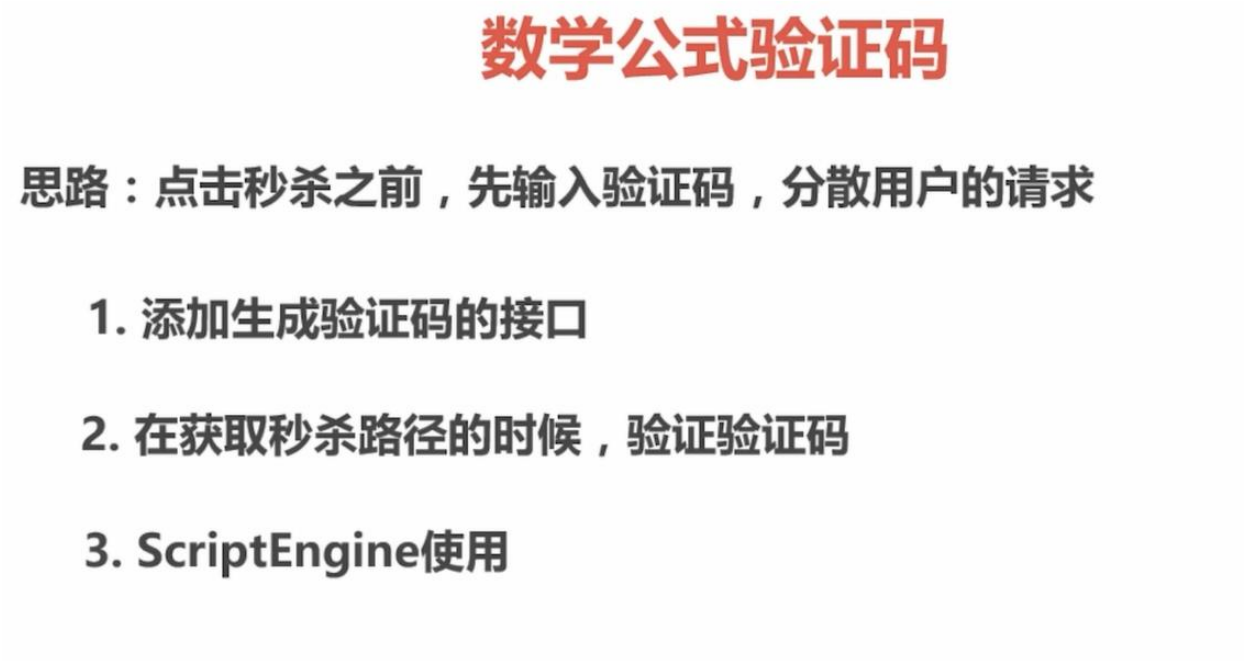
第七章 安全优化

1. 秒杀接口地址隐藏
2. 数学公式验证码
3. 接口限流防刷

秒杀接口地址隐藏

思路：秒杀开始之前，先去请求接口获取秒杀地址

1. 接口改造，带上PathVariable参数
2. 添加生成地址的接口
3. 秒杀收到请求，先验证PathVariable



秒杀系统的相关设计思路以及面试问答：https://www.jianshu.com/p/06479ea2dc46

1、秒杀接口隐藏

实现手段：在前端页面做秒杀请求的时候不是一下就访问秒杀接口，而是请求一个秒杀的接口URI，path由后端根据user和goods生成并且存在redis中，返回path给前端然后前端拼接成完整uri发送请求进行秒杀，同时秒杀controller也会检查path的正确性，如果不正确会返回非法请求。

2、验证码

后端代码生成验证码并且存redis中，请求秒杀接口uri时会验证验证码，如果不正确就不返回path给前端，并且返回验证码错误。

3、接口限流

为了让服务器能抗住压力，需要做限流，让用户对该秒杀请求在一定时间内有访问次数，访问过多就返回访问太过频繁不给访问，过段时间才给访问。

实现手段是写一个拦截器，拦截请求，如果超过访问次数就不放行。并且用自定义注解来注解接口。

注解类：

```
@Retention(RUNTIME)
@Target(METHOD)
public @interface AccessLimit {
  int seconds();
  int maxCount();
  boolean needLogin() default true;
}
```

拦截器：

```
@Service
public class AccessInterceptor  extends HandlerInterceptorAdapter{

  @Autowired
  MiaoshaUserService userService;

  @Autowired
  RedisService redisService;

  @Override
  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
      throws Exception {
    if(handler instanceof HandlerMethod) {
      MiaoshaUser user = getUser(request, response);
      UserContext.setUser(user);
      HandlerMethod hm = (HandlerMethod)handler;
      AccessLimit accessLimit = hm.getMethodAnnotation(AccessLimit.class);
      if(accessLimit == null) {
        return true;
      }
      int seconds = accessLimit.seconds();
      int maxCount = accessLimit.maxCount();
      boolean needLogin = accessLimit.needLogin();
      String key = request.getRequestURI();
      if(needLogin) {
        if(user == null) {
          render(response, CodeMsg.SESSION_ERROR);
```

```
            return false;
        }
        key += "_" + user.getId();
      }else {
        //do nothing
      }
      AccessKey ak = AccessKey.withExpire(seconds);
      Integer count = redisService.get(ak, key, Integer.class);
       if(count  == null) {
           redisService.set(ak, key, 1);
       }else if(count < maxCount) {
           redisService.incr(ak, key);
       }else {
          render(response, CodeMsg.ACCESS_LIMIT_REACHED);
          return false;
       }
    }
    return true;
}

private void render(HttpServletResponse response, CodeMsg cm)throws Exception {
    response.setContentType("application/json;charset=UTF-8");
    OutputStream out = response.getOutputStream();
    String str  = JSON.toJSONString(Result.error(cm));
    out.write(str.getBytes("UTF-8"));
    out.flush();
    out.close();
}

private MiaoshaUser getUser(HttpServletRequest request, HttpServletResponse response) {
    String paramToken = request.getParameter(MiaoshaUserService.COOKI_NAME_TOKEN);
    String cookieToken = getCookieValue(request, MiaoshaUserService.COOKI_NAME_TOKEN);
    if(StringUtils.isEmpty(cookieToken) && StringUtils.isEmpty(paramToken)) {
      return null;
    }
    String token = StringUtils.isEmpty(paramToken)?cookieToken:paramToken;
    return userService.getByToken(response, token);
}

private String getCookieValue(HttpServletRequest request, String cookiName) {
    Cookie[]  cookies = request.getCookies();
```

```
        if(cookies == null || cookies.length <= 0){
            return null;
        }
        for(Cookie cookie : cookies) {
            if(cookie.getName().equals(cookiName)) {
                return cookie.getValue();
            }
        }
        return null;
    }

}
```

mvc注入时需要获取用户的登录信息即MiaoshaUser，可以在拦截器完成用户信息的封装，并且将MiaoshaUser对象放入ThreadLocal（是一个本地线程副本变量工具类）,用户的每个请求都会给出独立的线程，各线程中的变量不互相干扰，我们把用户对象放入其中，自定义的mvc注入UserArgumentResolver 类可以借此得到对象。

```
public class UserContext {

    private static ThreadLocal<MiaoshaUser> userHolder = new ThreadLocal<MiaoshaUser>();

    public static void setUser(MiaoshaUser user) {
        userHolder.set(user);
    }

    public static MiaoshaUser getUser() {
        return userHolder.get();
    }

}
```

最后在接口函数上加上注解即可

```
@AccessLimit(seconds=5, maxCount=5, needLogin=true)
@GetMapping("/path")
@ResponseBody
public Result<String> getMiaoshaPath(HttpServletRequest request, MiaoshaUser user,
```

5秒内可以访问5次，需要登录。