本章內容：

- 號誌
- Wait 的統計數據
- 參考指南

2

Waits

在一個 Oracle instance 中，有許多行程（或單一行程的多個執行緒）是相互運作的。為了彼此間的通力合作，行程必須能夠溝通，而其中一個主要的溝通方式是透過號誌（semaphore）。號誌即為信號（signal）。它有點像鐵路的信號，告知火車該停車、候車，以及何時行駛。基於下列因素，Oracle 的伺服器行程（server process）經常必須停下來等候：

- 有時是因為無可用資源。
- 有時是因為沒有工作可做。
- 有時是因為它們得等待另一個伺服器行程執行必須先完成的工作。

號誌會讓 Oracle 的伺服器行程停下來並等待，然後在行程需要繼續進行原先的工作時，通知行程。

號誌

每一個 Oracle 的伺服器行程都有一個號誌。行程會在無可用資源，沒有工作可執行，或有必須先行執行的工作時，等候本身的號誌。一旦資源被釋放，或有工作可做，或必須先執行的工作已完成時，其號誌會以信號的方式通知（post）行程，以停止等待。

例如，當一個使用者行程準備將重做（redo）資訊拷貝到重做日誌緩衝區（redo log buffer）時，LGWR（Log Writer process，日誌寫入行程）可能因為當下沒工作可做，正等候其號誌。當使用者下達 commit（確認，完成交易的執行）時，LGWR 必需將 redo 及 commit 記號（marker）寫入日誌檔（此時使用者尚在等待中）。為了完成這件事，使用者行程會通知 LGWR 的號誌，告知 LGWR 不必等了，因為目前已有工作要做。接著，使用者行程會等待自身號誌。當日誌檔的 I/O 完成時，LGWR 便會通知使用者行程的號誌，告訴它現在可以開始下一個交易，因為 commit 操作已完成。接著，LGWR 會再度等待本身號誌，因為它手上已無工作了。

另一個例子是：行程 A 想更新一筆資料列，但發現行程 B 尚未 commit 先前對該筆資料列的更新，因此，行程 A 必須等待行程 B 的 commit。為了辦到這件事，行程 A 必須等待本身號誌。當行程 B commit 時，它會通知行程 A 的號誌，告知行程 A，現在可以更新該筆資料了。

號誌的機制

號誌屬於作業系統的機制。當一個 Oracle 行程在等待本身號誌時，作業系統並不會將該行程列入 CPU 的排程（schedule）。以作業系統的術語來說：行程被堵塞（blocked），而非可執行（runnable）狀態。當號誌被通知時，該行程的作業系統狀態會從堵塞轉為可執行，並被列入排程，以便儘快執行。

有些作業系統支援超過一種的號誌。System V 的號誌最為普遍。System V 的號誌之資料結構，為核心記憶體中的固定陣列（fixed array），其陣列大小由 SEMMNS 核心參數所控制。行程必須用 semop 系統呼叫，來通知或等待號誌。因為號誌被實作於作業系統的核心內，所以，System V 的號誌承受了因大量且不必要的系統呼叫，所帶來環境切換（context switch）的負擔，以及因存取核心資料結構的有序化（serialization）需求，而導致的低延展性。

為求更好的效能及延展性，許多作業系統支援了其它不同的號誌操作。這些號誌被實作於虛擬裝置驅動程式（pseudo device driver）中，稱為 post-wait driver（「通知 — 等待」驅動程式）。由於這些號誌的資料結構位於使用者記憶體中，而非核心記憶體，因此，虛擬裝置驅動程式可在使用者環境（user context）中使用。這減少了因系統呼叫而導致的環境切換，也改善了延展性，不過，這會因系統的不同而有所差異。

POSIX 的即時系統的擴充小組委員會（real-time extension subcommittee）意識到，制定使用者記憶體的號誌機制之標準的必要性，他們在 POSIX.1b 標準（即先前的 POSIX.4）中，制定了相關的介面和實作需求，於是，一個集優雅、效率和移植性的號誌機制問世了。現在，POSIX.1b 號誌的蹤跡，已遍佈於許多作業系統中。

使用哪一種號誌機制，端視作業系統和版本而定。若你的 Oracle 安裝手冊提到 SEMMNS 參數之設定，那麼預設是用 System V 的號誌。不幸地，多數作業系統都是如此。附帶一提，一般建議將 SEMMNS 值設為 200，但這並未考慮到 Oracle 行程數目的規劃，或其它系統或應用軟體之需求，所以，200 是欠缺深思熟慮的。除了表格 2-1 所詳加解釋的需求外，你得讓每一個 Oracle 的伺服器行程擁有一個號誌。

你還應該知道，在一些平台中，每一個 Oracle instance 得將所屬號誌，配置於單一的號誌集（semaphore set）裡。所以，SEMMNI 參數只消讓每一個 instance 擁有一個號誌識別碼（semaphore identifier），而 SEMMSL（若有定義）必須不小於所有 instance 中最大的 PROCESS 參數。此時，啟動（enable）向量式通知（vector post）是必要的。向量式通知主要為關鍵的背景行程（像是 LGWR、DBWn 等）所用，以通知在單一號誌操作中多個等待中的行程。是否使用向量式通知，取決於 _USE_VECTOR_POSTS 參數。

此外，若你在作業系統中定義了 SEMMNU 這個核心參數，那麼，其值應大於整個系統規劃中並行號誌之數目。對於許多用到號誌的客戶端行程之系統來說，其預設值可能不敷使用。果真使用預設值，那麼號誌的操作會在系統的尖峰時期，發生斷斷續續的失敗，並得到 ORA-7264 或 ORA-7265 的錯誤訊息。為了避免發生這種情況，SEMMNU 參數必須設成，至少需與「CPU 個數」加上「CPU 在系統尖峰時期執行之佇列的長度」之總合相等。

隱藏參數

大凡以底線開頭的參數（如 _USE_VECTOR_POSTS），皆為隱藏參數（hidden parameter）。這種參數無法在 V$PARAMETER 中尋得，或藉由 SHOW PARAMETER 指令取得，因為它們是隱藏的。通常你也無法從 Oracle 文件中找到相關解說，因為它們並未公開。然而，你可以用我的 *hidden_parameters.sql* APT 指令稿，來取得相關描述，並使用 *all_parameters.sql* 指令稿，來檢視這些隱藏的參數值。

有些隱藏參數是特定作業系統才有的。而一些只在不尋常的回復（recover）狀況中才會用到。某些則是用來取消或啟動新功能。許多隱藏參數與罕見的效能議題有關。由於和所有未公開的新功能有關，隱藏參數可能會隨著版本的更新而消失或改變。所以，不到最後關頭，決不輕言使用，除非你已和 Oracle 的客服部門商議過才可使用，並將相關問題好好地訴諸於文件，以利往後交接。

表 2-1 System V 的號誌參數

參數	描述
SEMMNS	系統的號誌數目。除了作業系統和其它軟體的需求外，你還應該讓每一個 Oracle 的伺服器行程，至少擁有一個號誌；亦即系統內所有 Oracle instance 的 PROCESS 參數之總合。若用到號誌的客戶端行程，並不一定會依嚴格的順序來啟動和關閉，那麼我會建議 SEMMNS 得額外再加上一些數量，即行程在號誌的單一需求中之最大量。 此外，控制單一具名之使用者，在同一時間所能擁有之行程數目上限值的核心參數（通常是 MAXUP），至少要等於 SEMMNS 的設定值，好讓其它名為 oracle 的使用者（不需號誌）所擁有之管理行程得以共存。然而，本參數不應太大，而導致另一使用者的多個行程，會有將核心行程表格（kernel process table）塞爆之虞。因此，控制所有使用者可同時執行之行程數目上限的核心參數（通常是 NPROC），應至少是 SEMMNS 值的 3 倍。
SEMMSL	單一號誌集之大小限制。本參數在一些作業系統上並未定義。若已定義，Oracle 會要求 instance 內的所有號誌，皆配置到同一號誌集，本參數必須至少等於任何 instance 之 PROCESSES 參數的最大值。
SEMMNI	系統中號誌集識別碼的數目。除了作業系統和其它軟體之外，你應該讓每個 instance 有一個識別碼，甚至更多 —— 若 SEMMSL 參數被設為允許 instance 可擁有超過一個以上的號誌集。

參數	描述 （續）
SEMMNU	系統中，號誌復原（undo）之資料結構的數目。復原結構用來回復意外死亡行程（行程掛掉時，正在使用號誌）之核心號誌的資料結構。SEMMNU 應大於尖峰時期，執行中（running）和可執行（runnable）之行程的數目。

如果 Oracle 在你的作業系統中，預設上採用 System V 號誌，但也支援 post-wait driver，那你應改用 post-wait driver。這通常是將 USE_POST_WAIT_DRIVER 參數設為 TRUE（有時還得設定 POST_WAIT_DEVICE）。請查閱你的 Oracle 安裝手冊，因為相關指令是依作業系統和版本而定。

若你的安裝手冊壓根兒沒提到核心號誌參數或 post-wait driver 的設定，那可以斷定號誌機制的選擇和組態，在你的作業系統裡是自動化且不需調校。

請注意，號誌參數是作業系統的核心參數，無法在 Oracle 用來初始化參數的檔案（*INIT.ORA*）裡被設定。

排程延遲

當一個行程被通知（post）時，它的作業系統狀態會從堵塞轉變成可執行。然而，這並不表示該行程會馬上被排去使用 CPU。它至少須等到作業系統的排程行程被執行時，才有機會，若有高優先權的行程也在等待執行，那麼該行程恐怕還得等更久。從行程被通知，到開始執行的這段時間，稱之為排程延遲（scheduling latency）。排程延遲會影響 Oracle 的反應時間（如圖 2-1 所示），因此，減少排程延遲是效能調校中重要的一環。

許多作業系統的排程演算法，會根據行程最近使用 CPU 的時間量，來調整其執行優先權。在極為忙錄的 Oracle 環境？，這會產生一個很不好的影響，例如，LGWR、DBWn、LCKn，以及 LMDn 等關鍵的背景行程，其執行優先權會被降低，因而增加這些行程的排程延遲；也可能因為這樣，而造成整個 instance 的瓶頸。

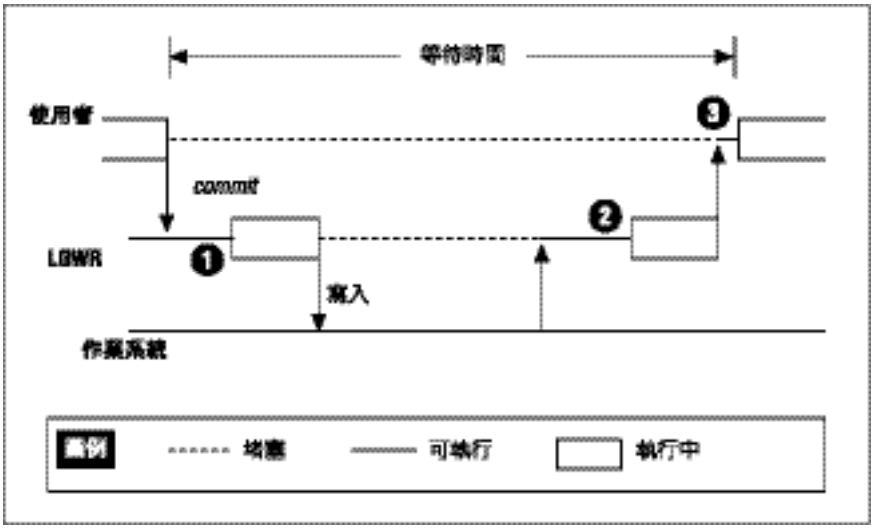

圖 2-1 一個 commit 過程中的三個排程延遲

有些作業系統支援多重排程演算法。如果可以，你應該選擇一個不會降低行程優先權的排程演算法。如果不行，你的作業系統不妨提供固定優先權的機制。如果行程的優先權能固定，就不會發生優先權降低的情形。在一些案例中，所有使用者的優先權皆可固定，而 Oracle 會自動使用。有些案例則是只有系統管理員與具有特別權限之使用者才能辦到。若是這種情況，則名為 oracle 的使用者應被賦予這樣的權限，或是系統管理員應從固定優先權的 command shell 中，啟動 Oracle instance，如此一來，所有 Oracle 行程的優先權就都是固定的。

若無固定優先權的功能，則避免優先權降低的替代方案還有：手動提高關鍵背景行程的執行優先權，或者，甚至將它們以即時（real-time）的優先權等級執行。Oracle 通常會建議，所有 Oracle 行程的優先權應該都一樣，在這樣的基礎上，你可能很不願意比照上述的替代方案來處理。該建議的立論基礎，是為了避免低優先權的行程，無法釋放所持有的關鍵性資源；這導因於 CPU starvation（飢餓），而其它高優先權的行程會試著不斷地取得資源之故。然而，這樣的理論卻忽略了提昇背景行程的優先權。若這些背景行程無法獲得所需資源，它們會很快地睡去，除此之外，還會依 instance 其它部分所完成之工作量的比例，來消耗 CPU 時間。因此，其它 Oracle 行程不會有 CPU starvation 的風險。

逾時

Oracle 的伺服器行程絕不會永無止境地等下去，以免因為未被通知而陷入無限期的等待。還好，號誌的等待可被中斷。因此，在 Oracle 行程開始等待其號誌時，會藉由設定鬧鈴（alarm clock，或稱之為逾時(timeout)），來安排睡眠的中斷。當行程被通知時，它會關掉鬧鐘，並繼續執行。若超過逾時的時效，其 wait 會被 SIGALRM 信號中斷。如此一來，該行程還有機會重回之前的狀態，並決定是否繼續等待。

舉例來說，一個等待佇列鎖（enqueue lock，請參閱第三章）的行程，在超過等待時間後，可能會執行死結（deadlock）的偵測。若發現了死結，那麼執行中的敘述（statement）會被倒回（roll back），並丟出一個例外（exception）；若無死結，行程會設定一個新的逾時，並開始再次等待該號誌。

有時，行程被通知的時間會非常接近逾時，而鬧鈴會如同行程試著關掉它一般的消失。在這種情況下，Oracle行程會將訊息寫入追蹤檔（trace file）裡：

```
Ignoring SIGALRM
```

若你發現某些追蹤檔裡有這樣的訊息，別怕，沒事。這不過是告訴你，有時等待中的行程不是很快地會被通知。此時，倒是該好好留意 wait 的統計數據（statistics）。

Wait 的統計數據

Oracle 之 wait 的統計數據可真是寶，但也不需過份重視。許多效能問題可經由 wait 的統計數據，很容易地找出。如果 Oracle 等待資源的時間過久（如 latch、空的快取緩衝區、佇列鎖等等），那麼，wait 的統計數據便能幫你揪出問題，並加以量化。經驗告訴我們，你也可以用 wait 的統計數據，來找出網路和磁碟的效能問題。在解決這類問題時，wait 的統計數據提供了有價值的回饋。

但若你的應用程式作了過多解析，或磁碟 I/O 的負荷超出所需的工作量，那麼，wait 的統計數據是幫不上忙的。它們看來似乎是為 instance 的健康開了一張空白支票，也確實如此。但 wait 的統計數據所能反映之效能不彰的問題，卻僅限於資料庫伺服器及其以下等級。因此，對於「因應用程式層級的效能問題，而導致資料庫的負荷增加，但不至於造成效能不彰」的情況沒輒。

然而，在進一步考慮資料庫調校的細節時，你應該已知悉所有應用程式的效能問題。唯有如此，wait 的統計數據對於資料庫的調校才有助益。不過，這些數據必須經過計時，才有百分百的價值。

計時的統計數據

只有當 TIMED_STATISTICS 參數被設為 TRUE 時，wait 才會被計時。之前有人提到：「計時的統計數據，其負擔可被忽略」，且讓我來為這樣的說法背書。你若想說服自己，不妨使用 SQL*Plus 的 SET TIMING ON 指令，來測量「具效能指標意義的查詢」之經歷時間。否則，可在一個閒置（idle）的系統中，分別使用和不用計時（timed）統計數據，做十次以上的量測。你會發現其中的差異微乎其微。

倘若不對統計數據加以計時，Oracle 會記錄每個 wait 開始等待及結束的原因，以及是否發生逾時。但當統計數據的計時生效時，Oracle 會檢查每個 wait 等待前後的時刻，並記錄等候所花的時間（以百分之一秒（厘秒）為單位）。

在 Oracle 8.1.5 中，如果用了 TIMED_STATISTICS 或 SQL_TRACE，那麼 Bug 918002 會抑制 SQL 敘述的共享，並造成 latch 的競用。而在 Oracle 8.1.6 中，若同時使用上述的兩個功能，那麼 Bug 1210242 也會造成同樣的效能問題。

wait 的型態

V$SYSTEM_EVENT 會顯示 wait 和 timeout（逾時）的總合，並累計 instance 生命週期中之所有行程，所紀錄之每種事件（event）的等待時間之總合。它通常會將等待事件依其等候所花時間的總合降冪來排列，作為每種型態的 wait 所帶來之潛在嚴重性的指標。

不過，等候所花時間之總合，必須針對資源的等待才具意義。若行程因沒事而等待，很顯然這種等候時間是不重要的。如果它們所等待的是像磁碟 I/O 的例行工作，那麼等候時間的總合，將視其工作量而定。在這些案例中，等候時間的平均值，比總合還令人感興趣。

wait 的型態可分類為：idle（閒置）wait，routine（例行性）wait 及 resource（資源）wait，這對於 wait 的統計數據之了解非常重要。因此，

APT 有針對 routine wait 及 resource wait（忽略了 idle wait）的指令稿。*routine_waits.sql* 僅顯示每種 routine wait 的平均等候時間。而 *resource_waits.sql*（請參閱範例 2-1）除了會依等候所花時間的總合之降冪順序，顯示所等待的資源種類外，還會顯示等候時間的平均值。

範例 2-1、取自 resource _waits.sql 的輸出樣本

```
SQL> @resource_waits

EVENT                                    TIME_WAITED AVERAGE_WAIT
---------------------------------------- ----------- ------------
write complete waits                         3816218       212.02
buffer busy waits                            1395921        21.79
enqueue                                       503217       529.15
log file switch completion                    144263        90.11
latch free                                     31173         0.61
free buffer waits                              19352       302.38
row cache lock                                   876        73.00
library cache pin                                131        18.71
library cache load lock                           29         2.64
non-routine log file syncs                         0         2.32
```

resource_waits.sql 所回報的平均時間，並非我們想要的。因為逾時的關係，某一資源之單一邏輯的 wait 可能被回報為多個不同的 wait，而這些被回報的 wait 中，除了最後一個，每個都有逾時。邏輯 wait 之數目近似於，等候中的行程被通知以停止等待的次數；意即，不同 wait 的數目，減去發生逾時的 wait 數目。每一個邏輯的 wait 之平均等候時間，對於解決 resource wait 所耗時間而言，是較好的指標（相較於多個不同 wait 的平均等候時間）。因此，除了 latch free wait【譯註】外，指令稿回報了所有的 resource wait。latch free wait（閂釋出延緩）會逾時是正常的，因為 latch wait 的通知（post）屬例外狀況，而非常態。除了 latch 的競用，在逾時之後取得 latch 是正常的。所以，多個不同 wait 的平均等候時間，相較於 latch free wait 的經歷時間，是較好的指標值。

譯註　因 latch 被別的行程所持有，而必須等待 latch 被釋放。

Session wait

V$SESSION_EVENT 會顯示每一個運作中的 session（交談期）之 wait 的統計數據。雖然 wait 對行程的影響比 session 來得大，但因為 session 可在行程間移動，故 wait 會依 session 被記錄（例如，多執行緒伺服器（MTS）的組態）。Session 所累積之 wait 的統計數據有兩個主要用途。

第一，若某使用者突然傳出一個效能不佳的消息，那麼，其 session 之 wait 的統計數據，就可拿來檢驗該使用者的問題。*session_times.sql* 指令稿（請參閱範例 2-2）會針對每種被等待的事件，顯示依 session 所累計的等候時間，以及該 session 耗掉 CPU 時間的總合。這讓我們可以很容易地知道，session 是在執行還是在等候，以及等待的對象（如果 session 是在等候中）。

範例 2-2、取自 session_times.sql 輸出的樣本

```
SQL> @session_times
Enter SID: 29
EVENT                                                    TIME_WAITED
---------------------------------------------------------- -----------
SQL*Net message from client                                  2954196
CPU used by this session                                     1657275
db file sequential read                                       246759
write complete waits                                          139698
buffer busy waits                                              61832
log file sync                                                  32601
enqueue                                                         9576
log file switch completion                                      3530
SQL*Net message to client                                       2214
db file scattered read                                          1879
SQL*Net more data to client                                      952
SQL*Net more data from client                                    908
latch free                                                       840
free buffer waits                                                100
buffer deadlock                                                   57
row cache lock                                                     1
SQL*Net break/reset to client                                      0
```

第二，若等待某一資源的時間過長，那麼，session 之 wait 的統計數據可用來找出，有哪些連線中的 session，涉入等待過久的問題中。APT 指令

稿 *resource_waiters.sql* 可勝任這項工作。它會藉由出問題的資源之 session 的等候時間，顯示故障之處。不會再度活動的 session 之等候時間的總合，也會被顯示。舉例來說，若已有大量的 buffer busy wait【譯註】，那麼看看 session 之 wait 的統計數據，或許可找出問題所在，或將範圍縮減到特定幾個 session。

延緩參數

wait 的統計數據非常有用，因為它會告訴你有哪些 session 處於等待狀態，以及所等待的資源種類。等待對象可能是 latch、資料庫區塊、佇列鎖，或其它資源。將等待對象弄清楚後，才會有正確的調整方向，以免徒勞無功。而延緩參數（wait parameter）比 wait 的統計數據更具價值。延緩參數可精確地告訴你，被等待的是哪個資源 —— 哪一個 latch、哪一個資料庫區塊，或哪一個佇列鎖。wait 的統計數據僅能鎖定問題的範圍，真正能指出問題所在的是延緩參數。

不幸地，延緩參數是難以捕捉的。它們常閃現於 V$SESSION_WAIT 視界（view）中。該視界會顯示每一 session 目前和最近（以及等候期間）的 wait 之延緩參數（如果可以知道）。然而，查詢 V$SESSION_WAIT 是很耗時的（相較於大部分的 wait）。若你在極短的時間間隔內，快速查詢本視界兩次，並注意 SEQ# 欄位（其值會隨不同的 wait 而遞增），你會發現多數等待中的事件，在第二次查詢時已不見了（因為查詢 V$SESSION_WAIT 實在太花時間了），這很普遍。連續重複地查詢 V$SESSION_WAIT 的代價很昂貴，因此，監看延緩參數的用處有限。

幸運的是，延緩參數可在追蹤檔（透過新的 DBMS_SUPPORT 套件所產生），或底層的 10046 事件來觀察。此追蹤和 SQL_TRACE 功能所產生的一樣，但每一行還包含了每個 wait 的延緩參數。

舉例來說，如果 buffer busy wait（緩衝區忙碌延緩）出了問題，那麼你可以用 APT 指令稿 *trace_wait.sql*，來啟動對於倍受影響之 session 的追蹤。接下來，只剩下從追蹤檔中取出 buffer busy wait 的資料，並檢視延緩參數，以找出檔案和被等待區塊之區塊號碼。在 buffer busy wait 的案例中，p1 和 p2 參數為其檔案和區塊號碼。請參閱範例 2-3：

譯註　因所需的緩衝區被其它 session 佔用，所以必須等候緩衝區被釋放。

範例 2-3、trace_wait.sql 的執行片段

```
SQL> @trace_waits

This script uses event 10046, level 8 to trace the event waits in
the top N sessions affected by waits for a particular resource.

Select sessions waiting for: buffer busy waits
Number of sessions to trace: 5
Seconds to leave tracing on: 900

Tracing ... Please wait ...

PL/SQL procedure successfully completed.

SQL> exit
$ cd udump
$ grep 'buffer busy waits' ora_*.trc |
> sed -e 's/.*p1=/  file /' -e 's/ p2=/  block /' -e 's/ p3.*//' |
> sort |
> uniq -c |
> sort -nr |
> head -5
  42    file 2    block 1036
  12    file 24   block 3
  10    file 2    block 1252
   7    file 2    block 112
   6    file 7    block 5122
$
```

每一種等待事件（wait event）的延緩參數之意義，可從 V$EVENT_NAME 中看到，而《Oracle 8i Reference》手冊中也有說明。不過，這是 Oracle 文件中相當脆弱的一節。許多資訊是難以理解、過時或不正確。由於延緩參數對進階的效能調校非常重要，故本書會針對每一種所提到的等待事件之延緩參數，均解釋其意義。

參考指南

本節為第二章提到的參數、事件、統計數據、及 APT 指令稿的快速參考指南。

參數

參數	說明
_USE_VECTOR_POSTS	向量式通知（vectorpost）能讓多個等待中的行程，在單一的號誌作業中被通知。
POST_WAIT_DEVICE	postwait driver 是虛擬裝置驅動程式。當相關作業處理到本參數所指定裝置之特殊裝置檔時，其功能就會運作。本參數指定了 post-wait driver 的裝置檔之路徑。
TIMED_STATISTICS	在需要針對調校目的的時序（tming）資訊時，應將本參數設為TRUE。
USE_POST_WAIT_DRIVER	若本參數存在，應設為 TRUE，以便使用 postwait driver，而非一般的號誌機制。

事件

事件	說明
10046	事件係用來實作 DBMS_SUPPORT 追蹤（Oracle 的 SQL_TRACE 工具為其子集）。在等級 4，繫結呼叫（bind call）包含在追蹤之輸出中。等級 8 則包含 wait event（DBMS_SUPPORT 的預設層級）；而等級 12 包含了繫結和 wait。請參閱 Oracle Note 39817.1 5 中，關於追蹤檔案之原始資訊的精闢解說。

統計數據

統計數據	來源	描述
total_waits	V$SYSTEM_EVENT V$SESSION_EVENT	不同 wait 的總數。
total_timeouts	V$SYSTEM_EVENT V$SESSION_EVENT	因逾時而未被通知的 wait 數目。
logical_waits	total_waits - total_timeouts	邏輯的 wait 是指，同一事件中一系列不同的 wait。除了最後一個會被通知，其餘的都逾時。
time_waited	V$SYSTEM_EVENT V$SESSION_EVENT	等候時間的總合。

統計數據	來源	描述 （續）
average_wait	V$SYSTEM_EVENT V$SESSION_EVENT	每一個 wait 的平均時間。
average_logical	time_waited / logical_waits	每一個邏輯 wait 的平均時間。
max_wait	V$SESSION_EVENT	session 事件中，最長的 wait。

APT 指令稿

指令稿	說明
resource_waiters.sql	顯示等待某類資源的 session，以及等候的時間。
resource_waits.sql	依其重要性，顯示 Oracle instance 的生命週期中，所有被等待的資源，以及等候時間的總合。
routine_waits.sql	回報每一個 routine wait 的平均等待時間。
session_times.sql	顯示某一 session 曾經運作或等待的時間，以及等待的對象。
trace_waits.sql	將被某種 resource wait 影響最多的 sessions 之 DBMS_SUPPORT 追蹤（事件 10046，等級 8）啟動一段時間。用來對延緩參數取樣，以診斷效能問題。