Serialization issues

TL;DR

Fixing serialisation issues is a big, risky effort and should mostly done by using custom serialisation rather than the default mechanism.

Intro

Java Object serialization API provides a framework for encoding objects as byte streams and reconstructing objects from their byte-stream encodings.

Problem

MGNLUI-351 - Getting issue details... STATUS

As explained in the issue above, Magnolia is currently unable to serialise and deserialise its own Admincentral UI. This may be a problem especially in clustered environments where a user session may need to be replicated across multiple Java VMs.

Cause

In Magnolia's case the root object being serialised when a servlet container shuts down is `VaadinSession` which is a `Serializable` class. Most Vaadin classes are serializable, including `UI` from which our `AdmincentralUI` inherits.

Goals

We want to **detect and remove all serialisation issues** in our code.

Furthermore, we want to **find a way to detect serialization issues during Magnolia build** so that they are caught and tackled early.

Risks of Serialisation

Before proceeding with a possible solution I came up during my investigations and trials I'd like to point out some of the not so negligible disadvantages Serialization entails.

The points are mostly taken from *Effective Java*, 2nd Edition by Joshua Bloch which begins his chapter on Serialization with this significant warning: **implement Serializable judiciously.**

- **Implementing Serializable decreases the flexibility to change a class's implementation once it has been released**
    - When a class implements Serializable, its byte-stream encoding (or serialized form) becomes part of its exported API. Once you distribute a class widely, you are generally required to support the serialized form forever, just as you are required to support all other parts of the exported API. If you do not make the effort to design a custom serialised form, but merely accept the default, the serialised form will forever be tied to the class's original internal representation.

- **Increases the testing burden associated with releasing a new version of a class**
    - When a serializable class is revised, it is important to check that it is possible to serialise an instance in the new release and deserialise it in old releases, and vice versa. [...] These tests cannot be constructed automatically because, in addition to binary

compatibility, you must test for semantic compatibility. In other words, you must ensure both that the serialization-deserialization process succeeds and that it results in a faithful replica of the original object.

- **It can consume excessive space**
  - Especially using the default serialization mechanism you might end up with a huge object graph which painstakingly and recursively mirror every field and/or entry in a Collection.
- **It can consume excessive time**
  - The serialization logic has no knowledge of the topology of the object graph, so it must go through an expensive graph traversal.

Detecting and removing serialization issues

Suggested approach

After several attempts (which will be mentioned later on), this is the approach I would suggest

- start tomcat with this JVM parameter **`-Dsun.io.serialization.extendedDebugInfo=true`**
- login into Magnolia
- stop and restart Magnolia
- in the logs something like the following stack trace will show up

Serialization stacktrace

```
Jun 16, 2016 3:41:20 PM org.apache.catalina.session.StandardManager doLoad
SEVERE: IOException while loading persisted sessions: java.io.WriteAbortedException: writing aborted; java.io.
NotSerializableException: info.magnolia.event.ResettableEventBus$1
- field (class "info.magnolia.ui.admincentral.shellapp.applauncher.AppLauncherShellApp$3", name:
"val$systemRegistration", type: "interface info.magnolia.event.HandlerRegistration")
- object (class "info.magnolia.ui.admincentral.shellapp.applauncher.AppLauncherShellApp$3", info.magnolia.ui.
admincentral.shellapp.applauncher.AppLauncherShellApp$3@662420ac)
- field (class "com.vaadin.event.ListenerMethod", name: "target", type: "class java.lang.Object")
- custom writeObject data (class "com.vaadin.event.ListenerMethod")
- object (class "com.vaadin.event.ListenerMethod", com.vaadin.event.ListenerMethod@4637e20e)
- custom writeObject data (class "java.util.HashSet")
- object (class "java.util.LinkedHashSet", [com.vaadin.event.ListenerMethod@4637e20e])
- field (class "com.vaadin.event.EventRouter", name: "listenerList", type: "class java.util.LinkedHashSet")
- object (class "com.vaadin.event.EventRouter", com.vaadin.event.EventRouter@50986eaf)
- field (class "com.vaadin.server.AbstractClientConnector", name: "eventRouter", type: "class com.vaadin.event.
EventRouter")
- object (class "info.magnolia.ui.vaadin.applauncher.AppLauncher", info.magnolia.ui.vaadin.applauncher.
AppLauncher@7a66f618)
- custom writeObject data (class "java.util.EnumMap")
- object (class "java.util.EnumMap", {APPLAUNCHER=info.magnolia.ui.vaadin.applauncher.AppLauncher@7a66f618,
PULSE=com.vaadin.ui.VerticalLayout@141b10e1, FAVORITE=com.vaadin.ui.VerticalLayout@22f80d16})
- field (class "info.magnolia.ui.vaadin.gwt.client.magnoliashell.shell.MagnoliaShellState", name: "shellApps",
type: "interface java.util.Map")
- object (class "info.magnolia.ui.vaadin.gwt.client.magnoliashell.shell.MagnoliaShellState", info.magnolia.ui.
vaadin.gwt.client.magnoliashell.shell.MagnoliaShellState@4a4fd810)
- field (class "com.vaadin.server.AbstractClientConnector", name: "sharedState", type: "class com.vaadin.shared.
communication.SharedState")
- object (class "info.magnolia.ui.vaadin.magnoliashell.MagnoliaShell", info.magnolia.ui.vaadin.magnoliashell.
MagnoliaShell@6ceda10d)
- field (class "com.vaadin.ui.AbstractSingleComponentContainer", name: "content", type: "interface com.vaadin.
ui.Component")
- object (class "info.magnolia.ui.admincentral.AdmincentralUI", info.magnolia.ui.admincentral.
AdmincentralUI@619fbf56)
- custom writeObject data (class "java.util.HashMap")
- object (class "java.util.HashMap", {0=info.magnolia.ui.admincentral.AdmincentralUI@619fbf56})
- field (class "com.vaadin.server.VaadinSession", name: "uIs", type: "interface java.util.Map")
- root object (class "com.vaadin.server.VaadinSession", com.vaadin.server.VaadinSession@7238b701)
```

- With this information try to fix the "offending" class.
- In the case above, an inner class of `ResettableEventBus` seems to be not serialisable. It is interesting to notice how the root object being serialised is a `VaadinSession`. Thanks to the `extendedDebugInfo` you can follow up the whole serialisation path in the object graph until it

throws a `java.io.NotSerializableException`

Downsides

Unfortunately the report provided by `extendedDebugInfo` is not a full one, meaning that it stops at the first error encountered. This means your only option is to fix and start over the process outlined above until no more serialization exceptions show up.

Failed attempts

Before giving up to the laborious manual process above, I tried several options, including some fancy recursive scripts through the whole object graph using reflections and other magic. But to no avail. You can read about them in my comment to JIRA issue.

Use default serialization mechanism sparingly

In the first attempt I tried, I basically made each problematic class implement Serializable and let Java do the job. This soon turned out into having a gigantic object graph to be serialised where even Magnolia's core classes came into the picture. Consider the following example.

ShellImpl.java

```
@Singleton
public class ShellImpl extends AbstractUIContext implements Shell, MessageEventHandler {

    /**
     * Provides the current location of shell apps.
     */
    public interface ShellAppLocationProvider {
        Location getShellAppLocation(String name);
    }

    private EventBus admincentralEventBus;

    private AppController appController;

        private MessagesManager messagesManager;

        private MagnoliaShell magnoliaShell;

        private EventHandlerCollection<FragmentChangedHandler> handlers = new
EventHandlerCollection<FragmentChangedHandler>();

    private ShellAppLocationProvider shellAppLocationProvider;
...
```

By simply implementing `Serializable` you get rid of the `NotSerializableException` for `ShellImpl`, still you now need to do the same for all its fields and the fields they're made of internally and so on and so forth. `AppControllerImpl`, for one, has a reference to `info.magnolia.module.ModuleRegistry` which brings in core classes and soon you get a big, unmanageable mess of classes throwing `NotSerializableException`.

The most reasonable way to proceed so far seems to use custom serialisation by making some or all fields transient and then reconstructing the object with the special method *`private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException`*

For instance, the following could be a custom serialization for `ShellImpl`

ShellImpl.java custom serialization

```
@Singleton
public class ShellImpl extends AbstractUIContext implements Shell, MessageEventHandler, Serializable {

    [omitted]

    private transient EventBus admincentralEventBus;

    private transient AppController appController;

    private transient MessagesManager messagesManager;

    private transient MagnoliaShell magnoliaShell;

    private transient EventHandlerCollection<FragmentChangedHandler> handlers = new
EventHandlerCollection<FragmentChangedHandler>();

        //set by ShellAppController
    private transient ShellAppLocationProvider shellAppLocationProvider;


    @Inject
    public ShellImpl(@Named(AdmincentralEventBus.NAME) final EventBus admincentralEventBus, final AppController
appController, final MessagesManager messagesManager) {
        super();
        this.messagesManager = messagesManager;
        this.admincentralEventBus = admincentralEventBus;
        this.appController = appController;
        initShell();
    }

    private void initShell() {
        this.magnoliaShell = new MagnoliaShell();
        admincentralEventBus.addHandler(AppLifecycleEvent.class, new AppLifecycleEventHandler.Adapter() {
           [omitted]
        });

        this.admincentralEventBus.addHandler(MessageEvent.class, this);
        this.magnoliaShell.setListener(new MagnoliaShell.Listener() {
                [omitted]
        });

    }

        [more class internals omitted]

    /**
     * Override default deserialization logic to account for transient fields.
     */
    private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException {
        this.appController = Components.getComponent(AppController.class);
        this.messagesManager = Components.getComponent(MessagesManager.class);
        this.admincentralEventBus = Components.getComponentWithAnnotation(EventBus.class, Components.named
(AdmincentralEventBus.NAME));
        initShell();
    }
}
```

At any rate, when adopting custom serialisation one should be aware of some pitfalls as outlined by Effective Java, **Item75: Consider using a custom serialised form.**

Finally doing all this for Magnolia UI is a **quite a big effort,** considering the amount of classes involved and the cumbersome run and fix process outlined above.

Finding a way to detect serialization issues during Magnolia build

In this case I came up with a very simple Groovy script

```
import org.apache.commons.lang3.SerializationUtils
import com.vaadin.server.VaadinSession
/**
* This script will attempt to serialise a VaadinSession, that is the root object which is
* usually serialised by a Servlet container when shutting down.
* It will throw a java.io.NotSerializableException if something goes wrong in the serialization process at any
point of the object graph.
* The Servlet container should be run with the following JVM option -Dsun.io.serialization.
extendedDebugInfo=true
* in order to have a useful debugging output in case of error. */
vaadinSession = VaadinSession.getCurrent()
SerializationUtils.serialize(vaadinSession)
```

This works fine when run through the Magnolia Groovy Console and will basically output the same stack trace by Tomcat. I thought it could be run as an integration test, like we do for our `crawler.groovy`. However this does not work, the Groovy Maven plugin being basically disconnected from the Magnolia test instance (two different threads).

Proposal

Find a way to run the script above as an integration test against a real Magnolia instance. One idea could be registering a GroovyTestServlet which will get passed the script source and executes it. Then we assert that the output does not contain `java.io.NotSerializableException`.

I crafted this test https://git.magnolia-cms.com/projects/PLATFORM/repos/ce/compare/diff?targetBranch=refs%2Fheads%2Fmaster&sourceBranch=refs%2Fheads%2Fserialization-test&targetRepoId=545 but for some reason it does not seem to work in our integration tests environment, possibly due to the fact that no actual `VaadinSession` object is created (those tests are "headless") while the script works as expected when run against the magnoliaTest instance.

Another way to check for serialization issues against an actual Magnolia instance could be

- startup Magnolia
- login
- open some apps so to create a sort of real scenario (if you just login without opening apps, several UI classes won't be part of the session)
- stop Magnolia
- restart Magnolia
- scan logs for java.io.NotSerializableException

Architecture meeting decisions

https://wiki.magnolia-cms.com/pages/viewpage.action?spaceKey=ARCHI&title=2016-06-23+Meeting+notes

- Disable session persistence in Tomcat MGNLCE-46 - Getting issue details... STATUS
- Investigate other containers DOCU-751 - Getting issue details... STATUS
- make vaadin-session transient in http-session
    - no dice: I tried to wrap `VaadinSession` (see code below) and make it transient but this fails with

ClassCastEx

```
ERROR magnolia.ui.admincentral.AdmincentralVaadinServlet: An internal error has occurred in the
VaadinServlet.
javax.servlet.ServletException: com.vaadin.server.ServiceException: java.lang.ClassCastException:
info.magnolia.ui.admincentral.AdmincentralSessionListener$VaadinSessionWrapper cannot be cast to
com.vaadin.server.VaadinSession
        at com.vaadin.server.VaadinServlet.service(VaadinServlet.java:352)
        at info.magnolia.ui.admincentral.AdmincentralVaadinServlet.service
(AdmincentralVaadinServlet.java:131)
```

VaadinSession transient in HttpSession

```
public class AdmincentralSessionFilter extends AbstractMgnlFilter {

    private static final Logger log = LoggerFactory.getLogger("info.magnolia.debug");
    private boolean isWrapped = false;

    @Override
    public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain
chain) throws IOException, ServletException {

        if (!isWrapped) {
            // wrap com.vaadin.server.VaadinSession.AdminCentral
            Object session = request.getSession().getAttribute("com.vaadin.server.VaadinSession.
AdminCentral");
            if (session != null &amp;amp;&amp;amp; session instanceof VaadinSession) {
                VaadinSessionWrapper wrapper = new VaadinSessionWrapper((VaadinSession) session);
                request.getSession().setAttribute("com.vaadin.server.VaadinSession.AdminCentral",
wrapper);
                isWrapped = true;
            } else {
                log.error("Somesing wong!");
            }
        }
        chain.doFilter(request, response);
    }

    private static final class VaadinSessionWrapper implements HttpSessionBindingListener,
Serializable {

        private transient VaadinSession vaadinSession;

        public VaadinSessionWrapper(final VaadinSession session) {
            this.vaadinSession = session;
        }
                // bunch of delegate methods omitted
    }
}
```

Further ideas

This seems worth investigating https://apacheignite.readme.io/docs/web-session-clustering (thanks to Martin Drápela)

Investigation in memory usage related to IoC seems to be closely relevant to the topic as well (and quite revealing): Session dehydration: improvement proposal of IoC infrastructure in AdminCentral UI.