



ORACLE



JavaOne

Harnessing the Power of Java™ Platform, Enterprise Edition (Java™ EE) With Spring

Mike Keith

Oracle

<http://otn.oracle.com/oc4j>

Colin Sampaleanu

Interface21

<http://www.interface21.com>

<http://www.springframework.org>

TS-43350

GoalGoal

To learn more about how applications can use Spring to leverage Java Platform, Enterprise Edition (Java EE Platform) architectures and features, and how Spring can enhance the Java EE platform development experience.

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration with Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration with Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

What Is Spring?

- Spring is an “Application Framework”
- A software layer that sits between the application and the runtime environment
- A suite of services that applications can selectively choose from according to their needs
- An open source project with integrated support for most popular Java platform projects in common use
- An enabling technology that promotes good object-oriented design principles
- Apache 2.0 licensed and free

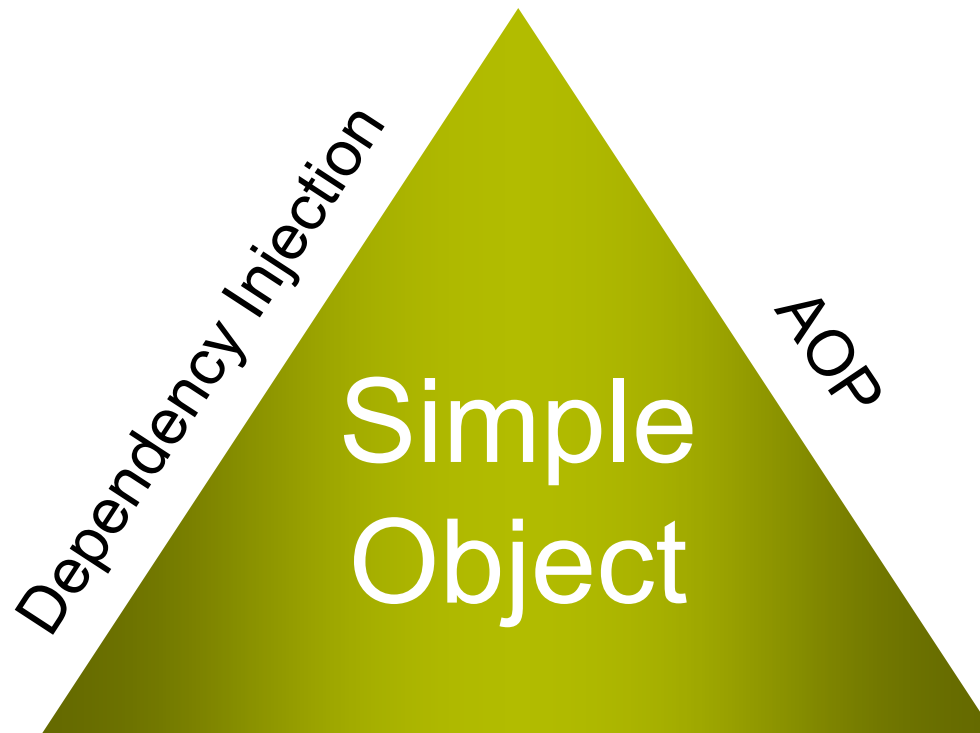
What Does Spring Do?

- Provides configuration, wiring, and life cycle management of application components
- Offers abstraction layer of common subsystems like transactions and data sources
- Hosts and provides simplified access to standard persistence services such as Java DataBase Connectivity (JDBC™) API and Java Persistence API (JPA)
- Normalization of data access exceptions from heterogeneous database platforms
- Weaves aspect-oriented advice into application code (when AOP advice is supplied)

What Is Spring Not?

- Application Server
 - Spring does not implement system services like thread pools or transactions, but relies instead upon the services of the server in which it is deployed
- Java EE Platform Container
 - Spring typically runs within a Java EE platform or web container
- Dependent upon Java EE platform
 - Spring can be configured to run in any hosted Java Platform, Standard Edition (Java SE) or Java EE application environment—including older versions
- Anti-standards
 - Spring leverages and prefers to use existing standards

Enabling Technologies



Portable Service Abstractions

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration With Java EE Platform

Web Tier

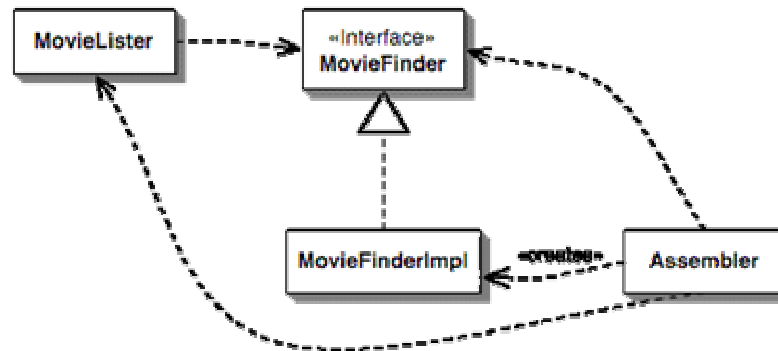
Persistence Tier

Testing Enterprise Applications

Summary

Dependency Injection and Inversion of Control

- Allows a software component to express what it depends on to work
 - Without hard-coding
- A third party, called an assembler, is responsible for “plugging in” an implementation



The Assembler

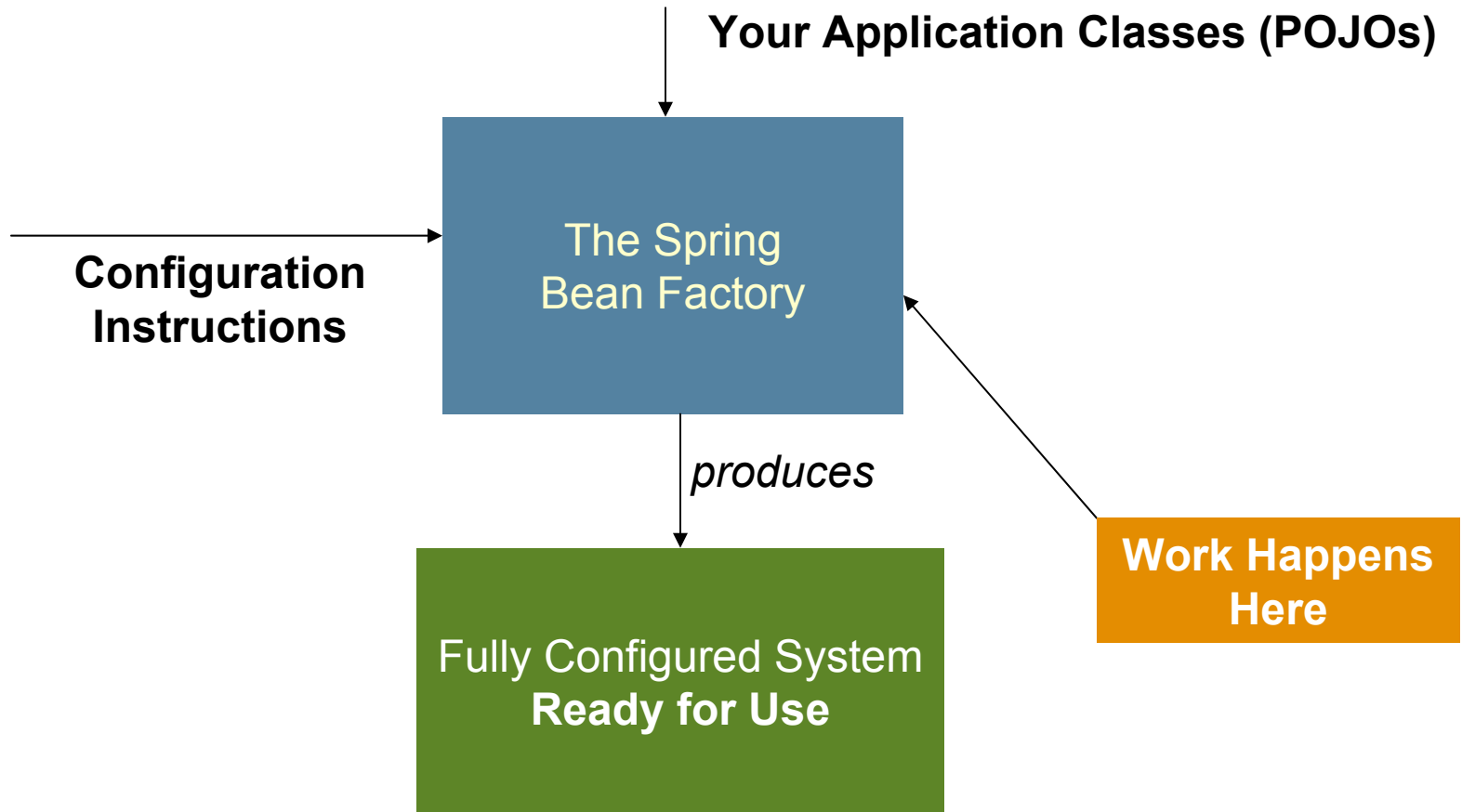
The “assembler” often called a “container” or “factory”



Spring Beans

- Basic unit of state, logic, and configurability
- May have zero or more **dependencies** on other beans, or may be a dependency of zero or more other beans
- Beans and their dependencies are defined individually and collectively in XML file
- Bean dependencies are satisfied by Spring at runtime through the use of **injection**
- Grouped together and accessible from a Spring **application context**

The Bean Factory



Example—Service Interface

```
public interface RewardService {  
    public int rewardPoints(  
        int accountNumber,  
        PurchaseEvent purchase);  
}
```

Example—Service Implementation

```
public class SimpleRewardService implements RewardService {  
    public AccountRepository accountRepository;  
  
    public void setAccountRepository(AccountRepository repo) {  
        this.accountRepository = repo;  
    }  
  
    public int rewardPoints(...) {  
        Account account = accountRepository.loadAccount(accountNumber);  
        int points = calculatePointsFor(purchase);  
        account.addPoints(points);  
        accountRepository.updateAccount(account);  
        return points;  
    }  
}
```

Example—Configuration

```
<beans>  
  
  <bean id="rewardService" autowire="byType"  
        class="org.acme.SimpleRewardService"/>  
  
  <bean id="accountRepository"  
        class="org.acme.JdbcRepository"/>  
    <property name="dataSource" ref="dataSource"/>  
  </bean>  
  
  <jee:jndi-lookup id="dataSource"  
                  jndi-name="jdbc/accountDataSource"/>  
</beans>
```


Example—Getting the First Bean

```
...  
RewardService rewardService;  
...  
private void initialize() {  
    ApplicationContext appCtx;  
    appCtx = new ClassPathXmlApplicationContext(  
        "META-INF/app-context.xml");  
    rewardService = (RewardService) appCtx.getBean("rewardService");  
}
```

OR...

```
...  
@Resource  
RewardService rewardService;  
...
```

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration With Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

Spring AOP—What’s the Point?

- Look at this requirement
 - “Perform a role-based security check before every application method”



A sign this requirement is a cross-cutting concern

Spring AOP—What's the Point?

- Allows a ***cross-cutting*** requirement to be implemented as a *single* module
 - Don't Repeat Yourself (DRY) principle
 - Using AOP avoids ***scattering***—Code is no longer repeated
 - Separation of Concerns (SOC) principle
 - Using AOP avoids ***tangling***—Each piece of code is focused on one clear task

Spring AOP

- Use AspectJ ***pointcut expressions*** or annotations to indicate the ***join points*** in the application where “something” needs to happen
- Write code to implement the “something”; this is called ***advice***
- ***Weaving*** of advice into the application happens at runtime so no pre-compile step is needed
- Advice may be applied temporally or conditionally relative to the join points
 - Before, after, around, on exception, on success, etc.
- The AOP advice often leverages abstractions over Java Platform APIs (e.g., transactions)

AOP Example—Apply via Annotations

```
@Secured({"IS_AUTHENTICATED_REMEMBERED"})  
public interface Clinic {  
  
    ...  
  
    @Secured({"ROLE_SUPERVISOR"})  
    void storeVisit(Visit visit) throws ...;  
}
```

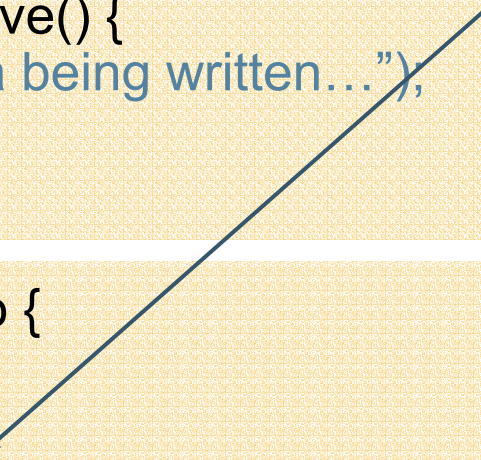
AOP Example—Apply via Pointcut

```
@Aspect
public class SaveTracker {
    private Logger logger = Logger.getLogger(getClass());

    @Before("execution(void org.acme..*Dao.save*(*))")
    public void trackSave() {
        logger.info("Data being written...");
    }
}
```

```
public class UserDao {

    public void save() {
        ...
    }
}
```



Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration With Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

Portable Service Abstractions

- Software abstractions for specific enterprise services
 - Enable pluggability of implementations
 - Reduce boilerplate
 - Single configuration and API for common features
- Examples
 - PlatformTransactionManager
 - Data access templates (JPA/JDBC API and ORM)
 - Java Message Service operations
 - Java Management Extensions (JMX™) service exporting and access

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration With Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

Transactions

- Two principal transaction handling strategies in Java EE platform
 - JTA: Allows access to container's Transaction Manager, including possible participation in distributed transactions
 - JDBC API transaction used directly, on the Connection, or via wrapper layer in persistence framework

Spring Transaction Integration

- Two main choices for transaction handling strategy in Java EE Platform Spring-based applications
 - Container-managed transactions via Enterprise JavaBeans™ (EJB™) architecture session beans
 - Spring demarcated transactions
 - Using JTA transaction manager
 - Using local (JDBC API) transactions

Container-Managed Transactions

- EJB beans are marked as transactional through the use of annotations or XML
- EJB beans hand off to Spring-managed beans to do some of the work
- Implies JTA and relies upon the container Transaction Manager
- Spring able to bind DataSources and other resources to the JTA transaction, for re-use and sharing

Spring Transaction Demarcation

- Abstract Spring transaction layer allows application to choose the most appropriate infrastructure for the need at hand
 - Can easily switch implementations later
 - Can work declaratively or programmatically
 - Declaratively—Annotations or XML
 - Programmatically—TransactionTemplate
 - Including legacy Java SE application environments and Java EE application environments
- In the Java EE platform container, JTA strategy typically plugged in
 - Delegating to container's transaction manager

Declarative Transactions: Annotations

can also specify at
class/interface level

overriding attributes at
the method level

```
@Transactional  
public class RiskServiceImpl implements RiskService {  
  
    @Transactional(isolation=Isolation.REPEATABLE_READ,  
                    timeout=60)  
    public RiskConfirmation calculateRisk(RiskCriteria r) {  
        // atomic unit-of-work  
    }  
  
    @Transactional(readOnly=true)  
    public List<Policies> listPoliciesFrom(Date d) {  
        // atomic unit-of-work  
    }  
}
```

Data Sources

- Normally managed by the container, maintain connection pools, and are JTA-bound
- Abstract Spring data source layer permits application to be loosely coupled to the container
- Can inject different types of data sources depending upon what is available/desired
 - Container-managed JTA data source
 - Vanilla JDBC API data source
 - Pooled non-managed data source
 - Custom or test data source

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration With Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

Web Tier Integration

- Can use light integration to combine Spring with any web framework, or no framework at all
- Register a Spring `ServletContextListener` to cause an application context to be bound to `ServletContext` on web app startup
 - Optionally configure resource path to application context configuration file(s)
- Loading application context causes Spring to start managing the defined beans
 - Can access usual Spring features through normal techniques

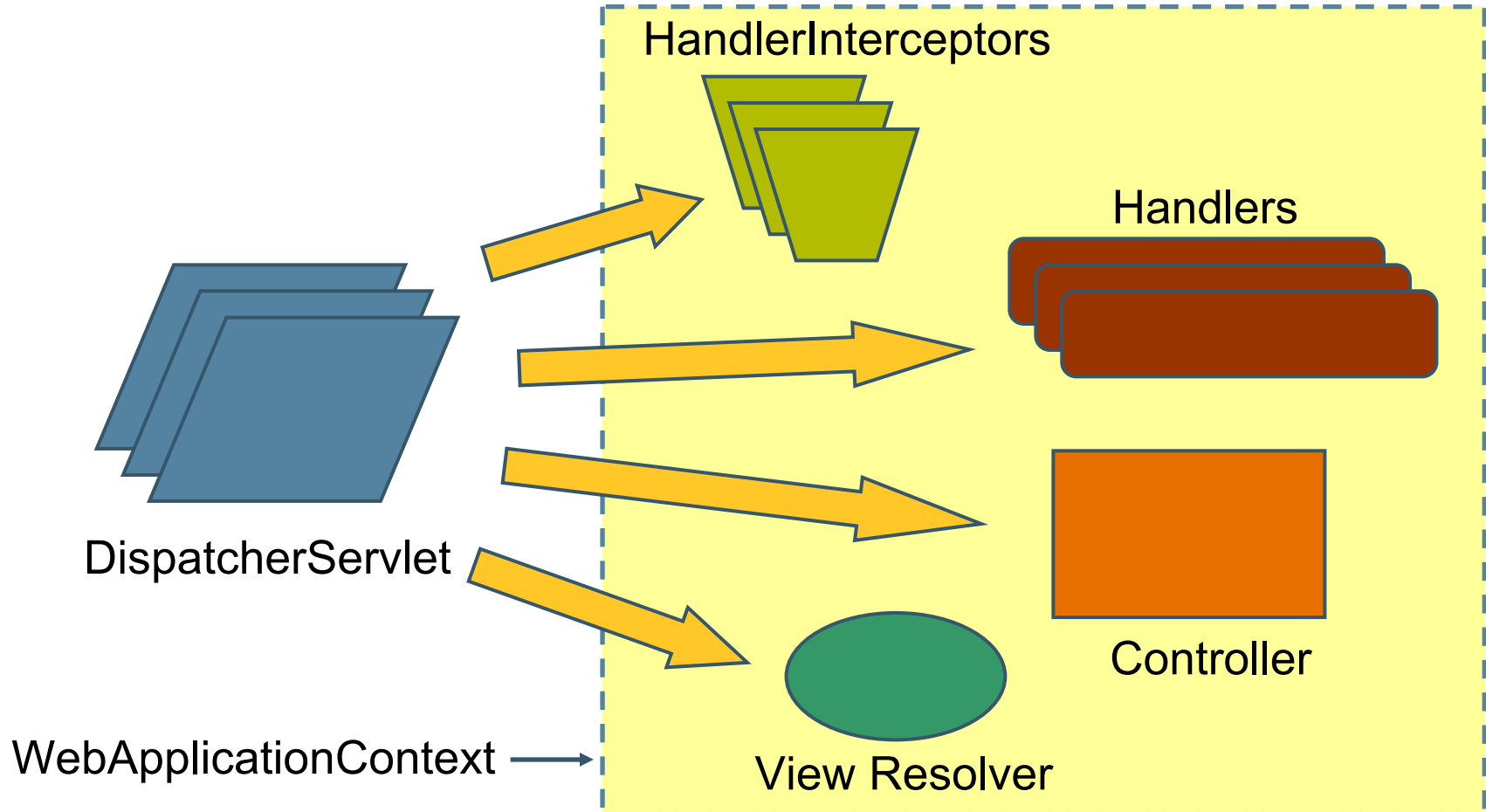
Example web.xml

```
<web-app>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/app-context.xml</param-value>
  </context-param>
  ...
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  ...
</web-app>
```

Spring MVC

- Flexible lower-level HTTP dispatcher infrastructure, as used in parts of Spring Remoting
or...
- Complete action-oriented Model-View-Controller (MVC) framework for web applications
- DispatcherServlet front end uses strategies to dispatch requests to registered handlers, controllers, and views
- Each servlet may also have its own web application context
- **Handler interceptors** allow interception of handler and request processing

Spring MVC—The Big Picture



Spring Web Flow

- Used on top of JavaServer™ Faces technology or other UI-focused web frameworks such as Spring MVC, Struts, etc.
- Capture a logical flow of your web application as a self-contained module, at a higher level
 - In a declarative fashion
 - Essentially a black box, including sub flows
 - Representing a user conversation
 - Introduces new scope: Flow Scope
 - Highly manageable
- Part of Spring's web stack—A full subproject

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration With Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

Integration With JPA

- Java Persistence API is portable persistence API that works in both Java EE platform and Java SE platform
- Exists a Service Provider Interface (SPI) for a host container to invoke a JPA implementation
 - Provides standardized injection facility
 - Automatic detection of persistent entities
 - Initialization and management of entity managers
 - Persistence context propagation across components
 - Extra support for provider entity weaving
 - Cleaner integration story (more opportunities for optimization)

Integration With JPA

- Spring acts as a host container for JPA and implements the SPI
- Spring transaction layer with JPA supports additional transaction combinations
- Common configuration to access extended features supported by major JPA providers
- Increased testability
- Can be used in conjunction with other Spring features, like AOP

Spring JPA Configuration

```
<bean id="entityManagerFactory"  
  class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">  
  <property name="dataSource" ref="dataSource"/>  
  <property name="loadTimeWeaver">  
    <bean class="org.springframework.orm.jpa.support.ReflectiveLoadTimeWeaver"/>  
  </property>  
  <property name="jpaVendorAdapter">  
    <bean class="org.springframework.orm.jpa.vendor.TopLinkJpaVendorAdapter"/>  
  </property>  
</bean>  
  
<bean id="transactionManager"  
  class="org.springframework.orm.jpa.JpaTransactionManager">  
  <property name="entityManagerFactory" ref="entityManagerFactory"/>  
</bean>
```

Spring Bean Using JPA

```
public class JpaAccountRepository implements AccountRepository {  
  
    @PersistenceContext  
    EntityManager em;  
  
    public Account loadAccount(int accountNumber) {  
        return em.find(accountNumber);  
    }  
  
    public void updateAccount(Account account) {  
        em.merge(account);  
    }  
}
```

Spring JDBC API in a Nutshell

```
int count = jdbcTemplate.queryForInt(  
    "SELECT COUNT(*) FROM CUSTOMER");
```

- Acquisition of the connection
- Participation in the transaction
- Execution of the statement
- Processing of the result set
- Handling any exceptions
- Release of the connection

**All handled by
the template**

Spring JDBC API in a Nutshell:

Querying for Objects

```
List<Person> results = jdbcTemplate.query("select * from ...",  
    new ParameterizedRowMapper<Person>() {  
        public Person mapRow(ResultSet rs, int row) {  
            // map the current row to a Person  
            return new Person(...);  
        }  
    });
```

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration With Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

Testing

- **The Problem:** It's usually hard to unit test Java EE application environment logic
- **The Reason:** Java EE application environments tend to have dependencies on Java EE platform resources which are not available outside the container
- **The Solution:** Let Spring inject resources that fulfill the same contract as Java EE platform so the application can function outside the container

Testing

- All of the examples that we have shown can run both inside and outside a Java EE platform container
 - Can do both unit and integration testing outside the container
 - No source code needs to be changed in order to run in Java SE platform
 - Can create JUnit tests with little or no additional framework or setup code
 - Configuration is externalized in XML files so multiple test configurations do not require multiple code bases

Example—Testing JPA Applications

Without Spring:

- Running JPA in Java SE platform has a different transaction model than in Java EE platform
 - Container-managed persistence contexts use JTA
 - Bootstrapped entity managers use resource-local transactions
- Typically use `@PersistenceContext` annotation to inject entity manager
 - Recognized by the container only, not the provider
 - Object must be managed in order to inject resources into it

Example—Testing JPA Applications

With Spring:

- Spring acts as a JPA host container
- Use container-managed persistence contexts in code
 - Can run the same code in Java SE platform or Java EE platform
 - Use `@PersistenceContext` annotation to inject EntityManagers
 - Can configure a Spring data source for use in Java SE platform
 - Inject the transaction manager according to the environment

Agenda

Introduction to Spring

Dependency Injection and IoC

Spring AOP

Portable Service Abstractions

Integration With Java EE Platform

Web Tier

Persistence Tier

Testing Enterprise Applications

Summary

Summary

- ✓ Java EE platform containers provide valuable core services for server-side applications
- ✓ Spring adds additional services that can be used in Java EE application environments
- ✓ Spring helps to make applications more portable across Java EE platforms
- ✓ Spring can make testing Java EE application environments outside a Java EE platform container much easier
- ✓ Containers can provide additional Spring features to achieve a completely seamless integration

For More Information

Technical Sessions

- Many Spring-related sessions at 2007 JavaOneSM Conference, including:
 - TS-7755: Advanced Spring Framework
 - TS-6821: Spring Web Flow: A Next-Generation Web Application Controller Technology
 - TS-7082: Building JavaServer Faces Applications With Spring and Hibernate
 - TS-4948: Unleashing the Power of JAX-WS RI: Spring, Stateful Web Services, SMTP, and More

For More Information

Resources

- ***“Using the Java Persistence API With Spring 2.0”***
Mike Keith, Rod Johnson
Java Developers Journal—May, 2007

Books

- ***Professional Java Development With the Spring Framework***
Rod Johnson, Juergen Hoeller, Aref Arendsen, Thomas Risberg, Colin Sampaleanu
Wrox 2005
- ***Pro EJB 3: Java Persistence API***
Mike Keith, Merrick Schincariol
Apress 2006



ORACLE



JavaOne

Harnessing the Power of Java™ Platform, Enterprise Edition (Java™ EE) Technology With Spring

Mike Keith

Oracle

<http://otn.oracle.com/oc4j>

Colin Sampaleanu

Interface21

<http://www.interface21.com>

<http://www.springframework.org>

TS-43350