redhat.

Source Development Trigger Framework

Proposal for a new trigger framework to work with a new development model.

Bill C Riemers
Principle Engineer
2020-12-15

Salesforce Best Practices

Click to add subtitle

Salesforce recommends the following:

- Streamlining Multiple Triggers on the Same Object
  - Avoids redundant queries and for statements.
  - Control order of operation.

Others are more strict and recommend

- One trigger per object, logicless triggers, context specific handler methods.

Red Hat's Solution for Org Base Development

AbstracTrigger

The features of the Red Hat AbstractTrigger class solution include:

- One before, one after trigger per SObject. Each trigger calls a handler class.
- One handler class per trigger.
- Many context dependant methods per handler class.
- Boolean Metadata and Boolean Hierarchy to selectively activate and deactivate methods.
- 100% coverage is often of handler classes achievable with few or 0 DML operations.

AbstractTrigger Disadvantages

The disadvantage of the AbstractTrigger method is:

- Vendors and consultants have a learning curve to use it and have to strictly follow our coding styles to make the methods managable.
- Many developers end up editing the same class for different projects and routinely these changes need to be cherry picked to build a production release.
- Code developed for unmanaged packages may need to be completely rewritten to fit in the framework.
- Developers need to understand code that is not context dependant to their project.

Source Dependant Development Blockers

The AbstractTrigger unviable for packaged based deployments for the following reasons:

- Projects need to isolate code into independent units of work.
  If every project needs to update the same set of core trigger handlers, this independence was lost.
- Dependencies of non-org dependent packages need a strict chain.
  Which means all packages need to depend on the AbstractTrigger package. Furthermore, if package A depends on fields and package B, and package B depends on a trigger handler in package A, then the developers of both packages need to collaborate to introduce a package C that both A & B depend on.

Desired Solution Requirements

Ideally the trigger framework we create to solve this should meet the following requirements:

- Easy for an Salesforce Developer to adopt.
- Easy to migrate unmanaged packages.
- Only a loose dependency chain.
  While we can still implement helper classes, abstract classes, and interfaces, the use of the framework should not depend on them.
- Keep the context dependent methods, control order of invocation, easy activation and deactivation.
- Does not require porting all existing AbstractTrigger implementations.

Proposed Solution

Callable Triggers

Salesforce provides a Callable interface. Our proposal is to use it for triggers in the following manner:

- Each trigger handler implements a call method that accepts an action and a map as input parameters.
- The action is the name of the trigger method to invoke. The map is everything one would normally find in the System.Trigger object.
- The SObject trigger either can call our triggermanagement class directly, or it can use reflection to only call the trigger management class when available, otherwise call the handler directly.
- The AbstractClasss can call directly into the trigger management class, to avoid writing new triggers on every SObject.

Making It Easy

Callable Triggers

The following features can make it really easy:

- Simple substitutions like Trigger.isBefore can be (Boolean)args.get('isBefore').
- Triggers methods can loosely communicate with each other by passing values in the args.
- Boolean Metadata, Boolean Hierarchy and order of invocation are the job of the trigger manager.
- A TriggerArguments class can be used to make compatibility even more seamless. E.g. Trigger.isBefore is equivalent to triggerArgument.isBefore
- This is actually simpler to implement and use than our current processTriggers method.

Making It Easy

Callable Triggers

The following features can make it really easy:

- Simple substitutions like Trigger.isBefore can be (Boolean)args.get('isBefore').
- Triggers methods can loosely communicate with each other by passing values in the args.
- Boolean Metadata, Boolean Hierarchy and order of invocation are the job of the trigger manager.
- A TriggerArguments class can be used to make compatibility even more seamless. E.g. Trigger.isBefore is equivalent to triggerArgument.isBefore
- This is actually simpler to implement and use than our current processTriggers method.

The Downsides

Callable Triggers

By switching such an architecture we have the following disadvantages:

- Developers are less likely to share queries. Even though the framework allows it, it takes more planning to do so.
- We are far more reliant on metadata. A set of methods maybe written to expect a certain order of operations, and a different project could change that order of operations on the fly, perhaps in an attempt to share queries.
- This is a very much a home grown solution.

Next Steps

Callable Triggers

A proof of concept of this new trigger framework has been installed in Dev1. There is 100% code coverage, and several example callable triggers available, that are actively running.

Try it, and provide feedback. If your project depends on it, we'll promote it with your project, provided no significant design flaws are found.

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos