com.fr.report.core.watermark.WaterMarkProvider

8.09.010.02221011

WaterMarkProvider.java

```
package com.fr.report.core.watermark;

import com.fr.base.io.AttrMark;
import com.fr.base.iofile.attr.WatermarkAttr;
import com.fr.stable.fun.mark.Mutable;

/**
 *
 *
 * @author Hoky
 * @date 2021/12/20
 */

public interface WaterMarkProvider extends Mutable {

    int CURRENT_LEVEL = 1;

    String XML_TAG = "WaterMarkProvider";

    /**
     *
     *
     * @return
     */
    boolean isWaterMarkValid(AttrMark template);

    WatermarkAttr getWaterMark(AttrMark template);

    /**
     *
     *
     * @return
     */
    WaterMarkProvideLevel getWaterMarkProvideLevel();
}
```

FR	10.0		

FR	11.0		

plugin.xml

```
<extra-report>
        <WaterMarkProvider class="your class name"/>
</extra-report>
```

getWaterMarkProvideLevel

WaterMarkProvideLevel.java

```
package com.fr.report.core.watermark;

/**
 *
 *
 * @author Hoky
 * @date 2021/12/20
 */
public enum WaterMarkProvideLevel {
    PLUGIN(1),
    TEMPLATE(2),
    SERVER(3),
    OTHER(4);

    private Integer level;

    WaterMarkProvideLevel(int level) {
        this.level = level;
    }

    public Integer getLevel() {
        return level;
    }
}
```

> > >

isWaterMarkValid

WaterMarkFactory.java

```
package com.fr.report.core.watermark;

import com.fr.base.io.AttrMark;
import com.fr.base.iofile.attr.WatermarkAttr;
import com.fr.event.Event;
import com.fr.event.EventDispatcher;
import com.fr.event.Listener;
import com.fr.log.FineLoggerFactory;
import com.fr.plugin.context.PluginContext;
import com.fr.plugin.injectable.PluginModule;
import com.fr.plugin.observer.PluginEventType;
import com.fr.report.ExtraReportClassManager;
import com.fr.stable.Filter;
import com.fr.workspace.WorkContext;

import java.util.Comparator;
```

```
import java.util.Set;
import java.util.TreeSet;
import java.util.stream.Collectors;

/**
 *
 *
 * @author Hoky
 * @date 2021/12/20
 */
public class WaterMarkFactory {
    private static final Set<WaterMarkProvider> SORTED_LOADING_SET = new TreeSet<>(
            Comparator.comparing(o -> o.getWaterMarkProvideLevel().getLevel())
    );

    public static final WaterMarkProvider DEFAULT_MAKER = new DefaultWaterMarkProvider();

    private WaterMarkFactory() {
        SORTED_LOADING_SET.add(DEFAULT_MAKER);
        SORTED_LOADING_SET.add(new ServerWaterMarkProvider());
        SORTED_LOADING_SET.add(new TemplateWaterMarkProvider());
    }

    /**
     *
     *
     * @param template
     * @return
     */
    public WatermarkAttr getWaterMark(AttrMark template) {
        for (WaterMarkProvider waterMarkProvider : SORTED_LOADING_SET) {
            synchronized (this) {
                if (waterMarkProvider.isWaterMarkValid(template)) {
                    return waterMarkProvider.getWaterMark(template);
                }
            }
        }
        return new WatermarkAttr();
    }

    /**
     *
     *
     * @param template
     * @param loadLevel
     * @return
     */
    public WatermarkAttr getWaterMark(AttrMark template, WaterMarkProvideLevel loadLevel) {
        WaterMarkProvider waterMarkProvider = SORTED_LOADING_SET.stream()
                .filter(loader -> loader.getWaterMarkProvideLevel().equals(loadLevel))
                .findFirst().orElse(DEFAULT_MAKER);
        return waterMarkProvider.getWaterMark(template);
    }

    /**
     *
     *
     * @param loader
     */
    public void addLoader(WaterMarkProvider loader) {
        SORTED_LOADING_SET.add(loader);
    }

    public synchronized void installPluginLoader() {
        ExtraReportClassManager provider = PluginModule.getAgent(PluginModule.ExtraReport);
        if (provider != null) {
            Set<WaterMarkProvider> waterMarkProviders = provider.getArray(WaterMarkProvider.XML_TAG);
            if (!waterMarkProviders.isEmpty()) {
                waterMarkProviders.forEach(this::addLoader);
            }
        }
```

```
    }

    public synchronized void uninstallPluginLoader() {
        Set<WaterMarkProvider> pluginWaterMarkProviders = SORTED_LOADING_SET.stream()
                .filter(loader -> loader.getWaterMarkProvideLevel().equals(WaterMarkProvideLevel.PLUGIN))
                .collect(Collectors.toSet());
        SORTED_LOADING_SET.removeAll(pluginWaterMarkProviders);
    }

    private final static WaterMarkFactory FACTORY = new WaterMarkFactory();

    public static WaterMarkFactory getFactory() {
        return FACTORY;
    }

    public void initPluginListener() {
        //
        try {
            installPluginLoader();
            Filter<PluginContext> filter = context ->
                    context.getRuntime().contain(WaterMarkProvider.XML_TAG) && WorkContext.getCurrent().
isLocal();

            FineLoggerFactory.getLogger().info("init PluginListener");
            EventDispatcher.listen(PluginEventType.AfterRun, new Listener<PluginContext>() {
                @Override
                public void on(Event event, PluginContext pluginContext) {
                    //
                    installPluginLoader();
                }
            }, filter);

            EventDispatcher.listen(PluginEventType.BeforeStop, new Listener<PluginContext>() {
                @Override
                public void on(Event event, PluginContext pluginContext) {
                    //
                    uninstallPluginLoader();
                }
            }, filter);
        } catch (Throwable e) {
            FineLoggerFactory.getLogger().error(e.getMessage(), e);
        }

    }

}
```

public WatermarkAttr getWaterMark(AttrMark template)

public WatermarkAttr getWaterMark(AttrMark template, WaterMarkProvideLevel loadLevel)

TemplateWaterMarkProvider

```
package com.fr.report.core.watermark;

import com.fr.base.io.AttrMark;
import com.fr.base.iofile.attr.WaterMarkProvideConstant;
import com.fr.base.iofile.attr.WatermarkAttr;

/**
 *
 *
 * @author Hoky
 * @date 2021/12/20
 */
public class TemplateWaterMarkProvider extends AbstractWaterMarkProvider {

    @Override
    public boolean isWaterMarkValid(AttrMark template) {
        return template != null && (template.getAttrMark(WatermarkAttr.XML_TAG) != null
                && ((WatermarkAttr) template.getAttrMark(WatermarkAttr.XML_TAG)).isValid());
    }

    @Override
    public WatermarkAttr getWaterMark(AttrMark template) {
        WatermarkAttr attrMark = (WatermarkAttr) template.getAttrMark(WatermarkAttr.XML_TAG).clone();
        attrMark.setWaterMarkProvider(WaterMarkProvideConstant.TEMPLATE);
        return attrMark;
    }

    @Override
    public WaterMarkProvideLevel getWaterMarkProvideLevel() {
        return WaterMarkProvideLevel.TEMPLATE;
    }

}
```

ServerWaterMarkProvider

```
package com.fr.report.core.watermark;

import com.fr.base.io.AttrMark;
import com.fr.base.iofile.attr.WaterMarkProvideConstant;
import com.fr.base.iofile.attr.WatermarkAttr;
import com.fr.decision.config.WatermarkConfig;
import com.fr.decision.security.WatermarkData;

/**
 *
 *
 * @author Hoky
 * @date 2021/12/20
 */
public class ServerWaterMarkProvider extends AbstractWaterMarkProvider {
    @Override
    public boolean isWaterMarkValid(AttrMark template) {
        return WatermarkConfig.getInstance().isValid();
    }

    @Override
    public WatermarkAttr getWaterMark(AttrMark template) {
        WatermarkData watermarkData = WatermarkConfig.getInstance().getWatermarkData();
        WatermarkAttr watermarkAttr = new WatermarkAttr(watermarkData);
        //validfalsevalid
//        watermarkAttr.setValid(true);
        watermarkAttr.setWaterMarkProvider(WaterMarkProvideConstant.SERVER);
        return watermarkAttr;
    }

    @Override
    public WaterMarkProvideLevel getWaterMarkProvideLevel() {
        return WaterMarkProvideLevel.SERVER;
    }

}
```

PLUGIN

demodemo-water-mark-provider