Spring Framework

Luiz Daniel Creão Augusto

laugusto@ime.usp.br

Sistemas de Middleware Avançados

IME-USP - 2006/02

Agenda

- Introdução
- Inversão de Controle
- Spring AOP
- Portable Service Abstractions
    - Spring Beans, JDBC, EJBs, ...
- Spring 2.0
- Referências

1. Introdução: O que é Spring?

- Spring é um framework leve com inversão de controle e orientado à aspectos.
- "*Spring torna o desenvolvimento em J2EE mais fácil e divertido.*"
- Introduzida por Rod Johnson em "J2EE Design & Development"

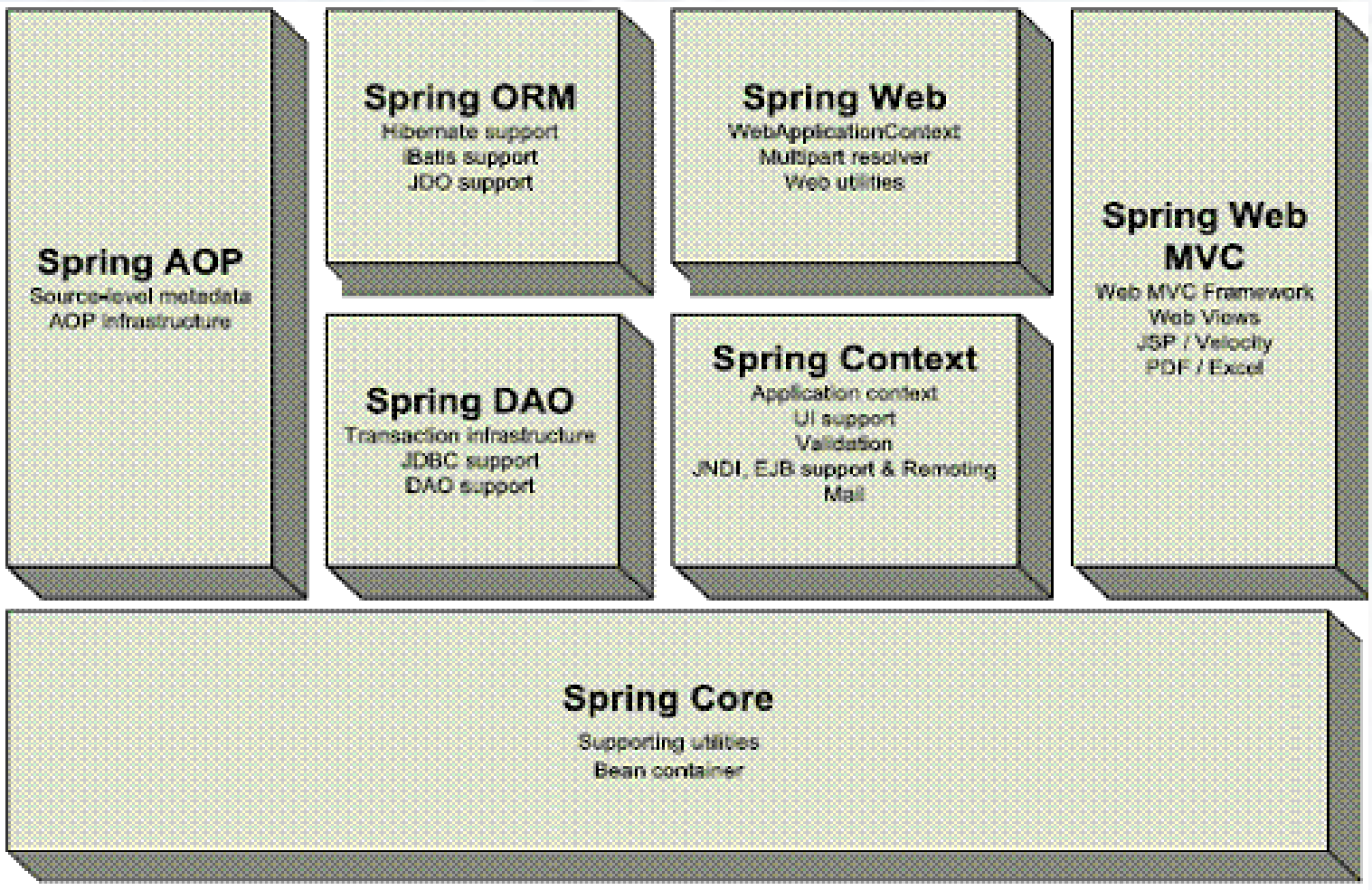
1. Introdução: Benefícios

- Container leve elimina a necessidade de código repetitivo, como *lookups*
- Framework com inversão de controle que resolve automaticamente as dependências entre beans
- Código comum a várias seções, como gerenciamento de transações, são implementados como aspectos.
- Disponibiliza camadas de abstração para tecnologias populares (como Hibernate e JDBC), fáceis de usar

1. Introdução: 'Princípios'

- O código da aplicação não deve depender da API Spring
- Spring não deve competir com boas soluções já existentes, mas deve ser capaz de se integrar (WebSphere)
- Teste de código é uma tarefa crítica e o container não deve interferir neste objetivo
- Spring deve ser fácil e agradável de se utilizar

1. Introdução: Esquema Geral



1. Introdução: Arquitetura e Utilização

- Considerado alternativa/substituto leve aos EJBs
- Não é exclusivo Java (Spring.NET)
- Soluções prontas para problemas típicos:
    - JDBC, LDAP, Web Services, ...
- Código do Spring é comprovadamente bem estruturado
    - Possivelmente o melhor dentre os frameworks
    - Análise usando Structure 101
    - 139 pacotes
    - Livre de ciclos de dependências

1. Introdução: Os Três Alicerces

- Injeção de Dependências
    - Também conhecido como IoC (Inversão de Controle)
- Programação Orientada à Aspectos
    - Baseado em injeções em tempo de execução
- Portable Service Abstractions
    - Todo o 'resto' do Spring
    - ORM, DAO, Web MVC, Web, etc.
    - Permite utilização sem saber como eles funcionam

2. Injeção de Dependências

- O que é?
  - Forma de se resolver as dependências entre objetos automaticamente 'injetando' referências sob demanda
  - Já visto em outro seminário de SMA
- Quem determina como os objetos devem ser injetados?
  - O framework IoC, geralmente via arquivos de config. XML
- Benefícios
  - Responsabilidade de objetos dependentes vai pra config.
  - Reduz o acoplamento e encoraja desenv. baseado em interfaces
  - Permite a aplicação ser reconfigurada sem mexer no código fonte (somente alterando o XML, por exemplo)

2. Injeção de Dependências: Forma tradicional

```
private AccountService accountService = null;
public void execute(HttpServletRequest req, ….) throws Exception {
  Account account = createAccount(req);
  AccountService service = getAccountService();
  service.updateAccount(account);
}
private AccountService getAccountService() throws … {
  if (accountService == null) {
    Context ctx = new InitialContext();
    Context env = (Context) ctx.lookup(“java:comp/env”);
    Object obj = env.lookup(“ejb/AccountServiceHome”);
    AccountServiceHome home = (AccountServiceHome)
        PortableRemoteObject.narrow(env, AccountService.class);
    accountService = home.create();
  }
  return accountService;
}
```

2. Injeção de Dependências: Com Spring IoC

· O container faz a 'injeção' de uma implementação apropriada

```
private AccountService accountService = null;
public void setAccountService(AccountService accountService) {
    this.accountService = accountService;
}
public void execute(HttpServletRequest req, ....) throws Exception {
    Account account = createAccount(req);
    accountService.updateAccount(account);
} ...
```

• Mais a frente quando lidar com Beans, o XML será mostrado

3. Spring AOP: Conceitos

- Aplicações devem se preocupar com coisas como:
    - Transações, logging, segurança, ...
- Estas responsabilidades pertecem as classes de impl?
    - Deveria nossa classe de serviço ser responsável por gerenciar transações, fazer logging ou segurança?
- Estas questões são geralmente referidas como 'preocupações ortogonais'

3. Spring AOP: Conceitos

- Tentativas de se separar problemas, aumentar modularidade e reduzir redundância
  - Separation of Concerns (SoC)
    - *Break up features to minimize overlap*
  - Don't Repeat Yourself (DRY)
    - Minimizar duplicação de código
  - Cross-Cutting Concerns
    - Aspectos de programação que afetam diversos trechos de código (como logging)
- AspectJ é o principal pacote AOP (AOP estática)
- Spring AOP utiliza AOP dinâmica para impl. suporte a transações, logging e segurança

4. Portable Service Abstractions

- Migração de apps com esses serviços exigem pouco recode
    - Idealmente rodam em qualquer sistema
        - Ex.: Hibernate, JDBC, LDAP, Spring Beans, ...
    - Abstração sem expor as dependências do serviço
        - Ex.: Acesso LDAP sem conhecer o servidor LDAP
- PSA no Spring é basicamente tudo que não é IoC ou AOP

4. Spring PSA: Alguns Módulos

- Spring Core: representa as principais funcionalidades do Spring, no qual o principal elemento é o BeanFactory.
- Spring DAO: camada de abstração para JDBC, eliminando grande parte da codificação necessária para interagir com um banco de dados.
- Spring ORM: provê integração do Spring com outros frameworks para persistência de objetos, como Hibernate e iBatis.
- Spring Web: provê funcionalidades específicas para projetos Web
- Spring MVC: fornece uma implementação similar aos Struts.

4. Spring PSA: Spring DAO

```
Connection con = null;
    try
    {
      String url = "jdbc://blah.blah.blah;";
      con = myDataSource().getConnection();
      Statement stmt = con.createStatement();
      String query = "SELECT TYPE FROM SENSORS";
      ResultSet rs = stmt.executeQuery(query);
      while (rs.next()) {
        String s = rs.getString("TYPE");
        logger.debug(s + "   " + n);
      }
    } catch (SQLException ex)
    {
      logger.error("SQL ERROR!",ex);
    }
    finally
    {
      con.close();
    }
```

4. Spring PSA: Spring DAO

```
DataSource ds = (DataSource) bf.getBean("myDataSource");
JdbcTemplate temp = new JdbcTemplate(ds);
List sensorList = temp.query("select sensor.type FROM sensors",
  new RowMapper() {
     public Object mapRow(ResultSet rs, int rowNum)
       throws SQLException;
        return rs.getString(1);
     }
  });
```

```
<bean id="myDataSource"  class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName"> <value>org.gjt.mm.mysql.Driver</value>
</property>
  <property name="url">  <value>jdbc:mysql://romulus/sensors</value> </property>
  <property name="username"> <value>heater</value> </property>
  <property name="password"> <value>hotshot</value> </property>
</bean>
```

4. Spring PSA: Spring Bean Container

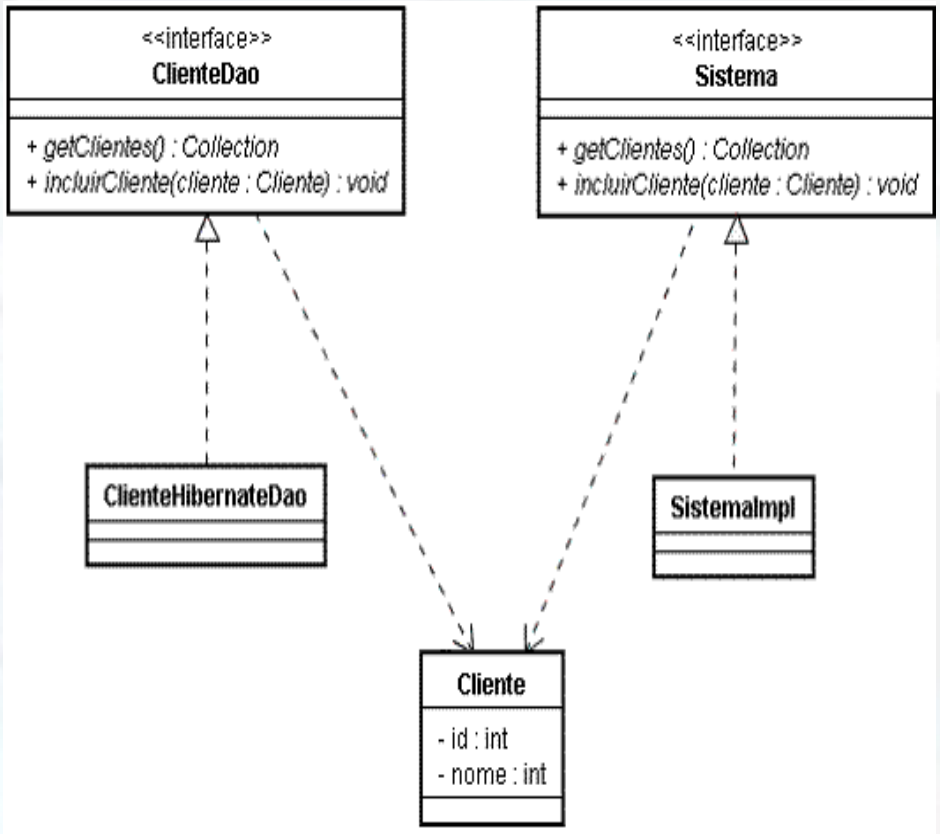
- Utiliza o IoC para gerenciar componentes.
- Componentes construídos como JavaBeans tradicionais.
- Container gerencia os relacionamentos entre os beans e é o responsável pela configuração.
- Spring tarda ao máximo a criação dos Beans
- Beans são tipicamente definidos em um arquivo XML

4. Spring PSA: Tipos de Bean Containers

- Bean Factory
  - Provê suporte básico para a injeção de dependência
  - Gerencia configs e ciclo de vida de beans
- Application Context
  - Cria um Bean Factory e adiciona serviços:
    - Trata mensagens para internacionalização
    - Carrega recursos genéricos
    - Dispara e trata eventos

4. Spring PSA: Exemplo de uso de Bean Containers

- *ClienteDao* e *Sistema* delimitam o comportamento de objetos que implementam a persistência de objetos *Cliente* e o acesso ao sistema
- *ClienteHibernateDao* implementa a interface *ClienteDao* e realiza a persistência de objetos do tipo *Cliente* através do Hibernate
- *SistemaImpl* que fornece uma implementação concreta para a interface *Sistema.*



4. Spring PSA: Exemplo de uso de Bean Containers

```
### Arquivo applicationContext.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
        <bean id="ClienteDao" class="ClienteHibernateDao"/>
        <bean id="Sistema" class="SistemaImpl">
                <property name="clienteDao" ref="ClienteDao"/>
        </bean>
</beans>

public class Aplicacao {
      public static void main ( String[] args ) {
            XmlBeanFactory factory = new XmlBeanFactory ( new
FileSystemResource ( "applicationContext.xml" ) ) ;
            Sistema sistema = (Sistema) factory.getBean ( "Sistema" ) ;
            sistema.incluirCliente ( new Cliente() ) ;
      }
}
```

4. Spring PSA: Bean Containers - Considerações

- Container cria beans em ordem de acordo com grafo de deps.
- Beans são criados usando contrutores (geralmente sem argumentos) ou métodos de factory
- Dependência não injetadas via contrutores são resolvidas por setters
- Sem chamadas a Contexts e lookups
- Ambos beans iniciados apenas na chamada factory.getBean()
- Programação orientada a interfaces: implementações utilizadas podem mudar com alteração apenas no XML

5. Spring 2.0: Principais novidades

- Integração da JPA ao Spring
- Spring Web foi o módulo com mais modificações
- Spring-OGSi (OGSi serviços definidos como beans)
- AOP: Utiliza declarações semelhantes ao @AspectJ
- Abstração para agendamento de tarefas
- E muitas outras novidades e pequenas melhorias

6. Referências

- Spring Framework: http://www.springframework.org/
- The Spring Framework for J2EE - Dan Hayes
- http://static.springframework.org/spring/docs/2.0.x/reference/new-in-2.html
- http://dev2dev.bea.com/pub/a/2005/07/better_j2eeing.html
- Spring 2.0 Update: http://www.infoq.com/articles/spring-2-update
- Intro. to Spring Framework and Dependency Injection - Aaron Zeckoski