Title: spring integration test fail to load context 'Another resource already exists with name dataSource'

Post Body:

I am using test annotation introduced in spring-boot 1.4.3 for my integration tests

```
@RunWith(SpringRunner.class) @SpringBootTest public class MyServiceIT { }
```

According to documentation, test context is cached and reused to speed up integration tests. This behavior is what I want since it takes significant amount of time to initialize application context. My failsafe plugin is configured with

```
<forkCount>1</forkCount> <reuseForks>true</reuseForks>
```

to allow integration tests to run in the same process to take advantage of application context caching.

Recently, I wrote a integration test used @MockBean annotation to mock behavior for some beans.

```
@RunWith(SpringRunner.class) @SpringBootTest public class AnotherServiceIT {   @MockBean   SomeService service1 }
```

While the test runs fine on it's own, when running through maven verify, multiple integration tests fails with the error message

> javax.naming.NamingException: Another resource already exists with name dataSource - pick a different name

If I skip this particular test with JUnit @Ignore annotation, everything goes back to normal.

This behavior seems to indicate that using @MockBean changes the caching behavior, and each test attempts to create its own datasource. I should also mention that I am using an **AtomikosDataSourceBean** created through **XADataSourceAutoConfiguration**.

How can I overcome this issue so my integration test can still use cached context and use `@MockBean` at the same time?

Accepted Answer: None

Highest Rated Answer:

Hmm, does SomeService relate to your Datasource in any way?

Because your context is cached and @MockBean does the following:

> used to add mocks to a Spring ApplicationContext ... Any existing single bean of the same type defined in the context will be replaced by the mock,

and

> If there is more than one bean of the requested type, qualifier metadata must be specified at field level:

```
 @RunWith(SpringRunner.class)  public class ExampleTests {   @MockBean  @Qualifier('example')  private ExampleService service;
```

Edit:

So if your SomeService is an implementation of a DataSource try adding a Qualifier. If SomeService has a DataSource in it, and you need to access some methods in it, you could try to use @Mock and specify the any objects that need to be returned either through their own mock or autowire.

```
@Mock SomeService someService;  @Mock SomeDependency mockDependency;  @Autowired OtherDependency realDependency;  @Before publ
```