SpringBoot笔记

# 第一章 基础类

## 一 Date类相关问题

1、java.text.DateFormat:是日期/时间格式化子类的抽象类

1.1 作用：格式化（也就是日期 —> 文本）、解析（文本 —> 日期）

1.2 成员方法:

1. String format(Date date) 按照指定的模式,把Date日期,格式化为符合模式的字符串。
2. Date parse(String source) 把符合模式的字符串,解析为Date日期。

1.3 DateFormat类是一个抽象类,无法直接创建对象使用,可以使用DateFormat类的子类：java.text.SimpleDateFormat extends DateFormat。

(1) 构造方法：SimpleDateFormat(String pattern)：用给定的模式和默认语言环境的日期格式符号构造 SimpleDateFormat。

参数：String pattern:传递指定的模式

模式：区分大小写的

y 年

M 月

d 日

H 时

m 分

s 秒

写对应的模式,会把模式替换为对应的日期和时间

"yyyy-MM-dd HH:mm:ss"

注意:

模式中的字母不能更改,连接模式的符号可以改变

"yyyy年MM月dd日 HH时mm分ss秒"

**【测试代码】日期格式化以及日期的比较：**

public static void main(String[] args) {

SimpleDateFormat dateFormat= new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

Calendar calendar = Calendar.*getInstance*();

calendar.setTime(new Date());

calendar.add(Calendar.*DATE*,-15);

Date date = calendar.getTime();

System.*out*.println("Format-currentDate: " + dateFormat.format(date));

System.*out*.println("Date-currentDate: " + date);

System.*out*.println("Date-currentDate.getTime: " + date.getTime());

String time1 = "2022-10-25 18:55:58";

try {

Date parseTime1 = dateFormat.parse(time1);

System.*out*.println("time1: " + parseTime1);

System.*out*.println("time1.getTime: " + parseTime1.getTime());

System.*out*.println("date在time1前 : " + (date.getTime() < parseTime1.getTime()));

} catch (ParseException e) {

e.printStackTrace();

}

}

## 二 注解

### 1、@Controller

@Controller标识的类，该类代表控制器类(控制层/表现层)。这里控制层里面的每个方法，都可以去调用@Service标识的类（业务逻辑层）。

@Service标识的类中的方法可以继续调用@Resposity标识的接口实现类（Dao层/持久层）

MVC其实我们可以理解为是VCM：JSP/HTML页面数据–>控制器–>数据逻辑处理并持久化。

用@Controller定义一个控制器类，

用@RequestMapping给出外界访问方法的路径，或者说触发路径 ，触发条件。

用@ResponseBody标记Controller类中的方法。把return的结果变成JSON对象返回。（如果没有这个注解，这个方法只能返回要跳转的路径即跳转的html/JSP页面。有这个注解，可以不跳转页面，只返回JSON数据）。

### 2、@Produces

指定http响应的MIME类型，默认是\*/\*，表示任意的MIME类型。该注解的值是数组类型，支持多个MIME类型，MediaType常用的MIME类型如下:

application/xml

application/json

application/x-www-form-urlencoded

multipart/form-data

application/octet-stream

text/plain

text/xml

text/html

用于定义方法的响应实体的数据类型，可以定义一个或多个，同时可以为每种类型定义质量因素（qualityfactor）。质量因素是取值范围从0到1的小数值。如果不定义质量因素，那么该类型的质量因素默认为1。

### 3、@Consumes

指定http请求的MIME类型，默认是\*/\*，表示任意的MIME类型。该注解的值是数组类型，支持多个MIME类型。MediaType常用的MIME类型见2。

### 4、@Path

@Path注解可以标记在类名之上，也可以标记在方法名上。该注解接收一个value参数，表示定义资源的地址。另外，资源地址相同，但是HTTP方法不同的两个方法是完全两个不同的REST接口，HTTP方法和资源地址相结合在一起才可以完成对一个资源的定位。

我们可以发现其属于 javax.ws.rs 包，javax.ws.rs 是 jax-rs 规范中定义的包名，jax-rs 全称是：java API for RESTful Services，这是一套规范。

简单点说就是这套规范是为构建restful服务的 【restful可以理解为web接口】，我们知道Spring自带了对restful的支持，但是呢，Spring并没有遵循这个协议，言外之意就是不具有移植特性。尽管Spring并不遵循这个协议，但还是Spring好用。

再回到jax-rs，目前能够实现jax-rs标准的框架有很多，比如：

Apache CXF，开源的Web服务框架。

Jersey， 由Sun提供的JAX-RS的参考实现。Jersey 是Spring Boot @RestController.的替代。

RESTEasy，JBoss的实现。

Restlet，由Jerome Louvel和Dave Pawson开发，是最早的REST框架，先于JAX-RS出现。

Apache Wink，一个Apache软件基金会孵化器中的项目，其服务模块实现JAX-RS规范

说这么多，其实是想get一点，就是在当下都用Spring的环境下，还是有许多可以实现web服务的

### 疑问：

1、@GZIP

## 三 pom文件

### 1、<packaging>pom</packaging>

在父级项目中的pom.xml文件使用的packaging配置一定为pom。父级的pom文件只作项目的子模块的整合，在maven install时不会生成jar/war压缩包。

## 四 其他

### 1、IDEA .iml文件及.idea文件夹

#### 1.1 .iml文件

idea 对module 配置信息之意， infomation of module。每个模块都有一个iml文件。IDEA中的.iml文件是项目标识文件，缺少了这个文件，IDEA就无法识别项目。跟Eclipse的.project文件性质是一样的。并且这些文件不同的设备上的内容也会有差异，所以我们在管理项目的时候，.project和.iml文件都需要忽略掉。

iml文件是IntelliJ IDEA自己创建的模块文件，用于java应用开发，存储一些模块相关的信息，比如一个Java组件，插件组件，Maven组件等。存储一些模块路径信息，依赖信息以及别的一些设置。

#### 1.2 .idea文件

存放项目的配置信息，包括数据源，类库，项目字符编码，历史记录，版本控制信息等。

（1）Artifacts（人工制品，手工制品，手工艺品）是maven中的一个概念，表示某个module要如何打包，例如war exploded、war、jar、ear等等这种打包形式；

（2）inspectionProfiles：检查配置文件

（3）jsLinters：js语法检查配置文件

（4）libraries：类库

（5）.gitignore：git提交忽略文件

（6）misc.xml：各种杂项

（7）modules.xml：模块

（8）vcs.xml：版本控制系统

（9）workspace.xml：工作空间。保存着一下项目开发过程中的过程性信息，可以不要。

### 2、REST架构风格详解

#### 2.1 URI和URL的区别和联系

2.1.1 URI（统一资源标识符）

URI，指能够唯一的标记一个网络资源的符号。URI是一种抽象宽泛的定义。即，不管用什么方法表示，只要能唯一标记某个资源，它就叫URI。

例如：要标识一个人，可以通过DNA序列，或者电话号码，也可以通过身份证，还可以通过户籍。这些都可以作为URI。

可见，标记资源的方式多种多样，可以使用具体的地址来标记资源，也可以使用其他方式标记资源。

2.1.2 URL（统一资源定位符）

URL，是通过网络路径地址标记资源的符号。可以理解为：URL就是资源的地址。

上面例子中，户籍就是一种URL，通过它可以精确的找到这个人。（如果不在家，可以跳转定位继续寻找）

URL只是URI的实现方法之一。当然也可以使用其他方法实现URI，只是URL现在比较流行而已。

可见，URL是URI的子集。

2.1.3 URL的组成

URL包括（ 协议+//+域名+端口+路径+文件+?+参数+#+锚定 ）

比如下面的一个URL：

http://www.aspxfans.com:8080/news/day01/index.asp?boardID=5&pwd=24618&page=1#name

协议部分 http:

域名部分 www.aspxfans.com

端口部分 :8080

目录部分 /news/day01/

文件部分 index.asp

参数部分 boardID=5&pwd=24618&page=1

锚定部分 name

#### 2.2 REST架构风格简介

REST是 Representational State Transfer (在表示层上的状态传输)的缩写。REST是一种WEB应用的架构风格。

RESTful API (具有REST风格的API): 用URL定位资源，用HTTP动词（GET、HEAD、POST、PUT、PATCH、DELETE）描述操作，用响应状态码表示操作结果。

但是REST远远不仅是指API的风格 ，它是一种网络应用的架构风格。而且REST的原则不仅仅适用于HTTP协议。

我们浏览一个网站，说到底就是与这个网站中的资源进行互动（获取、提交、更新、删除）。前端的工作，就是为用户从服务端获取资源、展示资源、请求服务端改变资源。

RESTful API有助于客户端和服务端的功能分离，服务器完全扮演着一个“资源服务商”的角色。各种不同的客户端都可以通过一致的API与这个“资源服务商”交流，从而与资源进行互动。

#### 2.3资源

在REST架构中，“资源”扮演者主要角色。具有以下特点：

资源是任何可以操作 （获取、提交、更新、删除）的数据，比如一个文档（document）、一张图片……

资源的集合也是一种资源，比如 blogs 表示博客（资源）的集合。

进行资源操作的时候， 用URI来指定被操作的资源 。如果一个URI不仅能标识一个网络上的资源，还能够定位这个资源，那么这个URI也叫URL。

资源是一个抽象的概念，资源无法被传输，只能传输资源的表示 （representation）。一个资源可以有多种表示，比如，一个资源可以用HTML、XML、JSON来表示。总的来说，“表示”是“资源”的存储和传输形式，“资源”是“表示”的内容（抽象概念）。不管用什么形式来表示，始终描述的是这个资源。

表示（representation）包括数据（data，表示资源本身）和元数据（metadata，用于描述这个representation）。严格来说，“json字符串”并不是完整的representation， 整个HTTP响应才是representation 。HTTP body中的是数据，HTTP header中的是元数据（尤其是 Content-Type 这种字段）。

#### 2.4用URL定位资源

在RESTful架构风格中，URL用来指定一个资源。资源就是服务器上可操作的实体（可以理解为数据）。比如说URL： /API/Users 表示的是该网站的所有用户，这是一种资源，可以与之互动（获取、提交、更新、删除）。既然URL表示一个资源，自然就不应该包含动词，它应该由名词组成。

#### 2.5操作资源

对这个资源(URL)使用不同的HTTP方法，就代表对这个资源的不同操作：

GET（SELECT）：从服务器获取资源（一个资源或资源集合）。

POST（CREATE）：在服务器新建一个资源（也可以用于更新资源）。

PUT（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源）。

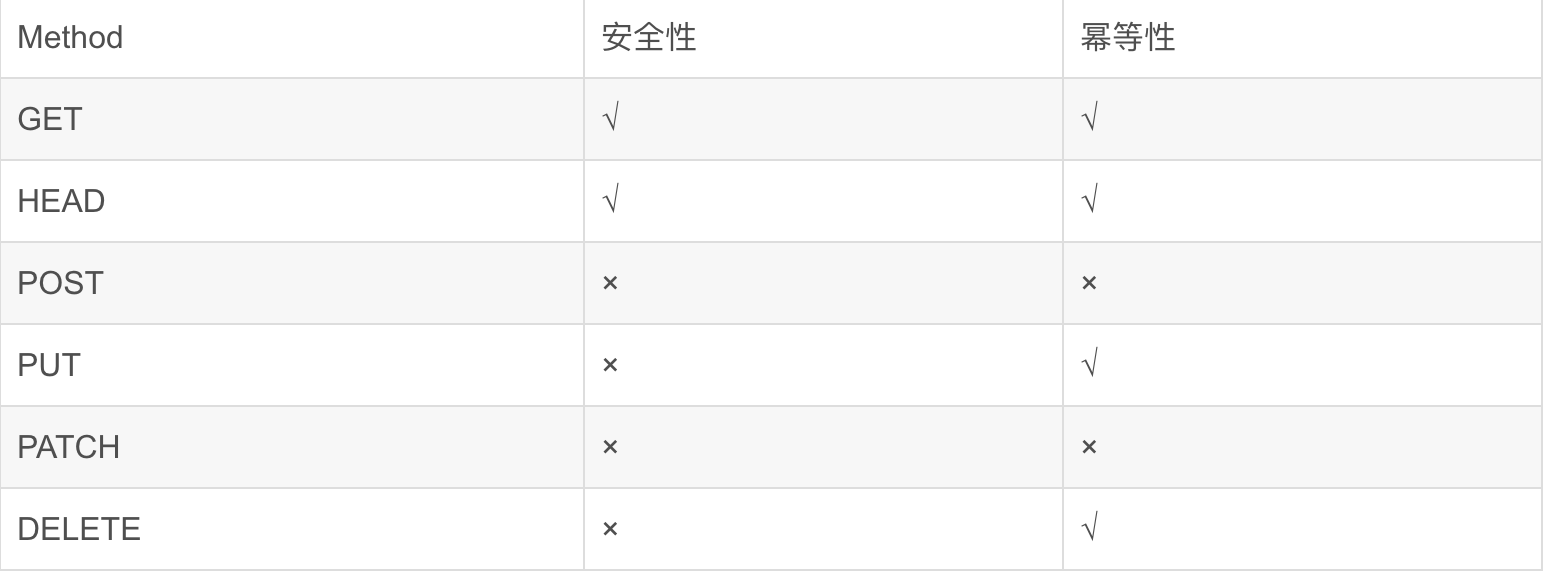
PATCH（UPDATE）：在服务器更新资源（客户端提供改变的部分）。

DELETE（DELETE）：从服务器删除资源。

HEAD：获取资源的元数据。

OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的。

GET、HEAD、PUT、DELETE方法是幂等方法 (对于同一个内容的请求，发出n次的效果与发出1次的效果相同)。GET、HEAD方法是 安全方法 (不会造成服务器上资源的改变)。PATCH不一定是幂等的。PATCH的实现方式有可能是”提供一个用来替换的数据”，也有可能是”提供一个更新数据的方法”(比如 data++ )。如果是后者，那么PATCH不是幂等的。



#### 2.6 通过HTTP状态码表示操作的结果

200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。

201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功。

202 Accepted - [\*]：表示一个请求已经进入后台排队（异步任务）

204 NO CONTENT - [DELETE]：用户删除数据成功。

400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。

401 Unauthorized - [\*]：表示用户没有权限（令牌、用户名、密码错误）。

403 Forbidden - [\*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。

404 NOT FOUND - [\*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。

406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。

410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。

422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。

500 INTERNAL SERVER ERROR - [\*]：服务器发生错误，用户将无法判断发出的请求是否成功。

#### 2.7 如何设计RESTful API

将一个“动作”理解为“操作一个资源”。这里的“操作”是指HTTP的方法。

例如：对于转账动作，就可以理解为“新建一个转账事务”（转账事务是资源），因此API就可以设置成这样: POST /transactions，请求体为：to=632&amount=500 。这样的设计不但简洁明了，而且我们可以将这个URL用于别的用途，通过 GET /transactions 来获取该用户的所有转账事务。还可以将 GET /transactions/456828 定义为“获取某一次转账记录”。

使用RESTful风格可以克服传统架构风格的那4个缺陷：

1.设计API工作量减少，因为功能需求一旦出来，需要操作的资源、操作的方式立刻就能分析出来，因此资源URL和API的使用方式(GET, POST…)都很容易得到。

2.没有了操作之间的依赖。资源之间虽然可能有关联，但是小得多。

3.对资源的操作也就那么几种(获取、新建、修改、删除)，API的一致性、自我描述性很强，不需要过多解释。

4.对于GET请求，我们都可以考虑使用缓存，因为在RESTful的架构中，GET请求代表获取数据，必须是安全、幂等的。

#### 2.8服务器无状态

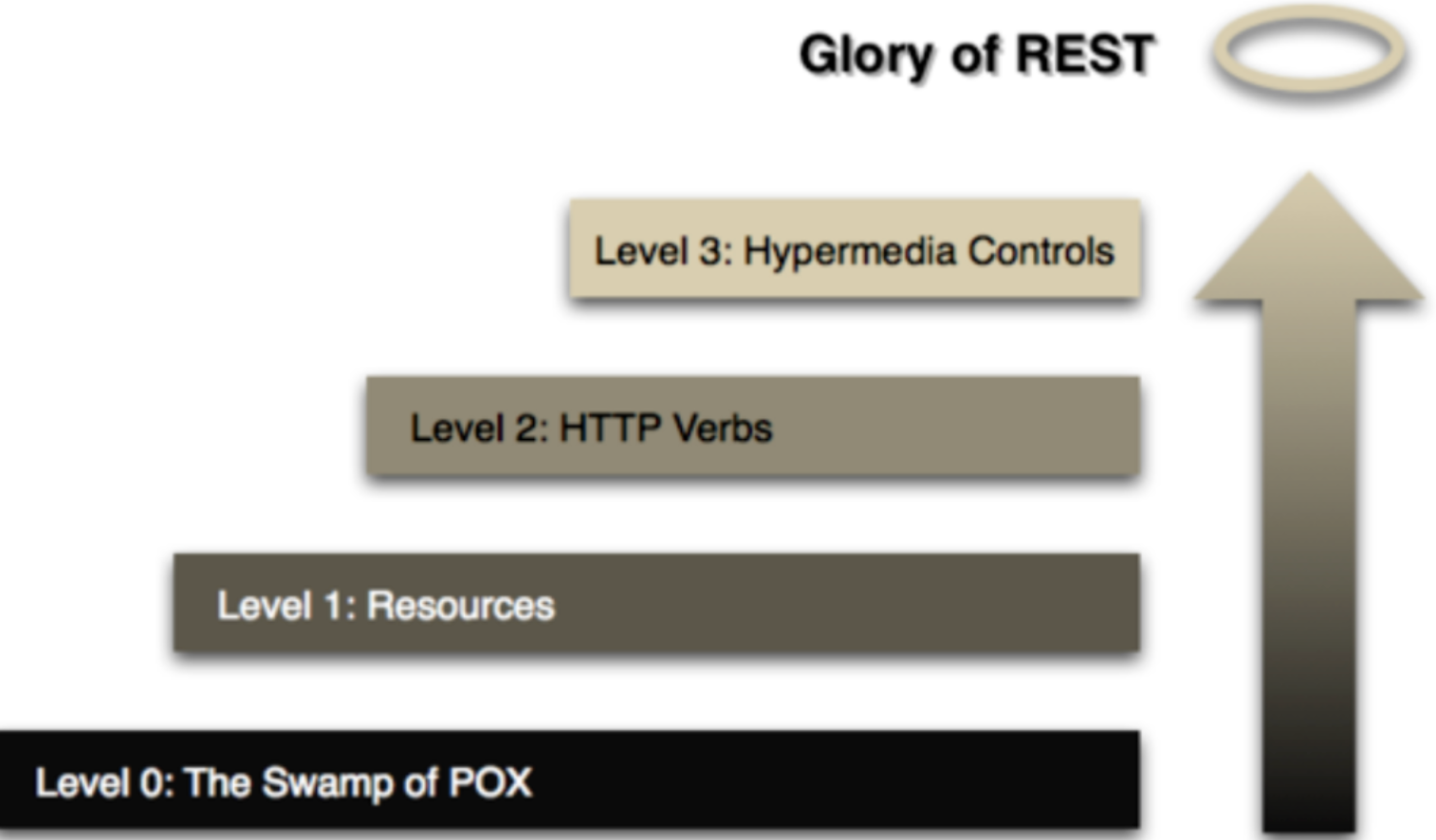
根据 REST的架构限制 ，RESTful的服务器必须是无状态的，这意味着来自客户的每一个请求必须包含服务器处理该请求所需的所有信息， 服务器不能利用任何已经存储的“上下文(context，在这里表示用户的状态)”来处理新到来的请求，会话状态只能由客户端来保存，并且在请求时一并提供。

这里注意两点：

1. 服务器不能存储“上下文”不代表连数据库都不能有，“上下文”指那些在服务器内存中的、非持久化的数据。

2.无状态不代表不能有会话(sessions)，无状态仅仅指 服务器无状态 ，会话的状态由客户端提供。

#### 2.9 HATEOAS



实际上REST还有一个更高的层次：HATEOAS(Hypermedia As The Engine Of Application State)。对于客户端的资源请求，服务器不仅要返回所请求的资源，而且要返回客户端所处的状态和可转移的状态。（ 客户端有状态）。

这像10086的“电话自动语音应答服务”：你想要查询你的手机流量，只需要会拨打“10086”，对方会提示你按下哪些按键就能进入哪些状态。进入下一个状态以后，又会有语音提示你接下来能够按哪些按键……最终，你能进入到你想要的那个状态（流量查询服务）。你需要记住的仅仅是“10086”这个号码而已！

#### 2.10 “资源”和“状态”的关系

在HATEOAS中，服务端将客户端所处的状态和可以达到的状态传输给客户端。这就很好的解释了Representational State Transfer中的State Transfer(状态传输)是什么意思了。

其实在REST架构风格中，“传输状态”和“传输资源表示”是同一个意思。客户端所处的状态，是由它接收到的资源表示来决定的。比如，客户端接收到 /user/csr/blogs 资源，那么客户端的状态就变成 /user/csr/blogs (显示csr的文章列表)。

按照客户端从前往后的顺序梳理一遍：

客户端请求根资源

=> 服务器 返回根资源的表示 ，以及相邻资源的Hypermedia

=> 客户端 进入“根资源”状态 （比如说，展示首页）

=> 客户端显示所有相邻状态的Hypermedia供用户选择（比如，在首页有一个导航栏，里面有几个链接）

=> 用户选择了某个Hypermedia（比如，点击了“查看文章列表”的链接）

=> 客户端请求“文章列表”资源

=> 服务器返回“文章列表”资源的表示，以及相邻资源的Hypermedia

=> 客户端进入“文章列表”状态

=> 客户端显示所有相邻状态的Hypermedia供用户选择（比如，在文章列表里，显示所有文章的链接）

……

不难发现，客户端接收到一个新的资源表示，就会跳转到新的状态，这个过程称为 状态传输（服务器给客户端传输新状态）。因此 状态传输 是通过传输资源表示 来完成的。

#### 2.11 总结

至此，我们应该能够体会到REST已经不仅仅是一种API风格了，它是一种软件架构风格(REST本身不是一种架构)。REST风格的软件架构具有很强的演化、拓展能力：

1.一致的URL和HTTP动词使用：确保系统能够接纳多样而又标准的客户端，保证客户端的演化能力。

2.无状态：保证了系统的横向拓展能力、服务端的演化能力。

3.HATEOAS：保证了应用本身的演化能力(功能增加、改变)。