Internationalization

In the Spring framework

EHSDI

eBuzima

Why internationalization?

- An application such as OpenMRS needs to be easily configured to work in different countries
- This requires easy switching of..
  - The interface language
  - The date formatting etc

Why internationalization?

- If strings are hard coded in the application, then it will be impossible to change languages without rewriting the code
- We want to avoid having different versions of source files for different languages, e.g.

login_english.jsp

```
...
<p>Welcome to OpenMRS</p>
...
```

login_french.jsp

```
...
<p>Bienvenue a OpenMRS</p>
...
```

Language classification

- There needs to be a way of labeling text as being a particular language that a computer program can easily understand

- **ISO 639**: international standards for language names and codes:
  - **ISO 639-1**: 185 languages identified by 2 letter codes (e.g. "en", "fr", "rw")
  - **ISO 639-3**: 7589 languages identified by 3 letter codes (e.g. "eng", "fra", "kin")

Region classification

- Different countries might speak the same language, but...
  - There might be small variations
  - Other things like date formats might be different

- **ISO-3166**: international standards for country, province, etc codes
  - **ISO 3166-1 alpha-2**: countries identified by 2 letter codes (e.g. "`GB`", "`FR`", "`RW`")
  - **ISO 3166-1 alpha-3**: countries identified by 3 letter codes (e.g. "`GBR`", "`FRA`", "`RWA`")

IETF language 'tags'

- This is the standard for describing the language of HTML or XML data
- Uses a combination of **ISO 639** language codes and **ISO 3166** region codes
- List of valid 'tags' maintained by IANA e.g.
  - `en`: English
  - `en-US`: English as spoken in USA
  - `en-RW`: English as spoken in Rwanda
  - `rw`: Kinyarwanda
  - `rw-GB`: Kinyarwanda as spoken in the UK

IETF language 'tags'

- We can specify language in XML using the `xml:lang` attribute and in HTML using `lang`
- For example:

```
<p lang="en">Welcome to OpenMRS</p>
```

- This can even be matched in CSS, e.g.

```
:lang(en) { color: red }
```

Rule only applied to elements where language is "*en*"

IETF language 'tags'

- A less specific tag will match a more specific tag, but not vice versa, i.e.

```
<p lang="en-GB">Welcome to OpenMRS</p>
```

```
<p lang="en-US">Welcome to OpenMRS! Yeehar!</p>
```

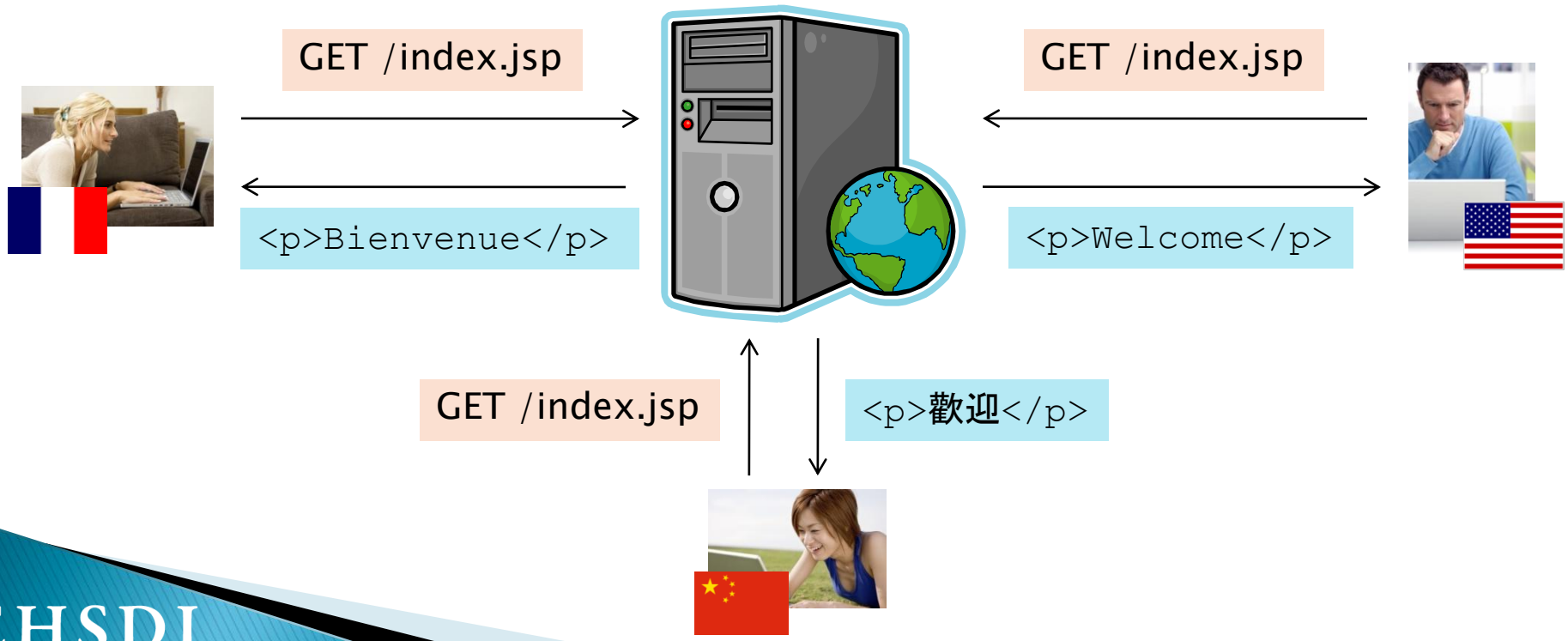
- are both matched by `en`, but

```
<p lang="en">Welcome to OpenMRS</p>
```

is not matched by `en-US`, `en-GB` etc

Server-side internationalization

- The content of our application should be customized for a locale (language and region) on the server, e.g.



Using Spring

- Spring makes it easier to add this kind of server-side internationalization to our web apps...

- Supports using separate language files for text displayed on a web page
- Can automate the process of locale detection

Message sources

- Spring uses message codes – instead of writing visible text in a JSP – we specify a message code
- We then give the language dependent value of the code in a separate ***message source***, e.g.

header.jsp

```
<p>Log in</p>
```

```
<p><spring:message code="header.login" /></p>
```

messages.properties

```
...
header.login=Log in
...
```

Message sources

- Spring provides a bean called `ResourceBundleMessageSource` which manages loading of message sources

```
<bean id="messageSource" class="org...ResourceBundleMessageSource">
  <property name="basename"><value>messages</value></property>
</bean>
```

This tells spring that the default message source is called *messages.properties*

messages.properties

```
# This is a comment
header.login=Log in
header.logout=Log out
header.help=Help
...
```

Message sources

- Message sources for other locales should use the `basename` value plus the locale code
- Spring can detect the locale of the current user, and load the correct source file
- The values in these will override the values in the default source

messages_fr.properties

```
header.login=Entrer
header.logout=Sortir
header.help=Aide
...
```

messages_en_GB.properties

```
header.login=Log in
header.logout=Log out
header.help=Help please
...
```

messages_rw.properties

```
header.login=Kwinjira
header.logout=Gusohoka
header.help=Gufasha
...
```

Message sources

GET /login.jsp

Entrer

GET /login.jsp

Log in

<spring:message code="header.login" />

ResourceBundleMessageSource

messages.properties

messages_fr.properties

messages.properties

Locale resolving

- Choosing the correct message source requires choice of a locale
- Spring looks for a `LocaleResolver` bean to make this choice

- `SessionLocaleResolver` – uses a session attribute to store the locale
- `CookieLocaleResolver` – uses a cookie value to store the locale

Message arguments

- What can we do when page text contains a dynamic value? (e.g. EL expression)
- For example:

```
<p>Search returned ${count} results</p>
```

we could split this into two messages...

```
<p>
  <spring:message code="search.resultsStart" />
  ${count}
  <spring:message code="search.resultsEnd" />
</p>
```

"Search returned"

"results"

Message arguments

- But a better way is to put an argument in the message
- Arguments are inserted into the message text as `{0}`, `{1}`, `{2}` etc, e.g.

```
<p>
<spring:message code="search.results" arguments="${count}" />
</p>
```

↓

```
"Search returned {0} results"
```

- ... if `${count}` is `5` then the message becomes

```
"Search returned 5 results"
```

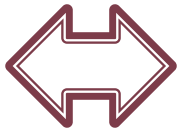
Detecting user's locale

- Locale can often be automatically detected using the `AcceptHeaderLocaleResolver` bean which examines the HTTP header
- Most browsers send an IETF language tag in the HTTP header, e.g.

```
GET /index.jsp
Accept-Language: en-us,en
...
```

Client is asking for US English, if possible, if not any kind of English will do

Switching locales

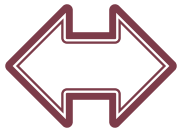


- Adding a `LocaleChangeInterceptor` bean means that locale switching can be performed using a request parameter
- Bean is attached as an *interceptor* to a URL mapping bean, e.g.

```
<bean id="localeChangeInterceptor" class="org...LocaleChangeInterceptor"/>

<bean id="urlMapping" class="org...SimpleUrlHandlerMapping">
  <property name="interceptors">
    <list><ref local="localeChangeInterceptor"/></list>
  </property>
  <property name="mappings">
    <props> ... </props>
  </property>
</bean>
```

Switching locales



- Now any URL containing the request parameter locale will switch the locale, e.g.

```
http://localhost/myapp?locale=fr
```

will switch the locale to French

- We can get the locale in Java code using

```
Locale locale = RequestContext.getLocale();
```

References

- Websites
  - http://www.w3.org/International/articles/language-tags/Overview.en.php
  - http://www.langtag.net/registries.html – IANA registry of language subtags