Things on a content management system

Tips and tricks for Day Communique and CQ5

CQ5 coding patterns: Sling vs JCR (part 1)

CQ5 as a complex framework is built on top of various other frameworks, on the server-side the most notably ones are JCR (with its implementation Apache Jackrabbit) and Apache Sling. Both are very powerful frameworks, but both of them have some overlap in functionality:

- reading data from the repository (Sling Resources vs JCR nodes/properties)
- notification on repository events (Sling Eventing vs JCR observation)

In these 2 areas you can work with both frameworks, and achieve good results with both. So, the question is, in what situation should you prefer Sling and in what situation pure JCR.

First, and I hope you agree here, in 99% of all cases use the an abstracted framework is recommended over the use of concrete technology (everybody uses JDBC and noone a direct database interface for e.g. MySQL) . It usually offers more flexibility and an easier learning cure. Same here. While on pure JCR you only work with raw repository structures (nodes and properties), the Sling resource abstraction offers you easier handling (no need to deal with the repository exceptions any more) and much more options to interact with you business objects and services.

As an example let's assume, that we need to read the title of CQ page /content/geometrixx/en/services. Using the JCR API it would look like this:

```
String readTitle (Session session) {
  Node page = session.getNode("/content/geometrixx/en/services");
  Node jcrcontent = page.getChild("jcr:content");
  Property titleProp= jcrcontent.getProperty ("title"):
  String title = titleProp.getValue().getString();
  return title;
}
```

This is a very small example, but it shows 3 pro

- We need to know, that all CQ page prop e below the page
- the title of the page is encoded in the pr
- Properties are not available immediately :ed from Properties to a Value to our expected type (String)

The same example with Sling:

```
String readTitle (ResourceResolver r
  Resource r = resolver.getResource(  );
  Page page = r.adaptTo(Page.class);
  String title = page.getTitle();
  return title;
}
```

We don’t need to deal with the low-level information (like the jcr:content node) and properties, but we use the appropriate business object (a CQ page object), which is available out of the box and offers a better level of abstraction.

On a Sling resource level we also a bunch of helpers available, which offer some great benefits over the use of plain nodes and properties:

- * the adaptTo() mechanism allows to convert a resource into appropriate objects representing a certain aspect of this resource, for example:

```
 LiveCopyStatus lcs = resource.adaptTo (LiveCopyStatus.class);
PageManager pm = resource.adaptTo (PageManager.class);
```

see http://localhost:4502/system/console/adapters for a (incomplete?) list.

- The ValueMap is abstracting away the Property type, type conversions are then done implicitly.
- And likely many many more.
- And if you ever need to deal with JCR directly, just use

```
Node node = resource.adaptTo(Node.class);
```

So, there are many reasons to use Sling instead of the JCR API.

Related

CQ coding patterns: Sling vs JCR (part 2)
In "code"

CQ development patterns - Sling ResourceResolver and JCR sessions

CQ coding patterns: Sling vs JCR (part 3)
In "code"

This entry was posted in Uncategorized on November 6, 2012 [http://cqdump.wordpress.com/2012/11/06/cq5-coding-patterns-sling-vs-jcr-part-1/] .

4 thoughts on “CQ5 coding patterns: Sling vs JCR (part 1)”

Ryan D. Lunka
November 12, 2012 at 19:25

I completely agree with this post, specifically that the Sling API and adaptable models are the correct approach 99% of the time. However, I struggle with this a bit when it comes to write operations – not just reads.

As far as I’m aware, Sling doesn’t provide great facilities for writing to repository. The argument can be made that it’s not what Sling is for, but from a practical standpoint, the situation comes up a lot. My current thinking is that the best approach is to adapt whatever resource requires a write operation to a class that can abstract that write operation (internally as JCR code) into a cleaner method.

Curious what your thoughts are on this…maybe a follow-up post?

Pingback: CQ coding patterns: Sling vs JCR (part 2) « Things on a content management system

Pingback: CQ coding patterns: Sling vs JCR (part 3) « Things on a content management system

Rajesh Rao
December 7, 2012 at 18:56

Thanks for the great Post Jörg, beautifully explained when to use Sling API and when to use JCR API. I would surely remember and try to follow this best practice.

Comments are closed.