Оглавление

[SOAP 3](#_Toc14261228)

[@WebService 3](#_Toc14261229)

[@WebMethod 3](#_Toc14261230)

[@WebResult 3](#_Toc14261231)

[@OneWay 3](#_Toc14261232)

[@SoapBinding 3](#_Toc14261233)

[@SOAPMessageHandler 3](#_Toc14261234)

[JAXB 3](#_Toc14261235)

[@XmlAccessorType(XmlAccessType.FIELD) 3](#_Toc14261236)

[@XmlRootElement 3](#_Toc14261237)

[@XmlType 3](#_Toc14261238)

[@XmlTransient 3](#_Toc14261239)

[@XmlAttribute 3](#_Toc14261240)

[@XmlElement 3](#_Toc14261241)

[@XmlElementWrapper 3](#_Toc14261242)

[@XmlEnum 3](#_Toc14261243)

[@XmlElements 3](#_Toc14261244)

[REST 3](#_Toc14261245)

[@Path 3](#_Toc14261246)

[@Stateless 3](#_Toc14261247)

[@Context 3](#_Toc14261248)

[@GET 3](#_Toc14261249)

[--@Produces 3](#_Toc14261250)

[@POST 4](#_Toc14261251)

[--@Consumes 4](#_Toc14261252)

[--@Produces 4](#_Toc14261253)

[@DELETE 4](#_Toc14261254)

[--@Path("{id}") 4](#_Toc14261255)

[@QueryParam 4](#_Toc14261256)

[@GET 4](#_Toc14261257)

[@MatrixParam 4](#_Toc14261258)

[@CookieParam 5](#_Toc14261259)

[@HeaderParam 5](#_Toc14261260)

[@FormParam 5](#_Toc14261261)

[@DefaultValue 5](#_Toc14261262)

[@Produces(‘?’) 5](#_Toc14261263)

[@Path(value="/contacts") 5](#_Toc14261264)

[SPRING REST 6](#_Toc14261265)

[@RestController 6](#_Toc14261266)

[@RequestMapping(value = "/simple1") 6](#_Toc14261267)

[@GetMapping 6](#_Toc14261268)

[@ComponentScan - возможность автоматического сканирования при использовании java-конфигурации 6](#_Toc14261269)

[SPRING DATA 6](#_Toc14261270)

[@Entity 6](#_Toc14261271)

[@Table 6](#_Toc14261272)

[@Column 6](#_Toc14261273)

[@Id 6](#_Toc14261274)

[@GeneratedValue 6](#_Toc14261275)

[@Version 6](#_Toc14261276)

[@OrderBy 6](#_Toc14261277)

[@Transient 6](#_Toc14261278)

[@Temporal 7](#_Toc14261279)

[@Embeddable 7](#_Toc14261280)

[SPRING 7](#_Toc14261281)

[@Autowired 7](#_Toc14261282)

[@Inject 7](#_Toc14261283)

[--@Named 7](#_Toc14261284)

[@Resource 7](#_Toc14261285)

[@Value 7](#_Toc14261286)

[@Component 7](#_Toc14261287)

[@Controller 7](#_Toc14261288)

[@Repository 7](#_Toc14261289)

[@Service 7](#_Toc14261290)

[@Configuration 7](#_Toc14261291)

[@Bean 7](#_Toc14261292)

# SOAP

## @WebService

## @WebMethod

## @WebResult

## @OneWay

## @SoapBinding

## @SOAPMessageHandler

# JAXB

## @XmlAccessorType(XmlAccessType.FIELD)

@XmlRootElement определяет корневой элемент для содержимого XML.

@XmlType(propOrder = {"name", "age", "language"}) - используется для упорядочения элементов в XML.

@XmlTransient *-* помечается то, что не будет не записано в XML.

@XmlAttribute(name = "expiry\_date", required = true) - помечается то, что будет использовано в качестве атрибута.

@XmlElement(name = "lang") — создаст элемент с именем «qwerty».

@XmlElementWrapper — обертка вокруг коллекций для читабельности сгенерированного XML Объекты другого класса помечаются аннотацией @XmlSeeAlso***.***

@XmlEnum и ***@XmlEnumValue*** — для [Enum](https://javadevblog.com/perechisleniya-v-java-preimushhestva-ispolzovaniya-enum.html) и значений

@XmlElements — создание контейнеров для нескольких ***@XmlElement***

# REST

@Path("/book") путь в браузере

@Stateless - добавления возможностей EJB

## @Context

## @GET

--@Produces("text/plain")

## @POST

--@Consumes- задает типы содержимого MIME, принимаемые ресурсом Consumes сверяется с параметром Content-Type для того, чтобы определить, способен ли метод обработать содержимое полученного запроса.

--@Produces - типы MIME, возвращаемые ресурсом.

**@GET**

**@Path(value="/{lastName}")**

**@Produces(value="text/xml")**

**public ContactInfo getByLastName(@PathParam(value="lastName") String lastName) {**

**...**

**}**

## @DELETE

### --@Path("{id}")

@QueryParam - извлекает значение параметра запроса, содержащегося в URI.

## @GET

public Customers getCustomersByZipCode(@QueryParam("zip") Long zip, @QueryParam("city") String city) {

// URI : /customer?zip=75012&city=Paris }

@MatrixParam - @MatrixParam действует подобно @QueryParam, однако извлекает значение параметра матрицы URI (в качестве разделительного знака используется ;).

@GET

@Path("search")

public Customers getCustomersByName(@MatrixParam("firstname") String firstname, @MatrixParam("surname") String surname) {

// URI : /customer/search;firstname=Antonio;surname=Goncalves }

@CookieParam - извлекает значение cookie

@GET

public String extractSessionID(@CookieParam("sessionID") String sessionID) {

@HeaderParam - значение поля заголовка.

*@GET*

*@Path*("/get")

public Response getPerson(

*@HeaderParam*("person-agent") String personAgent)

@FormParam– связывает HTML параметры в Java методе.

*@POST*

*@Path*("/add")

public Response addPerson(*@FormParam*("name") String name,*@FormParam*("age") int age)

## @DefaultValue

@Produces(‘?’) скачать файл с помощью JAX-RS? Задать аннотацию @Produces(‘?’) над сервисным методом с указанием возвращаемого в ответе типа. Вместо ‘?’ используйте text/plain, image/png и т.д.

*@GET*

*@Path*("/get")

*@Produces*("image/png")

public Response getFile() {

File file = new File(FILE\_PATH);

ResponseBuilder response = Response.ok((Object) file);

response.header("Content-Disposition",

"attachment; filename=image\_from\_server.png");

return response.build();

}

}

@Path(value="/contacts")- Корневыми ресурсами являются Java-классы, отмеченные аннотацией @Path. Эта аннотация включает атрибут value, задающий путь к ресурсу. Его значением могут быть строка символов, переменные, а также переменные в сочетании с регулярным выражением.

# SPRING REST

@RestController - Аннотация @ComponentScan говорит Spring'у рекурсивно искать в пакете hello и его потомках классы, помеченные прямо или косвенно Spring аннотацией @Component. Эта директива гарантирует, что Spring найдет и зарегистрирует GreetingController, потому что он отмечен @RestController, который, в свою очередь, является своего рода @Component аннотацией. @RestController = @Component

@RequestMapping(value = "/simple1") сообщаем, что данный контроллер будет обрабатывать запрос, URI которого "/simple1"

### @GetMapping - это интерфейс, который содержит путь конечной точки (**uri**) для выполнения метода Get. Этот интерфейс **GetMapping** использует интерфейс **RequestMapping**, который может иметь метод «**path**, **value**, **params**, **headers**» для выполнения метода Get в более ранних версиях Spring.

@PostMapping - это интерфейс, который содержит путь к конечной точке (uri) для выполнения метода **Post**.

@PutMapping - это интерфейс, который содержит путь конечной точки (uri) для выполнения метода **Put** для обновления записей.

#### @PathVariable

## @ComponentScan - возможность автоматического сканирования при использовании java-конфигурации

# SPRING DATA

## @Configuration

@Entity — Указывает, что данный бин (класс) является сущностью.

@Table — указывает на имя таблицы, которая будет отображаться в этой сущности.

@Column — указывает на имя колонки, которая отображается в свойство сущности.

@Id — id колонки

@GeneratedValue — указывает, что данное свойство будет создаваться согласно указанной стратегии.

@Version — управление версией в записи сущности. При изменении записи увеличится на 1.

@OrderBy — указание сортировки.

@Transient — указывает, что свойство не нужно записывать. Значения под этой аннотацией не записываются в базу данных (так же не участвуют в сериализации). static и final переменные экземпляра всегда transient.

@Temporal — применяется к полям или свойствам с типом java.util.Date и java.util.Calendar. Например, если в БД время сохраняется как sql.Date, то чтобы использовать дату из java.util.Date указываем эту аннотацию.

@Embeddable и @Embedded («встроенный») — Определяет класс, экземпляры которого хранятся как неотъемлемая часть исходного объекта. Каждый из @Embedded экземпляров сопоставляется с таблицей базы данных сущности.

http://javastudy.ru/spring-data-jpa/annotation-persistence/

# SPRING

@SpringBootApplication - Эта аннотация состоит из аннотации, которые используется в Spring:

* **@Component** - сообщает компилятору, что следующий класс является компонентом, который должен быть включен при компиляции всего приложения.
* **@ComponentScan** - Этот сканирует, какие пакеты мы собираемся использовать в следующем классе Java.
* **@EnableAutoConfiguration** - включает механизм автоконфигурации Spring Boot для импорта важных модулей для запуска Spring Boot.

Это аннотации, используемые для запуска **Spring Boot Application** для запуска на сервере.

## @Autowired

## @Inject

--@Named – подсказывающая аннотация

## @Resource

## @Value

@Component(тут можно передать имя id) – универсальная аннотация, указывающая, что класс является компонентом Spring;

@Controller – указывает, что класс определяет контроллер Spring MVC;

@Repository – указывает, что класс определяет репозиторий данных;

@Service – указывает, что класс определяет службу;

любая пользовательская аннотация, определенная с помощью аннотации @Component.

@Configuration – класс в котором будут описаны бины.

@Bean – Метод просто создает и возвращает экземпляр класса

# Java аннотации

@PostConstruct

и

@PreDestroy. С помощью этих аннотаций можно помечать методы бина, автоматически вызываемые после создания бина и перед тем как бин будет удален из контейнера соответственно.

# SPRING SOAP

@PayloadRoot перед методом определяет, при получении какого запроса будет вызван данный метод @PayloadRoot(localPart = "ServiceRequest", namespace = namespaceUri)

@Endpoint - регистрирует класс Spring WS как потенциальный кандидат для обработки входящих SOAP сообщений

## @PayloadRoot используется Spring WS для выбора метода обработчика на основе **namespace** и **localPart** сообщения

## [@RewuestPayload](http://docs.spring.io/spring-ws/sites/2.0/apidocs/org/springframework/ws/server/endpoint/annotation/RequestPayload.html) указывает на то, что входящее сообщение будет сопоставлено параметру request метода

## @ResponsePayload создает соответствующее значение возвращаемому значению полезной части ответа.

## DefaultWsdl11Definition - предоставляет стандартный WSDL 1.1 с использованием [XsdSchema](http://docs.spring.io/spring-ws/sites/2.0/apidocs/org/springframework/xml/xsd/XsdSchema.html)

## MessageDispatcherServlet – MessageDispatcherServlet

ннотация @HandlerChain (цепочка обработчиков) используется для указания SOAP веб-сервису списка классов-обработчиков. В общем случае эти классы используются для изменения входящих и исходящих сообщений SOAP веб-сервиса. Если вы знакомы с сервлетами или перехватчиками, то хендлеры являются аналогом Servlet Filters или CDI\EJB Interceptors.