http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:dev2:tst:mvc_test

MVC Test

Outline

Spring MVC test through API is suppored in Spring MVC Test Framework MVC Test Framework can internally load real spring component via TestContext and perform MVC test without using execution ServletContext.

Spring mvc server-side test

Previous method used until Spring version 3.2 to test Spring MVC Controller was to objectify the Controller or to insert the Controller as an object, then use a mock object(MockHttpServletRequest, MockHttpServletResponse) to compose a unit test.
However, the test method fails to support/confirm all logics involved in annotation functions and request processings within the Controller. (@initBinder, @ModelAttribute, @ExceptionHandler, etc.)

A more convenient method of conducting Spring MVC Test in Spring MVC Test Framework is provided in versions released after Spring 3.2.

Spring MVC Test is based on "Mock" realization, and operates without the execution of sublet container. Therefore, Request and Response processing, with the exception of JSP rendering, is supported. However, Forward/Redirect does not actually operate; the URL summoned by "Forward" or "Redirect" is saved, and the expected value can be confirmed within the test code.

View types such as Freemarker, Velocity, Thymeleaf, as well as JSP, is supported in Spring MVC Test. Vaious processing methods such as HTML, JSON, XML type rendering are also supported.

Let's take a look at an example of a test code before exporing more on Spring MVC test.
The following is an example of a JSON request:

```
importstatic org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
importstatic org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
importstaticorg.springframework.test.web.servlet.MockMvcBuilder.*;

@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration("test-servlet-context.xml")
publicclassExampleTests{

    @Autowired
privateWebApplicationContextwac;

privateMockMvcmockMvc;

    @Before
publicvoid setup(){
this.mockMvc=MockMvcBuilders.webAppContextSetup(this.wac).build();
}
```

```
    @Test
publicvoidgetAccount()throwsException{
this.mockMvc.perform(get("/accounts/1").accept(MediaType.parseMediaType("application/json;char
set=UTF-8")))
            .andExpect(status().isOk())
            .andExpect(content().contentType("application/json"))
            .andExpect(jsonPath("$.name").value("Lee"));
}

}
```

In the code above, a request was made by "perform" from MockMvc and values received from JSON can be confirmed from the responses such as status value(Status:200) and content type("application/json”).

Setup Option

TestContext of MVC Test operates as WebApplicationcontext.
During setup before test, objects of MockMvc needed for Spring MVC Test must be imported. There are two setup options.

1. Perform SetUp by reading configurations from @ContextConfiguration

@Creating mockMvc by reading configuration from the xml of @ContextConfiguration and inserting WebApplicationContext.

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration("my-servlet-context.xml")
publicclassMyWebTests{

    @Autowired
privateWebApplicationContextwac;

privateMockMvcmockMvc;

    @Before
publicvoid setup(){
this.mockMvc=MockMvcBuilders.webAppContextSetup(this.wac).build();
}

// ...

}
```

2. Performing SetUp by creating controller object without reading Spring Configuration

As a Controller is created, a basic Spring MVC is created as well.

```
publicclassMyWebTests{

privateMockMvcmockMvc;

    @Before
publicvoid setup(){
```

```
this.mockMvc=MockMvcBuilders.standaloneSetup(newAccountController()).build();
}

// ...

}
```

Since configuration information is needed to call Beans in accordance with the components of Web MVC Layer, the first setup method is recommended.

Static Import

When using testcode, it is recommended that necessary APIs are called via Static Import declaration. For example, declare classes that are frequently used such as MockMvcRequestBuilders.*, MockMvcResultMatchers.* via Static Import.

```
importstatic org.springframework.test.web.server.request.MockMvcRequestBuilders.*;
importstatic org.springframework.test.web.server.result.MockMvcResultMatchers.*;
```

MVC Test

The following are MockMvc functions that can be used in Spring MVC testcodes.

Function	**Description**	**Example:**
perform	Request is made to the respective path. The URL and HTTP METHOD to call can be configured at this time.	.perform(get("/account/1")
param	Configure parameter	.param("key", "value")
cookie	Configure Cookie	.cookie(new Cookie("key", "value")
sessionAttr	Configure Session	sessionAttr("key", "value")
accept	Configure response accept value	.accept(MediaType.parseMediaType("application/json;charset=UTF-8")))
andExpect	Asser function of expected value.	andExpect(status().isOk()
andDo	Process request/response	andDo(print())
andReturn	Process as Return	.andReturn()

Example of Perform Function Processing

For processing Requests, perform function of MockMvc can be used to interneally configure MockHttpServletRequest value to perform a request.

```
mockMvc.perform(post("/hotels/{id}", 42).accept(MediaType.APPLICATION_JSON));
```

Aside from the HTTP method, an upload request can be performed by internally creating MockMultipartHttpServletRequest Object by using the fileUpload method.

```
mockMvc.perform(fileUpload("/doc").file("a1", "ABC".getBytes("UTF-8")));
```

Query String parameters can be designated in URI template.

```
mockMvc.perform(get("/hotels?foo={foo}", "bar"));
```

request parameters can be added as well.

```
mockMvc.perform(get("/hotels").param("foo", "bar"));
```

It is recommended that contextPath and servletPath is left out of request URL. However, when Full URI must be tested together upon request, set contextPath and servletPath for proper functioning of request mapping.

```
mockMvc.perform(get("/app/main/hotels/{id}").contextPath("/app").servletPath("/main"))
```

Since configuring contextPath and servletPath for each request would be inconvenient, it is recommended that the configuration is completed in the setup process.

```
publicclassMyWebTests{

privateMockMvcmockMvc;

    @Before
publicvoid setup(){
mockMvc=standaloneSetup(newAccountController())
            .defaultRequest(get("/")
                .contextPath("/app").servletPath("/main")
                .accept(MediaType.APPLICATION_JSON).build();
}

}
```

Example of andExpect Function Processing

andExpect function is used for expected value, and more than one can be used. MockMvcResultMatchers.* can be defined as static import to use provided functions within the andExpect function.
Two types of functions are provided in MockMvcResultMatchers.

- Response Properties Value (response status, header, content, etc.)
- Values from request processing (Exception, Model, View, request value, session value, etc.)

When checking response status,

```
mockMvc.perform(get("/accounts/1")).andExpect(status().isOk());
```

When using more than one andExpect functions,

```
mockMvc.perform(post("/persons"))
  .andExpect(status().isOk())
  .andExpect(model().attributeHasErrors("person"));
```

request results can be printed as well. When requests are processed through print method, all related data results are printed

```
mockMvc.perform(post("/persons"))
   .andDo(print())
   .andExpect(status().isOk())
   .andExpect(model().attributeHasErrors("person"));
```

Return results can be re-returned by andReturn method.

```
MvcResultmvcResult=mockMvc.perform(post("/persons")).andExpect(status().isOk()).andReturn();
```

If identical result values are expected, configure as below during setUp.

```
standaloneSetup(newSimpleController())
       .alwaysExpect(status().isOk())
       .alwaysExpect(content().contentType("application/json;charset=UTF-8"))
       .build()
```

Example of addFilters Function Processing

To register two or more Filter Instances, add filters as follows during mockMvc setting

```
mockMvc=standaloneSetup(newPersonController()).addFilters(newCharacterEncodingFilter()).build();
```

References

Spring reference 3.2.x : spring-mvc-test-framework