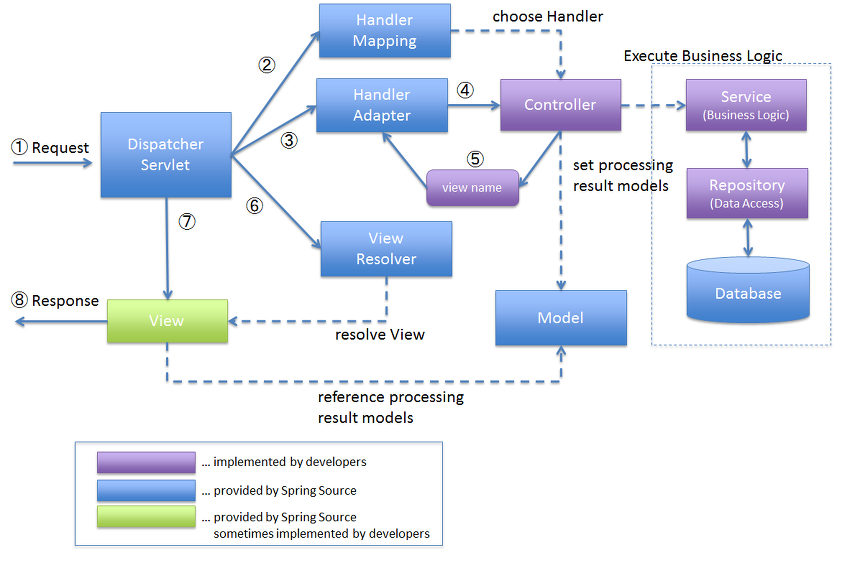
**Spring MVC의 동작 원리**



1. DispatcherServlet이 브라우저로부터 요청을 받는다.

2. DispatcherServlet은 요청된 URL을 HandlerMapping 객체에 넘기고, 호출해야 할 Controller 메소드(핸들러) 정보를 얻는다.

3. DispatcherServlet이 HandlerAdapter 객체를 가져온다.

4. HandlerAdapter 객체의 메소드를 실행한다.

※ 보다 정확하게 표현한다면, HandlerMapping은 DispatcherServlet로부터 전달된 URL을 바탕으로 HandlerAdapter 객체를 포함하는 HandlerExecutionChain 객체를 생성하며, 이후 DispatcherServlet이 HandlerExecutionChain 객체로부터 HandlerAdapter 객체를 가져와서 해당 메소드를 실행하게 된다.

5. Controller 객체는 비즈니스 로직을 처리하고, 그 결과를 바탕으로 뷰(ex. JSP)에 전달할 객체를 Model 객체에 저장한다. DispatcherServlet에게 view name을 리턴한다.

6. DispatcherServlet은 view name을 View Resolver에게 전달하여 View 객체를 얻는다.

7. DispatcherServlet은 View 객체에 화면 표시를 의뢰한다.

8. View 객체는 해당하는 뷰(ex. JSP, Thymeleaf)를 호출하며, 뷰는 Model 객체에서 화면 표시에 필요한 객체를 가져와 화면 표시를 처리한다.

* 실질적으로 Controller를 실행하는 것은 Handler Adapter이다.
* View Resolver는 "전략 객체"   
  View Resolver에게 전달되는 정보는 view name뿐만이 아니라, header 정보(accept 등)도 전달된다
* header 정보 내의 Accept는 기본적으로는 HTML에 따라서는 JSON, XML인 경우도 있다.
* View Resolver는 전달된 정보를 바탕으로 사용자에게 보여줄 view가 무엇인지를 결정한다.
* JSP의 경우 JstlView 객체가 생성된다. JstlView 객체가 "abcd.jsp"에 포워딩하여 결과를 보여준다(JSP 객체를 생성하는 것이 아니다).

**Spring MVC의 구성 요소**

|  |  |
| --- | --- |
| **구성요소** | **역할** |
| DispatcherServlet | - Front Controller를 담당   - 모든 HTTP 요청을 받아들여서 다른 객체들 사이의 흐름을 제어   - Spring MVC에서 제공하는 DispatcherServlet 클래스를 그대로 적용 |
| HandlerMapping | - 클라이언트의 요청을 바탕으로 어떤 Handler(Controller 메소드)를 실행할지 결정 |
| Model | - Controller에서 View로 넘겨줄 객체가 저장되는 곳.   - key(String)-value pair |
| ViewResolver | - view name을 바탕으로 View 객체를 결정 |
| View | - 뷰에 화면 표시 처리를 의뢰 |
| 비즈니스 로직 | - 비즈니스 로직을 실행 |
| Controller | - 클라이언트 요청에 맞는 presentation layer의 처리를 실행해야 한다. |
| 뷰(ex. JSP, Thymeleaf) | - 클라이언트에게 화면을 표시한다. |

**ApplicationConfig는 언제 생성되는가?**

web.xml에는 ContextLoaderListener와 DispatcherServlet의 2가지 설정(기본적으로 xml)이 있다.

WAS를 처음 작동시키면, web.xml(컨테이너)를 로딩하여 Servlet Container가 구동되고,

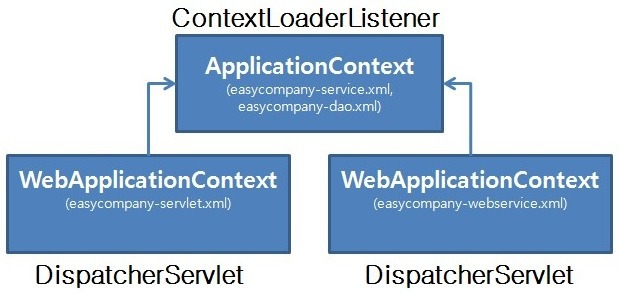
Servlet Container는 ContextLoaderListener 객체를 자동으로 메모리에 생성(pre-loading)한다.

ContextLoaderListener 객체는 먼저 <context-param>으로 등록된 설정 파일(ex. ApplicationConfig.java)을 읽어서 첫번째 Spring Container(Root Container)를 구동하고 비즈니스 객체들(Service, DAO)을 생성한다.

이를 부모 컨테이너라고 부른다. 부모 컨테이너는 @ComponentScan으로 DAO・Service Bean을 찾고 메모리에 생성한다.

 다음으로, '/' 요청(=> 모든 요청)이 들어오면, Servlet Container가 DispatcherServlet 객체를 생성한다. DispatcherServlet이 내부적으로 init()을 호출할 때, <init-param>으로 등록된 Spring 설정 파일을 찾아 두번째 Spring Container를 구동한다.

이를 자식 컨테이너라고 부르기로 하며, @ComponentScan으로 'Controller' Bean(+ 'HandlerMapping', 'ViewResolver')을 찾고 메모리에 생성한다.



* 자식 컨테이너는 부모 컨테이너의 것을 사용할 수 있다.   
  (부모-자식은 상속의 개념은 아니다.)
* url-pattern을 다르게 해서 여러 개의 DispatcherServlet이 있을 수 있다(자주 있는 경우는 아님).
* DispatcherServlet이 Controller들을 bean으로 등록하고, HandlerMapping 객체에 컨트롤러 내의 메소드마다 붙은 @RequestMapping 정보를 담아 둔다.
* 원래는 '/'에 대한 요청(즉, 모든 요청)은 WAS(ex. Tomcat)가 제공하는 DefaultServlet이 처리한다.   
  요청 URL에 해당하는 HandlerMapping 정보를 찾아 해당 Controller가 비즈니스 로직을 처리한다.   
  해당 HandlerMapping 정보가 없으면, 로직 처리를 DefaultServlet에게 넘긴다. 그래도 없으면 404 error가 발생한다.