Fehlersuche in Apex – ein praktischer Einstieg

Peter Raganitsch, FOEX GmbH

Fehler zu vermeiden, ist ein guter Plan, in der Realität werden sich in Apex-Anwendungen aber immer mal wieder kleine Probleme einschleichen. Dieser Artikel gibt einen Überblick über die Debug-Funktionalität in Apex und bietet somit einen Einstieg in die Fehlersuche.

Etwas läuft hier falsch, oder nicht? Was passiert hier eigentlich und wieso läuft meine Apex-Applikation nicht so, wie ich will? Das sind typische Fragen, die sich wohl jeder Entwickler immer mal wieder stellt. Um herauszufinden, was passiert, bietet Apex ein paar Werkzeuge an, allen voran den sogenannten „Debug". Streng genommen handelt es sich dabei um ein Logging, das – sobald aktiviert – in einer Log-Tabelle mitprotokolliert, was beim Laden oder Speichern einer Seite alles passiert und welche einzelnen Verarbeitungsschritte Apex durchführt.

Der Begriff „Debugging" ist in Wikipedia recht schön so definiert: „Debugging is the process of finding and resolving bugs or defects that prevent correct operation of computer software or a system. Debugging tends to be harder when various subsystems are tightly cou-

pled, as changes in one may cause bugs to emerge in another" *(siehe „https://en.wikipedia.org/wiki/Debugging")*.

Die Apex-Architektur

Ein Blick auf die Architektur von Apex ruft uns das vorangegangene Zitat über Debugging gleich nochmals in Erinnerung, denn Apex ist tatsächlich ein System, bestehend aus mehreren Subsystemen *(siehe Abbildung 1)*. Ein Fehler in Apex kann also seine Ursache in jedem der verwendeten Subsysteme haben – also entweder in der Datenbank, am Webserver, im Netzwerk oder im Browser ausgelöst worden sein. Das Apex Debugging kann nicht alle Subsysteme mitprotokollieren, aber mit seiner Hilfe kann man die Ursache zumindest lokalisieren.

Apex Debug

Der Apex-Debug-Modus lässt sich über die Developer Toolbar ganz einfach aktivieren *(siehe Abbildung 2)*. Anschließend wird die aktuelle Seite neu geladen und alle Verarbeitungsschritte werden im Hintergrund mitprotokolliert. In der Browser-Adressleiste kann man nun den aktivierten Debug durch das Schlüsselwort „YES" an der fünften Position der Apex-

Abbildung 1: Apex-Architektur, Quelle: http://www.oracle.com/technetwork/developer-tools/apex/overview/apex-dba-1867077.pptx

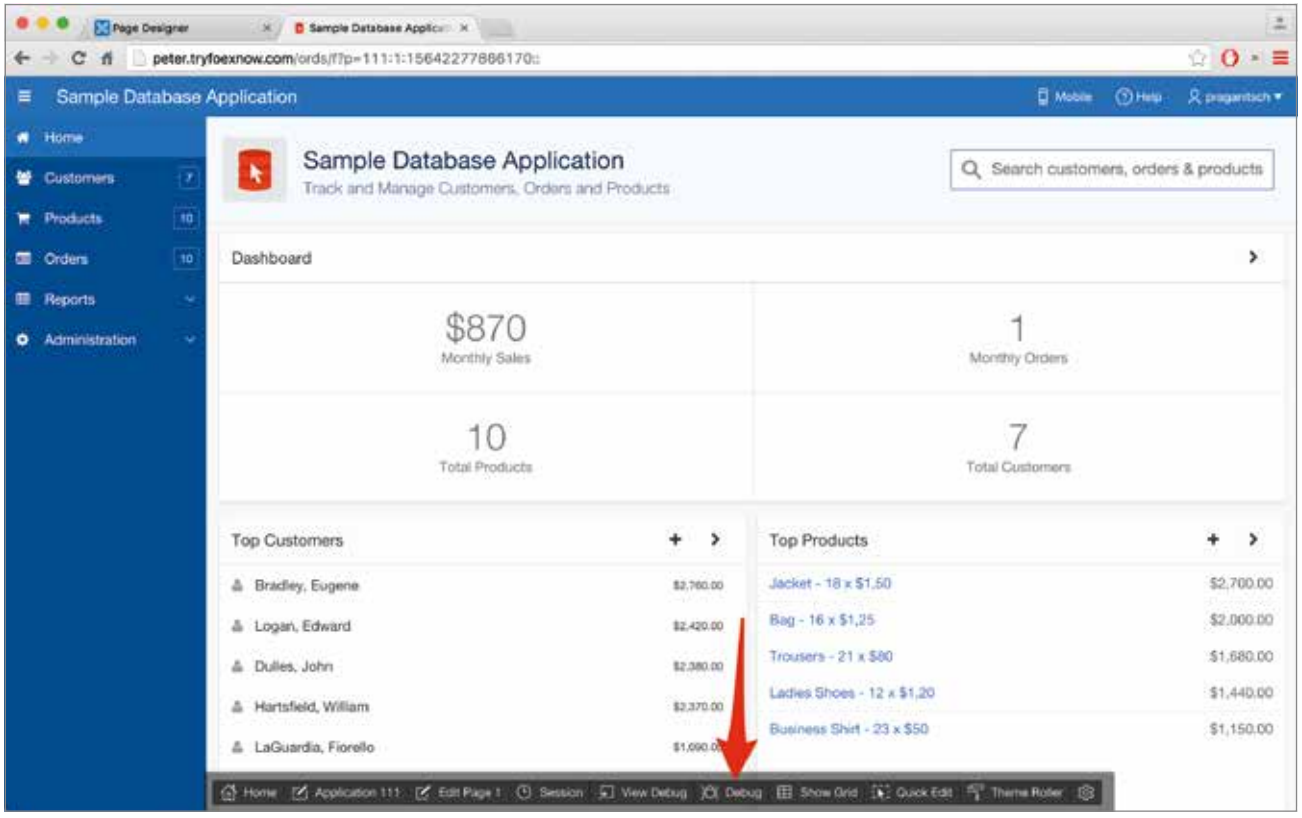


Abbildung 2: Debug über Developer Toolbar aktivieren

Level	Beschreibung
1 – Error	Kritische Fehler
2 - Warning	Weniger kritische Fehler
4 – Info	Default Level (=YES), wenn Debug ohne Angabe von Levels aktiviert wird. Zeigt alle normalen Debug-Informationen
5 – App Enter	Applikation: Meldungen zum Aufruf von Prozeduren
6 – App Trace	Applikation: Andere Meldungen innerhalb von aufgerufenen Prozeduren
8 – Engine Enter	Apex Engine: Meldungen zum Aufruf von internen Prozeduren
9 – Engine Trace	Apex Engine: Andere interne Meldungen

Tabelle 1

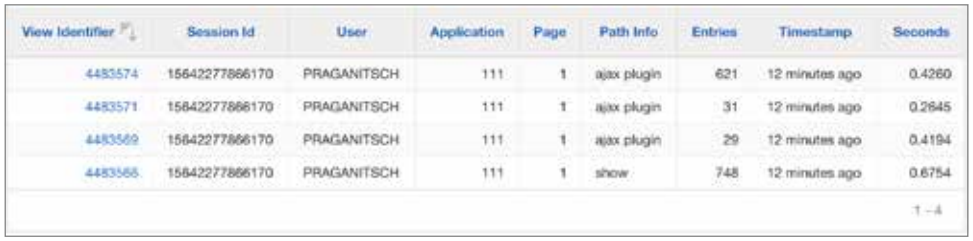
URL erkennen: „f?p=111:1:15642277866170::YES“. Neben der einfachen Aktivierung des Debug ist es seit Version 4.2 möglich, die Granularität des Debug zu definieren. Dafür stellt Apex neun Debug-Level zur Verfügung *(siehe Tabelle 1)*. Weitere Informationen dazu stehen in der Apex-Dokumentation unter „API-Reference“ im Kapitel „Apex Debug“ *(siehe „https://docs.oracle.com/cd/E59726_01/doc.50/e39149/apex_debug.htm#AEAPI29184“)*.

An dieser Stelle sei angemerkt, dass durch die Aktivierung von Debug und mit jedem höheren Level etwas mehr Zeit für die Verarbeitung erforderlich ist als mit ausgeschaltetem Debug. Das hängt schlicht von der Anzahl der zu schreibenden Log-Einträge ab und kann auf komplexeren Seiten in die tausende Datensätze gehen. Was genau mitgeloggt wird, kann man sich wieder über die Developer Toolbar, diesmal über den Button „View Debug“ ansehen *(siehe Abbildung 3)*.

In dieser Übersicht sehen wir sogenannte „Debug-Vorgänge“. In dem Beispiel geht es um die Startseite der „Sample Database Application“ und man sieht, dass gleich vier Vorgänge protokolliert wurden. In der Spalte „Path Info“ ist als Hinweis vermerkt, worum es sich handelt: „show“ ist das eigentliche Laden einer Seite, „ajax plugin“ ist der AJAX-Callback eines Plug-ins, das auf der Seite verwendet wird. Im Beispiel werden also erstmal die Seite selbst geladen („show“) und danach drei Plug-ins mit AJAX-Callbacks eingesetzt. Im Falle eines Speichervorgangs („submit“) würde in der „Path Info“ übrigens „accept“ stehen, gefolgt von dem verwendeten Request.

Klickt man nun auf einen der Debug-Vorgänge, so werden dessen Details angezeigt. Unter dem Eintrag „show“ zeigt schon die Übersicht, dass jetzt 748 Detail-Einträge folgen *(siehe Abbildung 4)*.

Die Detail-Ansicht bietet eine Reihe interessanter Informationen. In dem Block ganz oben steht als „Elapsed Time“ die gesamte Dauer der Verarbeitung (in diesem Fall das Laden der Seite), die Apex in der Datenbank aufwendet. Das ist also die Netto-Zeit der Datenbank, ganz ohne Netzwerk, Webserver und den Browser des Endanwenders. Die „Maximum Execution Time“ ist die längste Dauer eines Einzelschritts in der aktuellen Verarbeitung. Wenn hier eine große Zahl steht, dann findet man diesen Einzelschritt weiter unten in der Liste und kann ihn gezielt optimieren. Die Balkengrafik darunter gibt einen Überblick über die Dauer der jeweiligen



View Identifier	Session Id	User	Application	Page	Path Info	Entries	Timestamp	Seconds
4483574	15642277866170	PRAGANITSCH	111	1	ajax plugin	621	12 minutes ago	0.4260
4483571	15642277866170	PRAGANITSCH	111	1	ajax plugin	31	12 minutes ago	0.2645
4483569	15642277866170	PRAGANITSCH	111	1	ajax plugin	29	12 minutes ago	0.4194
4483566	15642277866170	PRAGANITSCH	111	1	show	748	12 minutes ago	0.6754

1 - 4

Abbildung 3: Debug-Übersicht

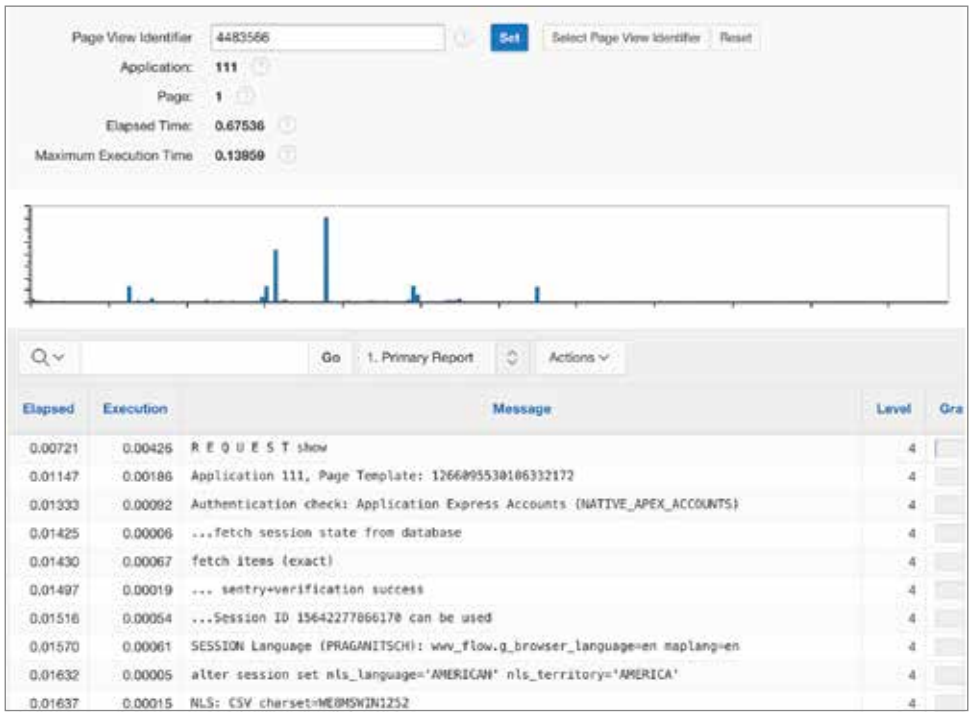


Abbildung 4: Debug Detail

Einzelschritte und vermittelt somit einen Eindruck davon, ob die Verarbeitungszeit gleichmäßig aufgeteilt ist oder ob es ein paar Ressourcen-Fresser gibt.

Danach folgt ein Interactive Report mit einer sortierten Auflistung aller Einzelschritte. In diesem Report kann nun Schritt für Schritt nachvollzogen werden, was Apex alles macht, bevor das Ergebnis (die fertige Seite) angezeigt wird. Wichtig für das Verständnis ist, dass die Einträge in der Reihenfolge ihrer Bearbeitung vorliegen. Dies kann einige Probleme aufklären, wenn beispielsweise durch den Debug klar wird, dass ein Element eventuell schon früher bearbeitet wird, als man das angenommen hatte. Beispiele für Detail-Einträge sind:

- Durchführung von Computations und Prozessen
- Das Nicht-Durchführen von Bearbeitungen, wenn dies von einer Condition, einer Authorization oder Ähnlichem verhindert wird
- Gesetzter Session State durch URL-Parameter, Computations/Prozesse etc.
- Ausgeführte Statements (Prozesse, Computations, Regionen)
- NLS-Settings der Datenbank-Session
- Aufgetretene Fehler
- Verarbeitete Regionen und Items

Den Debug Output lesen

Wie der Debug Output zu lesen ist, hängt in erster Linie von der Fragestellung ab, also davon, was man herausfinden möchte. Um nachzuvollziehen, was Apex genau macht, liest man den Debug am besten sequenziell von oben nach unten, in der zeitlichen Reihenfolge der Ereignisse. Ist man auf der Suche nach Performance-Hotspots, dann interessieren einen die Ausschläge in der Balkengrafik beziehungsweise man kann den Interactive Report so sortieren oder filtern, dass die Einträge mit der höchsten „Execution"-Zeit oben stehen.

Genau so entscheidend ist auch die Wahl des richtigen Debug-Levels. Mit dem Default „YES" (=Level 4) sind eventuell wichtige Einträge gar nicht vorhanden. Immer Level 9 zu aktivieren, ist aber auch nicht zielführend, da man hier von der Menge der Einträge erschlagen wird. Mit etwas Ausprobieren findet man für die jeweilige Situation bald die richtigen Einstellungen sowie die Balance zwischen der Anzahl der Log-Einträge und dem Informationsgehalt.

```
BEGIN
   Apex_DEBUG.ENTER('My Procedure');
   MY_PACKAGE.FUNCTION1;
   Apex_DEBUG.TRACE('Returned from Function1');
   MY_PACKAGE.FUNCTION2;
   Apex_DEBUG.TRACE('Returned from Function2');

   IF :P1_MY_ITEM = 'ERROR'
   THEN
       Apex_DEBUG.ERROR('Something went wrong');
   END IF;

   Apex_DEBUG.TRACE('Done, leaving My Procedure');
END;
```

Listing 1

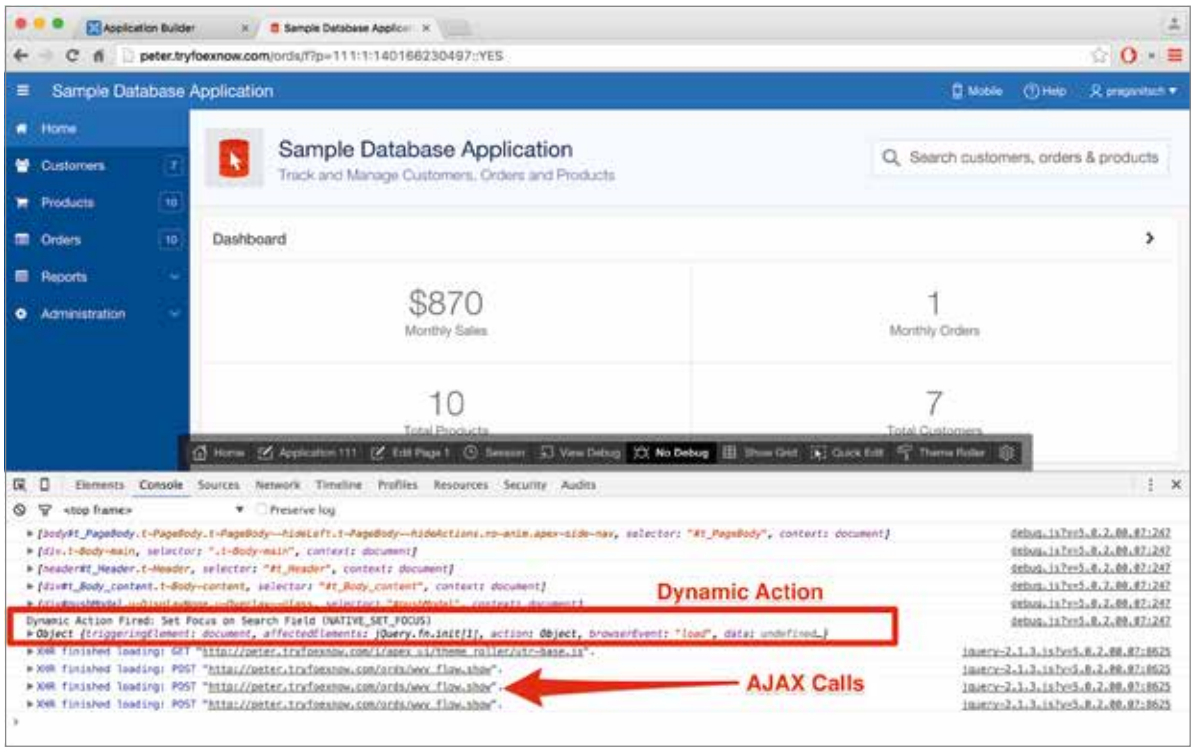


Abbildung 5: Debug-Meldungen in der Browser Console

Da der Debug Output als Interactive Report präsentiert wird, lassen sich hier auch verschiedene Ansichten als Private Reports speichern, in denen je nach Art der Nachforschung andere Spalten oder Sortierungen verwendet werden. Wer noch mehr Flexibilität will, der kann auch direkt die View „APEX_DEBUG_MESSAGES" abfragen und eigene Reports darauf aufsetzen. Diese View ist auf „Runtime only"-Produktionssystemen auch der einzige Weg, um an die Debug-Information zu gelangen.

Zeitliche Begrenzung der Debug-Information

Wie alle internen Apex-Log-Tabellen wird auch Debug periodisch gelöscht. Per Default ist das Lösch-Intervall auf 14 Tage gestellt, es kann aber über die Apex-Instanz-Administration auf maximal 180 Tage erhöht werden. Für die Speicherung werden intern zwei Tabellen verwendet, die bei Erreichen des Lösch-Intervalls rollieren: Erst wird in Tabelle1 geschrieben, nach 14 Tagen in Tabelle2. Weitere 14 Tage später wird Tabelle1 gelöscht und dann neu beschrieben. Das soll bedeuten, dass die Daten normalerweise „Löschintervall*2-1" Tage vorhanden sind. Die View „APEX_DEBUG_MESSAGES" fragt automatisch immer beide Tabellen ab.

Eigene Einträge hinzufügen

Mit den Debug-Einträgen von Apex lässt sich schon viel herausfinden. Sobald man aber eigene Business-Logik einsetzt, will man diese vielleicht auch im Debug mitprotokollieren. Dazu steht das „APEX_DEBUG-API" bereit, das mit diversen Log-Funktionen das Protokollieren in verschiedenen Levels ermöglicht: „ERROR", „WARN", „INFO", „TRACE", „ENTER" etc. Damit kann nun nicht nur in Apex-PL/SQL-Prozessen etwas in den Debug geschrieben werden, auch in aufgerufenen Packages und Funktionen lassen sich Log-Einträge erstellen *(siehe Listing 1)*.

Gut zu wissen ist, dass Debug-Einträge mit der Funktion „ERROR" immer geschrieben werden, unabhängig davon, ob Debug aktiviert ist. Das ist sehr praktisch, da man durch Selektieren von „APEX_DEBUG_MESSAGES" gezielt nach solchen Error-Einträgen suchen und damit eine automatisierte QA-Maßnahme aufsetzen kann.

Wird ein bestehendes Package aufgerufen, das Meldungen auf „DBMS_OUTPUT" schreibt, lassen sich diese mit „APEX_DEBUG.LOG_DBMS_OUTPUT" auf den Apex Debug umleiten. Ein sehr nützliches Feature in Zusammenhang mit der Verwendung älterer Bestands-Packages. „ENABLE" beziehungsweise „DISABLE" steuern den Debug-Modus auch programmatisch; so

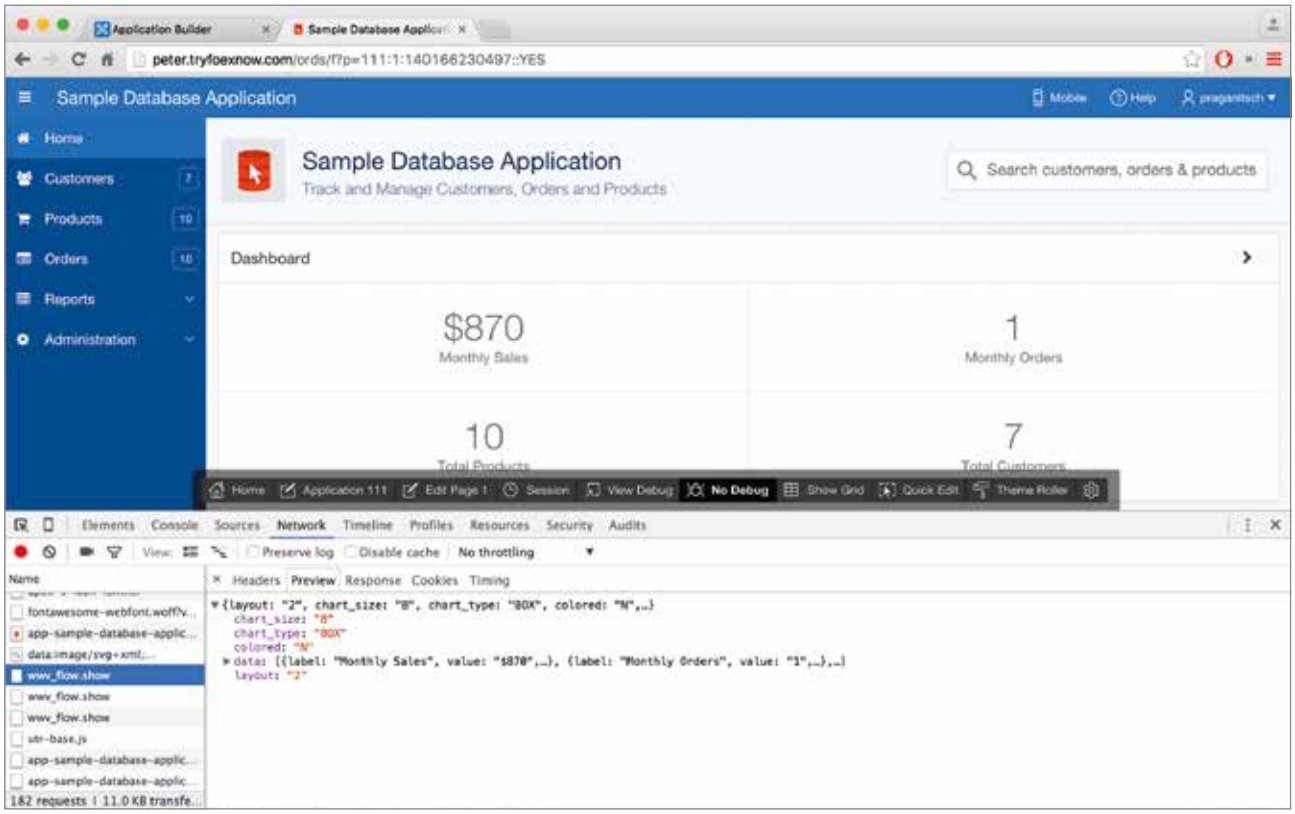


Abbildung 6: Das Ergebnis eines AJAX-Calls

kann man seinen Anwendern beispielsweise einen Button zum Aktivieren des Debug anbieten.

Debug auf dem Browser

Die bisher betrachtete Verwendung des Debug beschränkt sich auf die Verarbeitung der Apex-Seiten und AJAX-Calls innerhalb der Datenbank. In moderneren Anwendungen geschieht durch Dynamic Actions allerdings auch sehr viel im Browser. Nun ist es aber so, dass der Browser keine durchgehende Verbindung in die Datenbank hat und deswegen dort auch nicht in die Log-Tabelle schreiben kann. Dafür extra immer einen AJAX-Call abzusetzen, ist auch ausgeschlossen, das würde die Performance und Usability zerstören. Stattdessen bedient sich Apex hier der Browser-eigenen Console und deren Logging-Möglichkeiten.

Alle modernen Browser bieten inzwischen sogenannte „Developer Tools" an. Darin enthalten ist eine Console, die JavaScript-Meldungen protokolliert und sonstige Fehler und Warnungen anzeigt. Beim Aktivieren des Apex-Debug-Mode wird auch das JavaScript Logging aktiviert und man sieht Apex-Debug-Meldungen in der Console. Dort werden außerdem alle Dynamic Actions mitgeschrieben; so hat man auch im Browser einen guten Überblick über den Ablauf der Geschehnisse *(siehe Abbildung 5)*.

Die Abbildung zeigt wiederum die erste Seite der Sample Database Application, in der bei Page Load eine Dynamic Action ausgeführt wird. Gut sichtbar sind dort auch die drei AJAX-Aufrufe der Plug-ins. Klickt man auf einen dieser AJAX-Calls, kann man sich die Details und das Ergebnis anzeigen lassen, um so die empfangenen Daten zu kontrollieren *(siehe Abbildung 6)*.

Auch im JavaScript-Bereich ist es möglich, eigene Debug-Meldungen zu schreiben, das API dafür lautet „apex.debug" und funktioniert ganz ähnlich wie sein PL/SQL-Äquivalent.

Fazit

Sobald man alle Werkzeuge kennt, kann auch die Fehlersuche beginnen. Als Grundregel ist es dabei immer ratsam, die Browser Console zu öffnen und den Debug Mode zu aktivieren. Je nach Fehler beziehungsweise Problem, das untersucht werden soll, wird man sich dann mehr auf die Browser Console oder die Log-Einträge in der Datenbank konzentrieren. Für einen gesamthaften Überblick sollte aber beides betrachtet werden. Unter *„https://apex.oracle.com/pls/apex/f?p=25355"* gibt es eine Beispiel-Anwendung mit ein paar typischen Fehlern, die mithilfe von Debug einfach entdeckt werden können.

Peter Raganitsch
peter.raganitsch@tryfoexnow.com