CS544 EA

Applications

Spring MVC: Request Mapping

Request Mapping

- @RequestMapping can be used to map an incoming HTTP request to a method
- The following shortcuts also exist:
  - @GetMapping
  - @PostMapping
  - @PutMapping
  - @DeleteMapping
  - @PatchMapping

Best to use these, because most controller methods should be mapped to only one HTTP method

RequestMapping by Path and Method

```java
@Controller
public class CarController {
  @Resource
  private CarDao carDao;

  @RequestMapping(value="/cars", method=RequestMethod.GET)
  public String getAll(Model model) {
    model.addAttribute("cars", carDao.getAll());
    return "carList";
  }
}
```

Exactly the same

```java
@Controller
public class CarController {
  @Resource
  private CarDao carDao;

  @GetMapping(value="/cars")
  public String getAll(Model model) {
    model.addAttribute("cars", carDao.getAll());
    return "carList";
  }
}
```

Multiple Methods per Controller

```
@Controller
public class CarController {
  @Resource
  private CarDao carDao;

  @GetMapping(value="/cars")
  public String getAll(Model model) {
    model.addAttribute("cars", carDao.getAll());
    return "carList";
  }

  @PostMapping(value="/cars")
  public String add(Car car) {
    carDao.add(car);
    return "redirect:/cars";
  }

  ...
}
```

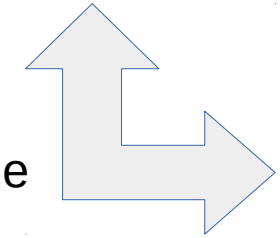
Class Level Path Mapping

```java
@Controller
public class CarController {

  @GetMapping(value="/cars/{id}")
  public String get(@PathVariable int id, Model model) {
    model.addAttribute("car", carDao.get(id));
    return "carDetail";
  }

  @PostMapping(value="/cars/{id}")
  public String update(Car car, @PathVariable int id) {
    carDao.update(id, car);
    return "redirect:/cars";
  }
}
```

Exactly the same



```java
@Controller
@RequestMapping(value="/cars")
public class CarController {

  @RequestMapping(value="/{id}", method=RequestMethod.GET)
  public String get(@PathVariable int id, Model model) {
    model.addAttribute("car", carDao.get(id));
    return "carDetail";
  }

  @RequestMapping(value="/{id}", method=RequestMethod.POST)
  public String update(Car car, @PathVariable int id) {
    carDao.update(id, car);
    return "redirect:/cars";
  }
}
```

Parameters and Headers

```
@Controller
public class CarController {
  @Resource
  private CarDao carDao;

  @GetMapping(value="/cars", params="myParam=myValue")
  public String getAllParam(Model model) {
    model.addAttribute("cars", carDao.getAll());
    return "carList";
  }

  @GetMapping(value="/cars", headers="myHeader=myValue")
  public String getAllHeader(Model model) {
    model.addAttribute("cars", carDao.getAll());
    return "carList";
  }

  ...

}
```

params="myParam" or
params="!myParam"
also possible

Only requests for "/cars?myParam=myvalue"
will be mapped here

Only Requests that have an http header:
myHeader: myValue
Will be mapped here

Useful for web services

Produces and Consumes

```
@RestController
public class WebService {
     @Resource
     private CarService carService;

     @GetMapping(value="/cars", produces="application/json")
     public List<Car> getAll() {
          return carService.getAll();
     }

     @PostMapping(value="/addCar", consumes="application/json")
     public void addCar(@RequestBody Car car) {
          carService.add(car);
     }
}
```

Mapping to non-Controllers

```
@Configuration
@EnableWebMvc
@ComponentScan("cs544")
public class WebConfig implements WebMvcConfigurer{
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/").setViewName("index");;
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**")
            .addResourceLocations("/resources/", "classpath:/static/", "files:/opt/files/")
            .setCachePeriod(31556926);
    }

    @Override
    public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }
}
```

View Controller
lets the request go directly
to a JSP page

For static resources such as:
CSS, JS, HTML

Lets us find resources
(static and dynamic)
through web.xml

XML

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

  <!-- Maps '/' requests to the 'home' view -->
  <mvc:view-controller path="/" view-name="index"/>

  <!-- Handles HTTP GET requests for /resources/** by efficiently serving
       up static resources in the ${webappRoot}/resources/ directory -->
  <mvc:resources mapping="/resources/**" location="/resources/" cache-period="31556926" />

  <!-- Lets us find resources (static and dynamic) through the web.xml -->
  <mvc:default-servlet-handler/>
</beans>
```

Same things now in XML