JSP Best Practises

Lars Larsson

Lecture #9

1. Data transfer
2. Software design and engineering
3. Tool utilisation and standards
4. JavaBeans and Enterprise JavaBeans
5. Sessions
6. The big picture
7. Client-side usability concerns
8. Summary

Data transfer

As you know, data transfer is very important in server-side web development. Obviously, we need to validate data coming in from the user, and we should always do so with great scrutiny. The rule of thumb is to write once and read many times: invalid data may corrupt our entire database, so any time spent validating it is worth it. We may use tools and technologies for this purpose, and we will get to that later in the lecture.

This first portion of the lecture deals with **how** we perform data transfer.

Use HTTP POST to send data

You are familiar with the difference between HTTP POST and GET. POST should be used for data transmission from the client to the server because:

- URLs have a short maximum length (Internet Explorer supports up to 2048 for the GET method),
- data is not exposed in the address bar and
- since POST data has to be entered (and cannot be embedded in links), we can hold the sender accountable for it.

Appropriate use of HTTP GET

Since HTTP GET allows embedded data in the request, we can create links to resources that need processing on the server side such as search results or choosing which sub-page we want to view.

It is a good idea to use GET if the interaction is like a question that may be asked several times, but not if we need to transmit one-shot data. GET also enables us to do caching and makes it possible for users to create bookmarks to resources.

GET should only be used if it is side-effect free: repeated GETs should not modify anything on the server side.

Use HTTPS

HTTPS introduces some overhead: connection establishment takes longer and more memory is required on the server side, since each document has to be sent over encrypted channels. Still, we should use HTTPS every time we send anything even remotely sensitive.

Many users distrust the Internet, since they don't feel safe and under constant surveillance. Help these people by using encryption, and your site might just become more popular than the competition's!

Cookies and safety

Never, **ever**, send anything sensitive in cookies! Cookies are stored on the client's machine (which we cannot know is safe) – in plain text. Of course, one might encrypt the contents of the cookie, but that leaves us with two problems:

- if we never change the encryption key, one can use the cookie and crack the key,
- if we do change the encryption key, we will not be able to decrypt the cookie after a change, rendering the cookie useless.

Do not rely on cookies – users may choose to block them and they can be deleted at random times.

Security checklist

There are many security checklists available on the web one might use, the key is to use them **during** development – not after. Security cannot be added later, as a feature – it has to be actively implemented constantly throughout the entire development process.

The following is a security checklist for ASP.NET 2.0, but most of it applies to JSP as well:

`http://msdn2.microsoft.com/en-us/library/ms998249.aspx`

Correct `mime-type`

Web browsers are growing increasingly more capable when it comes to displaying various files and at guessing what the contents of a file may be, but we must always send the correct `mime-type` when we deal with anything other than `text/html`. Never assume that the browser guesses correctly, they most certainly don't!

Software design and engineering

The topic of good software design and engineering is too large to cover here and we recommend that you take at least one course on it.

During this course, we have used several techniques and important concepts that you should apply during your future development.

Tag libraries

Large code blocks are hard to maintain, and they become very distracting while developing a site. It is much better to put these large code blocks in tag libraries, like the ones you used in the assignments. It is also much, much easier for a non-programmer to use: working in a group with a web designer, it is easier to make non-programmers add a single tag than to tell them to paste 100 lines of code that makes no sense to them.

Downloadable tag libraries

There are many tag libraries available for download that you can use in future projects. They simplify tasks that are often carried out and allow you to develop working sites faster. Visit the following sites:

`http://java.sun.com/products/jsp/jstl/`

`http://jakarta.apache.org/taglibs/doc/standard-doc/`

Structure and hierarchies

Always build a site as if it were ten times its intended size. Make sure that everything (code, images, static documents) is neatly placed in logical structures and hierarchies, because once your site happens to become popular and grow the initial costs of creating a good structure will be enormous winnings in simplified maintenance.

Design patterns

Design patterns are very important in software engineering. We have used some during the course, and you should continue using them, for instance:

- Accessors
- Model-View-Controller (MVC)

Before embarking on a large programming quest, make sure you have found a suitable design pattern and stick to it religiously. Wikipedia is a great, free, starting point:

`http://en.wikipedia.org/wiki/Design_pattern_(computer_science)`

Tried and tested code

The key to rapid development of secure and functional software is to use tried and tested code. We should never reinvent the wheel, unless we are forced to. Java itself comes with a plethora of useful classes and we can add many useful web components by downloading them from the Jakarta Commons project:

`http://jakarta.apache.org/commons/`

Tool utilisation and standards

Taking the "don't reinvent the wheel"-notion one step further, we should never perform tasks manually if we can automate them. Data validation, parsing, etc. are all good candidates for automation.

Validation via XML Schemas

We can, and should, use XML schemas for validation of data. The basic idea is that we take data, put it in an XML document and see if it validates using an XML schema. Since the XML schema language is so powerful, we can place a large number of constraints on the data. Therefore, if the XML document passes validation, it contains data in our required format.

Database standards

As far as possible, it is recommended that we make use of ANSI SQL and JDBC. The reason for this is that we should avoid becoming dependant on one single database vendor:

- Upgrading to a high-performance database management system will be easier and
- we know that our application will not break due to upgrading to a newer version of the same DBMS.

Dead links

Dead links lower users' opinion of your site by a lot, it looks unprofessional and is incredibly annoying. Therefore, you should always make sure that your links are alive – preferably using an automated tool. There are many tools that do this, and you should find one you are comfortable with using since you should do it often.

HTTP response codes

HTTP has response codes that you are familiar with (404 should be well known, for instance). These should always be used, since they let the browser handle it in a way that is (hopefully) customisable and familiar to the user.

Error pages

JSP has support for custom error pages, which should be used. Displaying stack traces and other technical information is very bad, since it reveals intimate details from your code that might be of use to an attacker. This is closely related to information leakage.

Error pages are special pages that are displayed automatically when an exception occurs, and they are provided with information on the exception that occurred.

Error pages, example

The following line allows a page to act as an error page:

```
<%@ page isErrorPage="true" %>
```

Within the other pages on the site, we set which error page should be used like so:

```
<%@ page errorPage="theErrorPage.jsp" %>
```

The exception that occurred is stored in a variable called `exception`, which can be analysed and handled in some way.

Using `web.xml`

When we deploy a web application, we should always use a deployment descriptor, where we can set such things as error pages and other settings that are important to the web application.

See chapter 5 in “More Servlets”, which is devoted entirely to this topic.

File uploads

We have not covered file uploads yet during this course but it is simple to do using the appropriate code from Jakarta Commons. Using ready-made code from the Jakarta Commons project, handling file uploads is a snap.

Study the User Guide and download the FileUpload package from this address to get started:

`http://jakarta.apache.org/commons/fileupload/`

JavaBeans – regular and Enterprise

JavaBeans and Enterprise JavaBeans help us as programmers create re-usable and easily maintained code. We shall take a look at these technologies, which are of tremendous value in large scale development.

JavaBeans

JavaBeans are simply Java classes that adhere to a certain naming convention, which makes it easy to analyse them and thus plug them into new situations.

- `getX()`, `setX()` – implies that the bean has a value X that can be manipulated. The type is inferred by the argument to `setX()` and the return type of `getX()`
- `isX()` – investigate the bean's boolean value X

All JavaBeans must have a constructor that takes no arguments, they must be serializeable and should not contain any required event-handling methods.

JavaBeans continued

The point of JavaBeans is that they can act as components of larger software and be plugged in to software on-the-fly with a minimum of effort. They can be manipulated by a visual tool, since the set of methods convey enough information on the capabilities of the bean.

Enterprise JavaBeans

Enterprise JavaBeans (EJB) encapsulate the business logic of an application (such as handling databases, security and other behind-the-scenes type of jobs). The idea is that business logic is typically re-usable, whereas user interfaces are not.

Application servers

EJBs reside in an application server environment. The application server provides the beans with persistence, transaction processing, concurrency control, events, naming and directory services, security, simple deployment, remote procedure calls and exposure of business methods as web services. EJBs are distributed objects.

Getting all of the above from the environment means that the EJB programmer can concentrate on the task at hand and not worry about these (hard!) topics.

EJB types

EJBs are placed in EJB containers, of which there are three main types:

- Session Beans:
  - Stateless Session Beans – allows concurrent access
  - Stateful Session Beans – maintains state, disallows concurrent access
- Entity Beans – distributed objects having a persistent state, not on a session basis
- Message Driven Beans – asynchronous beans

EJB tutorial and example

EJBs are too complicated to be covered during this lecture, but you are recommended and encouraged to find out more information on your own. Sun provides the following useful resource:

`http://java.sun.com/developer/onlineTraining/Beans/EJBTutorial/`

Sessions

Sessions have been used during this course and are of great importance in today's Internet, so we should study some best practises related to them.

Session security

Sessions are identified by session IDs. Thus, a common attack is to hijack an existing session ID and use it as your own. Using HTTPS alleviates this problem, and should be used unless there is a very compelling reason not to.

Secure all of the pages when you apply security to some part of your site – all resources from which as session is created (or accessed) must be either secured or unsecured. Mixing is not possible.

Session lifetime

Sessions consume resources, and as such, we do not want to keep them around longer than we have to. Remove objects that are no longer needed as quickly as possible, and (once a session object contains no more active objects) call `invalidate()` on the session object to free up valuable resources.

Objects in sessions

If you need to store objects in a session, do so by making the objects serializeable. Make sure that the classes are in the correct class path, so they can be deserialised correctly and without namespace conflicts.

Avoid storing large amounts of data in sessions, it is very inefficient.

The big picture

We need to keep track of the big picture when we develop web sites – the Internet grows quickly and so do our sites when we suddenly become successful.

Never simply write a bunch of JSP pages that you stick in a directory somewhere – it scales horribly. Make environments such as the one used in the assignments and create web applications – it pays off in the long run.

WSDL

Web services are useful and enable us to do many exciting things, as long as they work together. There are competing technologies, but for the same reason that we should use ANSI SQL and JDBC, we should strive to use the WSDL when we create web services. Being backed by the W3C is a major strength.

Offer APIs to web services

Allow others to use your web service by creating APIs to your web service! The major players right now (Google, for instance) are doing and thus they get increasing amounts of traffic and exposure. Don't miss out by trying to be secretive if you are offering a good product!

Client-side usability concerns

Since you should all have taken a course on client-side web development, it is sufficient to note the following:

- Don't use JavaScript unless you have to (AJAX is fine, but replacing standard browser features like handling links or scrollbars isn't!).
- Never assume that your users have any of the following (even if you do): good displays with high resolution and many colours, good vision, ability to use a mouse, colour vision, desire to read long texts or above average intelligence (per definition, the majority does not).

Usability resources

These are a few good links on usability, which were also shown during the final lecture in the client-side web development course (5DV077):

- `http://www.useit.com/`
- `http://www.uidesign.net/`
- `http://www.usabilityinstitute.com/`
- `http://www.usability.gov/`

Summary

During this lecture, we have studied some best practises related to JSP and web development in general. What it all boils down to in the end is common sense and being careful. The most resource-consuming part of any software development project is in the maintenance, so any extra time spent in the design phase will be repaid many times over later on.

Next lecture introduces other server-side web development languages and environments, and compares them to JSP.