

The banner features a green background with a grid pattern on the left and a blue sky with clouds on the right. The text "Spring Framework" is written in a white serif font, with a small yellow leaf icon above the letter 'i' in "Spring".

Spring Framework

Spring LDAP - Java LDAP Programming Made Simple

Mattias Arthursson & Ulrik Sandberg

jayway

About the Speakers

- :: Consultants for Jayway, Sweden
- :: Founders/Leads of Spring LDAP
- :: Speakers on Spring Experience, Spring ONE, etc.

Jayway

Agenda

- :: Template Programming and `LdapTemplate`
- :: Additional Features in Spring LDAP
- :: LDAP Transactions

jayway

Template Programming

LdapTemplate

Programming Against External Resource

- :: Procedural v/s object-oriented programming
- :: Traditional approach basically procedural
- :: Particularly applicable for data storage resources
 - :: Relational databases
 - :: Directories

Jayway

Traditional Approach: Basic Search

- :: Get a connection
- :: Prepare search query
- :: Execute search operation
- :: Loop through search result
 - :: Map data
- :: Handle errors
- :: Clean up resources

jayway

Traditional Approach: JNDI/LDAP

```
public List getAllPersonCommonNames() {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, "ldap://localhost:389/dc=example,dc=com");
    DirContext ctx;
    try {
        ctx = new InitialDirContext(env);
    } catch (NamingException e) {
        throw new RuntimeException(e);
    }
    LinkedList list = new LinkedList();
    NamingEnumeration results = null;
    try {
        SearchControls controls = new SearchControls();
        results = ctx.search("", "(objectclass=person)", controls);
        while (results.hasMore()) {
            SearchResult searchResult = (SearchResult) results.next();
            Attributes attributes = searchResult.getAttributes();
            String cn = (String) attributes.get("cn").get();
            list.add(cn);
        }
    } catch (NameNotFoundException e) {
        // The base context was not found, just clean up and exit
    } catch (NamingException e) {
        throw new RuntimeException(e);
    } finally {
        if (results != null) {
            try {
                results.close();
            } catch (Exception e) {
                // Never mind this.
            }
        }
        if (ctx != null) {
            try {
                ctx.close();
            } catch (Exception e) {
                // Never mind this.
            }
        }
    }
    return list;
}
```

The logo for Jayway, featuring the word "Jayway" in a stylized, italicized, red serif font.

Traditional Approach: JNDI/LDAP

```
public List getAllPersonCommonNames() {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, "ldap://localhost:389/dc=example,dc=com");
    DirContext ctx;
    try {
        ctx = new InitialDirContext(env);
    } catch (NamingException e) {
        throw new RuntimeException(e);
    }
    LinkedList list = new LinkedList();
    NamingEnumeration results = null;
    try {
        SearchControls controls = new SearchControls();
        results = ctx.search("", "(objectclass=person)", controls);
        while (results.hasMore()) {
            SearchResult searchResult = (SearchResult) results.next();
            Attributes attributes = searchResult.getAttributes();
            String cn = (String) attributes.get("cn").get();
            list.add(cn);
        }
        ...
    }
}
```

Jayway

Traditional Approach: JNDI/LDAP (cont)

```
...
} catch (NameNotFoundException e) {
    // The base context was not found, just clean up and exit
} catch (NamingException e) {
    throw new RuntimeException(e);
} finally {
    if (results != null) {
        try {
            results.close();
        } catch (Exception e) {
            // Never mind this.
        }
    }
    if (ctx != null) {
        try {
            ctx.close();
        } catch (Exception e) {
            // Never mind this.
        }
    }
}
return list;
}
```

Jayway

Demo

Traditional Approach, JNDI/LDAP

Traditional Approach: Summary

- :: Lots of boiler-plate code
- :: May be refactored to externalize common code
- :: Basic structure still needs to be repeated
 - :: Procedural programming
 - :: Code duplication: code smell

Jayway

Introducing the Template Approach

- :: Externalize program structure
 - :: Template class
- :: Template methods take care of basic structure, error management and resource cleanup
 - :: Supply relevant information
 - :: Provide callback interface implementations

Jayway

Introducing LdapTemplate

- :: `LdapTemplate` class
- :: Search methods take care of basic structure, list management, error management and resource cleanup
 - :: Base DN, search filter
 - :: `AttributesMapper` maps attributes to domain objects

Jayway

Comparing Traditional & Spring LDAP

```
public List getAllPersonCommonNames() {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, "ldap://localhost:389/dc=example,dc=com");
    DirContext ctx;
    try {
        ctx = new InitialDirContext(env);
    } catch (NamingException e) {
        throw new RuntimeException(e);
    }
    LinkedList list = new LinkedList();
    NamingEnumeration results = null;
    try {
        SearchControls controls = new SearchControls();
        results = ctx.search("", "(objectclass=person)", controls);
        while (results.hasMore()) {
            SearchResult searchResult = (SearchResult) results.next();
            Attributes attributes = searchResult.getAttributes();
            String cn = (String) attributes.get("cn").get();
            list.add(cn);
        }
    } catch (NameNotFoundException e) {
        // The base context was not found, just clean up and exit
    } catch (NamingException e) {
        throw new RuntimeException(e);
    } finally {
        if (results != null) {
            try {
                results.close();
            } catch (Exception e) {
                // Never mind this.
            }
        }
        if (ctx != null) {
            try {
                ctx.close();
            } catch (Exception e) {
                // Never mind this.
            }
        }
    }
    return list;
}
```

```
public List getAllPersonCommonNames() {
    return ldapTemplate.search("", "(objectclass=person)",
        new AttributesMapper() {
            public Object mapFromAttributes(Attributes attrs)
                throws NamingException {
                return attrs.get("cn").get();
            }
        });
}
```

The logo for Jayway, featuring the word "Jayway" in a stylized, red, cursive font.

Spring LDAP Approach (readable font)

```
public List getAllPersonCommonNames() {  
    return ldapTemplate.search("", "(objectclass=person)",  
        new AttributesMapper() {  
            public Object mapFromAttributes(Attributes attrs)  
                throws NamingException {  
                return attrs.get("cn").get();  
            }  
        });  
}
```

Jayway

Spring LDAP Approach (readable font)

```
public List getAllPersonCommonNames() {
    return ldapTemplate.search("", "(objectclass=person)",
        new AttributesMapper() {
            public Object mapFromAttributes(Attributes attrs)
                throws NamingException {
                return attrs.get("cn").get();
            }
        });
}
```

Jayway

Demo

Spring LDAP approach

Spring LDAP Approach: Summary

- :: Significantly less boiler-plate code
- :: Focus on important stuff
 - :: Where and what to find
 - :: Base DN
 - :: Search filters
 - :: What to do with the result
 - :: Map to domain objects
 - :: Create/modify/delete entries

Jayway

Is There More?

Additional Features in Spring LDAP

DN Management

- :: Java 1.4 does not have a DN class
- :: Spring LDAP provides `DistinguishedName` class
- :: Helps managing DNs
 - :: Building
 - :: Manipulating
 - :: Validating
 - :: Escaping

Jayway

Filter Management

- :: `org.springframework.ldap.filter` package
- :: Provides helper classes for working with search filters
 - :: Building
 - :: Escaping

Jayway

Demo

Filter Management in Spring LDAP

Attribute Management

- :: Attributes building and manipulation tedious and verbose
- :: Particularly applies to:
 - :: Multi-value attributes
 - :: Data modification (`modifyAttributes` operation)
 - :: `ModificationItem` production is tricky

Jayway

DirContextAdapter

- :: Spring LDAP provides `DirContextAdapter` class
 - :: Provides abstraction around attributes
 - :: Use together with `ContextMapper` in searches and lookups
 - :: Tracks modifications for `ModificationItem` production

Jayway

Demo

DirContextAdapter

LDAP Transactions

The Transaction Support in Spring LDAP

LDAP Transaction Need

- :: No server support for LDAP transaction
- :: Transaction support needed on several occasions
 - :: Atomic sequence of LDAP modifications
 - :: If one fails, all should fail
 - :: Some data in LDAP, some in RDB

Jayway

Spring Transaction Management

- :: Declarative transactions
- :: Defined in configuration files
 - :: Proxies apply transaction start/end

Jayway

Compensating Transaction Framework

- :: No server transaction support
- :: Compensating transactions
- :: Basic workflow:
 - :: Record state
 - :: Make modification
 - :: If anything fails, rollback by restoring original state from recorded data
- :: Pure client-side
 - :: No use if server goes down or connection breaks

Jayway

LDAP Compensating Transactions

- :: `org.springframework.transaction.compensating package`
 - :: General framework for compensating transactions
 - :: Hooks into Spring Framework's transaction management
- :: `o.sfwk.ldap.transaction.compensating package`
 - :: LDAP specific implementation of compensating transaction framework

Jayway

Different Approach for Different Operations

- :: Make record of original state
- :: Different approaches due to special features of LDAP
 - :: Not always possible to get the actual stored data
 - bind (create entry)
 - rename
 - unbind (delete entry)
 - rebind (delete original, replace with new data)
 - modifyAttributes

Jayway

Demo

Spring LDAP Transactions

Summary

Conclusion

- :: Spring LDAP provides template approach to LDAP programming
- :: Significantly less code needed for most common operations
- :: Enables programmer to focus on important stuff
- :: Spring LDAP provides additional useful features
 - :: dn, filter, attributes, modificationItems
- :: Builds on the JNDI API

jayway

Template in Other Environments

- :: Template programming could be used in other programming languages
 - :: Not necessarily limited to Java
 - :: Not even limited to object-oriented languages
- :: Significant benefits regardless of the environment

Jayway

Certified Java Professionals