

A Long Spring

19 Years of Living with Your Past Mistakes

Arjen Poutsma

**With
Anecdotes
and
Dancing!**

Introduction

Who is this guy anyway?

- Thirty years of experience in Enterprise Software Development
- Worked on Spring from 2005–2024
- Development lead of Spring Web Services, Spring MVC, Spring Scala
- Conceived @Controller
- Work on Embabel



Anecdote

Academic Background

Not Your Typical Background

- AI & Philosophy
- Rod Johnson: PhD in musicology
- Spring Team members have various backgrounds
- Diverse perspectives matter

Web Controllers

Controllers with XML

```
public class AccountController extends
MultiActionController {

    public ModelAndView list(HttpServletRequest
request, HttpServletResponse response) {
        List accounts = ...
        ModelAndView mav = new
ModelAndView("accounts/list");
        mav.addObject("accounts", accounts);
        return mav;
    }

    public ModelAndView show(HttpServletRequest
request, HttpServletResponse response) {
        String id = request.getParameter("id");
        Account account = ...
        ModelAndView mav = new
ModelAndView("accounts/show");
        mav.addObject("account", account);
        return mav;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

    <bean id="handlerMapping" class="SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/account.do">accountController</prop>
            </props>
        </property>
    </bean>

    <bean id="accountController" class="AccountController">
        <property name="methodNameResolver">
            <bean class="ParameterMethodNameResolver">
                <property name="paramName" value="action"/>
                <property name="defaultMethodName" value="list"/>
            </bean>
        </property>
    </bean>

</beans>
```

Controllers with Annotations

```
@Controller
@RequestMapping("/accounts")
public class AccountController {

    @GetMapping
    public String list(Model model) {
        List<Account> accounts = ...
        model.addAttribute("accounts", accounts);
        return "account/list";
    }

    @GetMapping("/{id}")
    public String show(@PathVariable("id") Long id, Model model) {
        Account account = ...
        model.addAttribute("account", account);
        return "account/show";
    }
}
```

Controllers with Functions

```
public class AccountHandler {

    @Bean
    RouterFunction<ServerResponse> accountsRoutes() {
        return route()
            .path("/accounts", b → b
                .GET("", this::list)
                .GET("/{id}", this::show)
            )
            .build();
    }

    private ServerResponse list(ServerRequest req) {
        List<Account> accounts = ...
        return ok().render("account/list", accounts);
    }

    private ServerResponse show(ServerRequest req) {
        Long id = Long.valueOf(req.pathVariable("id"));
        Account account = ...
        return ok().render("account/show", account);
    }

}
```

Anecdote

Why does Spring MVC use Bubble Sort?

Why Does Spring Use Bubble Sort?

Content Negotiation

- HTTP request expresses preferred media type through the **Accept** header
- **@Controller** expresses producible media types through **@RequestParam::produces**
- Provided in any order
- Most **specific**, acceptable media type compatible with the method's produces condition.

Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, */*;q=0.8

@RequestMapping(produces="application/json")



Why Does Spring Use Bubble Sort?

Specificity

1. `text/*` is more specific than `*/*`
2. `text/plain` is more specific than `text/*`
3. `text/plain;format=flowed` is more specific than `text/plain`
4. Otherwise, the media types are considered incomparable



Why Does Spring Use Bubble Sort?

Incomparable Media Types

- `text/html` and `text/plain`
- Neither more specific than the other:
incomparable
- Partial order
- Comparator requires total order
- Bubble Sort



HTTP Clients

RestTemplate

```
String response =  
    template.postForEntity(  
        "https://example.com",  
        "Hello World",  
        String.class);
```

```
m delete (URI url) void  
m acceptHeaderRequestCallback (Class<T> responseType) RequestCallback  
m postForLocation (URI url, Object request) URI  
m postForLocation (String url, Object request, Object... uriVariables) URI  
m postForLocation (String url, Object request, Map<String, ? extends Object> uriVariables) URI  
m delete (String url, Object... uriVariables) void  
m delete (String url, Map<String, ? extends Object> uriVariables) void  
m doExecute (URI url, String uriTemplate, HttpMethod method, RequestCallback requestCallback, Class<T> responseType) T  
m exchange (RequestEntity<?> entity, Class<T> responseType) ResponseEntity<T>  
m exchange (RequestEntity<?> entity, ParameterizedType<?> responseType) ResponseEntity<T>  
m exchange (URI url, HttpMethod method, HttpEntity<?> requestEntity, Class<T> responseType) ResponseEntity<T>  
m exchange (URI url, HttpMethod method, HttpEntity<?> requestEntity, ParameterizedType<?> responseType) ResponseEntity<T>  
m exchange (String url, HttpMethod method, HttpEntity<?> requestEntity, Class<T> responseType) ResponseEntity<T>  
m exchange (String url, HttpMethod method, HttpEntity<?> requestEntity, ParameterizedType<?> responseType) ResponseEntity<T>  
m execute (URI url, HttpMethod method, RequestCallback requestCallback, Class<T> responseType) T  
m execute (String uriTemplate, HttpMethod method, RequestCallback requestCallback, Class<T> responseType) T  
m execute (String uriTemplate, HttpMethod method, RequestCallback requestCallback, ParameterizedType<?> responseType) T  
m getErrorHandler () ResponseErrorHandler  
m getForEntity (URI url, Class<T> responseType) ResponseEntity<T>  
m getForEntity (String url, Class<T> responseType, Object... uriVariables) ResponseEntity<T>  
m getForEntity (String url, Class<T> responseType, Map<String, ? extends Object> uriVariables) ResponseEntity<T>  
m getForObject (URI url, Class<T> responseType) T  
m getForObject (String url, Class<T> responseType, Object... uriVariables) T  
m getForObject (String url, Class<T> responseType, Map<String, ? extends Object> uriVariables) T
```

RestClient

```
String response = client.post(RequestBodyUriSpec
    .body(ParameterizedTypeReference<T> bodyType)
    .toString()
    .onStatus(ResponseErrorHandler errorHandler)
    .body().equals(Object obj)
    .onStatus(Predicate<HttpStatusCode> statusPredicate, ... ResponseSpec
    .body().toBodilessEntity()
    .body().toEntity(Class<T> bodyType)
    .put(RequestBodyUriSpec
```

Anecdote

Why So Many URI Methods?

Why So Many URI Methods?

```
uri(String uriTemplate, Object... uriVariables)  
uri(String uriTemplate, Map<String, ?> uriVariables)  
uri(URL uri)
```

- **Varargs:** `uri("http://example.com/${foo}", "bar");`
- **Map:** `uri("http://example.com/${foo}/${foo}", Map.of("foo", "bar"));`
- **URI:** `uri(new URI("http://example.com/bar"));`
- String variants encode, URI variant does not

```
restClient.get().uri("http://example.com/foo%20bar");
```

Anecdote

Five Years to Fix a Mistake

Five Years to Fix a Mistake

The Story of **HttpMethod**

- **HttpMethod** **enum** introduced in Spring 3
 - RESTful, client-side usage
- Expanded in Spring 5 WebFlux
 - Bug: WebDAV methods (not in enum) caused exception
 - String value workaround (**getMethod** and **getMethodValue**)
- Change Event Horizon
 - Too late for proper fix, stuck with workaround

Five Years to Fix a Mistake

- Spring 6 required recompilation (Java EE to Jakarta EE)
- Opportunity to fix **HttpMethod**
- Change **enum** to **class**
- Most behavior was maintained

Closing Thoughts

Open Source is about People

Not just code

- Empathy
 - Focus on the common case, but do not block advanced use cases
 - Show sympathy in bug reports
- Humility
 - Accept that you might be wrong
 - Admit mistakes openly
- Restraint
 - “When in doubt leave it out” — *J. Bloch*
 - Do not overpromise

<https://poutsma-principles.com>



Questions?

Ask me in the app