CS520 Web Programming
Spring – MVC Framework

Chengyu Sun
California State University, Los Angeles

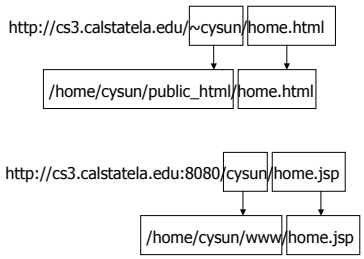
Roadmap

- **Request Processing**
- Controllers and validation
- Transactions and hibernate support
- Bits and pieces
  - Displaytag
  - Logging
  - Exception Handling
  - JSP pre-compilation
  - Application Deployment

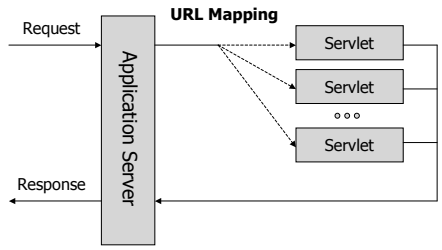
Understand Request Processing

- What happens when the server received a request like `http://sun.calstatela.edu/csns/instructor/home.html`

Direct Resource Mapping



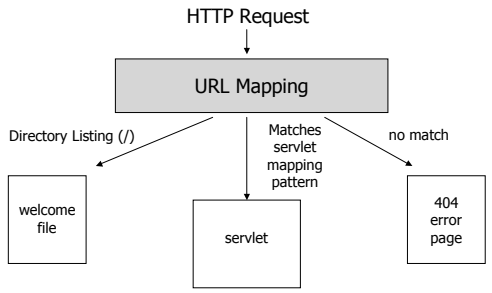
URL Mapping in a Java EE Application



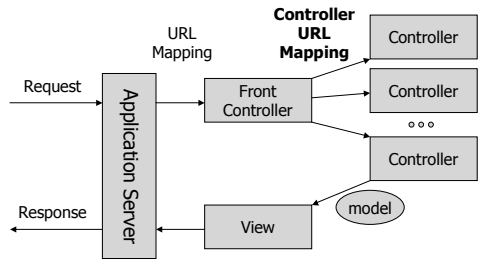
URL Mapping …

- Configured in web.xml
  - <servlet> and <servlet-mapping>
  - <welcome-file-list>
  - <error-page>
- Specified in the Servlet Specification

… URL Mapping



Request Processing in an MVC Framework



Spring Configuration File(s)

- <name>-servlet.xml
  - <name> must be the same as the <servlet-name> of the DispatcherServlet specified in web.xml
- Can have additional bean configurations files
  - E.g. csns-data.xml, csns-email.xml, csns-acegi.xml …

More About Configuration Files (Welcome to Metadata Hell!)

- Under classpath (/WEB-INF/classes)
  - hibernate.cfg.xml
  - ehcache.xml
  - *.properties
- Under /WEB-INF
  - web.xml
  - Spring configuration files
  - server-config.wsdd
- Under /META-INF
  - context.xml

Configuration Files in CSNS …

- All configuration files are under `/conf`
- `init` target in build.xml
  - Copy configuration files to the right places
  - Rename `spring-*.xml` to `${app.name}-*.xml`
  - Insert some parameter values into the configuration files

… Configuration Files in CSNS

- Advantages
  - One folder (i.e. `/conf`) for all metadata files
  - One file (i.e. `build.properties`) for all configurable parameters
  - Reusable for other applications
- Disadvantages
  - Non-standard
  - "Resource out of sync" error in Eclipse

Controller URL Mapping …

- Maps a URL pattern to a controller that will handle the request

BeanNameUrlHandlerMapping (default)

```
<bean name="/instructor/home.html"
      class="csns.spring.controller.instructor.ViewSectionsController">
```

… Controller URL Mapping …

SimpleUrlHandlerMapping

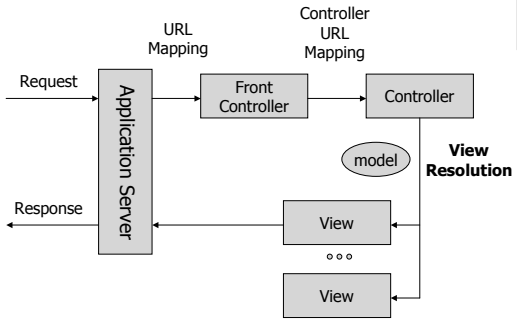
```
<bean id="home" class="csns.spring.controller.HomeController" />

<bean id="urlMapping"
  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
   <property name="mappings">
      <props>
         <prop key="/home.html">home</prop>
      </props>
   </property>
</bean>
```

… Controller URL Mapping

- More than one URL handler
  - <property name="order" value="0"/>
- No mapping found
  - 404 error

Request Processing in an MVC Framework



Model and View Examples

- AddInstructorController
- TakeSurveyController

ModelAndView

```
ModelAndView (
        String viewName,        ──→  Resolve to a view
        String modelName,       ──→  Attribute in Request
        Object modelObject      ──→  scope
)
```

```
ModelAndView( String viewName )
    .addObject( String modelName, String modelObject )
    .addObject( String modelName, String modelObject )
    ...
```

View Resolvers ...

- http://static.springframework.org/spring/docs/2.0.x/api/org/springframework/web/servlet/ViewResolver.html

... View Resolvers

- `InternalResourceViewResolver` for JSP
- Support for non-JSP view technologies
  - Velocity, FreeMarker, JsperReports, XSLT
- Views generated by Java classes
  - `BeanNameViewResolver`
  - `XmlViewResolver`
  - `ResourceBundleViewResolve`
- Multiple view resolvers
  - <property name="order" value="0"/>

InternalResourceViewResolver Example

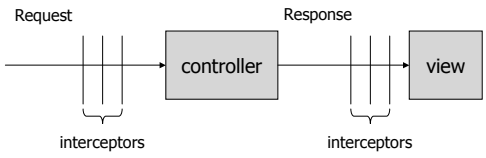
```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
</bean>
```

Prefix + ViewName + Suffix = JSP File

Special Views

- Redirect
  - new ModelAndView("**redirect:**" + url)
  - E.g. LogoutController
- Forward
  - new ModelAndView("**forward:**" + url)
- null
  - E.g. DownloadController

Interceptors



- A.K.A. *Handler Interceptors* (as oppose to Method Interceptors)
- Similar to Filters in J2EE specification

About Interceptors

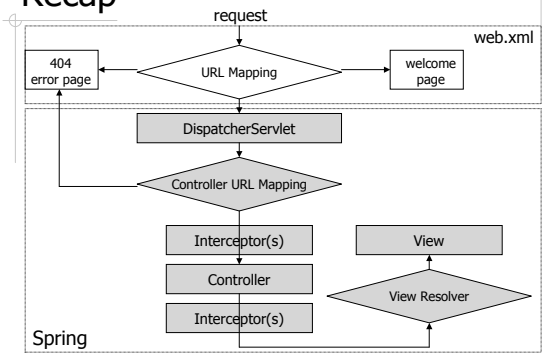
- Implement interceptors
  - `HnndlerInterceptor`
  - `HandlerInterceptorAdapter`
- Configure interceptors
  - In URL mapping bean

```
<bean id="urlMapping"
   class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
   <property name="interceptors">
      <list>
         <ref bean="openSessionInViewInterceptor" />
      </list>
   </property>
</bean>
```

Excercises

- csns/instructor/viewSubmissions.html?assignmentId=1234567

Recap



Roadmap

- Request Processing
- **Controllers and validation**
- Transactions and hibernate support
- Bits and pieces
  - Displaytag
  - Logging
  - Exception Handling
  - JSP pre-compilation
  - Application Deployment

Controller Interface

- org.springframework.web.servlet.mvc.Controller - http://static.springframework.org/spring/docs/2.0.x/api/org/springframework/web/servlet/mvc/Controller.html

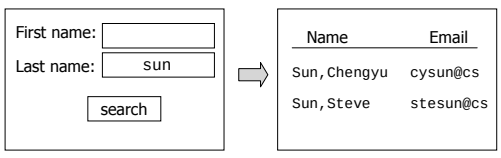
```
ModelAndView handleRequest (
     HttpServletRequest request,
     HttpServletResponse response )
```

Select A Controller

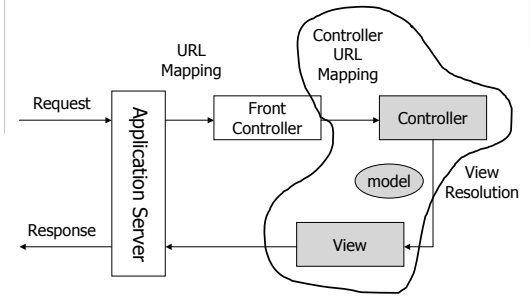
ParameterizableViewController	Simply display a view, i.e. the request does not need to be processed
Controller (interface) AbstractController	do not use request parameters or use only simple parameters
BaseCommandController AbstractCommandController	request parameters can be mapped to an object; can use *validators* to validate request parameters
AbstractFormController SimpleFormController	Handles form input
AbstractWizardFormController	Handles multi-page form input

Example: Find Email by Name

- Given the first name and/or the last name of a user, display the user's email address



So What Do We Need to Do?



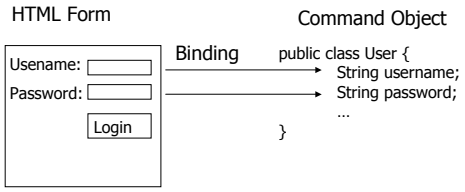
Models and Views

- Models
  - UserDao.getUsersByName() → List<User>
- Views
  - searchEmail.jsp
  - displayEmail.jsp

Controllers

- ParameterizableViewController + AbstractController
- SimpleFormController

Command Object



- Any class can be used as a command class in Spring
  - vs. ActionForm in Struts

Simple Form Handling

- The controller needs to handle two cases:
  - Display form
  - Process input data

Handle first request:

Create a command object and expose the object as a page scope variable

↓

Display the form view

Handle input:

Bind the request parameters to the command object

↓

Call controller.onSubmit()

Validation

- org.springframework .validation
  - Validator
  - Errors

Handle input:

Bind the request parameters to the command object

↓

Validator(s) —fail→ Form view

↓ success

Call controller.onSubmit()

messages.properties

- <name,value> pairs
- A single place for output messages
  - Easy to change
  - I18N
- Need to declare a `messageSource` bean in Spring configuration file
- Can be used by <fmt> tags in JSTL

Spring's `form` Tag Library

- Documentation - http://static.springframework.org/spring/docs/2.5.x/reference/view.html#view-jsp-formtaglib
- Tag reference – http://static.springframework.org/spring/docs/2.0.x/reference/spring-form.tld.html
- Example
  - admin/survey/addQuestion.jsp

Limitation of Spring Validation

- *Server-side only*
- Takes lots of coding for anything other than checking for empty/white spaces

Commons-Validator

- http://commons.apache.org/validator/
- Provide both *declarative* and *programmatic* validation
- Provide both client-side and server-side validation
  - Automatically generate cross-browser JavaScript code for client-side validation

Commons-Validator Declarative Validation Example

```
<form name="fooForm">

    <field property="name" depends="required">
        <arg0 key="fooForm.definition"/>
    </field>

    <field property="difficultyLevel"
           depends="required, integer">
        <arg0 key="fooForm.difficultyLevel"/>
    </field>

</form>
```

Commons-Validator Routines

- http://commons.apache.org/validator/api-1.3.1/org/apache/commons/validator/routines/package-summary.html
- Independent of the declarative validation framework
- A set of methods to validate
  - Date and time
  - Numeric values
  - Currency
  - ...

Use Commons-Validator with Spring

- Use the validation routines
- Use declarative validation
  - Spring Modules - https://springmodules.dev.java.net/

Recap

- Controller
  - Choose the right controller
  - Configure controller in Spring bean configuration file
- Validation
  - Validator
  - `message.properties` and Spring `form` tag library
  - Commons-Validator

Roadmap

- Request Processing
- Controllers and validation
- **Transactions and hibernate support**
- Bits and pieces
  - Displaytag
  - Logging
  - Exception Handling
  - JSP pre-compilation
  - Application Deployment

Programmatic vs. Declarative Transaction Management

Programmatic:

```
void saveUser( User u )
{
  transaction.start();
  …
  transaction.end();
}
```

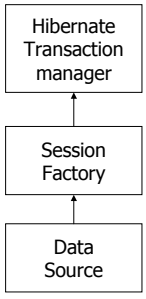
Declarative:

```
<transaction>
  <method>
    saveUser
  </method>
</transaction>
```

Spring Transaction Managers

- JDBC
- Hibernate
  - V3
  - Before V3
- JTA
- Object-Relational Bridge (ORB)

Configure Hibernate Transaction Manager



Transaction Manager in CSNS

- `/conf/spring-data.xml`

Connection Pooling

- Connection pooling
- DBCP
  - http://jakarta.apache.org/commons/dbcp/configuration.html

Configure Connection Pooling

- At the application server level
  - Provided through JNDI
  - E.g. http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html
- At the application level
  - E.g. <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" />

Add Transaction Support for DAO Classes

- TransactionProxyFactoryBean
  - target
  - transactionManager
  - transactionAttributeSource

```
class FooDao {

  void saveFoo( Foo foo )
  {
     save(foo);
  }
}
```

⇨

```
class FooDaoProxy {

  void saveFoo( Foo foo )
  {
     transaction.begin();
     save(foo);
     transaction.end();
  }
}
```

Transaction Attributes

- Isolation levels
- Read-only hints
- Transaction timeout period
- Method name patterns
- *Propagation behaviors*

Propagation Behaviors

- Determines whether the method should be run in a transaction, and if so, whether it should run within an existing transaction, a new transaction, or a nested transaction within an existing transaction.

The Need for Propagation Behaviors

```
Class FoobarDao {

  void saveFoo( Foo foo )  { save(foo); }

   void saveBar( Bar bar )  { save(bar); }

   void saveFoobar( Foobar foobar )
   {
      saveFoo( foobar.getFoo() );
      saveBar( foobar.getBar() );
    }

}
```

Propagation Behaviors in Spring

Propagation Behaviors	Run in …
PROPAGATION_MANDATORY	Existing transaction
PROPAGATION_NESTED	Nested transaction
PROPAGATION_NEVER	No existing transaction
PROPAGATION_NOT_SUPPORTED	Suspended during existing transaction
PROPAGATION_REQUIRED	Existing or new transaction
PROPAGATION_REQUIRES_NEW	New transaction
PROPAGATION_SUPPORTS	May run in existing transaction

Other Hibernate Support

- `HibernateDaoSupport` and `HibernateTemplate`
- `OpenSessionInViewFilter` and `OpenSessionInViewInterceptor`

HibernateTemplate

- http://static.springframework.org/spring/docs/2.0.x/api/org/springframework/orm/hibernate3/HibernateTemplate.html
- Examples in CSNS
  - CourseDaoImpl
  - UserDaoImpl
  - …

Access Hibernate Session Directly in HibernateTemplate

- getSessionFactory().getCurrentSession()
  - For versions of Hibernate before 3.0.1, the transactions are not managed by Spring if the session is obtained this way
- HibernateCallback
  - Object execute(HibernateCallback action)
  - List executeFind(HibernateCallback action)

Hibernate Callback Example

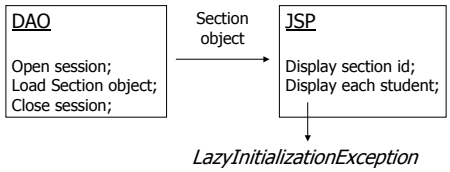
```
public Feedback getFeedback( final Integer itemId,
                             final Integer userId )
{
  return (Feedback) getHibernateTemplate().execute(
    new HibernateCallback() {

      public Object doInHibernate( Session session )
                  throws HibernateException, SQLException
      {
        String hql = "from Feedback f where "
            + "f.item.id = :itemId and f.user.id = :userId";
        Query query = session.createQuery( hql );
        query.setInteger( "itemId", itemId );
        query.setInteger( "userId", userId );
        return query.uniqueResult();
      }

  } );
}
```

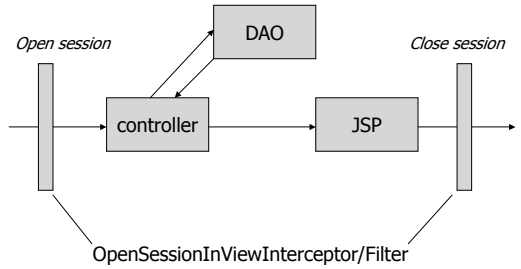
Hibernate LazyInitializationException

```
class Section {
    Integer id;
    Set<User> students;
}
```



LazyInitializationException

Open Session in View



Recap

- Configure Spring transaction support
- Write DAO classes
  - Extend `HibernateDaoSupport`
    - Use `HibernateTemplate`
    - Use `HibernateCallback`
- OpenSessionInView

Roadmap

- Web flow
- Controllers and validation
- Transactions and hibernate support
- **Bits and pieces**
  - Displaytag
  - Logging
  - Exception Handling
  - JSP pre-compilation
  - Application Deployment

Displaytag

- http://displaytag.sourceforge.net/
- Sortable columns and result paging
- displaytag vs. displaytag-el

Displaytag Examples

- instructor/viewSubmissions.jsp
- ...

<display:table>

- name
- requestURI
- pagesize
- uid
- class

<display:column>

- property
- sortable
- title
- sortProperty

Diaplaytag Properties

- http://displaytag.sourceforge.net/11/configuration.html
- displaytag.properties for the whole application
- <display:setProperty> for a particular table

Logging

- Use print statements to assist debugging
  - Why do we want to do that when we have GUI debugger??

```
public void foo()
{
   System.out.println( "loop started" );
   // some code that might get into infinite loop
   ...
   System.out.println( "loop finished" );
}
```

Requirements of Good Logging Tools

- Minimize performance penalty
- Support different log output
  - Console, file, database, …
- Support different message levels
  - Fatal, error, warn, info, debug, trace
- Easy configuration

Log4j and Commons-logging

- Log4j
  - A logging tool for Java
  - http://logging.apache.org/log4j/docs/
- Commons-logging
  - A wrapper around different logging implementations to provide a consistent API
  - http://jakarta.apache.org/commons/logging/

Log4j Example

- Message levels
- Output format

Log4j Configuration File

- *log4j.properties* or *log4j.xml*
- Appender
  - Output type
  - Output format
- Logger
  - Class
  - Message level

Log4j PatternLayout

- http://logging.apache.org/log4j/docs/api/org/apache/log4j/PatternLayout.html

Exception Handling

- An *Exception resolver* catches all exceptions thrown by controllers and chooses the proper view to display
- Examples
  - `SimpleMappingExceptionResolver`
  - `ExceptionResolver` in CSNS

JSP Pre-compilation

- Usually a JSP is converted to a servlet and then compiled into byte code *when the JSP is request for the first time*.
- JSP pre-compilation
  - Eliminate the "first request overhead"
  - Speed up development
- Tomcat provides a JSP pre-compiler which can be used as an ANT task
  - CSNS Example: `jspc` target in `build.xml`

Application Deployment

WAR Files

- Web application ARchive
- A JAR file for packaging, distributing, and deploying Java EE applications
- Create WAR files
  - The command line tool `jar`
  - Ant task `<war>`
    - CSNS Example: `war` target in build.xml

Deploy WAR Files to a Tomcat Server

- The Manager interface
- The `deploy` Ant task
  - CSNS Example: `deploy` target in build.xml