Information Retrieval

WS 2013 / 2014

Lecture 7, Tuesday December 3rd, 2013

(Cookies, CORS, UTF-8)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

UNI FREIBURG

Overview of this lecture

UNI FREIBURG

- Organizational
  - Your experiences with ES#6 (web application)
- CORS, Cookies, UTF-8
  - More practically relevant web app stuff:

    Cookies: store information across web sessions

    CORS: using resources from other sources

    UTF-8: how to encode characters like ä or € or 谢

Exercise Sheet 7: add a feature to your web app using cookies + convert ISO-8859-1 input to UTF-8

Experiences with ES#6 (search web app)

UNI FREIBURG

- Summary / excerpts last checked December 3, 14:30
  - Nice exercise sheet
  - Many of you had no prior experience with HTML, JavaScript, etc … therefore quite time-consuming to get used to it
  - Many of those with more experience spent quite some time on playing around and trying things

    Let's have a look at two examples today + more next week
  - Thanks Björn for the master solution for ES#5 … indeed !
  - Exercise 1: makes no sense / takes hours / better use a library

    Hmm, if it takes hours, then you can still learn a lot

    Using a library, you (1) don't realize how simple HTTP actually is, and you (2) don't learn all the little details

Cookies 1/5

UNI FREIBURG

- Basic mechanism
  - A cookie is simply a string associated with a web page that is stored on the client's computer

    Each client has it's own cookie

    Typically used for user data and preferences
  - A cookie can contain any contents, but the convention is that it contains a sequence of key-value pairs, separated by semicolons, for example:

    user=cookie-monster; prefers=kekse
  - Implementation in JavaScript is **very** simple, just read and write this string via the variable document.cookie

Cookies 2/5

UNI FREIBURG

- Adding key-value pairs to a Cookie
  - To add a key-value pair, just write

    ```
    document.cookie = "user=cookie-monster";
    ```

  - Multiple assignments **add** to the string … weird but true

    ```
    document.cookie = "user=cookie-monster";
    document.cookie = "prefers=kekse";
    ```

    Cookie string: user=cookie-monster; prefers=kekse

  - To overwrite the value for a key, just write again

    ```
    document.cookie = "prefers=kekse";
    document.cookie = "prefers=kruemel";
    ```

    Cookie string: prefers=kruemel

Cookies 3/5

UNI FREIBURG

- Getting the value for a particular key
  - In raw JavaScript, need some string processing:

```
var cookies = document.cookie.split(";");
for (var i = 0; i < cookies.length; i++) {
   var args = cookies.replace(/\s/g,"").split("=");
   if (args[0] == "user") alert("Hi " + args[1] + " !!!");
}
```

Cookies 4/5

UNI FREIBURG

- Different kinds of cookies
  - **Chocolate chip cookie**

    Accidentally developed by Ruth Wakefield in 1930
  - **Session cookie** … lasts as long as your browser is open

    user=cookie-monster
  - **Persistent cookie** … lasts until the specified date

    user=cookie-monster; expires=Wed 04 Dec 2013 17:45
  - **Third-party cookies** … from JavaScript from other domains

    Beware: these often give access to sensitive information

    In the JavaScript Console (Ctrl+Shift+J in Chrome), easily see and manage all Cookies under **Resources → Cookies**

Cookies 5/5

UNI FREIBURG

- In **jQuery** working with Cookies is super-easy
  - Setting a cookie

    ```
    $.cookie("user", "cookie-monster");
    ```

  - Value of a cooke

    ```
    var user = $.cookie("user");
    ```

  - Removing a cookie

    ```
    $.removeCookie("user");
    ```

  - Cookie with expiry date (10 days from now)

    ```
    $.cookie("user", "cookie-monster", { expires: 10});
    ```

CORS 1/4

UNI FREIBURG

- Cross-Site-Scripting (XSS)
  - Principle: inject malicious JavaScript code into web page
  - Example 1: enter JavaScript into search box

    Click me!
  - Example 2: send someone a mail with a link

    ...index.php?user=guest<script>alert("Ha!")</script>
  - Example 3: post to forum with some script in it

    I have a question on Exercise Sheet 7.
    <script>... JS code to send me user info by mail ...</script>

    Note: The <script>...</script> will not show on the website, but code will be executed by **any client** viewing the post

CORS 2/4

UNI FREIBURG

- The same-origin-policy
  - Domain + port of client and server URL must be **identical**

    http://**etna.cs.uni-freiburg.de:8888**/search.html

    http://**etna.cs.uni-freiburg.de:8888**/?q=zurich
  - To understand why, consider the following scenario:

    An application somehow managed to copy your session cookie for Facebook and redirect you to **www.evil.com**

    Without the same-origin-policy, the evil site could now use that cookie to log into your Facebook account and do all kinds of funny (or not so funny) stuff

    With the same-origin-policy it cannot

CORS 3/4

UNI FREIBURG

- Exceptions to the Same-Origin-Policy
  - JavaScript can be loaded from **anywhere**

    That way we could use jQuery without downloading it

    <script src="http://code.jquery.com/jquery1.10.2.js">
  - There are applications where it is actually desirable that everybody (or many people) can access then

    For example, our backend for query suggestions

    Or an API to a public database

CORS 4/4

UNI FREIBURG

- CORS = Cross-Origin Resource Sharing
  - Principle: the server explicitly specifies which web sites may use the results it returns
  - The implementation is very simple:

    Modern browsers send the following request header

    Origin: http://<host name>:<port>

    Depending on that header, or independent of it, the server can then send a response header like this:

    Access-Control-Allow-Origin: http://<host name>:<port>

    Browser then uses the result **only when the two agree**

Unicode 1/7

UNI FREIBURG

- Motivation
  - To represent text in binary, we need a standard for how to represent the characters of the alphabet, numbers, etc.
  - For a very long time, this standard was **ASCII** :

    1 Byte per symbol = can represent 256 different symbols
  - Obviously there are more than 256 symbols in the world

    Chinese alone has (tens of) thousands of different symbols

Unicode 2/7

UNI FREIBURG

- Solution before Unicode
  - Use the ASCII codes 0 – 127 for common symbols, which (almost) everybody needs

    a-z A-Z 0-9 ( ) [ ] { } , . : ; " ' …

    ASCII codes 0 – 31 used for control characters
  - For the ASCII codes 128 – 255, have (many) different variants, depending on the context

    For example, ISO-8859-1: use the codes to encode all the funny characters from most European languages

    à á â ã ä å ç è é ë ì í î ï ð ñ ò ó ô õ ö ø …
  - Problem: if you need more than one variant, you need to switch the encoding in the middle of the document

Unicode 3/7

UNI FREIBURG

- The Unicode solution
  - Simply assign a **unique** number, called **code point**, to (almost) every character / symbol in the world, e.g.

    a : 97 (hex = 61)
    A : 65 (hex = 41)
    ä : 228 (hex = E4)
    α : 945 (hex = 03B1)
    € : 8364 (hex = 20AC)
     : 128584 (hex = 1F648)

  - Unicode knows 1,114,112 code points (hex: 0 .. 10FFFF)

    Note: 1 Byte not enough, and 2 Bytes also not enough

Unicode 4/7

UNI FREIBURG

- UTF = Unicode Transformation Standard
  - There are different schemes for how to actually represent these code points in binary

    UTF-32: always use **4 bytes** per code point
    obviously enough for all 1,114,112 known code points

    UTF-16: use **2 bytes** for the common code points,
    and 4 bytes for the others … used for **String** in Java

    UTF-8: use **1 byte** for the very common code points,
    and 2 or 3 or 4 bytes for the others … see next 2 slides

    UTF-16 and UTF-8 are **variable-byte** encodings

Unicode 5/7

UNI FREIBURG

- Details of UTF-8
  - **1 Byte:** Code point in [0, 127] = xxxxxxx

    UTF-8 code: 0xxxxxxx 7 Bits
  - **2 Bytes:** Code point in [128, 2047] = yyyxxxxxxxx

    UTF-8 code: 110yyyxx 10xxxxxx 11 Bits
  - **3 Bytes:** Unicode in [2048, 65535] = yyyyyyyyxxxxxxxx

    UTF-8 code: 1110yyyy 10yyyyxx 10xxxxxx 16 Bits
  - **4 Bytes:** Unicode in [65536, 2^{21} - 1] = zzzzzyyyyyyyyxxxxxxxx

    UTF-8 code: 11110zzz 10zzyyyy 10yyyyxx 10xxxxxx 21 Bits

    In principle, could continue with 5-byte and 6-byte sequences, but UTF-8 stops here, since $2^{21} \approx 2M$ is enough RFC 3629

Unicode 6/7

UNI FREIBURG

- UTF-8 has the following nice properties
  - ASCII compatible = a string of characters with ASCII codes < 128 is the same in ASCII as in UTF-8

    So old C / C++ code only fails on the special characters
  - ISO-8859-1 characters (ä ã â …) with code point 1xyyyyyy have the 2-byte UTF-8 encoding 1100001x 10yyyyyy

    You may want to make use of this for Exercise 7.3
  - Only rarely used characters need more than 2 bytes
  - Easy to decode: codes start and end at byte boundaries
  - Can decode starting from anywhere within a string

    Just move left to the next byte not starting with 10

Unicode 7/7

UNI FREIBURG

- Some more properties of UTF-8
  - In a multi-byte UTF-8 character all bytes are ≥ 128, and vice versa such bytes occur only for multi-byte characters
  - The number of leading 1s in the first byte of a multi-byte character is equal to the number of bytes of its code
  - For every Unicode in $[0, 2^{21} - 1]$ there is **exactly one** valid UTF-8 multi-byte sequence
  - But vice versa not all multi-byte sequences are valid UTF-8
  - For example **110**0000x **10**xxxxxx is **not** valid

    Should be encoded with 1 byte: 0xxxxxxx

URL encoding and decoding

UNI FREIBURG

- Many characters not allowed in a URL
  - Only: a-z A-Z 0-9 $ % / - _ . + ! * … and a few more

    In particular: no space, and also no ä ã â …
  - Arguments of GET request are part of the URL

    In particular, the ?q=... part of your web app for ES6
  - Special characters are encoded as follows (by example)

    If encoding of web page is UTF-8

    ä : UTF-8 code C3A4 → URL-encoded as %C3%A4

    For decoding, do just the reverse … Exercise 7.2

    If encoding of web page is ISO-8859-1:

    ä : ISO-8859-1 code E4 → URL-encoded as %E4

Implementation Advice

UNI FREIBURG

- For Exercise Sheet 7
  - To view the **byte-wise** contents of a file, independent of it's encoding use the Linux tool xxd or xxd –b

    Inside an IDE, Text Editor, or Console what you see is already an interpretation of the contents of the file, assuming a certain encoding, e.g. UTF-8 or ISO-8859-1
  - In Java, when you read the contents of a file into a String, implicit conversion happens

    By default, Java assumes the encoding of the shell from which you have started the program

    For ES7, therefore read into a byte[] array first

References

UNI FREIBURG

- CORS
  - http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
  - http://en.wikipedia.org/wiki/Cross-site_scripting
- Cookies
  - http://en.wikipedia.org/wiki/HTTP_cookie
  - http://www.w3schools.com/js/js_cookies.asp
- UTF-8, URL-encoding and -decoding
  - http://en.wikipedia.org/wiki/UTF-8
  - http://www.utf8-chartable.de
  - http://www.w3schools.com/tags/ref_urlencode.asp