GOOGLE APP ENGINE DATASTORE AND JAVASERVER PAGES STANDARD TAG LIBRARY

Vítězslav Novák[1]

[1] *Vysoká škola báňská-Technická univerzita Ostrava, Ekonomická fakulta, Sokolská třída 33, 701 21 Ostrava*
Email:vitezslav.novak@vsb.cz

Abstract: The term "cloud computing" has been used frequently a lot lately. One of the distribution models of cloud computing is Platform as a Service also known as PaaS, where the providers provide complete tools for the support of the whole life cycle of development and providing of applications. One of the providers of PaaS is also Google with its Google App Engine. Applications of the Google App Engine are developed with Java Enterprise Editions technologies like Java Servlets and JavaServer Pages. One of the services offered by the Google App Engine is also Datastore database. Unfortunately, this database is not relational, which makes it impossible to use all the common tools of the Java Enterprise Editions web applications to the access the data. In the JavaServer Pages these tools are mainly tags of JavaServer Pages Standard Tag Library of the SQL type. This article will show you how these tools can be adapted for the use with the Datastore database.

Keywords: cloud computing, datastore, Google app engine, Java data objects, Java enterprise edition, JavaServer pages standard tag library, Java servlet, platform as a service.

JEL classification: L86

Doručeno redakci: 9.2.2012; Recenzováno: 6.5.2013; 15.4.2013; 20.5.2013; Schváleno k publikování: 11.9.2013

Introduction

In the recent years, the term cloud computing has been used frequently. The main reason for this situation is the technological progress of the ICT world. The performance of the PC and servers as well as small devices like mobile phones or tablets is increasing, and all of these devices are basically permanently connected to the Internet. This is this fact that gives cloud computing a chance to succeed.

What in fact is cloud computing? The cloud computing is a model of development and use of ICT technologies based on the Internet. As Hurwitz, Bloor, Kaufman and Halper (2010) say, the cloud computing is the next stage in evolution of the Internet. The cloud computing provides the means by which everything - from computing power to computing infrastructure, applications, business processes to personal collaboration - can be delivered to you as a service wherever and whenever you need.

According to Pochyla (2010) the main idea of the cloud computing is that all applications work directly on the Internet, from simple applications to complete operating systems. Using cloud computing is in case of some services simply much cheaper than creating the services yourselves, because you do not need any expensive and productive hardware, you do not have to look after the update of the system or buy a software, you simply rent everything and use the services with the browser and quick internet connection.

One of the distribution models of the cloud computing is Platform as a Service. Feuerlicht (2010) explains that PaaS services address the need for a reliable, secure and scalable development environment that can be rapidly configured and deployed without extensive technical expertise. Google App Engine can also be one of the examples.

The Google App Engine lets you run your web applications on Google's infrastructure. The Google App Engine makes it easy to build an application that runs reliably, even under heavy load and with large amounts of data. With Google App Engine, there are no servers to be maintained: you only upload your application, and it is prepared to serve your users.

According to Google (cit. 2011-04-15), the Google App Engine includes the following features:

- dynamic web pages, with full support for common web technologies,
- persistent storage with queries, sorting and transactions,
- automatic scaling and load balancing,
- APIs for authenticating users and sending email using Google Accounts,
- a fully featured local development environment that simulates Google App Engine on your computer,
- task queues for performing work outside the scope of a web request,
- scheduled tasks for triggering events at specified times and regular intervals.

The Google App Engine supports applications written in several programming languages. With Google App Engine's Java runtime environment, you can build your applications using standard Java technologies, including the JVM, Java Servlets and the Java programming language or any other language using a JVM-based interpreter or compiler, such as JavaScript or Ruby. The Google App Engine also features a dedicated Python runtime environment, which includes a fast Python interpreter and the Python standard library. The Java and Python runtime environments are built to ensure that your application runs quickly, securely, and without interference from other applications in the system.

1 Datastore Database

Like all other web applications those run on Google App Engine need to store data in databases as well. That is why Google App Engine provides the non-relational database Datastore.

According to Google (cit. 2011-05-03), the Google App Engine Datastore provides robust, scalable storage for your web applications, with an emphasis on query performance. An application creates entities, with data values stored as properties of an entity. The Datastore can perform queries over entities. The Datastore can execute multiple operations in a single transaction. By definition, a transaction cannot succeed unless every operation in the transaction succeeds. If any of the operations fail, the transaction is automatically rolled back. This is especially useful for distributed web applications, where multiple users may be accessing or manipulating the same data at the same time. Unlike traditional databases, the Datastore uses the distributed architecture to automatically manage scaling to very large data sets.

The Datastore provides a low-level API with simple operations on entities, including get, put, delete, and query. But the Java SDK of Google App Engine includes implementations of the Java Data Objects (JDO) and Java Persistence API (JPA) interfaces for modeling and persisting data, unfortunately not the Java Database Connectivity (JDBC). These interfaces include mechanisms for defining classes for data objects, and for performing queries in Datastore. Also the developers of the web applications have got two possible APIs for data persistence: JDO or JPA. The low-level API of the Datastore is actually a very low-level one and in comparison with JDO or JPA it does not offer any query language at the SQL level.

Roos (2003) defines the standard JDO as a standardized application interface for the persistence of objects in Java. JDO enables implementing of various ways of storing objects – it can serve as an application interface of object oriented database, but it can also do mapping of objects to relational tables and storing of data to relational database. It is very important that the application interface is completely independent on the implementation of persistence, and so the applications using JDO can store data into various relational and object oriented databases with no bigger changes. Several object oriented databases (for example ObjectDB or JDOInstruments) have been developed on the basis of the standard JDO. The JDO uses JDOQL language as its query language. For example a query that selects all words ordered by the date of storage can in the JDOQL language look like this:

```
SELECT FROM package.Word ORDER BY date DESC
```

On the contrary according to Oracle (cit. 2011-05-06), the JPA provides a POJO persistence model for object-relational mapping. The JPA was developed as part of the EJB 3.0, but its use is not limited to EJB software components. It can also be used directly by web applications and application clients, and even outside the Java EE platform, for example, in Java SE applications.The JPA uses JPQL language as its query language. The same query as the one above can in JPQL look like this:

```
SELECT word FROM Word word ORDER BY word.date DESC
```

As you can see from the examples of the query languages JDOQL and JPQL, these languages are similar mainly because they are both based on the SELECT command of the SQL language, which is deeply rooted in the database praxis. Other commands than SELECT actually do not exist in the JDOQL or JPQL. SQL commands like INSERT, UPDATE or DELETE are replaced by the methods of the Java programing language.

Unfortunately, the SELECT commands of the JDOQL or JPQL differ from the SELECT command of the SQL in syntax as well as in results of queries in Java. A result of an SELECT command of the SQL in the Java is typically a data type `java.sql.Resulset`, which is more or less only a table encapsulated by an object of this date type. Nevertheless, the programming language is object-oriented and in order to use SQL query data effectively, it must map the records of the result table to objects. Because of this, object-relational mapping frameworks are used in object-oriented languages to map the database tables to classes of programming languages. JPA is one of these frameworks.

A result of JDOQL or JPQL query in Java is not a table but a list of objects, usually `java.util.List`, which can be processed in the Java. In this case as well as for example when using object-oriented database, there is no need for any mapping.

2 The Issue of Using Tags of JSTL with Datastore Database

The JSP pages fill the view tier in a typical web application designed according to the Model-View-Controller design pattern. There is mainly necessary to execute selection queries in the view tier of the web application. Their result is later presented on the JSP pages. There are tags called JavaServer Pages Standard Tag Library (JSTL) to be used for this purpose in Java EE web applications.

According to Oracle (cit. 2011-05-07), the JavaServer Pages Standard Tag Library (JSTL) encapsulates like simple tags the core functionality common to many Web applications. JSTL

offers support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags and SQL tags.
The use of JSTL tags is very simple. When it is necessary to present a result of the SQL SELECT command, you can use the tag `<sql:query>`, the command is given in its body. There is also another tag `<c:forEach>`, which can iterate through the result, for example a HTML table can be created without using the programming language Java, which makes the JSP page better arranged, as you can see in the following part of an JSP page:

```
<sql:query dataSource="jdbc/myDatasource" var="table">
    SELECT word FROM Word ORDER BY date DESC
</sql:query>
<table>
    <c:forEach items="${table}" var="row">
        <tr><td>${row.word}</td></tr>
    </c:forEach>
</table>
```

When you use the non-SQL database, like Datastore, there is problem only with some of the JSTL tags. For example the above mentioned tag `<c:forEach>` can iterate not only through tables but also lists. There is only a problem with the tags with `sql` prefix, which, as you can guess from the `sql` prefix itself, can only process SQL commands, but no JDOQL or JPQL commands. You must create a library of our own tags, which can process JDOQL or JPQL queries and return the date type usable in other JSTL tags (typically a list), to be able to use other JSTL tags (tags with different prefixes than `sql`).

3 Suggestion of Tags Enabling the Execution of JDOQL and/or JPQL Queries
What could the use of such a new tag enabling an execution of a query e.g. with JDOQL look like? Such a tag could look like this:

```
<jdoql:query persistenceManagerFactory="${pmf}" var="list">
    SELECT FROM package.Word ORDER BY date DESC
</jdoql:query>
<table>
    <c:forEach items="${list}" var="object">
        <tr><td>${object.word}</td></tr>
    </c:forEach>
</table>
```

As you can see from the example, the tag `<jdoql:query>` would keep the syntax of the tag `<sql:query>`, which means that the SELECT command would be set to the body of the tag, reference to persistence manager factory would be set with help of the Expression language as the attribute `persistenceManagerFactory` of the tag, and the resulting list of the objects would be stored with help of an attribute named with the help of `var` attribute. Such list of objects could be easily processed by common JSTL tags.

The `<jpql:query>` tag using the JPQL language would look analogously, which means that `persistenceManagerFactory` attribute of the tag would replace the `entityManagerFactory` attribute referencing to entity manager factory, which need the JPA implementation, as you can see here:

```
<jpql:query entityManagerFactory="${emf}" var="list">
    SELECT word FROM Word word ORDER BY word.date DESC
</jpql:query>
```

Another question is what way would be the best to implement new tags. The most suitable technology for implementing the suggested tags would be the Custom tags technology, which is part of Java Enterprise Edition. According to Oracle (cit. 2011-05-09), Custom tags are user-defined JSP language elements that encapsulate recurring tasks.

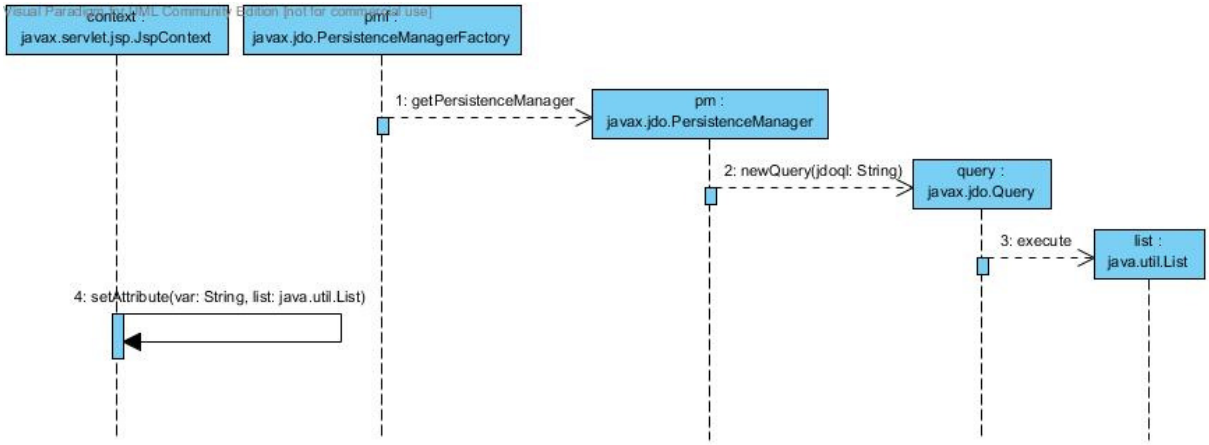
The Custom tags technology works according to the following principle: instead of a tag placed in JSP page a method, which does the asked action, of tag handler is processed. The tag library descriptor describes which class is a tag handler, if a tag can include a body and what attributes a tag can include. A description of the tag `<jdoql:query>` in the tag library descriptor could look like this:

```
<tag>
    <name>query</name>
    <tag-class>package.Query</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
        <name>persistenceManagerFactory</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
        <type>javax.jdo.PersistenceManagerFactory</type>
    </attribute>
    <attribute>
        <name>var</name>
        <required>true</required>
        <rtexprvalue>false</rtexprvalue>
        <type>java.lang.String</type>
    </attribute>
</tag>
```

From the above mentioned description of the tag `<jdoql:query>` in tag library descriptor it is obvious that the tag could include a body (query in the JDOQL) including the possibility to use the Expression Language and the tag also should have the required attributes `persistenceManagerFactory` and `var`.

In case of use of a newer and simpler type of Custom tags, the Simple tags, for implementation of a tag, the tag handler is a class extended from `javax.servlet.jsp.tagext.SimpleTagSupport`. In this class the method `void doTag()` is processed. And exactly in this method it is necessary to implement the java code, which will process a query in the JDOQL or JPQL, as you can see in Figure 1:

Figure 1: Processing of JDOQL Query in Tag Handler's Method `void doTag()` of Tag `<jdoql:query>`

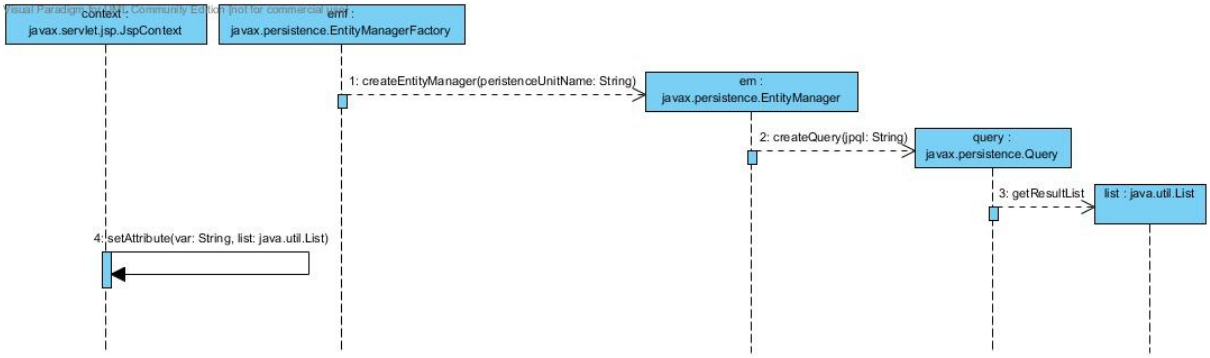


Source: Processed by author

Thanks to the attributes of the tag `<jdoql:query>` in the tag handler, there are available references to the persistence manager factory and the name of variable, under which the resulting list of objects is saved. As you can see from the sequence diagram above, in the method `void doTag()` of the tag handler of the tag we can only use the persistence manager factory to create the persistence manager. Based on the query of the JDOQL you can create and execute an object of a query with the help of the persistence manager. This is how you get the resulting list of objects. At the end it is necessary to set the list as an attribute in the page scope under the chosen name of variable with the help of JSP context.

In case of JPA interface use, the sequence diagram would look very similar, the date types and methods used by the JDO interface would replace the date types and methods used by the JPA interface, as you can see in Figure 2:

Figure 2: Processing of JPQL Query in Tag Handler's Method `void doTag()` of Tag `<jpql:query>`



Source: Processed by author

The above mentioned tags `<jdoql:query>` or `<jpql:query>` implemented according to the above mentioned suggestion would enable a full replacement of the tag `<sql:query>` in the standard library of the JSTL. In a similar way, if necessary, you may also replace other tags of the JSTL with `sql` prefix, such as `<sql:update>` and more. Other tags of the JSTL with other prefixes would stay fully usable.

Conclusion

Google App Engine is one example of cloud computing service based on the distribution model of Platform as a Service. Among others, it provides a space for deploying Java Enterprise Edition web applications with the possibility to store the data into the non-relational Datastore database. However, there is an issue related to the Datastore - when you decide to use the tags with `sql` prefix of the JavaServer Pages Standard Tag Library to retrieve data on JSP page of a web application. With these tags you cannot use the commands of JDOQL or JPQL, which are supported by the Datastore, but only the SQL commands. When you want to use the tags of the JavaServer Pages Standard Tag Library, you need to replace the tags with `sql` prefix by the new ones. In this article, the principles to be used when designing and implementing these tags were described.

References

[1] FEUERLICHT, G., 2010. Impact of Cloud Computing: Beyond Technology. In *System Integration*.Praha. s. 262-269. ISBN 978-80-245-1660-8

[2] Google. *What Is Google App Engine?* [online] [cit. 2011-04-15]. Available at: <http://code.google.com/intl/cs/appengine/docs/whatisgoogleappengine.html>

[3] Google. *Datastore Overview* [online] [cit. 2011-05-03]. Available at: <http://code.google.com/intl/cs/appengine/docs/java/datastore/overview.html>

[4] HURWITZ, J., R. BLOOR, M. KAUFMAN and F. HALPER, 2010. *Cloud Computing for Dummies*. 1st ed. Hoboken: Wiley Publishing. 310 p. ISBN: 978-0-470-48470-8.

[5] Oracle. *JavaServer Pages Standard Tag Library* [online] [cit. 2011-05-07]. Available at: <http://www.oracle.com/technetwork/java/index-jsp-135995.html>

[6] Oracle. *Java Persistence API* [online] [cit. 2011-05-06]. Available at: <http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>

[7] Oracle. *The Java EE 5 tutorial* [online] [cit. 2011-05-09]. Available at: <http://download.oracle.com/javaee/5/tutorial/doc/bnaiy.html>

[8] POCHYLA, M., 2010. Cloud computing pro malé a střední firmy. In *Informační technologie pro praxi 2010*.Ostrava. p. 114-123. ISBN 978-80-248-2300-3.

[9] ROOS, R. M., 2003. *Java Data Objects*. London: Addison-Wesley. 244 p. ISBN 0-321-12380-8