量中。下一个我们要写的函数是开始拍卖的函数。它有两个参数：物品和需要拍卖的玩家。

[b][size=130%]&#9675; 局部函数及其作用域(Local Functions and Their Scope)[/size][/b]

在创建这个函数之前，我们先考虑一下。如果将这个函数存储在一个局部变量中是一个好主意，因为我们只需要从这个内部(within)文件中访问它，而不需要从外部(outside)访问。但是这个变量的作用域(scope)呢？它的作用域在变量创建之后开始，这意味着在包含关键字local的语句之后开始。所以如果你有很多函数在局部变量中，可能会出现下面这样的情况：

[code=lua]

local function foo()

bar()

end

local function bar()

-- do something

end

[/code]

函数foo在这里尝试调用函数bar，但是它没有看到本地变量bar，因为它是在几行之后创建的。这意味着它将尝试访问一个名为bar的全局变量，而不是它应该访问的本地变量。因此，在脚本的开始部分，创建整个文件中可用的所有本地变量是一个好主意。这样就避免了这样的问题，因为局部变量的作用域在声明之后就开始了，初始化不会影响它。

如果我们考虑之后需要用到的函数，我们会得到以下局部变量：

[quote]

local startAuction, endAuction, placeBid, cancelAuction, onEvent

startAuction：开始一场新的拍卖。

endAuction：当前拍卖到到期时，它将把结果发送到聊天框。

placeBid：每当有人想要对当前进行的拍卖出价时，placeBid将被调用。

cancelAuction：可以提前取消拍卖。

onEvent：是我们的事件处理程序。

[/quote]

[b][size=130%]&#9675; 开始拍卖(Starting Auctions)[/size][/b]

我们现在可以开始实现这些函数了。下面的代码显示了startAuction函数。注意，此函数使用的所有字符串都是在函数外部创建的，并且函数及其变量包装在do-end块中。所以这些局部变量只在需要的地方可见。

将字符串存储在变量中能让我们轻松的更改它们，而无需搜索和更改每次出现的字符串。以后如果我们要翻译我们的插件时，这将会很有用(这个话题我们将在书的后面讨论)：

[code=lua]

do

local auctionAlreadyRunning = “There is already an auction running! (on %s)”

local startingAuction = prefix..”Starting auction for item %s, please please place your bids by whispering me. Remaining time: %d seconds.”

local auctionProgress = prefix..”Time remaining for %s: %d seconds.”

function startAuction(item, starter)

if currentItem then

local msg = auctionAlreadyRunning:format(currentItem)

if starter then

SendChatMessage(msg, “WHISPER”, nil, starter)

else

print(msg)

end

else

currentItem = item

SendChatMessage(startingAuction:format(item, SimpleDKP\_AuctionTime), SimpleDKP\_Channel)

if SimpleDKP\_AuctionTime > 30 then

SimpleTimingLib\_Schedule(SimpleDKP\_AuctionTime - 30, SendChatMessage, auctionProgress:format(item, 30), SimpleDKP\_Channel)

end

if SimpleDKP\_AuctionTime > 15 then

SimpleTimingLib\_Schedule(SimpleDKP\_AuctionTime - 15, SendChatMessgae, auctionProgress:format(item, 5), SimpleDKP\_Channel)

end

SimpleTimingLib\_Schedule(SimpleDKP\_AuctionTime, endAuction)

end

end

end

[/code]

[quote]

注意：请不要在此处使用local function startAuction(item, starter)。这将创建一个新的局部变量startAuction，但是我们要使用的是在文件开头创建的局部变量。

[/quote]

这个函数看起来又长又复杂，但其实很简单。它首先检查是否有拍卖正在进行，如果有，则输出错误信息。如果拍卖已经悄悄开始(started remotely)，则会将错误信息私聊告知给尝试启动拍卖的玩家。如果没有正在进行的拍卖，它将把当前物品设置为新物品，并向配置的频道(channel)发送聊天消息。接下来几行只是在剩下30、15和5秒时调度(scheduling)并发送状态信息。该函数在最后一次调用调度函数endAuction是在最后。

[b][size=130%]&#9675; 结束拍卖(Ending Auctions)[/size][/b]

下一个我们要写的函数是endAuction。你必须根据所使用的DKP系统对其进行调整。有以下四种情况，该函数必须执行：

[quote]

前三种情况非常简单：

1、根本没有人出价

2、只有一个玩家拍卖物品

3、超过一个玩家出价并且只有一个最高出价。

第四种情况是超过一个玩家出了相同的价格，并且都是最高处价。

[/quote]

出价将在bids表中存储并排序，因此在写从中读取它的函数之前，我们必须为该表考虑一种格式(format)。表中的条目包含玩家的名字和代表它们出价的数字，所以你可能会尝试使用名字作为键(key)，出价作为值(value)。但是，这不是一种聪明的方法，因为我们需要对表进行排序，而哈希表按照设计是无法被排序的。如果我们在这里使用哈希表，我们就必须为它构建一个数组并对数组进行排序。因此，更简单的方法是为每个出价创建一个带有键bid和name的小表，并将这些bid表存储在表bids.Auction，并按降序对该表进行排序，因此bid[1].name将成为拍卖的赢家。

下面展示了endAuction及其辅助函数对表bids进行排序的代码：

[code=lua]

do

local noBids = prefix..”No one want to have %s :(”

local wonItemFor = prefix..”%s won %s for %d DKP.”

local pleaseRoll = prefix..”%s bid %d DKP on %s, please roll!”

local highestBidders = prefix..”%d. %s bid %d DKP”

local function sortBids(v1, v2)

return v1.bid > v2.bid

end

function endAuction()

table.sort(bids, sortBids)

if #bids == 0 then -- case 1:no bid at all

SendChatMessage(noBids:format(currentItem), SimpleDKP\_Channel)

elseif #bids == 1 then -- case 2: one bid; the bidder pays the minmun bid

SendChatMessage(wonItemFor:format(bids[1].name, currentItem, SimpleDKP\_MinBid), SimpleDKP\_Channel)

SendChatMessage(highestBidders:format(1, bids[1].name, bid[1].bid), SimpleDKP\_Channel)

elseif bids[1].bid ~= bids[2].bid then -- case 3: highest is unique

SendChatMessage(wonItemFor:format(bids[1].name, currentItem, bid[2].bid +1), SimpleDKP\_Channel)

for i = 1, math.min(#bids, 3) do -- print the three highest bidders

SendChatMessage(highestBidders:format(i, bids[i].name, bids[i].bid), SimpleDKP\_Channel)

end

else -- case 4: more then 1 bid and the highest amount is not unique

local str = “” -- this string holes all players who bid the same amount

for i = 1, #bids do -- this loop builds the string

if bids[1].bid ~= bid[1].bid then -- found a player who bid less --> break

break

else -- append the player’s name to the string

if bids[i + 2] and bids[i + 2].bid == bid then

str = str..bids[i].name..”, ” -- use a coma if this is not the last

else

str = str..bids[i].name..” and ” -- this is the last player

end

end

end

string = str:sub(0, -6) --cut off the of the string as the loop generates a

-- string that is too long

SendChatMessage(pleaseRoll:format(str, bids[1].bid, currentItem), SimpleDKP\_Channel)

end

currentItem = nil -- set currentItem to nil there is no longer an

-- ongoing auction

table.wipe(bids) -- clear the table that holds the bids

end

end

[/code]

if块很清楚地显示了函数处理的四种情况。前三种情况很简单。这个函数只需要初始化一些字符串并将它们传递给SendChatMessage，在这里没有什么特别的。第四种情况必须构建一个字符串，其中包含所有出价数额相同的玩家的名字。得到的字符串类似于“Player1，Player2和Player3”，并在循环中被构建。

处理完拍卖后，该函数将变量currentItem设置为nil，并删去出价。这个插件现在已经准备好下次的拍卖。我们的下一个函数会让你更接近本章的主题：事件(events)。我们需要一个函数，让玩家通过私聊对一个物品进行出价。

[b][size=130%]&#9675; 出价(Placing bids)[/size][/b]

此函数需要处理事件CHAT\_MSG\_WHISPER，并检查当前是否正在进行拍卖，私聊消息是否为数字。之后在bids表中创建或更新一个条目(entry)。

[code=lua]

do

local oldBidDetected = prefix..”Your old bid was %d DKP, your bid is %d DKP.”

local bidPlaced = prefix..”Your bid of %d DKP has been placed!”

local lowBid = prefix..”The minimum bid is %d DKP.”

function onEvent(self, event, msg, sender)

if event == “CHAT\_MSG\_WHISPER” and currentItem and tonumber(msg) then

local bid = tonumber(msg)

if bid < SimpleDKP\_MinBid then

SendChatMessage(lowBid:format(SimpleDKP\_MinBid), “WHISPER”, nil, sender)

return

end

for i, v in ipairs(bids) do -- check if that player has already bid

if sender == v.name then

SendChatMessage(oldBidDectected:format(v.bid,bid), “WHISPER”, nil ,sender)

v.bid = bid

return

end

end

-- he hasn’t bid yet, so create a new entry in bids

table.insert(bids, {bid = bid, name = sender})

SendChatMessage(bidPlced:format(bid), “WHISPER”, nil, sender)

end

end

end

[/code]

事件处理程序检查是否设置了currentItem，这意味是否正在进行一个拍卖。它使用tonumber(msg)来确定私聊的内容是否是一个数字。该功能检查玩家是否已经出价，并在必要时进行更新，之后，他将在bids表中创建一个条目。

现在我们需要创建一个框体(frame)并使用它作为CHAT\_MSG\_WHISPER事件的事件监听器(event listener)，并将事件处理函数onEvent设置为脚本OnEvent的脚本处理程序(这里注意onEvent和OnEvent，开头分别是小写和大写)。回想一下，处理OnEvent脚本的脚本处理程序称为事件处理程序，因此每个事件处理程序都是一个脚本处理程序。将下面的代码放在我们上面插入的do-end代码块之后：

[code=lua]

local frame = CreateFrame(“Frame”)

frame:RegisterEvent(“CHAT\_MSG\_WHISPER”)

frame:SetScript(“OnEvent”, onEvent)

[/code]

到这，我们仍然不能在游戏中测试插件，因为没有办法创建一个拍卖。让我们构建一些斜杠命令来完成这个。

[b][size=130%]&#9675; 创建斜杠命令(Creating Slash Commands)[/size][/b]

我们将注册斜杠命令/simpledkp和/sdkp。表4-1显示了我们将要构建的命令。

[align=center][size=80%]表4-1 SimpleDKP的斜杠命令[/size][/align]

[table]

[tr]

[td50][align=center]命令[/align][/td]

[td][align=center]描述[/align][/td]

[/tr]

[tr]

[td][align=center]/sdkp start <item>[/align][/td]

[td][align=center]开始拍卖<物品>[/align][/td]

[/tr]

[tr]

[td][align=center]/sdkp stop[/align][/td]

[td][align=center]停止当前拍卖[/align][/td]

[/tr]

[tr]

[td][align=center]/sdkp channel <channel>[/align][/td]

[td][align=center]如果提供了<频道>，则将聊天频道设置为该频道。否则它将在当前频道打印输出。[/align][/td]

[/tr]

[tr]

[td][align=center]/sdkp time <time>[/align][/td]

[td][align=center]设置时间为< time >。如果省略，则打印当前设置。[/align][/td]

[/tr]

[tr]

[td][align=center]/sdkp minbid <minbid>[/align][/td]

[td][align=center]设置最低出价为<minbid>。如果省略，则打印当前设置。[/align][/td]

[/tr]

[tr]

[td][align=center]/sdkp acl[/align][/td]

[td][align=center]打印允许远程控制(control the addon remotely)插件的玩家列表。[/align][/td]

[/tr]

[tr]

[td][align=center]/sdkp acl add <names>[/align][/td]

[td][align=center]添加<names>到ACL。[/align][/td]

[/tr]

[tr]

[td][align=center]/sdkp acl remove <names>[/align][/td]

[td][align=center]把<names>从ACL中移除。[/align][/td]

[/tr]

[/table]

斜杠命令处理程序很长，因为我们要处理8个命令。acl add和acl remobe命令能够接受由空格分隔的名称列表。我们可以使用string.split从这个字符串中获取名称，但我们不知道会获得多少结果。因此，一个可能的解决方案代码如下：

[code=lua]

for i =1, select(“#”, string.split(“ ”, names)) do

local name = select(i, string.split(“ ”, name))

-- do something with the name

end

[/code]

但是它每次进入循环时都会执行string.split。因此，更好的方法是创建一个带有可变参数的辅助函数。然后，该函数可以对可变参数进行操作，分隔只会在调用该函数时执行一次。所以，斜杠命令处理程序看起来像这样：

[code=lua]

SLASH\_SimpleDKP1 = “/simpledkp”

SLASH\_SimpleDKP2 = “/sdkp”

do

local setChannel = “Channel is now \”%s\””

local setTime = “Time is now %s”

local setMinBid = “Lowest bid is now %s”

local addedToACL = “Added %s player(s) to the ACL”

local removedFromACL = “Removed %s player(s) from the ACL”

local currChannel = “Channel is currently set to \”%\””

local currTime = “Time is currently set to %s”

local currMinBid = “Lowest bid is currently set to %s”

local ACL = “Access Control List:”

local function addToACL(...) -- adds multiple players to the ACL

for i = 1, select(“#”, ...) do -- iterate over the arguments

SimpleDKP\_ACL[select(i, ...)] = true -- and add all players

end

print(addedToACL:format(select(“#”, ...))) -- print an info message

end

local function removeFromACL(...) -- removes player(s) from the ACL

for i =1, select(“#”, ...) do -- iterate over the vararg

SimpleDKP\_ACL[select(i, ...)] = nil -- remove the players from the ACL

end

print(removedFromACL:Format(select(”#”, ...))) -- print an info message

end

SlashCmdList[“SimpleDKP”] = function(msg)

local cmd, arg = string.split(“”, msg) --split the string

cmd = cmd:lower() -- the command should not be case-sensitive

if cmd == “start” and arg then -- /sdkp start item

startAuction(msg:match(“^start%s+(.+)”)) -- extract the item link

elseif cmd == “stop” then -- /sdkp stop

cancelAuction()

elseif cmd == “channel” then -- /sdkp channel arg

if arg then -- a new channel was provided

SimpleDKP\_Channel = arg:upper() -- set it to arg

print(setChannel:format(SimpleDKP\_Channel

))

else -- no channel was provided

print(currChannel:format(SimpleDKP\_Channel)) -- print the current one

end

elseif cmd == “time” then -- /sdkp time arg

if arg and tonumber(arg) then -- arg is provided and it is a number

SimpleDKP\_AuctionTime = tonumber(arg) -- set it

print(setTime:format(SimpleDKP\_AuctionTime))

else -- arg was not provided or it wasn’t a number

print(currTime:format(SimpleDKP\_AuctionTime)) -- print error message

end

elseif cmd == “minbid” then -- /sdkp minbid arg

if arg and tonumber(arg) then -- arg is set and a number

SimpleDKP\_MinBid = tonumber(arg) -- set the option

print(setMinBid:format(SimpleDKP\_MinBid))

else -- arg is not set or not a number

print(currMinBid:format(SimpleDKP\_MinBid)) -- print error message

end

elseif cmd == “acl” then -- /sdkp acl add/remove player, player2, ...

if not arg then -- add/remove not passed

print(ACL) -- output header

for k, v, in pairs(SimpleDKP\_ACL) do -- loop over the ACL

print(k) -- print all entries

end

elseif arg:lower() == “add” then -- /sdkp add player1, player2, ...

-- split the string and pass all players to our helper function

addToACL(select(3, string.split(“ ”, msg)))

elseif arg:lower() == “remove” then -- /sdkp remove player1, player2, ...

removeFromACL(select(3, string.split(“ ”, msg))) -- split & remove

end

end

end

end

[/code]

这段代码很长，但也很简单。它基本上只处理设置(setting)和检索(retrieving)选项，是一个很长的if-then-else块，用于处理不同的命令。唯一稍微困难一点的部分是处理ACL的部分。这部分拆分字符串并传递除了前两个参数给从表中添加或移除名称的辅助函数。

[b][size=130%]&#9675; 取消拍卖(Canceling Auctions)[/size][/b]

取消拍卖很容易。我们只需要将变量currentItem设置为nil，清空表bids，并取消所有计划好的任务。以下是代码：

[code=lua]

do

local cancelled = “Auction cancelled by %s”

function cancelAuction(sender)

currentItem = nil

table.wipe(bids)

SimpleTimingLib\_Unschedule(SendChatMessage)

SimpleTimingLib\_Unschedule(endAuction)

SendChatMessage(cancelled:format(sender or UniName(“player”)), SimpleDKP\_Channel)

end

end

[/code]

API函数UnitName(unitID)返回一个出现在游戏中的单位(unit)名称。名称“player”总是指你自己，所以UnitName(“player”)将返回你的角色名字。

[quote]

小贴士：单位ID(unitID)被用于识别一个单位，例如，target是你的目标，pet是你的宠物，party1是你队伍的第一个成员。你可以在附录B中找到所有可用的单位ID。

[/quote]

我们的取消函数暴露了我们计时库的一个问题：我们在这里取消了所有对SendChatMessage的调用，但其他使用这个库的插件也可能调用这个函数。然而我们不想取消这一计划。当我们更新库时，你将在第七章中看到如何解决问题。

[b][size=130%]&#9675; 远程控制(Remote Control)[/size][/b]

我想让玩家通过聊天命令在你的ACL表上创建和取消拍卖。你可以将公会官员和团长添加到这个列表中，这样当你不是团队中的战利品分配者(master looter)的时候，他们也可以开始拍卖。我们想要在团队以及公会、私聊和官员聊天中监听这些聊天命令，所以我们需要添加这些事件。在代码中找到注册CHAT\_MSGG\_WHISPER事件的那一行，并在那里添加以下代码行：

[code=lua]

frame:RegisterEvent(“CHAT\_MSG\_WHISPER”) -- look for this line...

frame:RegisterEvent(“CHAT\_MSG\_RAID”) -- ...and insert these four new lines after it

frame:RegisterEvent(“CHAT\_MSG\_RAID\_LEADER”)

frame:RegisterEvent(“CHAT\_MSG\_GUILD”)

frame:RegisterEvent(“CHAT\_MSG\_OFFICER”)

[/code]

现在也将为这些事件调用事件处理程序。他们的前两个参数和私聊是一样的。因此我们只需将事件处理程序中的代码更改为以下代码：

[code=lua]

function onEvent(self, event, msg, sender)

if event == “CHAT\_WHISPER” and currentItem and tonumber(msg) then

-- old code here

elseif SimpleDKP\_SimpleDKP\_ACL[sender] then

-- not a whisper or a whisper that is not a bid

-- and the sender has the permission to send commands

local cmd, arg = msg:match(“^!(%w+)%s\*(.\*)”)

if cmd and cmd:lower() ==”auction” and arg then

startAuction(arg, sender)

elseif cmd and cmd:lower() == “cancel” then

cancelAuction(sender)

end

end

end

[/code]

访问控制列表(access control list)上的玩家现在可以通过写入“!auction <item>”在聊天中创建一个拍卖。如要取消拍卖，他们必须输入“!cancel”。如果你还习惯使用正则表达式，那么支持此功能的模式(pattern)可能看起来比较复杂，现在让我们来分析一下。

第一个字符“^”，告诉我们它在寻找字符串的开头。这很有用，因为我们不想有人在聊天框输入类似“...!cancel ...”这样的文本时触发代码。下一个字符是“！”，它没有特殊含义。后面是匹配“%w+”(一个或多个字母数字字符)的捕获。这是第一个捕获(capture)，因此找到的字符串作为一个参数返回并存储在本地变量cmd中。下一个表达式是“%s\*”，这意味着我们需要查找空格，而“\*”表示它匹配0个或多个。下一个表达式是我们的第二个捕获，它匹配“.\*”，任意数量的字符，它匹配该行的其余部分。

这意味着我们在字符串的开头寻找一个感叹号，后面跟着一个单词。可以选择后面跟着空格或任何其他文本。第一个词是第一个返回值(cmd)，在可选项空格之后的可选文本是第二个返回值(arg)。

还有一个问题：我们不希望看到有私聊外发(outgoing whispers)和输入的出价(incoming bids)。

[b][size=130%]&#9675; 隐藏聊天信息(Hiding Chat Messages)[/size][/b]

暴雪提供了ChatFrame\_AddMessageEventFilter(event, func)函数，它允许我们为聊天信息设置过滤器。函数接收一个事件和一个函数作为参数，每次在显示该类信息事件的聊天消息之前调用func。此函数接收与对应事件的事件处理程序相同的参数。函数的第一个返回值是一个布尔值，它决定聊天框是否应该取消对该事件的处理。当设置该值为真时，将过滤信息。该函数还必须返回传递给它的事件的所有参数(聊天消息的arg1 ~ arg11)。这允许你修改这些参数来更改传入的聊天信息。

但我们只需要这个聊天过滤器API的简单部分，因为我们只想隐藏某些消息，而不是修改他们。我们需要为事件CHAT\_MSG\_WHISPER和CHAT\_MSG\_WHISPER设置过滤器，前者将过滤传入的出价，后者负责传出的私聊信息。合适的过滤器函数如下所示，你可以在你的DKP拍卖师(DKP auctioneer)文件的任何地方插入它们(最好在末尾)：

[code=lua]

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

local msg = ... -- get the message from the vararg

-- return true if there is an ongoing auction and the whisper is a number

-- followed by all event handler arguments

return currentItem and tonumber(msg), ...

end

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

local msg = ... -- extract the message

return msg:sub(0, prefix:len()) == prefix, ...

end

ChatFrame\_AddMessageEventFileter(“CHAT\_MSG\_WHISPER”, filterIncoming)

ChatFrame\_AddmessageEventFilter(“CHAT\_MSG\_WHISPER\_INFORM”, filterOutgoing)

[/code]

函数filterIncoming也与事件处理程序执行同样的检查，以确定传入的私聊是否为一个出价，如果是，则不显示消息。在脚本开始时，函数filterOutgoing检查消息是否带有我们前面定义的前缀，如果是，则接收消息。这两个函数也必须返回事件处理程序的所有参数，即使我们没有更改它们。

该模块现在已经准备好投入使用了。它是一个完整功能的DKP拍卖师，如果你使用的是一个不同的规则，也可以很容易地进行调整，以匹配你的DKP系统规则。

======

[color=blue][b][size=160%]&#9679; 总 结[/size]

[size=130%](Summary)[/size][/b][/color]

在本章中，你学习了如何使用框体来监听脚本和事件。这允许我们编写插件来响应游戏中的事件。实际上，每个插件都使用脚本处理程序和事件处理程序，因此这是《魔兽世界》API中非常重要的一部分。在本章，我们写了三个强大的示例模块：第一个是在聊天框的链接中添加鼠标悬停提示、第二个是一个允许我们经过一定时间后调用函数的调度函数库、第三个的例子是一个功能齐全的DKP拍卖师。

所有这些插件都使用了大量的脚本处理程序和事件。我们处理了很多事件，但只处理了几个脚本处理程序。这是因为大多数脚本处理程序都与图形用户界面相关。这就引出了我们的下一个主题，在《魔兽世界》中创建用户图形界面元素。