icw
eHEALTH FRAMEWORK

eHealth Framework

Validating a Service via Annotations

Imprint

InterComponentWare
Altrottstraße 31
69190 Walldorf
Tel.: +49 (0) 6227 385 0
Fax.: +49 (0) 6227 385 199

Document version: preliminary
Document Language: en (US)
Product Name: eHealth Framework
Product Version: 2.10.3
Last Change: 23.03.2010
Editorial Staff: BAS Technology

Notice

The wording in this document applies equally to women and men. The masculine form was selected to ease the comprehensibility and legibility of the text.

All company logos are a registered trademark of InterComponentWare AG.

The product names mentioned in this documentation are either trademarks or registered trademarks of the respective owners and are stated for identification purposes only.

Note:

The current version of this document has a draft status and various chapters are still in review.

The document is collaboratively built with the use of the Darwin-Information-Typing-Architecture (DITA) and has therefore a draft status concerning styles and layout. The necessary adaptations are currently also in a developmental stage.

Contents

1 Overview

Purpose

The howto explains how input validation can be switched on for classes and methods, using specific annotations of eHF.

Scope

You will see how you must annotate classes and methods in order to validate input. Further the necessary Spring configuration will be explained and last but not least you will get a short guidance to implement your own validator.

2 Input Validation via Annotations

Input validation is the key to “Defensive Programming” . Checking input of a method and to reject it in a controlled fashion will minimize the risk to run into unpredictable errors caused by malformed input data.

A “robust” application verifies input data at designated points in the control flow. As a “Rule of Thumb” an application should validate all input provided by external components. What an external component defines depends on the application, it could be:

- a jar file
- a different package
- a different class

3 Validation Enforcement

The eHF has a built in support for input validation based on JAVA 5 annotations:

- `com.icw.ehf.commons.metadata.validator.annotation.Validate`
- `com.icw.ehf.commons.metadata.validator.annotation.Param`
- `com.icw.ehf.commons.metadata.validator.annotation.SuppressValidation`

The input validation is enforced by intercepting a method call and if the `@Validate` annotation is found, the input parameter are validated against a pre-configured validator. The following code snippet shows how the annotations are used:

```
1   @Validate
2   public class ValidatingService implements ServiceInterface {
3
4   @SuppressValidation
5   public String serviceCall_I (String str) {...}
6
7   public String serviceCall_II (@Validate(
8       validator = "StringValidator",
9       params = {
10          @Param(key="KEY1", value="VALUE1"),
11          @Param(key="KEY2", value="VALUE2")
12      }
13  ) String token) {...}
14
15  public Object serviceCall_III (Object token) {...}
16  }
```

In line **1** the class is annotated with `@Validate`. This is necessary to enable input validation. Now all input of all methods will be validated. In line **4** the method `serviceCall_I` is explicitly marked not to validate any input by the annotation `@SuppressValidation`. The class wide input validation can be overwritten by using the `@Validate` annotation shown in lines **7**- **12**. The annotation variable `validator` is interpreted internally, e.g. a `java.lang.Class` can be used, or a Spring bean name.

The algorithm that fetches the `@Validate` annotation is described below in a "pseudo code" style:

```
METHOD build(Method method, Class baseClass) {
    IF (baseClass is annotated with Validate) AND
    IF (baseClass declares the method) THEN
    {
        IF (method is annotated with SuppressValidation) THEN
        {
            nothing to do RETURN
        }
        ELSE {
            IF NOT (Validate annotation support propagation) THEN
            {
                remember method and class
                RETURN
            }
        }
    }
    IF (get the superclass) AND
    IF NOT (superclass IS NOT java.lang.Object) THEN
    {
        CALL build(method, superclass)
    }
```

```
    FOREACH (interFace of baseClass) DO
    {
        CALL build(method, interFace)
    }
    RETURN
}
```

The input validation integrates with the Spring framework. The implementation provides adaptors for ordinary Spring AOP. This means that the annotated classes must be registered in the Spring context (see Configuration on page 6).

4 Writing Your Own Validator

Implementing your own validator is quite simple. You just have to provide an implementation of the interface `com.icw.ehf.commons.metadata.validator.Validator`. To use this validator in the context of *eHF*, it is necessary to supply an instance of this validator in the *Spring* configuration file (see Configuration on page 6).

5 Configuration

As already mentioned in Validation Enforcement on page 3 validation is enforced by intercepting a method call. The “Aspect Oriented Programming (AOP) Paradigm” is especially useful to introduce logic into business code which is not directly related to use cases. eHF provides a Spring integration to intercept method calls to “Spring Beans”. Input validation can be either enforced using the classical Spring AOP mechanisms, or an eHF proprietary way.

```
1   <bean id="entityValidatorRegistry" class="com.icw.ehf.commons.
2                                     metadata.validator.ValidatorRegistryImpl">
3       <property name="validatorRegistry" ref="entityValidatorRegistryList"/>
4   </bean>
5   <bean id="entityValidatorRegistryList" class="org.springframework.beans.
6                                           factory.config.ListFactoryBean">
7        <property name="sourceList">
8           <list>
9               <bean class="com.icw.ehf.commons.
10              metadata.validator.adapter.StringMetaDataValidatorAdapter">
11                  <property name="defaultParams">
12                       <map>
13                          <entry key="MIN_LEN" value="0"/>
14                          <entry key="MAX_LEN" value="255"/>
15                       </map>
16                  </property>
17              </bean>
18
19              <bean class="com.icw.ehf.commons.metadata.
20              validator.adapter.StringArrayMetaDataValidatorAdapter">
21                  <property name="defaultParams">
22                      <map>
23                          <entry key="MIN_LEN" value="0"/>
24                          <entry key="MAX_LEN" value="255"/>
25                      </map>
26                  </property>
27              </bean>
28
29              <bean class="com.icw.ehf.commons.metadata.
30              validator.adapter.StringCollectionMetaDataValidatorAdapter">
31                  <property name="defaultParams">
32                      <map>
33                          <entry key="MIN_LEN" value="0"/>
34                          <entry key="MAX_LEN" value="255"/>
35                      </map>
36                  </property>
37              </bean>
38
39              <bean class="com.icw.ehf.commons.
40              metadata.validator.adapter.ModuleMetaDataValidatorAdapter">
41                  <property name="metaDataValidator"
42                            ref="entityMetaDataValidator"/>
43              </bean>
44
45          </list>
46      </property>
47  </bean>
48
49  <ehf:input-validation module-name="entity"/>
```

In the lines **1**-**47** the validator registry is configured. This registry is used to resolve the validators which are included in the `@Validate` annotation. The resolution of validators is either explicit via a concrete name, or implicit via the class of the parameter of the validated

method. How the `validaor` field of the `@Validate` annotation is interpreted depends on the implementation of the registry (here it's the class name). Line **49** uses the `input-validation` tag of the `ehf-util` schema to turn on input validation. In the background it enables the Spring AOP method interception.

Note: The previous listing was taken from the eHF Entity module - hence the name of the two beans defined (at lines **1** and **5**) both begin with the word "entity". This will obviously be different for each module.

Extending Input Validation Configuration

By default within an eHF based module, the previous configuration is automatically generated for you in the module's `<modulename>-context.xml` Spring configuration file under `src/main/gen/META-INF`. This means that it is under the full control of the generator, and you can not directly modify it. You can however use the `inject` tag from the `ehf-commons` schema to modify the "`entityValidatorRegistry`" bean configuration and include your own `ValidatorAdapter`.

In your module's `<modulename>-custom-context.xml` Spring configuration file (usually found under `src/main/config`), you simply need to add the following configuration:

```
1   <ehf:inject target-ref="entityValidatorRegistry">
2       <ehf:propRef path="validatorRegistry" ref="myValidators" />
3   </ehf:inject>
4
5   <bean id="myValidators" parent="entityValidatorRegistryList"
6           class="org.springframework.beans.factory.config.ListFactoryBean">
7       <property name="sourceList">
8           <list merge="true">
9               <bean class="com.icw.ehf.MyCustomMetaDataValidatorAdapter"/>
10          </list>
11      </property>
12  </bean>
```

In lines **5** - **12** we define a new list of `ValidatorAdapters` (in this case it only contains one entry which is our new custom `ValidatorAdapter`, `MyCustomMetaDataValidatorAdapter`). However we use Spring's Collection Merging functionality to also include all entries from the original `entityValidatorRegistryList` as well.

In lines **1** - **3** we then inject this new list, `myValidators` into the `validatorRegistry` property of the original `entityValidatorRegistry` bean. This then means that our custom validator will also be used during input validation for the entity module.