ReloadableServices

File(s): *conf/services.xml, conf/services.properties*

Format: Native Spring

Legacy V2 File(s): *conf/services.xml*

Overview

The *services.xml* file is used to specify many of the other configuration files (or more generally, Spring Resources) to load to configure various important services within the IdP. The *services.properties* file provides a less granular way to identify the Spring beans containing the lists of resources, and also controls the dynamic reloading behavior of those services.

You might modify these files to:

- change the resources used, or more commonly **add** additional resources to supplement built-in defaults
- configure more specialized approaches such as Subversion resources or remote HTTP resources
- control how often to check for changes and reload configurations, if at all

The *services.xml* file contains a series of "list" beans that specify the Spring Resources to load into various services. The lists are named with specific bean IDs (see below) that direct the resources into the various services. If you wish to supply your own resource lists without modifying the delivered lists, you may control the bean IDs used by modifying *services.properties*.

Do not remove any of the beans from *services.xml* unless you alter a corresponding property in *services.properties* to direct the service to a different resource list bean, or the IdP will fail to initialize with an error referencing the missing bean.

Fail Fast

The fail-fast behavior, which can be adjusted by service to a limited degree, determines whether the IdP webapp context will initialize and serve clients if a particular service fails to initialize successfully. The point of this mechanism is to allow you to decide for yourself when problems should be discovered and how serious they should be treated. While it isn't the default, you may find it simpler when first starting out to enable fail-fast behavior globally while you work through mistakes.

Whether a given service succeeds or fails is ultimately an internal consideration, but generally we're talking about whether its configuration is valid and whether its pieces and parts themselves are considered to be successfully initialized. Often there may be individual fail-fast settings applying at the micro level that in turn dictate whether the surrounding service starts (and which then determines the overall result of the IdP startup process based on the service's fail-fast behavior). On top of that, some of the subsystems cause ripple effects when they fail that may make it impossible to really achieve non-fail-fast behavior in some cases.

As of V3.2, there are two levels of fail-fast properties that control service behavior (and described below). A global property called **idp.service.failFast** can be used to toggle all services to fail-fast at once (since the default is false for most, but true for a couple). In addition, or instead, you can control the behavior of specific services with properties specific to each service. The individual properties override the global setting, so you can mix and match.

Reloading Services

In addition to the "checkInterval" properties listed below to automatically reload services, you may reload a service at any time using the *reload-service* command line utility and the service ID. The service IDs are shown below in the Beans table (excluding the logging service, which is "shibboleth.LoggingService").

```
$ ./reload-service.sh -id shibboleth.AttributeResolverService
```

Alternatively, you may reload services by issuing a GET request to the following URL, assuming your client has access as defined in AccessControlConfiguration.

```
GET http(s)://[idp-base-url]/idp/profile/admin/reload-service?id=shibboleth.LoggingService
```

Reference

Beans

Beans defined in *services.xml* follow:

Bean ID	Type	Function	Reloadable Service ID
shibboleth.RelyingPartyResolverResources	java.util.List<Resource>	RelyingPartyConfiguration resources for a new or migrated installation.	shibboleth.RelyingPartyResolverService
shibboleth.LegacyRelyingPartyResolverResources	java.util.List<Resource>	RelyingPartyConfiguration using a deprecated V2 relying-party.xml file.	shibboleth.RelyingPartyResolverService
shibboleth.MetadataResolverResources	java.util.List<Resource>	MetadataConfiguration resources.	shibboleth.MetadataResolverService
shibboleth.AttributeResolverResources	java.util.List<Resource>	AttributeResolverConfiguration resources.	shibboleth.AttributeResolverService
shibboleth.AttributeFilterResources	java.util.List<Resource>	AttributeFilterConfiguration resources.	shibboleth.AttributeFilterService
shibboleth.NameIdentifierGenerationResources	java.util.List<Resource>	NameIDGenerationConfiguration resources.	shibboleth.NameIdentifierGenerationService
shibboleth.AccessControlResources	java.util.List<Resource>	AccessControlConfiguration resources.	shibboleth.ReloadableAccessControlService
shibboleth.MessageSourceResources	java.util.List<Resource>	Internationalizable user interface messages.	N/A
shibboleth.CASServiceRegistryResources 3.2	java.util.List<Resource>	Resources containing ServiceRegistry beans to be reloaded.	shibboleth.ReloadableCASServiceRegistry

Properties

Properties defined in *services.properties* follow:

Property	Type	Default	Function
`idp.service.failFast` 3.2	Boolean	false	Set default fail-fast behavior of all services unless overridden by service
`idp.service.logging.resource`	Resource path	%{idp.home}/conf/logback.xml	Logging configuration resource to use (the reloadable service ID is "shibboleth.LoggingService")
`idp.service.logging.failFast`	Boolean	true	Fail at startup if logging configuration is invalid
`idp.service.logging.checkInterval`	Duration	0	Time to notice changes to logging configuration and reload service. A value of 0 indicates that the logging configuration never reloads
`idp.service.relyingparty.resources`	Bean ID	shibboleth.RelyingPartyResolverResources	Name of Spring bean identifying resources to use for RelyingPartyConfiguration service
`idp.service.relyingparty.failFast`	Boolean	false	Fail at startup if RelyingPartyConfiguration is invalid
`idp.service.relyingparty.checkInterval`	Duration	0	Time to notice changes to RelyingPartyConfiguration configuration and reload service A value of 0 indicates that the relying party configuration never reloads
`idp.service.metadata.resources`	Bean ID	shibboleth.MetadataResolverResources	Name of Spring bean identifying resources to use for MetadataConfiguration service
`idp.service.metadata.failFast`	Boolean	false	Fail at startup if MetadataConfiguration is invalid
`idp.service.metadata.checkInterval`	Duration	0	Time to notice changes to MetadataConfiguration configuration and reload service A value of 0 indicates that the metadata configuration never reloads
`idp.service.attribute.resolver.resources`	Bean ID	shibboleth.AttributeResolverResources	Name of Spring bean identifying resources to use for AttributeResolverConfiguration service
`idp.service.attribute.resolver.failFast`	Boolean	false	Fail at startup if AttributeResolverConfiguration is invalid
`idp.service.attribute.resolver.checkInterval`	Duration	0	Time to notice changes to AttributeResolverConfiguration configuration and reload service A value of 0 indicates that the attribute resolver configuration never reloads
`idp.service.attribute.resolver.maskFailures` 3.1	Boolean	true	Whether attribute resolution failure should silently produce no attributes (the V2 behavior), or cause an overall profile request failure event
`idp.service.attribute.resolver.stripNulls` 3.4	Boolean	false	Whether null values should be stripped from the results of the attribute resolution (prior to filtering and encoding)

`idp.service.attribute.filter.resources`	Bean ID	shibboleth.AttributeFilterResources	Name of Spring bean identifying resources to use for AttributeFilterConfiguration service
`idp.service.attribute.filter.failFast`	Boolean	false	Fail at startup if AttributeFilterConfiguration is invalid
`idp.service.attribute.filter.checkInterval`	Duration	0	Time to notice changes to AttributeFilterConfiguration configuration and reload service A value of 0 indicates that the attribute filter configuration never reloads
`idp.service.attribute.filter.maskFailures` 3.1	Boolean	true	Whether attribute filtering failure should silently produce no attributes (the V2 behavior), or cause an overall profile request failure event
`idp.service.nameidGeneration.resources`	Bean ID	shibboleth.NameIdentifierGenerationResources	Name of Spring bean identifying resources to use for NameIDGenerationConfiguration service
`idp.service.nameidGeneration.failFast`	Boolean	false	Fail at startup if NameIDGenerationConfiguration is invalid
`idp.service.nameidGeneration.checkInterval`	Duration	0	Time to notice changes to NameIDGenerationConfiguration configuration and reload service
`idp.service.access.resources`	Bean ID	shibboleth.AccessControlResources	Name of Spring bean identifying resources to use for AccessControlConfiguration service
`idp.service.access.failFast`	Boolean	true	Fail at startup if AccessControlConfiguration is invalid
`idp.service.access.checkInterval`	Duration	0	Time to notice changes to AccessControlConfiguration configuration and reload service
`idp.service.cas.registry.resources` 3.2	Bean ID	shibboleth.CASServiceRegistryResources	Name of Spring bean identifying resources to use for CAS service registry configuration
`idp.service.cas.registry.failFast` 3.2	Boolean	false	Fail at startup if CAS service registry configuration is invalid
`idp.service.cas.registry.checkInterval` 3.2	Duration	0	Time to notice CAS service registry configuration changes and reload service
`idp.message.resources`	Bean ID	shibboleth.MessageSourceResources	Name of Spring bean identifying Spring message property resources
`idp.message.cacheSeconds`	Integer	300	Seconds between reloads of message property resources

V2 Compatibility

A similar function was performed by the *services.xml* file in V2, but in V3 this file is now a native Spring bean file and the older services XML schema is **not** supported or used.

Additionally, the V2 ResourceFilter feature is also not supported, so if you're using the Property Replacement Filter feature, you will need to adjust at least some of your configuration files. In most cases (but not always) you can leverage the Spring property replacement mechanism by changing the syntax slightly.

Advanced Notes

You can use any kind of Resource supported by Spring, along with additional custom resource types provided with the IdP for handling Subversion and HTTP resources.