*LOGGER*.isInfoEnabled()

1).直接使用logger.info("User " + userId + " is using app " + appId)来输出log，也能够达到log级别为INFO或在INFO以下时才输出：("User " + userId + " is using app " + appId)，因为logger.info方法内部有判断输出级别的代码。但是在进入logger.info函数之前，("User " + userId + " is using app " + appId) 这个表达式已经通过运算拼接成了一个字符串；而如果事先使用 if (logger.isInfoEnabled())进行判断，那么当log级别在INFO以上时，就能省去上述的字符串操作，在高并发和复杂log信息拼接的情况下，使用这种标准的方法输出log能够省去不小的系统开销。另外，如果构造log信息的过程需要大量字符串操作，建议使用StringBuilder来完成字符串拼接。

2).ERROR及其以上级别的log信息是一定会被输出的，所以只有logger.isDebugEnabled和logger.isInfoEnabled方法，而没有logger.isErrorEnabled方法。

//返回用户的IP地址  
public final String getClientIpAddress(HttpServletRequest request) {  
 for (String header : *HEADERS\_TO\_TRY*) {  
 String ip = request.getHeader(header);  
 if (ip != null && ip.length() != 0 && !"unknown".equalsIgnoreCase(ip)) {  
 return ip;  
 }  
 }  
  
 return request.getRemoteAddr();  
}

可以返回用户的真实IP地址，跨过代理服务器

public String getToken(String personId) {

String token = "";  
 //withAudience()用来存放需要存放的信息  
 token = JWT.*create*().withAudience(personId).withIssuedAt(start).withExpiresAt(end)  
 .sign(Algorithm.*HMAC256*(personId));  
  
 return token;  
}

Token的生成方法，token用来进行服务器端和用户端的交互，比如登录等。

@Value(“#{}”):获取{}中的值，属性或者方法

@Value(“#{1}”)

Private int age; 相当于age = 1;

@Value(“${}”):获取properties文件中的东西，比如properties文件中写了web.view.prefix =/WEB-INF/views/， @Value("${web.view.prefix}")

private String prefix; 相当于prefix = /WEB-INF/views/

String.valueOf(): 将（）类型转换成字符串类型

YearMonth: 是不可变的日期时间对象，表示年、月的组合。其实现了Comparable接口。

//将LocalDate转化成YearMonth  
private static YearMonth toYearMonth(Date date) {  
 LocalDate localDate = *toLocalDate*(date);  
 return YearMonth.*of*(localDate.getYear(), localDate.getMonth());  
}

DateTimeFormatter ：用于格式化显示，线程安全

private static Logger *logger* = LoggerFactory.*getLogger*(JsonUtils.class);

打印这个类的日志信息，JsonUtils.日志信息

String... values

可以传入多个String类型的值

JsonNodeFactory.instance:实例化一个Json工厂

JsonNodeFactory factory = new JsonNodeFactory.instance;

JsonNodeFactory.instance.nullNode:实例化一个空的Json工厂

*objectMapper*.readTree() ：将字符串类型返回成Json类型

ObjectMapper mapper = new ObjectMapper();  
String message = "{\"code\":\"EXCEPTION\",\"msgInfo\":\"交易支付异常\"} ";  
JsonNode jsonNode = null;  
try {  
 jsonNode = mapper.readTree(message);  
} catch (IOException e) {  
 e.printStackTrace();  
}  
JsonNode code = jsonNode.get("msgInfo");  
System.*out*.println(code);

*objectMapper*.readValue()：将一个Json转换成实体类

@SuppressWarnings（"unchecked"）：可以标注在类、字段、方法、参数、构造方法以及局部变量上。作用是：告诉编译器忽略指定的警告，不用在编译完成后出现警告信息。告诉编译器忽略 unchecked 警告信息，如使用List，ArrayList等未进行参数化产生的警告信息。除了unchecked还有很多，而且还有单类型，多类型和all。

@Component ：放到容器中，交给spring管理；如果这类中用到了容器里的东西，比如@Value，@Autowired等注解，使用到了spring容器里的东西，这个就必须加上@Component注解，否则spring找不到这个类，会报错。

BigDecimal:更加精确的处理小数。

@JsonFormat(pattern = “YYYY-MM-DD HH:mm:ss”, timezone =“GTM+8”):从数据库读出来的时间格式是英文格式，用此注解从数据库读到实体类的时候，date类型的时间会变成中文。

ResponseEntity可以定义返回的HttpStatus（状态码）和HttpHeaders（消息头：请求头和响应头）

@JsonIgnore

使用方法：一般标记在属性或者方法上，返回的json数据即不包含该属性。作用：在json序列化时将java 中的一些属性忽略掉，序列化和反序列化都受影响。

@ApiModel

Spring中注解，用在属性上

@NotNull://CharSequence, Collection, Map 和 Array 对象不能是 null, 但可以是空集（size = 0）。

@NotEmpty://CharSequence, Collection, Map 和 Array 对象不能是 null 并且相关对象的 size 大于 0。

@NotBlank://String 不能是 null 且去除两端空白字符后的长度（trimmed length）大于 0。

@Mapper 用在接口上，编译时会自动生成实现类。