Table of Contents

If you call a server-side function or an action, usually you don't need more than the result on the client (UI). If you have complex business logic on server side, or if you call multiple server-side functions or actions in one single call, it would be useful to have server-side events with call information. Sometimes it's really helpful to do something after a single or all calls, e.g., cleanup of states, commit, or rollback connection(s) (if you don't use autocommit).

Our JVx server implementation has some events which could be useful for your application. We've defined the interface jvx.rad.server.ICallHandler with the following methods:

```
public CallEventHandler<IBeforeFirstCallListener> eventBeforeFirstCall();
public CallEventHandler<IAfterLastCallListener> eventAfterLastCall();

public CallEventHandler<IBeforeCallListener> eventBeforeCall();
public CallEventHandler<IAfterCallListener> eventAfterCall();

public void invokeAfterCall(Runnable pRunnable);
public void invokeAfterLastCall(Runnable pRunnable);
public void invokeFinally(Runnable pRunnable);
```

The ICallHandler is accessible via

```
ServerContext.getCurrentInstance().getCallHandler()
```

or

```
SessionContext.getCurrentInstance().getCallHandler()
```

The difference is that ServerContext always returns the call handler for the master session and SessionContext returns the call handler for the current session.

If you register listeners for the ICallHandler of MasterSession, they will be notified about all calls in your application. All other sessions will be notified about own calls.

The events are one feature of ICallHandler. The other feature are one-time method calls via invoke... methods. The concept is similar to invokeLater of GUI toolkits like swing or JavaFX: Execute “something” after all other methods were called. On server-side it's not good to name a method invokeLater because there are different options, e.g., invoke after current call (invokeAfterCall) or invoke after all calls (invokeAfterLastCall).

We have an additional method that invokes “something” after all other operations. It's invokeFinally.

The call stack could look like the following, for a single action call:

```
connection.callAction("doServerAction");
```

```
BEFORE FIRST call
BEFORE call
    doServerAction
AFTER Call
  invokeAfterCall
AFTER LAST call
```

```
  invokeAfterLastCall
  invokeFinally
```

multiple action calls:

```
connection.callAction(new String[] {"doServerAction", "doMailAction"});
```

```
BEFORE FIRST call
BEFORE call
    doServerAction
AFTER Call
  invokeAfterCall
BEFORE call
    doMailAction
AFTER Call
  invokeAfterCall
AFTER LAST call
  invokeAfterLastCall
  invokeFinally
```

Listener Registration

You should register your listeners in a method annotated with @PostConstruct:

```
@PostConstruct
public void createSession()
{
    ICallHandler handler =
SessionContext.getCurrentInstance().getCallHandler();

    handler.eventBeforeFirstCall().addListener(this, "doBeforeFirstCall");
    handler.eventBeforeCall().addListener(this, "doBeforeCall");
    handler.eventAfterCall().addListener(this, "doAfterCall");
    handler.eventAfterLastCall().addListener(this, "doAfterLastCall");
}
```

Sure, it would be possible to register listeners in constructor of your life cycle object, but you should know that the constructor could be called unexpectedly, e.g., if you inherit one LCO from another one:

```
Application
|-Session
  |- Screen
    |- LockedScreen
```

If you're using LockedScreen, the constructors of Screen, Session, and Application will be called. You could check the class:

```
if (getClass() == LockedScreen.class)
{
    //register listeners
```

```
}
```

but it's better to use @PostConstruct.

Unregister Listeners

Usually, it's not necessary to unregister listeners because the ICallHandler will be available as long as the session is alive. But if you're working with ServerContext, it's a good idea to remove your listeners because JVx doesn't do this automatically.

Simply use a method, annotated with @PreDestroy:

```
@PreDestroy
public void destroySession()
{
    ICallHandler handler =
ServerContext.getCurrentInstance().getCallHandler();

    handler.eventAfterLastCall().removeListener(this);
}
```

Invoke Methods or Listener Registration

It's not always possible to register a listener, or you won't do this because you don't need it for all your server calls. If you want to call a one-time function, please use invokeAfterCall or invokeAfterLastCall. Be careful with invokeFinally because this method was “reserved” for JVx objects.