CS520 Web Programming

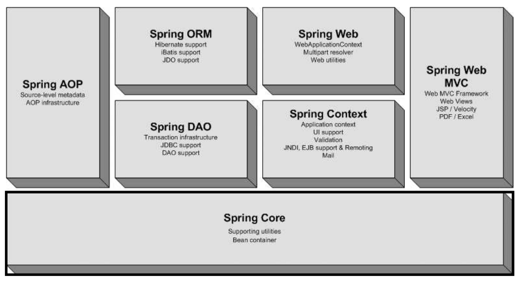
Spring – Inversion of Control

Chengyu Sun
California State University, Los Angeles

Background

- Originally developed by Rod Johnson
- Addresses many problems of EJB
  - Overly complex
  - Dated design
  - Hard to test
  - ...
- Described in *Expert One-on-One: J2EE Design and Development (2002)*

Spring Framework

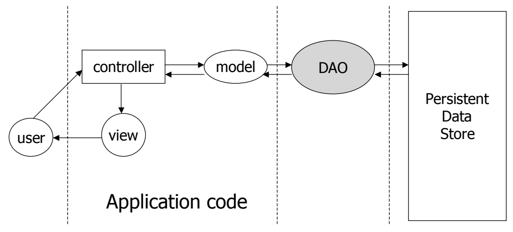


The Need for IoC

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

Data Access Object (DAO)

- A Java EE design pattern



UserDao in CSNS – Interface

```
public interface UserDao {

  public User getUserById( Integer id );
  public List getUsersById( Integer ids[] );
  public List getUsersByRoleName( String roleName );
  public User getUserByCin( String cin );
  public User getUserByName( String username );
  public User getUserByEmail( String email );
  public void saveUser( User user );

}
```

UserDao in CSNS – Implementation

◈Database access through Hibernate

```
public class UserDaoImpl
    extends HibernateDaoSupport
    implements UserDao {

  public User getUserById( Integer id )
  {
    return (User) getHibernateTemplate()
                    .get(User.class,id);
  }
  ... ...
}
```

UserDao in CSNS – Usage in Application Code

◈Used in more than twenty controllers, validators, and access decision voters
- Add instructor/student to class sections
- Validate whether a username is already used
- Check whether a user can access certain assignment or grade
- …

```
User instructor = userDao.getUserById( instructorId ) );
Section section = sectionDao.getSectionById( sectionId );

section.addInstructor( instructor );
sectionDao.saveSection( section );
```

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, JDO, etc.

Instantiate a UserDao Object in Application Code

1.
```
UserDaoHibernateImpl userDao =
      new UserDaoHibernateImpl();
```

2.
```
UserDao userDao =
      new UserDaoHibernateImpl();
```

Which one is better??

Problem Caused by Object Instantiation

◈What if we decide to use JDBC instead of Hibernate, i.e. replace `UserDaoHibernateImpl` 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

```
UsserDao 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 instantiate 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

```
<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`

Dependency Injection

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

Dependency Injection Example

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

Quick Summary of Bean Configuration

Bean	<bean>, “id”, “class”
Simple type property	<property>, “name”, “value”
Class type property	<property>, “name”, “ref” (to another <bean>)
Collection type property	<list>/<set>/<map>/<props>, <value>/<ref>/<entry>/<prop>
Constructor arguments	<constructor-arg>, “index”, same as other properties

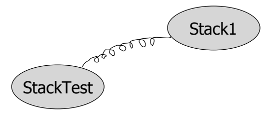
Some Bean Configuration Examples

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

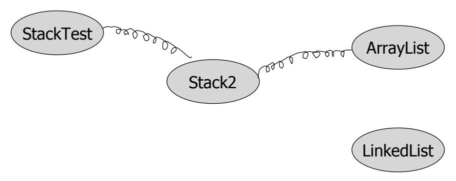
```
<property name="foo">
   <map>
      <entry key="key1">
         <value>bar1</value>
      </entry>
      <entry key="key2">
         <ref bean="bar2" />
      </entry>
   </map>
</property>
```

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

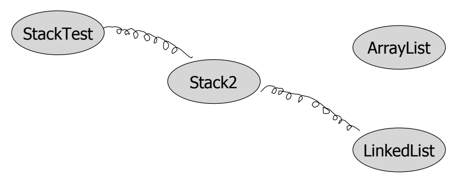
Wiring – The Stack Example (I)



Wiring – The Stack Example (II)



Wiring – The Stack Example (III)

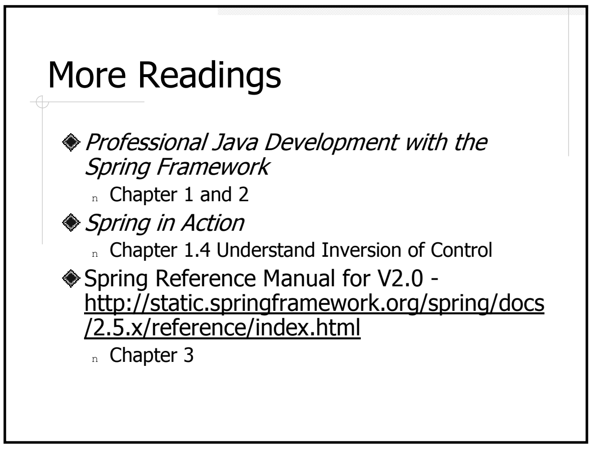


Auto Wiring

- <bean autowire="autowire type"/>
- <beans default-autowrire="autowire type">
- Auto wire types
  - byName
  - byType
  - constructor
  - autodetect

Advantages of IoC

- Separate application code from service implementation
- Centralized dependency management
- Singleton objects improve performance
  - *Singleton* vs. *Prototype*



More Readings

- *Professional Java Development with the Spring Framework*
  - Chapter 1 and 2
- *Spring in Action*
  - Chapter 1.4 Understand Inversion of Control
- Spring Reference Manual for V2.0 - http://static.springframework.org/spring/docs/2.5.x/reference/index.html
  - Chapter 3