22 | DI Container (9): 怎样重构测试代码?

> 现在的任务列表：暂时跟前一节没有什么改变，主要是做了重构

重构提取的InjectionTest

让测试的形式一致

优化测试 should_bind_type_to_a_class_with_default_constructor

- 把config.bind里面的Component.class提取成一个变量
- 把实现类也提取成一个变量
- 把bind和getContext两句提取成一个方法
- 处理提取的函数的泛型，<T, R extends R>，跑一下测试

```
private <T, R extends T> T getComponent(Class<T> type, Class<R> implementation) {
    config.bind(type, implementation);
    T component = config.getContext().get(type).get();
    return component;
}
```

- 把提取的两个变量inline回去

```
Component instance = getComponent(Component.class,
ComponentWithDefaultConstructor.class);
```

修改测试 should_bind_type_to_a_class_with_inject_constructor

- 复制上面提取的方法到这个测试里面
- 删除掉之前的获取方法，只留场景准备和绑定的代码

```
@Test
@DisplayName("should bind type to a class with inject constructor")
public void should_bind_type_to_a_class_with_inject_constructor() {
    Dependency dependency = new Dependency() {
    };
    config.bind(Dependency.class, dependency);
    Component instance = getComponent(Component.class,
ComponentWithInjectConstructor.class);
    assertNotNull(instance);
    assertSame(dependency, ((ComponentWithInjectConstructor)
instance).getDependency());
}
```

修改测试 should_bind_type_to_a_class_with_transitive_dependencies

- 也是复制方法过来，把bind Component和ComponentWithInjectConstructor的方法名，直接改成了上面提取的方法
- 删除掉了之前获取的方法，跑测试

```
@Test
@DisplayName("should bind type to a class with transitive dependencies")
public void should_bind_type_to_a_class_with_transitive_dependencies() {
    config.bind(Dependency.class, DependencyWithInjectConstructor.class);
    config.bind(String.class, "indirect dependency");


    Component instance = getComponent(Component.class,
ComponentWithInjectConstructor.class);
    assertNotNull(instance);


    Dependency dependency = ((ComponentWithInjectConstructor)
instance).getDependency();
    assertNotNull(dependency);
    assertEquals("indirect dependency", ((DependencyWithInjectConstructor)
dependency).getDependency());
}
```

修改测试 should_inject_dependency_via_field

- 把最后那个bind的行的方法名，改成getComponent
- 删除之前的config.getContext，跑测试

```
@Test
public void should_inject_dependency_via_superclass_inject_field() {
    Dependency dependency = new Dependency() {
    };
    config.bind(Dependency.class, dependency);
    SubclassWithFieldInjection component =
getComponent(SubclassWithFieldInjection.class, SubclassWithFieldInjection.class);
    assertSame(dependency, component.dependency);
}
```

修改测试 should_inject_dependency_via_superclass_inject_field

- 把最后那个bind的行的方法名，改成getComponent
- 删除之前的config.getContext，跑测试

```
@Test
public void should_inject_dependency_via_superclass_inject_field() {
    Dependency dependency = new Dependency() {
    };
    config.bind(Dependency.class, dependency);
    SubclassWithFieldInjection component =
getComponent(SubclassWithFieldInjection.class, SubclassWithFieldInjection.class);
    assertSame(dependency, component.dependency);
}
```

修改测试 should_call_inject_method_even_if_no_dependency_declared

- 把那个bind的行的方法名，改成getComponent
- 删除之前的config.getContext，跑测试

```
@Test
@DisplayName("should call inject method even if no dependency declared")
public void should_call_inject_method_even_if_no_dependency_declared() {
    InjectMethodWithNoDependency component =
getComponent(InjectMethodWithNoDependency.class,
InjectMethodWithNoDependency.class);
    assertTrue(component.called);
}
```

修改测试 should_inject_dependency_via_inject_method

- 把最后那个bind的行的方法名，改成getComponent
- 删除之前的config.getContext，跑测试

```
@Test
@DisplayName("should inject dependency via inject method")
public void should_inject_dependency_via_inject_method() {
    Dependency dependency = new Dependency() {
    };
    config.bind(Dependency.class, dependency);
    InjectMethodWithDependency component =
getComponent(InjectMethodWithDependency.class, InjectMethodWithDependency.class);
    assertSame(dependency, component.dependency);
}
```

修改测试 should_inject_dependencies_via_inject_method_from_superclass

- 把那个bind的行的方法名，改成getComponent
- 删除之前的config.getContext，跑测试

```
@Test
@DisplayName("should inject dependencies via inject method from superclass")
public void should_inject_dependencies_via_inject_method_from_superclass() {
    SubclassWithInjectMethod component =
getComponent(SubclassWithInjectMethod.class, SubclassWithInjectMethod.class);
    assertEquals(1, component.superCalled);
    assertEquals(2, component.subCalled);
}
```

修改测试 should_only_call_once_if_subclass_override_inject_method_with_inject

- 把那个bind的行的方法名，改成getComponent
- 删除之前的config.getContext，跑测试

```
@Test
@DisplayName("should only call one if subclass override inject method with
inject")
public void should_only_call_one_if_subclass_override_inject_method_with_inject()
{
    SubclassOverrideSuperclassWithInject component =
getComponent(SubclassOverrideSuperclassWithInject.class,
SubclassOverrideSuperclassWithInject.class);
    assertEquals(1, component.superCalled);
}
```

修改测试 should_not_call_inject_method_if_override_with_no_inject

- 把那个bind的行的方法名，改成getComponent
- 删除之前的config.getContext，跑测试

```
@Test
@DisplayName("should not call inject method if override with no inject")
public void should_not_call_inject_method_if_override_with_no_inject() {
    SubclassOverrideSuperClassWithNoInject component =
getComponent(SubclassOverrideSuperClassWithNoInject.class,
SubclassOverrideSuperClassWithNoInject.class);
    assertEquals(0, component.superCalled);
}
```

把对于Dependency的依赖，提出来放到setup里面去初始化

- 把new Dependency挪到外面，跟ContextConfig在同一级别
- 把config.bind放到setup里面，每一次启动都设置一下
- 然后把其它的测试方法里面dependency的都去掉

把getComponent重构成我们想要的样子

- 先在里面new ConstructorInjectionProvider<>(implementation)
- 在InjectionTest里面添加一个private的field叫context，并且直接用Mockito.mock(Context.class)赋值
- 在setup里面，mock掉对于context的读取

```
Mockit.when(context.get(eq(Dependency.class))).thenReturn(Optional.of(dependency))
```

- 把ContextConfig也改成private
- 把dependency field也改成mock的，Mockito.mock(Dependency.class)
- 在getComponent里面，直接return provider.get(context)
- inline一下provider，再跑测试，有一个失败的
- 在失败的测试里面，单独给context.get再设置一次返回项为Mockito.when(context.get(eq(Dependency.class))).thenReturn(Optional.of(new DependencyWithInjectConstructor("indirect dependency")))

```
Mockito.when(context.get(eq(Dependency.class)))
        .thenReturn(Optional.of(new DependencyWithInjectConstructor("indirect
dependency")));
```

- 删除getComponent里面的type参数，再inline回去
- should_bind_type_to_a_class_with_inject_constructor里面让instance直接是它的类型，把强制类型转换可以去掉
- should_bind_type_type_to_a_class_with_default_constructor里面的instance也可以直接写成要的类型，这样最后的instanceOf就不需要了

去掉config

- 直接把唯一一处用到config的地方注释掉，所有的测试也都是通过的
- 把setup里面的config也注释掉，测试还是通过的
- 把config那个field也删除掉
- 把mockito静态导入一下，简化代码里面的使用方法

修改测试 should_bind_type_to_a_class_with_transitive_dependencies

- 把获取的instance实例的类型改成具体的类，就可以把下面的类型强制转换去掉
- 把message判断的断言也可以删除了，已经没有什么意义了
- 然后由于设计决策的改变，这个测试跟上面的should_bind_type_to_a_class_with_inject_constructor已经差不多了，所以可以删除掉这个测试了

观察测试方法名并修改

- Should_bind_type_to_a_class_with_default_constructor改成should_call_default_constructor_if_no_inject_constructor
- should_bind_type_to_a_class_with_inject_constructor改成should_inject_dependency_via_inject_constructor

---

抽取测试分组 Injection （ConstructorInjection）

- 创建一个Nested的测试类class InjectionTest
- should_call_default_constructor_if_no_inject_constructor
- should_inject_dependency_via_inject_constructor
- should_include_dependency_from_inject_constructor
- 然后把里面的ComponentWithInjectConstructor改成InjectConstructor

抽取测试分组 IllegalInjectConstructor （ConstructorInjection）

- should_throw_exception_if_component_is_abstract
- should_throw_exception_if_component_is_interface
- should_throw_exception_if_multi_inject_constructors_provided
- should_throw_exception_if_no_inject_nor_default_constructor_provided
- 把上面4个测试跟相关的类放进创建的分组的测试类

抽取测试分组 Injection (FieldInjection)

- should_inject_dependency_via_field
- should_inject_dependency_via_superclass_inject_field
- should_include_field_dependency_in_dependencies改名成should_include_dependency_from_field_dependency

抽取测试分组 IllegalInjectFields （FieldInjection）

- should_throw_exception_if_inject_field_is_final

抽取测试分组 Injection （MethodInjection）

- should_call_inject_method_even_if_no_dependency_declared
- should_inject_dependency_via_inject_method
- should_inject_dependencies_via_inject_method_from_superclass
- should_not_call_inject_method_if_override_with_no_inject
- should_include_dependencies_from_inject_method

抽取测试分组 IllegalInjectMethod （MethodInjection）

- should_throw_exception_if_inject_method_has_type_parameter

> 测试天然并不是文档，测试天然是你实现这个过程的纪录，需要通过重构测试来让它具有文档的作用