实验2 单元测试实验

1. 实验目的

对被测模块进行单元测试

2. 实验内容

使用 JUnit 工具，针对 Spring Unit Testing 控制器代码中 ItemController 类进行测试，编 写对应的测试类以完成单元测试，最终提交测试代码。

3. 实验过程

3.1. 项目添加Junit依赖

在老师发的代码中，已经有了Springboot的依赖，现在我们只需要简单添加Junit的依赖即可，我选择了Junit4.13.2版本。

```
        <!-- 添加junit4依赖 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
```

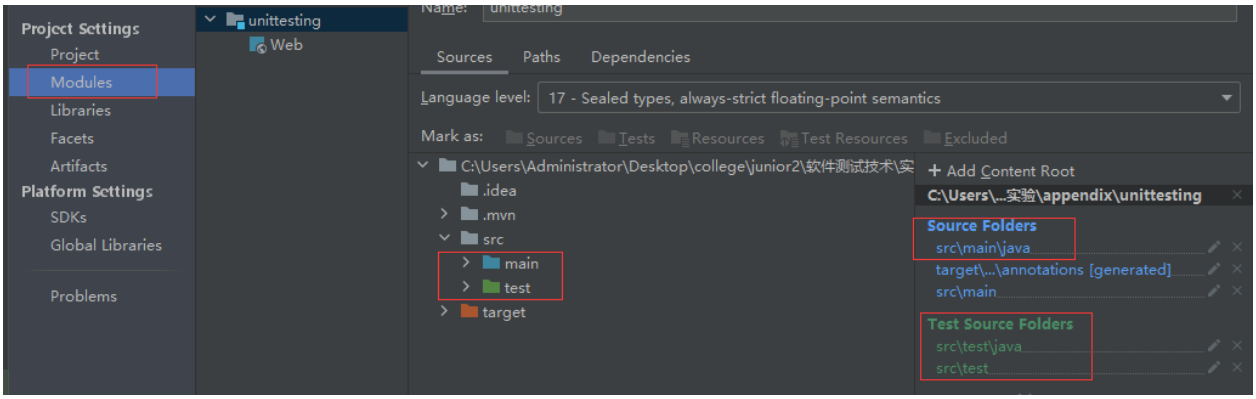
3.2. 定义Scope

其实这一步没有也可以，但为了项目的逻辑结构稍微严谨点，还是定义一下Scope。

显然，对于src下的java这个模块，就是Source（源代码）部分，而test这个模块，就是Test（测试代码）部分。

由于我用的Jetbrains旗下的集成开发环境idea，我可以利用GUI界面，定义项目各个模块所属的Scope。

Project Structure→Modules中设置main为Source，test为Test Source即可。



3.3. Junit单元测试模块配置

Junit单元测试模块，需要有Spring容器管理的Bean的上下文，并且我们要告诉这个模块的类，这是一个专门做测试的类，因此必须要有@RunWith和@SpringbootTest这两个注解。

然后我比较喜欢用一个Logger（log4j）来打印日志信息，所以我会通过LogManager为我的测试模块注入一个logger。

既然要测试的是ItemController，别忘了注入对应依赖：

```
package com.sprint.unittesting.unittesting;

import com.sprint.unittesting.unittesting.controller.ItemControl
import com.sprint.unittesting.unittesting.model.Item;
import jakarta.annotation.Resource;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringJUnit4Class

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
```

```
public class ItemControllerUnitTest {
    private static final Logger logger = LogManager.getLogger(It

    @Resource
    private ItemController itemController;

        // the test env is ready , do some tests!
}
```

3.4. 设计测试用例

到现在为止，所有配置方面的工作就做完了。接下来可以正式进入测试工作了。想做单元测试，一定要翻阅被测单元的源码，根据源代码和各种覆盖指标设计测试用例。

ItemController下包含三个接口

1. dummy-item：返回一个id=1,name="Ball",width=10,height=100的Item对象实例。
2. item-from-business-service：返回ItemBusinessService接口中retreiveHardcodedItem的一个Item实例，id=2,name="Table",width=120,height=200
3. all-items-from-database：返回ItemBusinessService接口中retrieveAllItems接口返回的一个Item对象的列表，下标i从0开始，共计5个元素，每个元素id=i+1,name="TestItem"+i,width=(i+1)*10,height=(i+1)*20。

我个人感觉这几个接口测试起来还是比较简单的，因为被测模块中没有分支、跳出某个循环、打断某个循环、多线程、线程通信之类的代码。因此覆盖指标上，就不存在语句覆盖、条件覆盖、判定覆盖、路径覆盖这些内容，因为没有任何分支的代码。我们只需要调用接口，判断返回的对象是否符合源码中的描述即可。

手动比较各个字段的工具方法

在进行测试之前，我写了个两个Item实例比较的方法，用来帮助测试

```
    // cmp Item src and target
    // both of them should not be null
    private boolean itemEqual(Item src,Item target){
        assertNotNull(target);
```

```
        logger.info("testing source item is not null");
        assertNotNull(src);

        logger.info("testing source item id is "+target.getId()
        assertEquals(src.getId(),target.getId());

        logger.info("testing source item name is "+target.getNam
        assertEquals(src.getName(),target.getName());

        logger.info("testing source item width is "+target.getWi
        assertEquals(src.getWidth(),target.getWidth());

        logger.info("testing source item height is "+target.getH
        assertEquals(src.getHeight(),target.getHeight());

        logger.info("2 items are equal");

        return true;
    }
```

Junit作为一个成熟的单测框架，已经提供好了断言API，不用再使用Java原生的assert了。

利用反射比较各个字段的工具方法

不过后来我觉得这么写实在太不优雅了，如果有上百个字段，难道还要一个一个比较吗？我就改成反射了。

```
    public void fieldNotEqual(String methodName,Object srcValue,
        logger.error("Field accessed by {} is not equal. Src val
                methodName, srcValue, targetValue);
    }

    // use Reflective to cmp src and target
    public boolean ReflectiveItemEqual(Item src,Item target){
```

```
        assertNotNull(target);
        logger.info("testing source item is not null");
        assertNotNull(src);

        Class<?> clazz = src.getClass();
        Method[] methods = clazz.getMethods();

        boolean isEqual = true;

        for (Method method : methods) {
            String methodName = method.getName();
            // get* method should have a signature which length
            // exclude getClass()
            if (methodName.startsWith("get") && methodName.lengt
                try {
                    Object srcValue = method.invoke(src);
                    Object targetValue = method.invoke(target);

                    // for case:String
                    if (srcValue == null && targetValue == null)
                        continue;
                    }
                    // for case:String
                    if (srcValue == null || targetValue == null)
                        fieldNotEqual(methodName,srcValue,target
                        isEqual = false;
                        continue;
                    }

                    if(!srcValue.equals(targetValue)){
                        fieldNotEqual(methodName,srcValue,target
                        isEqual = false;
                        continue;
                    }

                    logger.info("Testing field accessed by {} is
```

```
                } catch (Exception e) {
                    logger.error("Error while invoking method {}
                    isEqual = false;
                }
            }
        }

        if (isEqual) {
            logger.info("2 items are equal");
        } else {
            logger.error("2 items are not equal");
        }
        return isEqual;
    }
```

dummy-item单次功能测试

接口功能：返回一个id=1,name=”Ball”,width=10,height=100的Item对象实例

测试目的：测试单次调用该接口是否能返回符合功能描述的值

测试用例：

1. 输入：无
2. 输出：若接口功能正确，应返回一个符合以上描述的对象实例

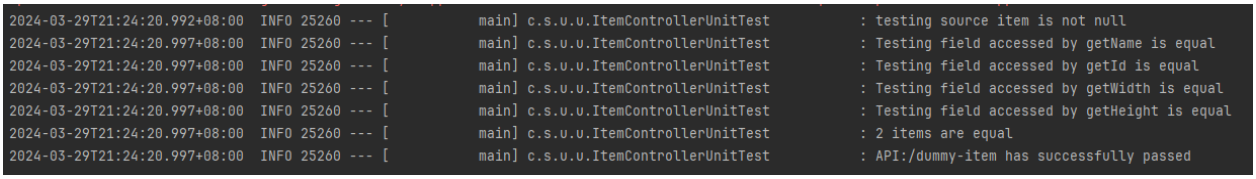
因此测试用例设计上，我们不给这个接口任何输入，然后判断接口的返回值是否符合功能描述即可。

```
    @Test
    public void testDummyItem() {
        Item dummy = itemController.dummyItem();

//        itemEqual(dummy,new Item(1, "Ball", 10, 100));
        boolean flag = ReflectiveItemEqual(dummy,new Item(1, "Ba

        if (flag){
            logger.info("API:/dummy-item has successfully passed
```

```
        }else{
            logger.error("API:/dummy-item has failed");
        }
    }
```

你可以解开我对itemEqual的注释，采用非反射的方式比较字段。不过我测试的时候还是用的反射。

测试结果如下：



该测试用例通过。

dummy-item百万次调用耗时测试

测试目的：测试调用接口百万次耗时（接口效率）

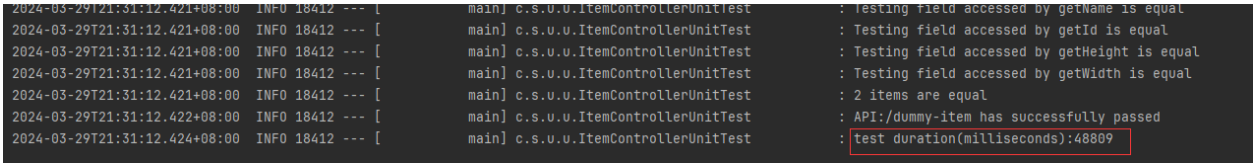
老样子，不给输入，直接看调用百万次的系统时间之差即可。

```
    @Test
    public void testDummyItem1e6(){
        long start = System.nanoTime();

        Item cmp = new Item(1, "Ball", 10, 100);
        boolean flag = true;
        for (int i = 0; i < (int) 1e6; i++) {
            flag &= ReflectiveItemEqual(itemController.dummyIte
        }

        long end = System.nanoTime();
        long duration = TimeUnit.NANOSECONDS.toMillis(end-start
        if(!flag){
            logger.error("API:/dummy-item has failed during its
        }else{
```

```
            logger.info("API:/dummy-item has successfully passed
        }

        logger.info("test duration(milliseconds):"+duration);
    }
```

我这里的测试，保证了每次调用dummyItem()后，都比较了一遍字段与符合功能描述的字段，并且把比较结果log了出来。1e6次反射+log，会大大降低调用接口的效率。如果你不想看每个字段是否正确，只是单纯地想知道调这个接口1e6次是什么样子，可以简单地修改flag&=处为注释的内容。

测试结果如下：



用了48809毫秒，其实严谨一点应该把这个耗时测试在不同硬件环境下分别跑1e6次，再分别取个平均值，我电脑硬件有点跟不上这种测试，作罢。

item-from-business-service单次功能测试

接口功能：返回ItemBusinessService接口中retreiveHardcodedItem的一个Item实例，id=2,name="Table",width=120,height=200

测试目的：测试单次调用该接口是否能返回符合功能描述的值

测试用例：

1. 输入：无
2. 输出：若接口功能正确，应返回一个符合以上描述的对象实例

```
    @Test
    public void testItemFromBusinessService(){
        boolean flag = ReflectiveItemEqual(itemController.itemFr

        if (flag){
            logger.info("API:/item-from-business-service has suc
```

```
        }else{
            logger.error("API:/item-from-business-service has fa
        }

    }
```

```
2024-03-29T21:41:03.774+08:00  INFO 21560 --- [          main] c.s.u.u.ItemControllerUnitTest          : testing source item is not null
2024-03-29T21:41:03.777+08:00  INFO 21560 --- [          main] c.s.u.u.ItemControllerUnitTest          : Testing field accessed by getName is equal
2024-03-29T21:41:03.777+08:00  INFO 21560 --- [          main] c.s.u.u.ItemControllerUnitTest          : Testing field accessed by getId is equal
2024-03-29T21:41:03.777+08:00  INFO 21560 --- [          main] c.s.u.u.ItemControllerUnitTest          : Testing field accessed by getHeight is equal
2024-03-29T21:41:03.777+08:00  INFO 21560 --- [          main] c.s.u.u.ItemControllerUnitTest          : Testing field accessed by getWidth is equal
2024-03-29T21:41:03.777+08:00  INFO 21560 --- [          main] c.s.u.u.ItemControllerUnitTest          : 2 items are equal
2024-03-29T21:41:03.777+08:00  INFO 21560 --- [          main] c.s.u.u.ItemControllerUnitTest          : API:/item-from-business-service has successfully passed
```

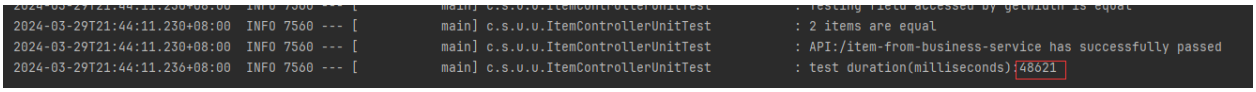
item-from-business-service百万次调用耗时测试

测试目的：测试调用接口百万次耗时（接口效率）

```
    @Test
    public void testItemFromBusinessService1e6(){
        long start = System.nanoTime();

        Item cmp = new Item(2, "Table", 120, 200);
        boolean flag = true;
        for (int i = 0; i < (int) 1e6; i++) {
//            itemController.itemFromBusinessService()
            flag &= ReflectiveItemEqual(itemController.itemFromB
        }

        long end = System.nanoTime();
        long duration = TimeUnit.NANOSECONDS.toMillis(end-start)
        if(!flag){
            logger.error("API:/item-from-business-service has fa
        }else{
            logger.info("API:/item-from-business-service has suc
        }
```

```
        logger.info("test duration(milliseconds):"+duration);
    }
```

测试结果如下：



用了48621ms。

all-items-from-database单次功能测试

接口功能：返回ItemBusinessService接口中retrieveAllItems接口返回的一个Item对象的列表，下标i从0开始，共计5个元素，每个元素id=i+1,name="TestItem"+i,width=(i+1)*10,height=(i+1)*20。

测试目的：测试单次调用该接口是否能返回符合功能描述的值

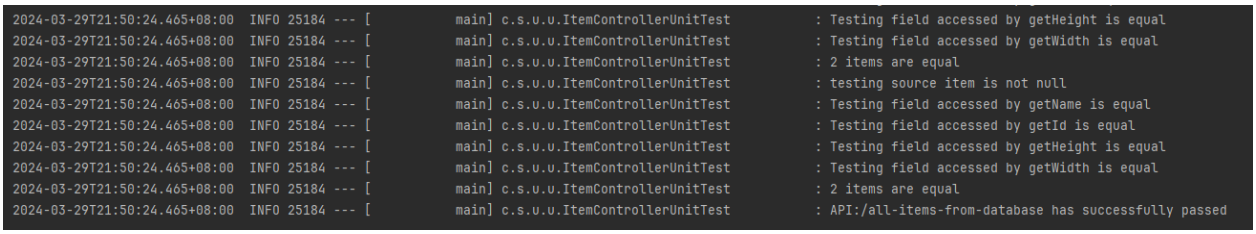
测试用例：

1. 输入：无
2. 输出：若接口功能正确，应返回一个符合以上描述的对象实例

```
    @Test
    public void testRetrieveAllItems(){
        List<Item> items = itemController.retrieveAllItems();

        boolean flag = true;
        for (int i = 0; i < items.size(); i++) {
            flag &= ReflectiveItemEqual(items.get(i),new Item(i+
        }

        if(!flag){
            logger.error("API:/all-items-from-database has faile
        }else{
            logger.info("API:/all-items-from-database has succes
```

```
        }
    }
```

测试结果如下：



all-items-from-database百万次调用耗时测试

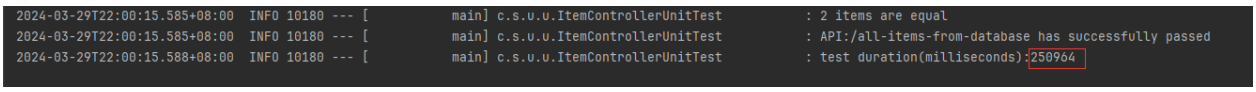
测试目的：测试调用接口百万次耗时（接口效率）

```
    @Test
    public void testRetrieveAllItems1e6(){
        long start = System.nanoTime();

        List<Item> list = new ArrayList<>();

        for(int i = 0;i < 5;i++){
            list.add(new Item(i+1,"TestItem" + i,(i+1)*10,(i+1)*
        }


        boolean flag = true;
        for (int i = 0; i < (int) 1e6; i++) {
//            itemController.retrieveAllItems()
            List<Item> items = itemController.retrieveAllItems()
            for (int j = 0; j < items.size(); j++) {
                flag &= ReflectiveItemEqual(items.get(j),list.ge
            }
        }

        long end = System.nanoTime();
        long duration = TimeUnit.NANOSECONDS.toMillis(end-start)
```

```
        if(!flag){
            logger.error("API:/all-items-from-database has faile
        }else{
            logger.info("API:/all-items-from-database has succes
        }

        logger.info("test duration(milliseconds):"+duration);
    }
```

实际上这里就是把两个Item对象进行比较改成了两个长度相等的Item对象的列表进行比较，两个列表相等当且仅当列表里的元素逐项相等。

测试结果如下：（这个测试用例跑了我很长时间）



测试耗时：250964ms

3.5. 扩展

中间件Mockito可以mock数据，创建Mock对象来模拟依赖对象的行为，进行单元测试。

给个例子：

```
    @Test
    public void testItemFromBusinessService() throws Exception {
        Item mockItem = new Item(2, "Table", 120, 200);
        given(businessService.retreiveHardcodedItem()).willRetur

        mockMvc.perform(get("/item-from-business-service")
                        .accept(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(2))
                .andExpect(jsonPath("$.name").value("Table"))
                .andExpect(jsonPath("$.width").value(120))
                .andExpect(jsonPath("$.height").value(200));
```

```
        verify(businessService, times(1)).retreiveHardcodedItem(
    }
```

实际上Mockito的API已经非常接近于人类语言了，比如given().willReturn()代码，意即：

1. given()：静态导入的方法，用于配置模拟对象的行为。当某个特定的方法调用发生时，使用 `given` 来定义该方法应该如何响应。
2. willReturn()：定义了当 `retreiveHardcodedItem` 方法被调用时，它应该返回 `mockItem`

这句话翻译成人类语言就是，当我调用retreiveHardcodedItem方法时，返回的值应该和mockItem一致。

而perform即调用接口的方法，andExpect是链式调用，期望某个字段的值与指定值相等。

而verify语句用来验证 `businessService` 模拟对象的 `retreiveHardcodedItem` 方法在测试中确实被调用了一次。

如果在测试中 `retreiveHardcodedItem` 方法没有被调用，或者调用的次数不是一次，那么Mockito 将抛出一个异常，指示验证失败。这有助于确保测试代码确实按照预期与模拟对象进行了交互。这是我自己编写的代码做不到的。

整个基于Junit与Mockito的测试类如下：

```
package com.sprint.unittesting.unittesting;

import com.sprint.unittesting.unittesting.business.ItemBusinessS
import com.sprint.unittesting.unittesting.controller.ItemControl
import com.sprint.unittesting.unittesting.model.Item;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilder
```

```
import java.util.Arrays;
import java.util.List;

import static org.mockito.BDDMockito.given;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.springframework.test.web.servlet.request.MockM
import static org.springframework.test.web.servlet.result.MockMv
import static org.springframework.test.web.servlet.result.MockMv

@ExtendWith(MockitoExtension.class)
public class ItemControllerMockitoTest {

    @Mock
    private ItemBusinessService businessService;

    @InjectMocks
    private ItemController itemController;

    private MockMvc mockMvc;

    @BeforeEach
    public void setup() {
        mockMvc = MockMvcBuilders.standaloneSetup(itemControlle
    }

    @Test
    public void testDummyItem() throws Exception {
        mockMvc.perform(get("/dummy-item")
                        .accept(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(1))
                .andExpect(jsonPath("$.name").value("Ball"))
                .andExpect(jsonPath("$.width").value(10))
                .andExpect(jsonPath("$.height").value(100));
```

```
    }

    @Test
    public void testItemFromBusinessService() throws Exception {
        Item mockItem = new Item(2, "Table", 120, 200);
        given(businessService.retreiveHardcodedItem()).willRetur

        mockMvc.perform(get("/item-from-business-service")
                        .accept(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(2))
                .andExpect(jsonPath("$.name").value("Table"))
                .andExpect(jsonPath("$.width").value(120))
                .andExpect(jsonPath("$.height").value(200));

        verify(businessService, times(1)).retreiveHardcodedItem(
    }

    @Test
    public void testRetrieveAllItems() throws Exception {
        List<Item> mockItems = Arrays.asList(
                new Item(1, "TestItem0", 10, 20),
                new Item(2, "TestItem1", 20, 40),
                new Item(3, "TestItem2", 30, 60),
                new Item(4, "TestItem3", 40, 80),
                new Item(5, "TestItem4", 50, 100)
        );
        given(businessService.retrieveAllItems()).willReturn(moc

        mockMvc.perform(get("/all-items-from-database")
                        .accept(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$").isArray())
                .andExpect(jsonPath("$.length()").value(5));

        verify(businessService, times(1)).retrieveAllItems();
```

```
    }
}
```

4. 总结与体会

本次实验是基于Junit的单元测试。Junit可以视作是一个脚本测试的成熟框架，我们可以利用注解来限定测试环境与Spring上下文环境，完成测试类的环境配置。

在进行测试时，我们需要先阅读源代码，采用不同的覆盖指标（如语句、分支、判定、路径等覆盖指标）进行测试用例的设计与测试脚本的编写。

在编写测试脚本时，我们要尽可能地考虑到工具函数/方法的复用，例如我要比较两个同一个类的对象的各个字段是否相等，我就要考虑到当类对象的字段过多时，手动比较的方式需要耗费大量的编码时间，因此需要我们基于Java的反射机制来进行字段的比较，这样虽然牺牲了一些测试运行的时间，但是省下了巨量的编码时间，保证了测试代码的质量。

单元测试工具并不只有Junit，也包括了Mockito这样的Mock对象的工具，以及Coco这样的测试覆盖率工具。当我们集成一些高效的测试工具到我们的测试环境中，将更好地、更全面地对被测单元进行测试。

5. 附录

整个测试类代码如下：

```
package com.sprint.unittesting.unittesting;

import com.sprint.unittesting.unittesting.controller.ItemControl
import com.sprint.unittesting.unittesting.model.Item;
import jakarta.annotation.Resource;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringJUnit4Class
```

```
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class ItemControllerUnitTest {
    private static final Logger logger = LogManager.getLogger(It

    @Resource
    private ItemController itemController;

    // cmp Item src and target
    // both of them should not be null
    private boolean itemEqual(Item src,Item target){
        assertNotNull(target);

        logger.info("testing source item is not null");
        assertNotNull(src);

        logger.info("testing source item id is "+target.getId())
        assertEquals(src.getId(),target.getId());

        logger.info("testing source item name is "+target.getNam
        assertEquals(src.getName(),target.getName());

        logger.info("testing source item width is "+target.getWi
        assertEquals(src.getWidth(),target.getWidth());

        logger.info("testing source item height is "+target.getH
        assertEquals(src.getHeight(),target.getHeight());
```

```
        logger.info("2 items are equal");

        return true;
    }

    public void fieldNotEqual(String methodName,Object srcValue,
        logger.error("Field accessed by {} is not equal. Src val
                methodName, srcValue, targetValue);
    }

    // use Reflective to cmp src and target
    public boolean ReflectiveItemEqual(Item src,Item target){
        assertNotNull(target);
        logger.info("testing source item is not null");
        assertNotNull(src);

        Class<?> clazz = src.getClass();
        Method[] methods = clazz.getMethods();

        boolean isEqual = true;

        for (Method method : methods) {
            String methodName = method.getName();
            // get* method should have a signature which length
            // exclude getClass()
            if (methodName.startsWith("get") && methodName.lengt
                try {
                    Object srcValue = method.invoke(src);
                    Object targetValue = method.invoke(target);

                    // for case:String
                    if (srcValue == null && targetValue == null)
                        continue;
                    }
                    // for case:String
                    if (srcValue == null || targetValue == null)
```

```
                        fieldNotEqual(methodName,srcValue,target
                        isEqual = false;
                        continue;
                    }

                    if(!srcValue.equals(targetValue)){
                        fieldNotEqual(methodName,srcValue,target
                        isEqual = false;
                        continue;
                    }

                    logger.info("Testing field accessed by {} is
                } catch (Exception e) {
                    logger.error("Error while invoking method {}
                    isEqual = false;
                }
            }
        }

        if (isEqual) {
            logger.info("2 items are equal");
        } else {
            logger.error("2 items are not equal");
        }
        return isEqual;
    }

    @Test
    public void testDummyItem() {
        Item dummy = itemController.dummyItem();

//        itemEqual(dummy,new Item(1, "Ball", 10, 100));
        boolean flag = ReflectiveItemEqual(dummy,new Item(1, "Ba

        if (flag){
            logger.info("API:/dummy-item has successfully passed
```

```
        }else{
            logger.error("API:/dummy-item has failed");
        }
    }

    @Test
    public void testDummyItem1e6(){
        long start = System.nanoTime();

        Item cmp = new Item(1, "Ball", 10, 100);
        boolean flag = true;
        for (int i = 0; i < (int) 1e6; i++) {
//            itemController.dummyItem();
            flag &= ReflectiveItemEqual(itemController.dummyIten
        }

        long end = System.nanoTime();
        long duration = TimeUnit.NANOSECONDS.toMillis(end-start)
        if(!flag){
            logger.error("API:/dummy-item has failed during its
        }else{
            logger.info("API:/dummy-item has successfully passed
        }

        logger.info("test duration(milliseconds):"+duration);
    }

    @Test
    public void testItemFromBusinessService(){
        boolean flag = ReflectiveItemEqual(itemController.itemFr

        if (flag){
            logger.info("API:/item-from-business-service has suc
        }else{
            logger.error("API:/item-from-business-service has fa
        }
```

```
    }

    @Test
    public void testItemFromBusinessService1e6(){
        long start = System.nanoTime();

        Item cmp = new Item(2, "Table", 120, 200);
        boolean flag = true;
        for (int i = 0; i < (int) 1e6; i++) {
//            itemController.itemFromBusinessService()
            flag &= ReflectiveItemEqual(itemController.itemFromB
        }

        long end = System.nanoTime();
        long duration = TimeUnit.NANOSECONDS.toMillis(end-start)
        if(!flag){
            logger.error("API:/item-from-business-service has fa
        }else{
            logger.info("API:/item-from-business-service has suc
        }

        logger.info("test duration(milliseconds):"+duration);
    }

    @Test
    public void testRetrieveAllItems(){
        List<Item> items = itemController.retrieveAllItems();

        boolean flag = true;
        for (int i = 0; i < items.size(); i++) {
            flag &= ReflectiveItemEqual(items.get(i),new Item(i+
        }

        if(!flag){
            logger.error("API:/all-items-from-database has faile
```

```
        }else{
            logger.info("API:/all-items-from-database has succes
        }
    }

    @Test
    public void testRetrieveAllItems1e6(){
        long start = System.nanoTime();

        List<Item> list = new ArrayList<>();

        for(int i = 0;i < 5;i++){
            list.add(new Item(i+1,"TestItem" + i,(i+1)*10,(i+1)*
        }


        boolean flag = true;
        for (int i = 0; i < (int) 1e6; i++) {
//            itemController.retrieveAllItems()
            List<Item> items = itemController.retrieveAllItems()
            for (int j = 0; j < items.size(); j++) {
                flag &= ReflectiveItemEqual(items.get(j),list.ge
            }
        }

        long end = System.nanoTime();
        long duration = TimeUnit.NANOSECONDS.toMillis(end-start)
        if(!flag){
            logger.error("API:/all-items-from-database has faile
        }else{
            logger.info("API:/all-items-from-database has succes
        }

        logger.info("test duration(milliseconds):"+duration);
```

```
    }
}
```