Locales as part of the URL in Spring MVC

Asked 13 years, 11 months ago Modified 4 years, 4 months ago Viewed 13k times

22

I'm now looking for a framework for multilingual web-applications. At the moment it seems to me that the best choice is Spring MVC. But I faced the fact that all the guidelines for developers suggests to switch languages using LocaleChangeInterceptor in such way:

http://www.somesite.com/action/?locale=en

Unfortunately, there are a number of reasons why I would like avoid this. How could I make language code to be an essential part of URL? For example:

http://www.somesite.com/en/action

Thanks.

UPD: I've found following solution. It's not complete yet, but works. Solution consists in two parts - servlet filter and locale resolver bean. It's looks little bit hackish, but I do not see other way to solve this problem.

```
public class LocaleFilter implements Filter
{

    ...

    private static final String DEFAULT_LOCALE = "en";
    private static final String[] AVAILABLE_LOCALES = new String[] {"en", "ru"};

    public LocaleFilter() {}

    private List<String> getSevletRequestParts(ServletRequest request)
    {
        String[] splitedParts = ((HttpServletRequest)
request).getServletPath().split("/");
        List<String> result = new ArrayList<String>();

        for (String sp : splitedParts)
        {
            if (sp.trim().length() > 0)
                result.add(sp);
        }

        return result;
    }

    private Locale getLocaleFromRequestParts(List<String> parts)
    {
        if (parts.size() > 0)
        {
            for (String lang : AVAILABLE_LOCALES)
            {
                if (lang.equals(parts.get(0)))
                {
                    return new Locale(lang);
                }
            }
        }

        return null;
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws IOException, ServletException
    {
        List<String> requestParts = this.getSevletRequestParts(request);
        Locale locale = this.getLocaleFromRequestParts(requestParts);

        if (locale != null)
        {
            request.setAttribute(LocaleFilter.class.getName() + ".LOCALE",
locale);

            StringBuilder sb = new StringBuilder();
            for (int i = 1; i < requestParts.size(); i++)
            {
                sb.append('/');
                sb.append((String) requestParts.get(i));
            }

            RequestDispatcher dispatcher =
request.getRequestDispatcher(sb.toString());
            dispatcher.forward(request, response);
        }
        else
        {
            request.setAttribute(LocaleFilter.class.getName() + ".LOCALE", new
Locale(DEFAULT_LOCALE));
            chain.doFilter(request, response);
        }
    }

    ...
}

public class FilterLocaleResolver implements LocaleResolver
{

    private Locale DEFAULT_LOCALE = new Locale("en");

    @Override
    public Locale resolveLocale(HttpServletRequest request)
    {
        Locale locale = (Locale)
request.getAttribute(LocaleFilter.class.getName() + ".LOCALE");
        return (locale != null ? locale : DEFAULT_LOCALE);
    }

    @Override
    public void setLocale(HttpServletRequest request, HttpServletResponse
response, Locale locale)
    {
        request.setAttribute(LocaleFilter.class.getName() + ".LOCALE", locale);
    }

}
```

So there is no need to map locale in each action in controllers. The following example will work fine:

```
@Controller
@RequestMapping("/test")
public class TestController
{

    @RequestMapping("action")
    public ModelAndView action(HttpServletRequest request, HttpServletResponse
response)
    {
        ModelAndView mav = new ModelAndView("test/action");
        ...
        return mav;
    }

}
```

url spring-mvc locale

Share Edit Follow

edited Jul 11, 2010 at 17:41

asked Jul 11, 2010 at 2:32
Gris
323 ●1 ●3 ●10

Hi @Gris... Have you polished this solution? – user683887 Oct 14, 2011 at 22:08

2 Also wondering if you have polished the solution – Piotr May 9, 2012 at 13:53

I do not understand how you set the locale in the setLocale method, you only set an attribute in the request if I have understood it right... Could you clarify? – 0m4r Nov 29, 2013 at 17:45

Keep an eye out for this issue - github.com/spring-projects/spring-framework/issues/25791 – Julius Krah Sep 20, 2020 at 10:16

5 Answers

Sorted by: Highest score (default)

12

I implemented something very similar using a combination of Filter and Interceptor.

The filter extracts the first path variable and, if it's a valid locale it sets it as a request attribute, strips it from the beginning of the requested URI and forward the request to the new URI.

```
public class PathVariableLocaleFilter extends OncePerRequestFilter {
private static final Logger LOG =
LoggerFactory.getLogger(PathVariableLocaleFilter.class);

@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain)
        throws ServletException, IOException {
    String url =
defaultString(request.getRequestURI().substring(request.getContextPath().length()));
    String[] variables = url.split("/");

    if (variables.length > 1 && isLocale(variables[1])) {
            LOG.debug("Found locale {}", variables[1]);
            request.setAttribute(LOCALE_ATTRIBUTE_NAME, variables[1]);
            String newUrl = StringUtils.removeStart(url, '/' + variables[1]);
            LOG.trace("Dispatching to new url \'{}\'", newUrl);
            RequestDispatcher dispatcher = request.getRequestDispatcher(newUrl);
            dispatcher.forward(request, response);
        } else {
            filterChain.doFilter(request, response);
        }
    }

    private boolean isLocale(String locale) {
        //validate the string here against an accepted list of locales or whatever
        try {
            LocaleUtils.toLocale(locale);
            return true;
        } catch (IllegalArgumentException e) {
            LOG.trace("Variable \'{}\' is not a Locale", locale);
        }
        return false;
    }
}
```

The interceptor is very similar to the `LocaleChangeInterceptor`, it tries to get the locale from the request attribute and, if the locale is found, it sets it to the `LocaleResolver`.

```
public class LocaleAttributeChangeInterceptor extends HandlerInterceptorAdapter {
public static final String LOCALE_ATTRIBUTE_NAME =
LocaleAttributeChangeInterceptor.class.getName() + ".LOCALE";

@Override
public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) {

    Object newLocale = request.getAttribute(LOCALE_ATTRIBUTE_NAME);
    if (newLocale != null) {
        LocaleResolver localeResolver =
RequestContextUtils.getLocaleResolver(request);
        if (localeResolver == null) {
            throw new IllegalStateException("No LocaleResolver found: not in a
DispatcherServlet request?");
        }
        localeResolver.setLocale(request, response,
StringUtils.parseLocaleString(newLocale.toString()));
    }
    // Proceed in any case.
    return true;
}
}
```

Once you have them in place you need to configure Spring to use the interceptor and a `LocaleResolver`.

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new LocaleAttributeChangeInterceptor());
}

@Bean(name = "localeResolver")
public LocaleResolver getLocaleResolver() {
    return new CookieLocaleResolver();
}
```

And add the filter to the `AbstractAnnotationConfigDispatcherServletInitializer`.

```
@Override
protected Filter[] getServletFilters() {
    return new Filter[] { new PathVariableLocaleFilter() };
}
```

I haven't tested it thoroughly but it seems working so far and you don't have to touch your controllers to accept a `{locale}` path variable, it should just work out of the box. Maybe in the future we'll have 'locale as path variable/subfolder' Spring automagic solution as it seems more and more websites are adopting it and according to some it's the way to go.

Share Edit Follow

edited May 23, 2017 at 12:18
Community Bot
1 ● 1

answered May 24, 2014 at 16:51
Andrea Vacondio
904 ● 10 ● 19

Nice solution. I had something similar in mind while I was searching for already used solutions. Your approach confirms I'm on the right track. – Gregor Koukkoullis Jun 4, 2014 at 15:59

@Andrea how would be `RequestMapping` in `Controller` ? – exezxian Apr 28, 2017 at 21:11

This is the best solution so far for my requirements. I only need to add a redirect when there is no language specified (so I don't have multiple urls for the same page), but only for documents, not for resources (which don't change with language). – xtian Jul 25, 2017 at 13:41

which name of language file should be? – jhenya-d Jul 12, 2020 at 7:05

6

I found myself in the same problem and after do a lot of research I finally manage to do it also using a Filter and a LocaleResolver. A step for step guide:

First set the Filter in the **web.xml**:

```
<filter>
    <filter-name>LocaleFilter</filter-name>
    <filter-class>yourCompleteRouteToTheFilter.LocaleUrlFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>LocaleFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

In the **LocaleUrlFilter.java** we use regex to:

- add two attributes (Country code and Language Code) to the request that we will capture later on the LocaleResolver:
- strip the language from the url

```
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;

public class LocaleUrlFilter implements Filter{

    private static final Pattern localePattern = Pattern.compile("^/([a-z]{2})
(?:/([a-z]{2}))?(/.*)?");
    public static final String COUNTRY_CODE_ATTRIBUTE_NAME =
LocaleUrlFilter.class.getName() + ".country";
    public static final String LANGUAGE_CODE_ATTRIBUTE_NAME =
LocaleUrlFilter.class.getName() + ".language";

    @Override
    public void init(FilterConfig arg0) throws ServletException {}

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        String url =
request.getRequestURI().substring(request.getContextPath().length());
        Matcher matcher = localePattern.matcher(url);
        if (matcher.matches()) {
            // Set the language attributes that we will use in LocaleResolver and
strip the language from the url
            request.setAttribute(COUNTRY_CODE_ATTRIBUTE_NAME, matcher.group(1));
            request.setAttribute(LANGUAGE_CODE_ATTRIBUTE_NAME, matcher.group(2));
            request.getRequestDispatcher(matcher.group(3) == null ? "/" :
matcher.group(3)).forward(servletRequest, servletResponse);
        }
        else filterChain.doFilter(servletRequest, servletResponse);
    }

    @Override
    public void destroy() {}
}
```

Now the filter injected to the request two attributes that we will use to form the Locale and stripped the language from url to correctly process our requests. Now we will define a LocaleResolver to change the locale. For that first we modify our **servlet.xml** file:

```
<!-- locale Resolver configuration-->
<bean id="localeResolver"
class="yourCompleteRouteToTheResolver.CustomLocaleResolver"></bean>
```

And in the CustomLocaleResolver.java we set the language accordingly. If there is no Language in the url we proceed using the getLocale method of the request:

```
import java.util.Locale;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.LocaleResolver;

/*
* Set the Locale defined in the LocaleUrlFiltes. If none is defined (in the url)
return the request locale.
*/
public class CustomLocaleResolver implements LocaleResolver{

    @Override
    public Locale resolveLocale(HttpServletRequest servletRequest) {
        final String countryCode =
(String)servletRequest.getAttribute(LocaleUrlFilter.COUNTRY_CODE_ATTRIBUTE_NAME);
        if (countryCode != null) {
            String languageCode =
(String)servletRequest.getAttribute(LocaleUrlFilter.LANGUAGE_CODE_ATTRIBUTE_NAME);
            if (languageCode == null) {
                return new Locale(countryCode);
            }
            return new Locale(languageCode, countryCode);
        }
        return servletRequest.getLocale();
    }

    @Override
    public void setLocale(final HttpServletRequest servletRequest, final
HttpServletResponse servletResponse, final Locale locale) {
        throw new UnsupportedOperationException();
    }

}
```

Doing this you won't need to change anything in your controllers and visiting "/en/home" will be the same as visiting "/home" and using your language_en.properties file. Hope it helps

Share Edit Follow

answered Sep 21, 2014 at 14:33

Joan

239 ●4 ●6

Thanks for the answer. As I am using annotations and not xml configuration. I have inserted this @Bean(name = "localeResolver") public LocaleResolver getLocaleResolver() { return new CustomLocaleResolver(); } into my AppConfig instead of servlet.xml and web.xml – Black May 15, 2016 at 13:16

6

I came across very same problem recently. So I would like to have stateless locale not depending on session or cookie or anything else than simply URL.

I tried filter/interceptor/localeResolver solutions suggested in previous answers however these did not suite my needs as I had:

- static content (images etc ..)
- parts of page not locale dependent (admin panel)
- RestController inside same app

- multipart file uploader

I also wanted to avoid duplicated content for SEO reasons (In particular I do not want my english content to be accessible from both paths: /landingPage and /en/landingPage).

The solution that worked best for me was to create LanguageAwareController and then inherit from it in all controllers that I wanted to support multiple locales.

```
@Controller
@RequestMapping(path = "/{lang}")
public class LanguageAwareController {
    @Autowired
    LocaleResolver localeResolver;

    @ModelAttribute(name = "locale")
    Locale getLocale(@PathVariable(name = "lang") String lang, HttpServletRequest
request,
                     HttpServletResponse response){
        Locale effectiveLocale = Arrays.stream(Locale.getAvailableLocales())
            .filter(locale -> locale.getLanguage().equals(lang))
            .findFirst()
            .orElseGet(Locale::getDefault);
        localeResolver.setLocale(request, response, effectiveLocale);
        return effectiveLocale;
    }
}
```

Usage in one of controllers:

```
@Controller
public class LandingPageController extends LanguageAwareController{

    private Log log = LogFactory.getLog(LandingPageController.class);

    @GetMapping("/")
    public String welcomePage(Locale locale, @PathVariable(name = "lang") String
lang ){
        log.info(lang);
        log.info(locale);
        return "landing";
    }
}
```

Share Edit Follow

answered Mar 28, 2017 at 20:51

JJ Roman
4,472 ● 1 ● 28 ● 22

1

In spring 3.0 you can tell your controllers to look for path variables. e.g.

```
@RequestMapping("/{locale}/action")
public void action(@PathVariable String locale) {
    ...
}
```

Share Edit Follow

edited Jul 11, 2010 at 2:44

answered Jul 11, 2010 at 2:39

krock
29.4k ● 13 ● 80 ● 86

4 But in this case I have to manually switch the locale in each action. Is it possible to make an automatic switching, for example, using a filter? – Gris Jul 11, 2010 at 2:51

1

In addition to the provided answers here's a way how to let Thymeleaf prepend the locale in path after context path automatically by implementing a `ILinkBuilder`:

```
@Bean
public ILinkBuilder pathVariableLocaleLinkBuilder() {
    PathVariableLocaleLinkBuilder pathVariableLocaleLinkBuilder = new
PathVariableLocaleLinkBuilder();
    pathVariableLocaleLinkBuilder.setOrder(1);
    return pathVariableLocaleLinkBuilder;
}

@Bean
SpringTemplateEngine templateEngine(ThymeleafProperties properties,
ObjectProvider<ITemplateResolver> templateResolvers, ObjectProvider<IDialect>
dialects, ObjectProvider<ILinkBuilder> linkBuilders) {
    SpringTemplateEngine engine = new SpringTemplateEngine();
    engine.setEnableSpringELCompiler(properties.isEnableSpringElCompiler());

engine.setRenderHiddenMarkersBeforeCheckboxes(properties.isRenderHiddenMarkersBefore
    templateResolvers.orderedStream().forEach(engine::addTemplateResolver);
    dialects.orderedStream().forEach(engine::addDialect);
    linkBuilders.orderedStream().forEach(engine::addLinkBuilder);
    return engine;
}
```

And here's the LinkBuilder itself:

```
public class PathVariableLocaleLinkBuilder extends AbstractLinkBuilder {
    @Autowired
    private LocaleResolver localeResolver;

    @Override
    public String buildLink(IExpressionContext context, String base, Map<String,
Object> parameters) {
        Validate.notNull(context, "Expression context cannot be null");

        if (base == null) {
            return null;
        }

        if (!isLinkBaseContextRelative(base)) {
            return base;
        }

        if (!(context instanceof IWebContext)) {
            throw new TemplateProcessingException(
                    "Link base \"" + base + "\" cannot be context relative (/...)
unless the context " +
                            "used for executing the engine implements the " +
IWebContext.class.getName() + " interface");
```

```
        }

        final HttpServletRequest request = ((IWebContext) context).getRequest();
        return "/" + localeResolver.resolveLocale(request) + base;
    }

    private static boolean isLinkBaseContextRelative(final CharSequence linkBase)
{
        if (linkBase.length() == 0 || linkBase.charAt(0) != '/') {
            return false;
        }

        return linkBase.length() == 1 || linkBase.charAt(1) != '/';
    }
}
```

Share Edit Follow

edited Feb 9, 2020 at 12:55

answered Feb 6, 2020 at 21:27

dtrunk
4,755 ● 17 ● 69 ● 110