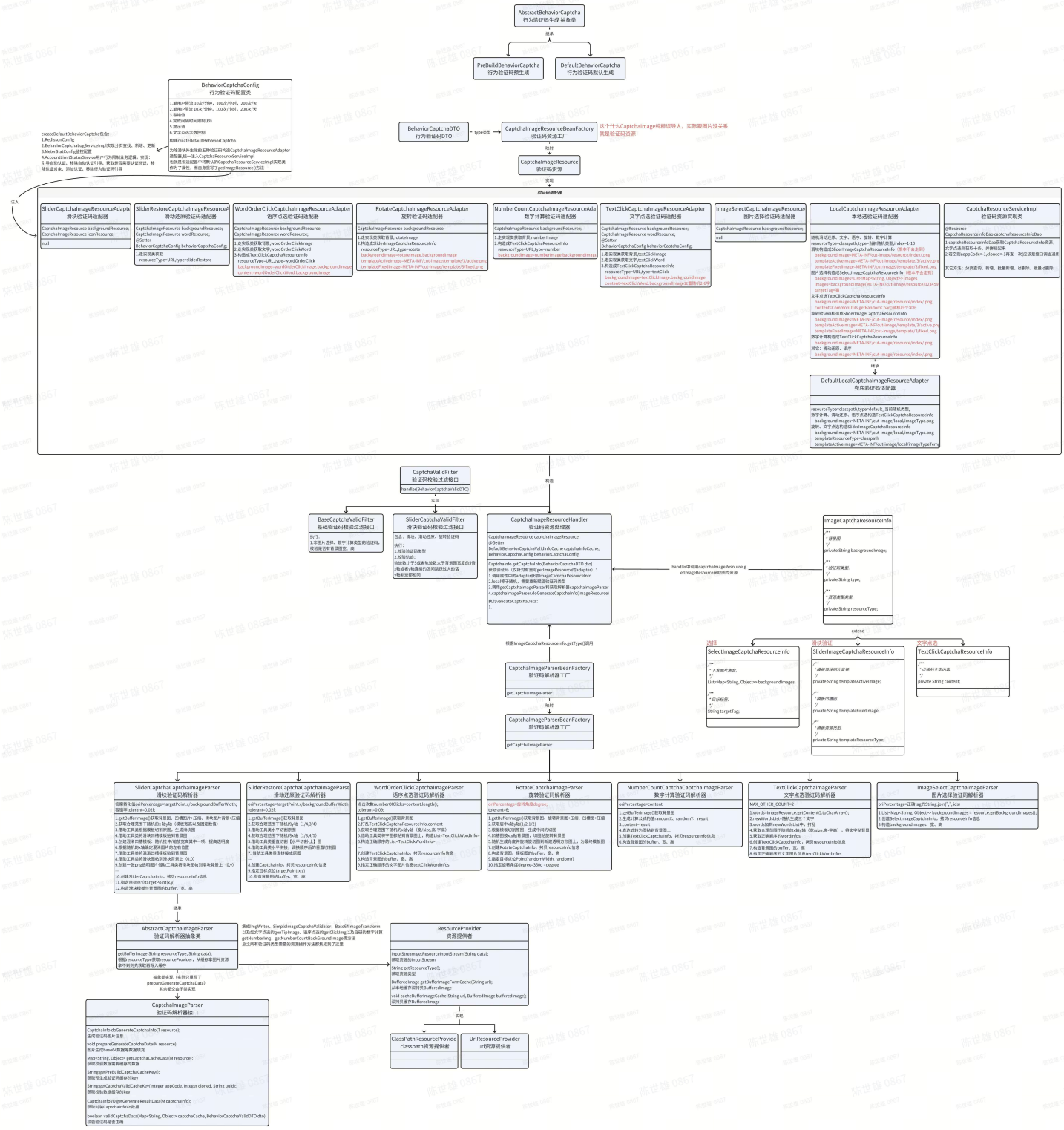
test

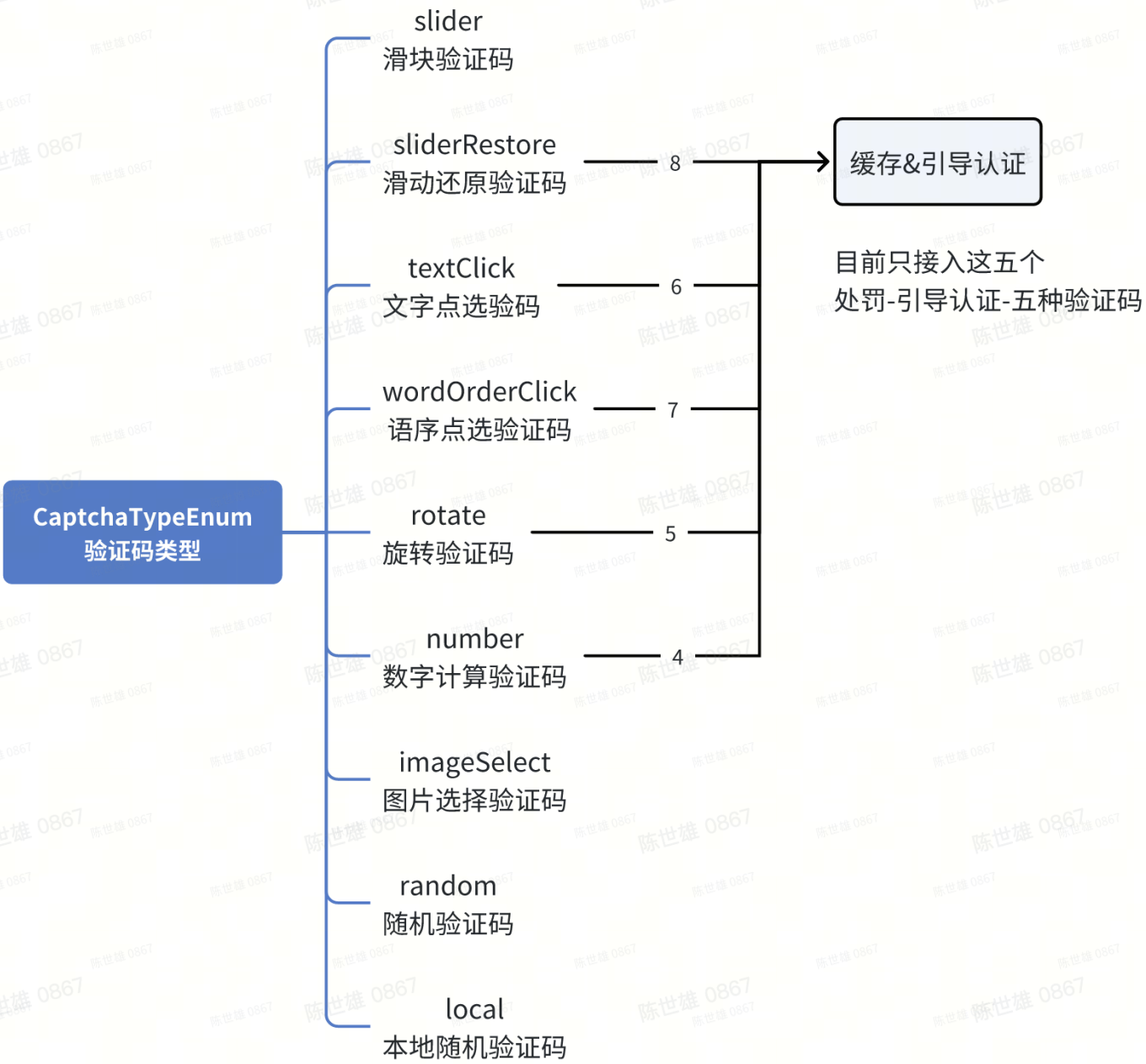


验证码-类关系

验证码图片信息


ImageCaptchaResourceInfo
/**
* 背景图.
*/
private String backgroundImage;
/**
* 验证码类型.
*/
private String type;
/**
* 资源类型类型.
*/
private String resourceType;
extend
图片选择?
SelectImageCaptchaResourceInfo
/**
* 下发图片集合.
*/
List<Map<String, Object>> backgroundImages;
/**
* 目标标签.
*/
String targetTag;
滑块验证
SliderImageCaptchaResourceInfo
/**
* 模板滑块图片背景.
*/
private String templateActiveImage;
/**
* 模板凹槽图.
*/
private String templateFixedImage;
/**
* 模板资源类型.
*/
private String templateResourceType;
文字点选
TextClickCaptchaResourceInfo
/**
* 点选的文字内容.
*/
private String content;

验证码类型



验证码-代码逻辑

定时生成验证码到缓存 (平台自己扩展)

```
1  //1.上述五种【缓存&引导认证类型】，每分钟各生成10个。这个方法可以改成void
2  public CaptchaInfoVO captchaGenerate(BehaviorCaptchaDTO dto)
3
4  //2.根据type类型映射资源类型，实际是Adapter适配器
5  CaptchaImageResource captchaImageResource =
   CaptchaImageResourceBeanFactory.getCaptchaImageResource(dto.getType());
6
7  //3.将映射的资源类型构造成handler处理器
```

```
CaptchaImageResourceHandler captchaImageResourceHandler = new CaptchaImageResourceHandler(captchaImageResource);

//4.根据type类型在handler处理器中映射获取CaptchaImageParser解析器
//这步完全可以跟23一样直接集成在处理器中的，所有的获取、生成、校验都交给处理器执行，才不会乱
CaptchaImageParser<ImageCaptchaResourceInfo, CaptchaInfo> captchaImageParser = captchaImageResourceHandler.getCaptchaImageParser(dto.getType());

//5.获取该解析器对应的缓存key信息eg:"j63:captcha:slider"，以及redis队列
String cacheKey = captchaImageParser.getPreBuildCaptchaCacheKey();
RDeque<CaptchaInfo> deque = redissonClient.getDeque(cacheKey, CommonConstant.JACKSON_CODEC);

//6.如果队列大小超过最大配置阈值1000则返回
if (deque.size() > Integer.parseInt(behaviorCaptchaConfig.getQueueCapacity())) {
    log.info("PreBuild Captcha deque full:{}", cacheKey);
    return null;
}

//7.获取验证码信息，包括:
//1、调用23中集成进来的CaptchaImageResource属性，获取ImageCaptchaResourceInfo
//2、再根据imageResource.type获取captchaImageParser解析器（所以4的时候就说明了集成在里面就行了，没必要一次生成拿两次Parser解析器）
//3、captchaImageParser.doGenerateCaptchaInfo(imageResource);生成验证码图片
CaptchaInfo captchaInfo = captchaImageResourceHandler.getCaptchaInfo(dto);

//8.填充验证码信息，包括
//1、captchaInfo.getType()获取captchaImageParser解析器
//2、背景、模板图，生成base64数据填充（这步是不是可以再7-3里面实现就可以了）
captchaImageResourceHandler.prepareGenerateCaptchaData(captchaInfo);

//9.增加到缓存队列
deque.add(captchaInfo);
```

客户端获取验证码 behaviorCaptcha/getCaptcha

```
//0.前置校验：验证码类型、用户uuid+ip的频控分+时+天、随机验证码转换具体验证码
beforeCaptchaGenerate(BehaviorCaptchaDTO dto)

//1.客户端获取验证码behaviorCaptcha.getCaptcha()

//2.走状态服务获取该用户需要什么类型的验证码
```

```
Map<String, String> accountCaptchaRiskStatus = accountStatusUnifiedService.getAccountCaptchaRiskStatus(dto.getAppCode(), dto.getAccountUuid());
String type = MapUtils.getString(accountCaptchaRiskStatus, "hitType");
CaptchaTypeEnum captchaTypeEnum = CaptchaTypeEnum.getCaptchaTypeByType(type);

//3.type->Resource(Adapter)->Handler->Parser.doGenerateCaptchaInfo生成验证码，返回CaptchaInfo
//填充验证码信息，转换图片base64:backgroundData/templateActiveData/tipImageData/backgroundImages.backgroundData/tipImageData的图Base64、tipImageBuffer
CaptchaInfo captchaInfo = generateCaptcha(captchaImageResourceHandler, dto);
    CaptchaInfo captchaInfo = captchaImageResourceHandler.getCaptchaInfo(dto);
    captchaImageParser.prepareGenerateCaptchaData(captchaInfo);

//缓存需要验证的结果数据
captchaImageResourceHandler.cacheGenerateCaptchaData(captchaInfo, dto);
    //1.缓存校验数据resultMap：为啥不跟oriPercentage一致?
    //  图片选择resultMap.put("id", 正确tag对应的ids);
    //  数字计算resultMap.put("result", resource.getContent());
    //  旋转resultMap.put("degree", resource.getDegree());resultMap.put("oriPercentage", targetPoint.x/backgroundBufferWidth);
    //  滑块resultMap.put("backgroundWidth", resource.getBackgroundBufferWidth());resultMap.put("oriPercentage", targetPoint.x/backgroundBufferWidth);
    //  滑动还原resultMap.put("backgroundWidth", resource.getBackgroundBufferWidth());resultMap.put("oriPercentage", targetPoint.x/backgroundBufferWidth);
    //  文字点选resultMap.put("workInfos", resource.getTextClickWordInfos());resultMap.put("backgroundWidth", resource.getBackgroundBufferWidth());resultMap.put("backgroundHeight", resource.getBackgroundBufferHeight());
    //  语序点选resultMap.put("workInfos", resource.getTextClickWordInfos());resultMap.put("backgroundWidth", resource.getBackgroundBufferWidth());resultMap.put("backgroundHeight", resource.getBackgroundBufferHeight());

    //2.填充答案oriPercentage:
    //  图片选择String.join(",", ids);
    //  数字计算resource.getContent()
    //  旋转resource.getDegree()为啥和resultMap不一样+resource.setTolerant(behaviorCaptchaConfig.getRotateTolerant());不应该是在doGenerateCaptchaInfo实现吗?
    //  滑块targetPoint.x/backgroundBufferWidth+resource.setTolerant(0.02f)
    //  滑动还原targetPoint.x/backgroundBufferWidth+resource.setTolerant(0.02f)
    //  文字点选0.09
```

```
    //  语序点选0.09
    Map<String, Object> captchaCacheData = captchaImageParser.getCaptchaCacheData(captchaInfo);
    String captchaCacheKey = captchaImageParser.getCaptchaValidCacheKey(dto.getAppCode(), dto.getCloned(), dto.getAccountUuid());
    // 新增下发时间startTime，存redis，三分钟后过期
    captchaInfoCache.cacheCaptchaInfoToHash(captchaCacheKey, captchaCacheData);

// 获取下发前端数据
CaptchaInfoVO generateResultData = captchaImageResourceHandler.getGenerateResultData(captchaInfo);
    //根据type将TextClickCaptchaInfo转换成CaptchaInfoVO

getCaptchaImageParser(captchaInfo.getType()).getGenerateResultData(captchaInfo);

//记录日志
saveOptLog(dto, captchaInfo, generateResultData);

//记录监控数据
meterStatConfig.incrMeter(CAPTCHA_MONITOR, CAPTCHA_MONITOR_ADD, CAPTCHA_MONITOR, CAPTCHA_MONITOR_ADD);
```

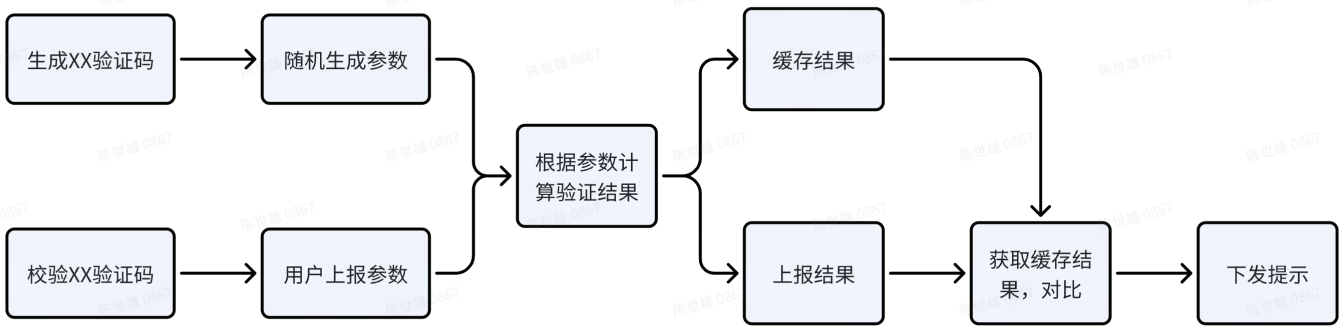
客户端校验验证码 behaviorCaptcha/validate

```
在初始化时，也就是校验前就已经先把CaptchaValidFilter的三个实现类BaseCaptchaValidFilter、SliderCaptchaValidFilter、CaptchaImageResourceHandler都当成【拦截器】全都丢到了AbstractBehaviorCaptcha的captchaValidFilterList里面
```

```
//校验数据，将结果写入ThreadLocal，更新日志
validateCaptchaGenerate(BehaviorCaptchaValidDTO dto)
    //前置条件：移除ThreadLocal
    invokeBeforeValidCaptchaPostProcessors(dto);
        //三个拦截器验证,构造结果，记录监控；正确则移除行为验证码引导
        //BaseCaptchaValidFilter:非数字计算、图片选择，验证宽高
        //SliderCaptchaValidFilter:滑块、滑动还原、旋转验证，验证迹数小于5/轨迹数大于背景图宽度的5倍/y轴轨迹都相同/x轴跳跃大于200/y轴跳跃大于100
        //CaptchaImageResourceHandler：使用各自Parser拿到写入的resultMap，将结果写入ThreadLocal
        //1拿不到就代表超时3分钟
        //2完成间隔大于intervalTime也超时
```

```
        //3各自Parser的validCaptchaData验证答案（这块写的真的是一坨，redis存储、校验方法、对照字段全都不统一瞎写，尤其是生成的时候oriPercentage和resultMap不一致，结果在这不还要重新推导oriPercentage结果。而且维护resultMap就够了，保持统一，没必要维护oriPercentage单纯的因为日志需要这个字段，保存日志时resultMap里面也能拿到了）
        captchaValidFilter.handler(dto)
        //移除缓存
        captchaImageResourceHandler.deleteCaptchaCacheData(dto);

//从ThreadLocal获取结果，再移除ThreadLocal
CaptchaValidVO captchaValidVO = DefaultBehaviorCaptcha.local.get();
DefaultBehaviorCaptcha.local.remove();
return captchaValidVO;
```

建议改进流程为：



验证码-底层源码解析

验证码开源项目：TIANAI-CAPTCHA 为了方便表述以该源码解读，重点在于每种验证码的实现，涉及到较多awt相关用法

验证码-底层源码解析

验证码开源项目：TIANAI-CAPTCHA 为了方便表述以该源码解读，重点在于每种验证码的实现，涉及到较多awt相关用法

公用部分代码

```
/**
 * 获取指定资源的输入流。
 * 如果提供的资源有效，并且输入流集合非空，则将该资源的输入流添加到输入流集合中。
 *
 * @param resource 需要获取输入流的资源对象。
 * @param inputStreams 用于收集输入流的集合，可以为null。
 * @return 返回资源对应的输入流，如果资源无效则返回null。
```

```
 */
protected InputStream getResourceInputStream(Resource resource,
Collection<InputStream> inputStreams) {
    // 尝试从资源管理器获取资源的输入流
    InputStream stream =
getImageResourceManager().getResourceInputStream(resource);
    // 如果获取成功且输入流集合非空，则将该输入流添加到集合中
    if (stream != null && inputStreams != null) {
        inputStreams.add(stream);
    }
    return stream;
}


/**
 * 将 InputStream 资源包装为 BufferedImage 对象。
 *
 * @param resource 输入流资源，不可为 null。
 * @return BufferedImage 对象，代表了输入流中的图像。
 * @throws IllegalArgumentException 如果资源为 null，则抛出此异常。
 */
@SneakyThrows
public static BufferedImage wrapFile2BufferedImage(InputStream resource) {
    // 检查资源是否为 null
    if (resource == null) {
        throw new IllegalArgumentException("包装文件到 BufferedImage 失败， file不
能为空");
    }
    // 关闭磁盘缓存以避免读取图像时使用磁盘缓存
    ImageIO.setUseCache(false);
    // 从输入流读取图像并返回
    return ImageIO.read(resource);
}


/**
 * 获取资源图片。
 *
 * @param resource 表示资源对象，用于定位图片资源。
 * @return 返回一个BufferedImage对象，表示加载成功的图片。
 */
protected BufferedImage getResourceImage(Resource resource) {
    // 从资源对象获取输入流
    InputStream stream = getResourceInputStream(resource, null);
    // 将输入流包装为BufferedImage对象
    BufferedImage bufferedImage =
CaptchaImageUtils.wrapFile2BufferedImage(stream);
```

```
    // 关闭输入流
    closeStream(stream);
    return bufferedImage;
}


/**
 * 分割图片功能实现
 * 此方法根据指定的位置和方向将输入的图片分割成两部分，并返回一个包含两部分的新图片数组。
 * 如果方向为水平，那么分割点将位于图片的垂直中心线上；如果方向为垂直，分割点将位于图片的水平中心线上。
 *
 * @param pos 分割点的位置。当direction为true时，pos表示图片底部的像素位置；当direction为false时，pos表示图片右侧的像素位置。
 * @param direction 分割方向。true表示沿水平方向分割，false表示沿垂直方向分割。
 * @param img 待分割的源图片，类型为BufferedImage。
 * @return BufferedImage[] 返回一个包含两个BufferedImage对象的数组，分别对应分割后的上半部分和下半部分（水平分割）或左半部分和右半部分（垂直分割）。
 * @throws IllegalArgumentException 如果pos小于0或大于图片相应方向的尺寸，将抛出此异常。
 */
public static BufferedImage[] splitImage(int pos, boolean direction, BufferedImage img) throws IllegalArgumentException {
    // 检查输入参数的有效性
    if (pos < 0 || (direction && pos > img.getHeight()) || (!direction && pos > img.getWidth())) {
        throw new IllegalArgumentException("Invalid position for splitting the image.");
    }

    // 初始化分割后图片的宽高参数
    int startImageWidth;
    int startImageHeight;
    int endImageWidth;
    int endImageHeight;
    int endScanX;
    int endScanY;

    // 根据分割方向计算起始图片和结束图片的宽高及扫描起点
    if (direction) {
        // 水平分割
        startImageHeight = img.getHeight() - pos;
        startImageWidth = img.getWidth();
        endImageWidth = img.getWidth();
        endImageHeight = pos;
        endScanX = 0;
        endScanY = startImageHeight;
```

```
91      } else {
92          // 垂直分割
93          startImageWidth = pos;
94          startImageHeight = img.getHeight();
95          endImageWidth = img.getWidth() - startImageWidth;
96          endImageHeight = img.getHeight();
97          endScanX = pos;
98          endScanY = 0;
99      }
100
101     // 分割图片操作
102     // 1. 提取起始图片的RGB值并创建新图片
103     int[] rgbArr = new int[startImageWidth * startImageHeight];
104     img.getRGB(0, 0, startImageWidth, startImageHeight, rgbArr, 0,
    startImageWidth);
105     int type = img.getColorModel().getTransparency();
106     BufferedImage startImg = new BufferedImage(startImageWidth,
    startImageHeight, type);
107     startImg.setRGB(0, 0, startImageWidth, startImageHeight, rgbArr, 0,
    startImageWidth);
108
109     // 2. 提取结束图片的RGB值并创建新图片
110     rgbArr = new int[endImageWidth * endImageHeight];
111     img.getRGB(endScanX, endScanY, endImageWidth, endImageHeight, rgbArr, 0,
    endImageWidth);
112     BufferedImage endImg = new BufferedImage(endImageWidth, endImageHeight,
    type);
113     endImg.setRGB(0, 0, endImageWidth, endImageHeight, rgbArr, 0,
    endImageWidth);
114
115     // 组装分割后的图片数组并返回
116     BufferedImage[] splitImageArr = new BufferedImage[2];
117     splitImageArr[0] = startImg;
118     splitImageArr[1] = endImg;
119     return splitImageArr;
120 }
121
122
123 /**
124  * 生成一个范围内的随机整数。
125  *
126  * @param origin 随机数的起始值（包含）。
127  * @param bound 随机数的上限值（不包含）。
128  * @return 在给定范围内的随机整数。
129  */
130 protected int randomInt(int origin, int bound) {
131     // 使用ThreadLocalRandom生成指定范围内的随机数
```

```
132     return ThreadLocalRandom.current().nextInt(origin, bound);
133 }
134 
135 
136 /**
137  * 拼接图片
138  *
139  * @param direction 指定拼接方向，true为水平方向，false为垂直方向
140  * @param width     拼接后图片的宽度
141  * @param height    拼接后图片的高度
142  * @param imgArr    要拼接的图片数组
143  * @return 返回拼接后的BufferedImage对象
144  */
145 public static BufferedImage concatImage(boolean direction, int width, int
    height, BufferedImage... imgArr) {
146     // 初始化位置
147     int pos = 0;
148     // 创建新的图片，其透明度与第一张图片保持一致
149     BufferedImage newImage = new BufferedImage(width, height,
    imgArr[0].getColorModel().getTransparency());
150     // 遍历所有输入的图片
151     for (BufferedImage img : imgArr) {
152         // 获取当前图片的RGB值
153         int[] rgbArr = new int[width * height];
154         img.getRGB(0, 0, img.getWidth(), img.getHeight(), rgbArr, 0,
    img.getWidth());
155         // 根据指定的方向进行拼接
156         if (direction) {
157             // 水平方向拼接
158             newImage.setRGB(pos, 0, img.getWidth(), img.getHeight(), rgbArr,
    0, img.getWidth());
159             pos += img.getWidth();
160         } else {
161             // 垂直方向拼接
162             newImage.setRGB(0, pos, img.getWidth(), img.getHeight(), rgbArr,
    0, img.getWidth());
163             pos += img.getHeight();
164         }
165     }
166     // 返回拼接后的图片
167     return newImage;
168 }
169 
170 
```

滑动还原验证码

```java
/**
 * 生成验证码图像的实现方法。
 * 此方法根据传入的CaptchaExchange对象中的参数，生成一个定制化的滑块验证码图像，
 * 并将必要的验证信息存储在CaptchaExchange对象中以便后续验证使用。
 *
 * @param captchaExchange {@link CaptchaExchange} 对象，封装了验证码生成所需的参数、
 *                        背景资源信息及最终要存储的生成结果。方法执行后，该对象将被填充
 *                        背景图像、分割与拼接后的滑块图像数据及验证所需的位置坐标。
 */
@Override
public void doGenerateCaptchaImage(CaptchaExchange captchaExchange) {
    // 获取验证码生成参数
    GenerateParam param = captchaExchange.getParam();

    // 根据类型和背景图片标签随机选择资源，作为验证码的基础背景
    Resource resourceImage = requiredRandomGetResource(param.getType(), param.getBackgroundImageTag());

    // 将资源转换为BufferedImage，以便进行图像处理
    BufferedImage bgImage = getResourceImage(resourceImage);

    // 计算垂直分割线的位置，使得分割线随机分布在图像高度的1/4到3/4之间
    int spacingY = bgImage.getHeight() / 4;
    int randomY = randomInt(spacingY, bgImage.getHeight() - spacingY);

    // 根据计算出的Y轴分割点，将背景图像分割为上下两部分
    BufferedImage[] bgImageSplit = splitImage(randomY, true, bgImage);

    // 计算水平分割线的位置，使得分割线随机分布在除前1/5宽度外的区域
    int spacingX = bgImage.getWidth() / 8;
    int randomX = randomInt(spacingX, bgImage.getWidth() - spacingX / 5);

    // 根据计算出的X轴分割点，将上半部分背景图像进一步分割
    BufferedImage[] bgImageTopSplit = splitImage(randomX, false, bgImageSplit[0]);

    // 将分割出的上半部分图像的两块按顺序横向拼接，生成滑块图像
    BufferedImage sliderImage = concatImage(true,
            bgImageTopSplit[0].getWidth()
                    + bgImageTopSplit[1].getWidth(), bgImageTopSplit[0].getHeight(), bgImageTopSplit[1], bgImageTopSplit[0]);

    // 将下半部分背景图像与滑块图像纵向拼接，完成最终的验证码图像
    bgImage = concatImage(false, bgImageSplit[1].getWidth(), sliderImage.getHeight() + bgImageSplit[1].getHeight(),
```

```
            sliderImage, bgImageSplit[1]);

    // 创建Data对象存储滑块的起始位置坐标
    Data data = new Data();
    data.x = randomX; // 滑块的水平起始位置
    data.y = randomY; // 分割线的垂直位置，虽然此处未直接用于滑块，但可能用于其他验证逻辑

    // 将生成的图像数据及验证数据设置回CaptchaExchange对象
    captchaExchange.setTransferData(data);
    captchaExchange.setBackgroundImage(bgImage);
    captchaExchange.setResourceImage(resourceImage);
}


/**
 * 在封装图像验证码信息之前进行处理。
 * 对于指定类型的验证码，打乱其背景图片的顺序，增加验证码的难度。
 *
 * @param captchaExchange 包含验证码交换信息的对象，进来的参数和图片会被修改
 * @param context 图像验证码生成的上下文信息，此处未使用
 */
@Override
public void beforeWrapImageCaptchaInfo(CaptchaExchange captchaExchange, ImageCaptchaGenerator context) {
    GenerateParam param = captchaExchange.getParam();
    // 检查是否需要对验证码图片进行打乱操作
    if (Boolean.TRUE.equals(param.getParam(SHUFFLE_IMAGE_KEY))) {
        // 判断验证码类型，对于特定的验证码类型执行打乱操作
        if (CaptchaTypeConstant.SLIDER.equals(param.getType())
                || CaptchaTypeConstant.ROTATE.equals(param.getType())
                || CaptchaTypeConstant.IMAGE_CLICK.equals(param.getType())
                || CaptchaTypeConstant.WORD_ORDER_IMAGE_CLICK.equals(param.getType())
                || CaptchaTypeConstant.WORD_IMAGE_CLICK.equals(param.getType())
                || CaptchaTypeConstant.ROTATE_DEGREE.equals(param.getType())
                || CaptchaTypeConstant.SCRAPE.equals(param.getType())
        ) {
            int x = 5; // 默认打乱的x轴坐标
            int y = 2; // 默认打乱的y轴坐标
            // 对于旋转类型验证码，调整打乱的坐标点
            if (CaptchaTypeConstant.ROTATE_DEGREE.equals(param.getType())) {
                y = 4;
                x = 4;
            }
            // 执行图片打乱操作，并更新验证码的背景图片和相关自定义数据
```

```
            ImageShuffleUtils.ShuffleImageResponse shuffle =
ImageShuffleUtils.shuffle(captchaExchange.getBackgroundImage(), x, y);
            captchaExchange.setBackgroundImage(shuffle.getImage());
            ShuffleViewData viewData = new ShuffleViewData();
            viewData.setX(x);
            viewData.setY(y);
            viewData.setPos(shuffle.getPos());
            // 将打乱后的坐标和位置信息保存到自定义数据中
            captchaExchange.getCustomData().putViewData("shuffle", viewData);
        }
    }
}


/**
 * 生成验证码图像。
 *
 * @param param 生成验证码的参数。
 * @return ImageCaptchaInfo 包含验证码图像和相关数据的信息对象。
 */
@Override
public ImageCaptchaInfo generateCaptchaImage(GenerateParam param) {
    assertInit(); // 确保初始化完成
    CustomData data = new CustomData(); // 创建自定义数据对象
    // 创建验证码交换对象，用于在生成过程中传递数据和参数
    CaptchaExchange captchaExchange = CaptchaExchange.create(data, param);
    // 在生成验证码核心逻辑之前调用， 用于拦截验证码生成、限流、自定义返回数据等处理
    ImageCaptchaInfo imageCaptchaInfo =
applyPostProcessorBeforeGenerate(captchaExchange, this);
    // 如果前处理返回了验证码信息，则直接返回，不再继续生成过程
    if (imageCaptchaInfo != null) {
        return imageCaptchaInfo;
    }
    // 执行实际的验证码生成逻辑
    doGenerateCaptchaImage(captchaExchange);
    // 在封装验证码信息前应用后处理，可能用于对生成结果的额外处理
    applyPostProcessorBeforeWrapImageCaptchaInfo(captchaExchange, this);
    // 封装验证码信息，准备返回（下发容错值等）
    imageCaptchaInfo = wrapImageCaptchaInfo(captchaExchange);
    // 应用生成后处理，可能用于对验证码信息的最后修改或验证
    applyPostProcessorAfterGenerateCaptchaImage(captchaExchange,
imageCaptchaInfo, this);
    return imageCaptchaInfo; // 返回生成的验证码信息
}


```

点选验证码（图片点选、文字点选、语序点选）

公共部分代码

```
/**
 * 生成文字点选验证码图像。此函数根据给定的参数，从资源库中随机选取背景图片和多个点击图片，
 * 并将这些图片随机地叠加到背景图片上，形成一个完整的文字点选验证码图片。
 *
 * @param captchaExchange 包含验证码生成参数和背景图片标签的对象。输入参数包括生成验证码的类型和背景图片标签，输出对象包含生成的验证码图片、需点击识别的元素列表以及原始资源图片。
 * @param captchaExchange.getParam() 生成验证码的参数，包括验证码类型和背景图片标签。
 * @param captchaExchange.getResourceImage() 输入的背景图片资源。
 * @param captchaExchange.getTransferData() 输出的点击识别元素列表，包括元素的提示信息、坐标和尺寸。
 * @param captchaExchange.setBackgroundImage() 设置生成的验证码图片。
 * @param captchaExchange.setResourceImage() 设置原始资源图片。
 * @throws IllegalStateException 如果随机生成的点击图片数量小于请求的数量，抛出异常。
 */
@SneakyThrows
@Override
public void doGenerateCaptchaImage(CaptchaExchange captchaExchange) {
    GenerateParam param = captchaExchange.getParam();
    // 根据参数类型和标签获取一个随机背景图片资源
    Resource resourceImage = requiredRandomGetResource(param.getType(), param.getBackgroundImageTag());

    BufferedImage bgImage = getResourceImage(resourceImage);

    List<Resource> imgTips = randomGetClickImgTips(param);
    int allImages = imgTips.size();
    List<ClickImageCheckDefinition> clickImageCheckDefinitionList = new ArrayList<>(allImages);
    int avg = bgImage.getWidth() / allImages;
    // 检查生成的点击图片数量是否等于请求的数量
    if (allImages < imgTips.size()) {
        throw new IllegalStateException("随机生成点击图片小于请求数量， 请求生成数量=" + allImages + ",实际生成数量=" + imgTips.size());
    }
    for (int i = 0; i < allImages; i++) {
        // 获取并处理一个需点击的图片元素
        ImgWrapper imgWrapper = getClickImg(imgTips.get(i));
        BufferedImage image = imgWrapper.getImage();
```

```
        int clickImgWidth = image.getWidth();
        int clickImgHeight = image.getHeight();
        // 计算图片元素在背景图上的随机位置
        int randomX;
        if (i == 0) {
            randomX = 1;
        } else {
            randomX = avg * i;
        }
        int randomY = randomInt(10, bgImage.getHeight() - clickImgHeight);
        // 将图片元素叠加到背景图上
        CaptchaImageUtils.overlayImage(bgImage, imgWrapper.getImage(), randomX, randomY);
        // 创建并存储图片元素的识别信息
        ClickImageCheckDefinition clickImageCheckDefinition = new ClickImageCheckDefinition();
        clickImageCheckDefinition.setTip(imgWrapper.getTip());
        clickImageCheckDefinition.setX(randomX + clickImgWidth / 2);
        clickImageCheckDefinition.setY(randomY + clickImgHeight / 2);
        clickImageCheckDefinition.setWidth(clickImgWidth);
        clickImageCheckDefinition.setHeight(clickImgHeight);
        clickImageCheckDefinitionList.add(clickImageCheckDefinition);
    }
    // 对点击图片元素进行筛选和排序
    List<ClickImageCheckDefinition> checkClickImageCheckDefinitionList = filterAndSortClickImageCheckDefinition(clickImageCheckDefinitionList);
    // 将生成的验证码图片和处理后的点击元素列表设置到输出对象中
    captchaExchange.setBackgroundImage(bgImage);
    captchaExchange.setTransferData(checkClickImageCheckDefinitionList);
    captchaExchange.setResourceImage(resourceImage);
}

```

图片点选

```
/**
 * 随机获取点击图片提示信息列表。
 * 该方法会根据指定条件随机获取一定数量的图片资源作为提示信息。
 *
 * @param param 生成参数对象，包含模板图片标签等信息。
 * @return 返回一个资源列表，列表中的每个元素都是一个图片资源对象。
 */
@Override
protected List<Resource> randomGetClickImgTips(GenerateParam param) {
    // 计算提示信息的总数量，包括干扰信息和点击检查信息
```

```
    int tipSize = interferenceCount + checkClickCount;
    // 使用HashSet存储提示信息，以确保元素唯一
    Set<Resource> tipSet = new HashSet<>(tipSize);

    // 循环，直到HashSet中存储的元素数量达到指定的tipSize
    while (tipSet.size() != tipSize) {
        // 从图片资源管理器中随机获取一个与指定模板图片标签匹配的资源
        Resource resource = getImageResourceManager().randomGetResource(CommonConstant.IMAGE_CLICK_ICON, param.getTemplateImageTag());
        // 将获取到的资源添加到HashSet中
        tipSet.add(resource);
    }
    // 将HashSet转换为ArrayList，并返回
    return new ArrayList<>(tipSet);
}


/**
 * 图片点击：生成点击图片。
 * 该方法首先根据提供的资源对象生成一张图片，然后对该图片随机旋转一个角度，最后确保图片的尺寸符合预设的宽高要求。
 * 如果图片尺寸与预设宽高不符，将进行缩放以符合要求。
 *
 * @param tip 资源对象，用于生成图片。
 * @return ImgWrapper 包含处理后图片和原始资源对象的包装类实例。
 */
@Override
public ImgWrapper getClickImg(Resource tip) {
    // 生成随机旋转角度
    int randomDeg = randomInt(0, 85);
    BufferedImage bufferedImage = getResourceImage(tip);
    // 对图片进行随机角度的旋转
    bufferedImage = CaptchaImageUtils.rotateImage(bufferedImage, randomDeg);
    // 检查图片尺寸是否符合预设要求，若不符合则进行缩放
    if (bufferedImage.getWidth() != clickImgWidth || bufferedImage.getHeight() != clickImgHeight) {
        // 缩放图片以符合预设的宽高
        bufferedImage = CaptchaImageUtils.toBufferedImage(bufferedImage.getScaledInstance(clickImgWidth, clickImgHeight, Image.SCALE_SMOOTH));
    }
    // 返回包含处理后图片和原始资源对象的包装类实例
    return new ImgWrapper(bufferedImage, tip);
}
```

文字点选

```java
/**
 * 随机获取点击文字提示信息的列表。
 * <p>
 * 根据给定的参数，生成一个包含指定数量随机提示文字的列表。
 * 列表中的每个元素都是一个不带图片资源的随机文字。
 *
 * @param param 生成参数，目前未使用，但保留以支持可能的未来扩展。
 * @return 返回一个包含随机提示文字的列表。列表大小由干扰计数和点击检查计数决定。
 */
@Override
protected List<Resource> randomGetClickImgTips(GenerateParam param) {
    // 计算提示信息的总数量
    int tipSize = interferenceCount + checkClickCount;
    // 获取当前线程的随机数生成器
    ThreadLocalRandom random = ThreadLocalRandom.current();
    // 初始化存储随机提示信息的列表
    List<Resource> tipList = new ArrayList<>(tipSize);
    // 生成随机提示信息并填充到列表中
    for (int i = 0; i < tipSize; i++) {
        // 从预定义的字符集中随机选择一个字符作为提示文字
        String randomWord = FontUtils.getRandomChar(random);
        // 将随机文字封装为资源对象并添加到列表
        tipList.add(new Resource(null, randomWord));
    }
    // 返回包含随机文字的列表
    return tipList;
}


/**
 * 文字点击：生成点击图片。
 *
 * @param tip 提示资源，包含需要展示的文本数据。
 * @return ImgWrapper 包含生成的图片和对应的提示资源。
 */
@Override
public ImgWrapper getClickImg(Resource tip) {
    ThreadLocalRandom random = ThreadLocalRandom.current();
    // 生成随机颜色
    Color randomColor = CaptchaImageUtils.getRandomColor(random);
    // 随机生成文字旋转角度
    int randomDeg = randomInt(0, 85);
    // 随机选择字体
    FontWrapper fontWrapper = fonts.get(randomInt(fonts.size()));
```

```
    Font font = fontWrapper.getFont();
    float currentFontTopCoef = fontWrapper.getCurrentFontTopCoef();
    // 使用选定的颜色、文本、字体等生成带有随机角度的文字图片
    BufferedImage fontImage = CaptchaImageUtils.drawWordImg(randomColor,
            tip.getData(),
            font,
            currentFontTopCoef,
            clickImgWidth,
            clickImgHeight,
            randomDeg);
    return new ImgWrapper(fontImage, tip);
}
```

语序点选

```
/**
 * 随机获取语序点选提示信息列表。
 * <p>此方法通过随机选择一定数量的图片验证码模板，然后将其中的提示信息收集到一个列表中返回。</p>
 *
 * @param param 生成参数，当前方法未使用此参数，但为了实现接口而保留。
 * @return 返回一个包含随机选择的图片验证码提示信息的列表。
 */
@Override
protected List<Resource> randomGetClickImgTips(GenerateParam param) {
    // 从图片资源管理器中随机获取一个验证码模板集合
    ResourceMap resourceMap = getImageResourceManager().randomGetTemplate(CaptchaTypeConstant.WORD_ORDER_IMAGE_CLICK, null);
    // 将模板集合中的资源转换为列表
    List<Resource> tips = new ArrayList<>(resourceMap.values());
    // 打乱提示信息列表顺序，以增加随机性
    Collections.shuffle(tips);
    return tips;
}


/**
 * 语序点选：生成点击式验证码的图像。
 * <p>
 * 根据给定的提示资源（tip），随机选择一种字体、颜色和角度，然后绘制一个包含随机文字的图像。
 * </p>
 *
 * @param tip 提示资源，包含需要在图像中显示的文字内容。
```

```
     * @return ImgWrapper 包含生成的图像和对应提示资源的包装对象。
     */
    @Override
    public ImgWrapper getClickImg(Resource tip) {
        // 随机选择一种字体
        FontWrapper fontWrapper = fonts.get(randomInt(fonts.size()));
        Font font = fontWrapper.getFont();
        float currentFontTopCoef = fontWrapper.getCurrentFontTopCoef();

        // 初始化随机数生成器
        ThreadLocalRandom random = ThreadLocalRandom.current();
        // 生成随机颜色
        Color randomColor = CaptchaImageUtils.getRandomColor(random);
        // 随机生成文字旋转角度
        int randomDeg = randomInt(0, 85);
        // 使用选定的字体、颜色、角度以及图像尺寸绘制文字图像
        BufferedImage fontImage = CaptchaImageUtils.drawWordImg(randomColor,
                tip.getData(),
                font,
                currentFontTopCoef,
                clickImgWidth,
                clickImgHeight,
                randomDeg);
        // 返回包含绘制结果的ImgWrapper对象
        return new ImgWrapper(fontImage, tip);
    }

```

旋转角度

```
/**
 * 生成带有随机旋转效果的验证码图像。
 * <p>
 * 该方法首先根据传入的参数从资源中获取背景图像，然后随机生成一个X坐标，转换为对应的角度，
 * 使用该角度旋转背景图像。旋转后的图像和相关数据（旋转角度、随机X坐标）被保存在CaptchaExchange对象中，
 * 以便后续处理。
 * </p>
 *
 * @param captchaExchange 验证码交换对象，包含生成验证码所需的参数，以及后续将保存旋转后的背景图像、原始资源图像和旋转数据。
 * @param GenerateParam 包含了验证码生成的相关参数，如类型和背景图片标签。
 * @param ResourceImage 背景图像的资源对象。
 * @param RotateData    保存旋转数据的内部类，包括旋转角度和随机X坐标。
 *
```

```
     * @see CaptchaExchange
     * @see GenerateParam
     * @see Resource
     * @see RotateData
     * @see CaptchaImageUtils#rotateImage(BufferedImage, double)
     */
    @Override
    public void doGenerateCaptchaImage(CaptchaExchange captchaExchange) {
        // 获取生成验证码所需的参数
        GenerateParam param = captchaExchange.getParam();
        // 根据参数获取随机的背景图像资源
        Resource resourceImage = requiredRandomGetResource(param.getType(),
    param.getBackgroundImageTag());
        // 加载背景图像资源为BufferedImage对象
        BufferedImage background = getResourceImage(resourceImage);

        // 随机生成一个X坐标，范围在背景图像宽度的1/3到宽度减去10之间
        int randomX = randomInt(background.getWidth() / 3, background.getWidth() -
    10);
        // 将随机X坐标转换为对应的角度，用于旋转背景图像
        double degree = 360d - randomX / ((background.getWidth()) / 360d);

        // 使用计算出的角度旋转背景图像
        BufferedImage rotateImage = CaptchaImageUtils.rotateImage(background,
    degree);

        // 将旋转后的图像和原始资源图像保存到CaptchaExchange对象中
        captchaExchange.setBackgroundImage(rotateImage);
        captchaExchange.setResourceImage(resourceImage);

        // 创建并设置旋转数据到captchaExchange中，以便后续使用
        RotateData rotateData = new RotateData();
        rotateData.degree = degree;
        rotateData.randomX = randomX;
        captchaExchange.setTransferData(rotateData);
    }
```

旋转图片

```
/**
 * 生成带有随机旋转效果的验证码图像。
 * <p>
 * 该方法首先根据传入的参数从资源中获取背景图像，然后随机生成一个X坐标，转换为对应的角度，
 * 使用该角度旋转背景图像。旋转后的图像和相关数据（旋转角度、随机X坐标）被保存在
CaptchaExchange对象中，
```

```
     * 以便后续处理。
     * </p>
     *
     * @param captchaExchange 验证码交换对象，包含生成验证码所需的参数，以及后续将保存旋转后的背景图像、原始资源图像
     *                        和旋转数据。
     *
     * @param GenerateParam 包含了验证码生成的相关参数，如类型和背景图片标签。
     * @param ResourceImage 背景图像的资源对象。
     * @param RotateData    保存旋转数据的内部类，包括旋转角度和随机X坐标。
     *
     * @see CaptchaExchange
     * @see GenerateParam
     * @see Resource
     * @see RotateData
     * @see CaptchaImageUtils#rotateImage(BufferedImage, double)
     */
    @Override
    public void doGenerateCaptchaImage(CaptchaExchange captchaExchange) {
        // 获取生成验证码所需的参数
        GenerateParam param = captchaExchange.getParam();
        // 根据参数获取随机的背景图像资源
        Resource resourceImage = requiredRandomGetResource(param.getType(), param.getBackgroundImageTag());
        // 加载背景图像资源为BufferedImage对象
        BufferedImage background = getResourceImage(resourceImage);

        // 随机生成一个X坐标，范围在背景图像宽度的1/3到宽度减去10之间
        int randomX = randomInt(background.getWidth() / 3, background.getWidth() - 10);
        // 将随机X坐标转换为对应的角度，用于旋转背景图像
        double degree = 360d - randomX / ((background.getWidth()) / 360d);

        // 使用计算出的角度旋转背景图像
        BufferedImage rotateImage = CaptchaImageUtils.rotateImage(background, degree);

        // 将旋转后的图像和原始资源图像保存到CaptchaExchange对象中
        captchaExchange.setBackgroundImage(rotateImage);
        captchaExchange.setResourceImage(resourceImage);

        // 创建并设置旋转数据到captchaExchange中，以便后续使用
        RotateData rotateData = new RotateData();
        rotateData.degree = degree;
        rotateData.randomX = randomX;
        captchaExchange.setTransferData(rotateData);
    }
```

验证码校验

基础校验

```
1  package cloud.tianai.captcha.validator.impl;
2
3  import cloud.tianai.captcha.common.constant.CaptchaTypeConstant;
4  import cloud.tianai.captcha.common.response.ApiResponse;
5  import cloud.tianai.captcha.common.response.ApiResponseStatusConstant;
6  import cloud.tianai.captcha.common.util.CaptchaUtils;
7  import cloud.tianai.captcha.common.util.CollectionUtils;
8  import cloud.tianai.captcha.common.util.ObjectUtils;
9  import
   cloud.tianai.captcha.generator.common.model.dto.ClickImageCheckDefinition;
10 import cloud.tianai.captcha.generator.common.model.dto.ImageCaptchaInfo;
11 import cloud.tianai.captcha.validator.ImageCaptchaValidator;
12 import cloud.tianai.captcha.validator.SliderCaptchaPercentageValidator;
13 import cloud.tianai.captcha.validator.common.constant.TrackTypeConstant;
14 import cloud.tianai.captcha.validator.common.model.dto.ImageCaptchaTrack;
15 import cloud.tianai.neuron.common.NumPy;
16 import lombok.Getter;
17 import lombok.Setter;
18 import lombok.extern.slf4j.Slf4j;
19
20 import java.util.ArrayList;
21 import java.util.HashMap;
22 import java.util.List;
23 import java.util.Map;
24 import java.util.stream.Collectors;
25
26 /**
27  * @Author: 天爱有情
28  * @date 2022/2/17 11:01
29  * @Description 基本的滑块验证校验 ， 值进行基本校验， 目前只校验用户是否滑动到缺口处，
   不校验行为轨迹
30  */
31 @Slf4j
32 public class SimpleImageCaptchaValidator implements ImageCaptchaValidator,
   SliderCaptchaPercentageValidator {
33
34     /** 默认的容错值. */
35     public static float DEFAULT_TOLERANT = 0.02f;
36     /** 验证数据 key. */
37     public static final String PERCENTAGE_KEY = "percentage";
```

```java
    /** 容错值key. */
    public static final String TOLERANT_KEY = "tolerant";
    /** 类型 key， 标识是哪张类型的验证码. */
    public static final String TYPE_KEY = "type";
    /** 计算当前验证码用户滑动的百分比率 - 生成时的百分比率，多个的话取均值. */
    public static final String USER_CURRENT_PERCENTAGE_STD =
"user_current_percentage_std";
    public static final String USER_CURRENT_PERCENTAGE =
"user_current_percentage";
    /** 容错值. */
    @Getter
    @Setter
    public float defaultTolerant = DEFAULT_TOLERANT;

    public SimpleImageCaptchaValidator() {
    }

    public SimpleImageCaptchaValidator(float defaultTolerant) {
        this.defaultTolerant = defaultTolerant;
    }

    /**
     * 计算给定位数值占最大位数值的百分比。
     *
     * @param pos 当前位数值，类型为Number，表示要计算的当前数值。
     * @param maxPos 最大位数值，类型为Number，表示用于比较的的最大数值。
     * @return 返回pos占maxPos的百分比，返回类型为float。
     */
    @Override
    public float calcPercentage(Number pos, Number maxPos) {
        // 计算百分比
        return pos.floatValue() / maxPos.floatValue();
    }

    /**
     * 检查新百分比与原始百分比的差异是否在可容忍范围内。
     * 这是一个重载方法，它会调用另一个带有容忍度参数的方法。
     *
     * @param newPercentage 新的百分比值。
     * @param oriPercentage 原始的百分比值。
     * @return 如果新百分比与原始百分比的差异在可容忍范围内，则返回true；否则返回
false。
     */
    @Override
    public boolean checkPercentage(Float newPercentage, Float oriPercentage) {
        return checkPercentage(newPercentage, oriPercentage, defaultTolerant);
    }
```

```
82
83
84      /**
85       * 检查新的百分比是否在原始百分比的一定误差范围内。
86       *
87       * @param newPercentage 新的百分比值，需要进行检查的值。
88       * @param oriPercentage 原始的百分比值，作为比较的基准值。
89       * @param tolerant 误差范围，允许新的百分比与原始百分比的最大偏差。
90       * @return 如果新的百分比值在原始百分比的误差范围内，则返回true；否则返回false。
91       */
92      @Override
93      public boolean checkPercentage(Float newPercentage, Float oriPercentage,
    float tolerant) {
94          // 检查传入的百分比值是否为null或非法值（如NaN或无穷大）
95          if (newPercentage == null || Float.isNaN(newPercentage) ||
    Float.isInfinite(newPercentage)
96                  || oriPercentage == null || Float.isNaN(oriPercentage) ||
    Float.isInfinite(oriPercentage)) {
97              return false;
98          }
99          // 计算误差范围的上下限
100         float maxTolerant = oriPercentage + tolerant;
101         float minTolerant = oriPercentage - tolerant;
102         // 检查新的百分比是否在误差范围内
103         return newPercentage >= minTolerant && newPercentage <= maxTolerant;
104     }
105
106     /**
107      * 生成图像验证码的有效数据。
108      * <p>此方法会先执行一些预处理，然后生成图像验证码的具体数据，最后进行后处理，将所有
    相关数据封装到一个Map中返回。</p>
109      *
110      * @param imageCaptchaInfo 包含图像验证码信息的对象，用于生成验证码的过程中。
111      * @return 返回一个包含验证码有效数据的Map，供其他部分使用。
112      */
113     @Override
114     public Map<String, Object> generateImageCaptchaValidData(ImageCaptchaInfo
    imageCaptchaInfo) {
115         // 初始化一个容量为8的HashMap，用于存放验证码数据
116         Map<String, Object> map = new HashMap<>(8);
117         // 在生成验证码数据之前执行预处理
118         if (beforeGenerateImageCaptchaValidData(imageCaptchaInfo, map)) {
119             // 如果预处理成功，则执行实际的验证码生成逻辑
120             doGenerateImageCaptchaValidData(map, imageCaptchaInfo);
121         }
122         // 在生成验证码数据之后执行后处理
123         afterGenerateImageCaptchaValidData(imageCaptchaInfo, map);
```

```
124         return map;
125     }
126 
127 
128     /**
129      * 在生成图像验证码之前验证数据的有效性。
130      *
131      * @param imageCaptchaInfo 包含图像验证码信息的对象，比如容错值和类型。
132      * @param map 用于存储验证码的额外信息，比如容错值和类型。
133      * @return 总是返回 true，表示数据总是有效的。具体的验证逻辑可能在后续扩展。
134      */
135     public boolean beforeGenerateImageCaptchaValidData(ImageCaptchaInfo
    imageCaptchaInfo, Map<String, Object> map) {
136         // 定义容错值
137         Float tolerant = imageCaptchaInfo.getTolerant();
138         // 如果容错值非空且大于0，则将其添加到map中
139         if (tolerant != null && tolerant > 0) {
140             map.put(TOLERANT_KEY, tolerant);
141         }
142         // 获取验证码类型，如果为空则默认为SLIDER类型
143         String type = imageCaptchaInfo.getType();
144         if (ObjectUtils.isEmpty(type)) {
145             type = CaptchaTypeConstant.SLIDER;
146         }
147         // 将验证码类型添加到map中
148         map.put(TYPE_KEY, type);
149         return true;
150     }
151 
152 
153     public void afterGenerateImageCaptchaValidData(ImageCaptchaInfo
    imageCaptchaInfo, Map<String, Object> map) {
154 
155     }
156 
157 
158     /**
159      * 生成图像验证码的有效数据。
160      * 根据传入的验证码类型和信息，计算并更新验证码相关的数据。
161      *
162      * @param map 用于存储验证码配置和计算结果的键值对映射。
163      * @param imageCaptchaInfo 包含验证码图像和扩展数据的对象。
164      *
165      * 注意：该方法不修改传入参数的原始数据，而是通过map参数来存储和传递计算结果。
166      */
167     public void doGenerateImageCaptchaValidData(Map<String, Object> map,
```

```
168                                                 ImageCaptchaInfo
imageCaptchaInfo) {
169         // 获取验证码类型，默认为滑动验证码
170         String type = (String) map.getOrDefault(TYPE_KEY,
CaptchaTypeConstant.SLIDER);
171         Object expand = imageCaptchaInfo.getData() == null ? null :
imageCaptchaInfo.getData().getExpand();
172
173         if (CaptchaUtils.isSliderCaptcha(type)) {
174             // 处理滑动验证码
175             addPercentage(imageCaptchaInfo, map);
176         } else if (CaptchaUtils.isClickCaptcha(type)) {
177             // 处理图片点选验证码
178             if (expand == null) {
179                 throw new IllegalArgumentException("点选验证码扩展数据转换为
List<ClickImageCheckDefinition> 失败， info=" + imageCaptchaInfo);
180             }
181             List<ClickImageCheckDefinition> clickImageCheckDefinitionList;
182             try {
183                 clickImageCheckDefinitionList =
(List<ClickImageCheckDefinition>) expand;
184             } catch (Exception e) {
185                 throw new IllegalArgumentException("点选验证码扩展数据转换为
List<ClickImageCheckDefinition> 失败， info=" + imageCaptchaInfo);
186             }
187             StringBuilder sb = new StringBuilder();
188             for (int i = 0; i < clickImageCheckDefinitionList.size(); i++) {
189                 // 计算每个选中区域的相对位置
190                 ClickImageCheckDefinition definition =
clickImageCheckDefinitionList.get(i);
191                 Integer x = definition.getX();
192                 Integer y = definition.getY();
193                 Integer width = imageCaptchaInfo.getBackgroundImageWidth();
194                 Integer height = imageCaptchaInfo.getBackgroundImageHeight();
195                 float vx = calcPercentage(x, width);
196                 float vy = calcPercentage(y, height);
197                 sb.append(vx).append(",").append(vy).append(";");
198                 // 计算并更新首次遇到的选中区域的容错值
199                 if (i == 0 && !map.containsKey(TOLERANT_KEY)) {
200                     float minLeft = calcPercentage(x - definition.getWidth() /
2f, width);
201                     float tolerant = vx - minLeft;
202                     map.put(TOLERANT_KEY, tolerant);
203                 }
204             }
205             // 更新map中存储的点选数据
206             map.put(PERCENTAGE_KEY, sb.toString());
```

```
207         } else if (CaptchaUtils.isJigsawCaptcha(type)) {
208             // 处理拼图验证码，直接使用扩展数据
209             map.put(PERCENTAGE_KEY, expand);
210         }
211     }
212
213
214     /**
215      * 验证图像验证码是否有效。
216      *
217      * @param imageCaptchaTrack 包含验证码的追踪信息，如背景图片宽度和滑动轨迹。
218      * @param imageCaptchaValidData 包含验证码的验证数据，如容错值和验证码类型。
219      * @return ApiResponse<?> 包含验证结果的响应对象，成功时携带验证相关信息，失败时携
    带错误信息。
220      */
221     @Override
222     public ApiResponse<?> valid(ImageCaptchaTrack imageCaptchaTrack,
    Map<String, Object> imageCaptchaValidData) {
223         // 读取并解析容错值和验证码类型
224         Float tolerant = getFloatParam(TOLERANT_KEY, imageCaptchaValidData,
    defaultTolerant);
225         String type = getStringParam(TYPE_KEY, imageCaptchaValidData,
    CaptchaTypeConstant.SLIDER);
226
227         // 执行验证前的检查
228         ApiResponse<?> beforeValid = beforeValid(imageCaptchaTrack,
    imageCaptchaValidData, tolerant, type);
229         if (!beforeValid.isSuccess()) {
230             return beforeValid; // 如果验证前检查失败，则直接返回错误响应
231         }
232
233         // 检查背景图片宽度是否合法
234         Integer bgImageWidth = imageCaptchaTrack.getBgImageWidth();
235         if (bgImageWidth == null || bgImageWidth < 1) {
236             return ApiResponse.ofCheckError("验证码背景图片宽度参数错误");
237         }
238
239         // 检查滑动轨迹是否存在
240         List<ImageCaptchaTrack.Track> trackList =
    imageCaptchaTrack.getTrackList();
241         if (CollectionUtils.isEmpty(trackList)) {
242             return ApiResponse.ofCheckError("没有解析到滑动轨迹");
243         }
244
245         // 执行验证码有效性验证
246         ApiResponse<?> response;
```

```
247         boolean valid = doValid(imageCaptchaTrack, imageCaptchaValidData,
    tolerant, type);
248         if (valid) {
249             // 验证成功，执行验证后的逻辑
250             response = afterValid(imageCaptchaTrack, imageCaptchaValidData,
    tolerant, type);
251         } else {
252             // 验证失败，返回基本的验证失败响应
253             response =
    ApiResponse.ofMessage(ApiResponseStatusConstant.BASIC_CHECK_FAIL);
254         }
255         return response;
256     }
257
258
259     /**
260      * 验证前
261      *
262      * @param imageCaptchaTrack sliderCaptchaTrack
263      * @param captchaValidData  captchaValidData
264      * @param tolerant          tolerant
265      * @param type              type
266      * @return boolean
267      */
268     public ApiResponse<?> beforeValid(ImageCaptchaTrack imageCaptchaTrack,
    Map<String, Object> captchaValidData, Float tolerant, String type) {
269         return ApiResponse.ofSuccess();
270     }
271
272     /**
273      * 验证后
274      *
275      * @param imageCaptchaTrack sliderCaptchaTrack
276      * @param captchaValidData  captchaValidData
277      * @param tolerant          tolerant
278      * @param type              type
279      * @return boolean
280      */
281     public ApiResponse<?>  afterValid(ImageCaptchaTrack imageCaptchaTrack,
    Map<String, Object> captchaValidData, Float tolerant, String type) {
282         return ApiResponse.ofSuccess();
283     }
284
285     public boolean doValid(ImageCaptchaTrack imageCaptchaTrack,
286                            Map<String, Object> imageCaptchaValidData,
287                            Float tolerant,
288                            String type) {
```

```
289         if (CaptchaUtils.isSliderCaptcha(type)) {
290             // 滑动类型验证码
291             return doValidSliderCaptcha(imageCaptchaTrack,
    imageCaptchaValidData, tolerant, type);
292         } else if (CaptchaUtils.isClickCaptcha(type)) {
293             // 点选类型验证码
294             return doValidClickCaptcha(imageCaptchaTrack,
    imageCaptchaValidData, tolerant, type);
295         } else if (CaptchaUtils.isJigsawCaptcha(type)) {
296             // 拼图类型验证码
297             return doValidJigsawCaptcha(imageCaptchaTrack,
    imageCaptchaValidData, tolerant, type);
298         }
299         // 不支持的类型
300         log.warn("校验验证码警告， 不支持的验证码类型:{}, 请手动扩展
    cloud.tianai.captcha.validator.impl.SimpleImageCaptchaValidator.doValid 进行校验
    扩展", type);
301         return false;
302     }
303
304     public boolean doValidJigsawCaptcha(ImageCaptchaTrack imageCaptchaTrack,
    Map<String, Object> imageCaptchaValidData, Float tolerant, String type) {
305         if (imageCaptchaTrack.getData() == null || !
    (imageCaptchaTrack.getData() instanceof String)) {
306             throw new IllegalArgumentException("拼图验证码必须传data数据，且必须是
    字符串类型逗号分隔数据");
307         }
308         String posArr = (String) imageCaptchaTrack.getData();
309         String successPosStr = getStringParam(PERCENTAGE_KEY,
    imageCaptchaValidData, null);
310         return successPosStr.equals(posArr);
311     }
312
313     /**
314      * 校验点选验证码
315      *
316      * @param imageCaptchaTrack      sliderCaptchaTrack
317      * @param imageCaptchaValidData imageCaptchaValidData
318      * @param tolerant               tolerant
319      * @param type                   type
320      * @return boolean
321      */
322     public boolean doValidClickCaptcha(ImageCaptchaTrack imageCaptchaTrack,
323                                        Map<String, Object>
    imageCaptchaValidData,
324                                        Float tolerant,
325                                        String type) {
```

```
326         String validStr = getStringParam(PERCENTAGE_KEY,
    imageCaptchaValidData, null);
327         if (ObjectUtils.isEmpty(validStr)) {
328             return false;
329         }
330         String[] splitArr = validStr.split(";");
331         List<ImageCaptchaTrack.Track> trackList =
    imageCaptchaTrack.getTrackList();
332         if (trackList.size() < splitArr.length) {
333             return false;
334         }
335         // 取出点击事件的轨迹数据
336         List<ImageCaptchaTrack.Track> clickTrackList = trackList
337                 .stream()
338                 .filter(t ->
    TrackTypeConstant.CLICK.equalsIgnoreCase(t.getType()))
339                 .collect(Collectors.toList());
340         if (clickTrackList.size() != splitArr.length) {
341             return false;
342         }
343         StringBuilder sb = new StringBuilder();
344         List<Double> percentages = new ArrayList<>();
345         for (int i = 0; i < splitArr.length; i++) {
346             ImageCaptchaTrack.Track track = clickTrackList.get(i);
347             String posStr = splitArr[i];
348             String[] posArr = posStr.split(",");
349             float xPercentage = Float.parseFloat(posArr[0]);
350             float yPercentage = Float.parseFloat(posArr[1]);
351
352             float calcXPercentage = calcPercentage(track.getX(),
    imageCaptchaTrack.getBgImageWidth());
353             float calcYPercentage = calcPercentage(track.getY(),
    imageCaptchaTrack.getBgImageHeight());
354             if (!checkPercentage(calcXPercentage, xPercentage, tolerant)
355                     || !checkPercentage(calcYPercentage, yPercentage,
    tolerant)) {
356                 return false;
357             }
358             if (i > 0) {
359                 sb.append("|");
360             }
361             sb.append(calcXPercentage).append(",").append(calcYPercentage);
362             percentages.add((double) ((calcXPercentage - xPercentage) +
    (calcYPercentage - yPercentage)));
363         }
364         // 存储一下当前计算出来的值
365         imageCaptchaValidData.put(USER_CURRENT_PERCENTAGE, sb.toString());
```

```
366         imageCaptchaValidData.put(USER_CURRENT_PERCENTAGE_STD,
String.valueOf(NumPy.std(percentages)));
367         return true;
368     }
369
370     /**
371      * 校验滑动验证码
372      *
373      * @param imageCaptchaTrack      sliderCaptchaTrack
374      * @param imageCaptchaValidData imageCaptchaValidData
375      * @param tolerant               tolerant
376      * @param type                   type
377      * @return boolean
378      */
379     public boolean doValidSliderCaptcha(ImageCaptchaTrack imageCaptchaTrack,
380                                         Map<String, Object>
imageCaptchaValidData,
381                                         Float tolerant,
382                                         String type) {
383         Float oriPercentage = getFloatParam(PERCENTAGE_KEY,
imageCaptchaValidData);
384         if (oriPercentage == null) {
385             // 没读取到百分比
386             return false;
387         }
388         List<ImageCaptchaTrack.Track> trackList =
imageCaptchaTrack.getTrackList();
389         // 取最后一个滑动轨迹
390         ImageCaptchaTrack.Track lastTrack = trackList.get(trackList.size() -
1);
391         // 计算百分比
392         float calcPercentage = calcPercentage(lastTrack.getX(),
imageCaptchaTrack.getBgImageWidth());
393         // 校验百分比
394         boolean percentage = checkPercentage(calcPercentage, oriPercentage,
tolerant);
395         if (percentage) {
396             // 校验成功
397             // 存储一下当前计算出来的值
398             imageCaptchaValidData.put(USER_CURRENT_PERCENTAGE,
String.valueOf(calcPercentage));
399             imageCaptchaValidData.put(USER_CURRENT_PERCENTAGE_STD,
String.valueOf(calcPercentage - oriPercentage));
400         }
401         return percentage;
402     }
403
```

```
404     public Float getFloatParam(String key, Map<String, Object>
    imageCaptchaValidData) {
405         return getFloatParam(key, imageCaptchaValidData, null);
406     }
407 
408     public Float getFloatParam(String key, Map<String, Object>
    imageCaptchaValidData, Float defaultData) {
409         Object data = imageCaptchaValidData.get(key);
410         if (data != null) {
411             if (data instanceof Number) {
412                 return ((Number) data).floatValue();
413             }
414             try {
415                 if (data instanceof String) {
416                     return Float.parseFloat((String) data);
417                 }
418             } catch (NumberFormatException e) {
419                 log.error("从 imageCaptchaValidData 读取到的 " + key + "无法转换成
    float类型, [{}]", data);
420                 throw e;
421             }
422         }
423         return defaultData;
424     }
425 
426     public String getStringParam(String key, Map<String, Object>
    imageCaptchaValidData, String defaultData) {
427         if (CollectionUtils.isEmpty(imageCaptchaValidData)) {
428             return defaultData;
429         }
430         Object data = imageCaptchaValidData.get(key);
431         if (data != null) {
432             if (data instanceof String) {
433                 return (String) data;
434             }
435             try {
436                 return String.valueOf(data);
437             } catch (NumberFormatException e) {
438                 log.error("从 imageCaptchaValidData 读取到的 " + key + "无法转换成
    String类型, [{}]", data);
439                 throw e;
440             }
441         }
442         return defaultData;
443     }
444 
```

```
    protected void addPercentage(ImageCaptchaInfo imageCaptchaInfo, Map<String, Object> imageCaptchaValidData) {
        float percentage = calcPercentage(imageCaptchaInfo.getRandomX(), imageCaptchaInfo.getBackgroundImageWidth());
        imageCaptchaValidData.put(PERCENTAGE_KEY, percentage);
    }
}
```

行为轨迹校验（平台自己扩展）

这里只进行基本检测, 用一些简单算法进行校验，如有需要可扩展

- 检测1: 滑动时间如果小于300毫秒 返回false
- 检测2: 轨迹数据要是少于背10，或者大于背景宽度的五倍 返回false
- 检测3: x轴和y轴应该是从0开始的，要是一开始x轴和y轴乱跑，返回false
- 检测4: 如果y轴是相同的，必然是机器操作，直接返回false
- 检测5x轴或者y轴之间的区间跳跃过大的话返回 false
- 检测6: x轴应该是由快到慢的， 要是速率一致，返回false
- 检测7: 如果x轴超过图片宽度的频率过高，返回false

```
/**
 * 滑动验证码验证后的处理方法，用于判断用户滑动轨迹是否符合预期的人工操作。
 * 方法会检查滑动轨迹的多个方面，如滑动时间、轨迹点数量、起始位置、轴向一致性等，以确保验证结果的有效性。
 *
 * @param imageCaptchaTrack ImageCaptchaTrack对象，包含滑动验证码的详细信息，如滑动开始和结束时间、背景宽度、轨迹列表等。
 * @param captchaValidData 验证数据，目前未使用，可能用于存储额外的验证信息。
 * @param tolerant 验证宽容度，目前未使用，可能用于调整验证的严格程度。
 * @param type 验证码类型，用于判断是否为滑动验证码。
 *
 * @return ApiResponse<?> 响应对象，成功时返回空对象，失败时包含错误定义。
 */
@Override
public ApiResponse<?> afterValid(ImageCaptchaTrack imageCaptchaTrack, Map<String, Object> captchaValidData, Float tolerant, String type) {
    if (!CaptchaUtils.isSliderCaptcha(type)){
        // 检查验证码类型，如果不是滑动验证码，则直接返回成功响应
        return ApiResponse.ofSuccess();
    }

    // 获取滑动开始和结束时间，以及背景宽度和轨迹列表
    long startSlidingTime = imageCaptchaTrack.getStartSlidingTime().getTime();
```

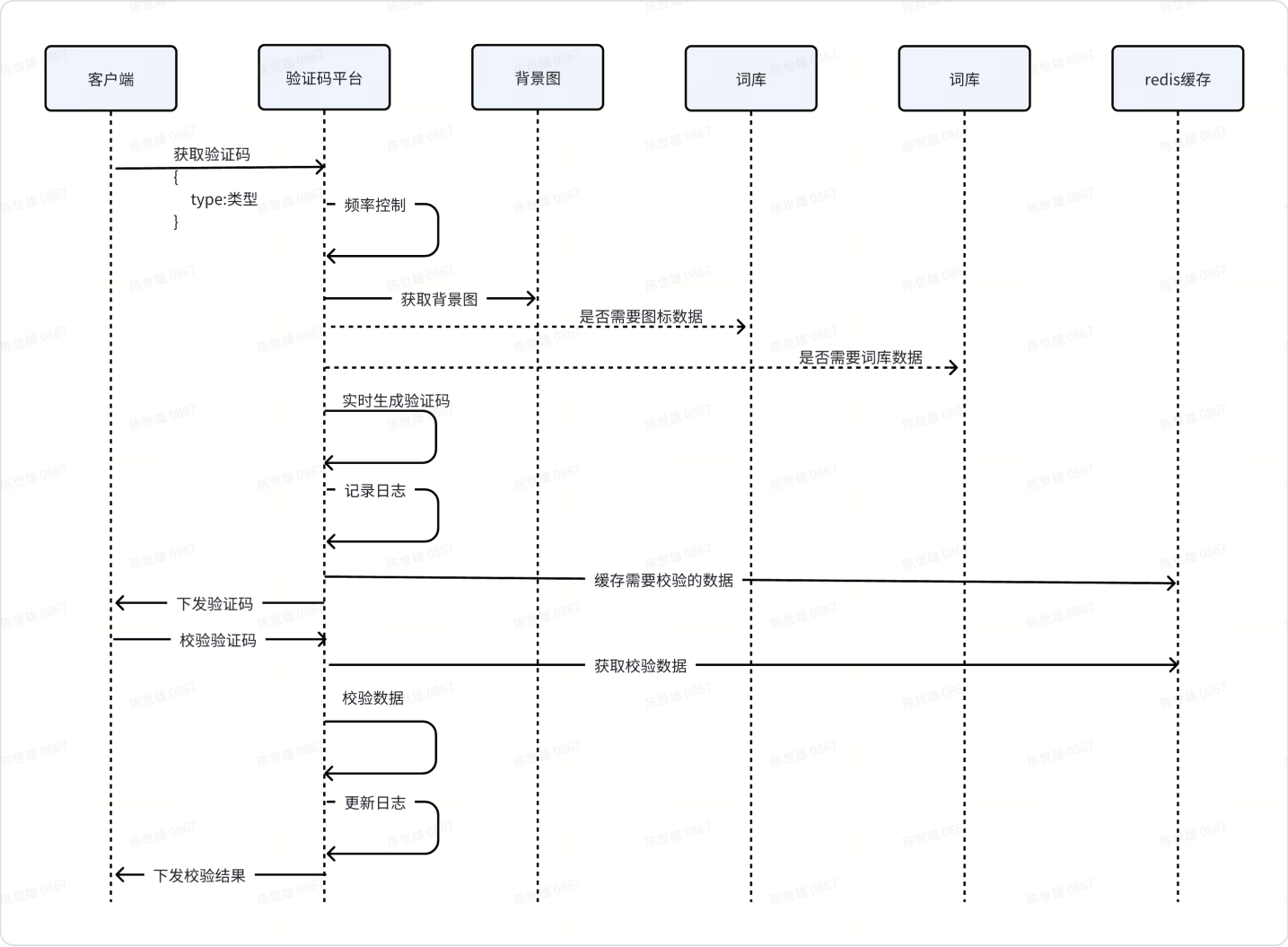
```
    long endSlidingTime = imageCaptchaTrack.getEndSlidingTime().getTime();
    Integer bgImageWidth = imageCaptchaTrack.getBgImageWidth();
    List<ImageCaptchaTrack.Track> trackList = imageCaptchaTrack.getTrackList();

    // 实施一系列滑动轨迹验证规则
    // 规则包括：滑动时间、轨迹点数量、起始位置、轴向一致性、轴向跳跃和速度等

    // 规则1：滑动时间不应少于300毫秒
    if (startSlidingTime + 300 > endSlidingTime) {
        return ApiResponse.ofMessage(DEFINITION); // 返回失败，时间过短
    }

    // 规则2：轨迹点数量应在10到背景宽度的5倍之间
    if (trackList.size() < 10 || trackList.size() > bgImageWidth * 5) {
        return ApiResponse.ofMessage(DEFINITION); // 返回失败，数量异常
    }

    // 规则3：起始点应接近(0, 0)
    ImageCaptchaTrack.Track firstTrack = trackList.get(0);
    if (firstTrack.getX() > 10 || firstTrack.getX() < -10 || firstTrack.getY()
> 10 || firstTrack.getY() < -10) {
        return ApiResponse.ofMessage(DEFINITION); // 返回失败，起始位置偏离过大
    }

    int check4 = 0; // 计数器，用于检测所有点的y轴是否相同
    int check7 = 0; // 计数器，用于检测x轴超出图片宽度的次数

    for (int i = 1; i < trackList.size(); i++) {
        ImageCaptchaTrack.Track track = trackList.get(i);
        float x = track.getX();
        float y = track.getY();

        // 规则4：所有点的y轴应该不相同
        if (firstTrack.getY() == y) {
            check4++;
        }

        // 规则7：x轴超出图片宽度的频率不能过高
        if (x >= bgImageWidth) {
            check7++;
        }

        // 规则5：相邻点之间的轴向跳跃不能过大
        ImageCaptchaTrack.Track preTrack = trackList.get(i - 1);
        if ((track.getX() - preTrack.getX()) > 50 || (track.getY() -
preTrack.getY()) > 50) {
            return ApiResponse.ofMessage(DEFINITION); // 返回失败，跳跃过大
```

```
        }
    }

    // 规则4和规则7的结果判断
    if (check4 == trackList.size() || check7 > 200) {
        return ApiResponse.ofMessage(DEFINITION); // 返回失败，不符合预期
    }

    // 规则6：x轴速度应从快到慢
    int splitPos = (int) (trackList.size() * 0.7);
    ImageCaptchaTrack.Track splitPostTrack = trackList.get(splitPos - 1);
    float posTime = splitPostTrack.getT();
    float startAvgPosTime = posTime / (float) splitPos;

    ImageCaptchaTrack.Track lastTrack = trackList.get(trackList.size() - 1);
    double endAvgPosTime = lastTrack.getT() / (float) (trackList.size() -
splitPos);

    boolean check = endAvgPosTime > startAvgPosTime;
    if (check) {
        return ApiResponse.ofSuccess(); // 成功，符合预期
    }
    return ApiResponse.ofMessage(DEFINITION); // 失败，不符合预期
}

```

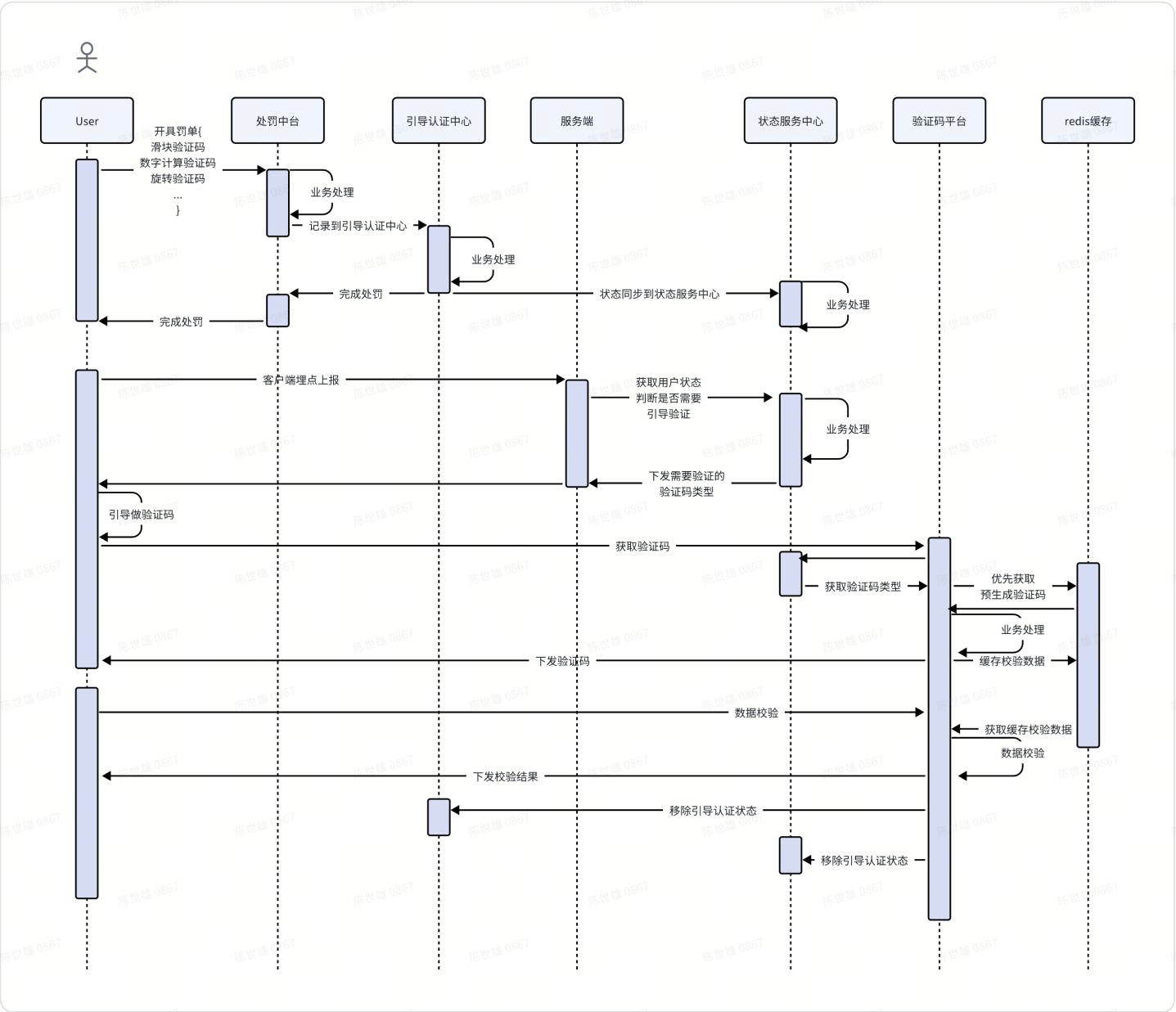
验证码-业务逻辑

模块	功能	类型	路径
行为验证码	获取验证码	新增	/behaviorCaptcha/getCaptcha
	校验验证码	新增	/behaviorCaptcha/validate
验证码日志	获取验证码日志列表	新增	/behaviorCaptchaLog/pageQuer
验证码管理	分页获取图片配置	新增	risk-punish/v1/captchaConfig/pa
	批量新增图片配置	新增	risk-punish/v1/captchaConfig/ba
	按ID变更图片配置	新增	risk-punish/v1/captchaConfig/up
	获取验证码配置	新增	risk-punish/v1/captchaConfig/ge
	保存或更新验证码配置	新增	risk-punish/v1/captchaConfig/saveOrUpdateCaptchaConfig
	分页获取词组配置	新增	risk-punish/v1/captchaConfig/pa
	批量新增词组配置	新增	risk-punish/v1/captchaConfig/ba
	按ID变更词组配置	新增	risk-punish/v1/captchaConfig/up

验证码获取/校验流程



验证码下发流程



重要接口（参考）

```
{
    "ticketId": "vaoshurg9q8",
    "params": {
        "type": "rotate",
        "answer": [
            "24",
            "13"
        ],
        "backgroundWidth": 300,
        "backgroundHeight": 160,
        "offset": "13.20",
        "tracks": [
            {
```

```
                "x": 0,
                "y": 0
            }
        ]
    }
}
// 检验的返回内容
{
    "msg": "success",
    "code": "0",
    "data": {
        "tips":"完成校验",
        "code":0, //0: 完成校验 1: 验证错误 2:验证超时 3:疑似人机 4:其他错误
        "time":10 //单位秒
    }
    "success": true,
    "extra": ""
}
```

```

{
    "ticketId":"vaoshurg9q8"
}
// 根据用户下方验证码内容
{
    "msg": "",
    "code": "0",
    "data": {

        "pictureId":1693505658135121920,
        //背景图base64,List是为了兼容后续可能添加的图选验证码，目前只会有一张背景图
        "backgroundImages":["",
""]
        //背景图宽度
        "backgroundWidth": 300,
        //背景图高度
        "backgroundHeight": 160,
        //滑块图 、文字点选/旋转图的base64 旋转验证码默认是在中心点位
        "sliderImage": "",
        "sliderWidth": 100,
        "sliderHeight": 100,
        "type": "number",
```

```
        //下发的问题
        "issue": "拖动滑块完成拼图",
        //点选类才有值，需要点击的次数
        "numberOfClicks": 4,
        //滑动还原的滑动y轴点位
        "y": 84
    },
    "success": true,
    "extra": ""
}

sliderRestore：滑动还原验证码
textClick：文字点选验证码
wordOrderClick：语序点选验证码
rotate：旋转验证码
number：数字计算验证码
```