Asta4D

Asta4D Framework User Guide

Rui Liu
Shunsuke Otani

Table of Contents

Part I. Introduction

Asta4D is a web application framework which is friendly to designer and flexible to developer. Asta4D affords high productivity than traditional MVC architecture by View First architecture. It also allows front-end engineers and back-end engineers work independently without interference by separating rendering logic from template files.

Asta4D is inspired by lift which is a famous scala web application framework and it is developed by astamuse company Ltd. locating at Tokyo Japan. We are concentrating on global innovation support and developing Asta4D for our own services. Currently, Asta4D is driving our new service development.

1. Overview

1.1 Why Asta4D

In the past decade, plenty of Java based web application frameworks are generated. Especially the MVC architecture and JSP tag libs (or other traditional template technologies) that has greatly released our productivity. But unfortunately, we are still suffering from the following situations:

- The designers or front-end engineers are keeping complaining the mixed-in dynamic code, as they disturb their efforts of redesigning the page style or structure. And in the mean time, the back-end developers are also complaining that the front-end guys break the working page frequently, because redesign or the new design is hard to merge due to the huge cost of source refactorying.
- The developers are complaining about the poor functionalities of template language which they are using and tired from the various magic skills for complex rendering logic.
- The developers are discontented with the counterproductivity of traditional MVC architecture and desire a more efficient approach.

1.2 How Asta4D helps us

Asta4D is our solution to combat those issues. Thanks to lift, from where we learn a lot. We designed Asta4D complying with the following points:

- Separate template and rendering logic

  Asta4D affords front-end engineers a friendly environment by separating rendering logic from template files which are pure html files. At the mean time, back-end engineers can use the powerful Java language to implement the rendering logic without being suffering from the "poor and sometimes magic" template languages.

- Testable rendering logic

  All of the rendering logic in Asta4D is testable and developers can simply test them by writing simple junit cases, which could replace over than half of selenium tests

- High security of being immune from cross-site(XSS/CSRF)

  Asta4D is, by nature, immune from cross-site(XSS/CSRF) problems. You do not need to take care of cross-site any more. All the rendered value would be escaped by default and your clients have no chance to put malicious contents to your server.

- View first

  Asta4D also affords higher productivity than traditional MVC architecture by View First mechanism. And it is also easier to change than traditional MVC architecture.

- Isolate side effect with request handler

  Asta4D imports the conception of "side-effect" from functional programming languages and separating the "side-effect" by request handlers, which afford more flexibility on page rendering because the view layer is side-effect free now. Therefore Asta4D allows parallel page rendering in multiple threads as a built-in feature.

- Advanced MVC

  Asta4D affords developers a evolved MVC architecture which is more clarified for the duty of each application layer than the traditional MVC architecture.

1.3 What does the name of "Asta4D" mean

The name of Asta4D is from our company's name: astamuse. We explain the "4D" as following ways:

- For Designer

  Asta4D consider the design friendliness as the most important factor of itself. We hope web designers can fulfil their maximum potential of creativity without squandering their time on the back-end technologies which they could never be adept at.

- For developer

  We hope Asta4D can help developers to achieve their work more easily. Developers would never be afflicted with complex rendering logic because they can use powerful Java language to do whatever they want since the rendering has been split from template files. View first also releases developers from the cumbersome MVC architecture, now they have more time to have a cup of coffee.

- 4 Dimension

  We believe that Asta4D can act as a wormhole that connects the front-end and the back-end. We can move quicker by Asta4D just like we are going through the 4 dimensional space.

Part II. User Guide

2. Flexible template

2.1 Inheritable template

The template of Asta4D is inheritable, child template can override, append or insert content to certain place in parent template. Let's see a sample:

```
<html>
    <head>
        <afd:block id="block1">
            <link href="parent1.css" rel="stylesheet" type="text/css" />
        </afd:block>

        <afd:block id="block2">
            <link href="parent2.css" rel="stylesheet" type="text/css" />
        </afd:block>

        <title>extension sample</title>
    </head>
    <body>
        <afd:block id="content">content</afd:block>
    </body>
</html>
```

Example 2.1 parent.html

Child template can declare inheritance by "afd:extension" and overriding can be declared by "afd:block":

```
<afd:extension parent="parent.html">

    <afd:block append="block1">
        <link href="child1.css" rel="stylesheet" type="text/css" />
    </afd:block>

    <afd:block insert="block2">
        <link href="child2.css" rel="stylesheet" type="text/css" />
    </afd:block>

    <afd:block override="content ">
        <div>hello</div>
        <afd:embed target="/templates/embed.html" ></afd:embed>
    </afd:block>

</afd:extension>
```

Example 2.2 child.html

"afd:block" support 3 types action: insert, append and override.

In the Example 2.2, “child.html”, there is an additional declaration of including by "afd:embed" which embed the target file to the current place of the current file. Let's see the embed.html:

```
    <afd:block append="block1">
        <link href="embed.css" rel="stylesheet" type="text/css" />
    </afd:block>
    <div>good embed</div>
```

Example 2.3 embed.html

The "afd:block" declaration in the embeded target file is available after the content of target file is inserted into the Example 2.2, “child.html”, all "afd:block" tags will be treated as overriding(inserting/appending) to the parent template and the contents out of "afd:blocks" will be inserted at the place where the "afd:embed" is declared.

After all the template files are merged, we will get the following result:

```
<html>
<head>

    <!-- block1 -->
    <link href="parent1.css" rel="stylesheet" type="text/css" />
    <link href="child1.css" rel="stylesheet" type="text/css" />
    <link href="embed.css" rel="stylesheet" type="text/css" />

    <!-- block2 -->
    <link href="child2.css" rel="stylesheet" type="text/css" />
    <link href="parent2.css" rel="stylesheet" type="text/css" />

    <title>extension sample</title>

</head>
<body>

    <!-- content -->
    <div>hello</div>

    <!-- embed -->
    <div>good embed</div>

</body>
</html>
```

Example 2.4 The result of template merging:

2.2 Parametrized embedding

Extra parameters can be passed to the target embeded file by specifying the DOM attribute of "afd:embed" when we are embeding files.

```
<afd:embed target="/xxx/showList.html" limit="30"></afd:embed>
```

Example 2.5 Sample of parametrized embeding:

The "limit" parameter can be accessed by variable injection in the rendering logic of showList.html, by which we can parameterize the embeding. This feature is very useful for creating a page component which encapsulates common html snippets and can be controlled by the passed parameters. Basically, embeding a file in a template file is similar to calling a function with(or without) arguments. Especially, the parameters are not limited to specified static values in the template file, they can be valued dynamicaly at runtime too. Further, the type of parameters is not restricted to the stringizable types such as String or Integer/Long, arbitrary Java type can be specified dynamically at runtime. See the reference of Renering logic to learn more details.

Simply, the idea of Asta4D's template system is akin to the traditional OOP model, the parent and child templates can be regarded as parent/child classes and the block can be viewed as a overridable virtual method. The embed external files can be also treated as funcation calling. As a matter of fact, since arbitrary type of value can be passed to the embed files, there is actually no difference between Asta4D's embeding and common function calling.

3. Renderer: easy to use, secure, testable

3.1 Split rendering logic from template

There is no dynamic code in a template file. An Asta4D template file is always a pure HTML file which can be easily maintained by front-end developers, it is very design friendly and we can reduce the workload for source refactoring by over 90%.

- Declare snippet class in template file

```
<section>
    <article>
        <div afd:render="SimpleSnippet">dummy text</div>
        <afd:snippet render="SimpleSnippet:setProfile">
            <p id="name">name:<span>dummy name</span></p>
            <p id="age">age:<span>0</span></p>
        </afd:snippet>
    </article>
</section>
```

Example 3.1 Sample of snippet declaration:

The "afd:render" attribute in div tag declares a Java class which is in charge of the concrete rendering logic, such Java class is usually called as a snippet class. A snippet class can be declared by a "afd:snippet" tag too, as you have seen in above sample. The rendering logic is secluded to independent Java classes by snippet declaration and it is not necessary to learn any new template language for back-end developers. The back-end guys can achieve all the rendering logic easily by powerful Java language which they have been adept to, no learning costs, no magic codes, succinct Java source only.

- Implement a snippet class

```
public class SimpleSnippet {

    public Renderer render(String name) {
        if (StringUtils.isEmpty(name)) {
            name = "Asta4D";
        }
        return Renderer.create("div", name);
    }

    public Renderer setProfile() {
        Renderer render = Renderer.create();
        render.add("p#name span", "asta4d");
        render.add("p#age span", 20);
        return render;
    }
}
```

Example 3.2 Sample of snippet class:

The Renderer class is provided by framework to declare rendering actions. Renderer uses traditional CSS selector to reference the rendering target and receive the rendering value at the same time, amazing and powerful.

If the above template file and snippet class are executed, we would get the following result:

```
<section>
    <article>
        <span>Hello Asta4D</span>
            <p id="name">name:<span>asta4d/span></p>
            <p id="age">age:<span>20</span></p>
    </article>
</section>
```

Example 3.3 Result of snippet execution:

Asta4D introduces 4 extra tags to the html template file: afd:extension, afd:block, afd:embed, afd:snippet, which will not disturb most html editors, accordingly Asta4D is extremely friendly to front-end engineers. On the other hand, we can see that all the rendering logic is fulfiled by Java code and the back-end engineers can compose their back-end logic and rendering logic very smoothly without magic skills and extra learning costs, which means highly boost of productivity.

3.2 Render data to page

Be careful of "ElementUtil.parseAsSingle" which would cause cross-site issues. Basically the parseAsSingle api is not recommended for common use. See more atSection 3.3, “Immune from cross-site hole”.

- In Asta4D, all the rendering logic is declared by a Renderer class which accepts various types as rendering value.

```
Renderer render = Renderer.create();
render.add("#someIdForInt", 12345);
render.add("#someIdForLong", 12345L);
render.add("#someIdForBool", true);
render.add("#someIdForStr", "a str");
render.add("#someIdForNull", (Object) null);
render.add("#someIdForClear", Clear);

// NOTE: parseAsSingle is not recommended to use
Element newChild = ElementUtil.parseAsSingle("<div></div>");

render.add("#someIdForElementSetter", new ChildReplacer(newChild));
render.add("#someIdForElement", ElementUtil.parseAsSingle("<div>eee</div>"));
render.add("#someIdForRenderer", Renderer.create("#value", "value"));
```

Example 3.4 Sample usage of Renderer

- List of data can be rendered via Renderer too:

```
Renderer render = Renderer.create()();
render.add("#someIdForInt", Arrays.asList(123, 456, 789));
render.add("#someIdForLong", Arrays.asList(123L, 456L, 789L));
render.add("#someIdForBool", Arrays.asList(true, true, false));
render.add("#someIdForStr", Arrays.asList("str1", "str2", "str3"));

// NOTE: parseAsSingle is not recommended to use

Element newChild1 = ElementUtil.parseAsSingle("<div>1</div>");
Element newChild2 = ElementUtil.parseAsSingle("<div>2</div>");

render.add("#someIdForElementSetter", Arrays.asList(new ChildReplacer(newChild1), new
 ChildReplacer(newChild2)));
render.add("#someIdForElement", Arrays.asList(newChild1, newChild2));
render.add("#someIdForRenderer", Arrays.asList(123, 456, 789), new
 RowRenderer<Integer>() {
    @Override
    public Renderer convert(int rowIndex, Integer obj) {
        return Renderer.create("#id", "id-" + obj).add("#otherId", "otherId-" + obj);
    }
});
```

Example 3.5 Sample of list rendering

- DOM attribution can be done too:

```
render.add("#id", "+class", "yyy");
render.add("#id", "-class", "zzz");

render.add("#id", "+class", "xxx");

render.add("#id", "value", "hg");
render.add("#id", "href", null);

render.add("#X", "value", new Date(123456L));
```

Example 3.6 Sample of DOM attribution rendering

3.3 Immune from cross-site hole

The previous samples show us that all the operations to DOM are delegated by Renderer except create DOM from String value by "ElementUtil#parseAsSingle". As a matter of fact, all the delegated operations cause all the rendering values are escaped by force, which means that your system is, by nature, immune from cross-site problems if there is no "ElementUtil#parseAsSingle" calling in your system. You do not need to be wary of illegal characters by from the clients since all the values will be escaped as HTML compulsorily.
On the other hand, according to our practice, it is rarely that you have to call “ElementUtil.parseAsSingle” to generate the rendering content from a raw HTML string, so just take care of your usage of this exceptional api then you can say bye-bye to the cross-site holes.

3.4 Test your rendering logic

Testing web page is always a complex task and we usually verify rendering results by selenuim which is as a common technology. By Asta4D, testable Renderer can replace over than half of selenium tests to simpler junit tests.

According to the previous samples, all the rendering logic is holded by an instance of Renderer class, so simply, we can verify almost rendering logic by testing the returned Renderer instance from the snippet method.

```
RendererTester tester = RendererTester.forRenderer(render);
Assert.assertEquals(tester.get("#someIdForInt"), 12345);
Assert.assertEquals(tester.get("#someIdForLong"), 12345L);
Assert.assertEquals(tester.get("#someIdForBool"), true);
Assert.assertEquals(tester.get("#someIdForStr"), "a str");
Assert.assertEquals(tester.get("#someIdForNull"), null);
Assert.assertEquals(tester.get("#someIdForClear"), Clear);
```

More samples can be found in source[1]
Example 3.7 Sample RendererTester

[1] https://github.com/astamuse/asta4d/blob/develop/asta4d-core/src/test/java/com/astamuse/asta4d/test/unit/RenderTesterTest.java

4. View first URL mapping and variable injection

4.1 View first

In Asta4D, we follow the principle of view first rather than the tradition MVC architecture. The declaration of URL rules does not need to include a controller and one url can be mapped to one template file directly, which is called as View First.

In Asta4D, URL mapping rules are not managed in a configuration file. The Framework provides a sort of DSL by a set of convinient APIs, which means the declaration of Asta4D's URL mapping rule is programmable and it affords much flexibility than the way with a static configuration file.

Users declare their own URL rules via implementing the interface UrlMappingRuleInitializer of Asta4D:

```
public class UrlRules implements UrlMappingRuleInitializer {

    @Override
    public void initUrlMappingRules(UrlMappingRuleHelper rules) {
        //@formatter:off
        rules.add(GET, "/")
             .redirect("/app/index");

        rules.add(GET, "/redirect-to-index")
             .redirect("p:/app/index");

        rules.add("/app/", "/templates/index.html");
        rules.add("/app/index", "/templates/index.html");
        rules.add("/app/{name}/{age}", "/templates/variableinjection.html")#
        ...

        //@formatter:on
    }
}
```

Example 4.1 Sample of declaring url rules:

4.2 The grammer of URL rule and path variable

The grammer of Asta4D's URL rule is almost the same as Spring MVC's URL mapping rule and the parts surrounded by braces are treated as path variables which can be retrieved in following process(in snippet classes or request handlers).

Further, extra path variables can be declared in a url rule by calling the method of "var" as following:

```
rules.add("/app/{name}/{age}", "/templates/variableinjection.html")
     .var("extraVar", 1234);
```

Example 4.2 Sample of declaring extra path variable:

4.3 Variable Injection

Asta4D implements a variable injection mechanism which is very closed to Spring MVC, all of the instance fields and method parameters in the implementation of snippet classes can be injected automatically by framework.

Further, all the snippet classes are singleton in request scope.In other words, there will be only one istance of any snippet class in a single request scope regardless of how many times the snippet class is called. Let us see some samples of variable injection in the snippet class.

```
public class InitSnippet implements InitializableSnippet {

    @ContextData
    private String value; // (1)
    private long id;
    private String resolvedValue;
    private int count = 0;

}
```

@ContextData indicates that the instance field "value" need to be injected after the instance of InitSnippet is created. Since the snippet instance is singeleton in the current request scope, the instance field "value" will be injected and initialized only once in the current request scope. The framework will search a variable named "value" in all the available variable scopes and inject the found value. The name of "value" is decided by the field name by default and can be sepcified by extra declaration(see the following samples).

Example 4.3

```
public class InitSnippet implements InitializableSnippet {

    @ContextData
    private void setId(long id) {  //(3)
        this.id = id;
    }

}
```

Instance field can be injected via setter methods too.

Example 4.4

```
public class InitSnippet implements InitializableSnippet {

    @Override
    public void init() throws SnippetInvokeException { //(2)
        resolvedValue = value + "-resolved";
        count++;
    }

}
```

The init method of InitializableSnippet is being implemented. This is not necessary but if a snippet class implements the interface of InitializableSnippet, the init method will be called once after all the instance fields are injected, by which customized snippet initial logic can be performed.

Example 4.5

```
public class InitSnippet implements InitializableSnippet {

    // (4)
    public Renderer getPathVarName(
    @ContextData(scope = WebApplicationContext.SCOPE_PATHVAR)
    int count) {
    }

}
```

Variable injection on snippet method. In this sample the extra search scope are delcared, which cause framework searches a variable named "count" in the path variable scope. As the same as the injection on instance field, the searching variable name is decided by the parameter name by default.

Example 4.6

```
public class InitSnippet implements InitializableSnippet {

    // (5)
    public Renderer getQueryParamName(
    @ContextData(name = "var", scope = WebApplicationContext.SCOPE_QUERYPARAM)
    String name) {
    }

}
```

In this sample, an extra variable name and search scope are declared, the framework will search a variable named "var" in the request parameters.

Example 4.7

```
public class InitSnippet implements InitializableSnippet {

    // (6)
    public Renderer getDefaultName(String name) {
    }

}
```

There is a difference between instance field injection and method parameter injection. A instance field without @ContextData annotation will not be injected, but for method paramters, all the paramters will be injected compulsorily. In this sample, the framework will search a variable named "name" in all the available scopes as following order:

1. HTML tag attribution(SCOPE_ATTR)

   Trace back to the top tag of the current HTML document recursively. This scope is only available to the snippet method.

2. path variable(SCOPE_PATHVAR)

3. request parameter(SCOPE_QUERYPARAM)

4. flash variable(SCOPE_FLASH)

   The variables passed from one request to another request, usually acrross a redirect.

5. cookie(SCOPE_COOKIE)

6. request hearder(SCOPE_HEADER)

7. request attribute(SCOPE_REQUEST)

8. session(SCOPE_SESSION)

9. global(SCOPE_GLOBAL)

   A global static variable pool

This search order is applied to the instance field injection too except that the first scope of HTML tag attribution is not available.
Example 4.8

5. Side effect and request handler

5.1 The role with responsibility to http request: Request Handler

Even Asta4D complies with the principle of view first, there is still a role similar with the controller in MVC architecture, such role is called request handler. We believe that there should be a role who takes responsiblity of a certain http request and such role can be considered as an alternative to controller of MVC in some situations too.

According to the view first rule, a request handler can be ignored in most cases. You can also assume that there is is a default request handler which navigates all the http requests to the corresponding template files(The real story is more complex but you can try to understand the theory conceptually by this way).

Now the question is when we need a request handler? Before we can answer this question, we have to discuss the conception of "side effect".

5.2 Side effect of system operations

A system always afford users various operations such as query, update, etc. All of these operations will affect the status of system in various ways. In Asta4D, we say that there are two types of action in a system, one is with “side effect”, another one is without “side effect”. “actions with side effect” are ones that will change the system status once they are performed. For instance, for the same URL, a login request (if succeeded) will cause a client’s privilege to be changed and the client could probably get a different page view from what the client get before login, because of which we say a login request is an action with side effect. Another obvious example is a database update operation. Once an update is committed, all the related clients will get a different output from the result before the update, which is also classified as “an action with side effect”.

How about a query? We consider a query as an operation without side effect, it means that a client will always get the same result regardless of how many times the query is executed. Some people may ask how about counting on query times? For counting on query times, we can still split the counting and query to two individual operations, one is with side effect and another one is without side effect.

5.3 Isolate side effect and multi thread rendering

We believe the actions with side effect should be managed seriously and we do that by putting all the actions with side effect to request handlers so that the view layer is purified and this makes the source more clear and maintainable. This is also means with Asta4D we can easily perform multi thread rendering on a single page because they are now all side-effect free, which is a high light function of Asta4D.

In traditional MVC architecture, the responsibility of controller is tangled, a controller does not only handle the duty of altering system status(date update), but also takes the responsibility of preparing data for view layer. We often have to expand the database session(transaction) to the view layer for handling lazy load issue, which makes transaction management more complicated. Additionally, especially for the pages that can be split to relatively individual blocks, hypothetically we can accelerate the page loading by retrieving data in multi threads for each block, but in reality it is impossible due to the source

complexity and development costs. The developers still have to access each data source sequentially to retrieve data for each block even the page is splitable as individual blocks.

For Asta4D, the logic layers of system is clarified by isolating side effects at architecture level. The database session(transaction) does no longer need keep opening cross layers, it is clear that the duty of request handler is altering system status but not preparing data for view layer. At the mean time, at the view layer, acquiring data and rendering data are performed at the same time, at the same place: We access data source and retrieve data in snippet method, and pass the data to Renderer to perform rendering immediately in the same snippet method. Further, the complexity can be reduced drastically because all the accesses to certain data are isolated in certain snippet methods and there is no cross layer invoking and coupling any more.

On the other hand, we can simply perform multi thread rendering by calling each snippet method in different threads, which is transparent to developer. Because all the side effects has been isolated in request handler layer, we do not need to considerate sync or mutex even we are performing multi thread rendering since the view layer is side effect free now.

Let us see how easy we can achieve parallel rendering

```
<div afd:render="ParallelTest$TestRender:snippetInDiv" afd:parallel>
    <div id="test">xx</div>
</div>

<afd:snippet render="ParallelTest$TestRender:snippetReplaceDiv" parallel>
    <div id="test">xx</div>
</afd:snippet>
```

All the snippets declared with parallel(afd:parallel) attribution will not block the http request main thread, the http request main thread will wait for the parallel rendering after all the non-parallel rendering finished, then merge the result and output to the client.

Example 5.1

6. Impement request handler and url mapping

6.1 @RequestHandler

It is not complicated to implement a request handler. @RequestHandler can be used to annotate a handle method in arbitrary Java class which will be treated as a request handler.

```
public class LoginHandler {

    @RequestHandler
    public LoginFailure doLogin(String flag) throws LoginFailure {
        if (StringUtils.isEmpty(flag)) {
            return null;
        }
        if ("error".equals(flag)) {
            throw new LoginFailure();
        }
        if (!Boolean.parseBoolean(flag)) {
            return new LoginFailure();
        }
        return null;
    }
}
```

Take a look at the parameters of the handle method, the handle method of a request handler accepts parameter injection as same as the snippet method. More details can be found at the descriptions of the snippet method.
Example 6.1

6.2 delcare url rule for request handler

Previously we have introduced how to forward a http request to a certain template file by url mapping. We can declare a request handler for q http request by the same way:

```
rules.add("/app/handler")
     .handler(LoginHandler.class) // (1)
     .forward(LoginFailure.class, "/templates/error.html") //(2)
     .redirect(FurtherConfirm.class, "/app/furtherConfirm")//(3)
     .forward("/templates/success.html"); //(4)
```

Simply explain:

1. Forward the request to "/app/handler" to the request handler "LoginHandler".
2. If "LoginHandler" returns a result of "LoginFailure", forward the request to the template file "error.html".
3. If "LoginHandler" returns a result of "FurtherConfirm", redirect the current request to "/app/furtherConfirm" by http code 302(302 is by default).
4. If "LoginHandler" does not return a meaningful result(it usually means success by a null return) , forward the request to the template file "success.html".

Example 6.2

More verbose details:

The handler method is used to add a request handler and accepts arbitrary type as the the parameter: an instance of java.lang.Class or an arbitrary instance. The framework explains received parameters by the implementation of DeclareInstanceResolver configured by WebApplicationConfiguration. Thee default implementation provided by framework follows the following rules to explain the declaration of request handler:

1. If an instance of java.lang.Class is specified, the instance of request handler will be created by invoke "newInstance()" on the specified Class.
2. If a string is specified, the string will be treated as a class name and an instance of java.lang.Class will be created by calling "Class#forName", then the instance of request handler will be created by invoke "newInstance()" on the created Class.
3. The specified parameter will be treated as a request handler directly if it is neither a Class nor a string. By this rule, it is interesting that we can declare an anonymous class as a request handler:

```
rules.add("/app/handler")
     .handler(new Object(){
        @RequestHandler
        public void handle(){
            //
        }
     });
```

Example 6.3

The asta4d-spring package also provides a resolver based on Spring IOC container, the request handler instance will be retrieved by passing the specified parameter to the "Context#getBean" method of spring container.

The forward method adds a transforming rule for the result of request handler.There must be a handler method annotated by @RequestHandler and the returned value of the handle method is viewed as the result of the request handler, thrown exceptions in the handle method are treated as the result too. The framework will attempt to match the result to the expected result specified by forward method then transforms the result to the corresponding template file(The real mechanism of result transforming is more complicated and is explained at ...). When the matching attempt is performed, the equals method will be used at first, if the equals method returns false and the expected result by forward method is an instance of java.lang.Class, "Class#isAssignableFrom" will be utilized, if false again, skip the current forward rule and do the same check on the next forward rule. A forward rule without the expected result specified will be viewed as a default rule, if matched forward rule for a result is missing or the request handler returns a pointless result(void declaration of handle method or returns null), the default rule will be applied.

The redirect method follows the same rule of forward method except it will cause a 302 redirect(302 is default, 301 can be declared) instead of forwarding to a template file.

6.3 Default request handler

You can not only declare request handler to a certain url, but also declare global request handlers which is available to all the urls and prior to the request handlers declared on certain urls. There is a conception of request handler chain for multi handlers, we will explain it in a later chapter.

```
rules.addDefaultRequestHandler(GlobalHandler.class);

rules.addDefaultRequestHandler("authcheck", AuthCheckHandler.class);
```

Example 6.4 Default request handler

At the second line declaration for AuthCheckHandler, an attribute can be specified at the same time, which cause the declared request handler is only available to the rules that declared the same attribute.

```
rules.add("/app/handler").attribute("authcheck")
        ...
```

Example 6.5

In the framework's internal implementation, a static match rule table will be generated after all the url mapping declaration finished. A global request handler that is only available to certain rules will be configured to the the certain url rules only, by which unnecessary performance cost is avoid.

In our practice, we found that it is very inconvenient to declare necessary attribute on every rule. In most situations, we will want to do something like configure a same default request handler to all the url paths under "/xxx", which can be done by delcaring a non attribute default request handler which judges the url pattern by itself, but such way will lose the performance benefit of attribute declaration. Thus, we provide an alternative way for this situation: url rule rewrite.

```
rules.addRuleRewriter(new UrlMappingRuleRewriter() {
    @Override
    public void rewrite(UrlMappingRule rule) {
        if(rule.getSourcePath().startsWith("/privatedata/")){
            rule.getAttributeList().add("authcheck");
        }
    }
});
```

Example 6.6

Please note that, untill now all the introduced configuration for url mapping are achieved by a group of interface called HandyRule which convert all the declaration to the instance of UrlMappingRule which is used by framework internally. But in the url rule rewritter implementation, developers have to cope with the raw UrlMappingRule which is difficult to understand, so complex url rewriting is not appreciated. Basically, the scenario of url rule rewriting is only for add attributes to rules in bulk.

6.4 Global forward/redirect

Previously we mentioned that the result of a request handler will be matched in the forward declaration, if there is no matched forward found, the global forward rule will be checked.

```
rules.addGlobalForward(PageNotFoundException.class, "/pagenotfound.html", 404);
```

The above source declares that if the result of request handler is PageNotFoundException(Exception is treated as the result of the request handler), forward the request to pagenotfound.html by http status code 404.

Example 6.7 Global forward

Global redirect can be declared by the same way.

```
rules.addGlobalRedirect(REDIRECT_TO_SOMEWHERE, "/somewhere");
```

Example 6.8 Global redirect

The global forward rules are applied before the default forward rule(the forward rule without result) on certain url mapping rule.

```
rules.addGlobalForward(PageNotFoundException.class, "/pagenotfound.html", 404);

rules.addGlobalRedirect(REDIRECT_TO_SOMEWHERE, "/somewhere");

rules.add("/app/handler")
     .handler(LoginHandler.class)
     .forward(LoginFailure.class, "/templates/error.html")
     .redirect(FurtherConfirm.class, "/app/furtherConfirm")
     .forward("/templates/success.html");
```

Example 6.9

In the above sample, the result of LoginHandler will be matched by the following order:

1. LoginFailer
2. FurtherConfirm
3. PageNotFoundException
4. REDIRECT_TO_SOMEWHERE
5. (default-> "success.html")

7. Advanced usage of request handler

This chapter is to show how you can make advanced usage of request handler on various situations.

7.1. Advanced MVC architecture

Even we emphasize that Asta4D is a view first framework, it is still compatible with MVC architecture. Further, we would say that Asta4D affords an advanced MVC architecture than traditional MVC frameworks by the view first mechanism.

In the MVC theory, a controller would act as multi roles in a full request process, from wikipedia, it says: *"A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document)."* But those are not all in most situations, we usually have to do more things in a controller such as querying additional assistant data for page rendering.

By traditional MVC architecture, we often have to expand the transaction from the controller layer across to the view layer to combat the lazy load issue, which ugly structure is essentially caused by the tangled controller which holds various unrelated duties.

It is also strange that we have to modify our controller's implementation at every time we change the page appearance at view layer. Such situation could not satisfy us since the layers are not uncoupled really.

We would say that the traditional controller is indeed a tangled magic container for most logics, a controller will unfortunately be coupled to most layers in the system even our initial purpose of MVC is to uncouple our logics. By contrast, Asta4D allows developers to really uncouple all the tangled logics easily. Basically we could split the traditional controller's duty to following parts:

- request handler

  Which takes the responsibilities of all the operations with side-effect.

- result matching in url rule

  Which dispatches the request to different views according to the result from request handler

- snippet class

  Which has the responsibility to render data to the page and also holds the obligation of preparing all the necessary data for page rendering.

By above architecture, we could perfectly uncouple our logics by clarifying the obligation of each layer. *Need sample source ...*

7.2. Normalize page url patterns

Commonly, we would probably have multiple path patterns for the same template file but we would hope our snippet implementation could deal with a unified pattern, which can be simply achieved by a request handler.

```
rules.add("/user/name/{name}").id("user-page")

rules.add("/user/{id}").id("user-page")
     .handler(new Object(){
        @RequestHandler
        public void preparePage(Context context, Integer id, String name){
            User user = null;
            if(id != null){
                user = queryUserById(id);
            }else if (name != null){
                user = queryUserByName(name);
            }
            if(user != null){
                //in the snippet class, a @ContextData annotation can by use to retrieve
 this value by injection
                context.setData("user-page-condition", user);
            }
        }
    }).forward("/user-page.html");
```

Somebody would argue that the above example is apparently a traditional MVC architecture, that is right or wrong. For the most simple situation, if the snippet class will only show the data from entity class User without any extra query, it is true that the above source shows a traditional MVC architecture. But for the most situations, the page rendering would require more additional queries, which would be done at the snippet side. Such structure is so far from the traditional MVC that we would rather to call it as view first.
Example 7.1 Normalize page url

In the above example, we do not cope with the case of user does not exist. The following source shows how we can cope with such situation:

```
rules.addGlobalForward(PageNotFoundException.class, "/template/PageNotFound.html", 404);
rules.addGlobalForward(Throwable.class, "/template/UnknownError.html", 500);

rules.add("/user/name/{name}").id("user-page")

rules.add("/user/{id}").id("user-page")
     .handler(new Object(){
        @RequestHandler
        public void preparePage(Context context, Integer id, String name){
            User user = null;
            if(id != null){
                user = queryUserById(id);
            }else if (name != null){
                user = queryUserByName(name);
            }
            if(user == null){
                throw new PageNotFoundException();
            }else
                //in the snippet class, a @ContextData annotation can by use to retrieve
 this value by injection
                context.setData("user-page-condition", user);
            }
        }
    }).forward("/user-page.html");
```

Example 7.2 Normalize page url and PageNotFoundException

See more details at the section called “Normalize page condition by request handler”

7.3. Restful and ajax

For the request which does not require response of html files, such as restful or ajax request, request handler can be used to cope with such situation.

There is a rest() method which does nothing on the current rule by default but can be used as a hint to suggest that the current rule will response customized contents as a restful api. A request handler can return a ContentProvider to supply customized response content. Details of ContenProvider can be found at later chapters, here we only show examples of how to response customized contents.

```
public class UpdateHandler {

    @RequestHandler
    public ContentProvider doUpdate(String id, String content) {
        if (idExists(id)) {
            try {
                updateContent(id, content);
                return new HeaderInfoProvider();
            } catch (Exception ex) {
                HeaderInfoProvider header = new HeaderInfoProvider(500);
                header.addHeader("exception", ex.getMessage());
                return header;
            }
        } else {
            return new HeaderInfoProvider(404);
        }
    }
}
```

Example 7.3 Return header only response

```
public class GetHandler {

    @RequestHandler
    public ContentProvider get(String id) {
        Object data = getContent(id);
        if (data == null) {
            return new HeaderInfoProvider(404);
        } else {
            HeaderInfoProvider header = new HeaderInfoProvider();// default to 200
            header.addHeader("Content-Type", "application/json");

            String json = toJson(data);
            BinaryDataProvider binaryData = new BinaryDataProvider(json.getBytes());

            return new SerialProvider(header, binaryData);
        }
    }
}
```

Example 7.4 Return customized binary data

For json data, there is a more convenience way to response a json string to client. A json() method can be used on rule declaration to ask the framework to convert the returned result from request handler to a json string automatically. the above

```
// in rule declaration
rules.add("/get").handler(GetHandler.class).json();


//the handler implementation
public class GetHandler {

    @RequestHandler
    public Object get(String id) {
        return getContent(id);
    }
}
```

Example 7.5 Declare json at rule

By default, a rest rule will do nothing on the result of handler and a ContentProvider instance is expected, by contrast, there is a built-in json transformer for rules which are declared as json. However, developers can still register a customized result transformer for rest rules or even register a customized json transformer to override the default built-in implementation. See details at the section called "UrlMappingRuleInitializer"

Further, since a request handler can access HttpServletResponse instance directly, developer can do more complex customization on HttpServletResponse instance directly.

```
public class XXXHandler {

    @RequestHandler
    public void handle(HttpServletResponse response) throws IOException {
        response.setStatus(200);
        response.addHeader("xxx", "");
        response.getOutputStream().write("OK".getBytes());
    }
}
```

Example 7.6 access HttpServletResponse directly

There are two built-in customized request handlers for handler static resouce files and mapping generic path template files. See the next section.

7.4. Handle static resouce files

The most simple way to handling static resource files is mapping their path in web.xml then they will be serviced by the servlet container. But Asta4D still affords a way to handle them at framework level.

```
rules.add("/js/**/*").handler(new StaticResourceHandler());

rules.add("/img/**/*").handler(new StaticResourceHandler("/resource/img"));

rules.add("/favicon.ico").handler(new StaticResourceHandler(/img/favicon.ico));
```

Example 7.7 handler static files

The wild-card "/**/*" in source path is necessary if you want to mapping all the urls with same path prefix to a certain base folder. The base path parameter in constructor of StaticResourceHandler is not necessary. if the base path is not specified, StaticResourceHandler will treat the part before the wild-card "/**/*" in source path as the base path.

StaticResourceHandler can be customized by path var configuration or overriding some protected methods by extending. See details in javadoc of StaticResourceHandler.

```
// by path var

rules.add("/js/**/*")
    .var(StaticResourceHandler.VAR_CONTENT_TYPE, "text/javascript")
    .handler(StaticResourceHandler.class);

// by overring
rules.add("/js/**/*").handler(new StaticResourceHandler(){
    @Override
    protected long decideCacheTime(String path) {
        return 24 * 60 * 60 * 1000; //force cache 24 hours
    }

});
```

Note that it is not necessary to specify content type for most situations because StaticResourceHandler will guess the content type by file name extension automatically.

Example 7.8 customize StaticResourceHandler

7.5. Handle generic path template files

Basically Asta4D asks developers to declare the matching relationship between url and template file one by one, but it still allows to declare a generic path matching for all files in same folder, which can be achieved by GenericPathTemplateHandler.

```
rules.add("/pages/**/*").handler(GenericPathTemplateHandler.class);
```

As same as the StaticResourceHandler, the wild-card "/**/*" is necessary and the base path parameter of constructor can be ignored.

Example 7.9 handler template files in bulk

8. Built in form flow

(Being written. See the online sample: Form Flow Sample[1])

Asta4D affords built-in form flow to accelerate development of traditional form process. Asta4D's form flow mechanism supports various flow style and supplies several classical flow style as built-in. For most cases, the developers only need to implement the init and update method to complete the entire form related logic.

8.1. Startup

Table of Contents

Form and form field

In a form process, the basic conception is a POJO which represents the whole form data. In Asta4D, we use @Form annotation to annotate a POJO as a form object which can be handled by the the form flow.

Also, there must be fields to represent the concrete data of a form, which can also be annotated by a set of form field annotations. There are several built-in form field annotations for common cases and you can implement your own form field annotation too(See Chapter 14, *details of form flow* for more details about form field annotation).

[1] http://asta4dsample-xzer.rhcloud.com/form

```
@Form
public class PersonForm extends Person{

    @Hidden
    public Integer getId() {
        return super.getId();
    }

    @Input
    public String getName() {
        return super.getName();
    }

    @Select(name = "bloodtype")
    public BloodType getBloodType() {
        return super.getBloodType();
    }

}
```

The @Hidden represents a hidden input of html, @Input represents a traditional common input and @Select is matched to the pull-down element in html. We also recommend to implement the form POJO by extending from the existing entity POJO, see details at Section 9.2, “Common usage of form flow”. Example 8.1 annotations on form POJO and form fields

Form handler

Table of Contents

Brief of BasicFormFlowHandlerTrait and FormProcessData

After we defined our form POJO, we need to declare a request handler to handle the form request. There is an BasicFormFlowHandlerTrait which affords most necessary common operations of form process.

The BasicFormFlowHandlerTrait is implemented as an interface with various default methods which afford a template that allows developer to override any method for customization.

To define a form flow, we need to plan a flow graph which describes how the flow flows.

(before first) <--> step 1 <--> step2 <--> step3 <--> ...

Assume we have a flow as above, note that there can be cycles or branches, which means you can go any step from any other step in the flow graph, what you need to do is to define how the step should be transfered. For example, the following flow graph is possible:

(before first) --> step 1 <--> step2 <--> step3 --> (finish)

(before first) --> step 1 <--> step4 <--> step5 --> (finish)

step 1 <-- step3

step 1 <-- step5

To describe the step transfer, there is an interface called FormProcessData which defined the basic step information:

```
public interface FormProcessData {

    public abstract String getStepExit();

    public abstract String getStepBack();

    public abstract String getStepCurrent();

    public abstract String getStepFailed();

    public abstract String getStepSuccess();

    public abstract String getFlowTraceId();

}
```

Example 8.2 FormProcessData

The FormProcessData requires the developer to tell how to handle a form submit, the name of current step, the target step for success and the target for failing, also want to know where to go if user want to go back to the previous step or exit. The getFlowTraceId will return a id which represents all the state of current flow(can be considered as a "session" id for the current in-progress form flow).

The default implementation of FormProcessData is SimpleFormProcessData which retrieves the step information from the submitted http query parameters which can be put into the HTML template files as a part of the submitting form. However developers can always decide how to retrieve the step information by implement their own FormProcessData.

```
@ContextDataSet
public class SimpleFormProcessData implements FormProcessData {

    @QueryParam(name = "step-exit")
    private String stepExit;

    @QueryParam(name = "step-back")
    private String stepBack;

    @QueryParam(name = "step-current")
    private String stepCurrent;

    @QueryParam(name = "step-failed")
    private String stepFailed;

    @QueryParam(name = "step-success")
    private String stepSuccess;

    @QueryParam(name = FormFlowConstants.FORM_FLOW_TRACE_ID_QUERY_PARAM)
    private String flowTraceId;
}
```

Example 8.3 SimpleFormProcessData

If we use the default SimpleFormProcessData, we will usually include the following HTML in our template file side:

```
<input type="hidden" name="step-current" value="confirm">
<input type="hidden" name="step-failed" value="input">
<button type="submit" name="step-success" class="btn btn-sm btn-
default" value="complete">send</button>
<button type="submit" name="step-back" class="btn btn-sm btn-default" value="input">back</
button>
<button type="submit" name="step-exit" class="btn btn-sm btn-
default" value="exit">cancel</button>
```

Example 8.4 HTML for SimpleFormProcessData

The details of how the BasicFormFlowHandlerTrait process the submitted form data and transfer the step can be found at the JavaDoc of BasicFormFlowHandlerTrait. Here we will introduce the points that what the developers have to do to decide the rule of a form flow. There are 3 methods should be overridden for a certain flow rule:

- String createTemplateFilePathForStep(String step)

  decide how to convert a step to the corresponding target template file path

- boolean skipStoreTraceData(String currentStep, String renderTargetStep, FormFlowTraceData traceData)

  decide whether the flow trace data should be stored

- boolean passDataToSnippetByFlash(String currentStep, String renderTargetStep, FormFlowTraceData traceData)

  decide how to pass the form data for rendering to snippet, by flash scope or not.

For most common situations, all the above things can be decided as general rules in the user project, so that a common parent class can be utilized to perform the common assumption. There are two built-in flows representing the classical situations: OneStepFormHandlerTrait and ClassicalMultiStepFormFlowHandlerTrait. Those two built-in interfaces will be introduced in the next section and can be considered as reference implementation of how to design and decide a form flow. User project is always recommended to extend from those two built-in flows rather than the basic mechanism trait BasicFormFlowHandlerTrait.

built-in flows

There are two built-in handler traits to handle the most common situations of the form flow, which are OneStepFormHandlerTrait and ClassicalMultiStepFormFlowHandlerTrait.

The OneStepFormHandlerTrait represents a most simple form process: there is a single input page, after submit, the submitted data will be handled and then return to a before-input page which is usually a list page of items. To use OneStepFormHandlerTrait, you need to put the following HTML in your form template files if you are using the default SimpleFormProcessData:

```
<input type="hidden" name="step-current" value="input">
<input type="hidden" name="step-failed" value="input">
<button type="submit" name="step-success" class="btn btn-sm btn-
default" value="complete">save</button>
<button type="submit" name="step-exit" class="btn btn-sm btn-
default" value="exit">cancel</button>
```

By OneStepFormHandlerTrait's default implementation, the "step-current" and "step-failed" must be "input", the "step-success" and the "step-exit" can be any non empty value (usually "complete" and "exit" is good enough).

Example 8.5 SimpleFormProcessData HTML for OneStepFormHandlerTrait

The ClassicalMultiStepFormFlowHandlerTrait represents a little bit complicated situations: there are multiple steps in the flow. ClassicalMultiStepFormFlowHandlerTrait assumes that there is at least one input page and one confirm page with a possible complete page. For the single input page case, ClassicalMultiStepFormFlowHandlerTrait can be used directly, but if there are multiple splitted input pages, the developer need to do more customization.

For a classical 3-step form flow(input, confirm, complete), the following HTML need to be put into the form template files:

```
<input type="hidden" name="step-current" value="input">
<input type="hidden" name="step-failed" value="input">
<button type="submit" name="step-success" class="btn btn-sm btn-
default" value="confirm">confirm</button>
<button type="submit" name="step-exit" class="btn btn-sm btn-
default" value="exit">cancel</button>
```

At input page, the "step-current" and "step-failed" must be "input", the "step-success" must be "confirm", the "step-exit" can be any non empty value (usually "exit" is good enough).

Example 8.6 SimpleFormProcessData HTML for ClassicalMultiStepFormFlowHandlerTrait - buttons of input page

```
<input type="hidden" name="step-current" value="confirm">
<input type="hidden" name="step-failed" value="input">
<button type="submit" name="step-success" class="btn btn-sm btn-
default" value="complete">send</button>
<button type="submit" name="step-back" class="btn btn-sm btn-default" value="input">back</
button>
<button type="submit" name="step-exit" class="btn btn-sm btn-
default" value="exit">cancel</button>
```

At confirm page, the "step-current" must be "confirm","the step-failed" must be "input", the "step-success" must be "complete", the "step-back" must be "input", the "step-exit" can be any non empty value (usually "exit" is good enough).

Example 8.7 SimpleFormProcessData HTML for ClassicalMultiStepFormFlowHandlerTrait - buttons of confirm page

```
<button type="submit" name="step-exit" value="exit">return</button>
```

At complete page, a non empty value of "step-exit" is enough.

Example 8.8 SimpleFormProcessData HTML for ClassicalMultiStepFormFlowHandlerTrait - buttons of complete page

Note that the value of steps' name are not fixed and you can always define your own names by overriding the corresponding methods about step names at ClassicalMultiStepFormFlowHandlerTrait.

Implement a real form flow handler

As introduced in previous sections, we should always extend our own handler from the built-in classical traits, before which we also have to decide what our flow should be: one step or multiple steps?

Further, In user project, a common parent class is always recommended. A project limited common parent class can be used to decide the special rules of the user project and the following two method is strongly recommended to be overridden to return a configured and validator.

- getTypeUnMatchValidator()

- getValueValidator()

```
/**
 * A common parent handler to configure the common actions of form flow process in
 application. <br>
 * For quick start, an empty class body would be good enough. You only need to do the
 customization when you really need to do it!!!
 *
 */
public abstract class Asta4DSamplePrjCommonFormHandler<T> implements
 ClassicalMultiStepFormFlowHandlerTrait<T> {

    // we use a field to store a pre generated instance rather than create it at every
 time
    private SamplePrjTypeUnMatchValidator typeValidator = new
 SamplePrjTypeUnMatchValidator(false);

    // as the same as type validator, we cache the value validator instance here
    private SamplePrjValueValidator valueValidator = new SamplePrjValueValidator(false);

    @Override
    public FormValidator getTypeUnMatchValidator() {
        return typeValidator;
    }

    @Override
    public FormValidator getValueValidator() {
        return valueValidator;
    }

}
```

At confirm page, the "step-current" must be "confirm","the step-failed" must be "input", the "step-success" must be "complete", the "step-back" must be "input", the "step-exit" can be any non empty value (usually "exit" is good enough).
Example 8.9 A common parent handler in project:

More details about validator can be found at later section.

In both OneStepFormHandlerTrait and ClassicalMultiStepFormFlowHandlerTrait, there are 3 methods which are required to be implemented by developers.

- Class<T> getFormCls()

  Which specify the form type of current

- T createInitForm()

  To create form instance from the current request context. By default, this method will build a form instance from context by the given type returned by getFormCls(). A handler which need to retrieve the initial form data from storage should override this method to afford a initial form instance. For common cases, a handler for adding is not necessary to override this method but a handler for updating should override this method to query database and build the corresponding form instance.

```
    @Override
    protected PersonFormForMultiStep createInitForm() throws Exception {
        PersonFormForMultiStep form = super.createInitForm();
        if (form.getId() == null) {// add
            return form;
        } else {// update
            // retrieve the form form db again
            return
PersonFormForMultiStep.buildFromPerson(PersonDbManager.instance().find(form.getId()));
        }
    }
```

Example 8.10 query updating target data from db in createInitForm()

- void updateForm(T form)

  Which is supposed to perform the final update logic of current form flow.

```
    @Override
    protected void updateForm(PersonFormForMultiStep form) {
        DefaultMessageRenderingHelper msgHelper =
DefaultMessageRenderingHelper.getConfiguredInstance();
        if (form.getId() == null) {// add
            PersonDbManager.instance().add(Person.createByForm(form));
            // output the success message to the global message bar
            msgHelper.info("data inserted");
        } else {// update
            Person p = Person.createByForm(form);
            PersonDbManager.instance().update(p);
            // output the success message to specified DOM rather than the global
message bar
            msgHelper.info(".x-success-msg", "update succeed");
        }
    }
```

Note that there is a built-in message rendering mechanism to help developer supply a responsive interaction more easily. The details will be introduced later, simply remember that you can output message by info/warn/error levels.

Example 8.11 update form data

HTML template of form

As the common pages of Asta4D, the HTML template of a form is as pure HTML too.

```
<form method="post" afd:render="form.SingleInputFormSnippet">

  <input name="name" type="text"/>

  <input name="age" type="text"/>

  <input id="sex" name="sex" type="radio"/><label for="sex">M</label>

  <select id="bloodtype" name="bloodtype">
    <option value="A">A</option>
    <option value="R" afd:clear>R</option>
  </select>

  <input id="language" name="language" type="checkbox"/><label for="language">M</label>

  <textarea name="memo"></textarea>

  <input type="hidden" name="id">

  <afd:embed target="/templates/form/singleInput/btns.html" />
</form>
```

Example 8.12 Form template

As you see in the above example, there is no special declaration in the template, let us see how to declare a form POJO to handle the various form field types:

```
//@Form to tell the framework this class can be initialized from context
//extend from the entity POJO to annotate form field definitions on getters.
@Form
public class PersonForm {
    @Hidden
    private Integer id;

    @Input
    private String name;

    @Input
    private Integer age;

    @Select(name = "bloodtype")
    private BloodType bloodType;

    // the field name would be displayed as "gender" rather than the original field name
 "sex" in validation messages
    @Radio(nameLabel = "gender")
    private SEX sex;

    @Checkbox
    private Language[] language;

    @Textarea
    private String memo;

}
```

Example 8.13 form template

The details of annotations will be introduced later, just remember that there is always an annotation which can represent a certain type of form field.

Another point here is that you can always declare all the fields by your business type rather than string type, the framework will handle the value conversion correctly.

Form snippet

To render the form values to template, we have to declare a snippet which extends from the AbstractFormFlowSnippet. As the same as the built-in two classical handler traits, there are two corresponding snippet traits: OneStepFormSnippetTrait and ClassicalMultiStepFormFlowSnippetTrait.

Also as the same as handlers, a project common parent snippet is recommended to perform common customization.

```
public class SingleInputFormSnippet extends Asta4DSamplePrjCommonFormSnippet {

    /**
     * override this method to supply the option data for select, radio and checkbox.
     */
    @Override
    protected List<FormFieldPrepareRenderer> retrieveFieldPrepareRenderers(String
 renderTargetStep, Object form) {
        List<FormFieldPrepareRenderer> list = new LinkedList<>();

        list.add(new
 SelectPrepareRenderer(PersonForm.class, "bloodtype").setOptionData(BloodType.asOptionValueMap));

        list.add(new
 RadioPrepareRenderer(PersonForm.class, "sex").setOptionData(SEX.asOptionValueMap));

        list.add(new
 CheckboxPrepareRenderer(PersonForm.class, "language").setOptionData(Language.asOptionValueMap));

        return list;
    }
}
```

Example 8.14 Form snippet implementation

Developers are usually asked to override the retrieveFieldPrepareRenderers method to supply extra data for field rendering, commonly the list of option data is required.

SelectPrepareRenderer can be used to afford option list for select element, RadioPrepareRenderer and CheckboxPrepareRenderer can be used for radio and checbox input element.

Finally, do not forget to put your snippet declaration in your template file.

```
<form method="post" afd:render="form.SingleInputFormSnippet">
</form>
```

Example 8.15 Declare snippet in form template

Basically, until now we have gotten a workable form flow implementation. There is only one thing left that is validation which will be described in the next section.

Cascade form POJO and array field

8.2. Advanced

Table of Contents

Validation

Asta4D allows any validation mechanism to be integrated and supports Bean Validation 1.1(JSR349 and JSR303) by default. We will explain how to use the built-in Bean Validation mechanism in this section. The later section will introduce how to customize the validation.

To use Bean Validation, just simply add validation annotations to your form POJO as following:

```
@NotBlank
@Size(max = 6)
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

@Max(45)
@NotNull
public Integer getAge() {
    return age;
}
```

Example 8.16 Declare validation annotations

More details of Bean Validation can be found at Bean Validation[2]. We are also using Hibernate Validator as the implementation(the only existing one in the earth currently). See details at Hibernate Validator[3].

The validation will be invoked before the calling of method updateForm, if there is any validation error, the page will be forwarded to the step specified by "step-failed" which is usually the input page, and the validation error message will be rendered to page too. More details about validation message rendering will be introduced in later sections.

As recommended before, a project common parent class is recommended to afford a configured validator or any other customized validator implementation.

Message rendering

Customize form field annotation

A sample of defining a new flow rule

(Include about how to implement multiple input steps)

[2] http://beanvalidation.org/
[3] http://hibernate.org/validator/

9. Best practice

In this chapter, we will introduce the best practice of Asta4D from our own experiences. There are some conventions and assumption in this chapter, absolutely there are always special cases which are out of our assumption, however you can break any rules to handle your special cases, we only discuss the common cases in this chapter.

9.1. Division of responsibilities between request handler and snippet

Table of Contents

Do update by request handler

MUST do all the operations with side effect at request handler side, that is so we called isolating side effect.

NEVER do any operations with side effect at snippet side.

Normalize page condition by request handler

Table of Contents

It is a little bit difficult to explain this rule. For a page, we assume that there are always only 2 types of pages. Let us see how we should divide our logic between the request handler and snippet.

The URL of current page represents the unique resource id of an entity(resource)

- request handler
  - Retrieve the entity(resource) from back-end storage(usually the db) by given id which is specified by the URL
  - Decide how/what to response
    - (for example)If the target entity does not exist, return a 404 to the client
    - (for example)If the target entity has been changed/migrated to another entity, return a 301 to the client
    - (usually)Save the entity to Context and then forward to the target html template

- snippet
  - Retrieve the saved entity(resource) from Context(by @ContextData)
  - Perform other necessary query for rendering

The URL of current page represents a search condition which can be used to retrieve a list of entities(resources)

- request handler
  - For simple cases, a request handler is not necessary

    Because the snippet can retrieve the condition by @ContextData directly. However we recommend to declare a POJO to represent your search condition for a little bit complicated situations.

```
@ContextDataSet
public class Condition{

  @QueryParam
  private String queryword;

  public String getQueryword(){
    return queryword;
  }
}
```

```
public class MySnippet{

  @ContextData
  private Condition condition;

}
```

Example 9.1 Snippet handling a search condition POJO

  - Normalize the search condition POJO

    In some complicated situations, you may need to retrieve the POJO in request handler then do some necessary normalization. After the POJO was normalized, you should save it into Context to pass to snippet.

```
@ContextDataSet(singletonInContext=true)
public class MyHandler{

  public static final String SEARCH_CONDITION = "SEARCH_CONDITION#MyHandler";

  @RequestHandler
  public void handle(Condition condition){
    //do some normalization here
    ...
    //This is not necessary
    //Context.getCurrentThreadContext().setData(SEARCH_CONDITION, condition);
  }
}
```

```
public class MySnippet{

  @ContextData
  private Condition condition;

}
```

There is a trick that if we define a ContextDataSet's singletonInContext to true, there will be only on single instance in the context, thus we do not need to declare the name of ContextData even we stored the POJO by name at the handler side. Even that you do not need to save the condition POJO explicitly since it is the single instance in the Context.

Example 9.2 Normalize the search condition POJO

- snippet
  - Retrieve the saved search condition POJO from Context(by @ContextData) or simply declare the search condition as its own fields(by @ContextData)
  - Perform the query by given search condition to retrieve the list of entities(resources)
  - Perform other necessary query for rendering

9.2. Common usage of form flow

To be written.

9.3. The best way of implementing a snippet

Table of Contents

Perform queries as simple as possible

We encourage developers to retrieve the data at the place where the data is required rather than retrieve a complicated data structure which holds all the necessary data for the whole page rendering in bulk. We make a convention that there are only 4 type of return value in our service layer(where the queries are performed exactly):

- a single entity which is usually the POJO for ORMAP
- an list of entities' ids(not the entities themselves)
- a count of some aggregation queries
- a map as <id, count> of some aggregation queries

In the implementation of render methods, we always retrieve the entity instance one by one even we are rendering a list of entity (exactly we only have a list of entities' ids). For the concern about performance, see the later description of cache. However, there must be some situations that we cannot be satisfied with the performance by retrieving data in a too small granularity. We handle these cases as following steps:

- Ask yourself, is it true that the granularity of data retrieving is the bottleneck of current performance issue.
- Ask your team members to confirm the bottleneck again.
- Implement a query method which returns a map as <id, entity> in bulk, then retrieve the entity from the map in the list rendering.
- If there is still a performance issue, ask yourself and ask your team members to confirm the bottleneck again.
- Build a query to retrieve all necessary data in bulk then return a complicated data structure from the query method just like what we are used to do in the traditional MVC mode.

In fact, the complicated data structures are rare in our system and most of our cases can be satisfied with the basic ORMAP entities.

Make use of ParallelRowRender

The ParallelRowRender splits the rendering action to threads with the size of rendered list. If there are some heavy operations in the list rendering, ParallelRowRender is a good option to reduce the time of rendering.

Cache your data

Since we are encouraging developers to retrieve data by a small granularity, the cache is very important to the system. Especially we often retrieve entities one by one in the list rendering, the cache of entities is the most important thing for performance. As a matter of fact, we can use the cache more effective because of the small data granularity.

Avoid duplicated query

An frequently asked question is how to avoid duplicated query in the snippet class. That is actually an problem since we encourage the developers perform queries at the place where the data is required. We have the following options to combat this issue:

- Global query lock via AOP

  This is the first option that we recommend for query heavy systems and exactly what we did in our system before we moved to saas architecture. We split all the real queries to a service layer then perform a global lock for all the equal calling parameters on the same method. We use spring as our IOC container and simply add the lock logic by the spring's AOP mechanism.

- ContextBindData

  For the lighter systems, ContextBindData would be a good choice for most situations. The get method of ContextBindData will be executed only once in the current Context. See the detail of ContextBindData at the section called “ContextDataFinder”.

  ContextBindData can also be used as a graceful simple wrapping mechanism for data retrieving at snippet layer. Developers can declare the common queries of a snippet class as ContextBindData fields of it.

- InitializableSnippet

  A snippt class can also implement an interface called InitializableSnippet. The init() method of a snippet which implements the InitializableSnippet will be called once after the snippet instance is created. Developer can do some prepare work for rendering by this mechanism such as performing common queries. However, performing query in init() method is not recommended but everything is by your choice.
  In multiple thread rendering, the snippet classes may still be instantiated multiple times. There is no warranty about it.

Prepare snippet instance(set default value) by InitializableSnippet

As we mentioned above, InitializableSnippet can be used to do some initialization work after the snippet instance is created. We do not recommend to perform common queries in init() method of InitializableSnippet, but the InitializableSnippet interface is still a good option for preparing a snippet instance.

The classical usage of InitializableSnippet is that we can set default value of declared @ContextData fields if some of them are set to null.

"x-" convention selector

Though we can use various selectors to render data into the template file, we recommend to use css class name selector in common cases, which affords the best compatibility to the frond-end refactoring. We also made a convention which is called "x-" convention. we will always add a css class started with "x-" to the data rendering target, which tells the front-end guys try their best to keep the compatibility against the "x-" marked elements when they are refactoring the html sources.

Although we always use "x-" convention to render our data, there are still some acceptable exceptions in our practice:

- use "a" directly when rendering the link to the "href" attribute
- use "img" directly when rendering the link to the "src" attribute
- use "li" directly when rendering a list
  This one is controversial, some of our members argue that we should use "x-" instead of direct "li" selector.

Remove contents rather than add contents for conditional rendering

Sometimes we need to do conditional rendering. For instance, we assume that there are two tabs which can be switched by different query parameter, a possible way is to create the target tab contents dynamically at snippet side as following:

```
public Renderer render(int id){
    Element elem = null;
    if(id == 0){
        elem = ElementUtil.ParseAsSingle("<div>" + id + "</div>");
    }else{
        elem = ElementUtil.ParseAsSingle("<div class="great">" + id + "</div>");
    }
    return Renderer.create(".x-target", elem);
}
```

Example 9.3 create contents dynamically

Apparently there is bad smell in above example, we generate DOM element at snippet side, which makes front-end refactoring difficult. The recommended way is to write all the possible contents in template file and remove the unnecessary ones in snippet.

```
public Renderer render(int id){
    Renderer renderer = Renderer.create();
    //remove the clear mark
    renderer.add(".x-target-" + id, "-class", "x-clear");
    //then clear the left ones which is not necessary
    renderer.add(".x-clear", Clear);
    return renderer;
}
```

```
<div class="x-target-0 x-clear">0<div>
<div class="x-target-1 x-clear">1<div>
```

Example 9.4 remove redundant contents

Part III. Reference

10. Details of Template

10.1. Supported tags

Table of Contents

afd:extension

A "afd:extension" tag declares the relationship of inheritance between child template and parent template.

```
<afd:extension parent="parent.html">

...

</afd:extension>
```

Example 10.1 child.html

Attributes:

parent
: Identifies the parent template file of the current template.

afd:block

A "afd:block" tag declares a referenceable block in the template file. Child template file can reference the blocks in parent template by the same tag "afd:block" with extra action declaration: append, insert, override.

```
<html>
    <head>
        <afd:block id="block1">
            <link href="parent1.css" rel="stylesheet" type="text/css" />
        </afd:block>

        <afd:block id="block2">
            <link href="parent2.css" rel="stylesheet" type="text/css" />
        </afd:block>

        <title>extension sample</title>
    </head>
    <body>
        <afd:block id="content">content</afd:block>
    </body>
</html>
```

Example 10.2 parent.html

```
<afd:extension parent="parent.html">

    <afd:block append="block1">
        <link href="child1.css" rel="stylesheet" type="text/css" />
    </afd:block>

    <afd:block insert="block2">
        <link href="child2.css" rel="stylesheet" type="text/css" />
    </afd:block>

    <afd:block override="content">
        <div>hello</div>
    </afd:block>

</afd:extension>
```

Example 10.3 child.html

Attributes:

id
: The referenceable id of a block.

append
: Append all the children of current tag to the tail of the parent's block which id equals the declared value of the append attribute.

insert
: Insert all the children of current tag to the head of the parent's block which id equals the declared value of the insert attribute.

override
: Override the parent's block which id equals the declared value of the override attribute by all the children of current tag.

The result of merging the above two files by Asta4D's template engine can be found at Chapter 2, Flexible template.

afd:embed

A "afd:embed" tag declares an action of extracting and inserting an external file's content to the site where the "afd:embed" tag is declared.

```
<afd:extension parent="parent.html">

    <afd:block override="content ">
        <div>hello</div>
        <afd:embed target="/templates/embed.html" ></afd:embed>
    </afd:block>

</afd:extension>
```

Example 10.4 child.html

Attributes:

target
: The path of which file should be included at the declaration site.

In the embed target file, afd:block can also be used to append/insert/override the blocks in source template. Samples and the merging result can be found at Chapter 2, Flexible template.

afd:snippet

A "afd:snippet" tag declares an Java class which assumes the duty of rendering data to the html snippet contained by the declaring tag.

```
<afd:snippet render="SimpleSnippet:setProfile">
    <p id="name">name:<span>dummy name</span></p>
    <p id="age">age:<span>0</span></p>
</afd:snippet>
```

Example 10.5

Attributes:

render
: The snippet class name and render method name by format:<class name>:<method name>, the class name and the method name are split by colon. If the method name is omitted, the default method name "render" will be used.

parallel
: A snippet tag with "parallel" attribute will be executed in a separated thread from the main thread of current http request. It is not necessary to assign a value to this attribute.

clear
: A snippet tag with "clear" attribute will not be executed and will be removed from the final output page. This is useful for disabling certain contents temporarily in certain situations such as debug. It is not necessary to assign a value to this attribute.

There is an alternative way to declare a snippet by "afd:render" attribute.

```
<div afd:render="SimpleSnippet:setProfile">
    <p id="name">name:<span>dummy name</span></p>
    <p id="age">age:<span>0</span></p>
</div>
```

Example 10.6

By the same way, "afd:paralell" and "afd:clear" can be used on arbitrary html tags which will be treated as the same as a "afd:snippet" tag. Note that "afd:paralell" will not be available except the "afd:render" is declared too, but the "afd:clear" will be always available even there is no declaration of "afd:render" on the html tag.

afd:group

A "afd:group" tag declares a referenceable DOM element for back-end Java snippet class. It is useful to reference certain text without wrapping html tag or combine a group of coordinate html tags as a single node.

```
<p>We found <afd:group id="item-count">3</afd:group> matched items in our database.</p>

<afd:group id="list-item">
  <div id="title">the title</div>
  <div id="content">the content</div>
</afd:group>
```

Example 10.7

Attributes:

None.

afd:comment

A "afd:comment" tag declares a comment block in the template files and it will be removed after rendering.

```
<afd:comment>This is comment</afd:comment>

<afd:comment>
  <div>This is comment too.</div>
</afd:comment>
```

Example 10.8

Attributes:

None.

afd:msg

A "afd:msg" tag declares a message managed in external files. This tag is mainly designed for i18n support, but it also can be used as a simple string replacer. Please see Chapter 15, *i18n* for more details.

```
<afd:msg key="search.result.count"></afd:message>
```

Example 10.9

Attributes:

key
: The message key which should can be found in the configured message file.

10.2. Additional

- HTML5 convention

  Asta4D's template engine does only support html5, which relies on the underline library jsoup(verion 1.6.3)[1]. jsoup implements the WHATWG HTML5[2] specification and requires compliance from the target file. jsoup can fix some cases of breaking specification but you would still get unexpected result for certain cases. Basically, you should take care of the following things:

  - All tags and attributes are converted to lower case.
  - Self closed tags such as or <afd:embed target="embed.html" /> is not legal.
  - There are some tags that the parser "ensures".For example a <tbody> tag must be the first tag inside <table>, which breas the following example:

[1] http://jsoup.org
[2] http://whatwg.org/html

```
<table>
<afd:snippet render="ListSnippet">
<tr><td></td></tr>
</afd:snippet>
</table>
```

Example 10.10

But it can be replaced by the following:

```
<table>
<tr afd:render="ListSnippet"><td></td></tr>
</table>
```

Example 10.11

- body convention

  When a child template file uses "afd:extension" to declare its parent template, only the body part(all the children of the body except the body tag itself) of the child template will be processed by the template engine and the parts out of body will be ignored simply. The same convention is applied to the embed target file. The following two sample will not cause different results by Asta4D.

```
<afd:extension parent="parent.html">

    <afd:block override="content ">
        <div>hello</div>
        <afd:embed target="/templates/embed.html" ></afd:embed>
    </afd:block>

</afd:extension>
```

Example 10.12 child.html without body tag:

```
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<afd:extension parent="parent.html">

    <afd:block override="content ">
        <div>hello</div>
        <afd:embed target="/templates/embed.html" ></afd:embed>
    </afd:block>

</afd:extension>
</body>
</html>
```

Example 10.13 child.html with body tag:

This convention can be used for editor compatibility in case of your prefer html editor requires correct meta information (We add charset="UTF-8" to all of our template files for this reason). Note that this convention is only available on extended child template file or embedded target file, the parts out of body tag will be processed normally in the parent template file which is as a root template without further parent.

Further, explicit parameter information can be commented at body tag in parameterizable embed target file for better source readability.

```
<html>
<head><meta charset="UTF-8"></head>
<body itemsize="{Integer}" itemlist="{List}">
    <afd:block append="block1">
        <link href="embed.css" rel="stylesheet" type="text/css" />
    </afd:block>
    <div>good embed</div>
</body>
</html>
```

In Example 10.14, “embed.html”, we put some "strange" information in the body tag, which suggests that when this file is embedded, it requires an Integer parameter named "itemsize" and a List parameter named "itemlist". Note that such declaration can only be regarded as a hint for source reader and there is no warranty about any parameter check. You should consider it as a sample of how you can make interesting use of the body convention.
Example 10.14 embed.html

- "afd" name space

  The "afd" name space is configured as the default name space of Asta4D's special tags. This name space can be changed by Configuration#setTagNameSpace.

- TemplateResolver

  Asta4D searches certain template file by configured TemplateResolver. There are three pre-implemented TemplateResolver: FileTemplateResolver, ClasspathTemplateResolver and WebApplicationTemplateResolver. The default configured WebApplicationTemplateResolver will search template files with awareness of servlet container. The FileTemplateResolver and ClasspathTemplateResolver can be used for test purpose, you can also provide your own TemplateResolver for special search mechanism.

11. Details of snippet class

11.1. Rendering APIs

Table of Contents

Asta4D provides various rendering APIs to help developers render value to the page. By those APIs, developer can render text under a DOM element, set the attribute value for a DOM element, also can convert a list data to a list of DOM element, and also other various manipulation on DOM elements.

Create and add Renderer instance

There are almost same two sets of overloaded methods: create and add. The create methods are static and can be used to create a Renderer instance. The add methods are instance methods and can be used to add a implicitly created Renderer instance to an existing Renderer instance. Both of the create and add methods return the created Renderer instance therefore chain invoking can be performed as well.

```
Renderer renderer = Renderer.create("#someId", "xyz").add("sometag", "hello");
renderer.add(".someclass", "abc").add(".someclass2", "abc2");

Renderer renderer2 = Renderer.create("#someId2", "xyz2");
Renderer renderer3 = Renderer.create("#someId3", "xyz3");

//add renderer2 and renderer3 to renderer
renderer.add(renderer2);
renderer.add(renderer3);
```

Example 11.1

Note that the order of a Renderer instance being added is significant but the target instance which a Renderer is added to has no effect on the rendering order. In the following example, "add renderer2 to renderer then add renderer3 to renderer2" is completely equal to "add renderer2 and renderer3 to renderer" at above example.

```
//add renderer2 to renderer then add renderer3 to renderer2
renderer.add(renderer2);
renderer2.add(renderer3);
```

Example 11.2

The following is equal too:

```
//add renderer3 to renderer2 then add renderer2 to renderer
renderer2.add(renderer3);
renderer.add(renderer2);
```

Example 11.3

A instance of Renderer should not be considered as a single rendering declaration only, A instance of Renderer is exactly a rendering chain holder, you can call add method on any instance of the chain but the added Renderer instance will be always added to the tail of the chain. If the added Renderer instance is holding over than one Renderer instance in its own chain, the held chain will be added to the tail of the chain of the target Renderer.

There is also a non-parameter create method which can by used to create a "do nothing" Renderer for source convenience. In our practice, we write the following line at the beginning of most of our snippet methods.

```
Renderer renderer = Renderer.create();
```

Example 11.4

In following sections, we will introduce add method only, but you should remember that there should be a equal create method for most cases. You can also read the Java doc of Renderer for more details.

CSS Selector

Asta4D is using a modified version of jsoup library[1] to afford CSS selector function. Currently, we support the following selectors:

Table 11.1. Supported selectors

Pattern	Matches	Example
*	any element	*

[1] http://jsoup.org/

Pattern	Matches	Example
tag	elements with the given tag name	div
ns|E	elements of type E in the namespace ns	fb|name finds <fb:name> elements
#id	elements with attribute ID of "id"	div#wrap, #logo
.class	elements with a class name of "class"	div.left, .result
[attr]	elements with an attribute named "attr" (with any value)	a[href], [title]
[^attrPrefix]	elements with an attribute name starting with "attrPrefix". Use to find elements with HTML5 datasets	[^data-], div[^data-]
[attr=val]	elements with an attribute named "attr", and value equal to "val"	img[width=500], a[rel=nofollow]
[attr^=valPrefix]	elements with an attribute named "attr", and value starting with "valPrefix"	a[href^=http:]
[attr$=valSuffix]	elements with an attribute named "attr", and value ending with "valSuffix"	img[src$=.png]
[attr*=valContaining]	elements with an attribute named "attr", and value containing "valContaining"	a[href*=/search/]
[attr~=regex]	elements with an attribute named "attr", and value matching the regular expression	img[src~=(?i)\\.(png|jpe?g)]
	The above may be combined in any order	div.header[title]
	Combinators	
E F	an F element descended from an E element	div a, .logo h1
E > F	an F direct child of E	ol > li
E + F	an F element immediately preceded by sibling E	li + li, div.head + div

Pattern	Matches	Example
E ~ F	an F element preceded by sibling E	h1 ~ p
E, F, G	all matching elements E, F, or G	a[href], div, h3
	Pseudo selectors	
:lt(n)	elements whose sibling index is less than n	td:lt(3) finds the first 2 cells of each row
:gt(n)	elements whose sibling index is greater than n	td:gt(1) finds cells after skipping the first two
:eq(n)	elements whose sibling index is equal to n	td:eq(0) finds the first cell of each row
:has(selector)	elements that contains at least one element matching the selector	div:has(p) finds divs that contain p elements
:not(selector)	elements that do not match the selector. See also Elements.not(String)	div:not(.logo) finds all divs that do not have the "logo" class. div:not(:has(div)) finds divs that do not contain divs.
:contains(text)	elements that contains the specified text. The search is case insensitive. The text may appear in the found element, or any of its descendants.	p:contains(jsoup) finds p elements containing the text "jsoup".
:matches(regex)	elements whose text matches the specified regular expression. The text may appear in the found element, or any of its descendants.	td:matches(\\d+) finds table cells containing digits. div:matches((?i)login) finds divs containing the text, case insensitively.
:containsOwn(text)	elements that directly contains the specified text. The search is case insensitive. The text must appear in the found element, not any of its descendants.	p:containsOwn(jsoup) finds p elements with own text "jsoup".
:matchesOwn(regex)	elements whose own text matches the specified regular expression. The text must appear in the found element, not any of its descendants.	td:matchesOwn(\\d+) finds table cells directly containing digits. div:matchesOwn((?i)login) finds divs containing the text, case insensitively.

Pattern	Matches	Example
	The above may be combined in any order and with other selectors	.light:contains(name):eq(0)
	Structural pseudo selectors	
:root	The element that is the root of the document. In HTML, this is the html element. In a snippet method, this is the element where the current snippet method is declared. In a recursive Renderer, this is the target element which the current renderer is applied to.	:root
:nth-child(an+b)	elements that have an+b-1 siblings before it in the document tree, for any positive integer or zero value of n, and has a parent element. For values of a and b greater than zero, this effectively divides the element's children into groups of a elements (the last group taking the remainder), and selecting the bth element of each group. For example, this allows the selectors to address every other row in a table, and could be used to alternate the color of paragraph text in a cycle of four. The a and b values must be integers (positive, negative, or zero). The index of the first child of an element is 1. In addition to this, :nth-child() can take odd and even as arguments instead. odd has the same signification as 2n+1, and even has the same signification as 2n.	tr:nth-child(2n+1) finds every odd row of a table. :nth-child(10n-1) the 9th, 19th, 29th, etc, element. li:nth-child(5) the 5h li
:nth-last-child(an+b)	elements that have an+b-1 siblings after it in the document tree. Otherwise like :nth-child()	tr:nth-last-child(-n+2) the last two rows of a table

Pattern	Matches	Example
:nth-of-type(an+b)	pseudo-class notation represents an element that has an+b-1 siblings with the same expanded element name before it in the document tree, for any zero or positive integer value of n, and has a parent element	img:nth-of-type(2n+1)
:nth-last-of-type(an+b)	pseudo-class notation represents an element that has an+b-1 siblings with the same expanded element name after it in the document tree, for any zero or positive integer value of n, and has a parent element	img:nth-last-of-type(2n+1)
:first-child	elements that are the first child of some other element.	div > p:first-child
:last-child	elements that are the last child of some other element.	ol > li:last-child
:first-of-type	elements that are the first sibling of its type in the list of children of its parent element	dl dt:first-of-type
:last-of-type	elements that are the last sibling of its type in the list of children of its parent element	tr > td:last-of-type
:only-child	elements that have a parent element and whose parent element hasve no other element children	
:only-of-type	an element that has a parent element and whose parent element has no other element children with the same expanded element name	
:empty	elements that have no children at all	

Note: Because of the internal implementation of how to perform Renderer on target element, there may be unexpected temporary elements to be added into the Current DOM tree, which will be removed safely at the final stage of page production but may cause the structural pseudo selectors work incorrectly (However, the :root selector can still work well).

Render text

add(String selector, String value) can be used to render a text under the element specified by given selector. All child nodes of the target element specified by selector will be emptied and the given String value will be rendered as a single text node of the target element.

```
renderer.add("#someId", "xyz");
```

Example 11.5

Long/long, Integer/int, Boolean/boolean will be treated as text rendering too.

```
renderer.add("#someIdForLong", 123L);
renderer.add("#someIdForInt", 123);
renderer.add("#someIdForBool", true);
```

Example 11.6

Render DOM attribution

add(String selector, String attr, String value) can be used to render attribute value of a DOM element. There are some rules will be applied for the pattern of specified "attr" and "value":

- add("+class", value)

  call addClass(value) on target Element, null value will be treated as "null".

- add("-class", value)

  call removeClass(value) on target Element, null value will be treated as "null".

- add("class", value)

  call attr("class", value) on target Element if value is not null, for a null value, removeAttr("class") will be called.

- add("anyattr", value)

  call attr("anyattr", value) on target Element if value is not null, for a null value, removeAttr("anyattr") will be called.

- add("anyattr", SpecialRenderer.Clear)

  call removeAttr("anyattr") on target Element.

- add("+anyattr", value)

  call attr("anyattr", value) on target Element if value is not null, for a null value, removeAttr("anyattr") will be called.

- add("+anyattr", SpecialRenderer.Clear)

  call removeAttr("anyattr") on target Element.

- add("-anyattr", value)

  call removeAttr("anyattr") on target Element.

There is also an add method for attribution rendering that accepts arbitrary data as Object type: add(String selector, String attr, Object value). When the "value" is specified as a non-string value and the "attr" is specified as "+class" or "-class", A IllegalArgumentException will be thrown. When an arbitrary Object value is rendered to attribute by such method, an internal object reference id will be rendered to the target attribute instead of the original object since it cannot be treated as attribute string value directly. The object reference id can be used by variable injection for the nested snippet rendering. See the following example, we pass a Date instance to the nested snippet method:

```
<div afd:render="MySnippet:outer">
  <div id="inner" afd:render="MySnippet:inner">
    <span id="current-date"></span>
  </div>
</div>
```

```
public class MySnippet{

    public Renderer outer(){
        return Renderer.create("#innder", "now", new Date());
    }

    public Renderer inner(Date now){
        return Renderer.create("#current-date", now);
    }
}
```

Example 11.7

This mechanism can be used for parametrized embedding too. Parameters can be specified by attribution rendering in the snippet method of parent template file and can be retrieved by the snippet method of child template file as same as the above example.

```
<!-- parent template -->
<div afd:render="MySnippet:outer">
  <afd:embed id="inner" target="child.html"></afd:embed>
</div>
```

```
<!-- child template -->
  <div afd:render="MySnippet:embed">
    <span id="current-date"></span>
  </div>
```

```
public class MySnippet{

    public Renderer outer(){
        return Renderer.create("#innder", "now", new Date());
    }

    public Renderer embed(Date now){
        return Renderer.create("#current-date", now);
    }
}
```

Example 11.8

Clear an Element

On all the rendering methods, if the specified value is null, the target element will be removed. And there is also an enumeration value of SpecialRenderer.Clear which can be used to declare a remove operation for an element too.

```
//if the String value is null, the target element will be removed
String txt = null;
Renderer.create("#someId", txt);
```

```
import static com.astamuse.asta4d.render.SpecialRenderer.Clear

Renderer.create("#someId", Clear);
```

Example 11.9

Note that on the attribute rendering methods, if null value or SpecialRenderer.Clear are specified, as we mentioned in the previous section, the target attribute will be removed and the target element will remain.

Render raw Element

An existing DOM element can be replaced by a new element by specifying the new element as the rendering value. The new element can be generated by DOM APIs or simply parsed from raw html source.

```
Renderer renderer = Renderer.create();

Element ul = new Element(Tag.valueOf("ul"), "");
List<Node> lis = new ArrayList<>();
ul.appendChild(new Element(Tag.valueOf("li"), "").appendText("This text is created by
 snippet.(1)"));
ul.appendChild(new Element(Tag.valueOf("li"), "").appendText("This text is created by
 snippet.(2)"));
ul.appendChild(new Element(Tag.valueOf("li"), "").appendText("This text is created by
 snippet.(3)"));

renderer.add("#someId", ul);

String html = "<a href=\"https://github.com/astamuse/asta4d\">asta4d hp</a>";
renderer.add("#hp-link", ElementUtil.parseAsSingle(html));
```

Example 11.10

There are several utility methods in ElementUtil that can help you operate DOM easier.
ElementUtil#parseAsSingle would cause potential cross-site issues, so take care of it and make sure that you have escaped all the raw html source correctly. Since Asta4D has provided plenty of rendering methods, you should try your best to avoid create element from raw html source. If you have to do something on raw element level, your first option should be DOM APIs and parsing raw html source should be your last alternative.

Arbitrary rendering for an Element

Sometimes you will want to access the rendering target on raw element level and Asta4D allow you access the rendering target element by the interface of ElementSetter. ElementSetter asks the implementation of a callback method "set(Element elem)". As a matter of fact, the text rendering and attribute rendering are achieved by built-in implementation of ElementSetter(TextSetter and AttributeSetter).There is also another built-in ElementSetter implementation called ChildReplacer which can replace the children of rendering target by given element.

```
public class ChildReplacer implements ElementSetter {

    private Element newChild;

    /**
     * Constructor
     *
     * @param newChild
     *            the new child node
     */
    public ChildReplacer(Element newChild) {
        this.newChild = newChild;
    }

    @Override
    public void set(Element elem) {
        elem.empty();
        elem.appendChild(newChild);
    }
}
```

Example 11.11 source of ChildReplacer

You can regard the built-in TextSetter, AttributeSetter and ChildReplacer as reference implementation in case of you need implement your own ElementSetter. Following example shows how to use ChildReplacer or declare an anonymous class for ElementSetter on rendering.

```
Renderer renderer = Renderer.create();

renderer.add("#someId", new ChildReplacer(ElementUtil.parseAsSingle("<div>aaa</div>")));

renderer.add("#someNode", new ElementSetter(){
    public void set(Element elem) {
        if(elem.tagName().equals("div")){
            elem.addClass("someClass);
        }
    }
});
```

Example 11.12

On element level, you can also parse element by Element#html(String html) method which will also cause potential cross-site issues. As same as the raw element rendering, parsing raw html source should be your last alternative.

Recursive rendering

For a snippet method, the returned Renderer will be applied to the target element which declares the current snippet method. In the returned Renderer chain, you can also specify recursive Renderer which is only applied to the element specified by given CSS selector.

```
Renderer.create("#someId", Renderer.create("a", "href", "https://github.com/astamuse/
asta4d"));
```

Example 11.13

Debug renderer

Because the Renderer is applied after your snippet method finished, it is difficult to debug if there is something wrong. Asta4D affords rendering debug function too. There are two overloaded debug method, one accepts a log message only and another accepts a log message and a selector. The debug renderer will output the current status of rendering target element to log file, if the selector is not specified, the whole rendering target of current snippet method will be output, if the selector is specified, only the matched child elements will be output. Log level can be configured by "com.astamuse.asta4d.render.DebugRenderer".

```
//the whole target rendering target will be output before and after
renderer.addDebugger("before render value");
renderer.add("#someId", value);
renderer.addDebugger("after render value");

//only the a tag element will be output before and after
renderer.addDebugger("before render value for link", "a");
renderer.add("a", "href", url);
renderer.addDebugger("after render value for link", "a");
```

If you specified a selector on debugger but do not get any output, it usually means you specified wrong selector or the target element has been removed by previous rendering or outer snippet rendering.

Example 11.14

Missing selector warning

If your specified selector on rendering method cannot match any element, Asta4D will output a warning message to log as *"There is no element found for selector [#someId] at [com.astamuse.asta4d.test.render.RenderingTest $TestRender.classAttrSetting(RenderingTest.java:73)], if it is deserved, try Renderer#disableMissingSelectorWarning() to disable this message and Renderer#enableMissingSelectorWarning could enable this warning again in your renderer chain"*.

Note that the place where the missing selector declared is output to log too, which is disabled by default since it is expensive on production environment. You can enable the source row number output by Configuration#setSaveCallstackInfoOnRendererCreation.

Sometimes, the missing selector may be designed by your logic, you can disable the warning message on your rendering chain temporarily as following:

```
renderer.disableMissingSelectorWarning();

renderer.add("#notExistsId", "abc");

renderer.enableMissingSelectorWarning();
```

Example 11.15

List rendering

Asta4D does not only allow you to rendering single value to an element, but also allow you to duplicate elements by list data, which is usually used to perform list rendering. java.lang.Iterable of String/Integer/Long/Boolean can be rendered directly as text rendering:

```
Renderer renderer = Renderer.create();

renderer.add("#strList", Arrays.asList("a", "b", "c"));

renderer.add("#intList", Arrays.asList(1, 2, 3));

renderer.add("#longList", Arrays.asList(1L, 2L, 3L));

renderer.add("#boolList", Arrays.asList(true, false, true));
```

Example 11.16

On list rendering, the matched element will be duplicated times as the size of the list and the value will be rendered to each duplicated element. You can also perform complex rendering for arbitrary list data by RowConvertor:

```
render.add("#someIdForRenderer", Arrays.asList(123, 456, 789), new RowConvertor<Integer,
 Renderer>() {
    @Override
    public Renderer convert(int rowIndex, Integer obj) {
        return Renderer.create("#id", "id-" + obj).add("#otherId", "otherId-" + obj);
    }
});
```

Example 11.17

Since we will return Renderer in most cases, the RowConvertor can be replaced by RowRenderer which can omit the type declaration of Renderer:

```
render.add("#someIdForRenderer", Arrays.asList(123, 456, 789), new RowRenderer<Integer>()
 {
    @Override
    public Renderer convert(int rowIndex, Integer obj) {
        return Renderer.create("#id", "id-" + obj).add("#otherId", "otherId-" + obj);
    }
});
```

Example 11.18

The returned Renderer by RowConvertor/RowRenderer will be applied to each duplicated element by the given list data.

On list rendering, the rendering method accepts java.lang.Iterable rather than java.util.List, which afford more flexibilities to developers. Note that if the given iterable is empty, the target element will be duplicated 0 times, which means the target element will be removed.

Asta4D also afford parallel list rendering for you, simply use ParallelRowConvertor/ ParallelRowRenderer instead of RowConvertor/RowRenderer:

```
render.add("#someIdForRenderer", Arrays.asList(123, 456, 789), new
 ParallelRowRenderer<Integer>() {
    @Override
    public Renderer convert(int rowIndex, Integer obj) {
        return Renderer.create("#id", "id-" + obj).add("#otherId", "otherId-" + obj);
    }
});
```

Example 11.19

The parallel list rendering ability is provided by an utility class called ListConvertUtil which basically provides various transform methods for list data conversion like the map function in Scala/Java8. *For Java8, lambda and stream api support will be added in future.*

The ListConvertUtil uses a thread pool to perform parallel transforming, the size of pool can be configured by Configuration#listExecutorFactory#setPoolSize. There is also an import configuration of ParallelRecursivePolicy. When perform the parallel list transforming recursively, thread dead lock would potentially occur, so you have to choose a policy to handle this case:

- EXCEPTION

  When recursive parallel list transforming is identified, throw an RuntimeException.

- CURRENT_THREAD

When recursive parallel list transforming is identified, the child transforming will be performed in the same thread of parent transforming without picking up an usable thread from pool.

- NEW_THREAD

  When recursive parallel list transforming is identified, the child transforming will be performed in a newly generated thread out of the configured thread pool, the generated thread will be finished immediately after the child transforming finished.

We recommend EXCEPTION or CURRENT_THREAD and the default is EXCEPTION.

11.2. Other things about snippet and rendering

Table of Contents

Nested snippet

Snippet can be declared recursively.There is a convention that the outer snippet will always be executed on a prior order.

```
<div afd:render="OuterSnippet">
    <div id="inner" afd:render="InnerSnippet">
        <p id="name">name:<span>dummy name</span></p>
        <p id="age">age:<span>0</span></p>
    </div>
</div>
```

Example 11.20

In the above example, the "InnerSnippet" will be executed after the "OuterSnippet" has been executed, which also means you can configure the inner snippet dynamically.

```
public class OuterSnippet{

    public Renderer render(){
        return Renderer.create("#inner", "hasprofile", "true");
    }

}

public class InnerSnippet{

    public Renderer render(bool hasprofile){
        if(hasprofile){
            return Renderer.create("#name span", "Bob").add("#age span", 27);
        }else{
            return Renderer.create("*", Clear);
        }
    }

}
```

Example 11.21

InitializableSnippet

The snippet class instance is singleton in request scope and will be created at the first time a snippet is required. After the snippet class instance has been created, field value injection will be applied to the created instance once(About detail of value injection, see the later chapter). After all the field has been injected, the framework will check whether the current class implements the "InitializableSnippet" interface, if true, the init method of InitializableSnippet will be invoked once.

```
public static class InitSnippet implements InitializableSnippet {

    @ContextData
    private String value;

    private String resolvedValue;

    @Override
    public void init() throws SnippetInvokeException {
        resolvedValue = value + "-resolved";
    }
}
```

Example 11.22

In the above example, the field "value" will be injected after the instance is created, then the init method will be applied to finish the complete initialization logic of the snippet class.

Component rendering

For the view first policy, we will treat the template files as first class things. The embed file mechanism allow you separate some view blocks as independent components at template file layer. There is

also a snippet class level mechanism that afford you the same ability as embed file, which is called "Component".

Why we need "Component"? The embed file can only be include statically, but the "Component" is an extendible Java class which can be configured by arbitrary Java code and polymorphism can be easily performed too. Basically, The "Component" mechanism can help developer to build independent view component easier than static embed file.

The constructor of Component accepts a string value as an embed file path or an instance of Element, and also an optional AttributesRequire can be specified to provide some initialization parameters as same as the parametrized embedding introduced in the previous section.

```
    public Component(Element elem, AttributesRequire attrs) throws Exception {
        ...
    }

    public Component(Element elem) throws Exception {
        ...
    }

    public Component(String path, AttributesRequire attrs) throws Exception {
        ...
    }

    public Component(String path) throws Exception {
        ...
    }
```

Example 11.23 constructors of Component

```
public Renderer render(final String ctype) throws Exception {
    return Renderer.create("span", new Component("/
ComponentRenderingTest_component.html", new AttributesRequire() {
        @Override
        protected void prepareAttributes() {
            this.add("value", ctype);
        }

    }));
}
```

In this example, the target element specified by the selector "span" will be completely replaced by the result of Component#toElement().

Example 11.24 render a component

Commonly, we do not render a component as above example, we usually extend from Component and expose a constructor with business initialization parameters or supply a group of business initialization methods, by which we can build an independent view component simply.

Further, Component also has a "toHtml" method which will return the html source of the component rendering result. This method can be used by ajax request to acquire a dynamically rendered component.

SnippetInterceptor

SnippetExtractor

SnippetResolver

SnippetInvoker

12. Variable injection

12.1. Context

Table of Contents

Context/WebApplicationContext

Asta4D use "Context" to manage data life cycle. The "Context" in asta4d-core supplies the basic data management mechanism and the WebApplicationContext from asta4d-web afford more powerful extending for web application development. Commonly, we will only use WebApplicationContext in our application. There are two ways to retrieve an instance of WebApplicationContext.

```
WebApplicationContext context = Context.getCurrentThreadContext();
```

Example 12.1 Retrieve an instance of WebApplicationContext by static method

```
public class PageSnippet {

    public Renderer render(WebApplicationContext context) {
        ...
    }
}
```

Example 12.2 Retrieve an instance of WebApplicationContext by injection

As the name of "getCurrentThreadContext" suggested, the context is managed per thread, and also that the context will be initialized before every http request is processed and will be cleared after every http request process has finished.

WebApplicationContext affords several methods to retrieving raw servlet apis and other helpful information:

- getRequest

  Return the HttpServletRequest instance of current request.

- getResponse

  Return the HttpServletResponse instance of current request.

- getServletContext

  Return the ServletContext instance of current request.

- getAccessURI

  Return the rewritten url of current request. For details about URL rewritting, see later chapter.

Scope

The context provides a mechanism to manage data by scopes. There are only 3 scopes supplied by asta4d-core's Context, but the WebApplicationContext from asta4d-web supply more scopes to help web application development.
Available scopes(with the constant name):

global(SCOPE_GLOBAL)
: The data saved in global scope can be accessed cross all the contexts, you can treat it as a static object pool.

default(SCOPE_DEFAULT)
: The data saved in default scope can only be access by the current context.

element attribute(SCOPE_ATTR)
: This scope is related to snippet rendering. An element attribute scope is a wrapping of element attribute. When you try to retrieve data from element attribute scope, the attribute of current rendering target element will be checked and the corresponding attribute value will be returned, further, if there is no corresponding attribute in the current rendering target element's attributes, all the parents of current rendering target element will be checked recursively.

request(SCOPE_REQUEST)
: The data saved in default scope can only be access in the current http request process. Note that the request scope is simply delegated to the default scope and it is different from servlet api's http request attribute.

path var(SCOPE_PATHVAR)
: Path var scope represents the variables declared in url rules.

query parameter(SCOPE_QUERYPARAM)
: Query parameter scope represents the query parameters specified at the part after question mark in an url.

session(SCOPE_SESSION)
: Session scope represents the data saved in http session.

header(SCOPE_HEADER)
: Session scope represents the header data from a http request.

cookie(SCOPE_COOKIE)
: Cookie scope represents the cookie data from a http request.

flash(SCOPE_FLASH)
: Asta4D affords a mechanism to allow pass data to the redirected http request, which is called flash data.

Normally, the data saved in context can be accessed by variable injection. If you need get data manually, you can do it as following:

```
WebApplicationContext context = Context.getCurrentThreadContext();

//from default scope
Integer count = context.get("saved-count");

//from session
Integer sessionCount = context.get(WebApplicationContext.SCOPE_SESSION, "saved-count");
```

Example 12.3 Retrieve data from context

You can implement your own customized scope or afford you customized implementation for existing scopes. You need extend from WebApplicationContext then override the method called "createMapForScope".*(If you only want to split your data from default scope, you can simply specify a new scope name and a new scope will be created automatically, the created scope's life cycle is as the same as default scope)*

```
public class MyContext extends WebApplicationContext {
    @Override
    protected ContextMap createMapForScope(String scope) {
        ContextMap map = null;
        switch (scope) {
        case SCOPE_SESSION: {
            HttpServletRequest request = getRequest();
            map = new MemcachedSessionMap(request);
        default:
            map = super.createMapForScope(scope);
        }
        return map;
    }
}
```

Example 12.4 Customized memcached drived session management

See later chapter of Asta4dServerlet for how to configure Asta4D to use the customized Context implementation instead of the default WebApplicationContext.

ContextBindData

Since all the data query logic are performed at snippet class, we will want to avoid duplicated query on same page rendering. The recommended solution is caching your query result by cache library such as ehcache. But Asta4D also affords you an alternative solution for some simple cases, which is called ContextBindData.

ContextBindData can be treated as a lazy load instance field for Java class. It is warranted that a ContextBindData will be initialized only once in a same Asta4D context and it will only be initialized when you first call it. There are some exceptions for the warranty about initializing only once, we will explain them later.

```
public class PageSnippet {

    @ContextData
    private int count;

    private ContextBindData<Integer> data = new ContextBindData<Integer>(true) {
        @Override
        protected Integer buildData() {
            return count + 1;
        }

    };

    public Renderer render() {
        return Renderer.create("*", data.get());
    }
}
```

Example 12.5

You should have noticed that we pass a "true" to the constructor of ContextBindData, which means we require the warranty that it should be only initialize once in the current context. If we pass nothing or pass a "false" to the constructor, the buildData method may be invoked multiple times in multi-thread rendering, but which is faster than warranted initialization simply because of skipping the multi-thread lock process.

Again, we recommend to use cache library to avoid duplicated query, but you can use ContextBindData as a simple, light weight alternative. See detail of Section 9.3, “The best way of implementing a snippet”.

12.2. Injection

Table of Contents

@ContextData

All fields annotated by @ContextData in a snippet class will be initialized/injected automatically when the snippet class instance is initialized. Further, even the method parameters of snippet rendering method and request handle method are automatically injected, you can still customize the injection by extra @ContextData annotation.

Note that there are something different between field injection and method parameter injection. The fields of snippet class without @ContextData will not be injected, but method parameters will be always injected even without @ContextData.

Available configurations of @ContextData:

name
: The name of saved data in context. If it is not specified, the field name or parameter name will be used(This requires that you reverse the parameter names when compile).

scope
: The scope of saved data in context. If it is no specified, Asta4D will try to search the data in all the scopes by a predefined order:

  1. element attribute(SCOPE_ATTR)
  2. path var(SCOPE_PATHVAR)
  3. query parameter(SCOPE_QUERYPARAM)
  4. flash(SCOPE_FLASH)
  5. cookie(SCOPE_COOKIE)
  6. hearder(SCOPE_HEADER)
  7. request/default(SCOPE_REQUEST/SCOPE_DEFAULT)
  8. session(SCOPE_SESSION)
  9. global(SCOPE_GLOBAL)

  This order can be configured via WebApplicationContextDataFinder#setDataSearchScopeOrder.See details at the section called “ContextDataFinder”

typeUnMatch
: The policy of how to handle the unmatched type on data conversion. The default action is throw Exception and another two value of "DEFAULT_VALUE" and "DEFAULT_VALUE_AND_TRACE" can be specified.

There are several scope specified annotations can be used as convenience.

- @SessionData
- @QueryParam
- @PathVar
- @HeaderData
- @CookieData
- @FlashData

All the above annotation have the same configurable items with the original @ContextData annotation except being fixed with according data scope.

@ContextDataSet

A pojo class annotated by @ContextDataSet will be treated as a variable holder. If the type of a snippet class field or a method parameter is a class annotated by @ContextDataSet, when Asta4D do injection,

a new instance will be created at first, then all the fields of current instance will be scanned and all the fields annotated by @ContextData will be injected as same as the snippet class field injection. @ContextDataSet can be used to allocate all of the form data into one single instance as a convenience.

```
@ContextDataSet
public static class MySet {

    @ContextData
    private long f1;

    @ContextData
    private String f2;

    public long getF1() {
        return f1;
    }

    public String getF2() {
        return f2;
    }
}

public static class MySnippet {

    public void render(MySet holder) {
    }

}

public static class MySnippet2 {

    @ContextData
    private MySet holder;

    public void render() {
    }

}
```

Note that even you declared @ContextDataSet on the class, you still need to declare the @ContextData on the field declaration. For method parameter, it is not necessary since all the parameters are injected automatically. Further, for a declared @ContextDataSet, you still need to declare @ContextData on every field you want to be injected.

Example 12.6

Available configurations of @ContextData:

singletonInContext
: A boolean value to indicate that whether the target ContextDataSet should be singleton in a single context life cycle. The default value is false, which means a new instance would be created at every time when a ConextDataSet is required to inject.

factory

A factory class can be used to indicate how to create an instance of ContextDataSet. The default factory will call Class#newInstance() to create a new instance of ContextDataSet.

ContextDataFinder

Asta4D search data in context by pre configured ContextDataFinder implementation. The default implementation is WebApplicationContextDataFinder. You can supply your own search mechanism by implementing the ContextDataFinder interface or extending from WebApplicationContextDataFinder. WebApplicationContextDataFinder defines the scope search order if scope is not specified, and also returns some special data by hard coding type check:

- Context/WebApplicationContext
- ResourceBundleHelper
- ParamMapResourceBundleHelper
- HttpServletRequest
- HttpServletResponse
- ServletContext
- UrlMappingRule

The first 3 types are afforded by the parent class of WebApplicationContextDataFinder: DefaultContextDataFinder. To supply your own logic, you can override the findDataInContext method and the WebApplicationContextDataFinder's source can be treated as reference implementation.

DataConvertor

ContextDataHolder

13. URL rule

13.1. Rule apis

Table of Contents

UrlMappingRuleInitializer

All the url mapping rules can be supplied by implementing the UrlMappingRuleInitializer interface which can be configured by WebApplicationConfiguration#setUrlMappingRuleInitializer.

UrlMappingRuleInitializer's initUrlMappingRules method does not return configured rules directly, it supply a UrlMappingRuleHelper instance by parameter instead. The UrlMappingRuleHelper afford a group of methods to help developers build url mapping rules.

UrlMappingRuleHelper allows add new rule by add methods, which will be discussed in next section, let us see the global configuration methods of UrlMappingRuleHelper at first.

- addDefaultRequestHandler(Object... handlerList)
- addDefaultRequestHandler(String attribute, Object... handlerList)

  There are two overloaded methods of addDefaultRequestHandler. Added request handlers without attribute declaration will be applied to all the rules and the ones with attribute declaration will be applied to therules with same attribute only.

- addGlobalForward(Object result, String targetPath)
- addGlobalForward(Object result, String targetPath, int status)

  There are two overloaded methods of addGlobalForward. The one with status parameter can customize the http response code with specified status code or the default code 200 will be returned to client.

- addGlobalRedirect(Object result, String targetPath)

  A global redirect result mapping can be declared. There is a convention about the format of target path(the normal redirect on special rules follows the same convention):

  - common url string("http://www.example.com/somepage.html")

    The target url will be redirected as temporary(code 302).

  - common url string with prefix("301:http://www.example.com/somepage.html")

    The target url will be redirected as specified code by the prefix. 301 or 302 are acceptable and "p" or "t" can be used to represent "permanent" or "temporary" as well(which means "301:http://www.example.com/somepage.html" equals to "p:http://www.example.com/somepage.html").

- addRequestHandlerInterceptor(Object... interceptorList)

- addRequestHandlerInterceptor(String attribute, Object... interceptorList)

  The request handler interceptor is need to be clarified more in the implementation. We skip the description at present.

- registerRestTransformer(ResultTransformer transformer)

- registerJsonTransformer(ResultTransformer transformer)

  The returned value of a request handler which is declared as json at rule will be transformed to standard json string, and developers can still register a customized result transformer to generate the response by the own way. Further, even the framework will do nothing on the returned value of a handler if which is declared on a rest rule, developer can still use "registerRestTransformer" to add customized result transforming on a rule declared as rest.

- setDefaultMethod(HttpMethod defaultMethod)

  The default matching http method for url patterns without http method specified. The default value is GET.

Handy rules

There several add methods from UrlMappingRuleHelper which can be used to add new url rules. According to the certain being called add method, different interface which is called handy rules will be returned for further declaration. Further, all the methods of the returned handy rule interface will return different interface according to the certain method you are calling, it is complex to describe which interface will be returned on certain case, basically we suppose that the code auto completion function of you editor would help you to know what you can do in the next step. All of these handy rules and their methods can be treated as a group of DSL to help you build your own url rules for your site. We will describe all the available methods at following regardless of which certain interface it belongs to. After that, we will also describe the basic rule of how the returned handy rules change.

- add(String sourcePath)

- add(String sourcePath, String targetPath)

- add(HttpMethod method, String sourcePath)

- add(HttpMethod method, String sourcePath, String targetPath)

  Add a new url rule. If the http method is not specified. the configured default method(see above setDefaultMethod) will be specified by default. If you want the sourcePath is matched to all the http methods, null can be specified. The targetPath should be a template file path or a redirect target url string with prefix "redirect:", the part after prefix "redirect:" follows the convention of redirect string format introduced in global redirect configuration.

- id(String id)

  One added rule can be identified by a unique id.

- reMapTo(String ruleId)

  Multi url patterns can be mapped to the same rule by reMapTo method. See following sample:

```
rules.add("/app/", "/templates/index.html").id("index-page");

rules.add("/app/index").reMapTo("index-page");

rules.add("/app/handler").id("app-handler")
     .handler(LoginHandler.class)
     .handler(EchoHandler.class)
     .forward(LoginFailure.class, "/templates/error.html")
     .forward("/templates/success.html");

rules.add("/app/ex-handler").reMapTo("app-handler").attribute("ex-handler");
```

In above sample, "/app/index" will be treated as same as the rule identified as "index-page" which sourc path is "/app/". Aslo the "/app/ex-handler" will be configured as same as "/app/handler" with its special attribute configuration "ex-handler". Special path variable and priority can be configured to remapped rules as well.
Example 13.1 Remap the rule to a existing rule

- attribute(String attribute)

  Multi attributes can be configured to a rule. The attributes can be used by global rule configurations to add certain extra operations on certain urls which is attributed by certain attribute.

- var(String key, Object value)

  Static path variables can be configured by this method, that can be accessed by context's path var scope and also can be automatically injected to snippets or request handlers.

- handler(Object... handlerList)

  Request handlers can be configured by this methods. Multi request handlers are acceptable and the details about multi request handlers will be explained in later section.

  The parameter of handler method can be arbitrary type: an instance of java.lang.Class or an arbitrary instance. The framework explains received parameters by the implementation of DeclareInstanceResolver configured by WebApplicationConfiguration. The default implementation provided by framework follows the following rules to explain the declaration of request handler:

  1. If an instance of java.lang.Class is specified, the instance of request handler will be created by invoke "newInstance()" on the specified Class.

  2. If a string is specified, the string will be treated as a class name and an instance of java.lang.Class will be created by calling "Class#forName", then the instance of request handler will be created by invoke "newInstance()" on the created Class.

  3. The specified parameter will be treated as a request handler directly if it is neither a Class nor a string. By this rule, it is interesting that we can declare an anonymous class as a request handler:

```
rules.add("/app/handler")
     .handler(new Object(){
        @RequestHandler
        public void handle(){
            //
        }
     });
```

Example 13.2

The asta4d-spring package also provides a resolver based on Spring IOC container, the request handler instance will be retrieved by passing the specified parameter to the "Context#getBean" method of spring container. See details of integration of Spring IOC.

- forward(String targetPath)
- forward(String targetPath, int status)
- forward(Object result, String targetPath)
- forward(Object result, String targetPath, int status)

  The request will be forwarded to the target template file with specified status code. The result parameter will be used to match the result from request handler(if there is any request handler configured for current rule).About result matching, see details in next section.

- redirect(String targetPath)
- redirect(Object result, String targetPath)

  As the same as forward, the request will be redirected(301/302) to the target path. The target path string follows the convention of redirect string format introduced in global redirect configuration.

- json()

  The result of request handler will be converted to a json string and be returned the client with MIME type "application/json".

- rest()

  This method does nothing for the current url by default, but declared as a hint that suggests the current rule will act as a restful api and will return customized response which would be head only response in many cases.

13.2. Request handler result process

Table of Contents

This section will describe some internal mechanism of how asta4d handle the result from a request handler. Especially for the part of result transforming, which is not necessary for normal development but can help you understand how the request handler chain works.

Content provider

Result transforming

Request handler chain

14. details of form flow

To be written.

15. i18n

15.1. message stringization

Table of Contents

Asta4D affords basic i18n support by a built-in I18nMessageHelper. I18nMessageHelper requires a MessagePatternRetriever implementation to retrieve message by key. There is also a default implementation JDKResourceBundleMessagePatternRetriever which uses the JDK's built-in resource bundle mechanism.

I18nMessageHelper

I18nMessageHelper is an abstract classes which affords the basic interface for handling messages:

- getMessage(String key)
- getMessage(Locale locale, String key);
- getMessageWithDefault(String key, Object defaultPattern)

  return defaultPattern#toString() if message not found
- getMessageWithDefault(Locale locale, String key, Object defaultPattern)

  return defaultPattern#toString() if message not found

There are also two extended classes from I18nMessageHelper, which afford more flexible message functionalities especially about formatting message by given parameters. Configuration#setI18nMessageHelper can be used to customized which helper you want to use. The default is OrderedParamI18nMessageHelper. (*I18nMessageHelperTypeAssistant#getConfiguredMappedHelper and I18nMessageHelperTypeAssistant#getConfiguredOrderedHelper can be used to retrieve a type safe configured helper.*)

- MappedParamI18nMessageHelper

  Allow format message by a given parameter map, A MappedValueFormatter is required to supply concrete formatting style and the default is ApacheStrSubstitutorFormatter which uses StrSubstitutor from Apache Common lang3.
- OrderedParamI18nMessageHelper(default)

  Allow format message by a given parameter array. A OrderedValueFormatter is required to supply concrete formatting style and the default is JDKMessageFormatFormatter which uses JDK's MessageFormat to format message string.

There are several value formatter can be used by the above helpers. Notice that The MappedParamI18nMessageHelper requires implementation of MappedValueFormatter and the OrderedParamI18nMessageHelper requires implementation of OrderedValueFormatter.

Table 15.1. Built-in value formatters

formatter	interface	Description	Example
ApacheStrSubstitutorFormatter(default)	MappedValueFormatter	Use Apache Commons Lang's StrSubstitutor#replace[1] to format messages. Use "{" and "}" to represent variables place holder.	"There are {count} items in the {item-name} list."
JDKMessageFormatFormatter(default)	OrderedValueFormatter	Use JDK's MessageFormat#format[2] to format messages.	"There are {1} items in the {2} list."
SymbolPlaceholderFormatter	OrderedValueFormatter	Use JDK's String#format[3] to format messages.	"There are %d items in the %s list."

MessagePatternRetriever

The I18nMessageHelper requires a MessagePatternRetriever implementation to retrieve message by key. There is a default implementation JDKResourceBundleMessagePatternRetriever which uses the JDK's built-in resource bundle mechanism. The base name of resource bundle files can be specified by JDKResourceBundleMessagePatternRetriever#setResourceNames which accepts a list of base name. The configured resource bundles will be passed to JDK's ResourceBundle#getBundle(String baseName, Locale locale, ClassLoader loader)[4] to retrieve messages.

As the implementation of JDKResourceBundleMessagePatternRetriever, if the given locale is null, the default locale decision mechanism will be performed. See the section called “Default locale”.

Further, the JDKResourceBundleMessagePatternRetriever accepts ResourceBundleFactory to allow customize how to retrieve a resource bundle. The default is CharsetResourceBundleFactory which allows specify the encoding of message bundle files. Another built-in implementation is the LatinEscapingResourceBundleFactory, as the name suggests, the message files must be escaped by JDK's standard mechanism(via native2ascii).

Default locale

The default locale will be decided by the following order and stop at the first non-null returned value.

1. Context.getCurrentThreadContext().getCurrentLocale()

[1] http://commons.apache.org/proper/commons-lang/javadocs/api-release/org/apache/commons/lang3/text/StrSubstitutor.html#replace(java.lang.CharSequence)
[2] http://docs.oracle.com/javase/7/docs/api/java/text/MessageFormat.html#format(java.lang.String, java.lang.Object...)
[3] http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#format(java.lang.String, java.lang.Object...)
[4] http://docs.oracle.com/javase/7/docs/api/java/util/ResourceBundle.html#getBundle(java.lang.String, java.util.Locale, java.lang.ClassLoader)

2. Locale.getDefault()

3. Locale.ROOT

Since the Locale.getDefault() will return the local machine's locale by default, you may need to override the default locale to ROOT at the staring of your web application.

afd:msg

"afd:msg" tag can be used to declare an i18n aware message in a template file. The basic usage is as following:

```
<section>
    <article>
        <div class="panel panel-default"><div class="panel-body">
            <!-- the inner html will be treated as default message if the key is not found
 in message bundle -->
            <afd:msg id="notexistingkey" key="sample.notexistingkey" p0="parameter works
 as well.">text here to be treated as default message.{0}</afd:msg><br/>
        </div></div>

        <div class="panel panel-default"><div class="panel-body">
            <!-- simply specify parameters on tag -->
            <afd:msg id="peoplecount" key="sample.peoplecount" p0="4">dummy text</
afd:msg><br/>
            <afd:msg id="peoplecount" key="sample.peoplecount" p0="7" locale="ja_JP">dummy
 text</afd:msg><br/>
        </div></div>

        <div class="panel panel-default"><div class="panel-body">
            <!-- the parameter can be rendered by snippet too -->
            <afd:snippet render="ComplicatedSnippet:setMsgParam">
                <afd:msg id="peoplecount" key="sample.peoplecount" p0="4">dummy text</
afd:msg><br/>

  <afd:msg id="peoplecount" key="sample.peoplecount" p0="7" locale="ja_JP">dummy text</
afd:msg><br/>
            </afd:snippet>
        </div></div>

        <div class="panel panel-default"><div class="panel-body">
            <!-- attribute started with "@" will be treated as a key -->

  <afd:msg id="weatherreport" key="sample.weatherreport" @p0="sample.weatherreport.sunny">#
#</afd:msg><br/>

  <afd:msg id="weatherreport" key="sample.weatherreport" @p0="sample.weatherreport.sunny" locale="ja_JP">#
#</afd:msg><br/>
        </div></div>

        <div class="panel panel-default"><div class="panel-body">
            <!-- attribute started with "#" will be treated as a sub key of current key,
 thus sample.weatherreport.sunny will be searched -->
            <afd:msg id="weatherreport" key="sample.weatherreport" #p0="rain">##</
afd:msg><br/>

  <afd:msg id="weatherreport" key="sample.weatherreport" #p0="rain" locale="ja_JP">##</
afd:msg><br/>
        </div></div>

        <div class="panel panel-default"><div class="panel-body">
            <!-- treat message as text -->
            <afd:msg key="sample.textUrl"></afd:msg><br/>

            <!-- treat message as html -->
            <afd:msg key="sample.htmlUrl"></afd:msg><br/>

            <!-- treat message as text even it begins with html: -->
            <afd:msg key="sample.escapedUrl"></afd:msg><br/>
        </div></div>
    </article>
</section>
```

```
public Renderer setMsgParam() {
    Renderer render = Renderer.create();
    render.add("#peoplecount", "p0", 15);
    return render;
}
```

There are something need to be explained:

- default message

  As commented in the source, the inner content of "afd:msg" will be treated as default message in case of the message of specified key cannot be found. Note that the default message will be formatted by specified parameters too.

  This mechanism can also be used as a simple text replacing mechanism in the html template file in case of necessity.

- parameter name

  If the configured I18nMessageHelper is MappedParamI18nMessageHelper, the attribute names of current "afd:msg" element will be treated as parameter names directly. And if the configured I18nMessageHelper is OrderedParamI18nMessageHelper which is the default one, the attribute names started with single "p" and following with 0-started numbers will be treated as the values of parameter value array.

- recursive parameter

  The attributes which name started with "@" or "#" will be treated as recursive message's key. The different between "@" and "#" is that the "@" will be treated as a complete key but the "#" will be treated as a sub key of current message key.

- parameter value

  The value of attributes can be specified directly in the html template file, also can be rendered by a snippet as in the above example.

- locale

  Locale can be specified explicitly by ISO 639 locale string, if not, the default locale decision mechanism will be performed. See the section called “Default locale”.

15.2. file search order

The built-in template resolver and static resource handler can match the target file name with awareness of locale. The match locale will be decided by the default locale decision mechanism(See the section called “Default locale”). After the locale is decided, for example, for a locale as Locale("fr", "CA", "UNIX") to search a html template file "index.html", the target file will be searched by following order:

1. index_fr_CA_UNIX.html

2. index_fr_CA.html

3. index_fr.html
4. index.html

For the static resource files such as js, css, jpg, etc. will be searched by the same order as well.

16. Asta4dServlet and configuration

16.1. Asta4dServlet

All the http requests are handled by Asta4dServlet which allows developers to override several methods to customize its actions. The web.xml would be like following:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_2_5.xsd">

    <!-- standalone without spring -->
    <servlet>
        <servlet-name>Asta4D Servlet</servlet-name>
        <servlet-class>com.astamuse.asta4d.web.servlet.Asta4dServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Asta4D Servlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Example 16.1

The following methods can be overridden:

- createConfiguration()

  Overriding this methods allows developers to configure the framework by code.

```
    @Override
    protected WebApplicationConfiguration createConfiguration() {
        WebApplicationConfiguration conf = super.createConfiguration();

        conf.setSnippetResolver(new DefaultSnippetResolver() {

            @Override
            protected Object createInstance(String snippetName) throws
SnippetNotResovlableException {
                return super.createInstance("com.astamuse.asta4d.sample.snippet." +
snippetName);
            }

        });

        conf.setUrlMappingRuleInitializer(new UrlRules());

        conf.getPageInterceptorList().add(new SamplePageInterceptor());

        return conf;
    }
```

Example 16.2

- service()

  Developers can intercept the http request by overriding this method. The following example shows how to rewrite the access url by adding pre request logic:

```
    @Override
    protected void service() throws Exception {
        WebApplicationContext context = Context.getCurrentThreadContext();
        HttpServletRequest request = context.getRequest();
        String requestURI = request.getRequestURI();
        if (requestURI.startsWith("/abc/")) {
            context.setAccessURI(requestURI.substring(5));
        }
        super.service();
    }
```

Example 16.3

  Request time measuring can be performed too:

```
@Override
protected void service() throws Exception {
    long startTime = System.currentTimeMillis();
    try {
        super.service();
    } finally {
        long endTime = System.currentTimeMillis();
        System.out.println("request time:" + (endTime - startTime));
    }
}
```

Example 16.4

- createAsta4dContext()

  Overriding this methods allows developers to afford customized Context implementation.

- createConfigurationInitializer()

  Overriding this methods allows developers to customize the configuration file initialization logic. See details at next section.

16.2. Configuration file

In the previous section, we introduced that Asta4D can be configured by overriding createConfiguration() method and we also introduced a method called createConfigurationInitializer which allows customize the configuration file initialization logic. Asta4D allows to be configured by an extra configuration file which will be analyzed by the created configuration initializer.

The default implementation of configuration initializer will seek the configuration file by following order:

1. Calling retrievePossibleConfigurationFileNames(Overridable) to retrieve all possible configuration file names which could be an array of String. The default value is "asta4d.conf.properties" only.
2. Search all the possible names in classpath by order. The first found one will be used.
3. If there is no configuration file found in classpath, calling retrieveConfigurationFileNameKey(Overridable) to get a key. The default key is "asta4d.conf".
4. Retrieve the value of key retrieve in previous step by the order as: ServletConfig, JDNI Context, System properties.
5. If there is a value found in the previous step, open a file as configuration file by treating the found value as a file path.

As mentioned above, the two methods of retrievePossibleConfigurationFileNames and retrieveConfigurationFileNameKey can be overridden to customize configuration file name and the file path can be also configured by the key "asta4d.conf" at servlet mapping configuration(ServletConfig), JDNI configuration(JDNI context) or command line parameter(System properties).

Further, the default implementation does only support the format of configuration file as Java standard properties file. All the configured key/value will be treated as setXXX method calling on

WebApplicationConfiguration instance. The key can be a dot split string which will be treated as recursive property accessing on WebApplicationConfiguration instance.

```
urlMappingRuleInitializer=com.astamuse.asta4d.sample.UrlRules

cacheEnable=false

snippetExecutorFactory.threadName=asta4d-sample-snippet
snippetExecutorFactory.poolSize=100

listExecutorFactory.threadName=asta4d-sample-list
listExecutorFactory.poolSize=100
```

Example 16.5

The above configuration equals to the following source:

```
WebApplicationConfiguration conf = super.createConfiguration();

conf.setUrlMappingRuleInitializer(new UrlRules());

conf.setCacheEnable(false);

((DefaultExecutorServiceFactory) conf.getSnippetExecutorFactory()).setThreadName("asta4d-
sample-snippet");
((DefaultExecutorServiceFactory) conf.getSnippetExecutorFactory()).setPoolSize(100);

((DefaultExecutorServiceFactory) conf.getListExecutorFactory()).setThreadName("asta4d-
sample-list");
((DefaultExecutorServiceFactory) conf.getListExecutorFactory()).setPoolSize(100);
```

Example 16.6

All the configurable items can be found at JavaDoc[1].

[1] http://astamuse.github.io/asta4d/javadoc/0.14.5.1-SNAPSHOT/com/astamuse/asta4d/web/WebApplicationConfiguration.html

17. Integration with Spring

17.1. Integrate with Spring IOC

Asta4D can be integrated with any dependency injection mechanism. We will introduce how to integrate with Spring IOC as example.

There is a package called asta4d-spring which supplies reference implementation to show how to integrate a DI container to Asta4D. Firstly, as an alternative for Asta4dServlet introduced in the previous chapter, SpringInitializableServlet can be used to initialize Spring configuration automatically. The web.xml file should be like following:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_2_5.xsd">

    <!-- integrate with spring -->
    <servlet>
        <servlet-name>Asta4D Servlet</servlet-name>
        <servlet-class>com.astamuse.asta4d.misc.spring.SpringInitializableServlet</
servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring/configuration.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Asta4D Servlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

Example 17.1

Secondly, the WebApplicationConfiguration instance should be declared as a bean in spring with additional Spring special configurations:

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

  <bean id="asta4dConfiguration" class="com.astamuse.asta4d.web.WebApplicationConfiguration">
        <property name="snippetResolver">
            <bean class="com.astamuse.asta4d.misc.spring.SpringManagedSnippetResolver"/>
        </property>
        <property name="instanceResolverList">
            <list>

  <bean class="com.astamuse.asta4d.misc.spring.SpringManagedInstanceResolver"/>
            </list>
        </property>
    </bean>
```

The SpringManagedSnippetResolver will retrieve snippet instance from Spring container and the SpringManagedInstanceResolver will also retrieve request handler instance from Spring container. Further the SpringManagedSnippetResolver will treat the snippet name in html template files as bean id.
Example 17.2

The following example shows a full image of Spring's configuration for Asta4D:

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

  <bean id="asta4dConfiguration" class="com.astamuse.asta4d.web.WebApplicationConfiguration">
        <property name="snippetResolver">
            <bean class="com.astamuse.asta4d.misc.spring.SpringManagedSnippetResolver"/>
        </property>
        <property name="instanceResolverList">
            <list>

  <bean class="com.astamuse.asta4d.misc.spring.SpringManagedInstanceResolver"/>
            </list>
        </property>
        <property name="pageInterceptorList">
            <list>

  <bean class="com.astamuse.asta4d.sample.interceptor.SamplePageInterceptor"/>
            </list>
        </property>
        <property name="snippetInvoker">

  <bean id="snippetInvoker" class="com.astamuse.asta4d.snippet.DefaultSnippetInvoker">
                <property name="snippetInterceptorList">
                    <list>

  <bean class="com.astamuse.asta4d.sample.interceptor.SampleSnippetInterceptor"/>
                    </list>
                </property>
            </bean>
        </property>
        <property name="urlMappingRuleInitializer">
            <bean class="com.astamuse.asta4d.sample.UrlRules"/>
        </property>
    </bean>


  <bean id="ComplicatedSnippet" class="com.astamuse.asta4d.sample.snippet.ComplicatedSnippet" scope="prototy
>

  <bean id="FormSnippet" class="com.astamuse.asta4d.sample.snippet.FormSnippet" scope="prototype"/
>

  <bean id="SimpleSnippet" class="com.astamuse.asta4d.sample.snippet.SimpleSnippet" scope="prototype"/
>

  <bean id="ShowCodeSnippet" class="com.astamuse.asta4d.sample.snippet.ShowCodeSnippet" scope="prototype"/
>

    <bean class="com.astamuse.asta4d.sample.handler.LoginHandler"/>
```

Note that Asta4D always treats snippet instance as singleton per request for field injection on snippet, so the scope of snippet beans should be "prototype" basically except there is no field injection in snippet implementation. For the request handlers, "prototype" is not necessary since there is no field injection on request handler.

Example 17.3

17.2. Integrate with Spring MVC

Asta4D's template engine can be integrated to Spring MVC as view solution too. The following configuration shows how:

```
<bean class="com.astamuse.asta4d.misc.spring.mvc.Asta4dTemplateInitializer"/>
<bean class="com.astamuse.asta4d.misc.spring.mvc.SpringWebPageViewResolver"/>
```

Example 17.4