Apex Testing

Monday, August 08, 2016 10:27 AM

Getting Started with Apex Unit Tests

Apex code can only be written in a sandbox environment or a Developer org, not in production.
App developers can distribute Apex code to customers from their Developer orgs by uploading packages to the Force.com AppExchange.

The following are the benefits of Apex unit tests.

- Ensuring that your Apex classes and triggers work as expected
- Having a suite of regression tests that can be rerun every time classes and triggers are updated to ensure that future updates you make to your app don't break existing functionality
- Meeting the code coverage requirements for deploying Apex to production or distributing Apex to customers via packages
- High-quality apps delivered to the production org, which makes production users more productive
- High-quality apps delivered to package subscribers, which increase your customers trust

Note

Before each major service upgrade, Salesforce runs all Apex tests on your behalf through a process called Apex Hammer. The Hammer process runs in the current version and next release and compares the test results. This process ensures that the behavior in your custom code hasn't been altered as a result of service upgrades. The Hammer process picks orgs selectively and doesn't run in all orgs. Issues found are triaged based on certain criteria. Salesforce strives to fix all issues found before each new release.

Maintaining the security of your data is our highest priority. We don't view or modify any data in your org, and all testing is done in a copy that runs in a secure data center.

At Least 75% of Apex code must be covered by tests, and all those tests must pass. Every trigger must have some coverage.

When building Test classes, make sure to test common use cases within the app, including positive and negative test cases, and bulk and single record processing.

Test methods take no arguments and have the following syntax:

```
1  @isTest static void testName() {
2      // code_block
3  }
```

Alternatively, a test method can have this syntax:

```
1  static testMethod void testName() {
2      // code_block
3  }
```

Using the isTest annotation instead of the testMethod keyword is more flexible as you can specify parameters in the annotation.

The visibility of a test method doesn't matter. Access modifiers are omitted from the syntax. Test methods and classes must be annotated with isTest.

The visibility of a Test Class depends on whether or not the test class is used for data factory classes.

Verification is done via System.assertEquals() method, which takes 2 parameters: The expected value and the actual variable. An optional 3rd parameter is for a string that describes the comparison being done. This string is logged if the assertion fails.

When you modify Apex code, rerun tests to refresh code coverage results. To update your code coverage results, use Run all instead of New Run.

Test invalid inputs and boundary conditions.

Earn higher code coverage by covering all data values for conditional code execution.

Note

The equality operator (==) performs case-insensitive string comparisons, so there is no need to convert the string to lower case first. This means that passing in 'ca' or 'Ca' will satisfy the equality condition with the string literal 'CA'.

By default, Apex tests don't have access to pre-existing data in the org, except to setup and metadata objects. Create test data either directly in the test class or by using a utility test class. To access pre-existing org data, annotate the test method with @isTest(SeeAllData=true) ---> Note: this is not a best practice.

You can save 3MB of Apex code in each org, but classes annotated with @isTest don't count toward this limit. Test methods don't send emails or make callouts to external services. There is no separate database used for testing, so inserting duplicate sObject records results in an error. SOSL searches return empty results in tests. To ensure predictable results, use Test.setFixedSearchResults() to define the records to be returned by the search.

Test Apex Triggers

When testing Triggers, you'll need to do DML statements to ensure the trigger is responsive. Test.startTest() and Test.stopTest() are method pairs that delimit a block of code that should get fresh governor limits. It is a good idea to isolate the data setup's limit usage from the test process's.

Create Test Data for Apex Tests

Add a Test Utility class to create records on class invocations instead of creating them on the fly. Test utility classes are excluded from the org's code size limit.

Ughhh.. Apex lists… - When I tried to instantiate a list with a predetermined size, I got an error for my

return type. Also, using the list.size() functionality caused a CPU Time Limit Exception.