GT2004Ugo

Developing Enterprise Web Applications with Cocoon and Spring

Ugo Cei, cocoon user since 1.7, committer since last year

What makes an application 'enterprise'?
have to manage data (create update etc) by many users at the same time

What are the issues:
think about concurrency, locking, cache, avoiding overwrites.

This presentation is not about a new version of Cocoon. Everything you will see can be done with cocoon 2.1.5 (current version)

Strengths of Cocoon

- Promotes separation of concerns
- SAX based pipelining
- Powerful caching
- Continuation based flow control
- Powerful forms framework

CForms and Continuations are the most important aspect of Cocoon for making it the best WebApp development framework.

The Spring framework Enterprise application is typically multi-tiered. You have to deal with business logic and things like persistence. Cocoon is not inclusive, doesn't force you to choose one thing or the other. For instance for heavy applications you can use EJB, but it is falling out of fashion, because it is heavyweight and difficult to test. So the last developments are more lightweight containers like Spring, based on Inversion of Control and Dependency Injection

Aim of the new lightweight frameworks is to simplify development, by reducing what you have to write just to cope with the development framework.

Ugo will show how you can leverage the strength of the spring framework with Cocoon.

Spring features:

- Setter and Constructor based Inversion of Control (Dependency Injection)
- Decouple how a component is instantiated from the user of the component.
- Nice layer of abstraction for JDBC.
- Hibernate, OJB, JDO work within Spring.
- Transaction management, difficult to do, so that it is scalable. Spring helps you with this.
- Aspect oriented programming, abstract away cross-cutting concerns in your application (own framework, can use AspectJ and AspectWerkz
- MVC Framework
- Lots more... (EJB, JMS, Mail etc.)

Spring provides you with a whole stack of enterprise services

Spring aims:

- simplicity
- modularity - include only the parts you use
- extensibility - add your own stuff
- non invasive framework. Components do not have dependancy on the framework itself. No implementing of interfaces or lookup service to locate other components. So the code remains portable to other frameworks. This is not possible with EJB or Avalon
- Testibility - you can test your components separately from the framework. For instance, Unit testing for EJB is impossible.
- Promotes best practices
- In practice: a simpler way of developing J2EE applications without EJB
- Does not add stuff that is commonly available (Logging, Pooling or Object Relational Mapping etc.). But spring makes it easier to use those libraries.

Architectural Components

- Database layer: e.g. JDBC
- Persistence layer: e.g. Hibernate, JDO,...
- Data access objects: decouple business logic from specific Persistence
- Service layer: simple access point to your functionality
- Controller: well implemented by Cocoon's flowscript
- View layer: Cocoon's template languages

Patterns of Enterprise Architecture is a book by Martin Fowler, Gang Of Four type book for Enterprise Applications Ugo will show how you can implement these in Cocoon

Application controller

Decides which domain logic to run and with which view to display the response Cocoon flowscript ideal for this job.

Two step view

Multi channel presentation layer. Split act of getting data, present it as xml format and then transform to a specific view.

First step: Cocoon generator turns domain model into XML Second step: transformers.

Caching

Cache in front or inside Cocoon to avoid to repeat database queries on data that haven't been changed.

Service Layer

Single entry point to many business logic domains. Using spring, it's recommended to implement a service layer to centralize resource and transaction management.

Data Access Objects

Decouple business logic from persistence mechanism

Define abstract interface, then Spring will be used to define which implementor of that interface is called at runtime.

== Declarative Transaction Management Example of service layer in spring

- Define the service (xml configuration file)
- wrap an AOP proxy around it
- AOP will automatically execute the calls in the context of a transaction
- Service layer is a convenient place for transaction boundaries

Domain Model

Complex web, hierarchy of connected objects. Hard to represent in EJB. So it's better to implement in POJO's, in combination with a O/R mapper. This can be a valid alternative to EJB

Never use raw JDBC directly

- it's verbose (try/catch/finally...)
- it's difficult
- Not fully portable

** Proprietary codes in exceptions
** Blob handling
** Stored procedures...
** Proprietary SQL

Object Relational mapping

Hibernate

Lazy load

Any persistence layer should give you lazy loading. The object knows how to load the dependent loading on demand. Hibernate does this via proxies. Problem: lazy loading overlaps with transactions/database sessions Solution: Open Session in View Filter. Close connection after view has been rendered. A few simple lines to add to web.xml. Once again: without writing code

==Serialized Large Objects ==
Very easy with cocoon forms and persistence layer

Optimistic Locking

Hibernate detects conflicts by automatic versioning

Ugo shows some hibernate configuration example. Again: no hardcoded details, just xml configuration

accessing Spring application context

- ContextLoaderListener configured in web.xml. Attaches spring container to current Cocoon servlet.
- read spring application context from flowscript
- There is no step 3

Wrap up

Why use cocoon/spring instead of more traditional solutions? Because you have flowscript and forms, and no one else has that.

Why spring with cocoon instead of avalon? Spring is geared towards developing enterprise applications.

Conclusion

Spring petstore cocoon block: http://new.cocoondev.org/main/g1/43 Online since yesterday.