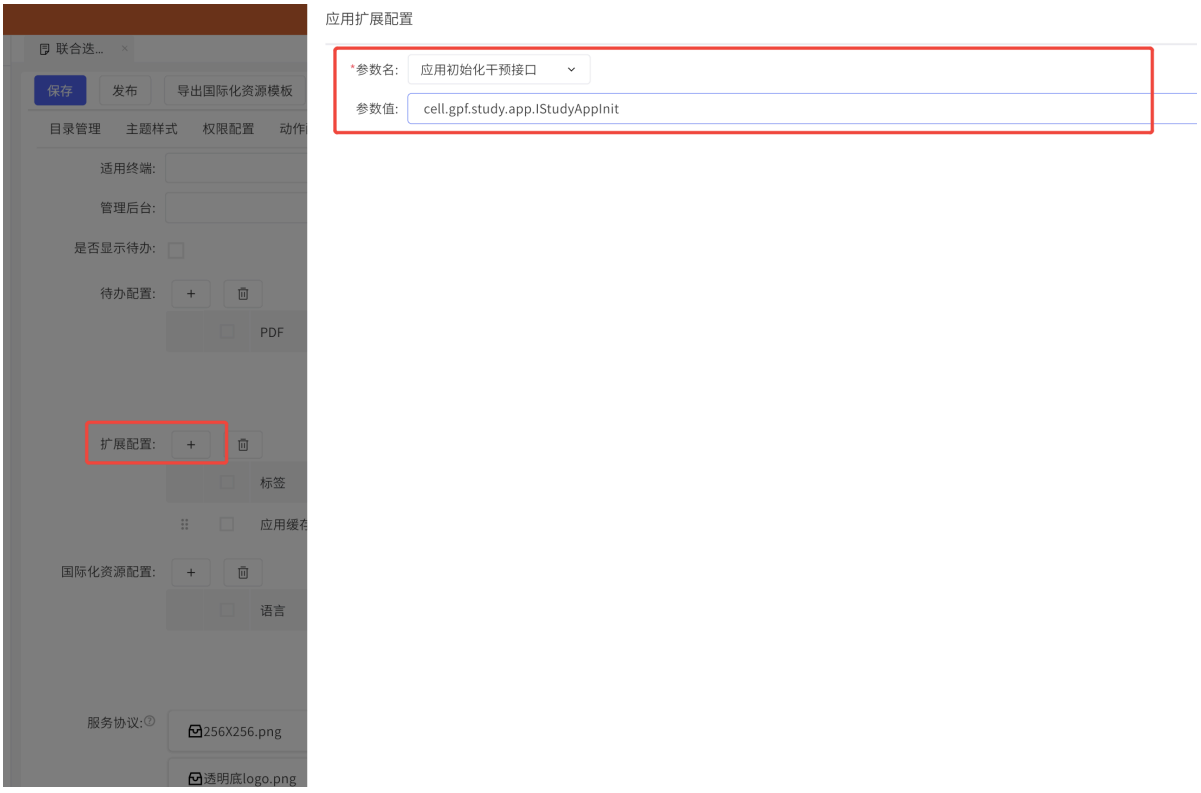
GPF应用干预

1.应用初始化干预接口



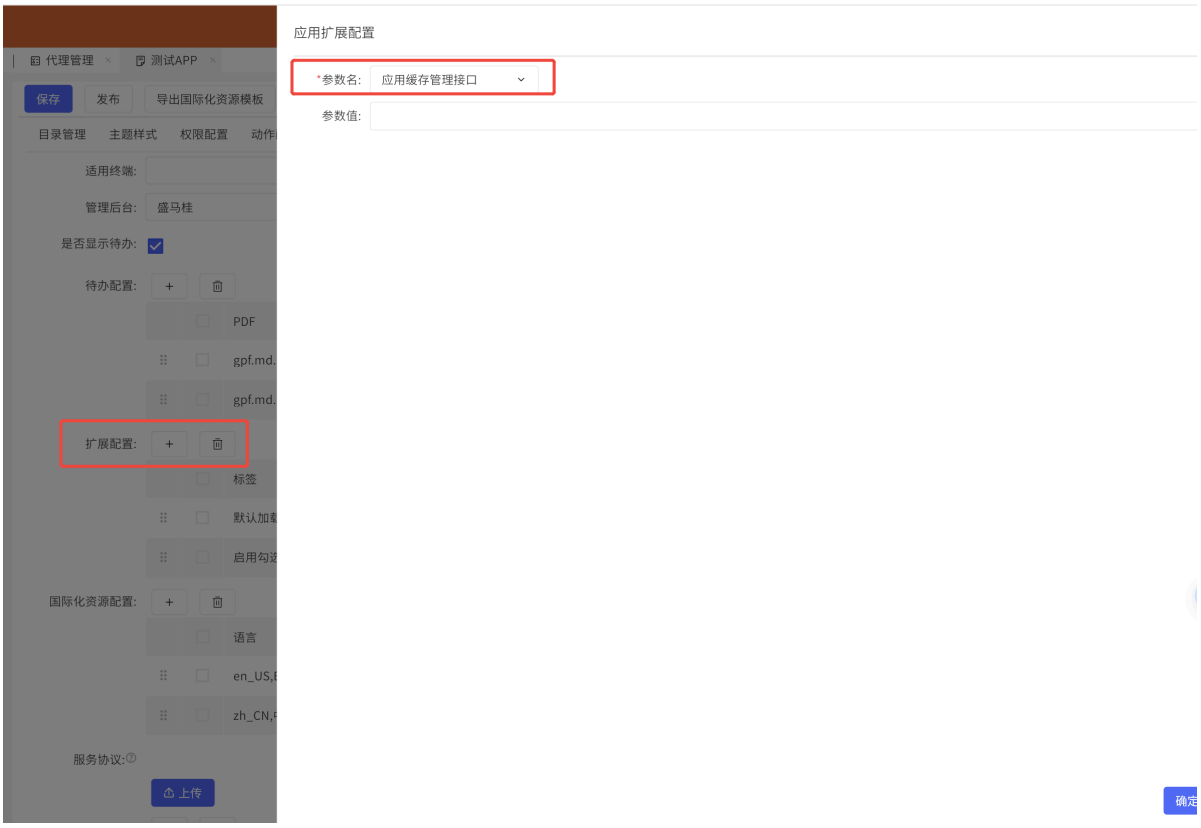
AppInitIntf的代码实现样例如下:

```
package cell.gpf.study.app;

import cell.CellIntf;
import cmn.anotation.ClassDeclare;
import cmn.util.TraceUtil;
import cmn.util.Tracer;
import fe.cmn.app.AppDto;
import fe.cmn.app.Context;
import fe.cmn.app.FontFamilyDto;
import fe.cmn.data.LocaleDto;
import fe.cmn.data.LocaleSettingsDto;
import fe.cmn.panel.PanelBuilder;
import fe.cmn.panel.PanelContext;
import fe.cmn.panel.PanelDto;
import gpf.dc.basic.fe.intf.AppInitIntf;
import gpf.dc.basic.param.view.dto.ApplicationSetting;
@ClassDeclare(label = "应用初始化干预接口实现样例"
,what="应用初始化干预接口实现样例"
, why = ""
, how = ""
,developer="陈晓斌"
,version = "1.0"
,createTime = "205-02-10"
,updateTime = "205-02-10")
public interface IStudyAppInit extends CellIntf,AppInitIntf{
```

```
    @Override
    default ApplicationSetting afterQueryApplicatinSetting(Context context,
ApplicationSetting setting)
            throws Exception {
        // 在查询后的应用配置后干预应用配置缓存
        return setting;
    }

    @Override
    default AppDto afterInitApp(Context context, AppDto appDto) throws Exception
{
        //可对应用配置初始化干预，例如
        // 设置语言对应的显示字体
        LocaleSettingsDto zhSettings = new LocaleSettingsDto()
                .setLocale(new LocaleDto("zh"))
                .setFontFamily(new FontFamilyDto("Zhi Mang Xing",
"https://kwaidoo.com/cdn-flutter/fonts/example/ZhiMangXing-Regular.ttf"));

        LocaleSettingsDto enSettings = new LocaleSettingsDto()
                .setLocale(new LocaleDto("en"))
                .setFontFamily(new FontFamilyDto("playwrite-ie-guides-regular",
"https://kwaidoo.com/cdn-flutter/fonts/example/PlaywriteIN-
VariableFont_wght.ttf"));

        LocaleSettingsDto esSettings = new LocaleSettingsDto()
                .setLocale(new LocaleDto("es"))
                .setFontFamily(new FontFamilyDto("BebasNeue-Regular",
"https://kwaidoo.com/cdn-flutter/fonts/example/BebasNeue-Regular.ttf"));

        appDto.setLocaleSettings(zhSettings, enSettings, esSettings);
        Tracer tracer = TraceUtil.getCurrentTracer();
        tracer.info("应用初始化干预~~");
        System.out.println("应用初始化干预~~");
        return appDto;
    }

    @Override
    default void beforeBuildHomePage(PanelBuilder builder, PanelContext context)
throws Exception {
        //在构建首页界面前干预应用缓存
    }

    @Override
    default PanelDto afterBuildHomePage(Context context, PanelDto homePage)
throws Exception {
        //在构建首页界面完成后干预
        return homePage;
    }
}
```

2.应用缓存干预接口

GPF应用的信道缓存可通过扩展缓存管理接口进行干预，可在页面初始化时添加新的应用缓存，或对应用配置缓存进行干预

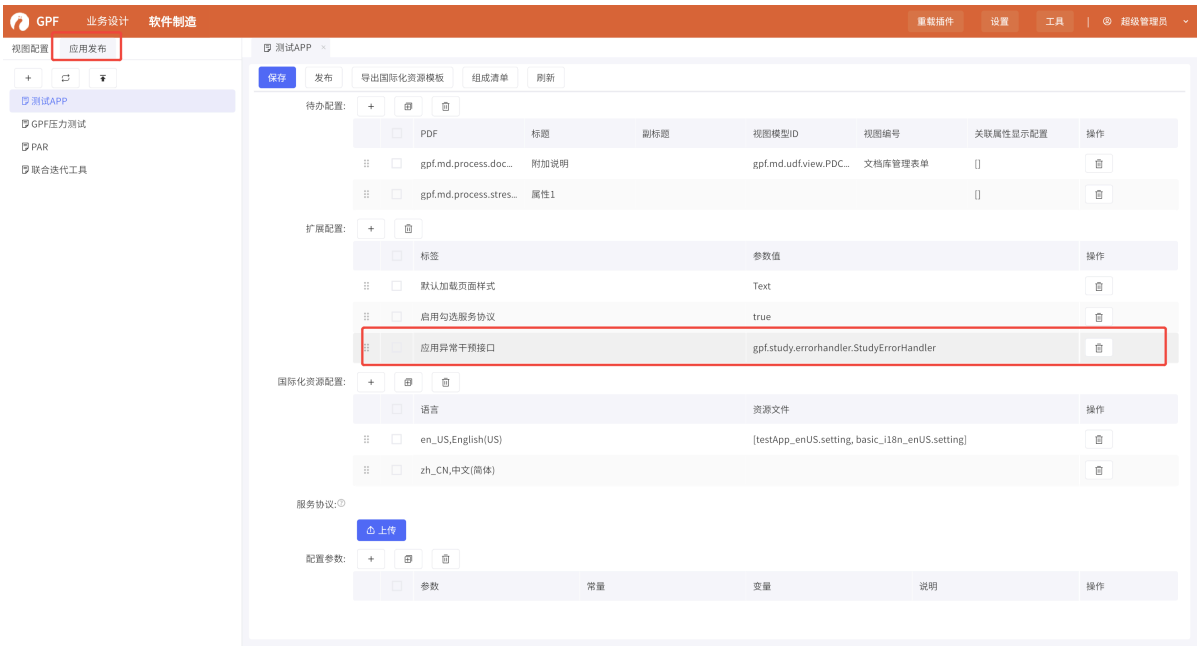


AppCacheMgrIntf的代码实现样例如下：

```
package cell.gpf.study.app;

import cell.CellIntf;
import fe.cmn.app.Context;
import gpf.dc.basic.fe.intf.AppCacheMgrIntf;
import gpf.dc.basic.param.view.dto.ApplicationSetting;

public interface IStudyAppCacheMgr extends CellIntf,AppCacheMgrIntf{


    @Override
    default void initCache(Context context) throws Exception {
        //在应用加载时初始化应用缓存
//      setCacheValue(context, key, value);
    }

}
```

3.应用异常处理干预

通过异常处理干预，可干预最终显示到界面上的报错信息



实现cmn.exception.handler.ErrorHandler接口，示例如下：

```
package gpf.study.errorhandler;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import com.kwaidoo.ms.tool.ToolUtilities;
import com.leavay.dfc.gui.LvUtil;

import cmn.anotation.ClassDeclare;
import cmn.enums.ErrorLevel;
import cmn.exception.BaseException;
import cmn.exception.ErrorInfoInterface;
import cmn.exception.handler.ErrorHandler;
@ClassDeclare(label = "异常处理类代码样例"
,what="异常处理类代码样例，演示如何对服务抛出的异常进行干预包装成业务可以读懂的异常，以下定义了错误码枚举类示例，具体可根据实际项目需要，调整为模型管理配置的错误码和匹配规则"
, why = ""
, how = ""
,developer="陈晓斌"
,version = "1.0"
,createTime = "205-02-14"
,updateTime = "205-02-14")
public class StudyErrorHandler implements ErrorHandler{

    /**
     *
     */
    private static final long serialVersionUID = 7752892622107640444L;

    /**
     * 错误码枚举类定义示例
     * 带有错误级别、错误码、错误描述
     */
    public static enum StudyErrorInfo implements ErrorInfoInterface{

        ConnectionFail(ErrorLevel.ERROR,"ERROR_0001","数据库连接异常"),
```

```
        TableNotFound(ErrorLevel.INFO,"ERROR_0002","表不存在"),
        Unkown(ErrorLevel.WARN,"ERROR_9999","未知异常")
        ;

        String errorCode;
        ErrorLevel errorLevel;
        String errorMsg;
        private StudyErrorInfo(ErrorLevel level,String errorCode,String errorMsg)
{
            this.errorLevel = level;
            this.errorCode = errorCode;
            this.errorMsg = errorMsg;
        }

        @Override
        public String getErrorCode() {
            return errorCode;
        }

        @Override
        public ErrorLevel getErrorLevel() {
            return errorLevel;
        }

        @Override
        public String getErrorMsg() {
            return errorMsg;
        }

    }

    @Override
    public Throwable handle(Throwable exception) {
        LvUtil.trace("处理异常：" + exception);
        String message = exception.getMessage();
        LvUtil.trace("message：" + message);
        String exceptionStack = ToolUtilities.getFullExceptionStack(exception);
        LvUtil.trace("exceptionStack：" + exceptionStack);
        //如果是异常基类，可以选择是原封不动抛出，或者是重新转译后抛出
        if(exception instanceof BaseException) {
            return new StudyBizException(((BaseException)
exception).getErrorLevel(), ((BaseException) exception).getErrorCode(),
exception.getMessage(), exception.getCause());
        }
        if(find(exceptionStack,"PSQLException:(.+)timed out")) {
            return new
StudyBizException(StudyErrorInfo.ConnectionFail,exception);
        }else if(find(exceptionStack,"PSQLException: 错误: 关系 (.+) 不存在")) {
            return new StudyBizException(StudyErrorInfo.TableNotFound,exception);
        }else {
            return new StudyBizException(StudyErrorInfo.Unkown,exception);
        }
    }
    /**
     * 检测堆栈日志是否匹配正则
     * @param errorStack
```

```
     * @param regexStr
     * @return
     */
    public boolean find(String errorStack,String regexStr) {
        Pattern regex = Pattern.compile(regexStr, Pattern.CASE_INSENSITIVE |
Pattern.UNICODE_CASE | Pattern.DOTALL | Pattern.MULTILINE);
        Matcher regexMatcher = regex.matcher(errorStack);
        return regexMatcher.find();
    }

    public static void main(String[] args) {
        String errorStack = "org.postgresql.util.PSQLException: 错误: 关系 测试表 不
存在\r\n" +
                "    at
cell.gpf.study.errorhandler.IStudyErrorHandler.testErrorHande";
         String regexStr = "PSQLException: 错误: 关系 (.+) 不存在";
        System.out.println(new StudyErrorHandler().find(errorStack, regexStr));
    }

}
```

应用效果：