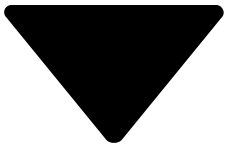
1. Asynchronous Apex



2. Schedule Jobs Using the Apex Scheduler

Schedule Jobs Using the Apex Scheduler

Learning Objectives

After completing this unit, you'll know:

- When to use scheduled Apex.
- How to monitor scheduled jobs.
- Scheduled Apex syntax.
- Scheduled method best practices.

Scheduled Apex

The Apex Scheduler lets you delay execution so that you can run Apex classes at a specified time. This is ideal for daily or weekly maintenance tasks using Batch Apex. To take advantage of the scheduler, write an Apex class that implements the Schedulable interface, and then schedule it for execution on a specific schedule.

Scheduled Apex Syntax

To invoke Apex classes to run at specific times, first implement the `Schedulable` interface for the class. Then, schedule an instance of the class to run at a specific time using the `System.schedule` method.

```
global class SomeClass implements Schedulable {
    global void execute(SchedulableContext ctx) {
        // awesome code here
    }
}
```

Copy

The class implements the `Schedulable` interface and must implement the only method that this interface contains, which is the `execute` method.

The parameter of this method is a `SchedulableContext` object. After a class has been scheduled, a `CronTrigger` object is created that represents the scheduled job. It provides a `getTriggerId` method that returns the ID of a `CronTrigger` API object.

Sample Code

This class queries for open opportunities that should have closed by the current date, and creates a task on each one to remind the owner to update the opportunity.

```
global class RemindOpptyOwners implements Schedulable {
   global void execute(SchedulableContext ctx) {
       List<Opportunity> opptys = [SELECT Id, Name, OwnerId, CloseDate
           FROM Opportunity
           WHERE IsClosed = False AND
           CloseDate < TODAY];
       // Create a task for each opportunity in the list
       TaskUtils.remindOwners(opptys);
   }

}
```

Copy

You can schedule your class to run either programmatically or from the Apex Scheduler UI.

Using the System.Schedule Method

After you implement a class with the Schedulable interface, use the `System.Schedule` method to execute it. The `System.Schedule` method uses the user's timezone for the basis of all schedules, but runs in system mode—all classes are executed, whether or not the user has permission to execute the class.

Note

Use extreme care if you're planning to schedule a class from a trigger. You must be able to guarantee that the trigger won't add more scheduled job classes than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.

The `System.Schedule` method takes three arguments: a name for the job, a CRON expression used to represent the time and date the job is scheduled to run, and the name of the class.

```
RemindOpptyOwners reminder = new RemindOpptyOwners();
// Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
String sch = '20 30 8 10 2 ?';
String jobID = System.schedule('Remind Opp Owners', sch, reminder);
```

Copy

For more information on the CRON expression used for scheduling, see the "Using the System.Schedule Method" section in Apex Scheduler.

Scheduling a Job from the UI

You can also schedule a class using the user interface.

1. From Setup, enter `Apex` in the Quick Find box, then select **Apex Classes**.
2. Click **Schedule Apex**.
3. For the job name, enter something like `Daily Oppty Reminder`.
4. Click the lookup button next to Apex class and enter `*` for the search term to get a list of all classes that can be scheduled. In the search results, click the name of your scheduled class.
5. Select Weekly or Monthly for the frequency and set the frequency desired.
6. Select the start and end dates, and a preferred start time.
7. Click **Save**.

Testing Scheduled Apex

Just like with the other async methods we've covered so far, with Scheduled Apex you must also ensure that the scheduled job is finished before testing against the results. To do this, use `startTest` and `stopTest` again around the `System.schedule` method, to ensure processing finishes before continuing your test.

```
@isTest
private class RemindOpptyOwnersTest {
    // Dummy CRON expression: midnight on March 15.
    // Because this is a test, job executes
    // immediately after Test.stopTest().
    public static String CRON_EXP = '0 0 0 15 3 ? 2022';
    static testmethod void testScheduledJob() {
        // Create some out of date Opportunity records
        List<Opportunity> opptys = new List<Opportunity>();
        Date closeDate = Date.today().addDays(-7);
        for (Integer i=0; i<10; i++) {
            Opportunity o = new Opportunity(
                Name = 'Opportunity ' + i,
                CloseDate = closeDate,
                StageName = 'Prospecting'
            );
            opptys.add(o);
        }
        insert opptys;

        // Get the IDs of the opportunities we just inserted
        Map<Id, Opportunity> opptyMap = new Map<Id, Opportunity>(opptys);
        List<Id> opptyIds = new List<Id>(opptyMap.keySet());
        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule('ScheduledApexTest',
            CRON_EXP,
            new RemindOpptyOwners());
        // Verify the scheduled job has not run yet.
        List<Task> lt = [SELECT Id
            FROM Task
            WHERE WhatId IN :opptyIds];
```

```
        System.assertEquals(0, lt.size(), 'Tasks exist before job has run');
        // Stopping the test will run the job synchronously
        Test.stopTest();

        // Now that the scheduled job has executed,
        // check that our tasks were created
        lt = [SELECT Id
            FROM Task
            WHERE WhatId IN :opptyIds];
        System.assertEquals(opptyIds.size(),
            lt.size(),
            'Tasks were not created');
    }
}
```

Copy

Things to Remember

Scheduled Apex has a number of items you need to be aware of (see Apex Scheduler in the Resources section for a complete list when you have time), but in general:

- You can only have 100 scheduled Apex jobs at one time and there are maximum number of scheduled Apex executions per a 24-hour period. See Execution Governors and Limits in the Resources section for details.
- Use extreme care if you're planning to schedule a class from a trigger. You must be able to guarantee that the trigger won't add more scheduled jobs than the limit.
- Synchronous Web service callouts are not supported from scheduled Apex. To be able to make callouts, make an asynchronous callout by placing the callout in a method annotated with `@future(callout=true)` and call this method from scheduled Apex. However, if your scheduled Apex executes a batch job, callouts are supported from the batch class.

Resources

- Apex Scheduler
- Execution Governors and Limits

Note

Remember, this module is meant for Salesforce Classic. When you launch your hands-on org, switch to Salesforce Classic to complete this challenge.

Assessment Complete!

+500 points

Asynchronous Apex
100%
Progress: 100%
Retake this Challenge
View more modules