Spring4 REST

REST Concepts

REST has quickly become the de-facto standard for building web services on the web because they're easy to build and easy to consume.

Why REST? *As per* Martin Fowler's phrasing, "the notion that the web is an existence proof of a massively scalable distributed system that works really well, and we can take ideas from that to build integrated systems more easily." REST embraces the precepts of the web itself, and embraces its architecture, benefits and all.

What benefits? Principally all those that come for free with HTTP as a platform itself.
Application security (encryption and authentication) are known quantities today for which there are known solutions.
Caching is built into the protocol. Service routing, through DNS, is a resilient and well-known system already ubiquitously support.

REST Concepts

Dr. Leonard Richardson put together a maturity model that interprets various levels of compliance with RESTful principles, and grades them.

It describes 4 levels, starting at **level 0**.

Level 0: the Swamp of POX - at this level, we're just using HTTP as a transport. You could call SOAP a **Level 0** technology. It uses HTTP, but as a transport. It's worth mentioning that you could also use SOAP on top of something like JMS with no HTTP at all. SOAP, thus, is *not* RESTful. It's only just HTTP-aware.

•**Level 1**: Resources - at this level, a service might use HTTP URIs to distinguish between nouns, or entities, in the system. For example, you might route requests to /customers, /users, etc. XML-RPC is an example of a **Level 1** technology: it uses HTTP, and it can use URIs to distinguish endpoints. Ultimately, though, XML-RPC is not RESTful: it's using HTTP as a transport for something else (remote procedure calls).

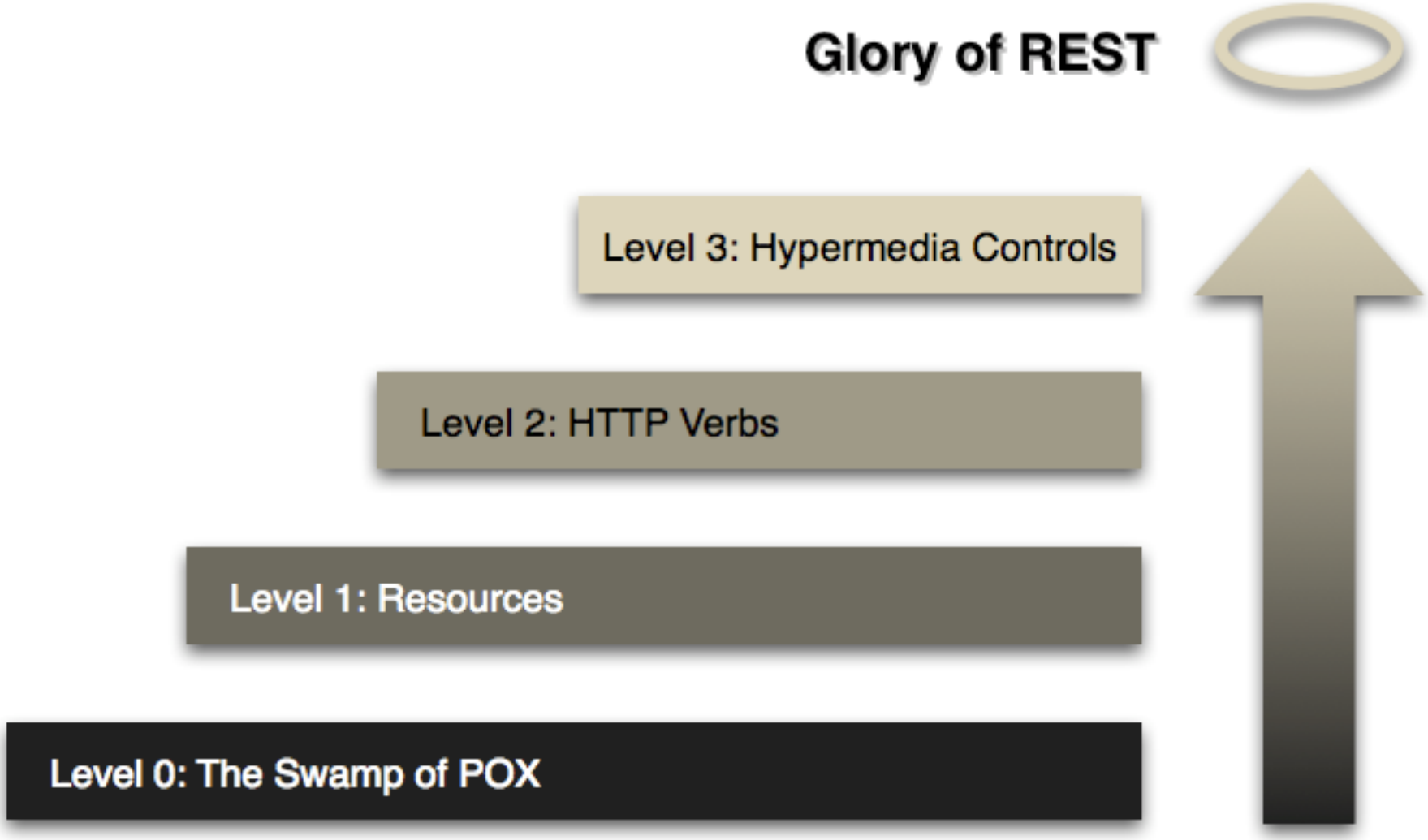
REST Concepts

•**Level 2**: HTTP Verbs - this is the level you want to be at. If you do **everything** wrong with Spring MVC, you'll probably still end up here. At this level, services take advantage of native HTTP qualities like headers, status codes, distinct URIs, and more. This is where we'll start our journey.

•**Level 3**: Hypermedia Controls - This final level is where we'll strive to be. Hypermedia, as practiced using the HATEOAS ("HATEOAS" is a acronym for the mouthful, "Hypermedia as the Engine of Application State") design pattern.

Hypermedia promotes service longevity by decoupling the consumer of a service from intimate knowledge of that service's surface area and topology. It **describes** REST services. The service can answer questions about what to call, and when. We'll look at this in depth later.

REST Concepts



Building REST services with Spring: pom.xml

```
<dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>4.3.5.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>4.3.5.RELEASE</version>
        </dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.3.5.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.3.5.RELEASE</version>
</dependency>



    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.5</version>
      <scope>test</scope>
    </dependency>
```

The best compatible versions
1)Spring 4.2.4 works with fasterxml Jackson 2.7.2 or 2.8.4
2)Spring 4.3.5 works with fasterxml Jackson 2.7.0

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>4.3.5.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.3.5.RELEASE</version>
</dependency>
 <!--
https://mvnrepository.com/artifact/com.fasterxml
.jackson.core/jackson-databind -->
<dependency>

<groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.0.2</version>
</dependency>
```

Building REST services with Spring: Greeting POJO

```
@Component
public class Greeting {

    private  Long id;
    private  String content;

    public Greeting() {

    }

    public Greeting(Long id, String content) {
        this.id = id;
        this.content = content;
    }

    public Long getId() {
        return id;
    }

    public String getContent() {
        return content;
    }
}
```

Building REST services with Spring : GreetingController

```
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name",
defaultValue="World") String name) {
        return new Greeting(counter.incrementAndGet(),
                            String.format(template, name));
    }
}
```

Note:
@RestController is equivalent to combination of @Controller and @ResponseBody

Building REST services with Spring

A key difference between a traditional MVC controller and the RESTful web service controller above is the way that the HTTP response body is created.

Rather than relying on a view technology to perform server-side rendering of the greeting data to HTML, this RESTful web service controller simply populates and returns a Greeting object. The object data will be written directly to the HTTP response as JSON.

This code uses Spring 4's new @RestController annotation, which marks the class as a controller where every method returns a domain object instead of a view. It's shorthand for @Controller and @ResponseBody rolled together.

The Greeting object must be converted to JSON. Thanks to Spring's HTTP message converter support, you don't need to do this conversion manually.

Because Jackson 2 is on the classpath, Spring's MappingJackson2HttpMessageConverter is automatically chosen to convert the Greeting instance to JSON.

New Spring REST Annotations

@PostMapping, @GetMapping, @PutMapping and @DeleteMapping

Ex.

```
@GetMapping("/user/{userid}")
```

is same as

```
@RequestMapping( value="/user/{userid}",method=RequestMethod.GET)
```

Note: The above annotations are available since Spring 4.3

New Spring REST Annotations

public class **ResponseEntity<T>** extends HttpEntity<T>

Extension of HttpEntity that adds a HttpStatus status code. Used in RestTemplate as well @Controller methods.

```
@RequestMapping("/handle")
 public ResponseEntity<String> handle() {
   URI location = ...;
   HttpHeaders responseHeaders = new HttpHeaders();
   responseHeaders.setLocation(location);
   responseHeaders.set("MyResponseHeader", "MyValue");
   return new ResponseEntity<String>("Hello World", responseHeaders, HttpStatus.CREATED);
 }
```

UserWebServices Spring REST Application

http://localhost:8080/UserWebServices/user-rest/user/{userid}

Ex.
http://localhost:8080/UserWebServices/user-rest/user/101

```
@SuppressWarnings({ "rawtypes", "unchecked" })
//@GetMapping("/user/{userid}")
@RequestMapping( value="/users/{userid}",method=RequestMethod.GET)
public ResponseEntity findUserById(@PathVariable(value="userid")
String id )  {
try {
User user=service.getUserById(Integer.parseInt(id));
if(user==null) {
throw new Exception( "No user found for ID " + id);
}
return new ResponseEntity(user, HttpStatus.OK);
} catch (Exception e) {
return new ResponseEntity(e.getMessage(), HttpStatus.NOT_FOUND);
}
}
```

UserWebServices Spring REST Application

http://localhost:8080/UserWebServices/user-rest/users

```
@GetMapping("/users")
public List<User> getAllUsers() throws UserException{
return service.getAllUsers();
}
```

Note: Except GET, for remaining HTTP methods, use postman or HttpClient tool

```
@SuppressWarnings({ "rawtypes", "unchecked" })
@PostMapping(value = "/users")
public ResponseEntity userRegistration( @Valid @RequestBody User user)  {
User userNew;
try {
userNew = service.userRegistration(user);
return new ResponseEntity(userNew, HttpStatus.OK);
} catch (Exception e) {
System.out.println(e.getMessage());
return new ResponseEntity("Invalid Data. Unable to add user",
HttpStatus.NOT_FOUND);
}
}
```

UserWebServices.zip

UserWebServices Spring REST Application

http://localhost:8080/UserWebServices/user-rest/users/101

```
@SuppressWarnings({ "rawtypes", "unchecked" })
@DeleteMapping("/users/{userid}")
public ResponseEntity deleteUser(@PathVariable(value="userid") Integer id){

try {
service.deleteUser(id);
return new ResponseEntity("Deleted user " + id, HttpStatus.FOUND);
} catch (Exception e) {
return new ResponseEntity(e.getMessage(), HttpStatus.NOT_FOUND);
}
}

//http://localhost:8080/UserWebServices/user-rest/users
@SuppressWarnings({ "rawtypes", "unchecked" })
@PutMapping("/users")
public ResponseEntity updateUser(@RequestBody User user) {
try {
service.updateUser(user);
return new ResponseEntity("User record updated " , HttpStatus.FOUND);
} catch (Exception e) {
return new ResponseEntity(e.getMessage(), HttpStatus.NOT_FOUND);
}
}
```

UserWebServices.zip

Thank You!