Spring

Spring
Introduction and Dependency Injection

Dan Hayes

October 27, 2005

CHARIOT SOLUTIONS

Agenda

- Introduction to Spring
- The BeanFactory
- The Application Context
- Inversion of Control
- Bean Lifecyle and Callbacks

Introduction to Spring

Spring is a lightweight inversion of control and aspect oriented container framework

Spring makes developing J2EE application easier

and more fun!!!

Introduced by Rod Johnson in “J2EE Design & Development”

Introduction to Spring

Spring helps J2EE developers by:

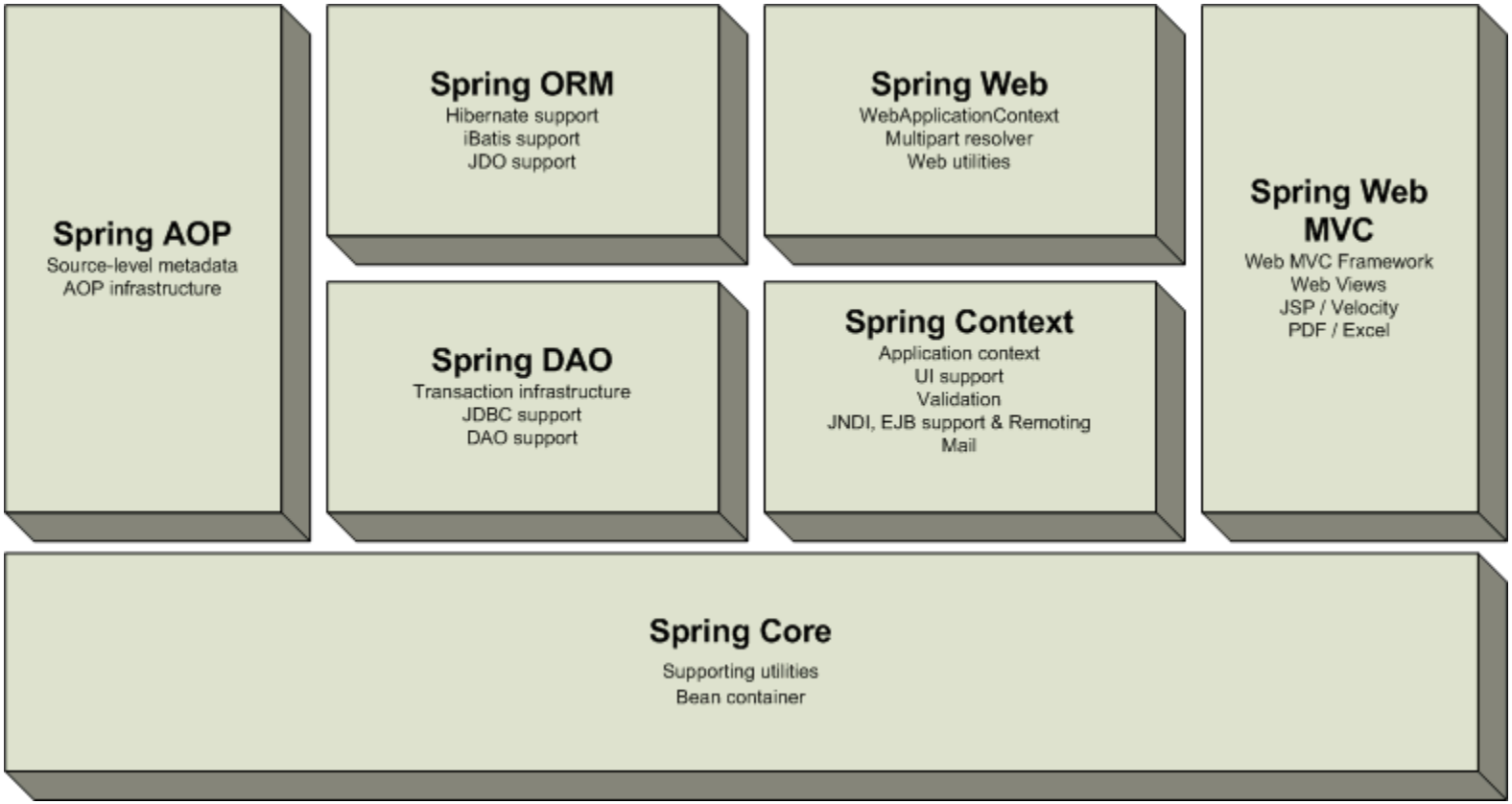
- Offering a lightweight JavaBean *container* that eliminates the need to write repetitive plumbing code such as lookups
- Providing an inversion of control framework that allows bean dependencies to be automatically resolved upon object instantiation
- Allowing cross cutting concerns such as transaction management to be woven into beans as “aspects” rather than becoming the concern of the business object
- Offering layers of abstraction on top of popular existing technologies such as JDBC and Hibernate that ease their use and organize configuration management

Introduction to Spring

…while adhering to certain principals:

- Your application code should not depend on Spring API's
- Spring should not compete with good existing solutions, but should foster integration
- Writing *testable* code is critical and the container should help - not interfere - with this objective
- Spring should be a pleasure to use

Introduction to Spring



Introduction to Spring

Architecturally speaking:

- Spring can be employed in a number of situations
  - Web app only applications
  - Applications that involve EJB's or other Services
  - Applications that access one or more resources (databases)
- However, Spring is emerging as the leading framework for "lightweight" J2EE application development
  - Do we really need heavyweight services, such as EJB, provided by traditional J2EE application servers?
  - Clustering of co-located (single JVM) applications

The Bean Factory

- Where it all starts…
- Configuration mechanism for managing beans
- Could theoretically be any type of configuration
  - XML
  - Properties File
  - Database
  - LDAP
- The most common is the XmlBeanFactory

Xml Bean Factory

```
<?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="..." class="...">
     ...
   </bean>
   <bean id="..." class="...">
     ...
   </bean>
   ...
</beans>
```

Bean Factory

EXAMPLES 1-3

Application Context

- ApplicationContext is a sub-interface of BeanFactory and adds more useful features
  - Support for resolving text messages
  - Generic means of loading file resources
  - Publishing of events
- WebApplicationContext is a sub-interface
  - XmlWebApplicationContext is the implementation used in web applications

Application Context

Examples 4-7

Inversion of Control

- What is it?
  - A way of sorting out dependencies between objects and automatically “injecting” references to collaborating objects on demand
- Who determines how the objects should be injected?
  - The IoC framework, usually via XML configuration files
- Benefits
  - Removes the responsibility of finding or creating dependent objects and moves it into configuration
  - Reduces coupling between implementation objects and encourages interface based design
  - Allows an application to be re-configured outside of code
  - Can encourage writing testable components

Inversion of Control

Traditional way of obtaining references….

```
…
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;
}
…
```

Inversion of Control

With Spring IoC the container will handle the "injection" of an appropriate implementation ….

```
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);
}

…
```

Inversion of Control

Examples 8-11

Bean Lifecycle

- Beans managed by Spring have a distinct and predictable lifecycle
- Spring provides simple ways of receiving callbacks from lifecycle events
- Hooks are purely optional, no dependency on Spring in your beans – unless you want to

Bean Lifecycle- Startup

1. Instantiate
2. Populate Properties
3. BeanNameAware's setBeanName()
4. BeanFactoryAware's setBeanFactory()
5. ApplicationContextAware's setApplicationContext()
6. Pre-initialization BeanPostProcessors
7. InitializingBean's afterPropertiesSet()
8. Call custom init-method
9. Post-initialization BeanPostProcessors

Bean Lifecycle-Shutdown

DisposableBean's destroy()

Call custom destroy-method

Bean Lifecycle

Examples 12 - 14

Extras

- Bean Auto-Wiring
  - By name
  - By type
  - Constructor
  - Autodetect
- Property Placeholder Configurer
  - A way of externalizing certain parameters in the bean definition file
  - Can only be one instance in the Application Context but it accepts multiple property files

Extras

Examples 15 - 16

Contact

- Dan Hayes
  - Chariot Solutions
  - Ft. Washington, Pennsylvania
  - dhayes@chariotsolutions.com