

你確定程式都測試完了嗎？--利用 JProbe 偵查程式狀態

一個系統專案都是由一群工程師分工合作完成的，而每個人開發的程式都一定要經過一連串的單元測試，試圖找出可能隱藏的錯誤與問題，而這個過程都是由工程師自己做出單元測試條件進行測試，待工程師確定沒有問題後才交付給專案負責人員，進行系統的整合測試，然而這樣的測試是否已經把整個程式都測試完畢了嗎？是否還有一些所謂的測試程式碼，仍包含在程式中？是不是所有的狀況都有完整的測試到？

上述所列的三個問題是專案負責人常會面臨的問題，一般而言並沒有任何資訊可以證明上述的問題，只有透過工程師的 **Test Case**，大概可以了解單元測試是否完整，剩下的問題只能待整合測試或是使用者測試時，發生後再來做問題的排解，然而這樣是屬較被動的方式，無法整個掌握系統的狀況。因此若能有工具能產出測試結果的資訊，提供給專案負責人了解整個測試狀況，這樣便可以更準確掌握程式品質，可以避免地雷程式碼對未來整合測試或上線後造成的錯誤。

在此介紹一個好用的 APM 的工具 JProbe，JProbe 提供開發人員在程式開發時，用來偵查撰寫的程式執行效能、是否有 **Memory Leak** 跡象與監看是否有未測試過的程式碼，透過這個工具的監看，用以確保程式的效能與品質；在進入主題內容前，我們先介紹 JProbe 的主要功能。

JProbe

Quest JProbe Suite 是一套完整的 Performance 分析工具，主要可以提供的功能項目分為三種，詳細說明如下：

- 效能瓶頸分析(Profiler)：
 - 記錄系統時間花費的狀況。
 - 使用快照(Snapshot)方式比較記錄元件使用狀況。
 - 顯示系統元件的時間花費的狀況。
- 記憶體損耗分析(Memory Debugger)：
 - 記錄記憶體中元件數量與關聯狀況。
 - 使用快照(Snapshot)方式比較記錄記憶體使用狀況。
 - 分析元件的關聯性的關係，以了解記憶體使用狀況。
- 程式碼分析(Coverage)：
 - 記錄統計程式碼執行狀況。
 - 使用快照方式顯示出未被測試過的程式碼。

JProbe 它是一個透過 JVMPi 的方式來監看 JVM 裡面的運作狀態，因此它上述的功除了可以用於監控 Java standard application，也可以監控 Java 2 EE 上面執行的 Web Application、EJB 或是 framework 等；因此它的監控設定主要可以分為 J2SE 與 J2EE 兩種不同設定，用以監

July 07 M-Power eNew

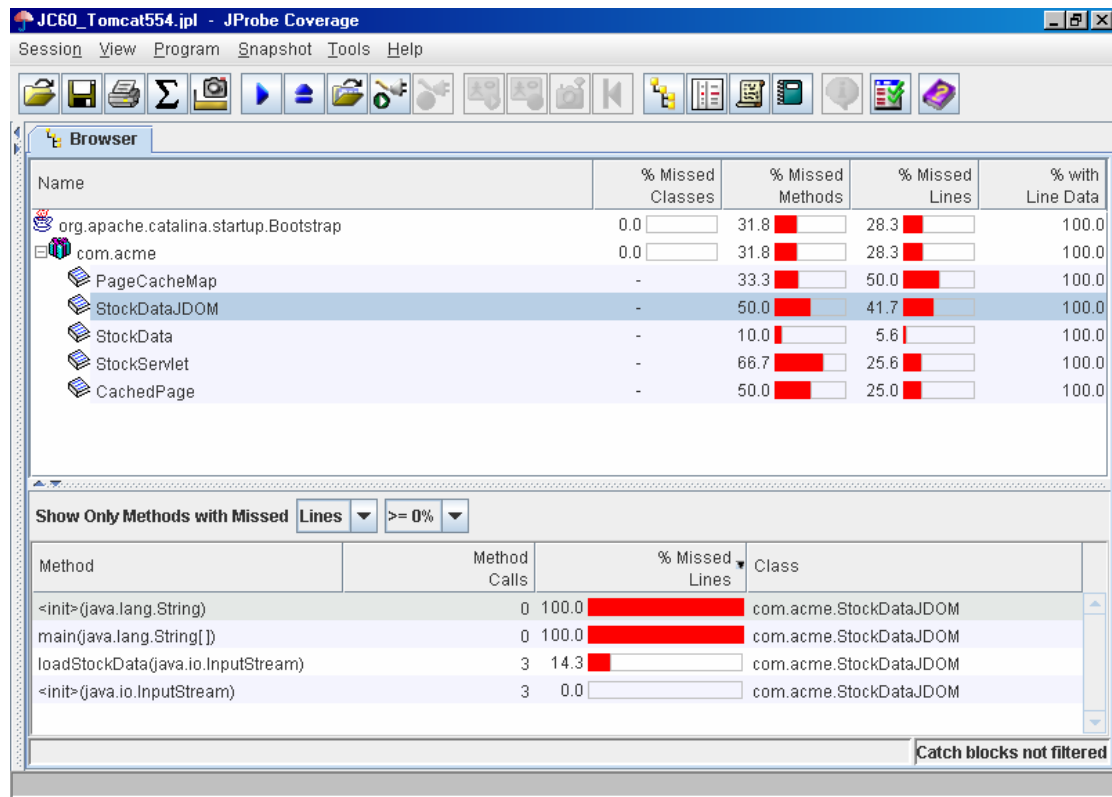
本篇文章版權為倍力資訊股份有限公司所有，未經書面同意，嚴禁複製、轉載

控不同的應用程式。

由於它能夠支援所有 J2EE 相容的 Application Server，不論是 Open Source 或是商用的 Application Server 都可以支援，如：Apache Tomcat、JBoss、Caucho Resin、BEA WebLogic、IBM Websphere、Sun ONE Webserver 等；都可以透過它內建的快速的 Application Server 環境設定的功能，建立各種不同 Application Server 的環境，以符合使用者環境狀況來監控應用程式；倘若有不在內建的 J2EE Application Server 的話，它也提供使用者自行設定的方式，以提供更多種 Application Server 的支援。本篇文章將以介紹 JProbe 功能中的 Coverage，介紹它的整個功能與使用時機。

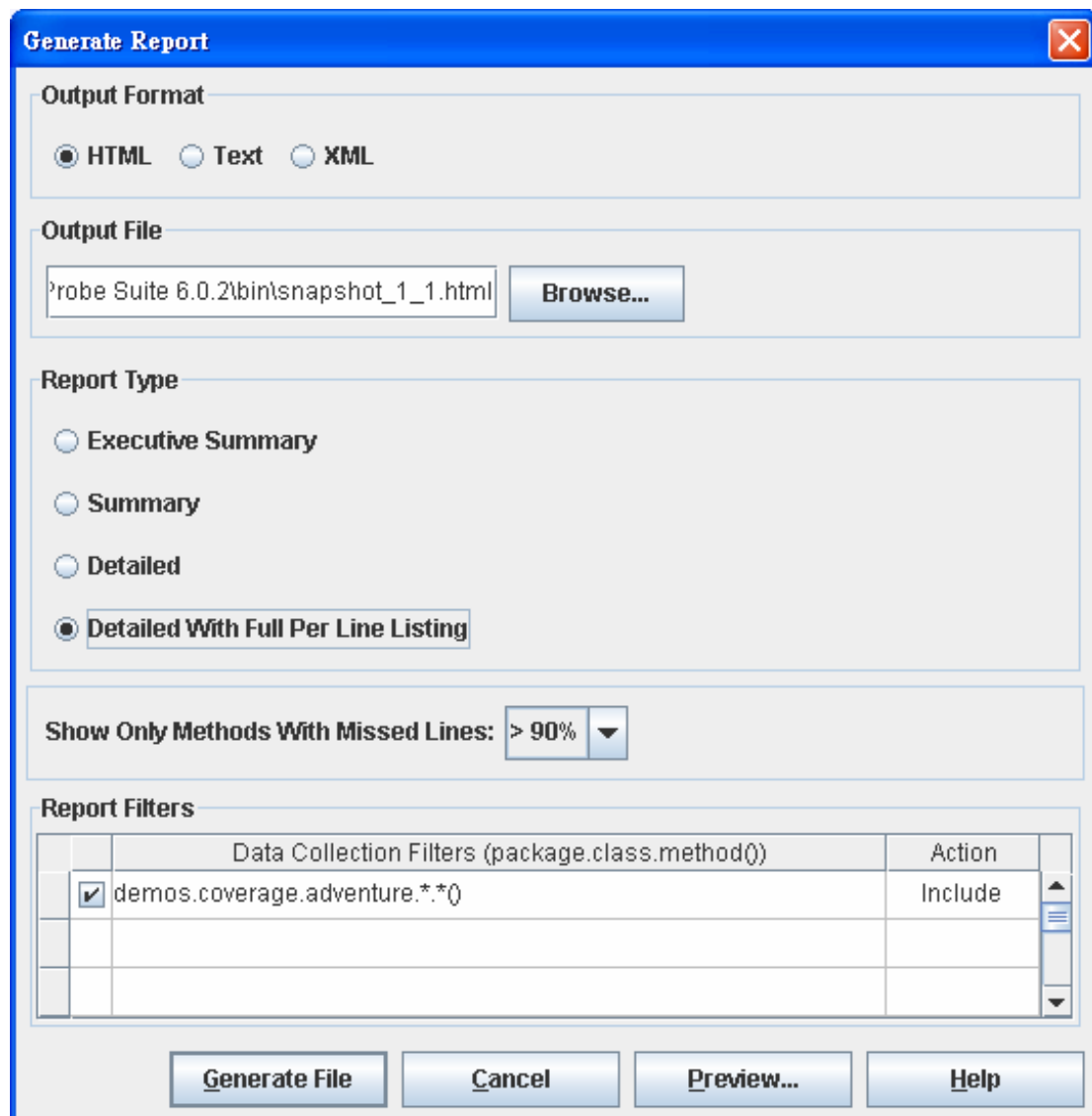
JProbe Coverage

當工程師做完單元測試後，一般而言，他的產出是交出所撰寫的程式碼，然而這樣的產出是否足夠呢？往往在進行整合測試時，就可以知道工程師是否有做完整的單元測試，這時才開始進行問題的排解，這樣只會造成將整個測試的時間拉長，並且無法有效掌握程式碼的品質；JProbe Coverage 是一個專門針對系統單元測試結果，我們可以驗證單元測試是否夠完整且所有的情況都有考慮進去，透過 JProbe Coverage 列出各個程式元件執行狀況，如下圖(一)所示，讓我們清楚看到整個單元測試結果，常有多少程式碼是我們錯誤未能完整測試到，做為下一次單元測試的依據，或是做為整個系統測試的依據，以求整個系統測試是能完整涵蓋到所有的程式碼與可能的系統使用情境。



圖(一) JProbe Coverage 監看結果

這些測試結果可以產出報告，做為單元測試的結果，如下圖(二)所示，其可以產出的格式有 HTML、Text 與 XML 三種文件格式；這樣的產出對於專案負責人員而言，除了是一個測試的記錄報表外，更可以清楚知道工程師所交出的程式碼測試的完整度，讓他更能掌握整個專案目前開發的狀況，與各項功能的進度；對於工程師而言，則是讓他們可以清楚知道，自己所做的單元測試還有那些狀況未能考慮進行，還有那些其他的情境測試需要再加入單元測試中，以求所有的程式碼都被測試過，以求整個程式碼的品質，降低未來出錯的可能性，並且可以有效提升系統的效能。



圖(二) 報表產出

使用 JProbe Coverage 監看的結果，若想追縱其原始程式碼的內容，可以透過 Coverage 將原始碼載入，呈現出原始碼測試的狀況，透過 Method 與 Line 兩種統計結果，讓我們可以更清楚知道是那些程式碼未被測試過，如下圖(三)所示。

Line #	Number of Calls	Method Time	Cumulative Time	Cumulative Objects	Source
145	381	3 (3.4%)	1,503 (6.8%)	1,907 (1.1%)	html += "<tr align='right'>";
146					} else {
147	375	2 (3.0%)	1,537 (7.0%)	1,877 (1.1%)	html += "<tr align='right' bgcolor='\"#E0E0E0\">";
148					}
149					
150	756	9 (11.5%)	1,895 (8.6%)	3,784 (2.1%)	html += "<td bgcolor='\"#C0C0C0\">\" + (i + 1) + \"</td>\";
151					
152	756	13 (15.9%)	2,359 (10.7%)	6,819 (3.8%)	html += "<td>\" + date.format(sd.getDate(i)) + \"</td>\";
153					
154	756	9 (11.8%)	3,423 (15.5%)	9,877 (5.5%)	html += "<td>\" + currency.format(sd.getOpen(i)) + \"</td>\";
155	756	7 (9.1%)	1,917 (8.7%)	9,828 (5.5%)	html += "<td>\" + currency.format(sd.getHigh(i)) + \"</td>\";
156	756	7 (8.4%)	1,939 (8.8%)	9,828 (5.5%)	html += "<td>\" + currency.format(sd.getLow(i)) + \"</td>\";
157	756	7 (8.4%)	1,934 (8.8%)	9,828 (5.5%)	html += "<td>\" + currency.format(sd.getClose(i)) + \"</td>\";
158					
159	756	8 (10.3%)	2,229 (10.1%)	6,048 (3.4%)	html += "<td>\" + volume.format(sd.getVolume(i)) + \"</td>\";
160					
161	756	5 (5.8%)	2,557 (11.6%)	3,780 (2.1%)	html += "</tr>\";
162					}
163					

圖(三) JProbe Coverage 原始碼瀏覽器

從上面圖(三)我自可以清楚看到，原始碼與測試結果的對照，以呈現出整個單元測試的結果與程式碼被執行的狀況，讓我們便於找出那些程式碼是未被執行過，或是它是屬於無效或無作用的程式碼，進而將那些無效與無作用的程式碼從原始碼中移除，以求整個原始程式碼的純淨與單純，這樣的程式碼才會是完整且具有一定品質以及其效能；透過 JProbe Coverage 的監看結果，將程式碼去蕪存菁也可以提升程式的效能。

透過 JProbe Coverage 記錄專案開發過程中系統單元測試的狀況，使用它的系統功能與測試結果產出的報表，做為專案的單元測試結果參考依據；同時也可以做為整個專案的品保的參考資訊之一，除了力求專案能達一定的品質水準，同時也讓整個專案開發過程能更佳順利。