Comparative Study of JAVA & APEX

Neha Tyagi (PhD scholar), Dr. (Prof.) Ajay Rana

Amity University, Noida

nehacs1988@gmail.com; ajay_rana@amity.edu

Abstract

C++, java and APEX all are object oriented languages. But if we go to the comparison of C++ and JAVA mostly market captured by JAVA due to its ultimate features.

Today's era is about cloud or in terms of computing we can say cloud computing. So, lots of applications are run over cloud. As a software developer we have to think that how these applications can be designed in a optimize way. If we take the example of Salesforce cloud, the APEX language is used for the development of applications of Salesforce cloud.

In this article we will discuss about the features of APEX language in comparison to C++ & JAVA.

Object oriented languages means orientation of objects and integrates the code and data using the concept of "objects". Here, objects corresponds to real world entities and designed in class pecking order.

Encapsulation & Information hiding is the features of object orientation practice. The essential approach following an object is that simulation. In object oriented languages program should be written to simulate the states and behavior of real world objects i.e. separately from looking data structures when modeling an object, we must also look at methods linked with that object means functions that amend the objects attributes. Examples o object oriented languages are: C++, Simula, JAVA,.Net, APEX etc.

What about APEX:

A little over a year ago, Salesforce.com announced the Apex development language. There has been tons written about Apex. Our goal here is to present what we think are the most salient points about Apex, why it's a big deal, and how it will make things better for my Salesforce.com users. We are not a killer programmer or industry analyst, so we will focus on our strengths and look at what new angles Apex will give us for solving our customers' business problems.[3]

The Apex programming language is saved and runs in the cloud-the Force.com multitenant platform.[2] Apex is tailored for data access and data manipulation on the platform and it enables you to add custom business logic to system events. While it provides many benefits for automating business processes on the platform, it is not a general purpose programming language As such, Apex cannot be used to:[2]

1) Render elements in the user interface other than error messages Change standard functionality
2) Apex can only prevent the functionality from happening, or add additional functionality
3) Create temporary files
4) Spawn threads

© 2015 British Journals ISSN 2047-3745

Tip:

All Apex code runs on the Force.com platform, which is a shared resource used by all other organizations. To guarantee consistent performance and scalability, the execution of Apex is bound by governor limits that ensure no single Apex execution impacts the overall service of Salesforce .This means all Apex code is limited by the number of operations (such as DML or SOQL)that it can perform within one process.[2]

All Apex requests return a collection that contains from 1 to 50,000 records. You cannot assume that your code only works on a single record at a time. Therefore, you must implement programming patterns that take bulk processing into account. If you dont, you may run into the governor limits.[2]

Functionalities:

1. Apex is a language to write procedural code that runs on the Salesforce servers. It's like writing stored procedures–you can access and manipulate data quickly–it on the server and it's compiled.[3]
2. This was a big move forward for Salesforce. Until Apex, procedural code had to be written in your language of choice and it used the web API for access to data. While this worked, accessing lots of data was very slow as all the database access had to traverse the Internet with the added overhead of SOAP encapsulation. Apex happens on the server, so there is no travel time for the data.[3]
3. Turns out it's not magic. It's really not all that innovative in what it does. Client server setups have had this kind of functionality for decades, it seems. But in the context of the multi-tennant architecture of Salesforce, it is really innovative. With Apex, you now can get the best parts of client server along with the best parts of on-demand. I can write my own code but not kill my server. Pretty cool.[3]
4. The language itself is syntactically appealing to me. It's very terse, and had a number of conventions that make sense in the context of Salesforce, but might not make sense elsewhere. For example, there is a data type for Salesforce record Ids. Another example is a simple for loop structure for looping through a database query result. In comparison to working with their AJAX toolkit, I've found Apex to be quick and light, with little wasted code.[3]
5. Another big positive of the Apex language is the Eclipse IDE plugin. The plugin for the popular Java development environment connects you directly to your Apex code on the Salesforce servers, and makes running unit tests a breeze. The developers I've shown the Eclipse plugin to have been jealous of features like automatically identifying the percentage of you code covered by your tests and listing which lines of code aren't being covered at all. It has made my entry into test-driven development pretty smooth.[3]
6. You can invoke Apex code in 2 ways: via triggers and through web-service calls. Triggers are code blocks that are run on changes to the data in your database. By using triggers you can run your arbitrary code every time a new Opportunity is inserted, for example. Having code run automatically based on data events is really powerful, and can do a lot for the user experience. You can easily automate User tasks that would have to be done by hand, making the application experience much nicer.[3]
7. Exposing your code via a web service makes it callable from the Salesforce UI. Apex has no facility for creating UI (newly announced VisualForce takes care of that), so exposing it as a web service makes that available to S-Controls, which can have UI. The User can click a button and fire your Apex code, replacing the need for complex S-Controls, and getting all the benefits mentioned earlier.[3]

© 2015 British Journals ISSN 2047-3745

So, Apex is faster than S-Controls, with simpler syntax. It can be called from buttons and data triggers. But what does that mean for user experience, which is the real test in a CRM application?

Application benefits of Apex

1. The first benefit we saw to Apex when it was announced was the ability to use triggers to eliminate steps the User had to take. Here's a real-world example.[3]
2. In the nonprofit world we care about households. We have donors and other supporters, and we often know other members of their family.[4] If we send them separate thank you letters, or (even worse) appeals for money, they might be offended. In most cases, we want to send one mailing to each household, naming the members of the household in the letter.[3]
3. In Salesforce, we model this as a simple custom object that can have multiple Contacts associated with it. Then, we can use this object when we want to send mailings. But to get Households in the system is a bit of a drag for the user. A User would have to create a Contact, then create a Household and relate them together. Households have addresses, but so do contacts, so the User would have to make address changes in two places. Lastly, if a new person was added to the Household, their name would have to be added
   to the name of the household, so that the letters mailed would identify all known members.[3]
4. Before Apex, we tackled these problems with S-Controls. We created an S-Control for creating Contacts that would automatically create a Household after the Contact save. The address fields would be synced at this time, but we had no way of keeping these automatically in sync later. We also got the name of the Household right, but the User was responsible for managing the data in the future.[3]

With Apex we've been able to improve this situation. We've created a couple triggers:

When a Contact is created without a Household, a new Household is automatically created, the address is saved to the Contact and the Household, and it is named correctly.[3]
When a Contact is added to or removed from a Household, the Household is renamed and the name is pushed to every Contact in the Household[3]
If a Household or a Contact address changes, the change flows through to everyone in the Household[3]

What this means is that all Householding activities are now fully automated. The User never need care about Households, they will just be there to be used when needed. As you can imagine, this is a big win and pretty exciting.[3]

The second benefit to Apex is the ability to denormalize data in interesting ways.[4] We've created triggers that do rollup calculations on related data and save it up to the Contact or Account. Apex can do complex calculations across much related data, so it's more flexible and powerful than Rollup Summary Fields. But why is this valuable?[3]

In Salesforce when you want to analyze data you build reports. Generally these reports look at one set of data– Contacts or Opportunities or Campaign Memberships. But often the questions that creative users come up with take multiple sets of data into account, or look at multiple pieces of data in the same set. Some examples of

© 2015 British Journals ISSN 2047-3745

questions you can't currently ask Salesforce are:[3]

Who gave at least two donations in the last 5 years? Who gave a donation and attended 3 events?

Are we converting our Members to Event attendees, or the other way around?
Apex lets us put strategic pieces of data on the Contact record. Then when we do a Campaign membership report, we can look at this other data at the same time. These native reports can fuel dashboards that are a level of analytics we don't have right now. And because they are native, we can use the reports to create prospect lists and funnel those directly into Campaigns. [3]

A third area where Apex can help immensely is with integrations. The plumbing of integration is pretty straightforward, but what is complex is supporting all the weird business cases that Users might have. Let's look at our Household example above. Much integration with Salesforce creates Contact records at some point in the flow. Event registration, online donation processing, even the Outlook integration has a Create Contact form. But with Householding, these integrations won't work, or will create incomplete data. A Contact with no Household is a problem, so we can't use these kinds of external integratons.[3]

But if Apex triggers fire whenever a Contact is created and take care of the Householding with no intervention, all of the sudden all these integrations work fine. The integrators don't need to know about Householding or any other crazy processes–they just create a Contact and leave it up to me to write the correct triggers. This is really exciting for people writing integrations, we can assure you. We're going to be offloading some of our Plone integration code that supports weird use cases to Apex triggers. Very happily, we might add. [3]

Apex can also make callouts to external web services, making just about anything possible. We haven't played with this yet but can't wait to do things like geocoding, address certification, and who knows what else.

Nothing's perfect

While we've been very positive so far, there are some limitations to Apex as it's currently released. First is that triggers don't cover all actions that could happen in the data. There are a number of junction tables that don't listen for events. The two biggest for me are Opportunity Contact Role and Campaign Member. These are tables that connect Contacts to Opportunities and Campaigns, respectively. Because there are no triggers on those tables. We could change the donor on a gift, but wouldn't be able to update the giving total to that Contact record. We also can't recalculate the events a person has attended when we mark them as having attended an event. We think these are pretty significant holes in the Apex infrastructure that hopefully will be fixed soon. With them unfixed, users can take actions that will make the data unreliable. We hope to minimize these cases, but there is only so much we can do.[3][4]

One way around gaps like the two we listed above is to use triggers as best we can and then run a nightly batch to clean up any changes that slipped through.[4] But Apex as it's currently released doesn't let us schedule the execution of blocks of code in a cron-like fashion. We think that functionality will come. [3]

The third limitation has to do with one of Salesforce's greatest strengths–it's multi-tenant architecture.[4] When

© 2015 British Journals ISSN 2047-3745

Salesforce decided to let crazy people like me run code on their servers, they had to protect themselves and their other customers from my bad code. So, they built a bunch of controls to make sure we couldn't hog resources. Part of this control system is a set of governors that limit the kinds of operations you can do. A trigger can only have 20 SOQL statements in it, for example. If a trigger is fired by a change to one record, only 100 other records can be updated. Arrays can only have up to 1000 members. The list goes on. [3]

So, the Apex cron job that would crawl through 20,000 opportunities nightly wouldn't work in the current governor regime.[4] A decent portion of the utility of Apex will hinge on how quickly these governors are lifted. It's a tough problem to solve, and one that has direct cost implications for them, so we'll see if they budge in the next 6 months. [3]

Conclusion and future steps

We are very bullish about Apex. It fills a need that has been wanting for a while. And it appears to fill it very nicely. There are some limitations in the current release, but still the benefit to user experience, analytics and integration are enormous, and the language and tools are very nice.[4] We're undergoing a process to rewrite as much of our custom code base from S-Controls to Apex. Initial experiments have been highly successful, and I've been very impressed by everything we've seen. Add to that the ability to adequately test the code we're writing for the first time and we are incredibly happy with the direction we're going.[3]

As we convert our codebase and add new functionality, we will report back on our status and [4] share code snippets and do some screen casts of interesting developments. We can't wait to really get going on the Apex path–we feel like it's going to change the way we do Salesforce, taking our systems to a level we've only been dreaming about.[3]

References:

1. http://www.sfdc99.com/sfdc99-reader-success-stories/
2. http://www.salesforce.com/us/developer/docs/apexcode/salesforce_apex_language_refere nce.pdf
3. **http://gokubi.com/archives/apex-what-it-is-and-how-it-improves-salesforcecom**
4. http://www.google.com/search

© 2015 British Journals ISSN 2047-3745