一 登陆认证拦截器

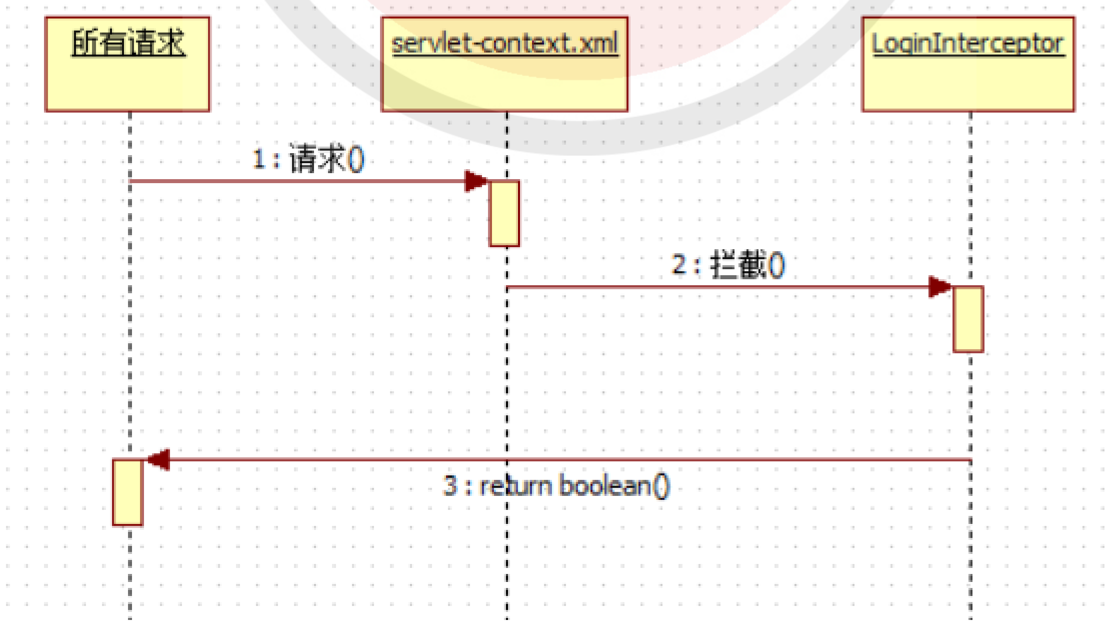
1 需求分析

根据用户信息存储规则，通过拦截器拦截用户操作请求获取用户cookie信息,获取cookie 中用户id 用户记录存在性校验, 校验失败，抛出自定义异常信息,编写用户自定义异常类，捕获系统异常信息并对自定义异常做相应页面响应处理。,servlet-context.xml配置自定义 异常与登录拦截器，实现用户登录请求拦截校验功能

2 servlet-context.xml

```xml
        <mvc:interceptors>
            <mvc:interceptor>
                <mvc:mapping path="/**"/>
                <mvc:exclude-mapping path="/index"/>
                <mvc:exclude-mapping path="/user/userLogin"/>
                <mvc:exclude-mapping path="/css/**"/>
                <mvc:exclude-mapping path="/jquery-easyui-1.3.3/**"/>
                <mvc:exclude-mapping path="/images/**"/>
                <mvc:exclude-mapping path="/js/**"/>
                <bean class="com.xxx.crm.interceptors.LoginInterceptor"/>
            </mvc:interceptor>
        </mvc:interceptors>
```

3 流程图



4 编写LoginInterceptor

```
          public class LoginInterceptor  extends HandlerInterceptorAdapter{
            @Resource
            public UserService userService;

            @Override
            public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
                String userId= CookieUtil.getCookieValue(request,"userId");
                AssertUtil.isTrue(null==userId,"用户未登录");
                userId=Base64Util.decode(userId);
                User user = userService.queryUserById(userId);
                AssertUtil.isTrue(null==user,"用户不存在");
                return true;//全局异常
            }
        }
```

二 定义全局异常

1 需求分析

定义全局异常说明明确，能够快速定位问题，统一代码业务性问题处理规范，方便错误数据的封装，后台数据机构的统一（统一异常拦截体现）；

2 编写GlobalExceptionResolver

```
@Component
public class GlobalException implements HandlerExceptionResolver {
    /**
     * 视图异常, return ModelAndView
     * json异常有ResponseBody return null; MessageModel httpServletResponse写出去
     *
     * 1 是否是未登录异常
     * 2 json异常
     * 3 视图异常
     * @param httpServletRequest
     * @param httpServletResponse
     * @param handler
     * @param e
     * @return
     */
    @Override
    public ModelAndView resolveException(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, Object handler, Exception e) {
        ModelAndView modelAndView = createDefaultModelAndView(httpServletRequest);
        ParamsException paramsException;
        if (handler instanceof HandlerMethod){
            //1 先来判断一下是否是未登录的异常
            if (e instanceof ParamsException){
                paramsException = (ParamsException) e;
                if(paramsException.getCode()==CrmConstant.LOGIN_NO_CODE){//现在的异常是未登录
的异常
                    modelAndView.addObject("code",paramsException.getCode());
                    modelAndView.addObject("msg",paramsException.getMsg());
                    return modelAndView;
                }
            }
            //2 json异常
            HandlerMethod handlerMethod = (HandlerMethod) handler;
            Method method = handlerMethod.getMethod();
            ResponseBody responseBody = method.getAnnotation(ResponseBody.class);
            if (null!=responseBody){//现在存在re json异常 返回messageModel
                MessageModel messageModel = new MessageModel();
                messageModel.setMsg(CrmConstant.OPS_FAILED_MSG);
                messageModel.setCode(CrmConstant.OPS_FAILED_CODE);
                if (e instanceof ParamsException){
                    paramsException= (ParamsException) e;
                    messageModel.setCode(paramsException.getCode());
                    messageModel.setMsg(paramsException.getMsg());
                }
                httpServletResponse.setContentType("application/json;charset=uft-8");
                httpServletResponse.setCharacterEncoding("utf-8");
                PrintWriter printWriter = null;
                try {
                    printWriter = httpServletResponse.getWriter();

                } catch (IOException ioE) {
                    ioE.printStackTrace();
                }finally {
                    if(printWriter!=null){
                        printWriter.write(JSON.toJSONString(messageModel));
                        printWriter.flush();
                        printWriter.close();
                    }
                }
                return  null;
            }else {//3视图异常
                if (e instanceof ParamsException){
                    paramsException= (ParamsException) e;
                    modelAndView.addObject("code",paramsException.getCode());
                    modelAndView.addObject("msg",paramsException.getMsg());
                    return modelAndView;
                }else {
                    return modelAndView;
                }
            }
        }
        return null;
```

```
    }

    public  static  ModelAndView createDefaultModelAndView(HttpServletRequest request){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("error");
        modelAndView.addObject("code", CrmConstant.OPS_FAILED_CODE);
        modelAndView.addObject("msg",CrmConstant.OPS_FAILED_MSG);
        modelAndView.addObject("ctx",request.getContextPath());//为了防止拦截器没有放行，没有走
controller，所以没有ctx
        modelAndView.addObject("uri",request.getRequestURI());
        return modelAndView;
    }
}
```

3 前台error

```
        <#include "common.ftl" >
        <script>
            $(function () {
                alert("${msg}")

                if("${uri}"=="/main"){
                    window.location.href=ctx + "/index";
                }else{
                    window.parent.location.href = ctx + "/index";
                }
            })
        </script>
```