**Core Spring**

Autowiring with: type, name, constructor (not prefer)

Circular dependency

Bean definition override (just 2 separate files)

@DependsOn/depend-on

Bean naming in xml, java based, annotation based

Lazy initialization (rarely use cases or alter scenarios): @Lazy/lazy-init – careful

Bean callback method: <context:annotation-config /> / @AnnotationDrivenConfig

* @PostConstruct/init-method/@Bean(initMethodName)
* @PreDestroy/destroy-method/@Bean(destroyMethodName)

Define bean base on runtime environment (ex: in development phase and release phase use differnent dbms): @Profile/profile att, @Value

**Spring web**

Create project: use spring tool suite plugin

* Config pom to new version: spring, jdk
* Config project facet

DispatcherServlet, ViewResolver, <component-scan>, <annotation-driven>

@RequestMapping (/ - app folder (/localhost/webstore/)  
“/index” not is a subset of “/”

* Class level
* Method level: default RequestMapping method (just 1)

Config ApplicationContext, Web ApplicationContext

* web.xml vs xxx-servlet.xml
* web.xml – khai báo/config xxx-servlet.xml
* xxx-servlet.xml – beans, config web app (Dispatcher Servlet - can change name)
* Config view resolver, static content handling (main/resources)
* Configure bằng java: AbstractAnnotationConfigDispatcherServletInitializer
  + getServletMapping, getServletConfigClasses, getRootConfigClasses (non web component)
  + @Configuration, @EnableWebMvc
  + @ComponentScan
  + Extends WebMvcConfigAdapter
  + Config view resolver, static content handling (main/resources)

Spring web flow (page 140 – beginner guide)

Web app architecture best practice:

* presentation
* service (business – thường sử dụng transaction in this layer)
* domain model (entities schema– vs to database)
* repository

@Component, @Controller, @AdviceController, @Service, @Repository: auto detect bean, limit control (just declare)  
vs @Bean: explicit declare trong configuration class, tách declare bean khỏi class

In jsp: ${product.name} -> call getter

Tranfer data client <-> server

* Pass data client to server
  + Resquest object (like servlet - HttpServletRequest)
  + @RequestParam (request parameter – name attribute): default value
  + @PathVariable: more resource oriented
    - Multiple var
    - Regex with path variable
  + @ModelAttribute and form backing bean (auto biding giữa form filed and bean object)

Spring 3.2 (and upper) auto map request parameter bean (pass bean trực tiếp trong controller)  
<http://stackoverflow.com/questions/16942193/spring-mvc-complex-object-as-get-requestparam> (without @ModelAttribute) - use Spring taglib

* + JSON: @RequestBody (use mapper bean for get value: jackson.binding)

Có thể map cả file (for upload) với Multipart field

* Pass data server up to client
  + Response object (like servlet - HttpServletResponse)
  + Model/ModelMap/ModelAndView(Pass list to client: normal + use forEach tag (page 157: beginner guide)
  + JSON: @ResponseBody (how to get?: later)

Spring URI template pattern:

* @PathVariable - @RequestMapping(“/{category}/{otherinfo}/…”)
* @MatrixVariable: (xem sau – page 197: beginner guide)

Map vs Model

Test with mockMvc and mockito: xem sau

Apache common lang for equals and hashcode

Css/javascript should put in main/webapps/resources/, link = href=*"*<c:url value=*"/resources/style.css" />”*

Set id cho html element in <c:forEach> id=${spitter.id}

-> get được id to do many thing

Default form action: same url of form

return “redirect:/spitter/”: thường dùng trong post form

return “forward:/spitter/”

Validation

* Client: use jQuery: asynchronous
* Server
  + Simple: use javax validation api/setter

@Valid (yêu cầu Spring validate), Errors, BindingResult  
Hibernate validator

* + Complex: implements Validator (xem sau)

ViewResolver, View interface

* Nhiều ViewResolver implementation (jsp, jaspper, tiles…) tùy yêu cầu mà sử dụng
* Config
  + Java
  + Xml

Spring form handling: slight improvement (auto biding model) + hỗ trợ @Valid (14 tags)

* Utility
* Binding model
* Model: object (form backing bean) – field tương ứng filed in view – DTO?
* View
* Controller

commandName vs modelAttribute

binding object at get request hanlder method (model name match model attribute in form)

When the form is submitted, Spring automatically binds the form’s field values to the backing object in the model

<http://www.codejava.net/frameworks/spring/spring-mvc-form-handling-tutorial-and-example>

see: multiple form backing bean: Map or DTO

Can use type=”email…như html5

<errors>: nếu có khả năng ảnh hưởng tới layour -> dùng <form:errors path=”\*” element=”div” cssClass=”errors” />, change label of error field use <form:lable>, cssErrorClass

<form:errors path=*"\*"* element=*"div"* cssClass=*"errors"* />

form:label path=*"name"* cssErrorClass=*"errors"* >Name: </form:label>

<form:input path=*"name"* cssErrorClass=*"errors"*/>

div.errors, input.errors, label.errors

Create message with properties file {username.size}, {username.notnull}, thuận tiện change and multi language (chưa setup được)  
Config message file

* Xml
* java

Spring general tag (10) – (detail sau)

* message: internationalize
* url: có thể gán vào var

@SupressWarning

Apache tiles: xem sau

MVC more configurations: xem sau

Spring multipart file: xem sau

Controller handling exception:

* map exception to status code (some are default mapped) - 500 auto for all other exceptions
* mannual: @ResponseStatus(value, reason)
* catch exception in controller (try/catch trong controller) – not prefer
* thay bằng method mới với @ExceptionHandler (1 controller)
* how about handler exception from all controller -> using Advicing Controller: @ControllerAdvice (@ExceptionHanlder – all depth), limit controller class effect: backPackages

see more: AdviceController with @InitBinder, @ModelAttribute

return “redirect:/abc/xyz”

redirect with specific link

* normal: “redirect:/abc/” + username (dangerous: giống sql injection)
* use url template: “redirect:/abc/{username}” (use model)
* flash attribute: ex: sau khi post redirect to profile page – bình thường thì ko truyền được data khi redirect (coi như request mới): use RedirectAttribute type with addFlashAttribute (alive cho đến hết next request) (page 217)

Spring Web Flow (xem sau)

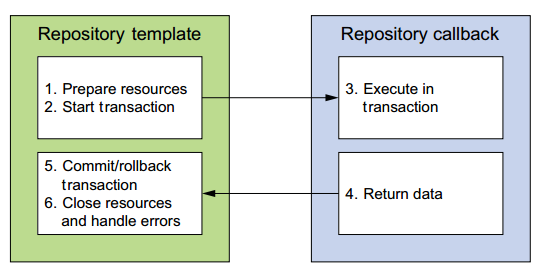
Security web app (xem sau)

**Spring Relational Database**

* normal jdbc
* jdbc with template
* orm: Hibernate (native), jpa (jpa vendor: Hibernate, EclipseLink…) iBatis, spring data jpa

Spring JDBC Exception: More specific than JDBC Exception, all unchecked exception (no need try/catch), DataAccessException

***Spring Data-access Template***: template (fixed part: resources manage, transaction…) and callback (custom part: execute query, handle result), use template method pattern



Có nhiều loại template: JdbcTemplate, HibernateTemplate, JpaTemplate…

Config datasource (thay vì connect bằng class DatabaseUtils – phải tự handle everything) – tất cả đều qua JDBC Driver Connector: establish connection through database throud data source.

* via JNDI (prefer) – try sau
* via pooled data source (use 3rd party implementation of connection pool: Apache DBCP, c3p0, BoneCP) – using pool connection for better performance (init/max/maxidle connections) – to reuse connection
* Use JDBC driver-based datasource (for small app and development environment) – vì cost to create and release Connection is expensive.
  + DriverManagerDataSource (mỗi request 1 connection)
  + SimpleDriverDataSource (mỗi request 1 connection)
  + SingleConnectionDataSource (1 connection for all request -> wait)
* Use embedded datasource (for development and testing, reset khi reset app)
* Use profiles to select datasource: choose different datasource for different environment -> turn on profile at runtime (see more: spring bean profile)

*Use JDBC template* with Spring (vs other persistance frameworks: Hibernate, Spring JPA, iBatis…): JDBC characteristics

* directly, improve sql performance by improve sql query, full control
* not flexibility + must handle every thing (exception, transaction, manage data resource (stmt, conn…)), long (boilerplate) -> use JdbcTemplate: chỉ cần làm: connection info + sql statement.

JDBCTemplate, NamedParameterJdbcTemplate, SimpleJdbcTemplate (deprecated).

Datasource class: util class to provide connection to database (bean with driver, username…info)

JdbcTemplate, JdbcOperations, RowMapper

JdbcTemplate methods: update(sql, object), single select: queryForObject(sql, object, rowMapper), total select: queryForInt(sql), batchUpdate

<http://examples.javacodegeeks.com/enterprise-java/spring/jdbc/spring-jdbctemplate-example/>

Java 8 lambdas with JdbcTemplate (later)

Using named parameter (vs indexed parameter): NamedParameterJdbcTemplate (map name -> ko cần thứ tự params – dùng map để put value theo name)

***Spring and ORM:*** map object (filed) to table (column)

ORM framework

Lazy loading (like: list PurchaseOrder, each include Image filed), eager fetching (thường default), cascading -> focus on business over database (and transparent exception handling, declarative transaction, resource management…)

***Hibernate (native vs*** *jpa**Hibernate vendor)****:***

* Session: basic persistance operation - crud
* SessionFactory interface: for create session
* SessioFactory implementations: LocalSessionFactoryBean (3/4), AnnotationSessionFactoryBean ( chose phụ thuộc vào version or mapping config type: xml/annotation)

***JPA: (****with specific vendors implementation)*

* EntityManager
* EntityManagerFactory: application managed vs container managed
* LocalEntityManagerFactoryBean, LocalContainerEntityManageFactoryBean
* Configure: do example later
* @Transactional

***Spring Data JPA:***

* extends JpaRepository<Spitter, Long>
* <jpa:repositories>/@EnabelJpaRepositories -> auto create implementations with pre-implement method (about 18)
* Add custom methods: verb + [Distinct] + [Subject] + By + Predicate
  + Verb: get/read/find/count
  + Subject: just more readable – any thing is ok here (return type is defined in class defination …extends JpaRepository<Type…>)
  + Predicate: page 322
* Custom Query: @Query

Example: <http://www.petrikainulainen.net/spring-data-jpa-tutorial/>

***Mixing mode (Spring data jpa and normal jpa):*** Implementation with postfix: Impl (SpitterRepositoryImpl – có thể set bằng config) will find and merge to method in interface.

**Spring NoSQL:**

**Spring transaction**

**Spring Security**

**Spring Batchs**

**Spring Integration**