# Spring

# REST Services

# REST

*RE*presentational *S*tate *T*ransfer

Ambiguous acronym, simple concept

- PUT
  - Create or Replace
- POST
  - Create or Modify
- GET
  - Retrieve
- DELETE
  - Can you guess?

# Example Requests

# Example Requests

- GET /users

# Example Requests

- GET /users
- GET /users/1234

# Example Requests

- GET /users
- GET /users/1234
- PUT /users/1234

# Example Requests

- GET /users
- GET /users/1234
- PUT /users/1234
- POST /users

# Example Requests

- GET /users
- GET /users/1234
- PUT /users/1234
- POST /users
- POST /users/1234

# Example Requests

- GET /users
- GET /users/1234
- PUT /users/1234
- POST /users
- POST /users/1234
- DELETE /users/1234

# REST

# REST

- High-level guidelines

# REST

- High-level guidelines
- Check the HTTP spec for each Method

# REST

- High-level guidelines
  - Check the HTTP spec for each Method
  - Makes no statements about payload

# REST

- High-level guidelines
  - Check the HTTP spec for each Method
  - Makes no statements about payload
  - Choose your own adventure

# SOAP

*Simple Object Access Protocol*

"Simple" hah!

# SOAP

*Simple Object Access Protocol*

"Simple" hah!

- Evil

# SOAP

*Simple Object Access Protocol*

"Simple" hah!

- Evil
- Built for RPC

# SOAP

*Simple Object Access Protocol*

"Simple" hah!

- Evil
- Built for RPC
- XML

# SOAP

*Simple Object Access Protocol*

"Simple" hah!

- Evil
- Built for RPC
- XML
- WS-*

# SOAP

*Simple Object Access Protocol*

"Simple" hah!

- Evil
- Built for RPC
- XML
- WS-*
- Frameworks are House of Cards

# Implementing

# Implementing

- Implement REST in a plain old Servlet

# Implementing

- Implement REST in a plain old Servlet
- A lot of if-statements and RegExp

# Implementing

- Implement REST in a plain old Servlet
- A lot of if-statements and RegExp
- RegExp is so fun in Java, why wouldn't you?

# Implementing

- Implement REST in a plain old Servlet
- A lot of if-statements and RegExp
- RegExp is so fun in Java, why wouldn't you?
- Exactly.

# Spring

# Spring

- We Java-types are all about Frameworks

# Spring

- We Java-types are all about Frameworks
- The Mother-of-All-Frameworks is Spring

# Spring

- We Java-types are all about Frameworks
- The Mother-of-All-Frameworks is Spring
  - Kitchen Sink / Swiss Army Knife of Java

# Spring

- We Java-types are all about Frameworks
- The Mother-of-All-Frameworks is Spring
  - Kitchen Sink / Swiss Army Knife of Java
- It does REST, and does it well

# Spring REST

# Spring REST

- Two key components:

# Spring REST

- Two key components:
- spring-mvc

# Spring REST

- Two key components:
  - spring-mvc
  - Marshallers

# spring-mvc

# spring-mvc

- Spring MVC as of 2.5 gives us Annotations to:

# spring-mvc

- Spring MVC as of 2.5 gives us Annotations to:
  - Direct URI traffic to our controllers

# spring-mvc

- Spring MVC as of 2.5 gives us Annotations to:
  - Direct URI traffic to our controllers
  - URL parameters to method parameters

# spring-mvc

- Spring MVC as of 2.5 gives us Annotations to:
  - Direct URI traffic to our controllers
  - URL parameters to method parameters
  - Integrate with marshallers for payload/ response

# @RequestMapping

```
@RequestMapping(value = "/users", method = RequestMethod.GET)
public List<User> getAll() {
    ...
}
```

# @RequestMapping

```
@RequestMapping(value = "/users", method = RequestMethod.GET)
public List<User> getAll() {
    ...
}
```

- Used to:

# @RequestMapping

```
@RequestMapping(value = "/users", method = RequestMethod.GET)
public List<User> getAll() {
    ...
}
```

- Used to:
  - Declaratively map a URI to a Controller

# @PathVariable

```
@RequestMapping(value = "/users/{userId}",method = RequestMethod.GET)
public User get(@PathVariable Long userId) {
    ...
}
```

# @PathVariable

```
@RequestMapping(value = "/users/{userId}",method = RequestMethod.GET)
public User get(@PathVariable Long userId) {
    ...
}
```

- Used to:

# @PathVariable

```
@RequestMapping(value = "/users/{userId}",method = RequestMethod.GET)
public User get(@PathVariable Long userId) {
    ...
}
```

- Used to:
  - Map a portion of a URI to a method parameter

# @PathVariable

```
@RequestMapping(value = "/users/{userId}",method = RequestMethod.GET)
public User get(@PathVariable Long userId) {
    ...
}
```

- Used to:
  - Map a portion of a URI to a method parameter
  - http://server/users/1234

# @RequestParam

```
@RequestMapping(value = "/users",method = RequestMethod.GET)
public User get(@RequestParam Long userId) {
    ...
}
```

# @RequestParam

```
@RequestMapping(value = "/users",method = RequestMethod.GET)
public User get(@RequestParam Long userId) {
    ...
}
```

- Used to:

# @RequestParam

```
@RequestMapping(value = "/users",method = RequestMethod.GET)
public User get(@RequestParam Long userId) {
    ...
}
```

- Used to:
  - Map a request parameter to a method parameter

# @RequestParam

```
@RequestMapping(value = "/users",method = RequestMethod.GET)
public User get(@RequestParam Long userId) {
    ...
}
```

- Used to:
  - Map a request parameter to a method parameter
  - http://server/users?userId=1234

# @ResponseBody

```
@RequestMapping(value = "/users/{userId}")
public User get(@PathVariable Long userId) {
    ...
}
```

# @ResponseBody

```
@RequestMapping(value = "/users/{userId}")
public User get(@PathVariable Long userId) {
    ...
}
```

- We see our Controller method returns a "User"

# @ResponseBody

```
@RequestMapping(value = "/users/{userId}")
public User get(@PathVariable Long userId) {
    ...
}
```

- We see our Controller method returns a "User"
- How will that work over HTTP?

# @ResponseBody

```
@RequestMapping(value = "/users/{userId}")
public @ResponseBody User get(@PathVariable Long userId) {
    ...
}
```

- We see our Controller method returns a "User"
- How will that work over HTTP?
- @ResponseBody

# Marshalling

# Marshalling

- @ResponseBody signals to spring-mvc Marshalling needs to occur

# Marshalling

- @ResponseBody signals to spring-mvc Marshalling needs to occur
  - Configure this directly, or let Spring "find a marshaller"

# @RequestBody

```
@RequestMapping(value = "/users", method = RequestMethod.POST)
public User save(@RequestBody User user) {
    ...
}
```

# @RequestBody

```
@RequestMapping(value = "/users", method = RequestMethod.POST)
public User save(@RequestBody User user) {
    ...
}
```

- We see our Controller method accepts a “User”

# @RequestBody

```
@RequestMapping(value = "/users", method = RequestMethod.POST)
public User save(@RequestBody User user) {
    ...
}
```

- We see our Controller method accepts a “User”
- How will that work over HTTP?

# @RequestBody

```
@RequestMapping(value = "/users", method = RequestMethod.POST)
public User save(@RequestBody User user) {
    ...
}
```

- We see our Controller method accepts a “User”
- How will that work over HTTP?
- @ResponseBody

# Unmarshalling

# Unmarshalling

- This controller method accepts a “User”

# Unmarshalling

- This controller method accepts a “User”
- Use @RequestBody to alert Spring that the incoming payload needs unmarshalling

# Unmarshalling

- This controller method accepts a “User”
- Use @RequestBody to alert Spring that the incoming payload needs unmarshalling
  - Note: without @RequestBody Spring uses basic conventions to unmarshal

# It’s not just you...

# It's not just you...

- Others were probably thinking:

# It's not just you...

- Others were probably thinking:
  - POST & GET can be done via <form>
    but what about PUT & DELETE?

# It's not just you...

- Others were probably thinking:
  - POST & GET can be done via <form> but what about PUT & DELETE?
    - Use Javascript or...

# It's not just you...

- Others were probably thinking:
  - POST & GET can be done via <form> but what about PUT & DELETE?
    - Use Javascript or...
    - HiddenHttpMethodFilter with <form>

# Testing

# Testing

- Unit test your REST Controllers without a container

# Testing

- Unit test your REST Controllers without a container
- Use HttpUnit for Functional Testing

# Wrapping Up

# Wrapping Up

- Spring is very flexible for REST services

# Wrapping Up

- Spring is very flexible for REST services
- There are many more “tricks”, we’ve only seen the basics

# Wrapping Up

- Spring is very flexible for REST services
- There are many more “tricks”, we’ve only seen the basics
- JAXB is still horrible, try Castor instead

# Wrapping Up

- Spring is very flexible for REST services
- There are many more “tricks”, we’ve only seen the basics
- JAXB is still horrible, try Castor instead
  - See Spring OXM

# Example

# Example

- https://github.com/ctataryn/wjpg-spring-rest