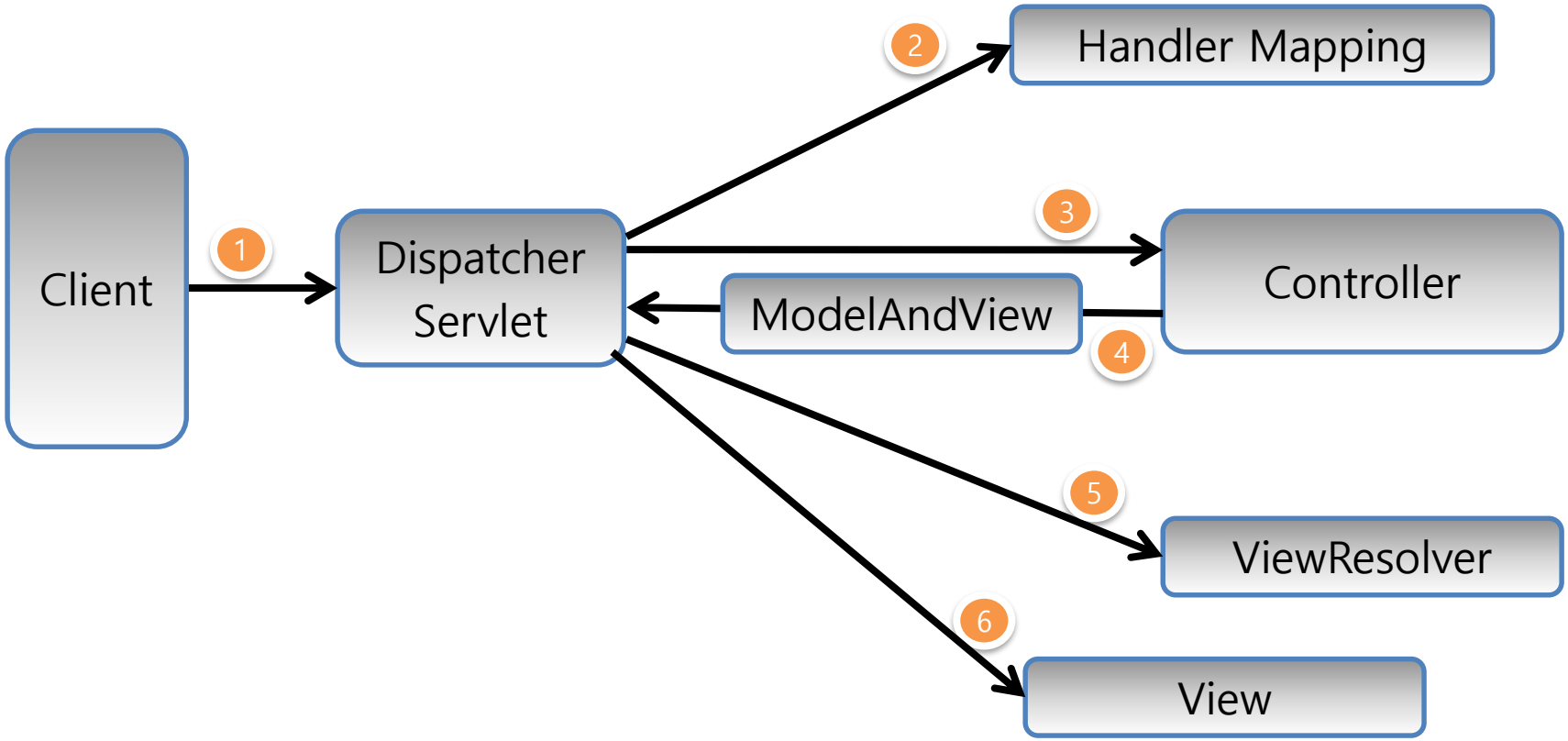
Spring MVC

Spring MVC 흐름 (1/2)

- Spring MVC
  - MVC 패턴 기반 웹 개발 프레임워크

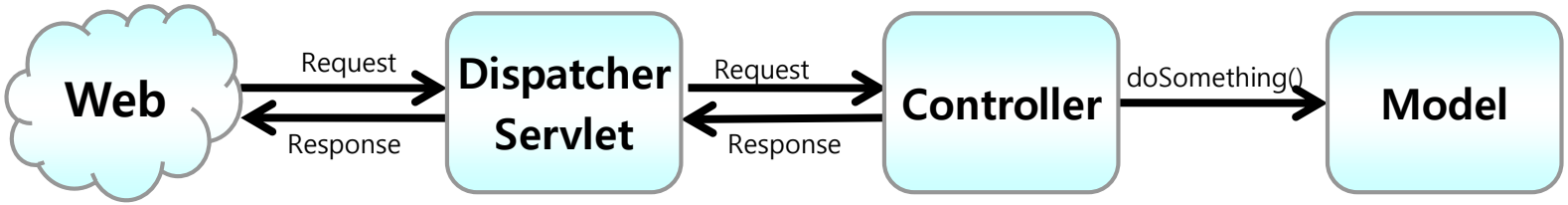


Spring MVC 흐름 (2/2)

- 요청 처리 순서
  - ① DispatcherServlet이 요청을 수신
    - 단일 front controller servlet
    - 요청을 수신하여 처리를 다른 컴포넌트에 위임
    - 어느 컨트롤러에 요청을 전송할 지 결정
  - ② DispatcherServlet은 HandlerMapping에 어느 컨트롤러를 사용할 것인지 문의
    - URL과 매핑
  - ③ DispatcherServlet은 요청을 컨트롤러에게 전송하고 컨트롤러는 요청을 처리한 후 결과 리턴
    - 비즈니스 로직 수행 후 결과 정보(Model)가 생성되어 JSP와 같은 뷰에서 사용 됨
  - ④ ModelAndView 오브젝트에 수행결과가 포함되어 DispatcherServlet에 리턴
  - ⑤ ModelAndView는 실제 JSP정보를 갖고 있지 않으며, ViewResolver가 논리적 이름을 실제 JSP 이름으로 변환
  - ⑥ View는 결과 정보를 사용하여 화면을 표현함.

Spring MVC 구현 Step

- Spring MVC를 이용한 어플리케이션 작성 스텝
  1. web.xml에 DispacherServlet 등록 및 Spring설정 파일 등록
  2. 설정파일에 HandlerMapping 설정
  3. 컨트롤러 구현 및 Spring 설정파일에 등록
  4. 컨트롤러와 JSP의 연결 위해 View Resolver 설정
  5. JSP 코드 작성



DispatcherServlet 설정과 ApplicationContext (1/3)

- DispatcherServlet 설정
  - web.xml에 등록
  - 스프링 설정파일 : "<servlet-name>-servlet.xml" 이고, WEB-INF₩아래 추가한다.
  - <url-pattern>은 DispatcherServlet이 처리하는 URL 매핑 패턴을 정의

```
<servlet>
   <servlet-name>dispatcher</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
   <servlet-name>dispatcher</servlet-name>
   <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

- **Spring Container**는 설정파일의 내용을 읽어 **ApplicationContext** 객체를 생성한다**.**
- 설정 파일명 : dispatcher-servlet.xml – MVC 구성 요소 (HandlerMapping, Controller, ViewResolver, View) 설정과 bean, aop 설정들을 한다.

DispatcherServlet 설정과 ApplicationContext (2/3)

- Spring 설정파일 등록하기
  - <servlet>의 하위태그인 <init-param>에 contextConfigLocation 이름으로 등록
  - 경로는 Application Root부터 절대 경로로 표시
  - 여러 개의 경우 , 또는 공백으로 구분

```
<servlet>
   <servlet-name>dispatcher</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
   <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/server-service.xml, dao-service.xml</param-value>
     </init-param>
</servlet>
```

DispatcherServlet 설정과 ApplicationContext (3/3)

- DispatcherServlet 여러 개 설정 시 공통 Spring 설정파일 등록
  - 컨텍스트 설정 파일(스프링 설정파일)들을 로드하기 위해 리스너 (ContextLoaderListener) 설정
  - 설정파일 <context-param>으로 등록

```
<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/service-service.xml
                              /WEB-INF/dao-data.xml
        </param-value>
</context-param>
```

HandlerMapping

- Client요청과 Controller를 연결을 설정
- 다양한 HandlerMapping 클래스를 Springframework가 제공 하며 Spring 설정파일에 <bean name="HandlerMapping">으로 등록하여 설정한다.
- BeanNameUrlHandlerMapping
  - bean의 이름과 url을 mapping
- SimpleUrlHandlerMapping
  - url pattern들을 properties로 등록해 처리

HandlerMapping

BeanNameUrlHandlerMapping 설정

```
<bean id="HandlerMapping"
            class="org.springframework.web.servlet.Handler.BeanNameUrlHandlerMapping"/>
<bean name="/hello.do" class="controller.HelloController"/>
<bean name="/welcome.do" class="controller.WelcomeController"/>
```

SimpleUrlHandlerMapping 설정

```
<bean id="HandlerMapping"
        class="org.springframework.web.servlet.Handler.SimpleUrlHandlerMapping">
<property name="mappings">
        <props>
                <prop key="/register.do">registerContoller</prop>
                <prop key="/delete.do">deleteController</prop>
         </props>
</property>
</bean>
<!--컨트롤러 bean으로 등록-->
<bean name="registerContoller" ...../>
<bean name="deleteController" ..../>
```

Controller 작성

- Controller 종류
  - Controller (interface)
  - AbstractController
  - AbstractComandController
  - MultiActionController

- 위의 interface/class를 상속하여 Controller 작성한다.

AbstractController (1/2)

- 가장 기본이 되는 Controller
- 작성
  - AbstractController 상속한다.
  - public ModelAndView HandlerequestInternal
    (HttpServletRequest request,
    HttpServletResponse response)
    throws Exception
    오버라이딩 하여 코드 구현
  - ModelAndView에 view가 사용 할 객체와 view에 대한 id 값을 넣어 생성 후 return

AbstractController (2/2)

```
public class HelloworldAbstractController extends AbstractController{
        protected ModelAndView HandlerequestInternal(
                                        HttpServletRequest request,
                                        HttpServletResponse response)
                                        throws Exception {
            //Model 호출 – Business Logic 처리
            //ModelAndView를 통해 view로 수행 넘김
            return new ModelAndView("hello","message",helloVO);

        }
}
```

AbstractCommandController (1/2)

- HttpServletRequest로 받아온 form data(parameter)를 동적으로 특정 데이터 객체(VO-Command지칭)로 바인드 하는 controller.
- Struts의 ActionForm의 기능과 유사
  - Spring의 경우 Strutrs 처럼 데이터객체가 framework-specific 인터페이스를 implement하지 않아도 된다.
  - 생성된 Command 객체는 requestScope에 속성으로 binding된다.
- 작성
  - 데이터 객체는 java beans 규약에 맞게 생성 (setter). parameter의 이름과 데이터객체의 property이름이 동일해야 한다.
  - AbstractCommandController를 상속하여 클래스 작성
  - Data 객체에 설정을 위해 아래 두 메소드 생성자에서 호출한다.
    - setCommandClass(Class commandClass) : Command클래스
    - setCommandName(String commandName) : requestScope에 Command 객체를 넣을 때 사용할 이름, 생략가능
    - protected ModelAndView handle(HttpServletRequest request,
      HttpServletResponse response,
      Object command, BindException be)
      throws Exception{}

      메소드를 overriding하여 구현

AbstractCommandController (2/2)

Command class(VO)

```
public class PersonCommand{
    private String id;
    private String name;
    ....
    setter/getter
}
```

Controller (VO)

```
public class RegisterController extends AbstractCommandController{
    public RegisterController(){
        setCommandClass(PersonCommand.class);
        setCommandName("personCommand");
    }
     protected ModelAndView handle(HttpServletRequest req, HttpServletResponse res,
                                                    Object command, BindException error){
            PersonCommand cmd = (PersonCommand)command;
                //Business Logic 처리
                ModelAndView mv = new ModelAndView("show_content");
            return mv;
     }
}
```

MultiActionController (1/3)

- 하나의 Controller에서 여러 개의 요청을 수행할 경우 사용
  - struts의 DispatcherAction과 동일한 역할
- 연관 된 request를 하나의 controller로 묶을 경우 사용.
- 작성
  - MultiActionController 상속
  - client의 요청을 처리할 메소드 구현

    public [ModelAndView|Map|void] 메소드이름(
    HttpServletRequest req, HttpServletResponse res
    [HttpSession|Command]) [throws Exception]{}
    - return type : ModelAndView, Map, void 중 하나
    - argument :

      1번 - HttpServletRequest, 2번 - HttpServletResponse

      3번 – 선택적이며 HttpSession 또는 Command

MultiActionController (2/3)

- MethodNameResolver 등록
  - 역할 : 어떤 메소드가 클라이언트의 요청을 처리할 것인지 결정
  - Spring 설정파일에 <bean>으로 등록
  - controller에서는 property로 주입 받는다.
  - 종류
    - ParameterMethodNameResolver : parameter로 메소드 이름 전송
    - InternalPathMethodNameResolver : url 마지막 경로 메소드 이름으로 사용
    - PropertiesMethodNameResolver : URL과 메소드 이름 mapping을 property로 설정

MultiActionController (3/3)

Controller class

```
public class MemberController extends MultiActionController{

        public ModelAndView registerMember(HttpServletRequest request,
                                           HttpServletResponse response) throws Exception{
            //Business Logic 구현
            ModelAndView mv = new ModelAndView();
            mv.setViewName("register_ok");
            return new ModelAndView();
        }
}
```

```
<bean id="methodNameResolver"
        class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">
        <property name="paramName" value="mode"></property>
</bean>

<bean id="memberController" class="controller.multiaction.MemberController">
            <property name="methodNameResolver">
                        <ref bean="methodNameResolver"/>
            </property>
</bean>
```

호출 : http://ip:port/applName/member?mode=registerMember

ModelAndView (1/2)

- Controller 처리 결과 후 응답 할 view와 veiw에 전달 할 값을 저장.
- 생성자
  - ModelAndView(String viewName) : 응답 할 view설정
  - ModelAndView(String viewName, Map values) : 응답 할 view와 view로 전달 할 값들을 저장한 Map 객체
  - ModelAndView(String viewName, String name, Object value) : 응답 할 view 이름, view로 넘길 객체의 name-value
- 주요 메소드
  - setViewName(String view) : 응답 할 view 이름을 설정
  - addObject(String name, Object value) : view에 전달 할 값을 설정 – requestScope에 설정됨
  - addAllObjects(Map values) : view에 전달 할 값을 Map에 name-value로 저장하여 한번에 설정 - requestScope에 설정됨
- Redirect 방식 전송
  - view이름에 redirect: 접두어 붙인다.
    ex) mv.setViewName("redirect:/welcome.html");

ModelAndView (2/2)

```
protected ModelAndView HandlerequestInternal(HttpServletRequest req,
                                    HttpServletResponse res) throws Exception{
        //Business Logic 처리
        ModelAndView mv = new ModelAndView();
        mv.setViewName("/hello.jsp");
        mv.addObject("greeting", "hello world");
        return mv;
}
```

ViewResolver

- Controller가 넘긴 view이름을 통해 알맞은 view를 찾는 역할
  1. Controller는 ModelAndView 객체에 응답 할 view이름을 넣어 return.
  2. DispatchServlet은 ViewResolver에게 응답 할 view를 요청한다.
  3. ViewResolver는 View 이름을 이용해 알맞는 view 객체를 찾는다.
- 다양한 ViewResolver를 SpringFramework는 제공한다.
  - InternalResourceViewResolver : 뷰의 이름을 JSP, HTML등과 연동한 View를 return
- ViewResolver – Spring 설정파일에 등록한다.

ViewResolver

Spring 설정파일에 설정

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

Controller

```
ModelAndView mv = new ModelAndView();
mv.setViewName("hello");
```

위의 경우
/WEB-INF/jsp/hello.jsp 를 찾는다.