SpringBoot#DAO#时间#处理与升级方案

【时间#处理与升级方案】

时间：2023年5月26日14:57:52

🌵现存方案

> *三者统一为GMT+8*

@JsonFormat

- *timezone*

JDBC#URL

- *serverTimezone*
- *connectionTimeZone*

MySQL#time-zone

- *@@global.time_zone*

> *时间格式化模式，使用Date*

@JsonFormat

- *日期：yyyy-MM-dd*
- *日期时间：yyyy-MM-dd HH:mm:ss.SSS*

🌵优化方案一（不考虑时区）

> *时区配置无强制要求*

@JsonFormat

- *timezone*

JDBC#URL

- *serverTimezone*
- *connectionTimeZone*

MySQL#time-zone

- *@@global.time_zone*

> *时间格式化模式，使用LocalDateTime、LocalDate、LocalTime*
>
> - *新的时间API，在使用需要严格匹配@JsonFormat时，格式化pattern字符串需要严格匹配。不能在LocalDateTime上仅使用yyyy-MM-dd。*

@JsonFormat

- *日期：yyyy-MM-dd*
- *时间：HH:mm:ss.SSS*
- *日期时间：yyyy-MM-dd HH:mm:ss.SSS*

🌵优化方案二（考虑多时区兼容）

> *强制：时区配置统一为UTC*

@JsonFormat

- *timezone*

JDBC#URL

- *serverTimezone*
- *connectionTimeZone*

MySQL#time-zone

- *@@global.time_zone*

> *时间格式化模式，只使用ZonedDateTime*
>
> - ☀️*新的时间API，对于前端传入的所有时间数据（日期、时间、日期时间），都需要传入完整的时间字符串（yyyy-MM-dd HH:mm:ss.SSS）*

> - ☀️*具体响应给前端的在任何时候都是完整的时间字符串，前端需要在业务逻辑中按需截取使用/显示。*

@JsonFormat

- *统一：yyyy-MM-dd HH:mm:ss.SSS*

↴提供时区转换器↴

Java | 复制代码

```java
import org.apache.commons.lang3.StringUtils;

import javax.servlet.http.HttpServletRequest;
import java.lang.reflect.Field;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.util.Arrays;
import java.util.List;

/**
 * 多时区转换器
 *
 * @Description <pre>
 * 多时区转换器
 * 1.提供世界级时区统一转换
 *   -> 请将三者统一为UTC时区：{@code @JsonFormat}、jdbcUrl#serverTimezone/connectionTimeZone、MySQL#time_zone
 *
 * 💡示例：
 * {@code
 *     @PostMapping("/demo02")
 *     public String demo02(HttpServletRequest request, @RequestBody User user) {
 *         TimeZoneConverter.tzConverterDown(request, user, User.class);
 *
 *         Example example = new Example(User.class);
 *         Example.Criteria criteria = example.createCriteria();
 *         criteria.andLessThanOrEqualTo(User.Fields.id, 400);
 *         List<User> users = userMapper.selectByExample(example);
 *
 *         TimeZoneConverter.tzConverterUp(request, users, User.class);
 *
 *         return JSONObject.toJSONString(users, JSONWriter.Feature.PrettyFormat);
 *     }
 * }
 * </pre>
 * @Author Trivis
 * @Date 2023/5/26 8:37
 * @Version 1.0
 */
public class TimeZoneConverter {

    static String TZ_HEADER = "tz";
    static String TZ_DEFAULT = "GMT+8";
```

```
43
44
45      static String TZ_PREFIX_LIMIT = "GMT";
        static List<String> TZ_POSTFIX_LIMIT = Arrays.asList("-12", "-11", "-
10",
46              "-9", "-8", "-7", "-6", "-5", "-4", "-3", "-2", "-1",
47              "-0", "+0",
48              "+1", "+2", "+3", "+4", "+5", "+6", "+7", "+8", "+9", "+10",
"+11", "+12", "+13", "+14");
49
50      /**
51       * <pre>
52       * 时区填充检测(仅支持大写GMT前缀)
53       *   -> 从Headers中获取tz属性
54       *   -> 在时区的标准表示法中，GMT（格林威治标准时间）后面的数字表示时区偏移量，以
分钟为单位。
55       *      -> 通常情况下，时区偏移量的范围是从GMT-12到GMT+14之间，这是因为地球上
最西边的地区与最东边的地区之间的时差最大为26小时。
56       * </pre>
57       *
58       * @return 时区字符串（GMT-12~GMT+14）
59       */
60      private static String fetchTZ(HttpServletRequest request) {
61          String userLocalTimeZone = request.getHeader(TZ_HEADER);
62          if (StringUtils.isEmpty(userLocalTimeZone)) return TZ_DEFAULT;
63          if ((userLocalTimeZone.length() == 6 || userLocalTimeZone.length(
) == 5)
64                  && userLocalTimeZone.startsWith(TZ_PREFIX_LIMIT)) {
65              if (TZ_POSTFIX_LIMIT.contains(userLocalTimeZone.substring(use
rLocalTimeZone.indexOf("GMT") + 3))) {
66                  return userLocalTimeZone;
67              }
68          }
69          return TZ_DEFAULT;
70      }
71
72      public static <T> void tzConverterDown(HttpServletRequest request, T
instance, Class<T> clazz) {
73          String userLocalTimeZone = fetchTZ(request);
74          for (Field field : clazz.getDeclaredFields()) {
75              if (field.getType() == ZonedDateTime.class) {
76                  field.setAccessible(true);
77                  try {
78                      ZonedDateTime x = (ZonedDateTime) (field.get(instance
));
79                      if (x != null) {
80                          field.set(instance, x.withZoneSameLocal(ZoneId.of
(userLocalTimeZone)));
81
```

```
                    }
                } catch (IllegalAccessException ignored) {
                }
            }
        }
    }

    public static <T> void tzConverterUp(HttpServletRequest request, T instance, Class<T> clazz) {
        String userLocalTimeZone = fetchTZ(request);
        for (Field field : clazz.getDeclaredFields()) {
            if (field.getType() == ZonedDateTime.class) {
                field.setAccessible(true);
                try {
                    ZonedDateTime x = (ZonedDateTime) (field.get(instance));

                    if (x != null) {
                        field.set(instance, x.withZoneSameInstant(ZoneId.of(userLocalTimeZone)));
                    }
                } catch (IllegalAccessException ignored) {
                }
            }
        }
    }

    public static <T> void tzConverterDown(HttpServletRequest request, List<T> instanceList, Class<T> clazz) {
        instanceList.forEach(instance -> tzConverterDown(request, instance, clazz));
    }

    public static <T> void tzConverterUp(HttpServletRequest request, List<T> instanceList, Class<T> clazz) {
        instanceList.forEach(instance -> tzConverterUp(request, instance, clazz));
    }

}
```

End.