com.fr.decision.fun.ComposeWebResourceProvider

-
-
-
-
-
-
-
-
-

WEBJS

1.
2.

ComposeWebResourceProviderWebResourceProviderWebResourceProviderComposeWebResourceProvider

ComposeWebResourceProvider.java

```
package com.fr.decision.fun;

import com.fr.stable.fun.mark.ComposeWebCoalition;

/**
 * @author Roger
 * @version 10.0
 * @date 2021/1/15
 * web
 */
public interface ComposeWebResourceProvider extends ComposeWebCoalition {

    String MARK_STRING = "ComposeWebResourceProvider";

    int CURRENT_LEVEL = 1;
}
```

ComposeWebCoalition.java

```
package com.fr.stable.fun.mark;


/**
 * @author Roger
 * @version 10.0
 * @date 2021/1/15
 *
 */
public interface ComposeWebCoalition extends Mutable {

    WebCoalition[] webCoalitions();
}
```

WebCoalition.java

```
package com.fr.stable.fun.mark;

import com.fr.web.struct.Atom;

/**
 * web
 */
public interface WebCoalition extends Mutable {

    /**
     *
     * @return
     */
    Atom attach();

    /**
     *
     *
     * @return
     */
    @Deprecated
    Atom client();

    /**
     *
     *
     * @return
     */
    Atom[] clients();

}
```

com.fr.web.struct.Atom

FR	10.0		
BI	5.1.3		

plugin.xml

```
<extra-decision>
        <ComposeWebResourceProvider class="your class name"/>
</extra-decision>
```

AtomActivator.java

```
package com.fr.web.struct;
```

```
import com.fr.module.extension.PrepareAdaptor;
import com.fr.plugin.context.PluginContext;
import com.fr.plugin.injectable.PluginInjectionFilter;
import com.fr.stable.ArrayUtils;
import com.fr.stable.fun.mark.ComposeWebCoalition;
import com.fr.stable.fun.mark.WebCoalition;
import com.fr.web.struct.extra.ModificatorInjectUtil;

/**
 * Created by juhaoyu on 2018/9/4.
 */
public class AtomActivator extends PrepareAdaptor {

    @Override
    public void prepare() {

        addMutable(PluginInjectionFilter.KEY, new PluginInjectionFilter<WebCoalition>(WebCoalition.class) {

            @Override
            public void on(WebCoalition webCoalition, PluginContext context) {
                register(webCoalition, context);
            }

            @Override
            public void off(WebCoalition webCoalition, PluginContext context) {
                remove(webCoalition, context);
            }
        });

        addMutable(PluginInjectionFilter.KEY, new PluginInjectionFilter<ComposeWebCoalition>
(ComposeWebCoalition.class) {

            @Override
            public void on(ComposeWebCoalition composeWebCoalition, PluginContext context) {
                for (WebCoalition webCoalition : composeWebCoalition.webCoalitions()) {
                    register(webCoalition, context);
                }
            }

            @Override
            public void off(ComposeWebCoalition composeWebCoalition, PluginContext context) {
                for (WebCoalition webCoalition : composeWebCoalition.webCoalitions()) {
                    remove(webCoalition, context);
                }
            }

        });
    }

    private void register(WebCoalition webCoalition, PluginContext context) {
        if (webCoalition.client() != null) {
            Atom wrappedAtom = ModificatorInjectUtil.wrapAtom(webCoalition.client(), context.getID());
            Registry.register(webCoalition.attach().getClass(), wrappedAtom);
        }
        if (ArrayUtils.isNotEmpty(webCoalition.clients())) {
            for (Atom client : webCoalition.clients()) {
                Atom wrappedAtom = ModificatorInjectUtil.wrapAtom(client, context.getID());
                Registry.register(webCoalition.attach().getClass(), wrappedAtom);
            }
        }
    }

    private void remove(WebCoalition webCoalition, PluginContext context) {
        if (webCoalition.client() != null) {
            Atom wrappedAtom = ModificatorInjectUtil.wrapAtom(webCoalition.client(), context.getID());
            Registry.remove(webCoalition.attach().getClass(), wrappedAtom);
        }
        if (ArrayUtils.isNotEmpty(webCoalition.clients())) {
            for (Atom client : webCoalition.clients()) {
                Atom wrappedAtom = ModificatorInjectUtil.wrapAtom(client, context.getID());
```

```
                Registry.remove(webCoalition.attach().getClass(), wrappedAtom);
            }
        }
    }
}
```

Registry.java

```
package com.fr.web.struct;

import com.fr.common.annotations.Open;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

/**
 *
 */
@Open
public class Registry {

    private static Map<Class<? extends Atom>, LinkedHashSet<Atom>>
            children = new ConcurrentHashMap<Class<? extends Atom>, LinkedHashSet<Atom>>();

    /**
     *
     * @param clazz
     * @param atom
     */
    public static void register(Class<? extends Atom> clazz, Atom... atom) {
        LinkedHashSet<Atom> set = children.get(clazz);
        if (set == null) {
            set = new LinkedHashSet<Atom>();
            children.put(clazz, set);
        }
        Collections.addAll(set, atom);
    }

    public static void remove(Class<? extends Atom> clazz, Atom... atom) {
        LinkedHashSet<Atom> set = children.get(clazz);
        if (set == null) {
            return;
        }
        set.removeAll(Arrays.asList(atom));
    }


    /**
     *
     * @param clazz
     * @return
     */
    public static Atom[] getChildren(Class<? extends Atom> clazz) {
        LinkedHashSet<Atom> set = children.get(clazz);
        if (set == null) {
            return new Atom[0];
        }
        set.remove(null);
        List<Atom> list = new ArrayList<Atom>();
        for (Atom atom : set) {
            Filter filter = atom.filter();
            if (filter == null || filter.accept()) {
                list.add(atom);
            }
        }
        return list.toArray(new Atom[0]);
    }
}
```

WebResourceProvider

1.plugin.xml

2.

WebResourceProvider

demodemo-compose-web-resource-provider

JSCSS