

Testowanie poprzez moduły Guice

Paweł Cesar Sanjuan Szklarz

9/04/2013

Injektowanie zależności nie jest prostym mapowaniem

Mit: Injektowanie zależności to prosta mapa `Map<Type, Object>`

Injektowanie zależności nie jest prostym mapowaniem

Mit: Injektowanie zależności to prosta mapa `Map<Type, Object>`

```
<bean id="exampleBean" class="examples.ExampleBean">
  <property name="beanOne"><ref bean="anotherExampleBean"/></property>
  <property name="beanTwo" ref="yetAnotherBean"/>
  <property name="integerProperty" value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean">
  <property name="beanThree" ref="nestedBean"/>
</bean>

<bean id="nestedBean" class="examples.YetOther"/>
```

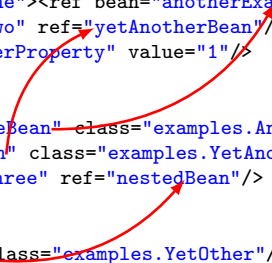
Injektowanie zależności nie jest prostym mapowaniem

Mit: Injektowanie zależności to prosta mapa `Map<Type, Object>`

```
<bean id="exampleBean" class="examples.ExampleBean">
  <property name="beanOne"><ref bean="anotherExampleBean"/></property>
  <property name="beanTwo" ref="yetAnotherBean"/>
  <property name="integerProperty" value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean">
  <property name="beanThree" ref="nestedBean"/>
</bean>

<bean id="nestedBean" class="examples.YetOther"/>
```



Konfiguracja Spring sprowadza się do `Map<String, Bean>??`

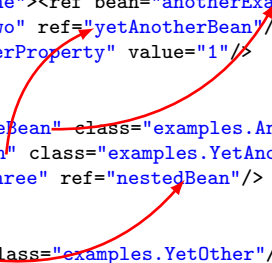
Injektowanie zależności nie jest prostym mapowaniem

Mit: Injektowanie zależności to prosta mapa `Map<Type, Object>`

```
<bean id="exampleBean" class="examples.ExampleBean">
  <property name="beanOne"><ref bean="anotherExampleBean"/></property>
  <property name="beanTwo" ref="yetAnotherBean"/>
  <property name="integerProperty" value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean">
  <property name="beanThree" ref="nestedBean"/>
</bean>

<bean id="nestedBean" class="examples.YetOther"/>
```



Konfiguracja Spring sprowadza się do `Map<String, Bean>??`

Nie, ale mało kto wie i używa pełną strukturę.

Pełna struktura - na przykład Guice

W Guice można zrobić `PrivateModule` i dać dostęp do ograniczonej listy kluczy `Key<?>`:

```
import com.google.inject.PrivateModule;
public class SimpleConfiguration extends PrivateModule {
    @Override
    protected void configure() {
        expose(ClientApi.class);
        bind(ClientApi.class).to(ClientApiImplementation.class);
        bind(ClientInternalApi.class).to(ClientInternal.class);
    }
}
```

Pełna struktura - na przykład Guice

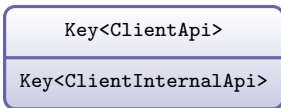
W Guice można zrobić `PrivateModule` i dać dostęp do ograniczonej listy kluczy `Key<?>`:

```
import com.google.inject.PrivateModule;
public class SimpleConfiguration extends PrivateModule {
    @Override
    protected void configure() {
        expose(ClientApi.class);
        bind(ClientApi.class).to(ClientApiImplementation.class);
        bind(ClientInternalApi.class).to(ClientInternal.class);
    }
}
```

```
@Inject
private Injector injector;
@Test
public void testExposedKeys(){
    assertNotNull(injector.getExistingBinding(Key.get(ClientApi.class)));
    assertNull(injector.getExistingBinding(Key.get(
        ClientInternalApi.class)));
}
```

Pełna struktura - na przykład Guice

Otrzymujemy injektor który "chowa" wewnętrzne klucze.



```
import com.google.inject.PrivateModule;
public class SimpleConfiguration extends PrivateModule {
    @Override
    protected void configure() {
        expose(ClientApi.class);
        bind(ClientApi.class).to(ClientApiImplementation.class);
        bind(ClientInternalApi.class).to(ClientInternal.class);
    }
}
```


Wynik: Drzewo injektorów

```
public class ComplexConfiguration extends PrivateModule {  
    @Override  
    protected void configure() {  
        expose(ClientApi.class);  
        bind(ClientApi.class).to(ClientApiImplementation.class);  
        bind(ClientInternalApi.class).to(ClientInternal.class);  
        // more bindings  
        install(new NestedModuleOne());  
        install(new NestedModuleTwo());  
    }  
}
```

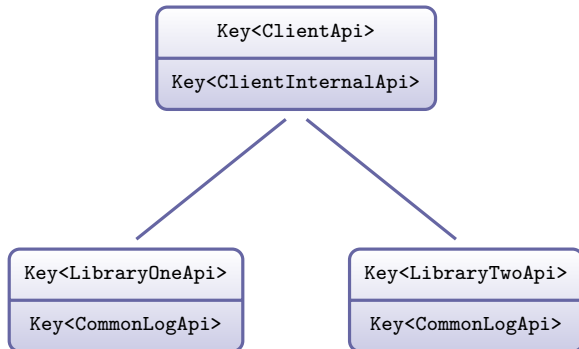
Wynik: Drzewo injektorów

```
public class NestedModuleOne extends PrivateModule {
    @Override
    protected void configure() {
        expose(LibraryOneApi.class);
        bind(LibraryOneApi.class).to(LibraryOneInternal.class);
        bind(CommonLogApi.class).toInstance(new CommonLogApi() {
            @Override
            public void log(String message) {
                System.out.println("Library ONE log:" + message);
            }
        });
    }
}
```

Wynik: Drzewo injektorów

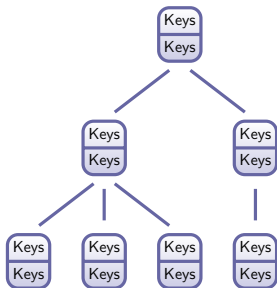
```
public class NestedModuleTwo extends PrivateModule {
    @Override
    protected void configure() {
        expose(LibraryTwoApi.class);
        bind(LibraryTwoApi.class).to(LibraryTwoInternal.class);
        bind(CommonLogApi.class).toInstance(new CommonLogApi() {
            @Override
            public void log(String message) {
                System.out.println("Library TWO log:" + message);
            }
        });
    }
}
```

Wynik: Drzewo injektorów



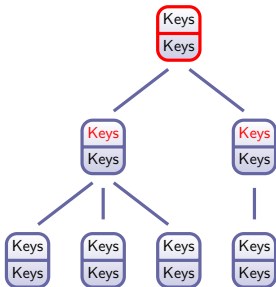
Jak testować lepiej?

- Aplikacja jako drzewo iniektorów



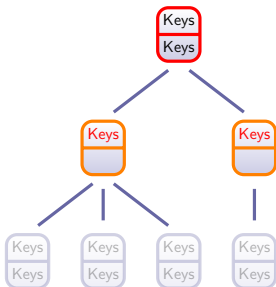
Jak testować lepiej?

- Aplikacja jako drzewo injektorów
- Izolacja komponentów w hierarchii



Jak testować lepiej?

- Aplikacja jako drzewo injektorów
- Izolacja komponentów w hierarchii
- Tworzenie testowych modułów



Przykład mockowania modułów

```
@Guice(modules = {SimpleConfiguration.class,  
    SimpleConfigurationTestIzolated.LibsMockTestModule.class})  
public class SimpleConfigurationTestIzolated {  
    public static class LibsMockTestModule extends AbstractModule {  
        static LibraryTwoApi twoApi = mock(LibraryTwoApi.class);  
        static LibraryOneApi oneApi = mock(LibraryOneApi.class);  
        @Override  
        protected void configure() {  
            bind(LibraryOneApi.class).toInstance(oneApi);  
            bind(LibraryTwoApi.class).toInstance(twoApi);  
        }  
    }  
    @Inject  
    private ClientApi clientApi;  
    @Test  
    public void testMockingInjection(){  
        clientApi.testLogging();  
        verify(LibsMockTestModule.oneApi).oneApiCall();  
        verify(LibsMockTestModule.twoApi).twoApiCall();  
    }  
}
```


A co jak injektor będzie rozproszony po wielu komputerach??

DEMO