Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2020, 7(5):391-394**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Implementing a Platform Event Trigger Framework in Salesforce

Chirag Amrutlal Pethad

PetSmart.com, LLC, Stores and Services
Phoenix, Arizona, USA
ChiragPethad@live.com, ChiragPethad@gmail.com, Cpethad@petsmart.com

Abstract: The document discusses the implementation of a Platform Event Trigger Framework in Salesforce Apex, highlighting the importance of Platform Events for event-driven architecture, the role of Apex Triggers, and the need for a structured trigger management framework. It outlines key components, benefits, best practices, and provides examples of trigger and handler class implementations, emphasizing improved scalability, maintainability, and performance in Salesforce development.

Keywords: Execution Context, Context Variables, Bulk, Modularity, Scalability, Governor Limits, Testability

Introduction

Platform Events is one of the powerful feature in Salesforce which allows for an event driven architecture within Salesforce ecosystem. Platform Events are used to deliver secure and scalable custom notifications within Salesforce or from external sources. Salesforce Apex Triggers allows you to execute custom logic in response to an event published within Salesforce or from an external source. This white paper explores the Platform Event Trigger Framework in Salesforce Apex, providing an in-depth look at its architecture, implementation, and use cases.

Overview of Salesforce Platform Events

Platform Events are a key component of Salesforce's event-driven architecture. They enable the integration of Salesforce with external systems in real time by facilitating the delivery of custom notifications. Platform Events can be published and subscribed to both within Salesforce and externally through REST API or SOAP API.

A. Key Characteristics of Platform Events

- **Asynchronous:** Platform Events are processed asynchronously, ensuring non-blocking operations.
- **Scalable:** Platform Events are designed to handle high volume of events efficiently.
- **Reliable:** Provide guaranteed delivery of messages, ensuring no events are lost.

Overview of Salesforce Platform Events Triggers

Apex Triggers are server-side scripts that execute automatically whenever specific platform events are raised or published in Salesforce. Platform event Triggers unlike S- Object triggers can only operate after the events are published in Salesforce. They provide way for applications perform additional actions, or business logic in response to these events.

A. Trigger Execution Context

- **After Triggers:** These triggers execute after the records are committed to the database. They are often used for tasks such as sending notifications, updating related records, or invoking external services.

- **Bulk Triggers**: Triggers in Salesforce are designed to handle bulk operations, meaning they should be written to process multiple records efficiently. Bulk triggers are essential for ensuring performance and scalability of your application.
- **Publish Behaviors:**
    - Publish After Commit to have the event message published only after a transaction commits successfully.
    - Publish Immediately to have the event message published when the publish call executes. With this option, a subscriber can receive the event message before data is committed by the publisher transaction.

B. Trigger Events and Context Variables

Salesforce provides trigger context variables that allow code logic to access information about the records being processed and the operation being performed. Some of the commonly used trigger context variables include:

- **Trigger.new**: Contains the list of new records being inserted or updated.

What Is a Framework

A framework is a highly optimized, reusable structure that serves as a building block. These building blocks provide common functionality that developers can override or specialize for their own needs. Reusable frameworks increase the speed of development, improve the clarity and efficiency of your code, and simplify code reviews and debugging.

Need For a Trigger Framework

As the complexity of a Salesforce org increases, so does the need for a robust mechanism to manage triggers efficiently. The primary goal of this framework is to avoid common pitfalls such as recursion, governor limit exceptions, and unmanageable code by providing a systematic way to structure trigger logic. Some challenges without a framework include:

- **Code Duplication:** Similar logic across multiple triggers.
- **Unorganized Code:** Hard to maintain and read.
- **Performance Issues:** Unoptimized trigger execution.
- **Lack of Reusability:** Difficulty in reusing common logic.

A Trigger Framework addresses these issues by providing a structured approach to trigger management.

Core Components of a Trigger Framework

A Trigger Framework typically consists of the following components:

- **Trigger Handler Class:** Contains the business logic for different trigger events.
- **Helper Classes:** Encapsulate reusable logic.
- **Utility Classes:** Provide common functionalities.
- **Custom Settings/Metadata**: Control trigger execution dynamically.

Benefits of Using a Trigger Framework

Implementing a Trigger Framework offers several advantages:

- **Reusability:** Trigger framework allows to implement shared logic that can be reused across multiple triggers.
- **Maintainability**: Trigger frameworks enable to maintain cleaner and more organized code.
- **Control:** Trigger frameworks provide ability to manage trigger execution order and context.
- **Modularity:** Trigger frameworks promote modular design by separating concerns and encapsulating trigger logic within handler classes.
- **Scalability**: Trigger frameworks enable to easily add, remove, or modify trigger logic without affecting existing functionality.
- **Testability:** Trigger logic encapsulated within handler classes can be easily unit tested, leading to improved code quality and reliability.

- **Governor Limits Management:** Trigger framework help in managing Salesforce governor limits more efficiently by optimizing the execution of trigger logic.
- **Version Control and Change Management**: Trigger framework encourage version control and change management procedures. This is especially significant in larger development teams or projects when several developers are working on the same codebase.
- **Collaboration:** Using a shared framework generates coding norms and patterns that facilitate developer collaboration. It encourages consistent code style and structure across the development team.

Implementing A Trigger Framework for Platform Events

A. Context Variables

We start by creating an Apex class to automatically populate the Trigger Context variables. This is much cleaner than having methods with 5 to 10 parameters while still giving full access to all standard functionality.

```
public class EventTriggerContext {
    private System.TriggerOperation operation {
        get; set{ operation = System.Trigger.operationType;}
    }
    private List<SObject> triggerNew {
        get; set{ triggerNew = System.Trigger.new;}
    }
}
```

B. Interface to declare Trigger Handler Blueprint

The Trigger Handler Pattern involves creating a handler interface and implementing the only operation available for Platform Events in specific Handlers.

Implement this interface to ensure consistency across different handler classes.

```
public interface ITriggerHandler {
   void afterInsert(Map<String, SObject> newItems);
}
```

C. Tigger Handler Metadata

Create a Helper Class that will retrieve the Platform Event trigger metadata which includes all the handlers associated with the Platform Event, Order of Execution, etc.

```
public class EventTriggerSettings {
  private List<ITriggerHandler> handlers = new List<ITriggerHandler>();
  public EventTriggerSettings(Schema.SObjectType objType) {
      //Write logic to get the handler Names from Custom Metadata
      // or Custom Settings. And then loop over the list
      // and Instantiate instances of those Trigger Handlers for SObjectType
      //for (TriggerHandlerMetadata_mdt triggerHandler : triggerHandlers) {
      //      Type t = Type.forName(handlerSetting.DeveloperName);
      //    this.handlers.add((ITrigger)t.newInstance());
      //}
  }
  public List<ITriggerHandler> getHandlers(){ return handlers; }
}
```

D. Tigger Dispatcher / Controller

Next, we implement a dispatcher class that will orchestrate all the execution of various handlers and logic associated with an Platform Event.

```
public class EventTriggerDispatcher {
  public static void dispatch(Schema.SObjectType objType) {
    // Set the EventTriggerContext
    EventTriggerContext  ctx = new EventTriggerContext();
    if (ctx != null && ctx.getOperation() != null && ctx.getOperation().equals(
        System.TriggerOperation.AFTER_INSERT)){
      // Get the Handlers Lis
      List<ITriggerHandler> handlers = new EventTriggerSettings(objType).getHandlers();
      for (SObject so : ctx.getTriggerNew()) {
        for (ITriggerHandler handler : handlers){
            handler.afterInsert(so);
        }
      }
    }
  }
}
```

Example Implementation

A. Trigger

Time and effort in using the framework reduces the time and effort of the developers so that they can focus on the business logic.

```
trigger MyCustomTrigger on My_Custom_Metadata__mdt (after insert) {
    EventTriggerDispatcher.dispatch(My_Custom_Metadata__mdt.sObjectType);
}
```

B. Handler

Developers just need to override and implement the business logic for the one event types i.e. after insert operation.

```
public class MyCustomTriggerHandler implements ITriggerHandler {
    override public void afterInsert() {
        // After insert logic here
    }
}
```

Best Practices

- **Bulkification:** Ensure all trigger operations handle bulk data efficiently.
- **Single Trigger Per Object:** Consolidate logic into a single trigger per object.
- **Limit DML Statements:** Minimize the number of DML statements and queries within triggers.
- **Error Handling:** Implement robust error handling and logging mechanisms.

Considerations

An Apex trigger processes platform event notifications sequentially in the order they are received. The order of events is based on the event replay ID. The maximum batch size in a platform event trigger is 2,000 event messages. The order of events is preserved within each batch. The events in a batch can originate from one or more publishers. Unlike triggers on standard or custom objects, triggers on platform events don't execute in the same Apex transaction as the one that published the event. The trigger runs asynchronously in its own process. As a result, there can be a delay between when an event is published and when the trigger processes the event.

Conclusion

Triggers are powerful tools in Salesforce development, but they require careful management to ensure scalability, maintainability, and performance. Implementing a Trigger Framework in Salesforce Apex significantly enhances code quality and maintainability. By organizing and structuring trigger logic, developers can ensure scalable and efficient trigger management, ultimately leading to a more robust Salesforce environment.

References

[1]. Salesforce Platform Events Developer Guide-https://developer.salesforce.com/docs/atlas.en-us.platform_events.meta/platform_events/platform_events_intro.htm
[2]. Salesforce Apex Developer Guide - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm
[3]. Salesforce Trigger Best Practices-https://trailhead.salesforce.com/content/learn/modules/apex_triggers/apex_triggers_intro.
[4]. Fflib-apex-common - https://github.com/apex-enterprise-patterns/fflib-apex-common
[5]. https://github.com/kevinohara80/sfdc-trigger-framework