Rex Jones II

Chapter 6:
Dependency Testing

Chapter 6.1 – Introduction To Dependency Testing

Introduction

Hello and Welcome To Chapter 6, Dependency Testing. Dependency Testing is one of the main features that separate TestNG from all other Test Frameworks. After reading Next Generation Java Testing, I came across a section called Dependent Code and it mentioned some developers in the testing community are strongly opposed to any hint of dependencies in their test.

One of their arguments state as soon as Test Methods depend on other Test Methods, it becomes hard to run those methods in isolation. That argument was true until TestNG. Now, we are allowed to run Test Methods in isolation and part of a dependency. TestNG provides a way to clearly indicate those dependencies. Therefore, it has no problem considering the requirements for any Test Method we want to run in isolation.

This chapter is a series of 3 videos. The first video is Chapter 6.1, Introduction To Dependency Testing. The second video is Chapter 6.2 and it will cover the Depends On Methods Attribute. The third video is Chapter 6.3 which will cover the Groups and Depends On Groups Attribute.

Attributes are used to assist annotations with our testing. All 3 of these attributes can be placed in the Configuration Annotations and the Test Annotation. This chapter is 6.1 and we are going to cover Introduction To Dependency Testing.

What Is Dependency Testing

What Is Dependency Testing? Dependency Testing is when 2 or more actions, 2 or more tasks, 2 or more functions in our Test Application depend on the order of their methods. For example, we must Sign In to the application before we can Sign Out of the application. Therefore, signing out depends on us first signing in. If we, reverse that order and execute, then the sign out method would fail.

We have seen a form of Dependency Testing in Chapter 3 with Annotations. The Test Methods depend on the Configuration Annotations. If a Configuration Annotation fail then the Test Methods would not pass. Here's an example of a Failed Configuration. Let's run.

We see 1 Configuration Failure and 1 Skip. The BeforeClass setUp Configuration Failed and the AfterClass tearDown Configuration was Skipped. Then we see the Test Methods. There are 3 Runs and 3 Skips. Zero Failures. The Test Methods were skipped because the setUp method Failed. At the core, that is Dependency Testing. It is Dependency Testing because the Test Methods were skipped and did not fail. You can get more information about Dependencies on TestNG website.
testng.org/doc/documentation-main.html

Click 5.7 – Dependencies and it takes you straight to that section. Next in Chapter 6.2, we will discuss the Depends On Methods Attribute which helps us with Dependency Testing.

Chapter 6.2 – Depends On Methods Attribute

Introduction

Welcome To Chapter 6.2, The Depends On Methods Attribute. In this chapter, we will cover execute Test Methods without the Depends On Methods Attribute then execute Test Methods with the Depends On Methods Attribute and execute an xml file to exclude a Test Method.

Execute Without Depends On Methods

In this example, we have 7 Test Methods. I placed set up Chrome in a Test Annotation. We want to test, we can set up Chrome. In addition, the other Test Methods are Open Orange HRM, Sign In, Search User, Search Employee, Search Candidate, and Sign Out. This example does not have a dependsOnMethods attribute. Here's the Test Steps with our Test Application. After signing in we will go to the Admin tab and Search for a User, go to the PIM tab and search for an employee, go to the Recruitment tab and search for a candidate then sign out.

Let's run our Test Script and see what happens. This site can't be reached. Refused to connect. Go to the Console. There's a cascade of failures. A cascade failure is when 1 failure force the remaining test in the suite to fail. We see 7 runs and 6 of those runs are Failures. Why do we have 6 Failures? We could not connect to the application to start our testing.

That's not right and the Test Report is not correct. We should have 1 Pass, 1 Failure, and 5 Skips. It should be 5 Skips because we could not open the OrangeHRM application to execute the last 5 Test Methods: Sign In, Search User, Search Employee, Search Candidate, and Sign Out. To solve this problem, we can use the Depends On Methods Attribute.

Execute With Depends On Methods

If we go to the annotations package, we can find the Depends On Methods attribute in a Configuration Annotation along with the groups and Depends On Groups attribute. However, we are going to use the Test Annotation that also have the Depends On Methods attribute, Depends On Groups attribute, and Groups attribute.

We add the attribute within parenthesis after the annotation. Lowercase d for dependsOnMethods equals. Next, we write the name of the Test Method that this Test Method depends on. Test2_OpenOrangeHRM depends on Test 1_SetUpChrome. Hover over the dependsOnMethods attribute and the description states "The list of methods this method depends on". Test3_SignIn depends on Test 2_OpenOrangeHRM and the list goes on. Test 4_Search User depends on Test 3_SignIn. There are some cases where a Test Method depend on more than one Test Method. It's not required in this Test Script but I want to show you how to add multiple Test Methods to depend on. Test 5 depends on Test 2 and Test 3. We surround the Test Methods with curly brackets and separate the Test Methods with a comma. The same for Test 6 and Test 7. Copy and Paste.

Let's Run. Now, we see 1 Failure and 5 Skips. Set Up Chrome is the only Test Method that Passed. The Results tab also shows 5 Skips.

Exclude Test Method via xml File

Exclude a Test Method using an xml File. I converted the dependsOnMethods_PASS class file into this xml file called Depends On Methods.

By default, an xml file does not include a methods tag because it's optional. For our example, I added an opening and closing methods tag then exclude a Test Method. That Test Method is test4_SearchUser.

Let's imagine, everyone in the organization know that searching for a user is not working. The defect has been reported and we will add it back to our Test Suite once the defect has been resolved. In the meantime, we don't have to keep executing that Test. Exclude it for now.

Let's Run. There are zero Failures, zero Skips, and we do not see Test 4 Search For User. Go to the Results tab, no Test 4 because it was not included at Runtime. The Depends On Methods attribute is great for small test when 1 Test Method depends on another Test Method. Next, is Chapter 6.3, groups and Depends On Groups Attribute which makes our Dependency Testing more robust and easy to scale.

Chapter 6.3 / groups and Depends On Groups Attribute

Introduction

Welcome to Chapter 6.3, Groups and Depends On Groups Attribute. In this chapter, we will discuss the Groups Attribute, Depends On Groups Attribute, and Execute Groups At Runtime using an xml File.

Groups Attribute

The Groups Attribute is a way to add any number of Test Methods to a named list. This feature is beneficial because it's an opportunity to run only those lists of Test Methods according to their group name. The group names can be a module of our application, a certain Test Type, whatever we decide. Plus, one Test Method can be a part of more than one group. For example, you can add the Sign In Test Method to a group for Smoke Test and to another group for Regression Test. Smoke Test is a Test Type that makes sure the most important functions in an application still work after a build is deployed to an environment. Regression is when a set of Test Scripts are re-executed to make sure the existing functions are still working after an application change.

In this example, test1_SetUpChrome is added to a group called initialize. The description for groups show "The list of groups this class/method belongs to". A class can also be added to a group.

Depends On Groups Attribute

The Depends On Groups Attribute is similar to the Depends On Methods attribute. Depends On Groups allow us to define a group name that our Test Method depends on. The next Test Method test2_OpenOrangeHRM depends on the initialize group. Hover over the attribute. The description states "The list of groups this method depends on. Every method member of one of these groups is guaranteed to have been invoked before this method."

Also, this Test Method is part of a group called env_ application. Let's pretend there is a different URL for each environment such as DEV Environment, QA Environment, and Production Environment. So, we have a group name for those environments.

We are not forced to add our Test Methods to a group. The remaining Test Methods do not belong to a group but only depends on a group. They depend on group env_application: test3_SignIn, test4_SearchUser, test5_SearchEmployee, test6_SearchCandidate, and test7_SignOut all depend on the same group.

Let's run. We see the application is in Spanish. It was changed. All Test Methods Pass. The Results tab also shows green for all Test Methods. Now, let's run and see what happens when there's a Failure. Could not connect to the application. There's 1 Failure and 5 Skips just like with the Depends On Methods attribute.

Execute Groups At Runtime via xml File

Execute Groups At Runtime using xml. TestNG provides a few ways to run groups at Runtime. The main way is to use an xml file but we can also use the command line or ant. Here's a fake Amazon class site with 6 Test Methods. Test 1 is Login and has been added to the smoke test group. Test 2 is Search Products and has been added to the smoke test and regression test groups. Test 3 is Place Order and it has been added to the regression test, integration test, and defect group. We have the option of being very descriptive with our group name so I added dot fix. This Test Method had a defect that should be fixed. If it is fixed then we can remove Place Order from the defect group.

Why do we have a defect group? We can add a defect group for a few reasons. When a Test Method keeps failing, we can add it to the defect group to avoid running that test because we know it's not fixed yet. On the flip side, we can execute only the defect group then verify which Test Methods are still broken.

Test 4 Send Confirmation has been added to system testing and defect.backlog group. This defect will not get fixed no time soon. Test 5 is Ship Order and has been added to the groups regression and defect.progress. Test 6 is Log Out and it has no groups.

Let's execute some scenarios for groups at runtime using the xml file – All Groups. Notice, the groups tag and run tags. The groups tag allows 2 tags. We see the run tag which list the groups that can be included or excluded. This file does not include or exclude any groups. Therefore, all groups will show up. The groups tag also allows a define tag which help us create a new group based on existing groups. Let's Run the Test Suite and we see all groups. Test Methods 1 – 6 show up and Passed.

The next xml file includes a regression group. The Console show Test Methods added to the regression group. We see other groups because those Test Methods are part of more than 1 group.

In the xml file, we can include more than 1 group at Runtime. This file includes the smoke test and system test group. There are 3 Test Methods.

Look what happens when we exclude regression. As expected, there are no Test Methods in the Console assigned to a regression group.

We have the ability to use regular expressions. In this xml file, defects have a regular expression using a dot and asterisk. Out all of those Test Methods assigned to regression. Only 1 Test Method shows up. That's because, the other Test Methods are assigned to a regression group and defect group. Therefore, it's not in the Console because defects are not included.

Run Test Methods According To Group Name

Let's recap the scenarios we covered when executing groups at Runtime. A run that has No Include and No Exclude will run all Test Methods. If we include a group like regression then only Test Methods assigned to that group will run. The same goes if we include more than 1 group. We used smoke and system. Only Test Methods assigned to those groups were executed. Excluding a group name runs all Test Methods except the Test Methods assigned to that group. Last but not least, we can use regular expressions. In this example, defect used a regular expression. If a Test Method belongs to a group that is included and excluded then it will not run because excluded wins. There were 3 Test Methods assigned to regression but only 1 showed up. The other 2 did not show up because they were assigned to regression and defect.

So far, up to this point, all of the Test Methods were executed from the same class. However, TestNG is not limited to executing a group using 1 class. We can execute groups using more than 1 class or even a package. This xml file has 2 class tags. By the way, we write the class name using the package name, dot operator sub package name, dot operator then the class name. For example, the first class name shows com.testautomationu. That's the package name and sub package name is chp6dependencyteting then we see Amazon_FakeSite. For this class, we include the regression group.

For the last class, we include env_application. Recall this class, include test2_OpenOrangeHRM that is assigned to group env_application.

Let's Run. All of the Test Methods show up. We saw the Test Script open OrangeHRM. The Results tab reflect the Console by showing both classes.

Next in Chapter 7, we will discuss Data Driven Testing which allow us to execute 1 Test Script with more than 1 set of Test Data.