스프링 테스트
- 서비스 객체 테스트 -

서비스 객체 테스트

❖ 스프링 테스트 의존성

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>1.9.5</version>
    <scope>test</scope>
</dependency>

<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-test</artifactId>
   <version>${org.springframework-version}</version>
   <scope>test</scope>
</dependency>
```

서비스 객체 테스트

❖ 스프링 테스트 의존성

```
<!-- 웹 소켓 테스트 -->
<dependency>
   <groupId>org.apache.tomcat.embed</groupId>
   <artifactId>tomcat-embed-websocket</artifactId>
   <version>7.0.52</version>
   <scope>test</scope>
</dependency>

<dependency>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
   <version>1.2.15</version>
</dependency>
```

스프링 테스트
- Dao, Service 테스트 -

❖ Dao, Service 테스트

- SpringCommon 프로젝트에서 테스트
- resources/database-context.xml
  - Aquarius의 root-context.xml의 내용을 여기로 이동

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …>
        :

   <bean id="sqlSessionFactory"
         class="org.mybatis.spring.SqlSessionFactoryBean">
      :
   </bean>

   <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
      <property name="basePackage"
            value="edu.iot.**.dao" />
   </bean>
</beans>
```

❖ Dao, Service 테스트

- SpringCommon 프로젝트에서 테스트
- resources/common-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …>

   <context:component-scan base-package="edu.iot.common" />
   <tx:annotation-driven transaction-manager="transactionManager" />

</beans>
```

Dao, Service 테스트

❖ 테스트 클래스 기본 골격

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(
   locations = { // 스프링 설정 파일 배열
      "classpath:config/*-context.xml",
   })
public class BaseTest {

}
```

Dao, Service 테스트

❖ Dao 테스트를 위한 기반 클래스

```
public class BaseDaoTest<D> extends BaseTest {
   @Autowired
   D dao;

   @Test
   public void test() {
      assert dao!= null :
         dao.getClass().getSimpleName() + " 바인딩 실패";
   }
}
```

Dao, Service 테스트

❖ MemberDao 테스트

```
public class MemberDaoTest extends BaseDaoTest<MemberDao> {

   @Test
   public void testCount() throws Exception {
      int count = dao.count();

      assert count > 0 : "count의 값은 0이상이어야 합니다.";
   }

   @Test
   public void testGetPage() throws Exception {
      Map<String, Object> map = new HashMap<>();
      map.put("start", 1);
      map.put("end", 10);
      List<Member> list = dao.getPage(map);

      assert list.size() > 0 : "getPage()의 길이가 0이상이어야 합니다.";
   }
}
```

Dao, Service 테스트

❖ Service 테스트를 위한 기반 클래스

```
public class BaseDaoTest<S> extends BaseTest {
   @Autowired
   S service;

   @Test
   public void test() {
      assert service!= null :
         service.getClass().getSimpleName() + " 바인딩 실패";
   }
}
```

Dao, Service 테스트

❖ MemberService 테스트

```
public class MemberServiceTest
        extends BaseServiceTest<MemberService>{

   @Test
   public void testGetPage() throws Exception {
      Map<String, Object> map = service.getPage(1);

      assert map.containsKey("pagination") :
               "Pagination 객체가 없습니다.";

      assert map.containsKey("list") : "List 객체가 없습니다.";
   }

}
```

❖ Spring MVC 테스트

- SpringMvc 프로젝트에서 테스트

Spring MVC 테스트

❖ HomController 테스트

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(
   locations = { // 스프링 설정 파일 배열
      "classpath:servlet-context.xml",
      "classpath:config/*-context.xml"
   })
@WebAppConfiguration
public class BaseMvcTest<C> {
   @Autowired
   C controller;
   @Test
   public void test() {
      assert controller!= null :
         controller.getClass().getSimpleName() + " 바인딩 실패";
   }

}
```

Spring MVC 테스트

❖ MemberServiceTest

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration( // 스프링 설정 클래스 배열
   classes= {WebConfig.class, RootConfig.class})
@WebAppConfiguration
public class HomeControllerTest {
   @Autowired
   HomeController controller;

   @Test
   public void testController() {
      assert controller != null : " controller 바인딩 실패";
   }
}
```

Spring MVC 테스트

```
public class HomeControllerTest
         extends BaseMvcTest<HomeController>{

   @Test
   public void listTest() throws Exception{
      MockMvc mockMvc = MockMvcBuilders
                           .standaloneSetup(controller)
                           .build();

      mockMvc.perform(get("/"))          // GET / 요청
         .andDo(print())                 // 처리 내용 출력
         .andExpect(status().isOk())     // 상태값은 OK
         .andExpect(view().name("home")) // 뷰의 이름은 home
         // "serverTime" 이름의 attribute 존재
         .andExpect(model().attributeExists("serverTime"));
   }
}
```

Spring MVC 테스트

❖ 결과

```
MockHttpServletRequest:
         HTTP Method = GET
         Request URI = /
          Parameters = {}
             Headers = {}

             Handler:
                Type = edu.iot.app.api.HomeController
              Method = public java.lang.String
edu.iot.app.api.HomeController.home(java.util.Locale,org.springframew
ork.ui.Model)

               Async:
       Async started = false
        Async result = null
```

Spring MVC 테스트

❖ 결과

```
  Resolved Exception:
                Type = null

         ModelAndView:
            View name = home
                 View = null
            Attribute = serverTime
                value = October 19, 2018 10:10:57 AM KST

             FlashMap:

MockHttpServletResponse:
               Status = 200
        Error message = null
              Headers = {}
         Content type = null
                 Body =
        Forwarded URL = home
       Redirected URL = null
              Cookies = []
```

❖ MockMvc

- perform()
  - 해당 컨트롤러에게 url로 요청
  - ResultActions 인터페이스 리턴

```
mockMvc.perform(get("/")) // basic
mockMvc.perform(post("/")) // send post
mockMvc.perform(get("/?foo={var}", "1")) // query string
mockMvc.perform(get("/").param("bar", "2")) // using param
mockMvc.perform(get("/").accept(MediaType.ALL)) // select
media type
```

  - post(), put(), delete() 등도 가능

❖ ResultActions 인터페이스

- andDo()
  - 요청에 대한 처리 실행
  - 일반적으로 print() 메소드 사용
- andExpect()
  - 예상 값 검증

```
// 200 응답 기대
.andExpect(status().isOk())
// contentType 검증
.andExpect(content().contentType("application/json;charset=utf-8"))
```

❖ ResultActions 인터페이스

- andReturn()
  - 테스트한 결과 객체를 받을 때 사용

```
MvcResult result = mockMvc
                    .perform(get("/"))
                    .andDo(print())
                    .andExpect(status().isOk())
                    .andExpect(model().attributeExists("serverTime"))
                    .andReturn();
```

❖ MockMvcResultMatchers

- status()
- view()
- model()
- header()
- flash()
- cookie()
- content()
- redirectedUrl()
- jsonPath()

```
/*
   {"message":"val"} 이라는 response를 받았는지 검증
*/
.andExpect(jsonPath("$.message").value("val"))
```

❖ Springboot 테스트

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(classes = DemoApplication.class)
@WebAppConfiguration
public class TestClass {
   @Autowired
   private WebApplicationContext context;
   private MockMvc mvc;

   @Before
   public void setup(){
      this.mvc = MockMvcBuilders
                  .webAppContextSetup(this.context)
                  .build();
   }

}
```