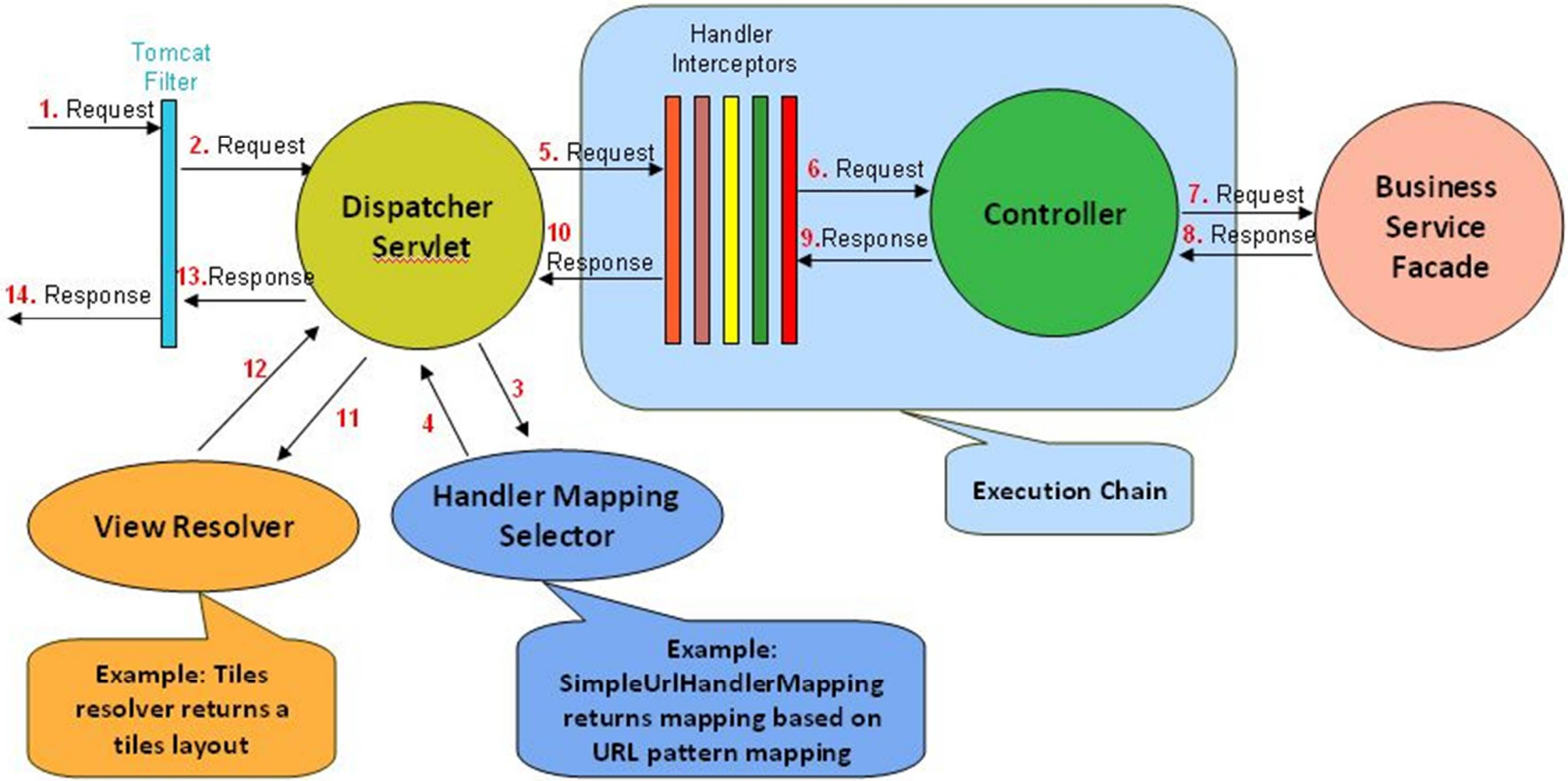
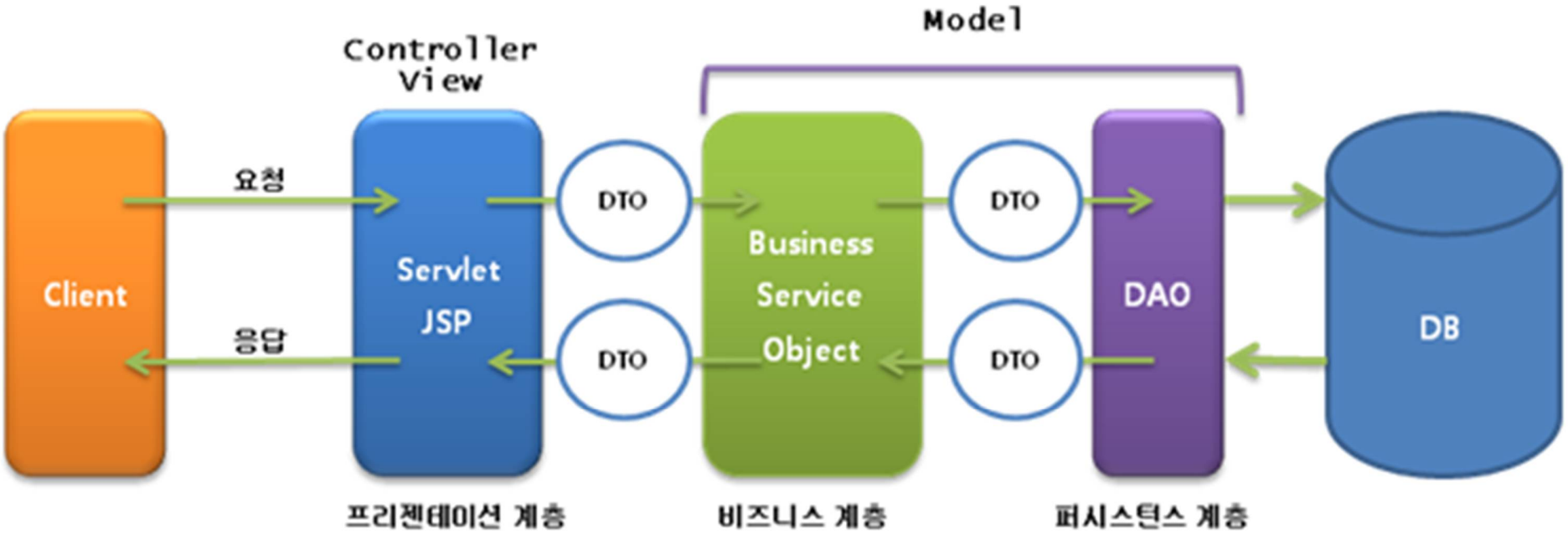
st09.스프링은 웹 MVC 프레임워크를 제공한다.docx

목차

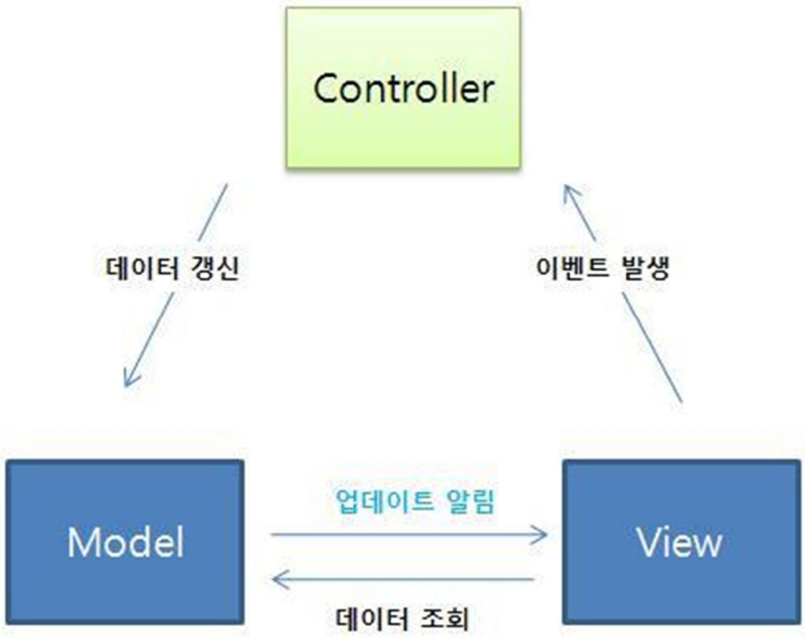

Tomcat
Filter
1. Request
2. Request
Dispatcher
Servlet
5. Request
Handler
Interceptors
6. Request
Controller
7. Request
Business
Service
Facade
8. Response
9.Response
10
Response
13.Response
14. Response
12
11
4
3
View Resolver
Handler Mapping
Selector
Execution Chain
Example: Tiles
resolver returns a
tiles layout
Example:
SimpleUrlHandlerMapping
returns mapping based on
URL pattern mapping


Model
Controller
View
Client
요청
응답
Servlet
JSP
DTO
DTO
Business
Service
Object
DTO
DTO
DAO
DB
프리젠테이션 계층
비즈니스 계층
퍼시스턴스 계층

1. MVC 패턴

1.1 MVC 패턴이란?

'MVC(Model View Controller)'란 비즈니스 규칙을 표현하는 도메인 **모델(Model)**과 프레젠테이션을 표현하는 **View** 를 분리하고 양측 사이에 컨트롤러(**Controller**)를 배치하도록 설계한 디자인 패턴이다.



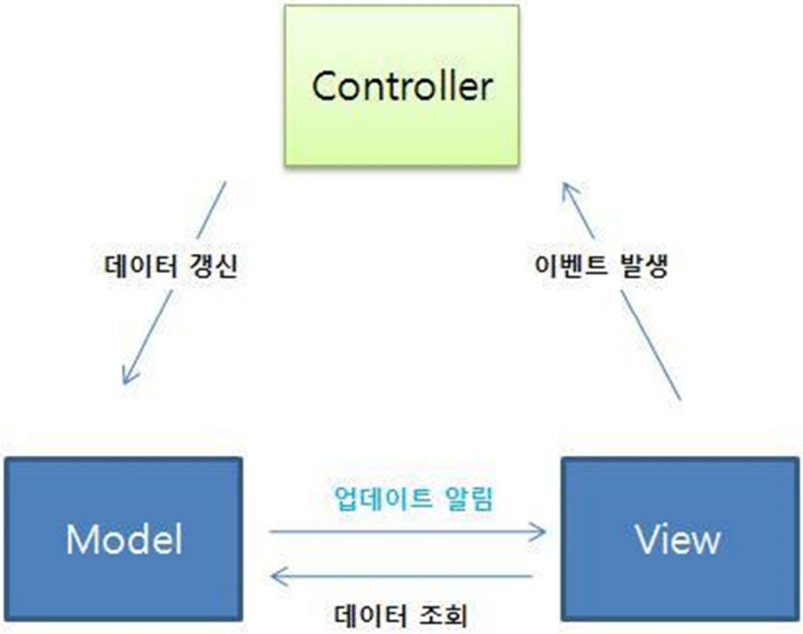
'스프링 '이란 웹 어플리케이션용 MVC 프레임워크이다. 스프링 MVC 를 이용함으로써 웹 어플리케이션의 Model, View, Controller 사이의 의존 관계를 DI 컨테이너에서 관리하고 통일된 유연한 웹 어플리케이션을 구축할 수 있다.

1.2 MVC 패턴의 등장 배경

-표현계층과 데이터(처리로직)를 분리하여 데이터 러치 로직이 중복 코딩되는 것을 막고 로직과 엔티티(데이터)를 재사용하는데 목적

2. MVC 패턴의 구성도 및 구성요소

2.1 MVC 의 구성도



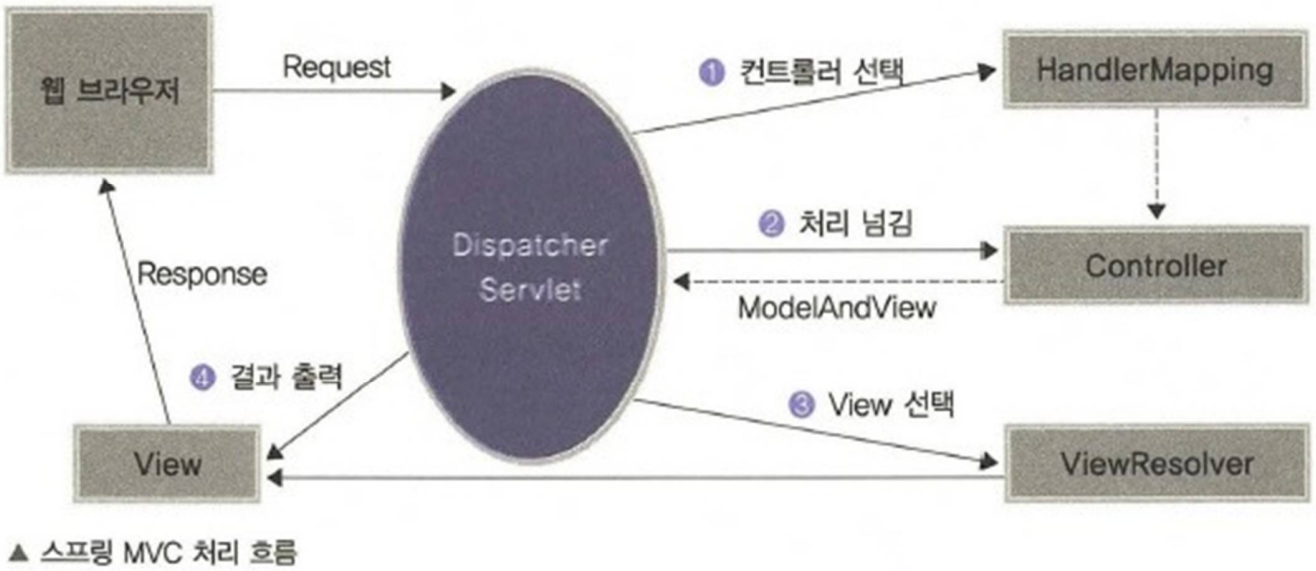
2.2 MVC 패턴의 구성요소

구분	내용
Model	사용자에게 보여줄 데이터
View	모델 데이터를 표시할 JSP 페이지
Controller	사용자의 입력 및 흐름 제어를 담당

3. 스프링 MVC 처리 흐름도

스프링 MVC 는 Front Controller 패턴을 채용하고 있다. FrontController 패턴이란 핸들러 오브젝트를 매개로 하여 요청을 분배함으로써 요청을 통합하고, 요청에 대한 통일된 처리를 기술 할 수 있도록 하기 위한 패턴이다.

브라우저로부터 송신된 요청은 모두 스프링 MVC DispatherServlet 클래스에 의해 관리되고 있다.



▲ 스프링 MVC 처리 흐름

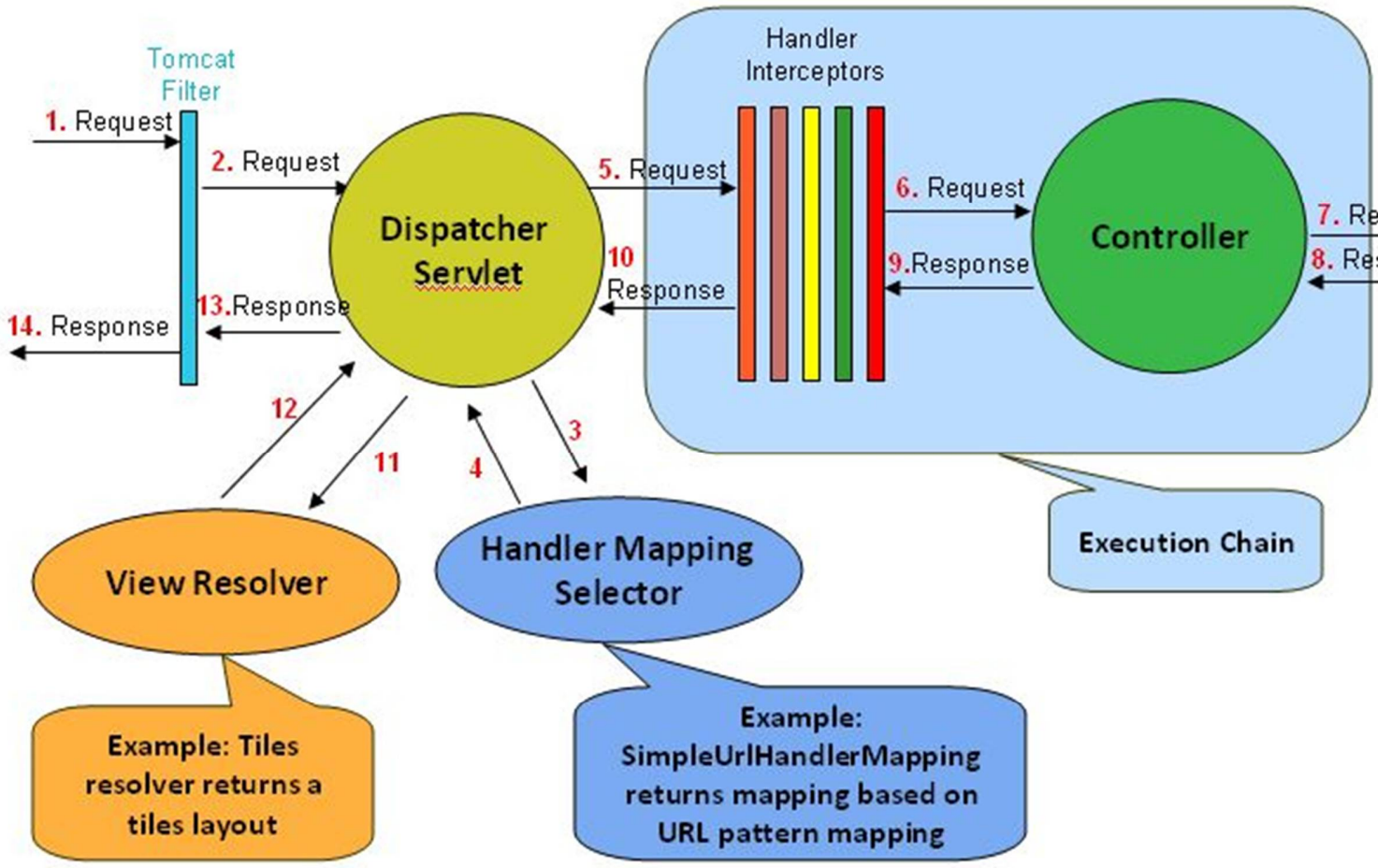
웹 브라우저의 요청(Request)은 **DispatcherServlet** 인스턴스로 송신된다. 요청을 받은 DispatcherServlet 인스턴스는 **Controller** 인스턴스를 호출한다. 이 때 DispatcherServlet 는 지정된 RequestURL 을 이용하여 Controller 를 찾게 되는데 그것은 **HanadlerMapping** 이다

HanadlerMapping 은 RequestURL 과 Controller 의 맵핑해 주는 역할을 하는 객체다. DispatcherSerץ let 은 HandlerMapping 가 반환한 Controller 를 이용하여 Controller 의 비즈니스 로직을 호출하고 **ModeAndView** 정보를 반환 받는다. 반환된 ModeAndView 에는 처리된 결과, 즉 **Model** 정보와 이동할 **View** 정보가 들어 있게 된다.

마지막으로 DispatcherServlet 는 ModeAndView 의 View 정보와 ViewResolver 를 이용하여 View 를 찾고 Model 을 View 에 rendering 하여 처리 결과를 브라우저에 표시하게 된다.

HandlerMapping, Controller, ViewResolver, View 는 모두 스프링 MVC 가 제공하는 인터페이스이다.

통상 개발에서는 Controller 를 제외한 HandlerlMapping, Viewresolver, View 인터페이스의 구현 클래스는 스프링 MVC 에서 제공히는 구현 클래스를 활용하면 된다.


Tomcat
Filter
1. Request
2. Request
Dispatcher
Servlet
5. Request
Handler
Interceptors
6. Request
Controller
7. Re
8. Re
9.Response
10
Response
13.Response
14. Response
12
11
4
3
View Resolver
Handler Mapping
Selector
Execution Chain
Example: Tiles
resolver returns a
tiles layout
Example:
SimpleUrlHandlerMapping
returns mapping based on
URL pattern mapping

3.1 DispatcherServlet

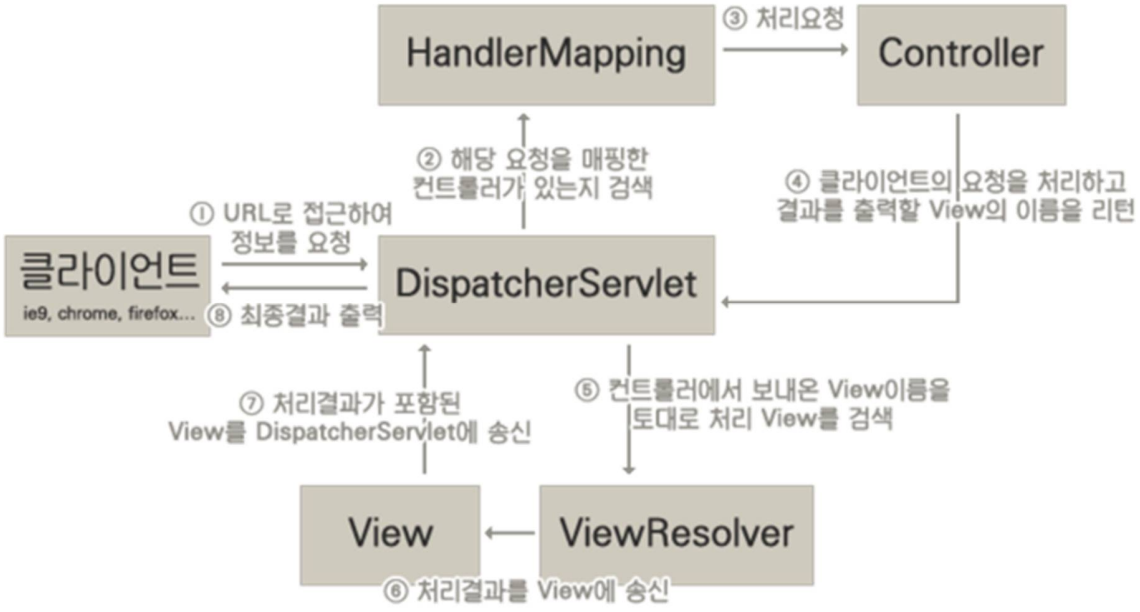
DispatherServlet 는 웹 브라우저로부터 송신된 Request 를 일괄적으로 관리한다 .
웹 브라우저로부터 요청은 DispatherServlet 인스턴스로 송신된다.

- DispatherServlet 은 클라이언트의 요청을 중앙에서 처리하는 스프링 MVC 의 핵심 구성 요소이다.
- web.xml 파일에 한 개 이상의 DispatherServlet 을 설정할 수 있으며, 각 DispatherServle 은 한 개의 WebApplicationContext 를 갖게 된다.
- 각 DispatherServlet 이 공유할 수 있는 빈을 설정할 수도 있다.

DispatherServlet 과 웹 어플리케이션을 위한 설정 방법 및 둘 사이의 관계에 대해서 살펴보자.

- DispatcherServlet 란 도데체 무엇일까?

우리가 각각 분리하여 만든 Model, Controller 그리고 View 를 조합하여 브라우저로 출력해주는 역할을 수행하는 오브젝트라 할 수 있읍니다.



간단하게 DispatcherServlet 이 어떤 식으로 클라이언트의 요청을 처리하고 응답하는지 UML 과 비슷한 방식으로 나타내 보았습니다.

위의 작업흐름을 풀어 자세히 설명하자면 다음과 같습니다.

① 클라이언트가 해당 어플리케이션에 접근하면 접근한 URL 요청을 DispatcherServlet 이 가로챕니다. 이렇게 요청을 가로챌 수 있는 이유는 web.xml 에 등록된 DispatcherServlet 의 <url-pattern>이 '/'와 같이 해당 어플리케이션의 모든 URL 로 등록 돼있기 때문입니다. 만약 특정 URL 만 적용하고 싶다면 <url-pattern>의 내용을 바꿔주어 범위를 변경 시키주면 됩니다.

② 가로챈 URL 정보를 HandlerMapping 에게 보내 해당 요청을 처리할 수 있는 Controller 를 찾아냅니다. (스프링은 기본적으로 5 가지의 핸들러 매핑을 제공합니다.) 이 부분은 스프링의 디폴트 전략에 의해 BeanNameUrlHandlerMapping 과 DefaultAnnotationHandlerMapping 이 기본으로 스프링 MVC 에 탑재되있기 때문에 특별한 경우가 아니라면 따로 설정할 필요가 없습니다.

③ 핸들러 매핑이 해당 요청을 처리할 컨트롤러를 찾아냈다면 요청을 컨트롤러에 보내줍니다. 컨트롤러는 사용자가 직접 구현해주는 부분입니다. @MVC 는 매우 다양한 코딩방식과 직관적이고 편리한 컨트롤러 작성방법을 제공하므로 이 부분에 대해서는 차후 심층적인 분석하여 자신에게 알맞는 전략을 선정해야 합니다.

④ 컨트롤러를 해당 요청을 처리한 후에 보통 컨트롤러는 요청을 응답받을 View 의 이름을 리턴하게 됩니다. (물론 다른 핸들러 매핑 전략을 이용한다면 응답 과정이 다를 수도 있습니다.) 그 때 이 이름을 ViwResolver 가 먼저 받아 해당하는 View 가 존재하는지 검색합니다.

⑤ 해당 View 가 있다면 처리결과를 View 에 보낸 후 ⑦ 이 결과를 다시 DispatcherServier 에 보낸 후 ⑧ DispatcherServlet 은 최종 결과를 클라이언트에 전송합니다.

매우 복잡한 과정으로 처리되긴 하지만 이런 DispatcherServlet 전략에서 사용자가 직접 구현해야할 부분은 컨트롤러와 뷰 밖에 없습니다. 나머지 핸들러 매핑이나 리졸버는 대략적인 흐름만 알고 있다가 나중에 필요할 때 필요한 클래스를 컨텍스트에 등록시키기만 하면 그만입니다.

DispatcherServlet가 처리하지 않면 안되는데...

여기서 하나 문제가 발생했습니다. 디스패처 서블릿이 모든 요청을 컨트롤러에 넘겨주는 방식은 괜찮은데 모든 요청을 처리하다보니 이미지, HTML 파일, 자바스크립트 파일 스타일시트 파일들을 불러오는 요청마저 전부 컨트롤러로 넘겨버리는 것입니다.

이미지나 파일들에 대한 요청을 디스패처 서블릿이 처리하지 않도록 하려면 어떻게 해야 할까요?

```
No mapping found for HTTP request with URI [/Market/resources/css/default.css] in DispatcherServlet with name
No mapping found for HTTP request with URI [/Market/resources/css/commons/login.css] in DispatcherServlet with
No mapping found for HTTP request with URI [/Market/resources/css/commons/login.css] in DispatcherServlet with
```

만약 이런 처리를 해주지 않는다면 위와 같은 에러가 로그에 기록될 것입니다. 디스패처 서블릿이 해당 요청을 처리할 컨트롤러를 찾지 못했다는 에러 메시지죠. 이 문제를 해결하기 위해 스프링은 처리할 URL 과 관련 없는 **리소스 영역 분리**를 이용합니다.

/apps - 클라이언트가 이 URL 로 접근하면 앞으로 디스패처 서블릿이 담당
/resources - 이 URL 은 디스패처 서블릿이 컨트롤할 수 없는 영역으로 분리

스프링은 **리소스 영역 분리**를 위해 <mvc:resources /> 를 이용하여 설정하게 됩니다. 만약 이클립스를 통해 저처럼 프로젝트를 Spring Project 생성하셨다면 servlet-context.xml 에서 다음과 같은 코드를 발견하실 수 있으실 겁니다.

<resources mapping="/resources/-*" location="/resources/" />

이 전략은 다음과 같습니다. 먼저 디스패처 서블릿을 통해 들어온 요청을 처리하는데 만약 디스패처 서블릿이 해당 요청에 대한 컨트롤러가 찾을 수 없다면 2 차적으로 위의 설정된 경로를 검색하여 해당 자원을 찾아내게 되는 것이죠. 저같은 경우는 기존의 resources 폴더의 Depoloyment Assembly 를 변경하여 아래와 같이 설정하는 것으로 문제를 해결했습니다.

만약 이런 비 어플리케이션 자원을 따로 분리하지 않고 어플리케이션 자원과 한데 섞어 작업하신다면 지금부터라도 이렇게 분리해서 작업하시길 권장해 드립니다. 먼저 이런 영역분리는 효율적인 리소스 관리와 확장을 돕기 때문입니다.

현대의 많은 대형 웹 서비스들의 비 어플리케이션 자원 URL 을 보시면 철저하게 static 성격의 외부 자원들을 분리시켜 사용하고 있습니다. 네이버같은 경우는 http://static.naver.net/란 URL 을 통해 이런 자원들을 분리하였으며 페이스북은 http://static.ak.fbcdn.net/과 같은 URL 로 분리시켰습니다. 이런 일례를 들어서라도 차후 확장을 위해 비어플리케이션 자원은 반드시 분리할 것을 권하고 싶네 요.

3.2 HandlerMapping

- RequestURL 과 Controller 클래스의 맵핑을 담당한다.
- BeanNameUrlHandlerMapping 클래스는 스프링 설정 파일 내에 정의된 Controller 클래스의 name 속성과 RequestURl 과 맵핑하는 HandlerMapping 인터페이스의 구현 클래스로 name 속성으로 지정된 값을 그대로 context root 이하의 패스로 처리한다.

디폴트 HandlerMapping 클래스: **BeanNameUrlHandlerMapping, DefaultAnnotationHandlerMapping**

◈ 스프링 설정 파일에서의 BeanNameUrlHandlerMapping의 정의

```
〈!-- Controller --〉
〈bean id = "indexController" name = "/index.html" class = "controller.IndexController" 〉
 〈property name = "shopService" 〉 〈ref bean = "shopService" /〉 〈/property〉
〈/bean〉
```

3.3 Controller

Controller 는 필요한 비즈니스 로직을 호출하여 Model 정보와 이동할 View 정보를 ModelAndView 를 이용하여 DispatherServlet 에 반환하게 된다.

그리고 DispatherServlet 는 Controller 로부터 반환된 이동할 View 이름을 이용하여 ViewResolver 에게 View 인스턴스를 찾게 하고 모델을 rendering 하여 처리 결과를 브라우저에 표시한다.

HandlerMapping, Controller, ViewResolver, View 는 모두 스프링 MVC 가 제공하는 인터페이스이다.

개발에서는 Controller 를 제외한 HandlerMapping, Viewresolver, View 인터페이스의 구현 클래스는 스프링 MVC 에서 제공하는 구현 클래스를 활용하면 된다.

3.4 ModelAndView

1) Controller 처리 결과 후 응답할 view 와 veiw 에 전달할 Model 값을 저장.

2) 생성자
- ModelAndView(String viewName) : 응답할 view 설정
- ModelAndView(String viewName, Map values) : 응답할 view 와 view 로 전달할 값들을 저장 한 Map 객체
- ModelAndView(String viewName, String name, Object value) : 응답할 view 이름, view 로 넘길 객체의 name-value

3) 주요 메소드
- setViewName(String view) : 응답할 view 이름을 설정
- addObject(String name, Object value) : view 에 전달할 값을 설정 - requestScope 에 설정됨
- addAllObjects(Map values) : view 에 전달할 값을 Map 에 name-value 로 저장하여 한번에 설정 - requestScope 에 설정됨

4) Redirect 방식 전송
- view 이름에 redirect: 접두어 붙인다.ex) mv.setViewName(“redirect:/welcome.html”);

```
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;
}
```

3.5 ViewResolver

- ViewResolver 는 ModelAndView 의 뷰 이름을 이용하여 실제 View 를 찾는다.
- ViewResolver - Spring 설정파일에 등록한다
- 디폴트 ViewResolver 는 InternalResourceViewResolver

1) Controller 가 넘긴 view 이름을 통해 알맞은 view 를 찾는 역할
   1. Controller 는 ModelAndView 객체에 응답할 view 이름을 넣어 return.
   2. DispatchServlet 은 ViewResolver 에게 view 이름을 주고 응답할 view 를 요청한다.
   3. ViewResolver 는 View 이름을 이용해 알 맞는 view 객체를 찾는다.

Controller 에서 아래와 같이 코딩 한다면 /WEB-INF/views/hello.jsp 를 찾는다.

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

3.6 View

- 프레젠테이션층으로의 출력 데이터를 설정한다.
- 기본 경로는 /src/main/webapp/WEB-INF/views

4. 스프링 설정

4.1 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">



    <!-- Spring Web 어플리케이션을 위한 메인 설정파일을 등록한다. -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>

    <!-- Spring Web 어플리케이션 컨테스트를 로딩한다. -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>

    <!-- Spring Web 어플리케이션의 맨 앞단 Controller(DispatcherServlet) 를 등록한다. -->
    <servlet>
        <servlet-name>appServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
```

```
        <servlet-name>appServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>


    <!-- 문자 인코딩 처리 필터 설정 -->
    <filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>


    <!--   -->
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>


    <!-- jsp 파일 utf-8 페이지 인코딩 설정 <%@page pageEncoding="UTF-8"%> -->
    <jsp-config>
```

```
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <page-encoding>UTF-8</page-encoding>
        </jsp-property-group>
    </jsp-config>

</web-app>
```

4.2 root-context.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.xsd">

    <!-- Root Context: defines shared resources visible to all other web components -->

</beans>
```

4.3 servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"

    xsi:schemaLocation="http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans
```

```
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- DispatcherServlet Context: defines this servlet's request-processing
infrastructure -->



    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static
resources in the ${webappRoot}/resources directory -->
    <resources mapping="/resources/**" location="/resources/" />


    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the
/WEB-INF/views directory -->
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>


    <!-- step1. annotation 설정 -->
    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />


    <!-- step2. component scan 설정 -->
    <context:component-scan base-package="com.spring11.mvc" />


    <!-- step3. 데이터베이스 설정 -->
    <!-- 커넥션 설정 -->
    <beans:bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
```

```
        <beans:property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
        <beans:property name="url" value="jdbc:oracle:thin:@//localhost:1521/pdborcl" />
        <beans:property name="username"  value="tester1" />
        <beans:property name="password"  value="1234" />
        <beans:property name="defaultAutoCommit" value="true"/>
        <beans:property name="poolPreparedStatements" value="true"/>
        <beans:property name="cacheState"  value="true"/>
    </beans:bean>


    <!-- SessionFactory 설정 :: MyBatis 가 사용할 Database 에 연결하도록 설정 -->
    <beans:bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <beans:property name="dataSource"         ref="dataSource" />
        <!-- <beans:property name="configLocation"     value="classpath:Configuration.xml"
/>  -->

        <!-- mybatis 디렉토리에 xml 파일만 추가해주면 알아서 xml 내의 쿼리 자동 인식 -->
        <!-- <beans:property name="mapperLocations"
value="classpath*:mybatis/**/mapper*.xml" />  -->
    </beans:bean>

    <!-- MyBatis 의 CRUD 템플릿을 사용할 수 있도록 설정 -->
    <beans:bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate"
scope="singleton">
        <beans:constructor-arg index="0" ref="sqlSessionFactory" />
    </beans:bean>


    <!-- step4. 트랜잭션 설정 :  commit, rollback -->
    <beans:bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <beans:property name="dataSource" ref="dataSource" />
    </beans:bean>
```

```
    <!-- 실질적으로 트랜잭션 advice 가 어디서 필요한지 알려줌 -->
    <aop:config>
        <!-- 패키지 mybatis. 이하의 모든 메서드에 pointcut 을 걸음. -->
        <aop:pointcut id="controllerTx"  expression="execution(public *
mybatis..service.Service*.*(..))" />

        <!-- ~Service2 라는 bean 이름을 가진 클래스에 pointcut 을 걸음.
        <aop:pointcut id="txAdvisePointcut" expression="bean(*Service2)" />
        -->

        <aop:advisor advice-ref="txAdvice" pointcut-ref="controllerTx" />
    </aop:config>


    <!-- Transaction 대상 설정 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="insert*"       rollback-for="RuntimeException" />
            <tx:method name="write*"        rollback-for="RuntimeException" />
            <tx:method name="add*"          rollback-for="RuntimeException" />
            <tx:method name="create*"       rollback-for="RuntimeException" />
            <tx:method name="regist*"       rollback-for="RuntimeException" />
            <tx:method name="set*"          rollback-for="RuntimeException" />

            <tx:method name="update*"       rollback-for="RuntimeException" />
            <tx:method name="modify*"       rollback-for="RuntimeException" />
            <tx:method name="edit*"         rollback-for="RuntimeException" />
            <tx:method name="change*"       rollback-for="RuntimeException" />

            <tx:method name="delete*"       rollback-for="RuntimeException" />
            <tx:method name="remove"        rollback-for="RuntimeException" />
            <tx:method name="terminate*"    rollback-for="RuntimeException" />
```

```
            <tx:method name="read*"         read-only="true" />
            <tx:method name="select*"       read-only="true" />
            <tx:method name="get*"          read-only="true" />
        </tx:attributes>
    </tx:advice>

</beans:beans>
```

5. Reference

http://devbox.tistory.com/entry/Spring-스프링-MVC-패턴-개요
http://devbox.tistory.com/entry/spring-스프링-MVC-인터페이스-구현-클래스
http://gangzzang.tistory.com/entry/스프링 Spring-MVC-프레임워크 Model-View-Controller-Framework-1
http://egloos.zum.com/springmvc/v/504151