ValidateThis Futures from cf.Objective()

Posted At : April 28, 2010 4:45 PM | Posted By : Bob Silverberg
Related Categories: ColdFusion, cfObjective, ValidateThis

I returned from cf.Objective() a few days ago and have been catching up on all sorts of things. I hope to post a review/discussion of the conference as a whole soon, but for now I just want to touch on some things that came up around ValidateThis.

I gave a presentation entitled *Easy and Flexible Validations for Objects* during which I described what ValidateThis is, and showed how easy it is to work with. I demonstrated that you can add client-side validations to a form with a single line of code, and can perform server-side validations with just a few lines. I also discussed the different types of validation scenarios that VT is designed to address. There were some questions and suggestions which have prompted me to think of future enhancements to the framework, so I wanted to discuss them here.

Internationalization for Validation Rules

The framework currently supports i18n for validation failure messages. If you have resource bundles which include failure messages for different languages the framework will make use of those and return messages in the language of your choosing. This works on both the client and server side. A question was asked, however, about supporting different validation rules for different locales. For example, the format of a phone number may be different based on the locale of the user. Wouldn't it be nice to be able to create a validation rule for *phoneNumber* which could be intelligent enough to deal with those differences? There was some immediate discussion of using *contexts* to deal with this issue, but really that is not a good fit. Contexts are more about the business context in which an object is being used (e.g., registration vs. login vs. checkout, etc.). The context would be the same regardless of the language.

Upon further reflection I realized that one could re-purpose contexts for this, by defining separate contexts for each locale (e.g., english_register, french_register, etc.), and that would work with VT as it currently stands, but it really isn't a great idea. I think what would work better would be to allow the framework to be more "locale-aware", for lack of a better term, so that one could define rules that apply to a given locale. In addition, one could create locale-aware validation types. For example one could create a *phoneNumber* validation type which could accept a locale and would then know how to validate the phone number for the various locales supported.

Both of those scenarios seem like good ideas to me and I plan on pursuing them in a future release.

Metadata-driven Conditions

I demonstrated the *condition* metadata element which allows a developer to define arbitrary CFML and/or JavaScript that will be run to determine whether or not a rule should be enforced. Here's an example:

```
<conditions>

 <condition name="MustLikeSomething"

  serverTest="getLikeCheese() EQ 0 AND getLikeChocolate() EQ 0"

  clientTest="$(&quot;[name='LikeCheese']&quot;).getValue() == 0

   &amp;&amp; $(&quot;[name='LikeChocolate']&quot;).getValue() == 0;" />

</conditions>
```

These conditions exist to provide maximum flexibility and I'm pretty happy with the way they work right now. It was suggested that perhaps these rules could be defined using metadata, rather than actual code, which would require some sort of parser and translation layer. I can see the benefit in this, as it would allow a user to define a condition once which could then be used for both client-side and server-side validations (which fits it well with the goals of the framework), but I only see this working in a limited number of scenarios.

For example, it wouldn't be a big deal to implement this to work with the example above, which is really just a simple condition based on the values of two properties, but what about a condition that is based on a more complex expression? I know that there will be situations in which defining these as metadata, rather than as actual code, simply will not be possible, and I am therefore unsure of the value of adding this to the framework. Of course, if anyone is interested in having this behaviour and is willing to write the code they are free to contribute it.

One thought I did have about the condition element, as a result of this conversation, was that rather than placing the actual code inside the xml file, one could point to a method in the object (just like one does with the *custom* validation type). This could use a method name on the server side and a JavaScript function name on the client side. I'm not sure how keen I am on the latter, as that would require more coupling between the view and the validation metadata, but I think it's definitely worth considering.

Support for Annotation-based Metadata

Currently there are two ways of defining your validation rules: via an xml file, or using the *addRule()* method provided by the framework. I still prefer the xml approach, but am aware of the fact that some do not, and am also aware of the fact that using annotations within objects is a common approach in other programming languages. Supporting annotations in cfcs is actually something that an existing contributor to the framework, Adam Drew, has been working on for awhile, and I regret to say that I've just been too busy to take a look at his work. I *would* like to see this added to the framework, so I'm going to make a concerted effort to work with Adam to look at his contributions. I'd welcome anyone else interested in being involved in that conversation to join us - more on that below.

Support for Validating a Structure

Currently the framework only works with objects. You take an object, populate it and then validate it using VT. I have had discussions with a few people who follow a different workflow than that. They prefer to validate their data *before* it gets into the object. I can understand that approach and see pros and cons with it. I personally like the fact that the data that I'm validating goes through the logic of the setters and getters before being validated, but I can totally understand why some people do it otherwise. I'm therefore considering adding support to the framework that would allow a developer to pass a structure (e.g., the form scope) into the framework to be validated. This would potentially allow folks to do validation both before *and* after population if they wished to do so.

A side benefit of this is that it would open up the framework to be used by anyone in any project. It would no longer be "just for objects".

A Hibernate Interceptor

Mark Mandel and I discussed the possibility of using a Hibernate interceptor (also known as an event handler in CF-speak) to automatically validate objects before they are persisted. This could act as a sort of "last line of defence" to prevent invalid data from getting into your database. I can see some issues with this, including the extra overhead of an additional validation check and also how to deal with contexts, but it's definitely something I think is worth pursuing.

Some Ajaxy Thing

OK, I feel terrible about this, but I remember talking to someone about something related to ajax and the framework, and he specifically mentioned being interested in contributing. Other than that it's a bit of a blur, so I wanted to reach out and say that although I seem to have forgotten the content of our conversation I am very interested in continuing it. At least I'm honest, right?

A ValidateThis Dev Group

I've created a new Google group, ValidateThis-dev which is intended to be a list for discussing future development of the framework. The existing group, ValidateThis will remain as the main location for discussions of how to use the framework, and is also appropriate for enhancement requests, whereas the ValidateThis-dev group will be for discussing the design and coding of the framework itself. This new group is a public group, so anyone is free to join, but I'd be particularly keen on having anyone who is interested in contributing, or even just sharing their ideas about how the framework can be improved, as a member of the group.

I see interesting times and cool things ahead for ValidateThis in 2010 and beyond!