Spring MVC

Charles Zhang

Fall 2019

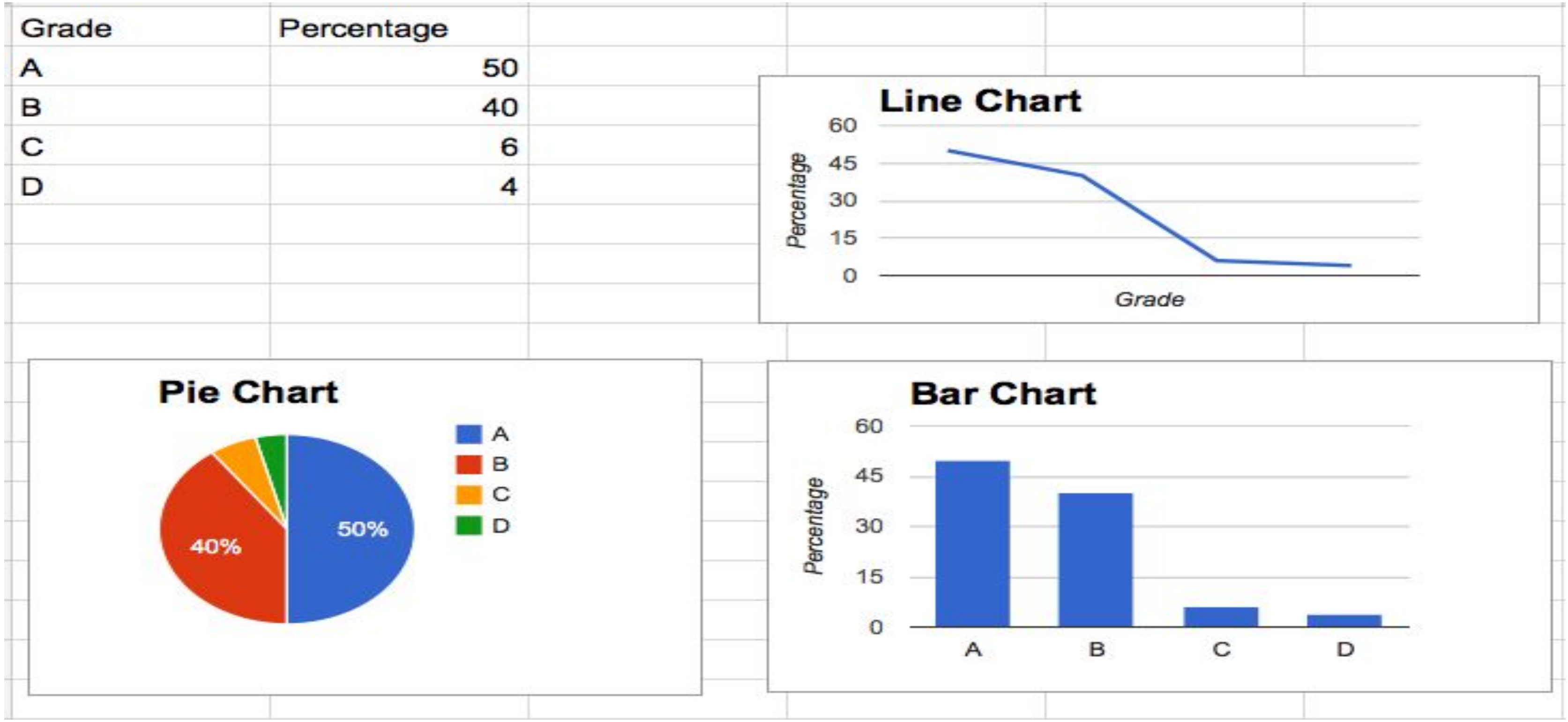
What did we cover in last lecture?

- What’s a crosscutting concern?
- What is AOP?
- What is an advice?
- What is a joinpoint?
- What is a pointcut?
- What are the method advice types?
- How to express a pointcut?
- What is aspect weaving?

Outline

- Introduction to MVC
- Spring MVC basics
- Servlet routing
- Handler interceptors
- View resolution
- Form handling and AJAX

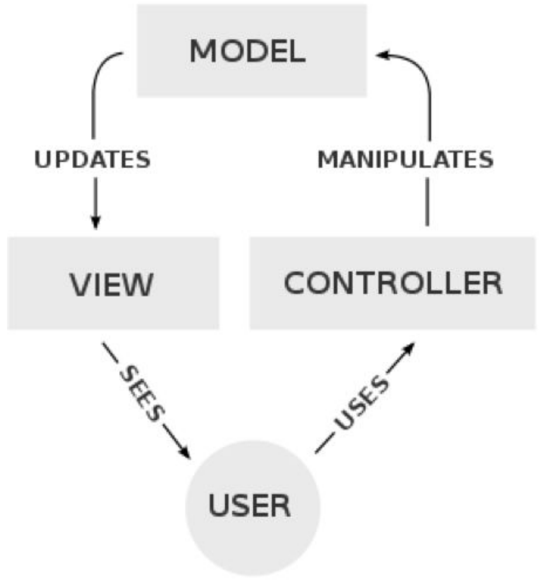
The same data can be presented in different ways



Grade	Percentage
A	50
B	40
C	6
D	4

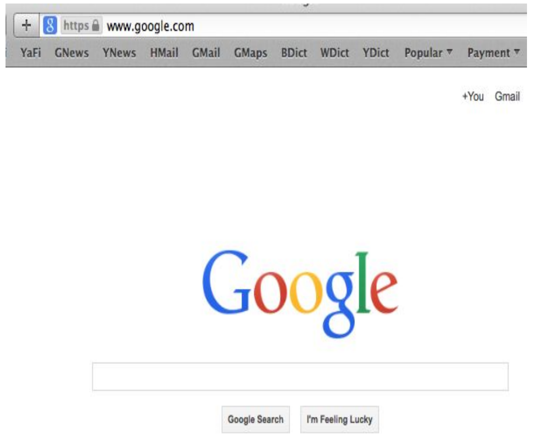
What is model-view-controller (MVC)?

- MVC: Design *pattern* for implementing user interfaces
- Model
  - Data and interfaces to manipulate data
- View
  - Any output representation of data
- Controller
  - Connects and controls view and model



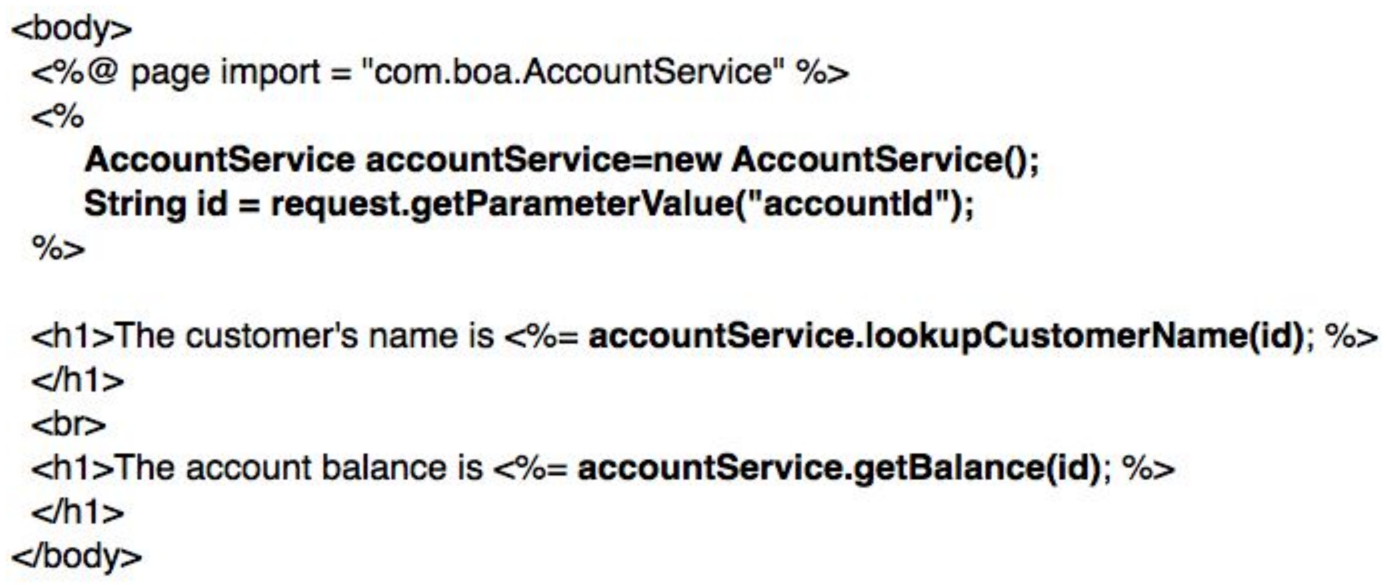
Why MVC?
- Separation of concerns

Where are M, V, and C?


https
www.google.com
YaFi GNews YNews HMail GMail GMaps BDict WDict YDict Popular Payment
+You Gmail
Google
Google Search
I'm Feeling Lucky

- May completely reside in the client
- May mostly live on the server

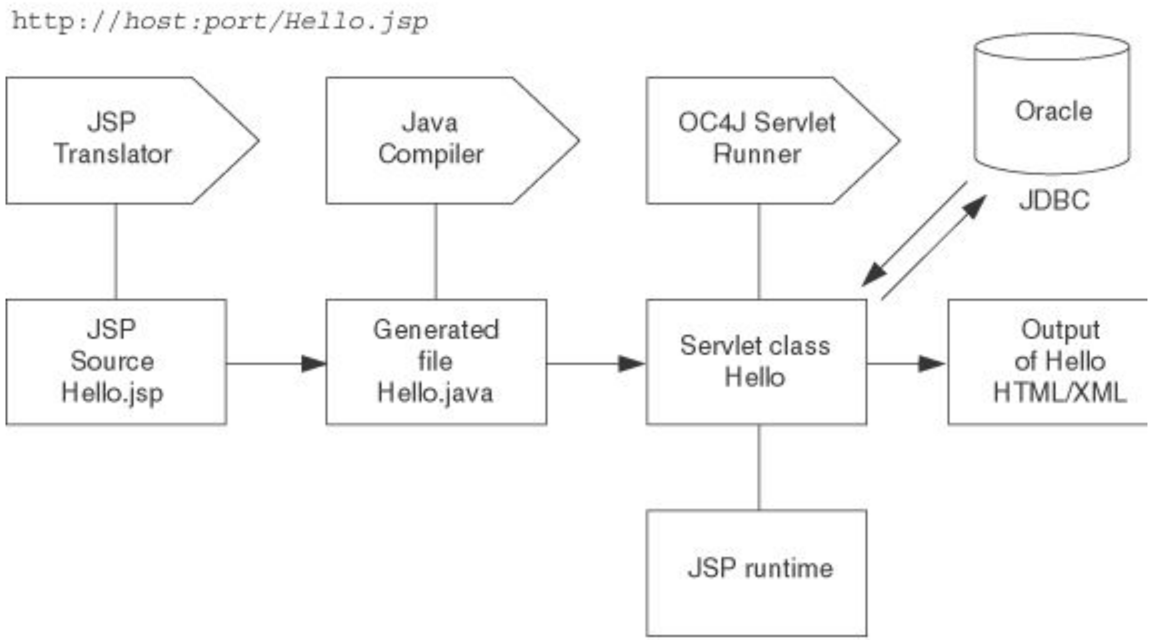
Java Servlet Page (JSP)



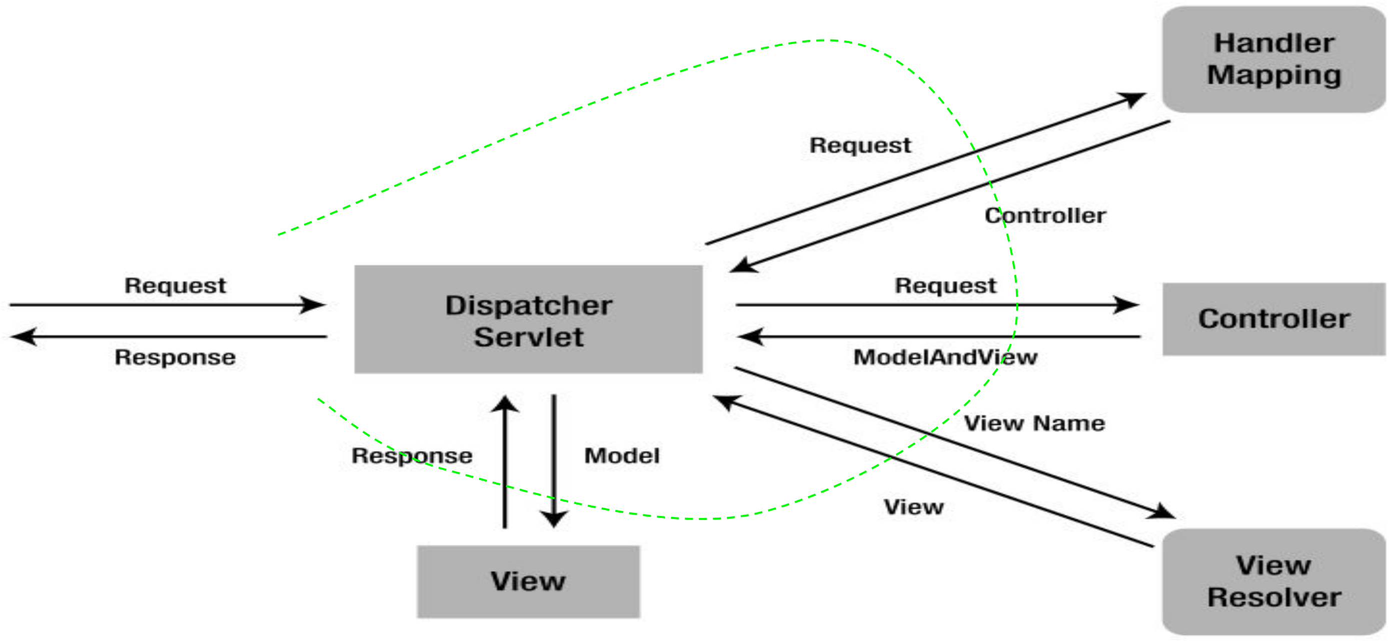
```
<body>
  <%@ page import = "com.boa.AccountService" %>
  <%
       AccountService accountService=new AccountService();
       String id = request.getParameterValue("accountId");
  %>

  <h1>The customer's name is <%= accountService.lookupCustomerName(id); %>
  </h1>
  <br>
  <h1>The account balance is <%= accountService.getBalance(id); %>
  </h1>
</body>
```

Can be a mix of model, view, and controller

How is a jsp served?

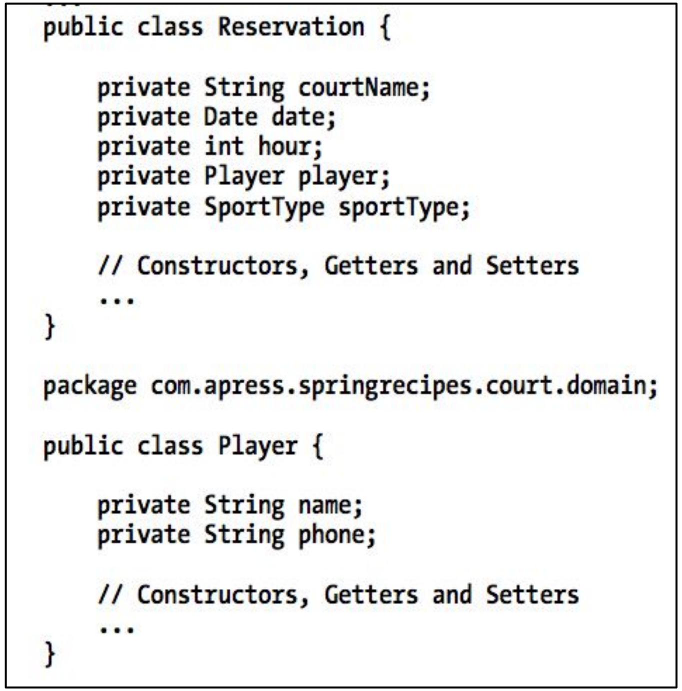


MVC in Spring

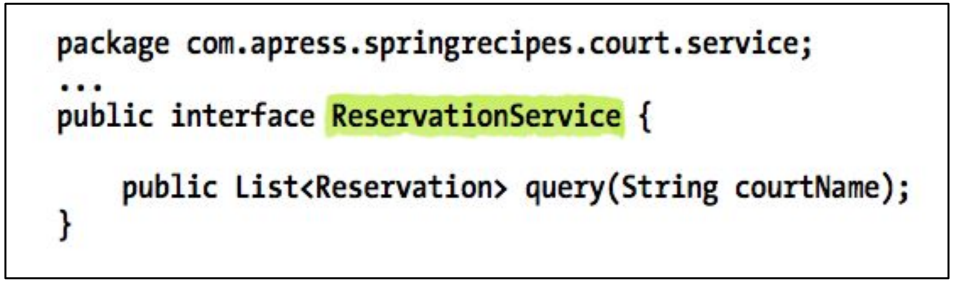


Flow of request handling in Spring MVC

Model for our examples



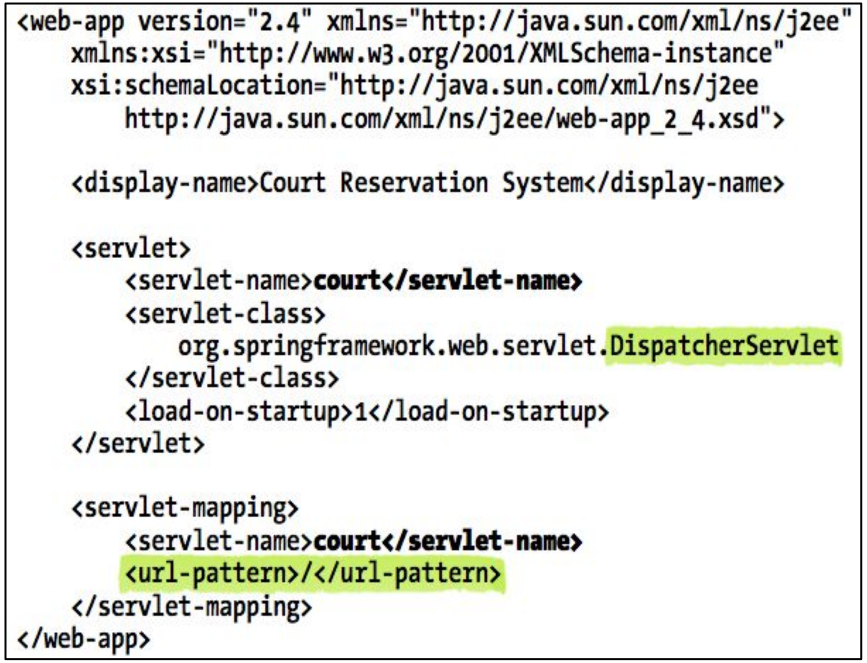
```
...
public class Reservation {

    private String courtName;
    private Date date;
    private int hour;
    private Player player;
    private SportType sportType;

    // Constructors, Getters and Setters
    ...
}

package com.apress.springrecipes.court.domain;

public class Player {

    private String name;
    private String phone;

    // Constructors, Getters and Setters
    ...
}
```



```
package com.apress.springrecipes.court.service;
...
public interface ReservationService {

    public List<Reservation> query(String courtName);
}
```

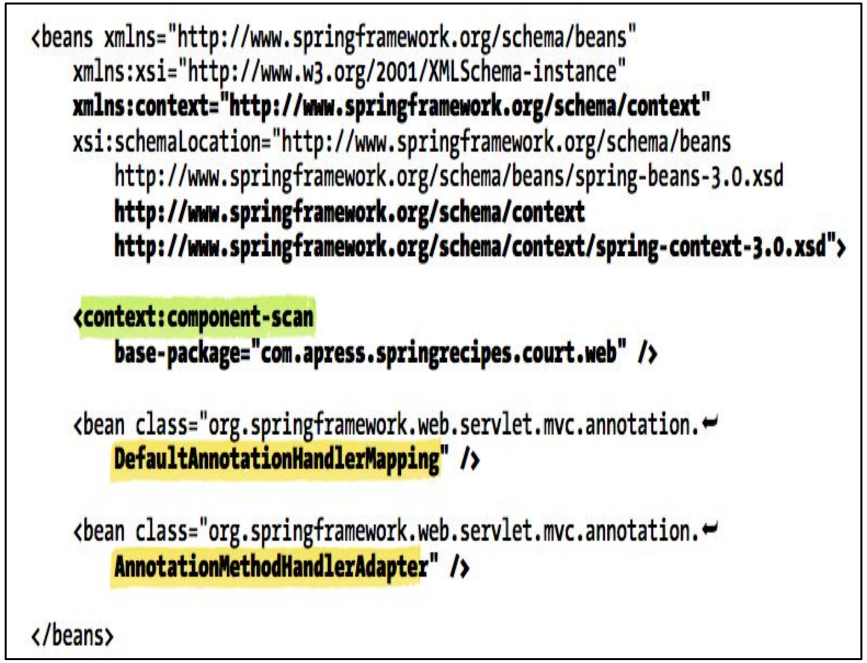
Domain data schemas and service interfaces

Spring MVC setup



```xml
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Court Reservation System</display-name>

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

    <servlet-mapping>
        <servlet-name>court</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

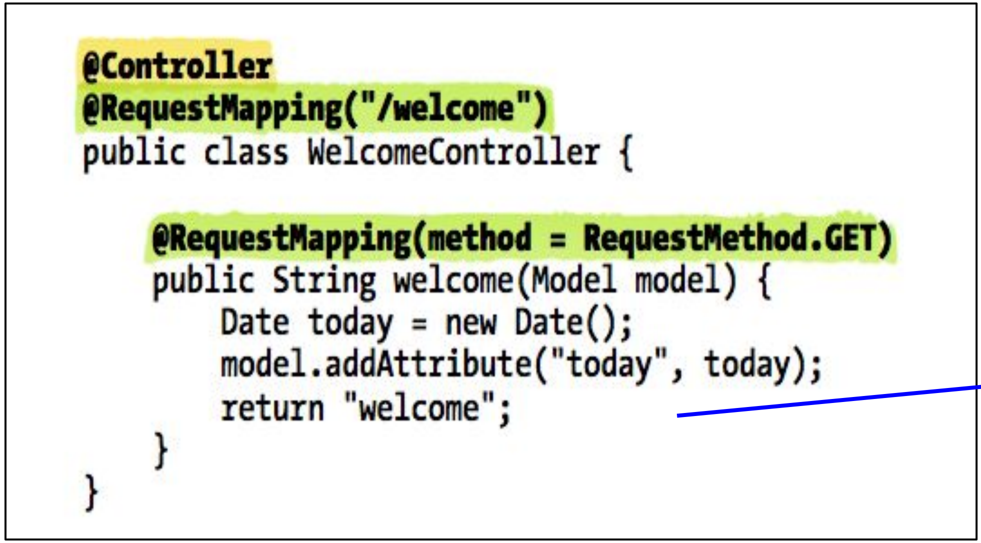
web.xml



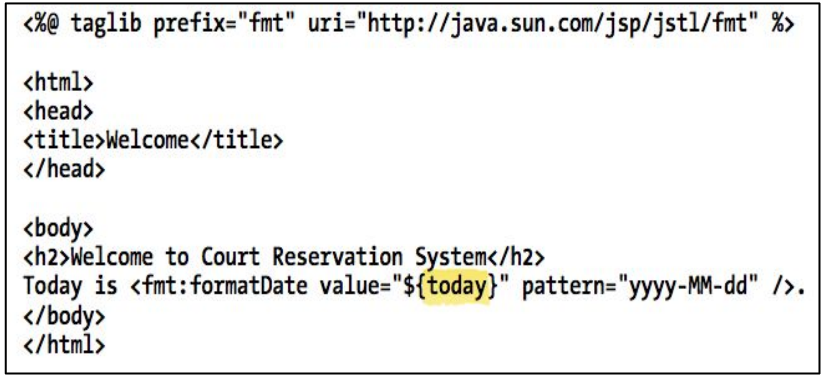
```xml
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan
        base-package="com.apress.springrecipes.court.web" />

    <bean class="org.springframework.web.servlet.mvc.annotation.↵
        DefaultAnnotationHandlerMapping" />

    <bean class="org.springframework.web.servlet.mvc.annotation.↵
        AnnotationMethodHandlerAdapter" />

</beans>
```

court-service.xm
l

MVC Controller



```
@Controller
@RequestMapping("/welcome")
public class WelcomeController {

    @RequestMapping(method = RequestMethod.GET)
    public String welcome(Model model) {
        Date today = new Date();
        model.addAttribute("today", today);
        return "welcome";
    }
}
```

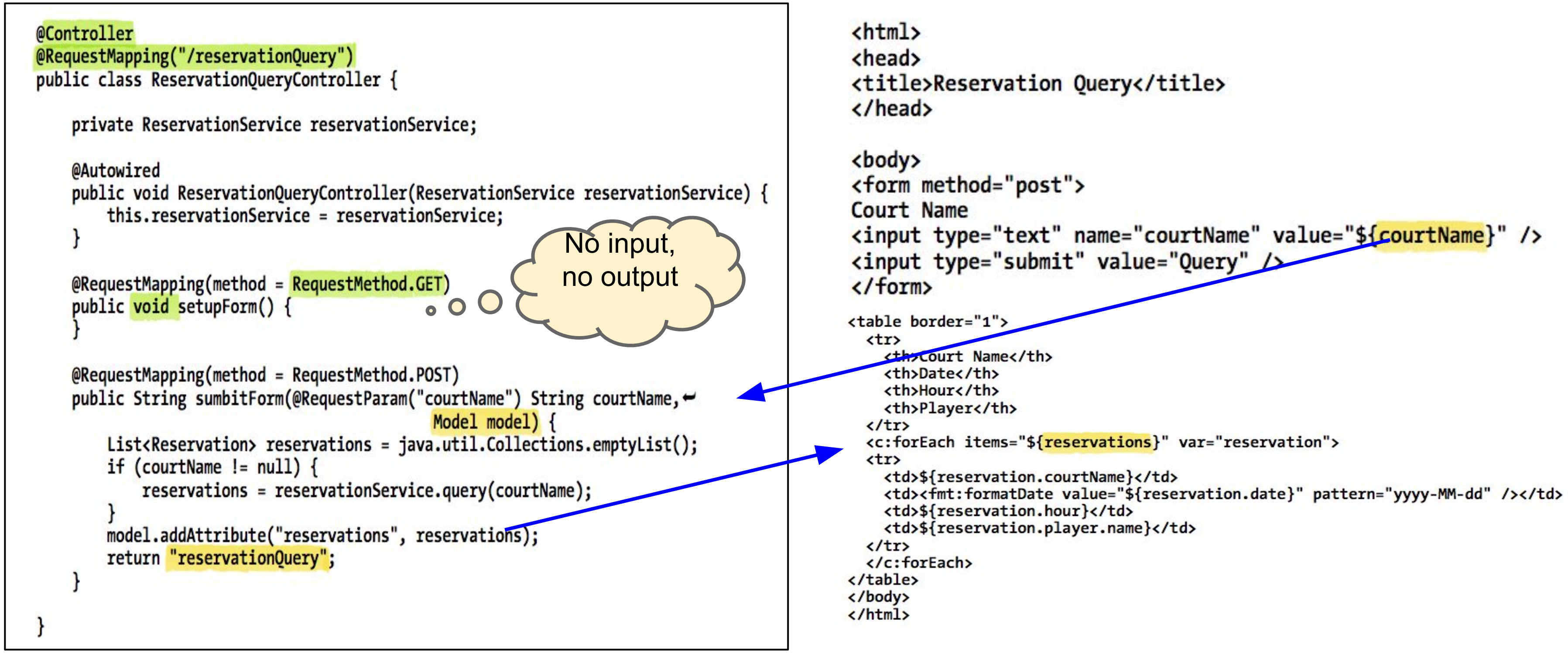


```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<html>
<head>
<title>Welcome</title>
</head>

<body>
<h2>Welcome to Court Reservation System</h2>
Today is <fmt:formatDate value="${today}" pattern="yyyy-MM-dd" />.
</body>
</html>
```

welcome.jsp

- @Controller defines the class as a Spring MVC controller
- Return value decides the view jsp
- Values are passed through *model* object

MVC flow revisited

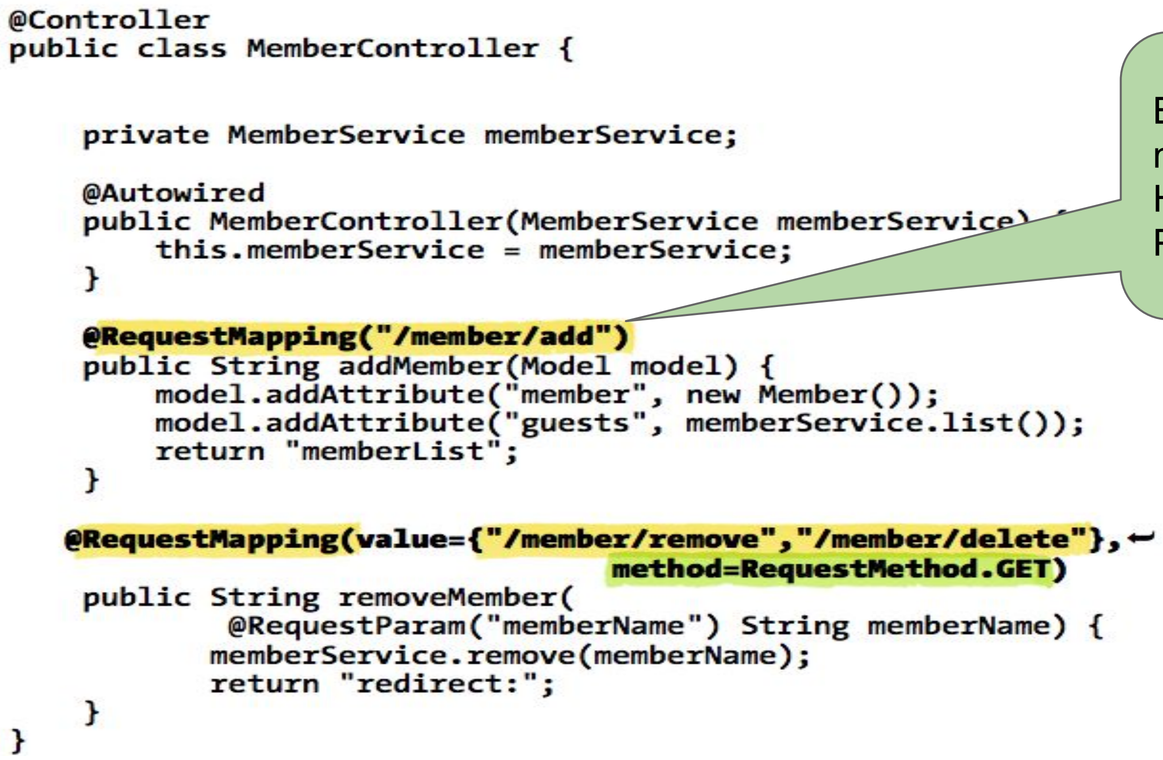


```
@Controller
@RequestMapping("/reservationQuery")
public class ReservationQueryController {

    private ReservationService reservationService;

    @Autowired
    public void ReservationQueryController(ReservationService reservationService) {
        this.reservationService = reservationService;
    }

    @RequestMapping(method = RequestMethod.GET)
    public void setupForm() {
    }

    @RequestMapping(method = RequestMethod.POST)
    public String sumbitForm(@RequestParam("courtName") String courtName,↵
                            Model model) {
        List<Reservation> reservations = java.util.Collections.emptyList();
        if (courtName != null) {
            reservations = reservationService.query(courtName);
        }
        model.addAttribute("reservations", reservations);
        return "reservationQuery";
    }

}
```

```
<html>
<head>
<title>Reservation Query</title>
</head>

<body>
<form method="post">
Court Name
<input type="text" name="courtName" value="${courtName}" />
<input type="submit" value="Query" />
</form>

<table border="1">
  <tr>
    <th>Court Name</th>
    <th>Date</th>
    <th>Hour</th>
    <th>Player</th>
  </tr>
  <c:forEach items="${reservations}" var="reservation">
  <tr>
    <td>${reservation.courtName}</td>
    <td><fmt:formatDate value="${reservation.date}" pattern="yyyy-MM-dd" /></td>
    <td>${reservation.hour}</td>
    <td>${reservation.player.name}</td>
  </tr>
  </c:forEach>
</table>
</body>
</html>
```

reservationQuery.jsp

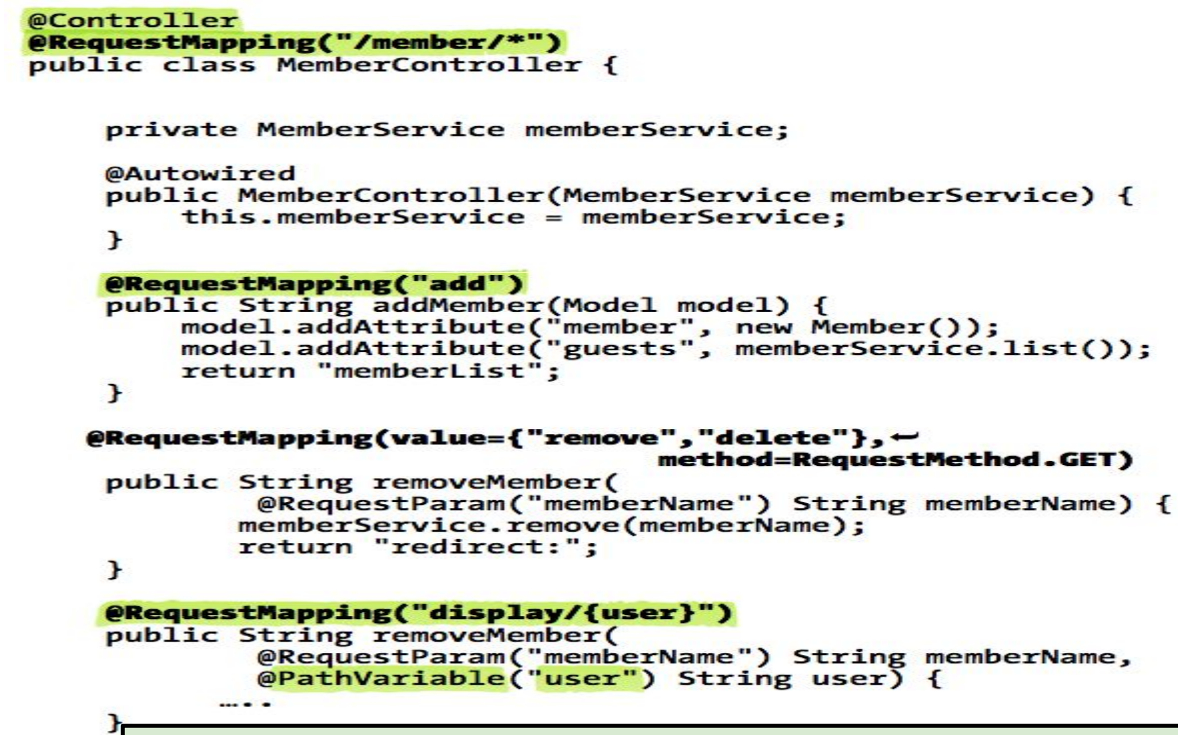
Request mapping by method



```
@Controller
public class MemberController {


    private MemberService memberService;

    @Autowired
    public MemberController(MemberService memberService) {
        this.memberService = memberService;
    }

    @RequestMapping("/member/add")
    public String addMember(Model model) {
        model.addAttribute("member", new Member());
        model.addAttribute("guests", memberService.list());
        return "memberList";
    }

   @RequestMapping(value={"/member/remove","/member/delete"},↵
                                          method=RequestMethod.GET)
    public String removeMember(
            @RequestParam("memberName") String memberName) {
           memberService.remove(memberName);
           return "redirect:";
    }
}
```

By default, supported methods include GET, HEAD, POST, PUT, PATCH and DELETE

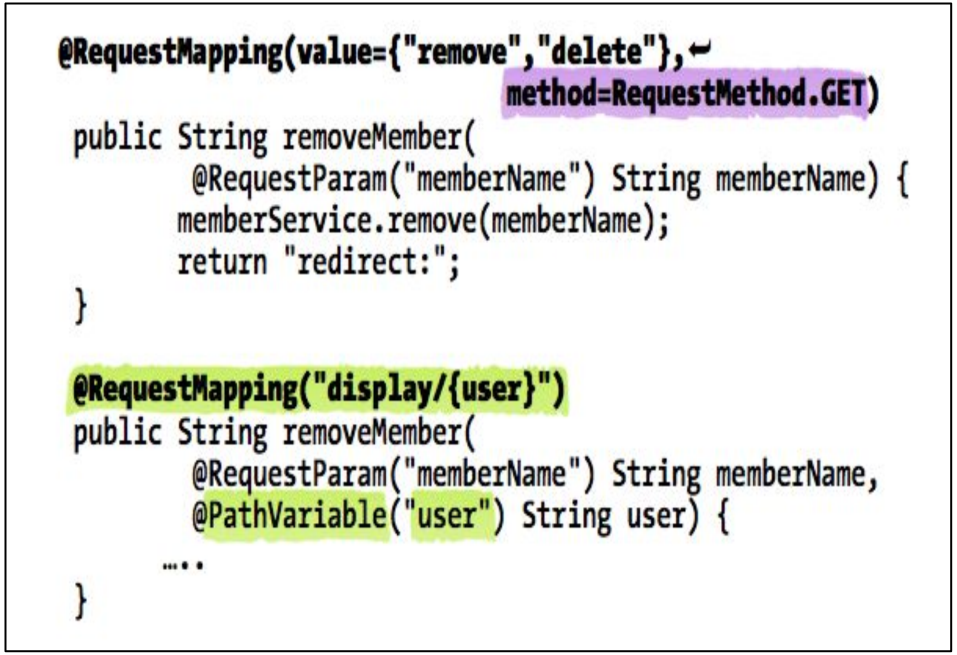
Request mapping by class



```
@Controller
@RequestMapping("/member/*")
public class MemberController {

    private MemberService memberService;

    @Autowired
    public MemberController(MemberService memberService) {
        this.memberService = memberService;
    }

    @RequestMapping("add")
    public String addMember(Model model) {
        model.addAttribute("member", new Member());
        model.addAttribute("guests", memberService.list());
        return "memberList";
    }

  @RequestMapping(value={"remove","delete"},↵
                                method=RequestMethod.GET)
    public String removeMember(
                @RequestParam("memberName") String memberName) {
              memberService.remove(memberName);
              return "redirect:";
    }

    @RequestMapping("display/{user}")
    public String removeMember(
                @RequestParam("memberName") String memberName,
                @PathVariable("user") String user) {
            .....
    }
```

Class path and method path concatenated together!

Request mapping by request type



```
@RequestMapping(value={"remove","delete"},↵
                        method=RequestMethod.GET)
 public String removeMember(
         @RequestParam("memberName") String memberName) {
        memberService.remove(memberName);
        return "redirect:";
 }

 @RequestMapping("display/{user}")
 public String removeMember(
         @RequestParam("memberName") String memberName,
         @PathVariable("user") String user) {
        .....
 }
```

- **GET**
- HEAD
- **POST**
- *PUT*
- *Patch*
- *DELETE*
- TRACE
- OPTIONS
- CONNECT

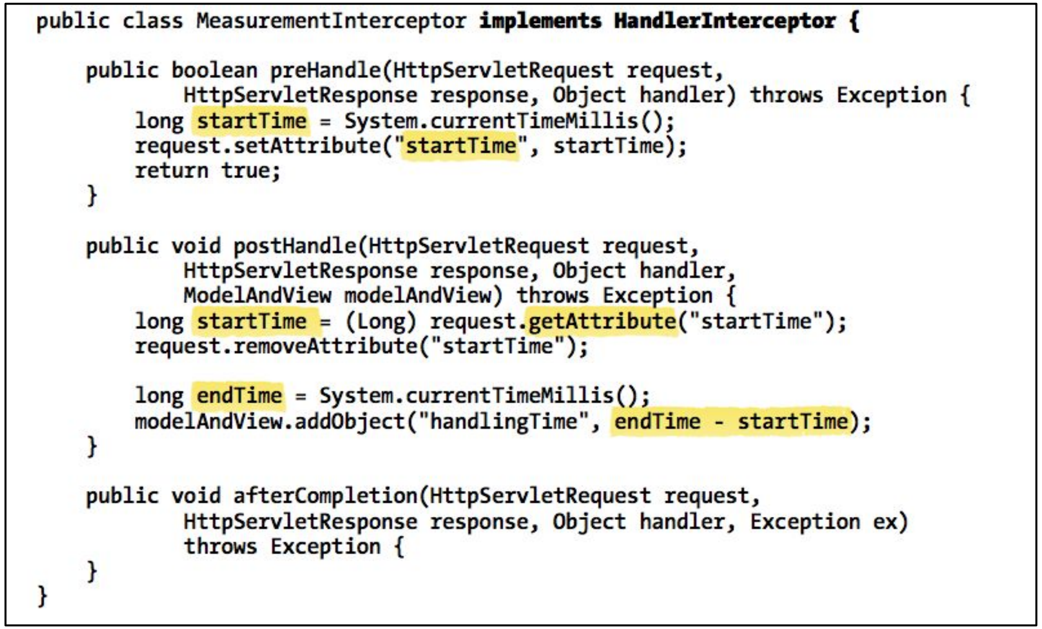
GET and POST are the **most** used HTTP request types

Intercepting requests with handler interceptors

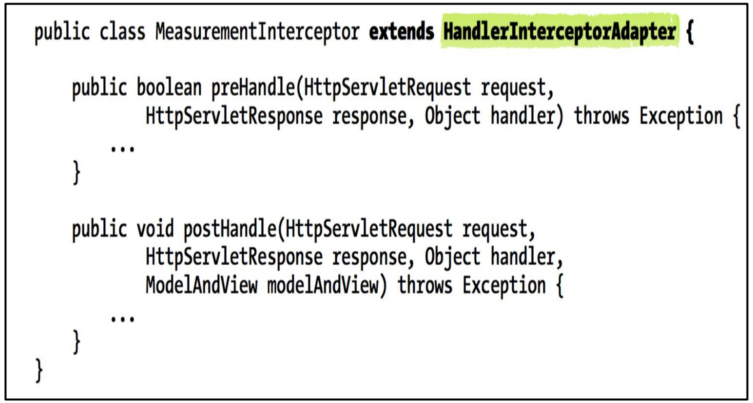
- Use cases
  - Logging
  - Performance profiling
  - Access control (IP/user)
  - ...

- Interceptor types
  - preHandle
  - postHandle
  - afterCompletion

Implement handler interceptors



```
public class MeasurementInterceptor implements HandlerInterceptor {

    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws Exception {
        long startTime = System.currentTimeMillis();
        request.setAttribute("startTime", startTime);
        return true;
    }

    public void postHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler,
            ModelAndView modelAndView) throws Exception {
        long startTime = (Long) request.getAttribute("startTime");
        request.removeAttribute("startTime");

        long endTime = System.currentTimeMillis();
        modelAndView.addObject("handlingTime", endTime - startTime);
    }

    public void afterCompletion(HttpServletRequest request,
            HttpServletResponse response, Object handler, Exception ex)
            throws Exception {
    }
}
```



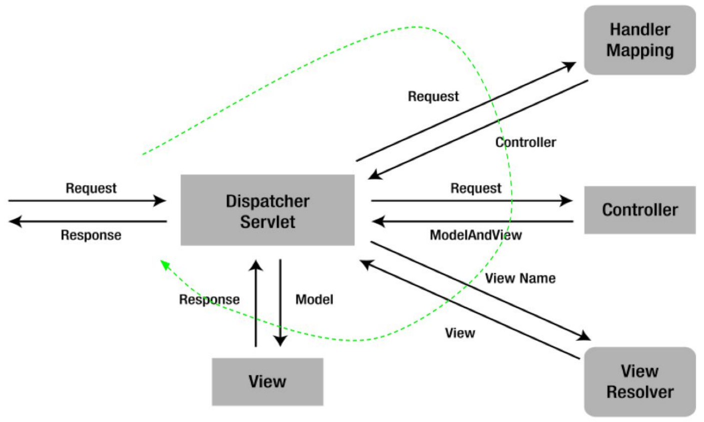
```
public class MeasurementInterceptor extends HandlerInterceptorAdapter {

    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws Exception {
        ...
    }

    public void postHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler,
            ModelAndView modelAndView) throws Exception {
        ...
    }
}
```

Adapter classes make it easier to implement an interface

How to turn on interceptors?

```
<beans ...>
    ...
    <bean id="measurementInterceptor"
        class="com.apress.springrecipes.court.web.MeasurementInterceptor" />

    <bean
        class="org.springframework.web.servlet.mvc.annotation.↵
                   DefaultAnnotationHandlerMapping">
        <property name="interceptors">
            <list>
                <ref bean="measurementInterceptor" />
            </list>
        </property>
        ...
    </bean>
</beans>
```

Resolving views by names



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

```
welcome ‘ /WEB-INF/jsp/welcome.jsp
reservationQuery ‘ /WEB-INF/jsp/reservationQuery.jsp
```

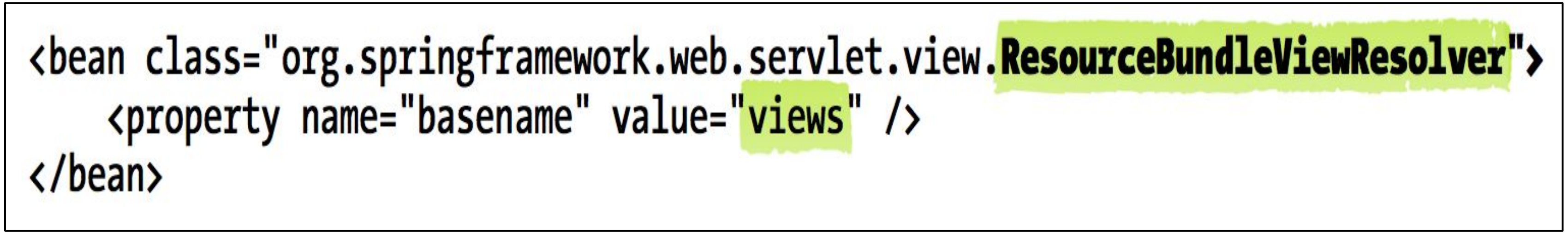
Simple strategy for DispatcherServlet to resolve views: Just by name

Resolving Views from an XML Configuration File

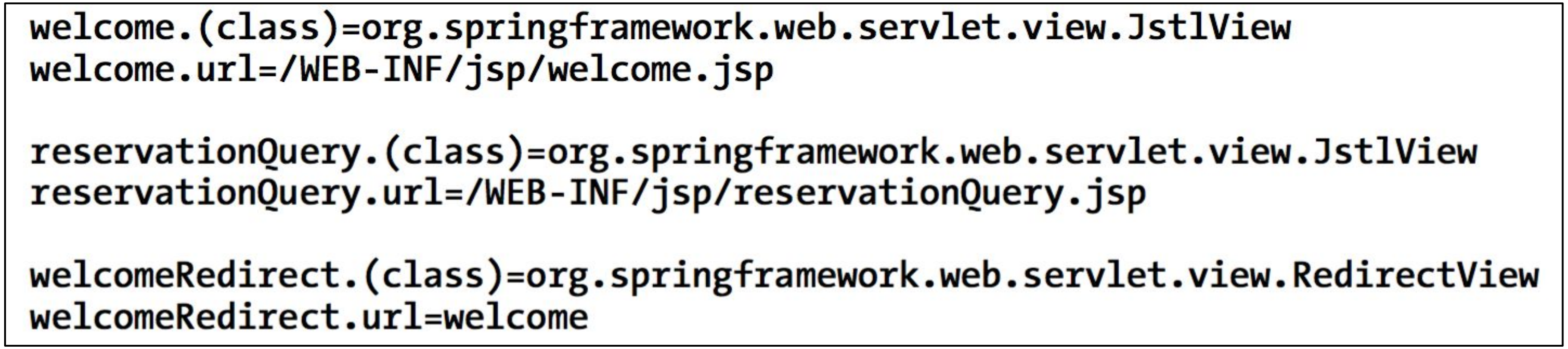
```
<bean class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="location">
        <value>/WEB-INF/court-views.xml</value>
    </property>
</bean>
```

```
 <bean id="welcome"
     class="org.springframework.web.servlet.view.JstlView">
     <property name="url" value="/WEB-INF/jsp/welcome.jsp" />
 </bean>

 <bean id="reservationQuery"
     class="org.springframework.web.servlet.view.JstlView">
     <property name="url" value="/WEB-INF/jsp/reservationQuery.jsp" />
 </bean>

 <bean id="welcomeRedirect"
     class="org.springframework.web.servlet.view.RedirectView">
     <property name="url" value="welcome" />
 </bean>
```

Resolving views from a resource bundle



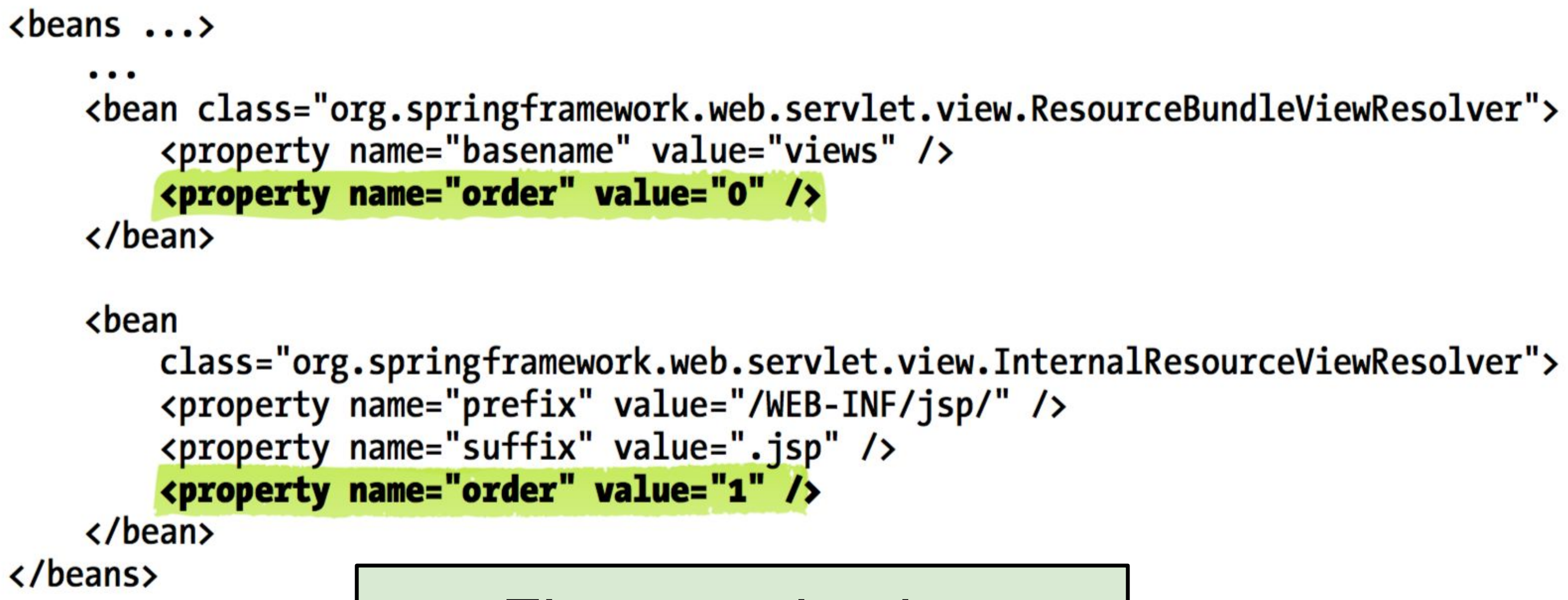
```
<bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="basename" value="views" />
</bean>
```



```
welcome.(class)=org.springframework.web.servlet.view.JstlView
welcome.url=/WEB-INF/jsp/welcome.jsp

reservationQuery.(class)=org.springframework.web.servlet.view.JstlView
reservationQuery.url=/WEB-INF/jsp/reservationQuery.jsp

welcomeRedirect.(class)=org.springframework.web.servlet.view.RedirectView
welcomeRedirect.url=welcome
```

views.properties

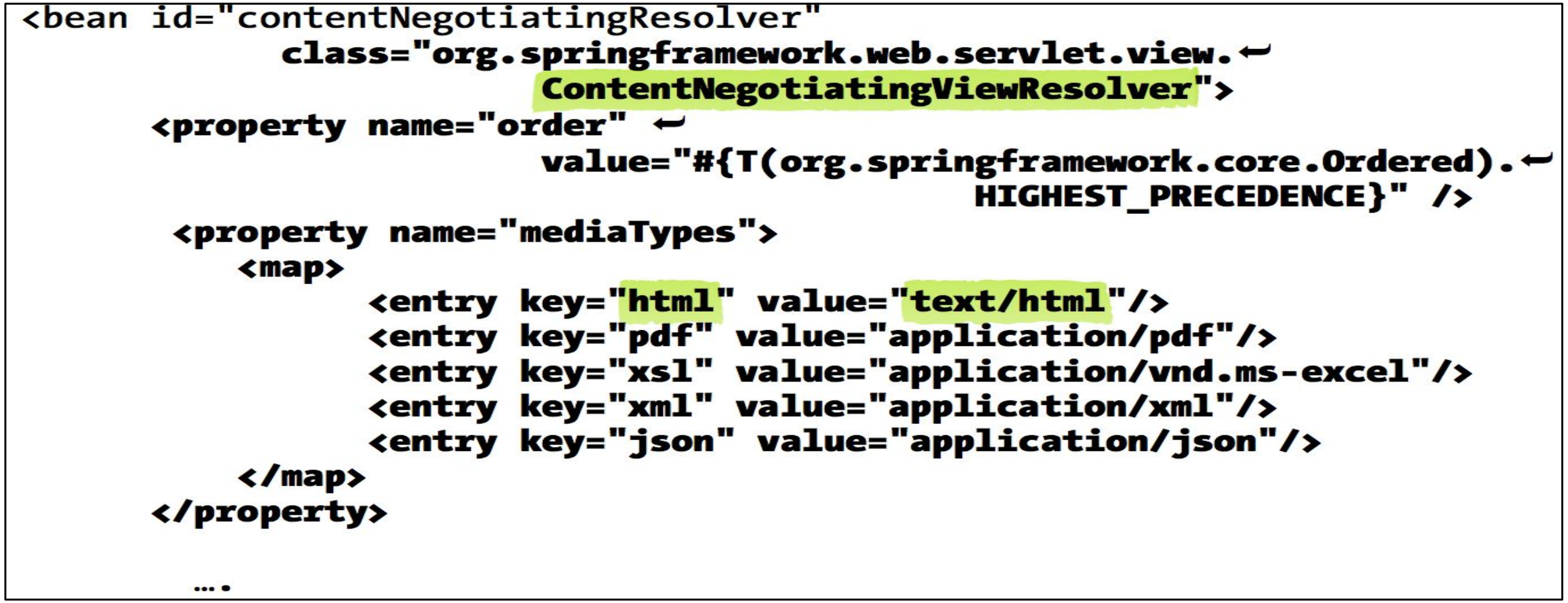
Can I use multiple view resolvers?



```
<beans ...>
    ...
    <bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
        <property name="basename" value="views" />
        <property name="order" value="0" />
    </bean>

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
        <property name="order" value="1" />
    </bean>
</beans>
```

First match wins

Views and Content Negotiation - *Can I get what I want*?

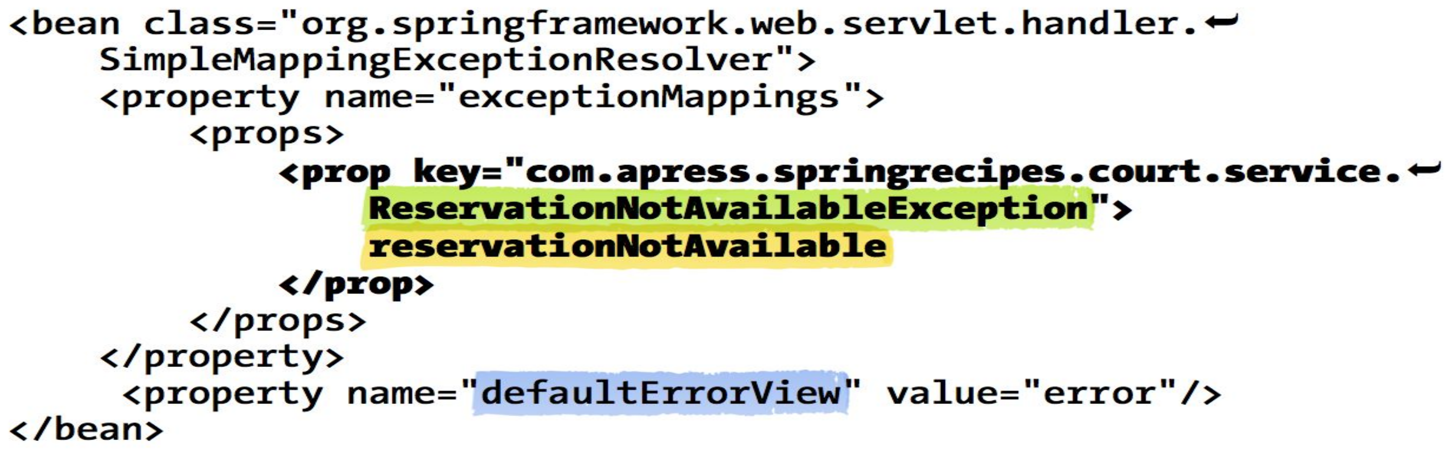


```
<bean id="contentNegotiatingResolver"
            class="org.springframework.web.servlet.view.↩
                        ContentNegotiatingViewResolver">
      <property name="order" ↩
                        value="#{T(org.springframework.core.Ordered).↩
                                    HIGHEST_PRECEDENCE}" />
       <property name="mediaTypes">
          <map>
                <entry key="html" value="text/html"/>
                <entry key="pdf" value="application/pdf"/>
                <entry key="xsl" value="application/vnd.ms-excel"/>
                <entry key="xml" value="application/xml"/>
                <entry key="json" value="application/json"/>
          </map>
      </property>

          ….
```

Content type resolution order

- Path extension against the mediaType map
    - e.g., .html mapes to text/html
- Path extension against FileTypeMap in Java Activation Framework.
- Use HTTP Accept header of the request

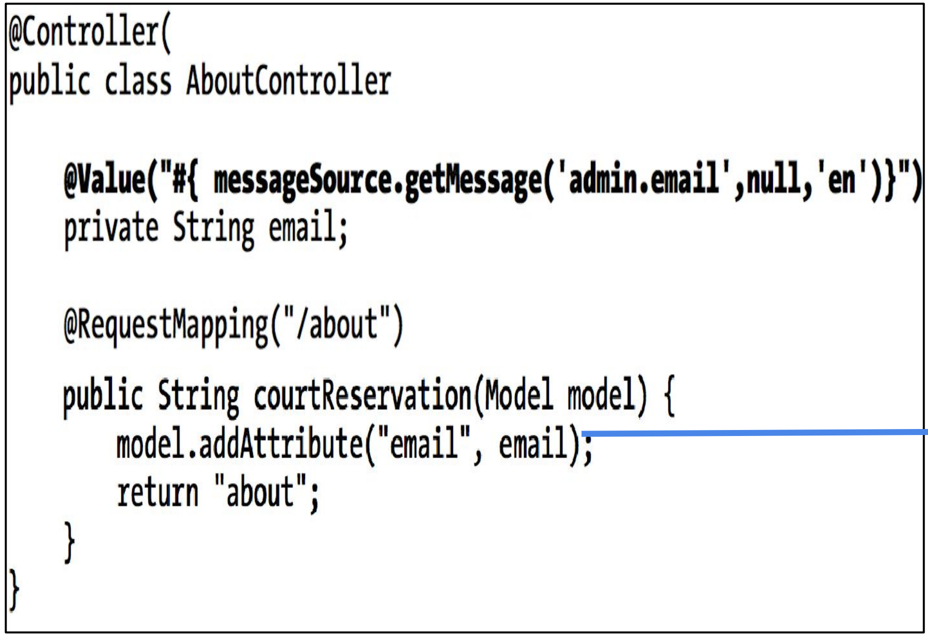
Error handling - *Map Exceptions to Views*



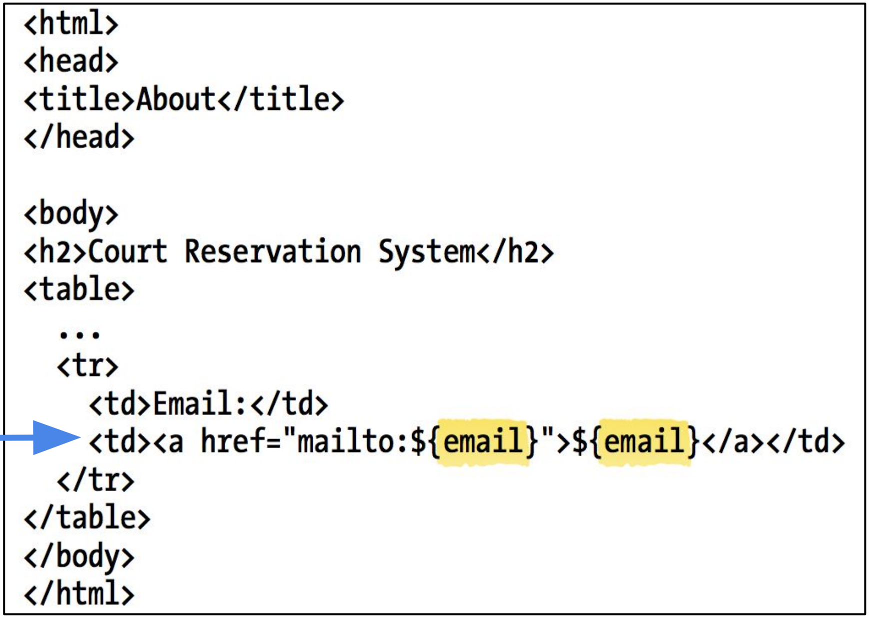
```
<bean class="org.springframework.web.servlet.handler.↩
    SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="com.apress.springrecipes.court.service.↩
                ReservationNotAvailableException">
                reservationNotAvailable
            </prop>
        </props>
    </property>
     <property name="defaultErrorView" value="error"/>
</bean>
```

Maps the exception to reservationNotAvaliable.jsp

Pass values through controller



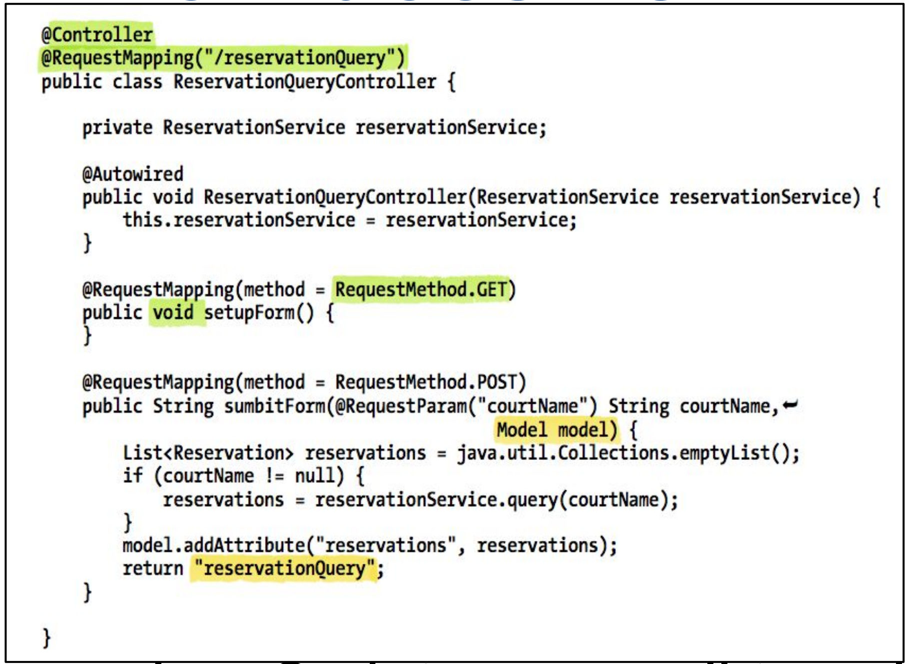
```
@Controller(
public class AboutController

    @Value("#{ messageSource.getMessage('admin.email',null,'en')}")
    private String email;

    @RequestMapping("/about")

    public String courtReservation(Model model) {
        model.addAttribute("email", email);
        return "about";
    }
}
```



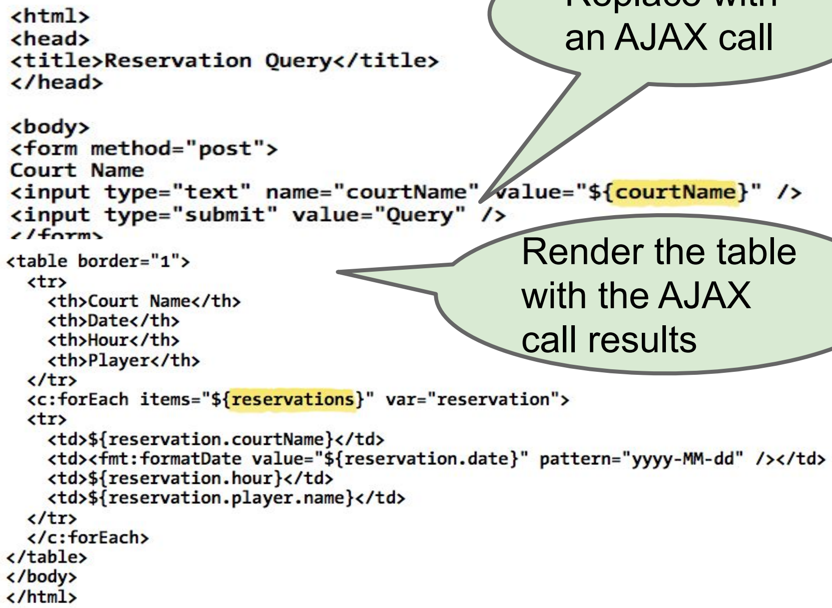
```
<html>
<head>
<title>About</title>
</head>

<body>
<h2>Court Reservation System</h2>
<table>
  ...
  <tr>
    <td>Email:</td>
    <td><a href="mailto:${email}">${email}</a></td>
  </tr>
</table>
</body>
</html>
```

Model is a map from attr names to values

How does AJAX work in Spring?



```java
@Controller
@RequestMapping("/reservationQuery")
public class ReservationQueryController {

    private ReservationService reservationService;

    @Autowired
    public void ReservationQueryController(ReservationService reservationService) {
        this.reservationService = reservationService;
    }

    @RequestMapping(method = RequestMethod.GET)
    public void setupForm() {
    }

    @RequestMapping(method = RequestMethod.POST)
    public String sumbitForm(@RequestParam("courtName") String courtName,↵
                                            Model model) {
        List<Reservation> reservations = java.util.Collections.emptyList();
        if (courtName != null) {
            reservations = reservationService.query(courtName);
        }
        model.addAttribute("reservations", reservations);
        return "reservationQuery";
    }

}
```



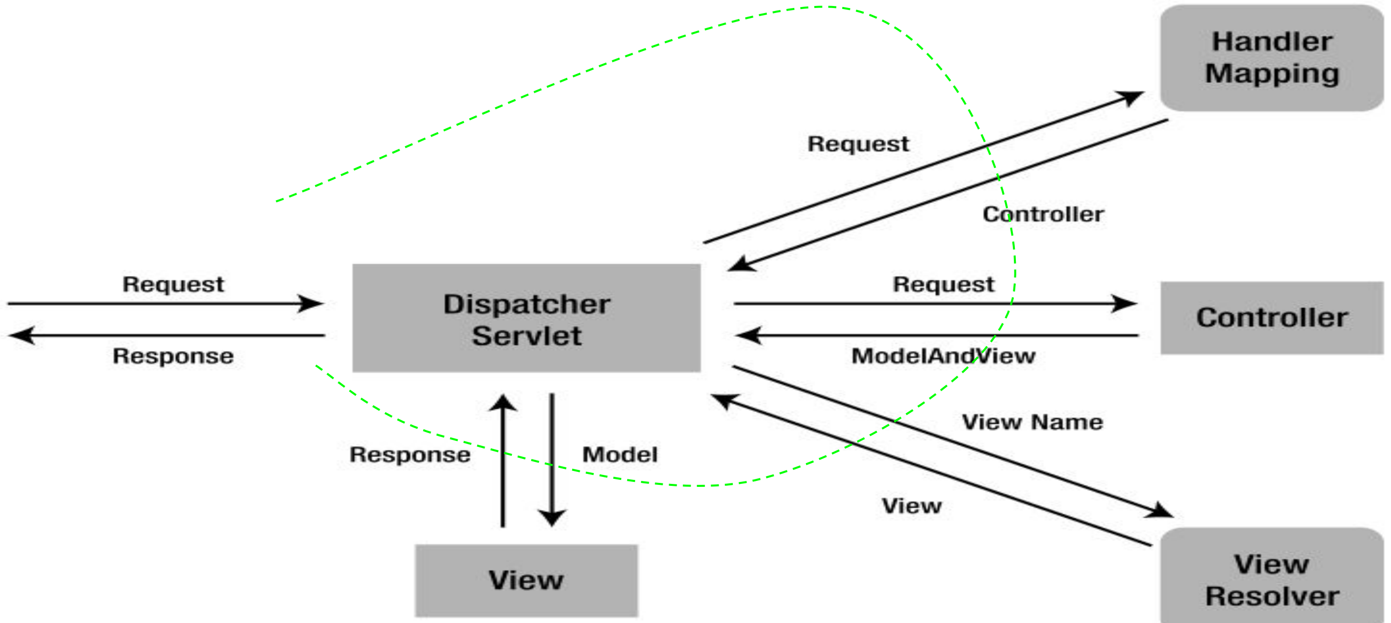
```html
<html>
<head>
<title>Reservation Query</title>
</head>

<body>
<form method="post">
Court Name
<input type="text" name="courtName" value="${courtName}" />
<input type="submit" value="Query" />
</form>
<table border="1">
  <tr>
    <th>Court Name</th>
    <th>Date</th>
    <th>Hour</th>
    <th>Player</th>
  </tr>
  <c:forEach items="${reservations}" var="reservation">
  <tr>
    <td>${reservation.courtName}</td>
    <td><fmt:formatDate value="${reservation.date}" pattern="yyyy-MM-dd" /></td>
    <td>${reservation.hour}</td>
    <td>${reservation.player.name}</td>
  </tr>
  </c:forEach>
</table>
</body>
</html>
```

Replace with an AJAX call

Render the table with the AJAX call results

JavaScript async call to only get domain data back and render the corresponding portion of the HTML

Spring MVC Summary


Handler Mapping
Request
Controller
Request
Response
Dispatcher Servlet
Request
Controller
ModelAndView
View Name
Response
Model
View
View
View Resolver