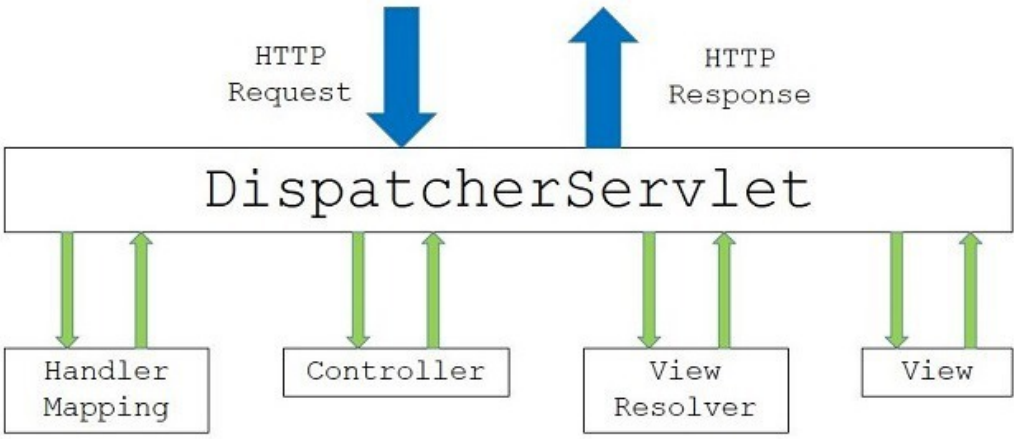


[Spring MVC 일반적인 구조]

**-Spring MVC 처리 순서-**

**1. 클라이언트 (Client) 가 서버에 어떤 요청을 한다면 스프링에서 제공하는 DispatcherServlet 이라는 클래스 (일종의 front controller)가 요청을 가로챈다.**
**(web.xml 을 살펴보면 모든 url( / )에 서블릿 매핑을 하여 모든 요청을 DispatcherServlet가 가로채게 해둠)**
**(변경가능)**

**2. 요청을 가로챈 DispatcherServlet는 HandlerMapping(URL 분석등) 에게 어떤 컨트롤러에게 요청을 위임하면 좋을지 물어본다.**
**(servlet-context.xml 에서 @Controller로 등록한 것들을 스캔해서 찾아준다.)**

**3. 요청에 매핑된 컨트롤러가 있다면 @RequestMapping를 통하여 요청을 처리할 메서드에 도달한다.**

**4. 컨트롤러에서는 해당 요청을 처리할 Service를 주입 (DI) 받아 비즈니스로직을 Service 에게 위임한다.**

**5. Service 에서는 요청에 필요한 작업 대부분 (코딩)을 담당하며 데이터베이스에 접근이 필요하면 DAO를 받아 DB처리는 DAO에게 위임한다.**

**6. DAO는 mybatis (또는 hibernate 등) 설정을 이용해서 SQL 쿼리를 날려 DB의 정보를 받아 서비스에게 다시 돌려준다.**

**7. 모든 로직을 끝낸 서비스가 결과를 컨트롤러에게 넘긴다.**

**8. 결과를 받은 컨트롤러는 Model 객체에 어떤 view(jsp) 파일을 보여줄 것인지등의 정보를 담아 DispatcherServlet 에게 보낸다.**

**9. DispatcherServlet는 ViewResolver에게 받은 view(jsp) 파일에 대한 정보를 넘긴다.**

**10. ViewResolver는 해당 JSP를 찾아서 (응답할 View를 찾아서) DispatcherServlet에게 알려준다.**
**(servlet-context.xml 에서 suffix, prefix를 통해 /WEB-INF/views/index.jsp 이렇게 만들어 주는것도 ViewResolver 이다.)**

**11. DispatcherServlet은 응답할 View에게 Render를 지시하고 View는 응답 로직을 처리한다.**

**12. 결과적으로 DispatcherServlet이 클라이언트에게 렌더링된 View를 응답한다.**

**스프링의 환경설정 파일 ( web.xml ) 중 일부**

```
1  <!-- 스프링의 환경설정 파일 로딩 -->                                    CS
2      <context-param>
3          <param-name>contextConfigLocation</param-name>
```

```
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <!-- 서블릿의 환경설정 -->
    <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>
```

Colored by Color Scripter

## 스프링의 DB연결 ( root-context.xml ) 중 일부

```
<!-- 드라이버 클래스 이름이 변경됨 -->
       <property name="driverClassName" value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
       <!-- 연결문자열에 log4jdbc가 추가됨 -->
       <property name="url"
           value="jdbc:log4jdbc:oracle:thin:@localhost:1521:xe" />
       <property name="username" value="hr" />
       <property name="password" value="hr" />
   </bean>
```

Colored by Color Scripter cs

## ( servlet-context.xml ) 중 일부

```
<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>
    <!-- 스프링 빈을 태그로 등록하지 않고 자동으로 검색(auto scan) -->
    <context:component-scan base-package="com.example.spring01" /> //이 프로젝트에 있는 파일들을 실행한다.

</beans:beans>
```

Colored by Color Scripter cs

## HomeController.java ( com.example.spring01에 뒤에 /를 붙이면 이 부분이 실행된다)

```
@RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale);

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate );

        return "home";  //home.jsp 로 이동해야한다.
        return "/WEB-INF/views/home"; //가 원래 코드이지만 코드가 길어서 위의 코드로 대체 하였다.
    }

}
```

Colored by Color Scripter cs

bean을 xml로 등록하는 방법

```
<bean id = "참조변수" class="클래스 이름">
</bean>

<bean id ="homeController" class = "com.example.spring01.HomeController">
</bean>
==> HomeController homeController = new HomeController( );
```

*bean을 자동으로 검색해서 등록

<context:component-scan base-package="기준 패키지 경로" />

@Controller : 컨트롤러 빈
@Repository : dao (데이터베이스 관련 작업) bean
@Service : 서비스 (비즈니스 관련 로직) bean
@Inject : 의존관계 주입

## -스프링의 디렉토리 구조-

**src/main/java - 자바 코드 (Controller, Model, Service)**

**src/main/resources - 자바 코드에서 참조하는 리소스 파일들**
**sqlMapConfig.xml, mybatis의 mapper**

**src/test/java - 테스트 관련 코드**

**src/test/resources - 테스트 코드에서 참조하는 리소스 파일들**

**src/main/webapp - 웹 서비스 루트 디렉토리 (외부에서 접근 가능)**
**(주소에 web-inf가 들어가 있으면 실행이 되지 않는다. 클래스를 경유해서 가는건 가능하지만 직접적으로 접근하는건 되지 않는다.)**
**(스프링에서는 jsp와는 다르게 안쪽에 숨기는것을 권장한다.)**

**src/main/webapp/resources - js, css, iamge 등의 웹 리소스 파일**

**src/main/webapp/WEB-INF/classes - 컴파일된 클래스**

**src/main/webapp/WEB-INF/spring - 스프링의 환경설정 파일**

**src/main/webapp/WEB-INF/spring/root-context.xml - 서블릿과 관련되지 않은 모든 리소스에 대한 설정**

**src/main/webapp/WEB-INF/spring/appServlet/servelt-context.xml - 서블릿과 관련된 리소스에 대한 설정**

**src/main/webapp/WEB-INF/views - html, jsp 페이지**

**src/main/webapp/WEB-INF - 외부에서 접근 불가능 (보안을 위해서)**
**컨트롤러를 경유해서 접근 가능**

**pom.xml - 메이븐에서 참조하는 설정 파일**

## -스프링의 주요 특징-

POJO (Plain Old Java Object) 기반의 구성 : 별도의 API가 필요하지 않은 일반적인 자바 코드를 이용하여 개발 가능

의존성 주입 (DI)을 통한 객체 간의 관계 구성

**의존성 주입이란 ?**

**의존성 주입(Dependency Injection, DI)은 프로그래밍에서 구성요소간의 의존 관계가 소스코드 내부가 아닌 외부의 설정파일 등을 통해 정의되게 하는 디자인 패턴 중의 하나이다.**

AOP (Aspect Oriented Programming) 지원 : 반복적인 코드를 줄이고 개발자가 비즈니스 로직에만 집중할 수 있도록 지원함

편리한 MVC 구조

WAS에 종속적이지 않은 개발 환경

### 1. IoC ( Inversion of Control, 제어의 역전 ) - 객체에 대한 제어권

기존에는 개발자에게 제어권이 있었음 (new 연산자)
객체의 제어권을 스프링에게 넘김
인스턴스의 라이프 사이클 (생성부터 소멸까지)을 개발자가 아닌 스프링 프레임웍이 담당

## 2. DI ( Dependency Injection, 의존관계 주입 )

객체 간의 의존성을 개발자가 설정하는 것이 아닌
스프링 컨테이너가 주입시켜 주는 기능
객체를 쉽게 확장하고 재사용할 수 있음

**ㄴ 의존 관계**

B의 코드가 변경되면 A의 코드도 변경되는 경우
A와 B는 의존관계가 있다.
A는 B에 의존한다.
강한 결합은 (결합도가 높으면) 유지보수를 어렵게 한다.
한 곳의 수정 -> 나비효과

**ㄴ 강한 결합과 느슨한 결합**

```
// 생성자가 private으로 바뀌면
// MemberUse 클래스도 수정해야 함

class MemberUse {
  public MemberUse( ){
    Member m = new Member();
  }
 }
 public class Member{
  String userid;
  String passwd;
  String name;
  private Member(){
  }
}
```

```
// 생성자가 private으로 바뀌어도
// 영향을 받지 않음

class MemberUse {
  public MemberUse(Member m){

  }
 }
 public class Member{
  String userid;
  String passwd;
  String name;
  private Member(){
  }
}
```