Embedding Apache Directory Server into Applications

By Alex Karasulu

Introduction: Coverage

- Core: Server Architecture
- Core Configuration Interfaces
- Schema Customizations
- Startup/Shutdown Sequence
- Enabling Protocol Services
- Testing
- Advanced Configuration
  - Hot Reconfiguration
  - Introducing New Interceptors (Aspects)

Experience and Expectations

What is your background experience and expectations from this session?

- LDAP Knowledge
- JNDI API Familiarity
- What directory servers have you used before?
- Have you tried ApacheDS?

Server Core Architecture

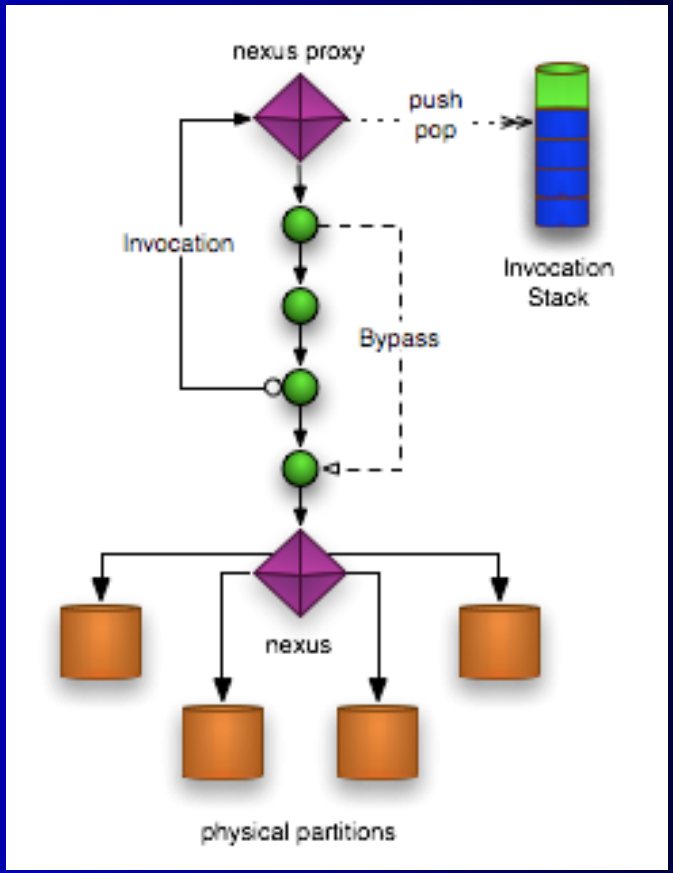
- What is the core?
- ApacheDS LDAP JNDI Provider
- Interceptor Mechanism
- Interceptors
- Partition Nexus
- Partitions

Core: What is it?

The ApacheDS core is a JNDI provider that manages a local hierarchical store of Attributes objects, based on the LDAP namespace.

Core: What's in there?

- Nexus Singleton
- Partitions
- Nexus Proxies
- Interceptors
- InvocationStack
- JNDI Interfaces



Core: Partitions

- Partitions store entries (javax.naming.directory.Attributes).
- Exposes CRUD operations mapping to LDAP operations.
- Multiple heterogeneous partitions may exist.
- Partitions store disconnected entry trees.
- Partitions store entries below some naming context called the partition suffix. The names of all entries within a partition end in the suffix.
- Partitions are kept as simple as possible: they only need to be concerned with entry access and storage.

Core: Partition Nexus

- Presently the nexus is a singleton.
- It is a partition that does not store entries.
- Calls are delegated to other partitions.
- Call routing is based on namespace.
- Several partitions may be “attached”.
- Custom implementations can be attached.
- Stores immutable RootDSE in memory.
- Has ops to add/remove/list partitions.

Core: System Partition

- Always present with suffix ou=system.
- Provides storage for configuration info.
- Implementation based on JDBM B+Trees.
- Cannot be detached from the nexus.

Core: JNDI Provider

- JNDI is the access API used to hide internals.
- Nexus, partitions etc. are all hidden.
- JNDI Contexts call internals to perform operations on Attributes objects in partitions.
- Feels like LDAP but it's not: just the namespace.
- Relative Name arguments to Contexts are transformed into absolute distinguished names.
- Contexts perform absolute operations on internals to satisfy JNDI calls.

Core: Nexus Proxy

- Nexus proxy objects do as their name suggests, they forward calls to the nexus.
- Forwarded calls are intercepted to introduce additional services.
- Each Context instantiated has a proxy.
- Contexts call proxies as if it is the nexus.
- Proxies start the interception mechanism.

Core: Interceptor Mechanism

- Interceptors trap Context => Nexus calls.
- Several interceptors in a chain trap calls.
- Each interceptor can:
  - change arguments,
  - alter return values,
  - bypass calls to the target method on the nexus,
  - transform, re-throw or consume exceptions.
- Interceptors introduce and centralize aspects.
- Interceptors are very powerful and dangerous.

Core: Interceptor Mechanism II

- Some aspects implemented with interceptors are:
  - Scheme Checking
  - Access Control Checks
  - Operational Attribute Maintenance & Filtering
  - Name Normalization
  - Collective Attribute Injection
  - Subentry Filtering
  - Authentication
  - Exception Handling
  - JNDI Event Delivery

Core: InterceptorChain

- Contains instances of all interceptors.
- A single chain is present.
- Invokes each interceptor in order.
- Last hard-coded interceptor calls nexus.
- Performs bypass instructions.
- Bypass feature prevents infinite recursion.
- Interceptors and the InterceptorChain work in conjunction with the InvocationStack.

Core: InvocationStack

- Contains a Stack of Invocation objects for each thread.
- Why?
  - Interceptors access Invocation objects for information
  - Interceptors call proxy methods to operate on the DIT
  - Triggers may invoke stored procedures that use JNDI
- Invocation objects contain:
  - the JNDI context making the nexus method call
  - the name of the method invoked
  - the values of invoked method arguments
  - the proxy object for the context

Core: What maintains the stack?

- Proxies create Invocation objects on each call.
- Before invoking the InterceptorChain the proxy pushes the Invocation onto the stack.
- After the InterceptorChain returns or an exception is thrown, the Invocation is popped off of the stack.

Core: Summary

The best way to summarize the architecture is to exercise a walk through a stack trace on an operation against a JNDI Context.

Intermission

Next: Core Configuration Interfaces

Config: JNDI Environment

- Core uses JNDI for configuration.
- Some standard JNDI keys are also supported as seen on the next slide …

Configuration: Standard JNDI Keys

- Context.PROVIDER_URL
  - Since provider is local just specify a DN
- Context.INITIAL_CONTEXT_FACTORY
  - org.apache.ldap.server.jndi.CoreContextFactory
  - org.apache.ldap.server.jndi.ServerContextFactory
- Context.REFERRAL
- Context.SECURITY_AUTHENTICATION
- Context.SECURITY_CREDENTIALS
- Context.SECURITY_PRINCIPAL
- Context.STATE_FACTORIES
- Context.OBJECT_FACTORIES

Configuration: LDAP Specific Keys

- java.naming.ldap.attributes.binary
- java.naming.ldap.control.connect
- java.naming.ldap.deleteRDN
- java.naming.ldap.derefAliases
- java.naming.ldap.ref.separator
- java.naming.ldap.referral.limit
- java.naming.ldap.typesOnly
- java.naming.security.sasl.authorizationId
- java.naming.security.sasl.realm
- java.naming.security.sasl.callback

Configuration: Provider Specific Keys

- An additional key/value pair is needed:

  org.apache.ldap.server.configuration.Configuration.JNDI_KEY

- Value is a subclass of:

  org.apache.ldap.server.configuration.Configuration

Configuration: StartupConfiguration

- A subclass of Configuration used to start the core.
- Contains additional settings as beans or bean properties:
  - workingDirectory (File)
  - allowAnonymousAccess (boolean)
  - accessControlEnabled (boolean)
  - authenticationConfigurations (Set <AuthenticationConfiguration>)
  - interceptorConfigurations (List <InterceptorConfiguration>)
  - bootstrapSchemas (Set <BootstrapSchema>)
  - contextPartitionConfigurations (Set <DirectoryPartitionConfiguration>)
  - testEntries (List <Attributes>)
- Default constructor configures all subordinate beans.
- Mutable version available for tweaking settings.
- First time use in InitialDirContext starts up the core.

Configuration: ShutdownConfiguration

- Shuts down the core when included in environment of new InitialContext.
- Returned Context, DeadContext, is useless.
- Automatically calls synch() on all partitions to push caches to disk.
- Startup automatically registers a JVM shutdown hook to shutdown the core with a ShutdownConfiguration instruction.

Configuration: SynchConfiguration

- Forces the Nexus to call synch() on all partitions to flush caches to disk.
- Without calling synch() caches may never flush depending on the partition implementation.

Configuration: main() with defaults.

- Let’s run the example application

Configuration: Custom Partitions

- A MutableStartupConfiguration is used.
- We prepare and populate a set with DirectoryPartitionConfigurations.
- Let’s run the example application with the custom partition.

Configuration: Default Schema

- By default ApacheDS comes with a standard set of published schema even though not all are setup by default.
- Default Schema:
  - CoreSchema (Highly Recommended)
  - CosineSchema
  - ApacheSchema (Required)
  - InetorgpersonSchema
  - JavaSchema
  - SystemSchema (Required)
  - CollectiveSchema

Configuration: Additional Schema

- Additional Schema:
  - ApachednsSchema
  - AutofsSchema
  - CorbaSchema
  - DhcpSchema
  - Krb5kdcSchema
  - MiscSchema
  - NisSchema
  - SambaSchema

Configuration: Additional Schema

- Just populate a set with all the schema you would like to use.
- Call setBootstrapSchemas() with your set.
- WARNING: Schema depend on other schema for syntaxes, matchingRules, attributeTypes and other objectClasses.
- Make sure the set of schema include dependent schema otherwise ApacheDS will let you know on startup when dependencies cannot be resolved.

Configuration: Using Custom Schema

- BootstrapSchema classes are generated from OpenLDAP schema files.
- The directory-maven-plugin is used to generate the source files.
- We recommend creating a maven subproject to generate these sources, compile them and to produce the jar.
- The jar can later be incorporated into the path used to startup an application that embeds ApacheDS and uses the custom schema.

Configuration: Custom Schema

- Copy the custom schema project.
- Add your schema file to ${basedir}/src/main/schema.
- Set properties in project.properties for the schema and its dependencies.
- Now run maven directory:schema.
- Check ${basedir}/target/schema for the generated schema files.
- You can now produce the jar by running maven jar.
- Let’s demonstrate a custom application that now uses this custom schema.

Protocols: Enabling Services

- This goes beyond just using the core.
- You can encapsulate yourself from the details by using the ServerContextFactory and the ServerStartupConfiguration that are part of apacheds/main.
- This InitialContextFactory implementation uses:
  - MINA: Multipurpose Infrastructure for Network Applications (ApacheDS Networking Layer)
  - Protocol Providers for LDAP, Kerberos, Change Password, NTP and DNS.

Protocols: Start what you like!

- By default only the LDAP protocol provider is started by the ServerStartupConfiguration.
- Other protocols can be configured to start up as well using a MutableServerStartupConfiguration.
- All protocols backend their content within the partitions of the core.
- Each protocol may have its own custom configuration parameters.

Protocols: LDAP Parameters

- LDAP configuration parameters are exposed as bean properties on the ServerStartupConfiguration:
  - enableNetworking (boolean:true)
  - ldapPort (int:389)
  - ldapsPort (int:636)

Protocols: LDAP Example

- Let's build a custom application that embeds ApacheDS and exposes LDAP access on port 10389.
- See projector for code.
- Let's run the application.
- Let's connect to the server with an LDAP Browser.

Protocols: Krb5 Parameters

- Presently most Kerberos configuration parameters are expected as env properties.
- The one configuration property on the ServerStartupConfiguration is a boolean to toggle it on or off.
- The list of configuration parameters are listed on the next slide.

Protocols: Krb5 Parameters II

- kdc.principal
- kdc.primary.realm
- kdc.default.port
- kdc.entry.basedn
- kdc.encryption.types
- kdc.allowable.clockskew
- kdc.buffer.size
- kdc.pa.enc.timestamp.required
- tgs.maximum.ticket.lifetime
- tgs.maximum.renewable.lifetime
- tgs.empty.addresses.allowed
- tgs.forwardable.allowed
- tgs.proxiable.allowed
- tgs.postdate.allowed
- tgs.renewable.allowed

Protocols: Kerberos Example

- Let build an application which embeds ApacheDS and only exposes the Kerberos protocol while disabling LDAP access.
- Let’s run the example application.
- Connect to it with kinit.

Testing: AbstractTestCase

- Within the core there is an AbstractTestCase class which extends JUnit TestCase.
- This can be used to conduct tests on your application which embeds ApacheDS with a custom configuration.

Testing: AbstractServerTest

- If you would like to test your application's configuration with networking protocols enabled then you can use this alternative JUnit TestCase.
- You can find this test case within the main project's jar.

Advanced: Possibilities

- ApacheDS partitions can be added and removed while the server is running.
- Aspects are powerful features. New aspects can be added to ApacheDS using custom Interceptors.

Advanced: Adding Partitions to a Live Instance

- An AddDirectoryPartitionConfiguration is used to add partitions to the core while it is running.
- As with other configuration objects, it is passed into the core using a new InitialContext.
- The environment must have a Configure.JNDI_KEY set to an instance of a AddDirectoryPartitionConfiguration.

Advanced: Removing Partitions From a Live Instance

- RemoveDirectoryPartitionConfiguration
- Apply the same pattern!

Advanced: Add Remove Example Application

- Let’s look at the code.
- Let’s run the application.

Advanced: Custom Interceptors

- Be forewarned Interceptors are very powerful and so can do very good things, however they can also give you a bad day if not implemented correctly.
- Introduce new aspects into the server by using custom interceptors.

Advanced: Example Interceptor for Managing Constraints

- In this example we build a custom Interceptor and add it to the core.
- The Interceptor maintains a changelog in LDIF file format.
- Let's look at the code.
- Let's run the application and test again.

Summary

- We’ve learned quite a few things:
  - ApacheDS Architecture
  - Configuring and embedding ApacheDS
  - Customizing configuration for new schemas, interceptors and partitions.
  - Enabling ApacheDS protocol providers.

Where to Get More Information

- The example projects for this tutorial are available on a public svn server here:
  https://svn.safehaus.org/repos/sandbox/apachecon
- Here you can also find the templates we used as well as this power point presentation.
- Other related sessions:
  - TU14 Introduction To MINA
  - TU23 Secure Single Sign On with Apache Directory and Apache Kerberos
- List books, articles:
  - http://www.screaming-penguin.com/main.php?storyid=4972