Università di Bologna
CdS Laurea Magistrale in Ingegneria Informatica
I Ciclo - A.A. 2013/2014

Sistemi Distribuiti M

Proposta di esercizio:

Spring

Obiettivi (1)

- Mostrare come i framework, le strategie e le scelte progettuali adottate nei precedenti sotto-progetti, possano essere integrate e cablate in ***maniera flessibile mediante un framework a componenti leggeri*** e basato su meccanismi di Dependency Injection (DI) come Spring
- Valutare in maniera critica la realizzazione di una architettura 3-tier di tipo Java Enterprise ***senza essere necessariamente vincolati alla presenza dell'infrastruttura complessa e ingombrante di un Application Server*** (AS) come JBoss

- Per motivi di tempo, non considerati framework Model-View-Controller (come Spring MVC, JSF o Struts): accesso “diretto” ai bean di logica di business direttamente dalle pagine JSP

Obiettivi (2)

Data l'applicazione Web di esempio, per la gestione di *"Libri"*, *"Autori"* e *"Editori"* in una ipotetica biblioteca e data l'implementazione (mediante Hibernate e/o JPA) del layer e degli oggetti DAO predisposta nel progetto precedente...

si richiede di:

- integrare un ***Application Context di Spring*** all'interno dell'infrastruttura di un ***Web Container (Tomcat)*** per arricchire l'applicazione di un Bean Container per la gestione del proprio layer di business
- configurare ***opportunamente l'infrastruttura di accesso al layer di persistenza***, dotando il container Spring di piene funzionalità di JPA container (gestione del contesto di persistenza, del ciclo di vita dell'Entity Manager, delle transazioni...)

Obiettivi (3)

Data l'applicazione Web di esempio, per la gestione di *"Libri"*, *"Autori"* e *"Editori"* in una ipotetica biblioteca e data l'implementazione (mediante Hibernate e/o JPA) del layer e degli oggetti DAO predisposta nel progetto precedente...

si richiede di:

- fare uso del ***Test Context Framework*** messo a disposizione da Spring per effettuare test Junit "offline" con possibilità di rollback sul DB
- implementare mediante ***Aspect-Oriented Programming (AOP) un semplice aspetto*** per la ***gestione della sicurezza***

Bootstrap di Spring in una Web Application

- Trattandosi di una Web application in esecuzione all'interno di un Web container, necessità di **individuare una strategia per effettuare il bootstrap dell'ApplicationContext di Spring** in modo automatizzato al deployment dell'applicazione sul server
- Ciascuna Web application conforme alle specifiche Java Servlet, dispone di un ***deployment descriptor*** (localizzato in `WEB-INF/web.xml` all'interno di WAR) contenente tutte le definizioni necessarie al Web container per poter avviare ed eseguire correttamente l'applicazione

- Spring mette a disposizione uno specifico ContextLoader (`ContextLoaderListener`) per l'avvio di un ApplicationContext all'avvio di qualsiasi Web Container conforme alle specifiche Java Servlet
  - `ContextLoaderListener` deve essere configurato come `<listener>` nel deployment descriptor del WAR
  - presenza di un `<context-param>` chiamato `contextConfigLocation`, contenente i file path dei descrittori XML dell'Application Context Spring

Spring come Container JPA

Spring implementa la porzione delle specifiche JPA relative ai container provider => può operare come JPA Container

- Setup di un environment JPA con Spring richiede la configurazione, all'interno dell'Application Context, di:
  - DataSource (MySQL, HSQL…)
  - JPA Vendor Adapter (Hibernate)
  - EntityManager Factory Bean

NB: grazie ai meccanismi di DI, commutare fra diverse implementazioni di DBMS o di Persistence Provider comporta la sola modifica dei descrittori XML

- La classe **`LocalCointainerEntityManagerFactoryBean`** fornisce controllo completo sulla configurazione di una EntityManagerFactory JPA
  - Richiede la presenza di un JPA Persistence Descriptor localizzato in **`META-INF/persistence.xml`**
  - l'EntityManagerFactory può essere successivamente fornita al layer DAO mediante DI (vedi **`@PersistenceContext`**)

Transaction Management

- Si richiede di fare uso delle astrazioni messe a disposizione dal framework per la ***gestione della transazionalità*** in maniera ***dichiarativa*** (mediante annotazioni o XML)

- Mediante l'annotazione `@Transactional`, ciascuna transazione verso il DB può essere iniziata e conclusa (o eventualmente riportata allo stato iniziale) con la granularità del singolo metodo in maniera automatica e trasparente

- Trattandosi di una Web application ad ***architettura locale e facente uso di un'unica base di dati***, i meccanismi transazionali forniti dall'EntityManager JPA sono più che sufficienti per assicurare semantiche corrette

Spring e Collaudo: Test Context Framework

Spring semplifica il ***collaudo*** dei componenti applicativi (che tipicamente sono POJO svincolati da API specifiche di un AS Java Enterprise) e fornisce un efficace supporto ai test di integrazione ("annotation-driven" e agnostico rispetto al testing framework)

- ***Implementare una serie di test*** (Junit4) in grado di coprire e validare le semantiche delle interfacce DAO
  - suggerimento: Spring offre completa integrazione con Junit4.4 tramite un custom runner. Annotando le classi di test con `@Runwith(SpringJUnit4ClassRunner.class)` si ha la possibilità di implementare test junit standard con i vantaggi del Test Context Framework, come caricamento degli application context, dependency injection delle istanze di test ed esecuzione transazionale dei metodi di test
- Si consiglia di estendere la classe `AbstractTransactionalJUnit4SpringContextTests` per il rollback delle transazioni verso il DB al termine di ogni test

Management - JMX

Il supporto a JMX fornito da Spring consente di integrare velocemente e trasparentem. POJO in infrastruttura JMX

- Mediante un MbeanExporter, ***qualsiasi bean del contesto Spring può essere registrato presso un Mbean Server esistente*** (quando si esegue, ad esempio, all'interno di un AS)
- MbeanServerFactoryBean consente invece di ***istanziare un Mbean Server in ambienti standalone***

Si richiede di implementare un ***semplice servizio JMX per la gestione e il monitoraggio del DataSource correntemente in uso*** nella Web application:

- POJO dovrà essere registrato presso un MBean Server istanziato all'avvio del container Spring
- dovrà fornire accesso in lettura e scrittura (get/set) a parametri del DataSource (maxActive, maxIdle, minIdle, numActive, numIdle...)
- Client JMX: MX4J (vedi riferimenti)

AOP: Sicurezza

Mediante ***AOP*** possono essere ***isolati e modularizzati componenti di logica crosscutting (verticali)*** rispetto al resto della logica primaria di business (sicurezza, transazionalità, logging, …)

- Si richiede di implementare un semplice bean (o quantomeno di impostarne l'implementazione!) per la ***gestione di problematiche di sicurezza, come autenticazione e controllo di accesso***
- Facendo uso del supporto fornito da Spring AOP, ***l'aspetto che si occupa della sicurezza*** (implementato come advice di tipo “around”) deve essere ***applicato alle invocazioni di tutti i metodi pubblici*** delle interfacce DAO, condizionandone l'invocazione in base alla corretta autenticazione o meno dell'utente

NB: non è intenzione dell’esercizio giungere a un'implementazione finale delle funzionalità di autenticazione e autorizzazione, ma di aiutare a capire come metodi di business logic possano essere “intercettati” al fine di iniettare trasparentemente nuove funzionalità nel flusso di logica principale

Riferimenti e Possibili Approfondimenti

- The Spring Framework - Reference Documentation, version 2.5.6
- JSR 220: Enterprise JavaBeansTM,Version 3.0, Java Persistence API
- Rod Johnson, *J2EE Development Without EJB*, Wrox
- Thomas Van de Velde, Bruce Snyder, Christian Dupuis, and Sing Li - *Beginning Spring Framework 2*, Wrox
- Spring IDE per Eclipse: http://springide.org/blog/
- Spring MVC (per chi voglia approfondire)

  http://static.springframework.org/docs/Spring-MVC-step-by-step/
- Spring Security (idem)

  http://static.springsource.org/spring-security/site/reference.html
- MX4J - http://mx4j.sourceforge.net/