BLAZEDS

BlazeDS Plugin - Reference Documentation

Authors: Sebastien Arbogast, Burt Beckwith

Version: 2.0

Table of Contents

1 Introduction to the BlazeDS Plugin

The BlazeDS plugin makes it easier to build Grails-powered Rich Internet Applications using the Adobe BlazeDS remoting and messaging framework, as well as the Spring Security Core plugin to secure your application. The plugin also adds support for lazy loading to make it easier to work with GORM and Hibernate.

1.1 Getting Started

The first step is installing the plugin:

```
grails install-plugin blazeds
```

This will transtitively install the Spring Security Core, and Spring Security ACL plugins. Refer to the documentation for these plugins for configuration options. At a minimum you'll need to run the `s2-quickstart` script to configure Spring Security Core, e.g.

```
grails s2-quickstart com.yourcompany.yourapp User Role
```

The Spring Security ACL plugin has no required initialization steps.

2 Configuration

There are a few configuration options for the plugin.

Property	Default	Meaning
grails.plugin.blazeds.defaultMessageChannels	none	a comma-delimited String `'my-streaming-amf,`
grails.plugin.blazeds.defaultRemoteChannels	none	a comma-delimited String `'my-amf'`
grails.plugin.blazeds.converterNames	[HibernateProxyConverter, PersistentCollectionConverterFactory, JpaNumericAutogeneratedIdConverter]	a list of class names of con clients
grails.plugin.blazeds.proxyIgnoreProperties	'class', 'metaClass', 'hibernateLazyInitializer' plus all event names from `grails.persistence.Event`	a list of property names to
grails.plugin.blazeds.disableOpenSessionInView	`false`	set to `true` to disable the

3 General Usage

Runtime configuration
The plugin manages the configuration of BlazeDS. This includes configuring `flex.messaging.HttpFlexSession` as a listener in web.xml along with the `flex.rds.server.servlet.FrontEndServlet` servlet for use by Flash Builder. It also configures `/messagebroker/*` URLs to be intercepted and handled as BlazeDS requests.

Data serialization
There are a few options for configuring how class instances are serialized to clients. All of the standard BlazeDS rules apply, but having Hibernate in the mix complicates things because of lazy-loaded collections and many-to-one references.
By default an Open Session in View filter is registered unless this is disabled by setting `grails.plugin.blazeds.disableOpenSessionInView` to `true` in `Config.groovy`. This acts like the analagous interceptor that Grails registers for controller requests, starting and binding a thread-local Hibernate Session at the beginning of a BlazeDS request and flushing and closing it after the response is rendered. This allows you to not have to worry about lazy loading exceptions when using domain classes that refer to other lazy-loaded classes.
There's a significant performance concern here though since you can potentially render a very large object graph to the client when only a small amount of data is needed. One option is to use Data Transfer Objects (DTOs) that you create using data from persistent domain classes. This allow you to control exactly what data gets sent to your clients.
Another option is to use the data converters that the plugin registers for you by default. These include HibernateProxyConverter which sends a `null` value for any uninitialized many-to-one reference, and PersistentCollectionConverterFactory which does the same for uninitialized collections. You can explicitly initialize any required many-to-one references or collections using the org.hibernate.Hibernate.initialize() method.
You can remove any of the configured converters (or add your own) by changing the value of `grails.plugin.blazeds.converterNames` in `Config.groovy`, which is ['org.springframework.flex.core.io.HibernateProxyConverter, 'org.springframework'flex.core.io.PersistentCollectionConverterFactory', 'org.springframework.flex.core.io.JpaNumericAutogeneratedIdConverter'] by default.

Grails services as remote destinations
It's simple to access a Grails service as a remote service from Flex - just annotate the service with the `org.springframework.flex.remoting.RemotingDestination` annotation. BlazeDS handles invoking method calls and marshalling parameters and return values for you.
One thing that's important to note is that although there is a `services-config.xml` configuration file (created when the BlazeDS plugin is installed) you probably won't need to make many changes there. In typical BlazeDS applications you would register remote services using XML, but Grails services are well suited as candidates for remote services.
The plugin also handles service reloading for you and re-registers recompiled annotated services as remoting destinations.

Manual configuration
You're not limited to using Grails services as the server-side implementation of remote services. You can use any class like you would in a non-Grails application, and the best place to configure this is in `grails-app/conf/spring/resources.groovy` using the BeanBuilder syntax equivalent of the Spring Flex XML configuration.
For example, this resources.groovy file contains four messaging destinations and two remoting destinations. Remoting destinations are configured like any other Spring bean, with the addition of a `flex.'remoting-destination'()` child element, optionally with configuration options. See the Spring Flex documentation for more information on what's available.

```
import flex.management.jmx.MBeanServerGateway
beans = {
   xmlns flex: 'http://www.springframework.org/schema/flex'
   flex.'message-destination'(id: 'chat')
   flex.'message-destination'(id: 'secured-chat', 'send-security-constraint': 'trusted')
   flex.'message-destination'(id: 'simple-feed')
   flex.'message-destination'(id: 'market-feed', 'allow-subtopics': true, 'subtopic-separato
   securityHelper(Security3Helper) {
      flex.'remoting-destination'()
   }
   RuntimeManagement(MBeanServerGateway) {
      flex.'remoting-destination'(channels: 'my-amf, my-secure-amf')
   }
}
```