3.9 事务

3.9.1 响应时间

事务是指用户在客户端做一种或多种业务所需要的操作集，通过事务函数可以标记完成该业务所需要的操作内容；另一方面事务可以用来统计用户操作的响应时间，事务响应时间是通过记录用户请求的开始时间和服务器返回内容到客户时间的差值来计算用户操作响应时间的，如图 3.159 所示。

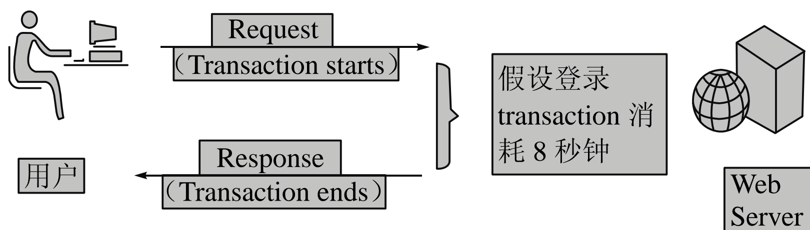


图 3.159 事务响应时间计算方式

注 意

这里的响应时间不包含客户端 GUI 时间（例如浏览器解释页面所消耗的时间）。

前面说响应时间是服务器返回和用户请求发出之间的时间差，那么得到这个时间就够了吗？

例如：现在有一场跑步比赛。当比赛完成后，可以得到每位运动员跑完整个比赛所需要消耗的时间，现在需要分析谁的起跑好、谁的冲刺好，能分析出来吗？答案是不能，虽然得到了最重要的完成比赛的响应时间，但是这对分析和优化几乎没有作用，因为只知道了结果而不知道过程。跑步的时间是由起跑、中途、冲刺等时间组成的，如果想要进行分析优化，必须先了解各个阶段所花费的时间和速度以及各个运动员的优缺点。

对于软件来说，通过事务得到的系统响应时间也是由非常多的部分组成的，一般来说响应时间由网络时间、服务器处理时间、网络延迟三大部分组成。先来看看当一个客户端发出请求到服务器返回需要经历哪些路径，如图 3.160 所示。

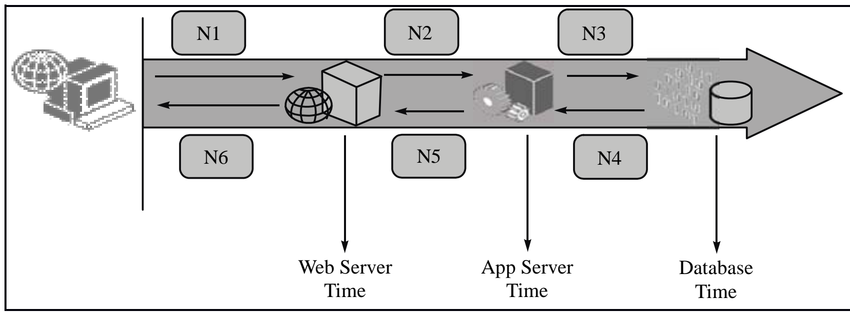


图 3.160 事务响应时间组成

1．网络时间

客户端发出请求首先通过网络来到 Web Server 上（消耗时间为 N1）；然后 Web Server 将处理后的请求发送给 App Server（消耗时间为 N2）； App Server 将操作数据指令发送给 Database （消耗时间为 N3）；Database 服务器将查询结果数据发送回 App Server（消耗时间为 N4）；App Server 将处理后的页面发给 Web Server（消耗时间为 N5）；最后 Web Server 将 HTML 转发到客户端（消耗时间为 N6）。这里的 N*x* 都是网络传输上的时间开销，没有计算业务处理所需要花费的时间。

2．服务器处理时间

另外一个方面还要考虑各个服务器处理所需要的时间 WT、AT、DT。

3．网络延迟

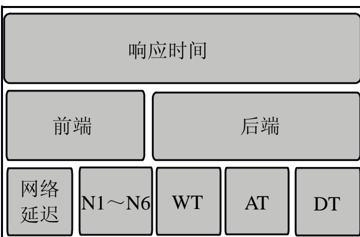


图 3.161 事务响应时间组成详解

除了上面两种时间开销以外，还要考虑网络延迟的问题。

所以最终的响应时间组成为：

响应时间 ＝ 网络延迟时间 ＋WT+AT+DT＋（N1+N2+N3）＋（N4+N5+N6）＋WT+AT+DT

也可以简单认为响应时间由网络开销（前端）和服务器端开销（后端）两大部分组成，如图 3.161 所示。

那么这些消耗的时间都花在什么事情上了呢？影响网络的因素一般包括以下内容：

1．前端 Network

- DNS Lookup
- Time to connect
- Time to first buffer
- Network Time
- Download Time
- SSL handshake
- FTP authentication
- Client Time

- Error Time
- 网络延迟

2. 后端服务

- Web Server
  - Servlet Time
  - Method Time
  - 静态动态压缩
- App Server
  - EJB Time
  - Method Time
  - JNDI Lookup
- Database Server
  - JDBC Time
  - Connect Time
  - Execute Time

这里会发现响应时间的组成是非常复杂的，当性能问题出现时，想要定位到具体的代码级别是相当困难的。

3.9.2 添加事务

通过事务监控响应时间，需要做的就是在请求的发出前添加一个事务开始的计数器，在请求结束的地方添加一个事务结束的计数器，VuGen 会自动计算函数间的时间差。

通过工具栏上的事务按钮即可完成事务的添加操作，在请求之前单击插入事务开始按钮，如图 3.162 所示。

在弹出对话框中填写事务名称 mainpage，如图 3.163 所示。

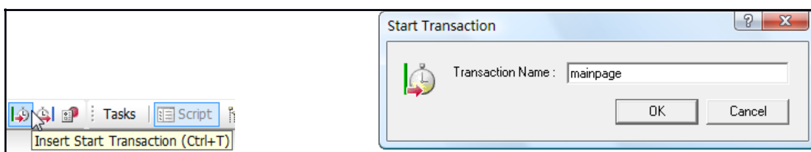


图 3.162 插入事务开始按钮　　　　图 3.163 设置事务名称

单击 OK 按钮后，可以得到事务开始的函数：

```
lr_start_transaction("mainpage");
```

通过这条语句开始了一个叫做 mainpage 的事务，然后在请求后添加一个事务结束函数，单击插入事务结束按钮，如图 3.164 所示。

这里提供了几个选项，选择 LR_AUTO 由 VuGen 自动判断状态，如图 3.165 所示。

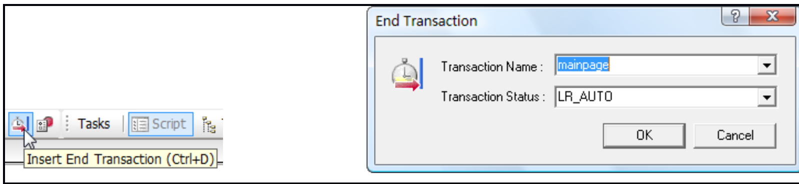


图 3.164　插入事务结束按钮　　　　图 3.165　设置事务状态判断为 LR_AUTO

这样就生成了一个事务结束的函数：

```
lr_end_transaction("mainpage", LR_AUTO);
```

注 意

这里事务的开始和结束名称需要配对。

事务设置结束，通过这两个函数完成了对事务的设计，运行脚本来看看效果。脚本如下所示：

```
Action()
{
                        lr_start_transaction("mainpage");

                       web_url("51testing","URL=http://bbs.51testing.com
",LAST);
                        lr_end_transaction("mainpage", LR_AUTO);
                        return 0;
}
```

在日志中可以看到以下信息：

```
Action.c(3): Notify: Transaction "mainpage" started.
//请求发送返回的相关日志
Action.c(5): Notify: Transaction "mainpage" ended with "Pass" status
(Duration: 5.9749).
```

事务结束的说明中包含了请求所花费的时间为 Duration=5.9749 秒和事务结束的状态。通过事务可以获得每个操作所消耗的准确时间，例如查询、登录、删除操作。但是对于性能分析来说，这个时间还是太大了，无法有效地帮助我们定位性能瓶颈，LoadRunner 能解决这个问题吗？抱歉，LoadRunner 只能对自己发出的请求和服务器返回的内容进行网络级别的分析，也就是说 LoadRunner 能够分析的时间为客户到 WWW 服务器的时间 N1 和 WWW 服务器返回到客户的时间 N6。这些时间主要和网络速度有关，可以用一个 LoadRunner 的名称来解释，叫做 Web Page Breakdown，相关的具体分析可以参考第 5.3.5 节。

也就是说 VuGen 可以分析的时间只有客户端到 Web Server 之间的部分，后面从 Web Server 到 App Server 再到 Database Server 的时间只能得到一个总和。

事务状态

在默认情况下使用 LR_AUTO 来作为事务状态，对于一个事务有以下 4 个状态可以选择。

1．LR_AUTO

LR_AUTO 是指事务的状态由系统自动根据默认规则来判断，结果为 PASS/FAIL/STOP。

2．LR_PASS

LR_PASS 指事务是以 PASS 状态通过的，说明该事务正确地完成了，并且记录下对应的时间，这个时间就是指做这件事情所需要消耗的响应时间。

3．LR_FAIL

LR_FAIL 是指事务以 FAIL 状态结束，该事务是一个失败的事务，没有完成事务中脚本应该达到的效果，得到的时间不是正确操作的时间，这个时间在后期的统计中将被独立统计。

4．LR_STOP

LR_STOP 将事务以 STOP 状态停止。

事务的 PASS 和 FAIL 状态会在场景的对应计数器中记录，包括通过的次数和事务的响应时间，方便后期分析该事务的吞吐量以及响应时间的变化情况。

事务和子事务

在 VuGen 中可以通过事务来完成一组操作的响应时间监控，如果想监控一个事务中某一步操作的响应时间，就需要使用子事务来完成（当然也可以直接使用事务嵌套）。

```
Lr_start_sub_transaction("子事务名","父事务名");
Lr_end_sub_transaction("子事务名","子事务状态");
```

注意

子事务虽然和父事务很像，但是父事务支持的很多函数，在子事务中都无法实现，所以应酌情考虑。

例如：需要做一个登录的事务，想知道打开登录页面和登录操作的时间，就可以使用子事务。

```
lr_start_transaction("login");
   lr_start_sub_transaction("loginpage","login");
   //打开登录页面
   lr_end_sub_transaction("loginpage",LR_AUTO);

   lr_start_sub_transaction("submitlogin","login");
   //提交登录表单
   lr_end_sub_transaction("submitlogin",LR_AUTO);
lr_end_transaction("login",LR_AUTO);
```

这样可以得到 3 个事务的时间，并且清楚地得到打开页面和登录操作以及整个操作的

时间。

事务相关的函数

```
lr_get_transaction_duration("事务名")
//获得对应事务达到该函数运行位置时持续的时间，返回 double 类型
lr_get_transaction_wasted_time("事务名")
//获得对应事务达到该函数运行位置时的 wasted 时间，返回 double 类型
lr_wasted_time(毫秒)
//为一个事务添加 wasted 时间，无返回
lr_stop_transaction("事务名")
//将一个事务暂停，该函数后的操作都不会被记入事务时间
lr_resume_transaction("事务名")
//将暂停的事务恢复
```

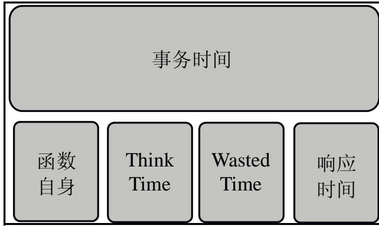
一般情况下都可以十分简便地获得请求的相应时间，但是对于下载操作来说就并不是那么方便了，这个时候需要利用一个 web_get_int_property()函数来解决。例如想获得一个下载操作的响应时间，可以这样写：

```
Action()
{
int i;
lr_start_transaction("download");
web_url("LoadRunner","URL=http://127.0.0.1/loadrunner.iso",LAST);//下载操作的请求
i=web_get_int_property(HTTP_INFO_DOWNLOAD_SIZE);
if(i>5000)//当下载的文件大小大于 5000 个字节时认为下载成功，否则失败
lr_end_transaction("download",LR_PASS);
else
lr_end_transaction("download",LR_FAIL);
}
```

当进行下载操作时，服务器会返回该下载操作的文件大小，所以只需要获得该信息即可，如果还希望对下载时的速度进行测试，那么需要配合关联函数将返回的内容保存到本地才能实现。一般来说对于下载操作的性能测试集中在需要多少时间弹出下载对话框，而后面的内容其实是属于网络带宽的问题了。另一方面现在很少使用 IE 进行直接下载，而采用迅雷一类的 P2P 下载工具，所以很少考虑具体下载文件操作对网络和服务器读写操作的影响。

3.9.3 事务时间

一个事务的时间是指持续时间，事务会完全记录下从事务开始到事务结束之间的时间差，那么事务的时间能真实地反映业务操作的时间吗？不能，就好像人用手按秒表来记录短跑时间一样，得出的时间并不是完全准确，存在观察的误差和操作的误差，对于一个事务时间来说，一般由四部分组成，如图 3.166 所示。



响应时间

这是事务的目的，通过事务记录业务操作所消耗的响应时间。

事务自身时间

事务中哪怕没有操作，也是需要时间的，不过这个时间一般在 0.01 秒左右，所以可以忽略。

```
lr_start_transaction("thinktime");
lr_end_transaction("thinktime", LR_AUTO);
```

运行上面的脚本后，可以看到：

```
Action.c(5): Notify: Transaction "thinktime" started.
Action.c(9): Notify: Transaction "thinktime" ended with "Pass" status (Duration: 0.0121).
```

思考时间（Think Time）

Think Time 是 LoadRunner 提供的一种模拟用户等待的方式，通过 lr_think_time()函数实现。在函数内写入对应的时间（单位是秒），当脚本在 Controller 中运行到该函数时就会等待相应的时间。注意在 VuGen 中，回放 Think Time 默认关闭。

Think Time 在进行性能测试的时候需要打开，只有这样每个虚拟用户才是真正按照用户的操作速度来完成请求，才能得到在真实情况下的系统数据。如果不打开 Think Time，测试获得的数据是在全负载下的一些理论峰值数据。

那么 Think Time 在事务中如何影响事务时间呢？编写如下脚本：

```
lr_start_transaction("thinktime");
lr_think_time(5);
lr_end_transaction("thinktime", LR_AUTO);
```

在 Run-time Settings 中设置 Think Time，启用 Replay Think Time 功能，运行之后可以看到以下结果：

```
Action.c(5): Notify: Transaction "thinktime" started.
Action.c(7): lr_think_time: 5.00 seconds.
Action.c(9): Notify: Transaction "thinktime" ended with "Pass" status
```

```
(Duration: 5.0254 Think Time: 4.9995).
```

所以 Think Time 会被算在事务的时间内，不过在 Analysis 中可以设置过滤规则将其扣除，另外我们也建议尽量不要在事务内使用 lr_think_time()函数。

浪费时间（Wasted Time）

在使用事务的时候，经常会看到在事务日志中有 Wasted Time。Wasted Time 是指事务中应该扣除的由于其他原因导致的时间浪费。在默认情况下 LoadRunner 会将自身脚本运行浪费的时间自动记入 Wasted Time。例如执行关联、检查点等函数的时间。

除了脚本自身浪费的时间，某些时候使用 C 语言等外部接口进行处理所消耗的时间也会影响事务的时间，而这个时间 LoadRunner 无法处理，在这种情况下就需要人为地计算第三方时间开销，并且将这个开销的时间记入 Wasted Time 中。

运行一下下面的代码：

```
Action()
{
    int i;
    int baseIter = 100;
    char dude[1000];
    merc_timer_handle_t timer;
    // Examine the total elapsed time of the action
    //Start transaction
    lr_start_transaction("Demo");
                           timer=lr_start_timer();
                           for (i=0;i<=baseIter*1000;i++) {
                                sprintf(dude,"This is the way we waste time
in a script = %d", i);
                             }
    wasteTime=lr_end_timer(timer);
    lr_wasted_time(wasteTime*1000);
    lr_end_transaction("Demo", LR_AUTO);
    return 0;
}
```

其中，lr_start_timer()是一个 LoadRunner 自带的时间计数器，它和 lr_end_timer()相对应，能够返回这两个函数间的时间差。

运行脚本后，等待一段时间脚本运行结束，可以看到以下日志。

```
Action.c(18): Notify: Transaction "Demo" started.
Action.c(27): wasted time is 85.860000
Action.c(28):Notify:Transaction "Demo" ended with "Pass" status (Duration:
85.8772 Wasted Time: 85.8600).
```

通过上面这个日志可以看到，在 VuGen 运行脚本的时候这个 1000 次的 C 语言操作所消耗的时间会被算在 Transaction 时间内，导致 Transaction 的时间变长。当通过 lr_start_timer()

计时函数将这个消耗时间加入 Wasted Time 后，这个脚本就能正确地计算出事务的时间和该事务时间的 Wasted Time 了。当在场景中运行的时候，事务的响应时间会自动扣除 Wasted Time。

为了确保响应时间的正确，需要扣除在运行脚本时自身的时间消耗，事务中尽量避免出现非请求的处理内容，如果无法避免请使用 lr_wasted_time()函数将多余的时间开销扣除。

例如这样的脚本：

```
merc_timer_handle_t timer; //变量声明
lr_start_transaction("Demo");
timer=lr_start_timer();
lr_load_dll("getkey.dll");
lr_save_string(getrandkey(),"key");
//通过调用 dll 获得密钥
wasteTime=lr_end_timer(timer);
lr_wasted_time(wasteTime*1000);
lr_end_transaction("Demo", LR_AUTO);
```

计算密钥是很消耗时间的，那么可以使用 timer 这个变量来记录计算的时间，并将这个时间从整个事务中扣除。

注 意

在计算 Wasted Time 时不要直接使用 lr_wasted_time()覆盖，而忘了加上脚本中 LoadRunner 函数的自身时间。通过 lr_get_transaction_wasted_time()函数可以获得事务自身的 Wasted Time，将这个时间累加上第三方统计的 Wasted Time 再通过 lr_wasted_time()函数覆盖。

3.9.4 手工事务

前面都是使用 LR_AUTO 来自动判断事务状态，现在来做一个脚本，看看 LoadRunner 的事务是如何自动判断状态的。

录制一个论坛注册用户的脚本，在提交注册表单处添加事务开始及结束标志，然后回放该脚本。事务的结果是 PASS 还是 FAIL 呢？虽然回放脚本注册用户是失败的（该用户已经存在），但是事务还是在 PASS 状态下完成了，而且会发现事务的持续时间很短。正常情况下注册一个用户到刷新首页一般都要 2 秒，现在只需要 0.3 秒。这是因为当服务器判断到该用户已存在后，就没有了数据插入和等待 1 秒刷新首页的操作，而是直接返回错误提示页面。这个 0.3 秒是系统处理错误的时间而不是注册用户所需要的时间。

LR_AUTO 也是根据服务器的返回状态信息来决定事务是以 LR_PASS 状态通过还是以 LR_FAIL 状态结束，只要服务器返回页面，那么事务就会认为请求成功发出去了，服务器看懂了请求也返回了内容，自然事务是 PASS 状态了。

这样由于事务自动判断的错误，导致虽然操作是失败的，但得到了一个响应时间，并

且这个响应时间又没有正确反映出做这件事情的真正时间，最终就会影响到性能测试得到的数据。

记得在论坛上就有朋友问过这样的问题，为什么系统在用户越来越多的情况下，响应时间不增反减？这种现象很有可能就是没有使用手工事务导致的结果。

对于这种情况就需要手工来判断操作是否成功，通过 web_reg_find()检查点函数来检查页面是否返回正确，然后通过 rowcount 的参数值来进行事务状态判断，做到智能判断事务结果。

例如：检查点函数的 rowcount 保存在参数 loginst 中，那么事务的状态就应该这样判断：

```
lr_start_transaction("login");
web_reg_find("Search=Body",
                            "SaveCount=loginst",
                            "Text=登录失败",
                            LAST);
//登录请求
If(atoi(lr_eval_string("{loginst}"))>=1))
                            lr_end_transaction("login", LR_FAIL);
else
                            lr_end_transaction("login",LR_PASS);
```

通过检查点来检查登录后页面是不是存在“登录失败”这样的内容，如果存在那么 loginst 的值就大于等于 1，然后把 loginst 的值取出来和 1 做比较，如果大于 1 那么就是登录失败，否则就是登录成功。

注 意

参数不能和值做比较，所以要先通过 lr_eval_string()函数将其转化成字符串，然后再通过 atoi()函数转化成整数，这样才能和 1 作比较。

在绝大多数情况下对于事务都需要采用手工事务的方式来确保事务的正确性和事务时间的有效性。

思考题：

对于 Discuz 论坛来说如何做一个有效的用户注册脚本通过手工事务并且获得准确注册操作的响应时间。

业务分析：

注册用户后，在系统的页面上会出现【欢迎：注册用户名】的信息，可以在注册后返回的页面中检查是否出现了这样的内容来判断注册事务是否成功。

通过检查页面可以得到需要判断的代码为：

```
欢迎:<a class="dropmenu" id="viewpro" onmouseover="showMenu(this.id)">
```

所以在检查点函数中需要添加这个内容，为了更好地判断，还需要把注册用户的名字

也加进去，最后可以得到下面的代码：

```
Action()
{
                    web_url("注册",
                        "URL=http://192.168.0.200/register.aspx",
                        "TargetFrame=",
                        "Resource=0",
                        "RecContentType=text/html",
                        "Referer=http://192.168.0.200/",
                        "Snapshot=t2.inf",
                        "Mode=HTML",
                        EXTRARES,

                    "URL=/templates/default/images/check_error.gif",
ENDITEM,

                    "URL=/templates/default/images/check_right.gif",
ENDITEM,
                        "URL=/images/level/3.gif", ENDITEM,
                        LAST);

                    lr_start_transaction("reg");

                    web_reg_find("Search=Body",
                        "SaveCount=regst",
                        "Text=   欢   迎   :<a   class=\"dropmenu\"
id=\"viewpro\" onmouseover= \"showMenu(this.id)\">{username}",
                        LAST);

                    web_submit_data("register.aspx",

                    "Action=http://192.168.0.200/register.aspx?create
user=1",
                        "Method=POST",
                        "TargetFrame=",
                        "RecContentType=text/html",
                        "Referer=http://192.168.0.200/register.aspx",
```

```
"Snapshot=t11.inf",
"Mode=HTML",
ITEMDATA,
"Name=username", "Value={username}", ENDITEM,
"Name=password", "Value=112212", ENDITEM,
"Name=password2", "Value=112212", ENDITEM,
"Name=email",  "Value={username}@cloud.chen",
ENDITEM,
"Name=submit", "Value=创建用户", ENDITEM,
"Name=question", "Value=0", ENDITEM,
"Name=answer", "Value=", ENDITEM,
"Name=realname", "Value=", ENDITEM,
"Name=idcard", "Value=", ENDITEM,
"Name=mobile", "Value=", ENDITEM,
"Name=phone", "Value=", ENDITEM,
"Name=gender", "Value=0", ENDITEM,
"Name=nickname", "Value=", ENDITEM,
"Name=bday_y", "Value=", ENDITEM,
"Name=bday_m", "Value=", ENDITEM,
"Name=bday_d", "Value=", ENDITEM,
"Name=location", "Value=", ENDITEM,
"Name=msn", "Value=", ENDITEM,
"Name=yahoo", "Value=", ENDITEM,
"Name=skype", "Value=", ENDITEM,
"Name=icq", "Value=", ENDITEM,
"Name=qq", "Value=", ENDITEM,
"Name=homepage", "Value=", ENDITEM,
"Name=bio", "Value=", ENDITEM,
"Name=templateid", "Value=0", ENDITEM,
"Name=tpp", "Value=0", ENDITEM,
"Name=ppp", "Value=0", ENDITEM,
"Name=newpm", "Value=radiobutton", ENDITEM,
"Name=pmsound", "Value=1", ENDITEM,
"Name=showemail", "Value=1", ENDITEM,
"Name=receivesetting", "Value=2", ENDITEM,
"Name=receivesetting", "Value=4", ENDITEM,
"Name=invisible", "Value=0", ENDITEM,
"Name=signature", "Value=", ENDITEM,
```

```
        "Name=sigstatus", "Value=1", ENDITEM,
        LAST);

    if(atoi(lr_eval_string("{regst}"))>=1)
        lr_end_transaction("reg", LR_PASS);
    else
        lr_end_transaction("reg",LR_FAIL);
    return 0;
}
```

这里的{username}是一个参数，用来存放注册的用户名，在参数列表中设置了该参数的取值方式和信息。

3.10 集合点

集合点函数可以帮助我们生成有效可控的并发操作。虽然在 Controller 中多用户负载的 Vuser 是一起开始运行脚本的，但是由于计算机的串行处理机制，脚本的运行随着时间的推移，并不能完全达到同步。这个时候需要手工的方式让用户在同一时间点上进行操作来测试系统并发处理的能力，而集合点函数就能实现这个功能。集合点只需要在脚本中插入 lr_rendezvous()函数即可。打开 Insert 菜单下的 Rendezvous 选项，如图 3.167 所示。

在弹出的对话框中输入集合点名称 run，确定后即可得到对应的脚本：

```
lr_rendezvous("run");
```

引号内的就是集合点名称，当脚本在多用户运行的情况下，每次运行到这个函数都会查看一下集合点的策略来决定是等待还是继续运行。集合点的设置内容存放在场景的设置中，当脚本中有集合点函数时，场景中的集合点设置功能就可以访问，如图 3.168 所示。

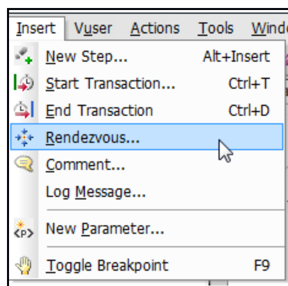


图 3.167 添加集合点函数

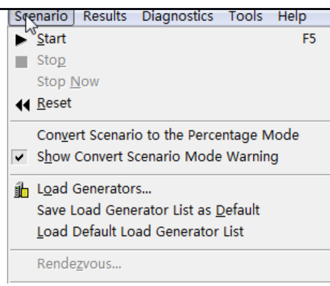


图 3.168 场景中的集合点设置

打开场景菜单下的集合点后，可以为集合点进行设置，包括哪些用户使用该集合点、集合点是否有效等，如图 3.169 所示。

如果脚本中没有集合点，那么场景中的 Scenario/Rendezvous 集合点功能将会是灰色显示。

集合点策略用来设置虚拟用户集合的方式，打开 Policy 对话框，如图 3.170 所示。集合点提供了以下 3 种策略：

1．当百分之多少的用户到达集合点时脚本继续。

2．当百分之多少的运行用户到达集合点时脚本继续。

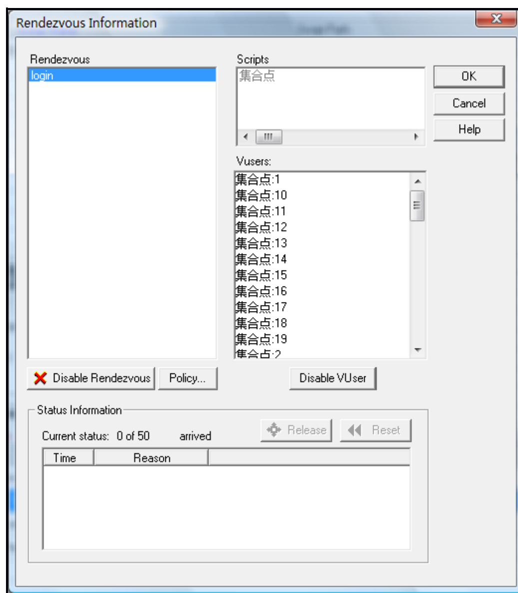


图 3.169　场景中的集合点设置窗口

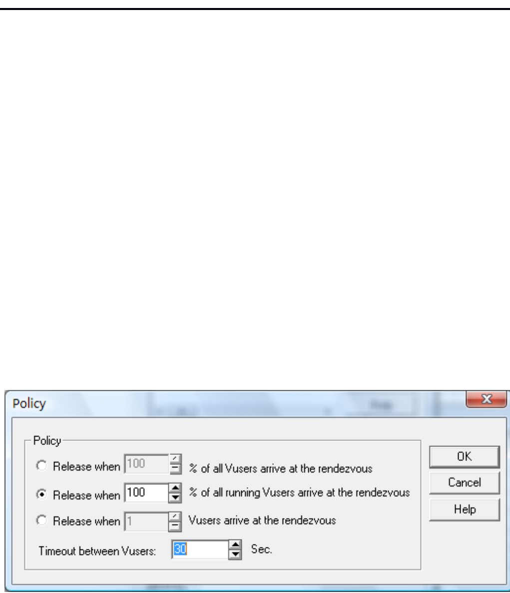


图 3.170　场景中的集合点策略

3．多少个用户到达集合点时脚本继续。

这 3 个策略的区别在于：假设脚本由 100 个用户来运行，但 100 个用户并不是一开始就共同运行的。假设每隔 1 分钟添加 10 个用户，也就是说 10 分钟后系统才有 100 个在线用户。这里 100 就是指系统访问的所有用户数，而不同时间的在线用户数是不同的。设置的集合点策略百分比均为 100%。

在场景运行时，当 Vuser 脚本运行到集合点函数时，该虚拟用户会进入集合点状态直到集合点策略满足后才释放。

策略 1 是指当全部用户都运行到了集合点函数才释放集合，让这 100 个用户并发运行后面的脚本。

策略 2 是指当前时间如果只有 10 个用户在线，那么只需要这 10 个用户都运行到了集合点函数就释放集合，让这 10 个用户并发运行后面的脚本。

策略 3 就比较好理解了，当到达集合点的用户数达到自己设置的数量后就释放等待，并发运行后面的脚本。

可以在多个脚本上设置相同的集合点名称来实现多个脚本同时并发的效果。

集合点超时

在脚本运行时，每个虚拟用户到达集合点时都会去检查一下集合点的策略设置，如果不满足，那么就在集合状态等待，直到集合点策略满足后，才运行下一步操作。但是可能存在前一个虚拟用户和后一个虚拟用户达到集合点的时间间隔非常长的情况，所以需要指定一个超时的时间，如果超过这个时间就不等待迟到的虚拟用户了。

超时时间是指虚拟用户之间的时间差，当出现两个虚拟用户到达集合点的时间差超过设定的超时时间时，所有在集合点处于等待状态中的用户将全部释放。

集合点和事务

集合点应该放在事务外，如果事务内存在集合点，那么虚拟用户在集合点等待的过程也会被算入事务时间，导致早进入集合点的用户的响应时间有误。

常见的田径比赛就是这样，大家先集合在同一起跑线上，鸣枪后开始计时，达到终点再计时，这样就能得到准确的事务时间。

小结

VuGen 是性能测试开始的第一步，也是性能测试最重要的一个步骤，如何形成一个有效的负载脚本，决定了性能测试的实施分析能否在一个坚强可靠的基础上进行。本章详细介绍了 VuGen 的使用方式和脚本开发的各个要点。

脚本开发是一个体力活，通过自行编写一个脚本对服务器进行负载，而脚本开发又基于需求，在掌握用户真正的行为特征后，即可根据其开发性能测试脚本。另外一个方面，根据性能测试的目的，脚本可以分为两大类：一类是针对某个具体功能的调试型性能测试脚本，这种脚本的操作尽可能少，主要是为了测试某一个功能而开发的；另一种是模拟用户行为的脚本，该脚本完全模拟了用户的常见操作及逻辑分支，主要是为了进行稳定性测试或系统级别的性能测试而开发的。

在完成了脚本的开发后，接着可以开始设计场景完成性能测试负载工作了。在下一章中我们将会介绍如何使用 Controller 进行性能测试场景设计、负载生成及数据监控。

本章需要掌握的重点：

- VuGen 录制脚本的录制等级和设置方式
- VuGen 脚本录制的流程及生成脚本的原理
- 如何设置脚本的 Runtime Settings，特别是 Run Logic（运行逻辑）
- 参数化原理及各种参数取值的方式
- 关联原理及如何准确获得服务器返回的部分内容
- 事务时间和响应时间的组成
- 手工事务的使用方法
- 集合点策略