Tweet Place this tag where you want the share button to render.
Place this tag after the last share tag.

"Boxed" up, RESTful Goodness


Jon Clausen
Mar 15, 2017

Leveraging the Coldbox REST Application Template

In February, we quietly released an updated version our our Coldbox REST Application Template. The enhancement aimed to consolidate a number of best practices and enhancements in RESTful API development that were worthy of being included in the template.

If you're not familiar with using this template, it's easy enough to take for a spin with `box coldbox create app name=myAppName skeleton=rest`. Along with more expressive syntax for handling status codes, the Response object has been improved to deliver more consistent response formats, which eliminate some of the unecessary verbosity in the previous version. For example, previously, the Response object would return the error information, even if no error was found, in addition to a secondary error code. When using RESTful principles, the HTTP response code *is intended* to be your error code. We've eliminated the previous convention, in favor of allowing REST to work as it should.

The BaseHandler now maintains a “static” construct, which allows you to leverage RESTful HTTP response codes by name:

```
STATUS = {
   "CREATED"           : 201,
   "ACCEPTED"          : 202,
   "SUCCESS"           : 200,
   "NO_CONTENT"        : 204,
   "RESET"             : 205,
   "PARTIAL_CONTENT"   : 206,
   "BAD_REQUEST"       : 400,
   "NOT_AUTHORIZED"    : 401,
   "NOT_FOUND"         : 404,
   "NOT_ALLOWED"       : 405,
   "NOT_ACCEPTABLE"    : 406,
   "TOO_MANY_REQUESTS" : 429,
   "EXPECTATION_FAILED": 417,
   "INTERNAL_ERROR"    : 500,
   "NOT_IMPLEMENTED"   : 501
};;
```

For example, upon creating a new entity (from a POST HTTP method) we would be able to specify the response code with the following:

```
prc.response.setData( myCreatedEntity.getMemento() );
prc.response.setStatusCode( STATUS.CREATED );
```

In addition, new utility method have been added to allow for solving common issues encountered within the flow of a request:

- `routeNotFound` - Can be used to deliver a 404 response when either a route or requested entity is not found

- `onExpectationFailed` - Can be called when an expectation of a request fails, such as invalid parameters, expected headers, etc.
- `onAuthorizationFailure` - Can be called to send a NOT Authorized status code and message. The method also allows an `abort` flag which, when passed, will perform a hard stop within the request to prevent further execution of the remainder of the request (use wisely, and as a last resort)

With the new BaseHandler, implementation, it is easier than ever to quickly scaffold and implement RESTful handlers. Since the BaseHandler's `aroundHandler` method handles error emission and encapsulates it in to a consistent response, it's easier than ever to write your handler methods. For example, a method to add a new entity (let's make this our `box` API handler) might be as simple as:

```
/**
* Adds a new box
* @event    the RequestContext
* @rc       the RequestContext public collection
* @prc      the RequestContext private collection
**/
public function add( event, rc, prc ){

   //Let's make our new `box` an ORM entity...
   var newBox = getInstance( "box@ortus" ).new(
       properties = rc,
       composeRelationships=true
   );

   if( newBox.isValid() ){

       newBox.save();
       //getMemento() is an Ortus convention for returning an ORM entity as a structural representation
       prc.response.setData( newBox.getMemento() );
       //Our status code will be returned RESTfully
       prc.response.setStatusCode( STATUS.CREATED );

   } else {

       prc.response.setMessages( newBox.getValidationResults().getAllErrors() );
       prc.response.setStatusCode( STATUS.EXPECTATION_FAILED );

   }

}
```

Feel free to take the updated app skeleton for a spin, when starting your new API projects. As a side-benefit, the conventions in the `Response` model can be duplicated and, apart from the HTTP-specific properties, used as a template to provide consistent service layer responses, as well (e.g. `ServiceResponse`).

For a more in-depth look at leveraging the REST app skeleton to develop full-featured API's, checkout the Building RESTFul Services workshop at the 2017 Into the Box conference in Houston, Texas. Topics for the workshop Include:

- REST Core Principles and Implementation Standards
- HTTP Dialect patterns
- RESTful Response patterns
- Developing and leveraging microservices with Coldbox
- API Security strategies and implementation
- API Documentation and Modeling
- Scaffolding and building applications and modules
- API versioning and release strategies
- Hands-on implementation of a working RESTful API

We hope to see you in Houston next month! Happy coding!

end main div -

---

www.ortussolutions.com