

Spring Framework

- 스프링 MVC를 이용한 웹 요청 처리

■ CONTENTS

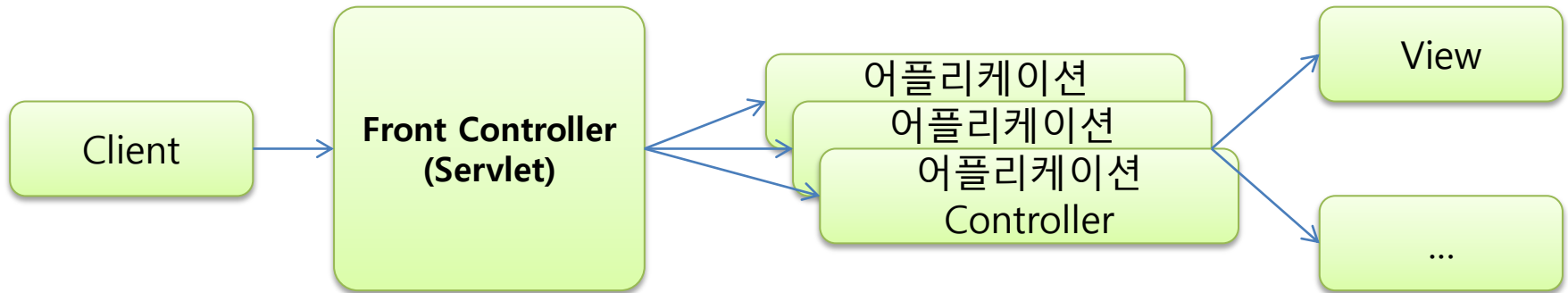
- 스프링 MVC
- 스프링 MVC의 주요 구성 요소 및 처리 흐름
- DispatcherServlet과 ApplicationContext
- 컨트롤러 구현
- 파일 업로드
- @PathVariable을 이용한 경로 변수 처리

■ 스프링 MVC

- Model2 구조를 프레임워크 차원에서 제공하는 모듈
- Spring은 DI나 AOP 같은 기능뿐만 아니라 웹 개발을 위한 MVC 프레임워크도 직접 제공하는 서블릿 기반의 MVC 프레임워크
- Spring MVC 프레임워크는 Spring을 기반으로 하고 있기 때문에 Spring이 제공하는 트랜잭션 처리나 DI 및 AOP등을 손쉽게 사용할 수 있다는 장점이 있음

■ 스프링 MVC

- 프론트 Controller 프로세스



- 클라이언트가 보낸 요청을 받아서 공통적인 작업을 먼저 수행
- 적절한 세부 Controller로 작업 위임
- 클라이언트에게 보낼 View를 선택해서 최종 결과를 생성하는 등의 작업 수행
 - 인증이나 권한 체크 처럼 모든 요청에 대하여 공통적으로 처리되어야 하는 로직이 있을 경우 전체적으로 클라이언트의 요청을 중앙 집중적으로 관리하고자 할 경우에 사용

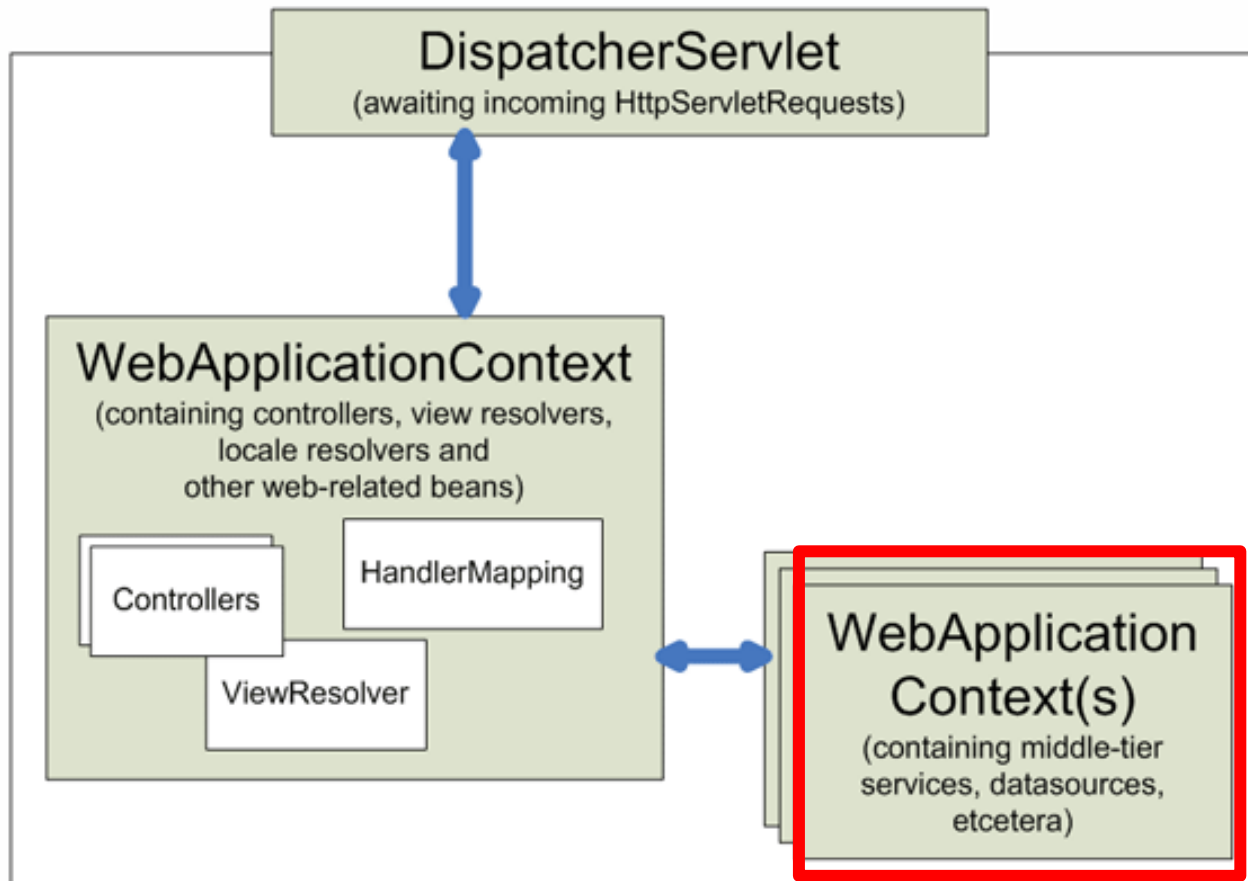
- 프론트 Controller 필요성

- 다양한 클라이언트의 요청을 제어하려면 많은 수의 controller가 필요하기 때문에 매우 복잡해지고 분산됨
- 전체 애플리케이션을 통합, 관리하기 위해 프론트 Controller 패턴을 적용
- **JavaEE Core Design Pattern에서 프리젠테이션 계층을 위한 패턴으로 대부분의 MVC 프레임워크들은 이 패턴을 적용해서 구현**
- 중앙 집중형 Controller를 프리젠테이션 계층의 앞 단에 놓고 서버로 들어오는 모든 요청을 먼저 받아서 처리하도록 구성하는 것으로 Spring MVC에 적용해서 클라이언트 요청을 관리 하고자 할 때 사용

- **DispatcherServlet API**

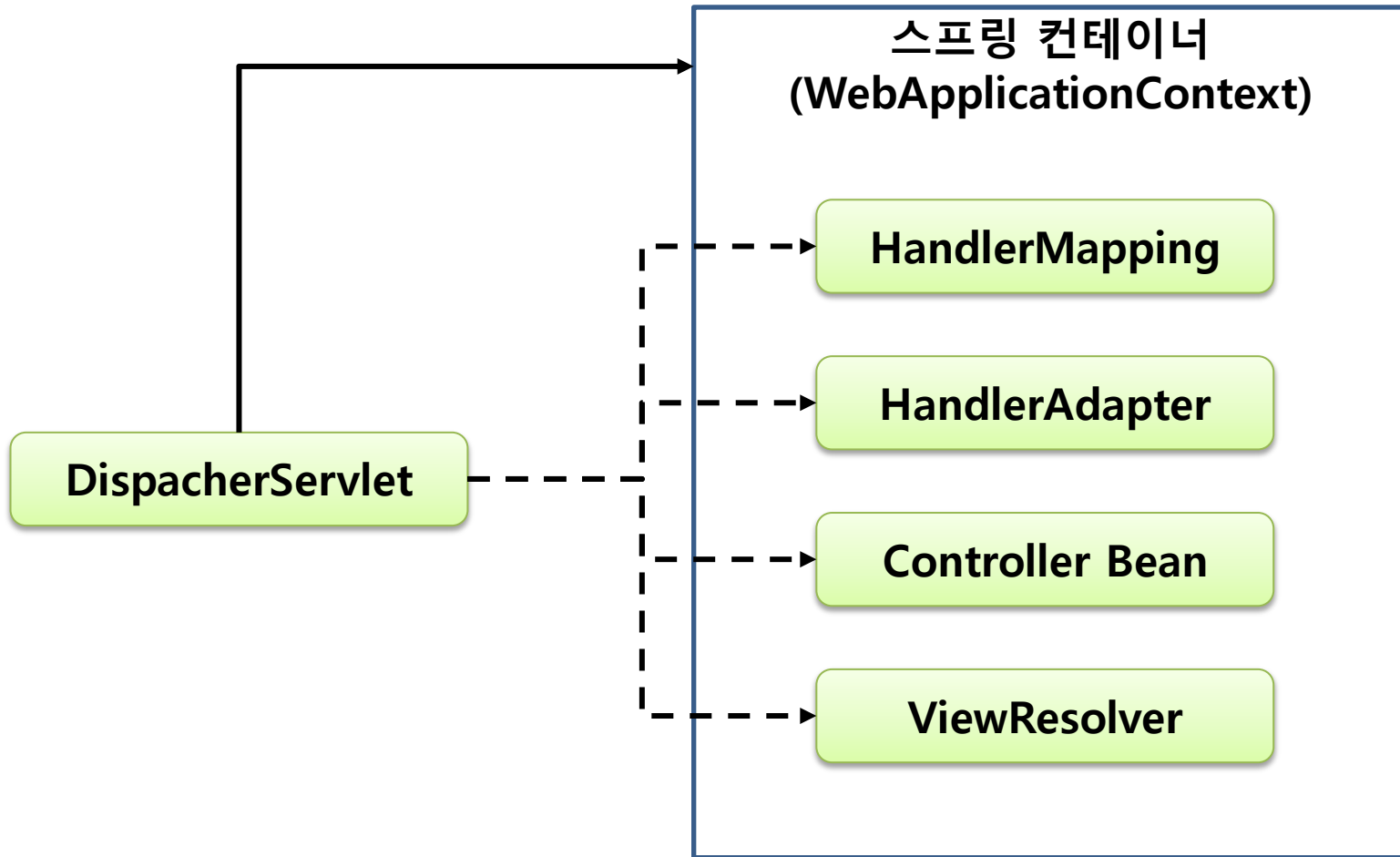
- web.xml에 설정
- 프론트 Controller 패턴 적용
- client 요청을 전달받음
- controller나 view와 같은 Spring MVC의 구성 요소를 이용하여 client에게 서비스 제공

■ 스프링 MVC의 주요 구성 요소



■ 스프링 MVC의 주요 구성 요소

DispatcherServlet은 스프링 컨테이너를 생성하고, 컨테이너로부터 필요한 빈 객체를 구한다.



■ 스프링 MVC의 주요 구성 요소

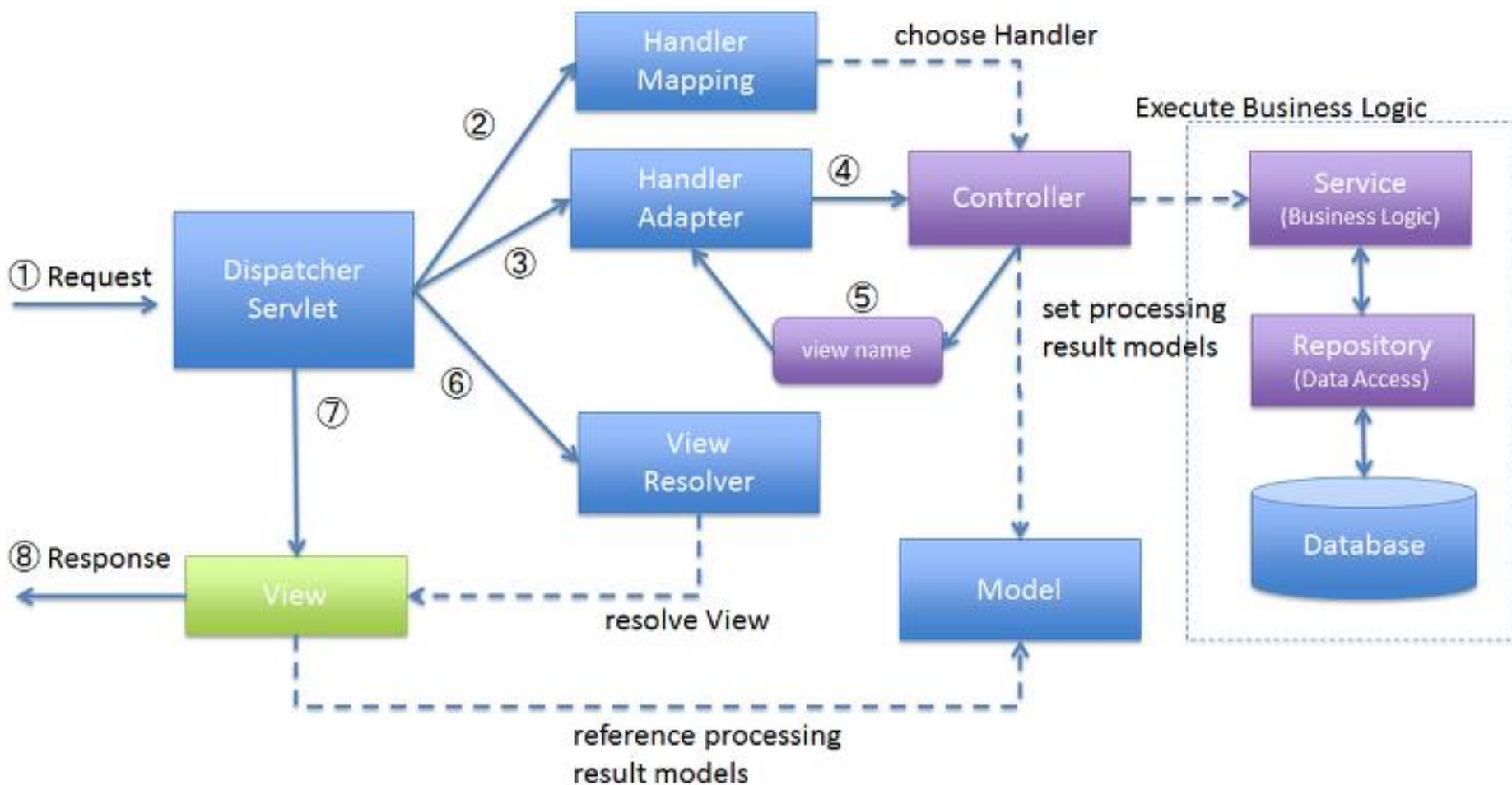
구성 요소	설 명
DispatcherServlet	클라이언트의 요청을 받아서 Controller에게 클라이언트의 요청을 전달하고, 리턴한 결과값을 View에게 전달하여 알맞은 응답을 생성
HandlerMapping	URL과 요청 정보를 기준으로 어떤 핸들러 객체를 사용할지 결정하는 로직의 객체 DispatcherServlet은 하나 이상의 핸들러 매핑을 가질 수 있음
Controller	클라이언트의 요청을 처리한 뒤 그 결과를 DispatcherServlet에게 알려 줌
ModelAndView	Controller가 처리한 결과정보(Model) 및 뷰 선택에 필요한 정보를 보유한 객체
ViewResolver	Controller가 리턴한 뷰 이름을 기반으로 Controller 처리 결과를 생성할 뷰를 결정
View	Controller의 처리 결과 화면을 생성함




■ 스프링 MVC의 주요 구성 요소

- **Spring MVC의 클라이언트 요청 처리 과정**

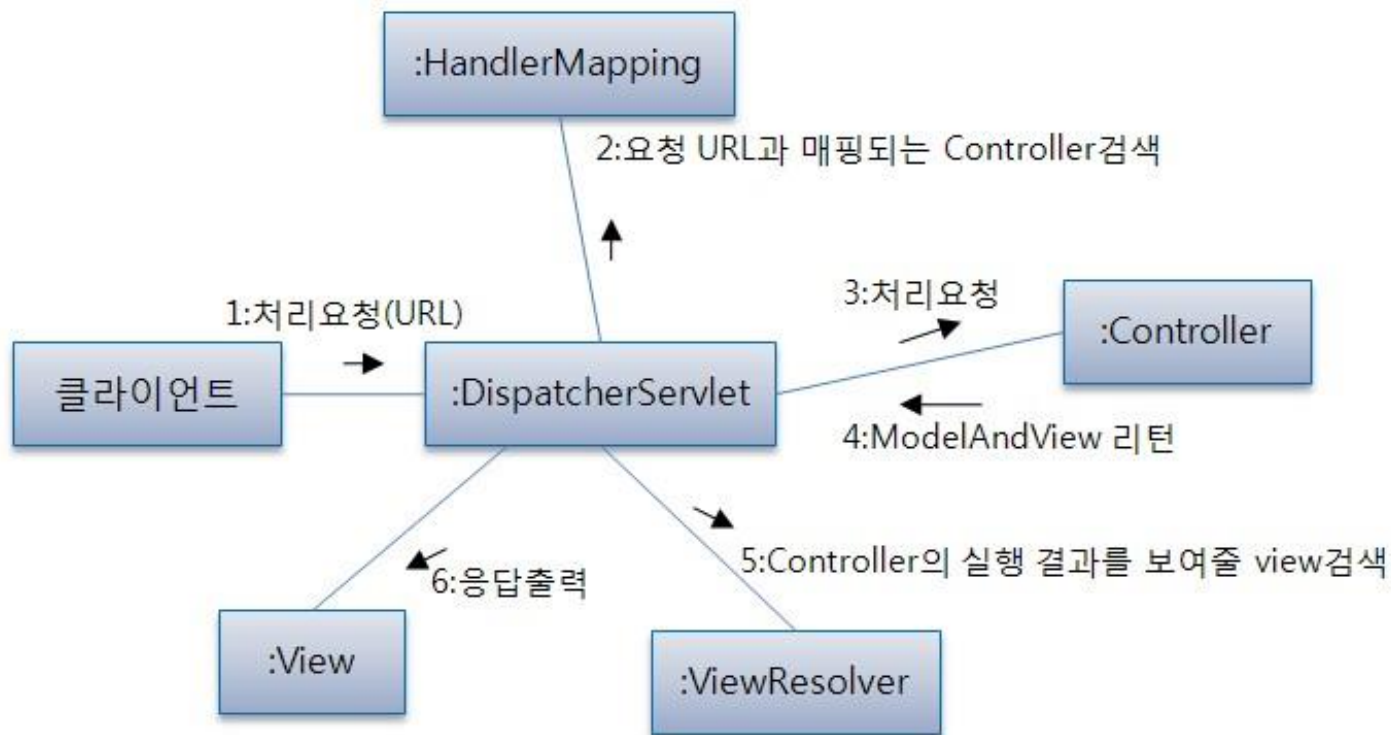
1. client의 요청이 DispatcherServlet에게 전달
2. DispatcherServlet은 HandlerMapping을 사용하여 client의 요청을 처리할 Controller 객체를 획득
3. DispatcherServlet은 controller 객체를 이용하여 client의 요청 처리
4. controller는 client의 요청 처리 결과 정보를 담은 ModelAndView 객체를 반환
5. DispatcherServlet은 ViewResolver로부터 응답 결과를 생성할 View 객체를 구함
6. View는 client에게 전송할 응답을 생성

■ 스프링 MVC의 주요 구성 요소

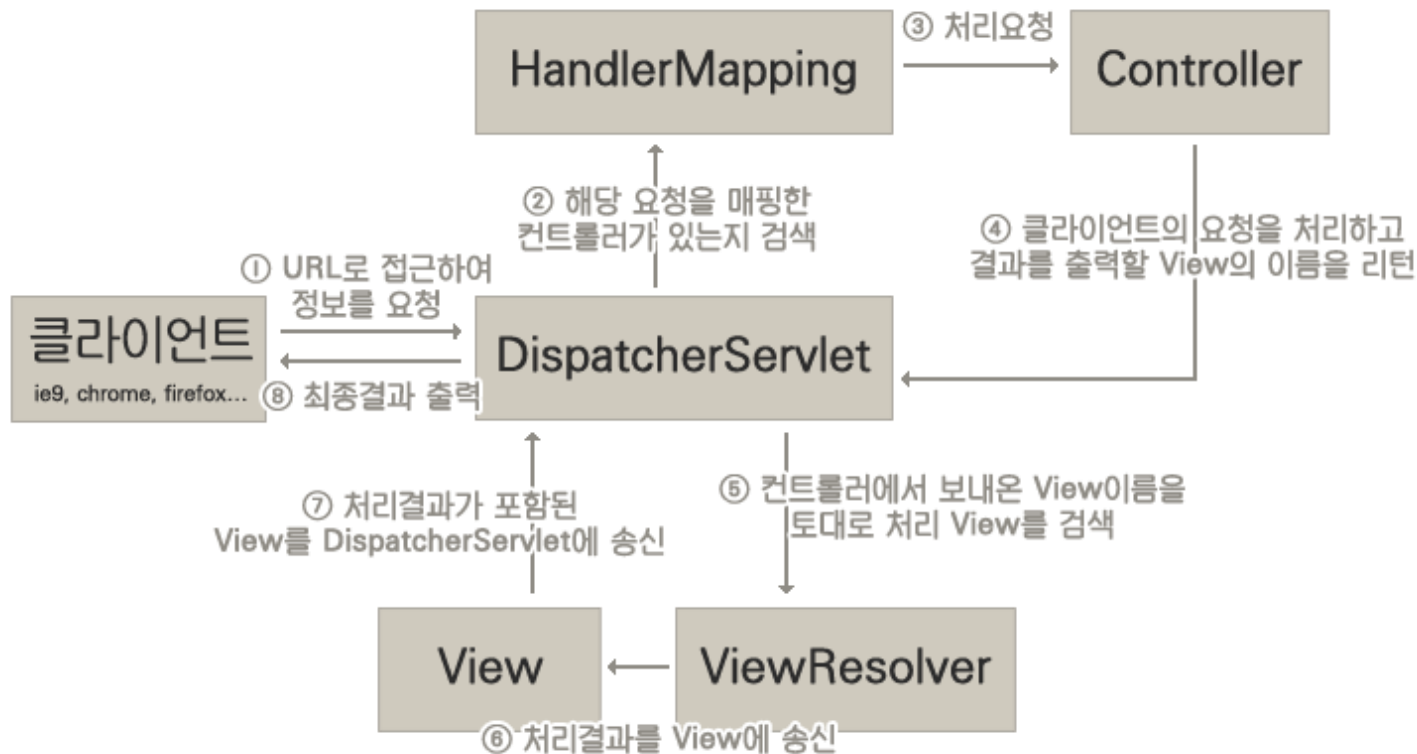


-  ... implemented by developers
-  ... provided by Spring Source
-  ... provided by Spring Source sometimes implemented by developers

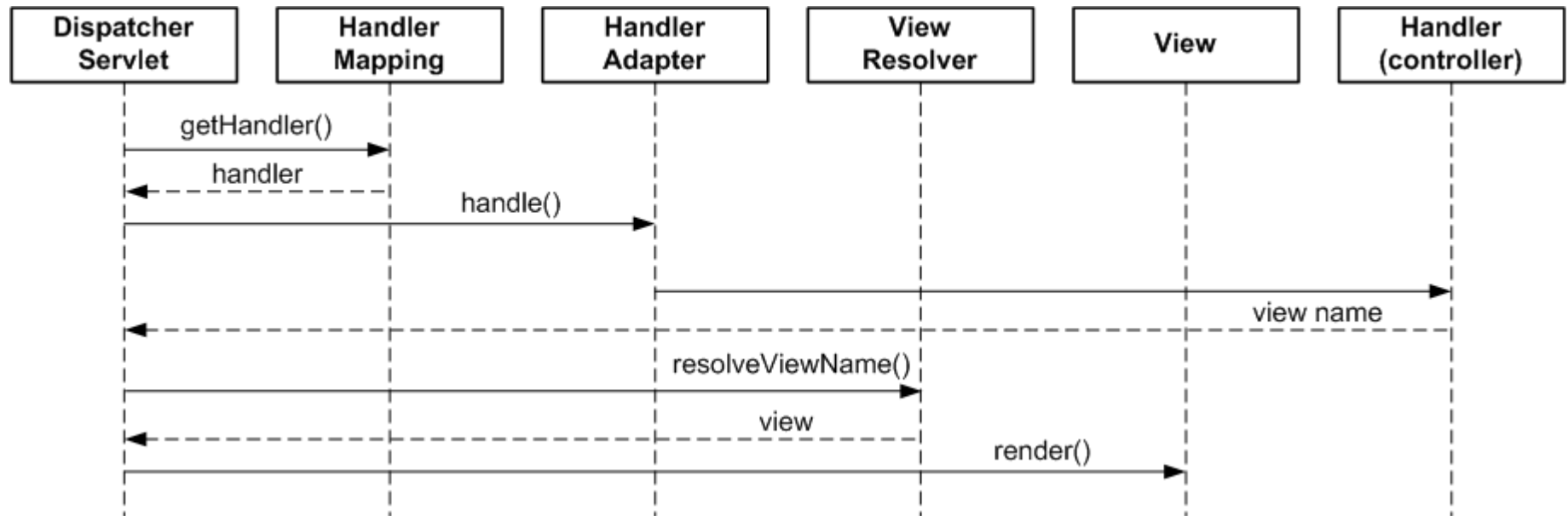
■ 스프링 MVC의 주요 구성 요소



■ 스프링 MVC의 주요 구성 요소



■ 스프링 MVC의 주요 구성 요소



- **Spring MVC 웹 애플리케이션 개발 단계**

1단계 – client의 요청을 받을 DispatcherServlet 을 web.xml 파일에 설정

2단계 – client의 요청을 처리할 Controller 작성

3단계 – Spring 빈으로 controller 등록 & ViewResolver 설정

4단계 – JSP를 이용한 뷰 영역의 코드 작성

5단계 – 실행

- **단계 1 : DispatcherServlet 설정 및 스프링 컨텍스트 설정**
 - ✓ client의 요청을 받을 DispatcherServlet 을 설정
 - ✓ 공통으로 처리할 어플리케이션 컨텍스트 설정

WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app ...>
```

```
<servlet>
```

```
    <servlet-name>dispatcher</servlet-name>
```

```
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
```

```
</servlet>
```

```
<!-- client의 요청을 전달받는 서블릿 Controller, 뷰와 같은 SpringMVC의 구성 요소를 이용하여 client에  
게 서비스 -->
```

```
<servlet-mapping>
```

```
    <servlet-name>dispatcher</servlet-name>
```

```
    <url-pattern>*.do</url-pattern>
```

```
</servlet-mapping>
```

```
<!-- 1. *.do로 요청되는 모든 client 의 요청을 DispatcherServlet이 처리하도록 요청  
2. dispatcher-servlet.xml이라는 이름의 Spring 설정 파일을 사용하도록 설정 -->
```

```
</web-app>
```

- 단계 2 : 컨트롤러 구현 및 설정 추가 하기

- ✓ 컨트롤러 구현

- @Controller – Spring MVC의 controller 를 구현한 클래스로 지정

- @RequestMapping – /hello.do라는 url값으로의 요청을 처리할 메소드 지정

- [<http://host:port/contextPath/hello.do>]

- ✓ 설정파일에 등록

- dispatcher-servlet.xml

mvctest.controller.HelloController.java

```
package mvctest.controller;
```

```
import java.util.Calendar;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.servlet.ModelAndView;
```

@Controller

```
public class HelloController {
```

```
    @RequestMapping("/hello.do")
```

```
    public ModelAndView hello() {
```

//ModelAndView는 컨트롤러의 처리 결과를 보여줄 뷰와 뷰에서 처리할 모델을 설정.

```
        ModelAndView mav = new ModelAndView();
```

```
        mav.setViewName("hello");
```

```
        mav.addObject("greeting", getGreeting());
```

```
        return mav;
```

```
    }
```

mvctest.controller.HelloController.java

```
private String getGreeting() {  
    int hour = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);  
    if (hour >= 6 && hour <= 10) {  
        return "좋은 아침입니다.";  
    } else if (hour >= 12 && hour <= 15) {  
        return "점심 식사는 하셨나요?";  
    } else if (hour >= 18 && hour <= 22) {  
        return "좋은 밤 되세요";  
    }  
    return "안녕하세요";  
}  
  
}
```

dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
       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-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <bean id="helloController" class="mvctest.controller.HelloController" />

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

- 단계 3 : 설정파일에 viewResolver 설정 추가하기

설정 추가

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

```
<mvc:view-resolvers>  
    <mvc:jsp prefix="/WEB-INF/view/" />  
</mvc:view-resolvers>
```

- **단계 4 : 뷰 코드 구현**

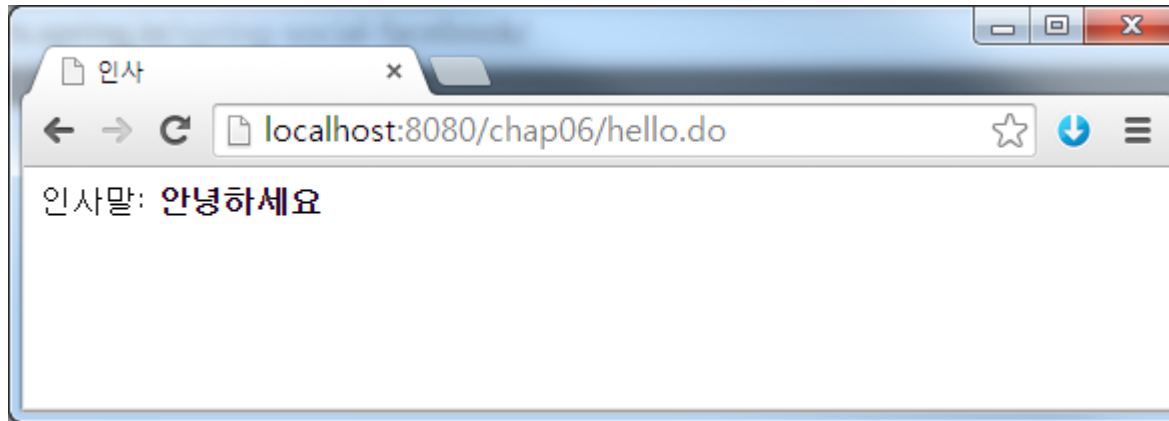
- ✓ HelloController 처리 결과를 보여줄 뷰 JSP 페이지 구현

WEB-INF/view/hello.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>인사</title>
</head>
<body>
인사말: <strong>${greeting}</strong>
</body>
</html>
```

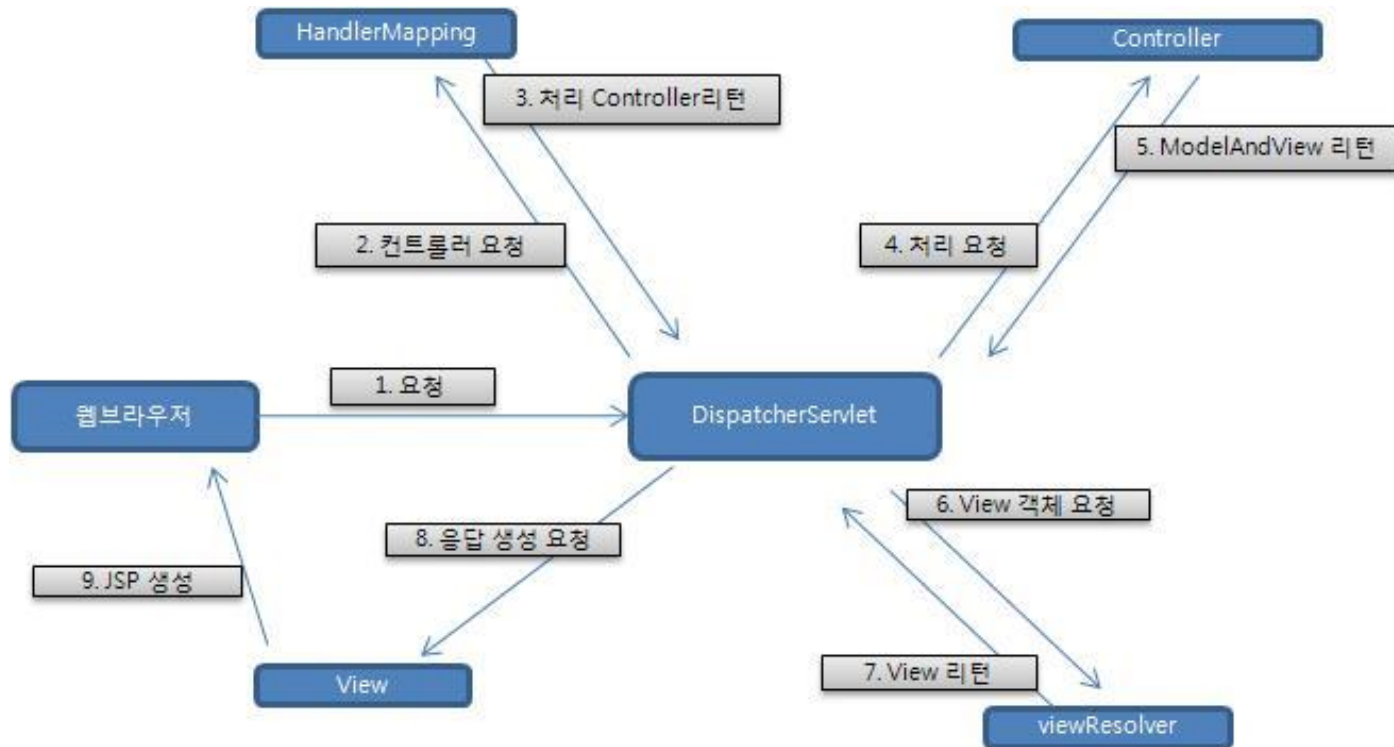

■ 스프링 MVC

- 단계 5 : 실행



■ 스프링 MVC

• 단계 6 : 실행흐름



■ 스프링 MVC

- DispatcherServlet 설정과 ApplicationContext의 관계
- 한 개 이상의 설정 파일을 사용해야 할 경우.

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/main.xml
      /WEB-INF/bbs.xml
    </param-value>
  </init-param>
</servlet>
```

- 초기화 파라미터는 설정 목록을 값으로 갖는다.
“, ”, “ ”, 탭, 줄바꿈, “;” 이용해서 구분

- 캐릭터 인코딩 처리를 위한 필터 설정
 - Spring은 servlet 필터를 이용하여 요청 파라미터의 캐릭터 인코딩 설정 가능

```
<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>
</filter>
```

■ 컨트롤러 구현 - @Controller

- @Controller 어노테이션 이용 권장
- controller 구현을 위한 필수 어노테이션
 - @Controller & @RequestMapping

- 개발 방법

1단계 - Controller 클래스에 @Controller 어노테이션 선언

2단계 - client의 요청을 처리할 메소드에

@RequestMapping 어노테이션 적용

3단계- [servletname]-servlet.xml에 controller 클래스를 빈으로 등록

■ 컨트롤러 구현 - @Controller

- Controller를 위한 핵심 어노테이션

@Controller

Controller 클래스 정의

@RequestMapping

HTTP 요청 URL을 처리할 Controller 메소드 정의

@RequestParam

HTTP 요청에 포함된 파라미터 참조 시 사용

@ModelAttribute

HTTP 요청에 포함된 파라미터를 모델 객체로 바인딩

@ModelAttribute의 'name'으로 정의한 모델 객체를 다음 뷰에게
사용 가능

■ 컨트롤러 구현 - @Controller

- 클라이언트의 요청을 처리할 메소드에 선언하는 어노테이션
- 문법
 - 문법 1 - 요청 URL만 선언할 경우
 - **@RequestMapping("/요청한 URL.do")**
 - 문법 2 - 요청 방식을 지정할 경우
 - **@RequestMapping(value="/요청한 URL.do", method=RequestMethod.POST)**
- Controller의 처리 결과를 보여줄 view 지정 방법
 - 메소드의 반환 값에 따른 view page지정 방법
 - 방법 1 - ModelAndView인 경우**
 - : setName() 메소드 파라미터로 설정**
 - 방법 2 - String인 경우 : 메소드의 리턴 값**

■ 컨트롤러 구현 - @Controller

- 동일한 URL로 Get/Post 방식 모두 처리할 경우의 설정

```
@Controller
@RequestMapping("/hello.do")
public class HelloController {
    @RequestMapping(method=RequestMethod.GET)
    public ModelAndView getHelloMessage()
    {
        //...
    }
    @RequestMapping(method=RequestMethod.POST)
    public ModelAndView getHelloMessage()
    {
        //...
    }
}
```


■ 컨트롤러 구현 - @Controller

컨트롤러 메서드의 파라미터 타입	설명
HttpServletRequest, HttpServletResponse, HttpSession	서블릿 API
java.util.Locale	현재 요청에 대한 Locale
InputStream, Reader	요청 콘텐츠에 직접 접근할 때 사용
OutputStream, Writer	응답 콘텐츠를 생성할 때 사용
@PathVariable 어노테이션	URI 템플릿 변수에 접근할 때 사용
@RequestParam 어노테이션	HTTP 요청 파라미터를 매핑
@RequestHeader 어노테이션	HTTP 요청 헤더를 매핑
@CookieValue 어노테이션	HTTP 쿠키 매핑
@RequestBody 어노테이션	HTTP 요청의 몸체 내용에 접근할 때 사용, <code>HttpMessageConverter</code> 를 이용해서 HTTP 요청 데이터를 해당 타입으로 변환
Map, Model, ModelMap	뷰에 전달할 모델 데이터를 설정할 때 사용
커맨드 객체	HTTP 요청 파라미터를 저장한 객체. 기본적으로 클래스 이름을 모델명으로 사용. <code>@ModelAttribute</code> 어노테이션을 사용하여 모델명을 설정
Errors, BindingResult	HTTP 요청 파라미터를 커맨드 객체에 저장한 결과. 커맨드 객체를 위한 파라미터 바로 다음에 위치

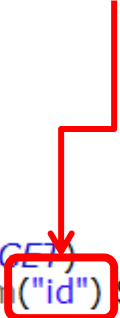
■ 컨트롤러 구현 - @Controller

- @RequestParam 어노테이션 – 파라미터 매핑

HTTP 요청 파라미터를 메소드의 파라미터로 전달받을 때 사용

http://localhost:8080/spring_step13MVC_1basic/getMethod.do?id=tester

```
30 @RequestMapping(value="/getMethod.do",  
31                  method=RequestMethod.GET)  
32 protected String getMethod(@RequestParam("id") String id) {  
33     //...중략  
34     return "getView";  
35 }  
36
```



■ 컨트롤러 구현 - @Controller

mvctest.controller.SearchController.java

```
package mvctest.controller;
```

```
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.servlet.ModelAndView;
```

@Controller

```
public class SearchController {
```

```
    @RequestMapping("/search/internal.do")
```

```
    public ModelAndView searchInternal(  
        @RequestParam("query") String query,
```

```
        @RequestParam(value = "p", defaultValue = "1") int pageNumber) {
```

```
        System.out.println("query=" + query + ",pageNumber=" + pageNumber);
```

```
        return new ModelAndView("search/internal");  
    }
```

■ 컨트롤러 구현 - @Controller

mvctest.controller.SearchController.java

```
@RequestMapping("/search/external.do")
public ModelAndView searchExternal(
    @RequestParam(value="query", required=false) String query,
    @RequestParam(value = "p", defaultValue = "1") int pageNumber) {
    System.out.println("query=" + query + ",pageNumber=" + pageNumber);
    return new ModelAndView("search/external");
}
```

■ 컨트롤러 구현 - @Controller

dispatcher-servlet.xml

...

```
<bean id= "searchController" class= "mvctest.controller.SearchController" />
```

...

■ 컨트롤러 구현 - @Controller

View/search/internal.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>내부 검색</title>
</head>
<body>
내부 검색
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

View/search/external.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>외부 검색</title>
</head>
<body>
외부 검색
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

- **HTML 폼과 자바 빈 객체**

- HTML 폼에 입력한 데이터를 자바 빈 객체로 전달하는 방법
 - @RequestMapping 이 적용된 메소드의 파라미터로 자바빈 타입 추가 설정

■ 컨트롤러 구현 - @Controller

- 데이터 입력 -> User빈 객체 생성 -> 각 멤버 변수에 입력된 데이터 대입 -> postMethod 메소드의 파라미터로 대입

```
25 post 방식으로 controller 호출해 보기
26 <form method="post" action="postMethod.do">
27   id : <input type="text" name="id"/><br>
28   이름 : <input type="text" name="name"/><br>
29   age : <input type="text" name="age"/><br>
30   <input type="submit" value="post요청" />
31 </form>
```

```
3 public class User {
4   private String id;
5   private String name;
6   private int age;
7   public User() {}
8   public User(String id, String name, int age) {
9     this.id = id;
10    this.name = name;
11    this.age = age;
12  }
13 }
```

```
21 @RequestMapping(value="/postMethod.do",
22                  method=RequestMethod.POST)
23 protected String postMethod(User user) {
24   return "postView";
25 }
26
```

■ 컨트롤러 구현 - @Controller

- view에서는 Controller의 @RequestMapping 메소드에서 전달받은 Java Bean 객체 access 가능
- view 코드
 - 자바빈 객체의 클래스 이름(첫 글자 소문자)를 이용하여 access
 - view 코드에서 사용할 자바 빈의 이름을 변경하고자 할 경우 @ModelAttribute 어노테이션 사용

```
21 @RequestMapping(value="/postMethod.do",
22                  method=RequestMethod.POST)
23 protected String postMethod(User user) {
24     return "postView";
25 }
26
```

client가 입력한 id값 - \${user.id}

```
3 public class User {
4     private String id;
5     private String name;
6     private int age;
7     public User() {}
8     public User(String id, String name, int age) {
9         this.id = id;
10        this.name = name;
11        this.age = age;
12    }
13 }
```

- **Spring MVC 웹 애플리케이션 개발 단계**

1단계 – client의 요청을 받을 DispatcherServlet 을 web.xml 파일에 설정

2단계 – client의 요청을 처리할 Controller 작성

3단계 – Spring 빈으로 controller 등록 & ViewResolver 설정

4단계 – JSP를 이용한 뷰 영역의 코드 작성

5단계 – 실행

■ 컨트롤러 구현 - @Controller

mvctest.controller.NewArticleController.java

```
package mvctest.controller;
```

```
import mvctest.service.ArticleService;
```

```
import mvctest.service.NewArticleCommand;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.ModelAttribute;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestMethod;
```

```
@Controller
```

```
@RequestMapping("/article/newArticle.do")
```

```
public class NewArticleController {
```

```
    @Autowired
```

```
    private ArticleService articleService;
```

■ 컨트롤러 구현 - @Controller

mvctest.controller.NewArticleController.java

```
@RequestMapping(method = RequestMethod.GET)
```

```
public String form() {  
    return "article/newArticleForm";  
}
```

```
@RequestMapping(method = RequestMethod.POST)
```

```
public String submit(@ModelAttribute("command") NewArticleCommand command) {  
    articleService.writeArticle(command);  
    return "article/newArticleSubmitted";  
}
```

```
public void setArticleService(ArticleService articleService) {  
    this.articleService = articleService;  
}
```

```
}
```

■ 컨트롤러 구현 - @Controller

mvctest.service.ArticleService.java

```
package mvctest.service;
```

```
public class ArticleService {
```

```
    public void writeArticle(NewArticleCommand command) {  
        System.out.println("신규 게시글 등록: " + command);  
    }
```

```
}
```

■ 컨트롤러 구현 - @Controller

mvctest.service.NewArticleCommand.java

```
package mvctest.service;

public class NewArticleCommand {

    private String title;
    private String content;
    private int parentId;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }
}
```

■ 컨트롤러 구현 - @Controller

mvctest.service.NewArticleCommand.java

```
public void setContent(String content) {
    this.content = content;
}

public int getParentId() {
    return parentId;
}

public void setParentId(int parentId) {
    this.parentId = parentId;
}

@Override
public String toString() {
    return "NewArticleCommand [content=" + content + ", parentId="
+ parentId + ", title=" + title + "]\n";
}
}
```


■ 컨트롤러 구현 - @Controller

dispatcher-servlet.xml

...

```
<bean id="newArticleController" class="mvctest.controller.NewArticleController"  
p:articleService-ref="articleService" />
```

```
<bean id="articleService" class="mvctest.service.ArticleService" />
```

...

■ 컨트롤러 구현 - @Controller

View/article/newArticleForm.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>게시글 쓰기</title>
</head>
<body>
게시글 쓰기 입력 폼:
<form method="post">
<input type="hidden" name="parentId" value="0" />
제목: <input type="text" name="title" /><br/>
내용: <textarea name="content"></textarea><br/>
<input type="submit" />
</form>
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

View/article/newArticleSubmitted.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>게시글 쓰기</title>
</head>
<body>
게시글 등록됨:
<br/>
제목: ${command.title}
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

- 스프링 MVC는 LIST 타입의 프로퍼티도 바인딩 처리 해준다.

■ 컨트롤러 구현 - @Controller

mvctest.controller.OrderController.java

```
package mvctest.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
```

@Controller

@RequestMapping("/order/order.do")

```
public class OrderController {
    @RequestMapping(method = RequestMethod.GET)
    public String form() {
        return "order/orderForm";
    }
    @RequestMapping(method = RequestMethod.POST)
    public String submit(OrderCommand orderCommand) {
        return "order/orderCompletion";
    }
}
```

■ 컨트롤러 구현 - @Controller

mvctest.controller.OrderCommand.java

```
package mvctest.controller;

import java.util.List;

import mvctest.model.Address;
import mvctest.model.OrderItem;

public class OrderCommand {

    private List<OrderItem> orderItems;
    private Address address;

    public List<OrderItem> getOrderItems() {
        return orderItems;
    }
}
```

■ 컨트롤러 구현 - @Controller

mvctest.controller.OrderCommand.java

```
public void setOrderItems(List<OrderItem> orderItems) {  
    this.orderItems = orderItems;  
}  
  
public Address getAddress() {  
    return address;  
}  
  
public void setAddress(Address address) {  
    this.address = address;  
}  
  
}
```

■ 컨트롤러 구현 - @Controller

mvctest.controller.OrderItem.java

```
public class OrderItem {

    private String itemId;
    private String number;
    private String remark;
    public String getItemId() { return itemId; }
    public void setItemId(String itemId) { this.itemId = itemId; }
    public String getNumber() { return number; }
    public void setNumber(String number) { this.number = number; }
    public String getRemark() { return remark; }
    public void setRemark(String remark) { this.remark = remark; }

    @Override
    public String toString() {
        return "OrderItem [itemId=" + itemId + ", number=" + number + ",
remark=" + remark + "]\n";
    }

}
```


■ 컨트롤러 구현 - @Controller

mvctest.controller.Address.java

```
public class Address {

    private String zipcode;
    private String address1;
    private String address2;

    public String getZipcode() { return zipcode; }
    public void setZipcode(String zipcode) { this.zipcode = zipcode; }
    public String getAddress1() { return address1; }
    public void setAddress1(String address1) { this.address1 = address1; }
    public String getAddress2() { return address2; }
    public void setAddress2(String address2) { this.address2 = address2; }

    @Override
    public String toString() {
        return "Address [zipcode=" + zipcode + ", address1=" + address1 + ",
address2=" + address2 + "]\n";
    }

}
```

■ 컨트롤러 구현 - @Controller

dispatcher-servlet.xml

...

```
<bean class="mvctest.controller.OrderController" />
```

...

■ 컨트롤러 구현 - @Controller

View/order/orderCompletion.jsp

```
<%@ page contentType="text/html; charset=utf-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>주문</title>
</head>
<body>
<form method="post">
상품1: ID - <input type="text" name="orderItems[0].itemId" />
개수 - <input type="text" name="orderItems[0].number" />
주의 - <input type="text" name="orderItems[0].remark" />
<br/>
상품2: ID - <input type="text" name="orderItems[1].itemId" />
개수 - <input type="text" name="orderItems[1].number" />
주의 - <input type="text" name="orderItems[1].remark" />
<br/>
```

■ 컨트롤러 구현 - @Controller

View/order/orderForm.jsp

```
상품3: ID - <input type="text" name="orderItems[2].itemId" />
개수 - <input type="text" name="orderItems[2].number" />
주의 - <input type="text" name="orderItems[2].remark" />
<br/>
배송지:
우편번호 - <input type="text" name="address.zipcode" />
주소1 - <input type="text" name="address.address1" />
주소2 - <input type="text" name="address.address2" />
<br/>
<input type="submit" />
</form>
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

View/article/newArticleSubmitted.jsp

```
<%@ page contentType="text/html; charset=utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>주문 완료</title>
</head>
<body>
주문 완료
<br/>
주문 아이템
<ul>
<c:forEach var="item" items="${orderCommand.orderItems}">
<li>${item.itemId} / ${item.number} / ${item.remark}</li>
</c:forEach>
</ul>
배송지: ${orderCommand.address}
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

- **@CookieValue** 어노테이션 – 파라미터 매핑

쿠키를 파라미터로 받을 수 있다.

■ 컨트롤러 구현 - @Controller

mvctest.controller.CookieController.java

```
package mvctest.controller;
```

```
import javax.servlet.http.Cookie;
```

```
import javax.servlet.http.HttpServletResponse;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.CookieValue;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@Controller
```

```
public class CookieController {
```

```
    @RequestMapping("/cookie/make.do")
```

```
    public String make(HttpServletResponse response) {
```

```
        response.addCookie(new Cookie("auth", "1"));
```

```
        return "cookie/made";
```

```
    }
```

■ 컨트롤러 구현 - @Controller

mvctest.controller.CookieController.java

```
@RequestMapping("/cookie/view.do")
public String view(
    @CookieValue(value = "auth", defaultValue = "0") String auth
){
    System.out.println("auth 쿠키: " + auth);
    return "cookie/view";
}
}
```


■ 컨트롤러 구현 - @Controller

dispatcher-servlet.xml

...

```
<bean id="cookieController" class="mvctest.controller.CookieController" />
```

...

■ 컨트롤러 구현 - @Controller

View/cookie/made.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>쿠키</title>
</head>
<body>
쿠키 생성함
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

View/cookie/view.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>쿠키</title>
</head>
<body>
쿠키 확인
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

- **@RequestHeader 어노테이션 – 파라미터 매핑**

HTTP 요청 헤더값을 메서드 파라미터로 전달 받을 수 있다.

■ 컨트롤러 구현 - @Controller

mvctest.controller.HeaderController.java

```
package mvctest.controller;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.RequestHeader;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

@Controller

```
public class HeaderController {
```

```
    @RequestMapping("/header/check.do")
```

```
    public String check(@RequestHeader("Accept-Language") String languageHeader) {
```

```
        System.out.println(languageHeader);
```

```
        return "header/pass";
```

```
    }
```

```
}
```

■ 컨트롤러 구현 - @Controller

dispatcher-servlet.xml

...

```
<bean class="mvctest.controller.HeaderController" />
```

...

■ 컨트롤러 구현 - @Controller

View/header/pass.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>헤더 검사 통과</title>
</head>
<body>
헤더 검사 통과됨
</body>
</html>
```

■ 컨트롤러 구현 - @Controller

- 컨트롤러 메서드의 리턴타입

리턴 타입	설명
ModelAndView	뷰 정보 및 모델 정보를 담고 있는 ModelAndView 객체
Model	뷰에 전달할 객체 정보를 담고 있는 Model 리턴. 이때 뷰 이름은 요청 URL로부터 결정 (RequestToViewNameTranslator를 통해 뷰 결정)
Map	뷰에 전달할 객체 정보를 담고 있는 Map 리턴. 이때 뷰 이름은 요청 URL로부터 결정 (RequestToViewNameTranslator를 통해 뷰 결정)
String	뷰 이름을 리턴
View 객체	View 객체를 직접 리턴. 해당 View 객체를 이용해서 뷰를 생성
void	메소드가 ServletResponse나 HttpServletResponse 타입의 파라미터를 갖는 경우 메소드가 직접 응답을 처리한다고 가정. 그렇지 않을 경우 요청 URL로부터 결정된 뷰를 보여줌 (RequestToViewNameTranslator를 통해 뷰 결정)
@ResponseBody	메소드에서 @ResponseBody 어노테이션이 적용된 경우, 리턴 객체를 HTTP 응답으로 전송. HttpMessageConverter를 이용해서 객체를 HTTP 응답 스트림으로 변환

■ 뷰 지정

- **View 이름의 명시적 지정**

ModelAndView 와 String 리턴 타입

- **View 이름 자동 지정**

URL로 부터 **View** 이름을 결정 한다.

- 리턴 타입이 Model이나 Map인 경우
- 리턴 타입이 void 이면서 ServletResponse 나 HttpServletResponse 타입의 파라미터가 없는 경우

- **리다이렉트 뷰**

View 이름에 "redirect:" 접두어를 붙이면, 지정한 페이지로 리다이렉트 됨

- redirect:/bbs/list : 현재 서블릿 컨텍스트에 대한 상대적 경로
- redirect:http://host/bbs/list : 지정한 절대 URL 경로

■ 모델 생성하기

- **View에 전달되는 모델 데이터**

@RequestMapping 메서드가 **ModelAndView**, **Model**, **Map**을 리턴 하는 경우 이에 담긴 모델 데이터가 뷰에 전달 된다.

- 커맨드 객체
- @ModelAttribute 어노테이션이 적용된 메서드가 리턴 한 객체
- 메서드의 Map, Model, ModelMap타입의 파라미터를 통해 설정된 모델

■ 컨트롤러 구현 - @Controller

mvctest.controller.GameSearchController.java

```
package mvctest.controller;
```

```
@Controller
```

```
public class GameSearchController {
```

```
    @Autowired
```

```
    private SearchService searchService;
```

```
    @ModelAttribute("searchTypeList")
```

```
    public List<SearchType> referenceSearchTypeList() {
```

```
        List<SearchType> options = new ArrayList<SearchType>();
```

```
        options.add(new SearchType(1, "전체"));
```

```
        options.add(new SearchType(2, "아이템"));
```

```
        options.add(new SearchType(3, "캐릭터"));
```

```
        return options;
```

```
    }
```

```
    @ModelAttribute("popularQueryList")
```

```
    public String[] getPopularQueryList() {
```

```
        return new String[] { "메시", "호날두", "손흥민" };
```

```
    }
```

■ 컨트롤러 구현 - @Controller

mvctest.controller.GameSearchController.java

```
@RequestMapping("/search/main.do")
public String main() {
    return "search/main";
}

public void setSearchService(SearchService searchService) {
    this.searchService = searchService;
}

}
```

■ 컨트롤러 구현 - @Controller

dispatcher-servlet.xml

...

```
<bean class="mvctest.controller.GameSearchController"  
p:searchService-ref="searchService" />
```

...

■ 컨트롤러 구현 - @Controller

View/search/main.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>게임 검색 메인</title>
</head>
<body>
인기 키워드: <c:forEach var="popularQuery" items="${popularQueryList}">${popularQuery}
</c:forEach>
<form action="game.do">
<select name="type">
<c:forEach var="searchType" items="${searchTypeList}">
<option value="${searchType.code}">${searchType.text}</option>
</c:forEach>
</select>
<input type="text" name="query" value=""/>
<input type="submit" value="검색" />
</form></body></html>
```

■ 컨트롤러 구현 - @Controller

- 컨트롤러 메서드의 리턴타입

리턴 타입	설명
ModelAndView	뷰 정보 및 모델 정보를 담고 있는 ModelAndView 객체
Model	뷰에 전달할 객체 정보를 담고 있는 Model 리턴. 이때 뷰 이름은 요청 URL로부터 결정 (RequestToViewNameTranslator를 통해 뷰 결정)
Map	뷰에 전달할 객체 정보를 담고 있는 Map 리턴. 이때 뷰 이름은 요청 URL로부터 결정 (RequestToViewNameTranslator를 통해 뷰 결정)
String	뷰 이름을 리턴
View 객체	View 객체를 직접 리턴. 해당 View 객체를 이용해서 뷰를 생성
void	메소드가 ServletResponse나 HttpServletResponse 타입의 파라미터를 갖는 경우 메소드가 직접 응답을 처리한다고 가정. 그렇지 않을 경우 요청 URL로부터 결정된 뷰를 보여줌 (RequestToViewNameTranslator를 통해 뷰 결정)
@ResponseBody	메소드에서 @ResponseBody 어노테이션이 적용된 경우, 리턴 객체를 HTTP 응답으로 전송. HttpMessageConverter를 이용해서 객체를 HTTP 응답 스트림으로 변환

■ 파일 업로드 처리

• MultipartResolver 설정

- Multipart 지원 기능을 사용하려면 MultipartResolver를 스프링 설정 파일에 등록
- Multipart 형식으로 데이터가 전송된 경우 해당 데이터를 스프링 MVC에서 사용 가능 하도록 변환
- 스프링이 제공하는 MultipartResolver는 CommonsMultipartResolver이다
- Commons FileUpload API 를 이용해서 Multipart를 처리해 준다.
- CommonsMultipartResolver를 MultipartResolver로 사용하기 위해서는 "multipartResolver" 이름의 빈으로 등록

```
<!-- 파일업로드 처리를 위한 multipartResolver bean 등록 -->
<beans:bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <beans:property name="defaultEncoding">
        <beans:value>utf-8</beans:value>
    </beans:property>
</beans:bean>
```


■ 파일 업로드 처리

- CommonsMultipartResolver 클래스의 프로퍼티

프로퍼티	타입	설명
maxUploadSize	long	최대 업로드 가능한 바이트 크기, -1은 제한이 없음을 의미 한다.
maxInMemorySize	int	디스크에 임시파일을 생성하기 전에 메모리에 보관할 수 있는 최대 바이트 크기, 기본값은 1024 바이트 이다.
defaultEncoding	String	요청을 파싱할 때 사용할 캐릭터 인코딩, 지정하지 않을 경우, <code>HttpServletRequest.setCharacterEncoding()</code> 메서드를 지정한 캐릭터 셋이 사용된다. 아무 값도 없을 경우 ISO-8859-1을 사용

pom.xml

```
<!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload -->
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.3</version>
</dependency>

<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.5</version>
</dependency>
```

■ 파일 업로드 처리

- **@RequestParam 어노테이션을 이용한 업로드 파일 접근**
 - 업로드한 파일을 전달 받는 방법은 @requestParam 어노테이션이 적용된 MultipartFile 타입의 파라미터를 사용.

```
@RequestMapping(  
    value = "/report/submitReport1.do",  
    method = RequestMethod.POST  
)  
public String submitReport1(  
    @RequestParam("studentNumber") String studentNumber,  
    @RequestParam("report") MultipartFile report) {  
    printInfo(studentNumber, report);  
    return "report/submissionComplete";  
}
```

■ 파일 업로드 처리

- MultipartFile인터페이스는 스프링에서 업로드 한 파일을 표현 할 때 사용되는 인터페이스
- MultipartFile인터페이스를 이용해서 업로드 한 파일의 이름, 실제 데이터, 파일의 크기 등을 구할 수 있다.

■ 파일 업로드 처리

- **MultipartHttpServletRequest** 이용한 업로드 파일 접근
 - MultipartHttpServletRequest 인터페이스를 사용.

```
@RequestMapping(  
    value = "/report/submitReport2.do",  
    method = RequestMethod.POST )  
public String submitReport2(MultipartHttpServletRequest request) {  
    String studentNumber = request.getParameter("studentNumber");  
    MultipartFile report = request.getFile("report");  
    printInfo(studentNumber, report);  
    return "report/submissionComplete";  
}
```

■ 파일 업로드 처리

- MultipartHttpServletRequest 인터페이스는 스프링이 제공하는 인터페이스로, Multipart 요청이 들어올 때 내부적으로 원본 HttpServletRequest 대신 사용되는 인터페이스.
- 웹 요청 정보를 구하기 위한 `getParameter()`나 `getHeader()`와 같은 메서드를 사용, 추가로 MultipartRequest 인터페이스가 제공한 Multipart 관련 메서드를 사용할 수 있다.
- MultipartRequest 인터페이스가 제공하는 메서드

메서드	설명
Iterator<String> getFileNames()	Dijqfhem 된 파일의 이름 목록을 제공하는 Iterator를 구함
MultipartFile getFile(String Name)	파라미터 이름이 name인 업로드 파일 정보를 구함
List<MultipartFile> getFiles(String name)	파라미터 이름이 name인 업로드 파일 정보 목록을 구함
MapM<String, MultipartFile> getFileMap()	파라미터이름을 키로 파라미터에 해당하는 파일 정보를 값으로 하는 Map을 구한다.

■ 파일 업로드 처리

- 커맨드 객체를 통한 업로드 파일 접근

- 커맨드 객체를 이용해도 업로드 한 파일을 전달받을 수 있다.
- 커맨드 클래스에 파라미터와 동일한 이름의 MultipartFile 타입 프로퍼티를 추가해 주어야 함.

```
public class ReportCommand {  
  
    private String studentNumber;  
    private MultipartFile report;  
  
    public String getStudentNumber() { return studentNumber; }  
    public void setStudentNumber(String studentNumber) {  
        this.studentNumber = studentNumber;  
    }  
    public MultipartFile getReport() { return report; }  
  
    public void setReport(MultipartFile report) {  
        this.report = report;  
    }  
  
}
```

■ 파일 업로드 처리

```
@RequestMapping(  
    value = "/report/submitReport3.do",  
    method = RequestMethod.POST  
)  
public String submitReport3(ReportCommand reportCommand) {  
    System.out.println(reportCommand.getStudentNumber());  
    System.out.println(reportCommand.getReport());  
    return "report/submissionComplete";  
}
```


■ 파일 업로드 처리

- **MultipartFile 인터페이스를 사용**

- MultipartFile 인터페이스는 업로드 한 파일 정보 및 파일 데이터를 표현하기 위한 용도로 사용.
- MultipartFile 인터페이스의 주요 메서드

메서드	설명
String getName()	파라미터 이름을 구한다.
String getOriginalFilename()	업로드 한 파일의 이름을 구한다.
boolean isEmpty()	업로드 한 파일이 존재하지 않는 경우 true를 리턴 한다.
long getSize()	업로드한 파일의 크기를 구한다.
byte[] getBytes() throws IOException	업로드 한 파일 데이터를 구한다.
InputStream getInputStream()	InputStream을 구한다.
void transferTo(File dest)	업로드 한 파일 데이터를 지정한 파일에 저장한다.

■ 파일 업로드 처리

mvctest.controller.ReportSubmissionController.java

```
package mvctest.controller;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.multipart.MultipartHttpServletRequest;
```

@Controller

```
public class ReportSubmissionController {
```

```
    @RequestMapping(
        value = "/report/upload",
        method = RequestMethod.GET )
    public String form() {
        return "report/submissionForm";
    }
}
```

■ 파일 업로드 처리

mvctest.controller.ReportSubmissionController.java

```
@RequestMapping(
    value = "/report/submitReport1",
    method = RequestMethod.POST )
public String submitReport1(
    @RequestParam("studentNumber") String studentNumber,
    @RequestParam("report") MultipartFile report) {
    printInfo(studentNumber, report);
    return "report/uploadOk";
}

private void printInfo(String studentNumber, MultipartFile report) {
    System.out.println(studentNumber + "가 업로드 한 파일: "
        + report.getOriginalFilename());
}
```

■ 파일 업로드 처리

mvctest.controller.ReportSubmissionController.java

```
@RequestMapping(  
    value = "/report/submitReport2",  
    method = RequestMethod.POST )  
public String submitReport2(MultipartHttpServletRequest request) {  
    String studentNumber = request.getParameter("studentNumber");  
    MultipartFile report = request.getFile("report");  
    printInfo(studentNumber, report);  
    return "report/uploadOk";  
}
```

```
@RequestMapping(  
    value = "/report/submitReport3",  
    method = RequestMethod.POST)  
public String submitReport3(ReportCommand reportCommand) {  
    printInfo(reportCommand.getStudentNumber(),  
reportCommand.getReport());  
    return "report/uploadOk";  
}  
}
```

■ 파일 업로드 처리

스프링 설정 파일 등록

...

```
<!-- 파일업로드 처리를 위한 multipartResolver bean 등록 -->
```

```
<beans:bean id="multipartResolver"
```

```
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
```

```
    <beans:property name="defaultEncoding">
```

```
        <beans:value>utf-8</beans:value>
```

```
    </beans:property>
```

```
</beans:bean>
```

```
<bean class="mvctest.controller.ReportSubmissionController" />
```

...

■ 파일 업로드 처리

mvctest.controller.ReportCommand.java

```
package mvctest.controller;

import org.springframework.web.multipart.MultipartFile;

public class ReportCommand {

    private String studentNumber;
    private MultipartFile report;

    public String getStudentNumber() { return studentNumber; }

    public void setStudentNumber(String studentNumber) {
        this.studentNumber = studentNumber;
    }

    public MultipartFile getReport() { return report; }

    public void setReport(MultipartFile report) { this.report = report; }
}
```

■ 파일 업로드 처리

views/report/uoloadForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Insert title here</title>
    <style type="text/css">
        body{ line-height: 240%;}
    </style>
</head>
<body>
<form action="submitReport1" method="post" enctype="multipart/form-data">
    학번 <input type="text" name="studentNumber"> <br>
    이름 <input type="text" name="studentName"> <br>
    리포트 파일 <input type="file" name="report"> <br>
    <input type="submit">
</form>
```

■ 파일 업로드 처리

views/report/uoloadForm.jsp

<h3>MultipartHttpServletRequest 사용</h3>

<form action="submitReport2" method="post" enctype="multipart/form-data">

학번: <input type="text" name="studentNumber" />

리포트파일: <input type="file" name="report" />

<input type="submit" />

</form>

<h3>커맨드 객체 사용</h3>

<form action="submitReport3" method="post" enctype="multipart/form-data">

학번: <input type="text" name="studentNumber" />

리포트파일: <input type="file" name="report" />

<input type="submit" />

</form>

</body>

■ 파일 업로드 처리

views/report/uploadOk.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>
학번 : ${studentNumber}<br>
이름 : ${studentName}<br>
파일이름 : ${fileName}<br>
파일사이즈 : ${fileSize}<br>

</h1>
</html>
```

■ Open Project

- Open Project 를 SPRING @MVC 모듈을 이용해서 구현해 봅시다.