Background

- Originally developed by Rod Johnson
- Addresses many problems of EJB
- One of the most popular Java web development frameworks
- Books
  - *Professional Java Development with the Spring Framework* (2005)

Spring Framework

- *Spring – Inversion of Control*
- Chengyu Sun
- California State University, Los Angeles
- Originally developed by Rod Johnson
- Addresses many problems of EJB framework
- One of the most popular Java web development frameworks
- Books
  - *Professional Java Development with the Spring Framework* (2005)

The Need for IoC

- The DAO Example
  - The Data Access Object (DAO) pattern
  - UserDao Example
    - Interface
    - Implementation
    - Usage in application code

Data Access Object (DAO)

- A Java EE design pattern
- Application code

DAO Interface

```java
public interface UserDao {
    User getUserById(Integer id);
    List getAllUsers();
}
```
DAO Implementation

**Implement UserDao using JPA**

```java
public class UserDaoImpl implements UserDao {
    private EntityManager entityManager;
    public User getUserById(Integer id) {
        return entityManager.find(User.class, id);
    }
    ...
}
```

DAO Usage in Application Code

**UserController**

```java
public class UserController {

    public String users(ModelMap models) {

        UserDao userDao;
        ...
        List<User> users = userDao.getAllUsers();
        ...
    }
}
```

Advantages of DAO

**Provide a data access API that is**
- Independent of *persistent storage types*, e.g. relational DB, OODB, XML flat files etc.
- Independent of *persistent storage implementations*, e.g. MySQL, PostgreSQL, Oracle etc.
- Independent of *data access implementations*, e.g. JDBC, Hibernate, etc.

Instantiate a UserDAO Object in Application Code

1. `UserDaoJpaImpl` userDao = new UserDaoJpaImpl();
2. `UserDao` userDao = new UserDaoJpaImpl();

```
Which one is better??
```

Problem Caused by Object Instantiation

**What if we decide to use JDBC instead of Hibernate/JPA, i.e. replace UserDaoJpaImpl with UserDaoJdbcImpl**
- The application is not really independent of the data access method
- Switching to a different UserDao implementation affects all the code that uses UserDao

Another Way to Instantiate UserDao

```java
UserDao userDao;
...
public void setUserDao(UserDao userDao) {
    this.userDao = userDao;
}
```

**No more dependency on a specific implementation of the DAO**
**But who will call the setter?**
Inversion of Control (IoC)

- A framework like Spring is responsible for instantiating the objects and pass them to application code
  - A.K.A. IoC container, bean container
- Inversion of Control (IoC)
  - The application code is no longer responsible for instantiating an interface with a specific implementation
  - A.K.A. Dependency Injection

Example: Hello World

- Message is a Java object (or bean) managed by the Spring container
  - Created by the container
  - Property is set by the container

Bean Configuration File

```xml
<beans>
  <bean id="msgBean"
       class="cs520.spring.hello.Message">
    <property name="message" value="Hello World!" />
  </bean>
</beans>
```

The string “Hello World” is injected to the bean msgBean

Understand Bean Container ...

- Without a bean container

```
new Message()
```

... Understand Bean Container

- With a bean container

```
new Message()
getBean("msgBean")
setMessage("Hello World")
```

Dependency Injection

- Objects that can be injected
  - Simple types: strings and numbers
  - Collection types: list, set, and maps
  - Other beans
- Methods of injection
  - via Setters
  - via Constructors
Dependency Injection Example

- DjBean
  - Fields of simple types
  - Fields of collection types
  - Fields of class types

Quick Summary of Bean Configuration

<table>
<thead>
<tr>
<th>Bean</th>
<th>&lt;bean&gt;, &quot;id&quot;, &quot;class&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple type property</td>
<td>&lt;property&gt;, &quot;name&quot;, &quot;value&quot;</td>
</tr>
<tr>
<td>Class type property</td>
<td>&lt;property&gt;, &quot;name&quot;, &quot;ref&quot; (to another &lt;bean&gt;)</td>
</tr>
<tr>
<td>Collection type property</td>
<td>&lt;list&gt;, &lt;set&gt;, &lt;map&gt;, &lt;props&gt;, &lt;value&gt;, &lt;ref&gt;, &lt;entry&gt;, &lt;prop&gt;</td>
</tr>
<tr>
<td>Constructor arguments</td>
<td>&lt;constructor-arg&gt;, &quot;index&quot;, same as other properties</td>
</tr>
</tbody>
</table>

Some Bean Configuration Examples

```xml
<property name="foo">
    <set>
        <value>bar1</value>
        <ref bean="bar2" />
    </set>
</property>

<property name="foo">
    <map>
        <entry key="key1">
            <value>bar1</value>
        </entry>
        <entry key="key2">
            <ref bean="bar2" />
        </entry>
    </map>
</property>

<property name="foo">
    <props>
        <prop key="key1">bar1</prop>
        <prop key="key2">bar2</prop>
    </props>
</property>
```

Wiring – The Stack Example (I)

```
StackTest
  Stack1
```

Wiring – The Stack Example (II)

```
StackTest
  Stack1
  Stack2
  ArrayList
  LinkedList
```

Wiring – The Stack Example (III)

```
StackTest
  Stack1
  Stack2
  ArrayList
  LinkedList
```
Annotation-based Configuration

- Activate annotation processing with `<context:annotation-config/>`
- Automatically scan for Spring bean with `<context:component-scan/>`
- Mark a class to be a Spring bean with `@Component`
- Enable auto wiring with `@Autowired`

---

XML Namespace ...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
>
  <context:annotation-config/>
  <component-scan base-package="cs520.spring_stack"/>
</beans>
```

---

... XML Namespace

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <annotation-config/>
  <component-scan base-package="cs520.spring_stack"/>
</bean:beans>
```

---

Component Scanning

- `@Component` for regular bean classes
- `@Repository` for DAO classes
- `@Controller` for controller classes
- `@Service` for service classes

---

Auto Wiring

- Auto wire types
  - `byName`, `byType`, `constructor`, `autodetect`
- For individual bean
  - `<bean autowire="" autowire-type"/>
- For all beans
  - `<beans default-autowire="" autowire-type"/>`
@Autowired

- The property does not need a setter
- Auto wired by type
- To auto wire by name
  - Use @Qualifier
  - Use @Resource

Advantages of IoC

- Separate application code from service implementation
- Centralized dependency management with a bean configuration file
- Singleton objects improve performance
  - Singleton vs. Prototype

Further Readings

- Spring in Action (3rd Ed)
  - Chapter 1-3
- Spring Framework Reference Documentation
    - Chapter 5 The IoC Container