



Igor Drobiazko

# Tapestry 5

Die Entwicklung von Webanwendungen  
mit Leichtigkeit



ADDISON-WESLEY





# 6 Lokalisierung

Bei einer sprachen- und kulturunabhängigen Gestaltung von Webseiten müssen mehrere Aspekte beachtet werden. Dazu gehören unter anderem:

- Übersetzung von Texten.
- Übersetzung von Grafiken, die Texte enthalten.
- Anpassung der Layouts. Beispielsweise besitzen die meisten semitischen Sprachen wie Hebräisch oder Arabisch linksläufige Schriften, d.h., sie werden von rechts nach links geschrieben.
- Beachten der lokalen Zahlen- und Datumsformatierung.

Für die Realisierung dieser Aspekte unterscheidet man in der Softwareentwicklung zwischen Internationalisierung und Lokalisierung:

- Internationalisierung (*I18N*): Design und Entwicklung einer Software derart, dass eine Lokalisierung einfach möglich wird.
- Lokalisierung (*L10N*): Der Prozess der Anpassung der Software an die sprachlichen und kulturellen Gegebenheiten. Zu Lokalisierung gehört beispielsweise die Übersetzung einer Software in eine neue Sprache.

## 6.1 Internationalisierung und Lokalisierung in Java

Um eine Anwendung sprachenunabhängig zu gestalten, müssen alle Zeichenketten, die einem Anwender präsentiert werden, im Java-Quellcode durch symbolische Namen ersetzt werden. Diese Namen werden dann mit Zeichenketten einer Landessprache in Ressourcendateien verknüpft. Die Ressourcendateien enthalten eine Menge von Schlüssel-Wert-Paaren und sind üblicherweise an der Erweiterung *\*.properties* zu erkennen. Auch XML-Format ist möglich. Für den Zugriff auf eine Ressourcendatei ist in Java die Klasse `ResourceBundle` zuständig. Wie in Listing 6.1 zu sehen ist, wird ein `ResourceBundle` über den Aufruf der statischen Methode `getBundle()` erzeugt, in die der Name einer Ressourcendatei als Parameter übergeben wird. In Tapestry müssen Sie dies nicht manuell durchführen. Wie an anderen Stellen auch werden Sie sehen, dass Tapestry hier sinnvolle Konventionen einsetzt und wieder das Prinzip *Convention over Configuration* gilt.

### Listing 6.1: Arbeiten mit ResourceBundle

```
ResourceBundle bundle = ResourceBundle.getBundle("MyStrings");
System.out.println(bundle.getString("welcome"));
```

## 6.2 Anwendungsweiter Nachrichtenatalog

In Tapestry werden die zu einer Komponente gehörenden Ressourcendateien als *Nachrichtenatalog* bezeichnet. Für die Verwendung von Nachrichten, die für die gesamte Anwendung gelten, wird der sogenannte *Anwendungs-Nachrichtenatalog* benutzt. Der Name der Ressourcendateien dieses Katalogs setzt sich aus dem Namen des *TapestryFilters*, der in *web.xml* vergeben wurde, und gegebenenfalls einem Locale-Suffix zusammen. Wurde dem Filter der Namen *app* vergeben, so könnten die Ressourcendateien des Applikations-Nachrichtenatalog *app.properties*, *app\_de.properties* (oder auch *app\_de\_DE.properties*) usw. heißen. Diese Dateien müssen im Verzeichnis *WEB-INF* der Anwendung abgelegt werden.

### Listing 6.2: WEB-INF/app.properties

```
welcome=Welcome
```

### Listing 6.3: WEB-INF/app\_ru.properties

```
welcome=Добро пожаловать
```

## 6.3 Komponenten-Nachrichtenatalog

Jede Seite und jede Komponente kann ihren eigenen *Nachrichtenatalog* besitzen, der von Tapestry als Erstes nach einem Wert für einen Schlüssel durchsucht wird. Falls kein Eintrag gefunden wird, sucht Tapestry im *Anwendungs-Nachrichtenatalog* weiter. Damit können Seiten und Komponenten die Schlüssel-Werte-Paare des *Anwendungs-Nachrichtenatalogs* überschreiben. Der Namen der Ressourcendateien eines *Nachrichtenatalogs* setzt sich aus dem Namen der jeweiligen Seiten- oder Komponentenklasse und gegebenenfalls einem Locale-Suffix zusammen. So könnte es für die Seite *MyPage* Übersetzungsdateien wie *MyPage.properties*, *MyPage\_de.properties* usw. geben.

## 6.4 Lokalisierte Templates

Es gibt viele Schriftkulturen, die eine andere Schreibrichtung als die lateinische Schrift besitzen. So wird beispielsweise in Hebräisch von rechts nach links geschrieben. Eine Lokalisierung Ihrer Anwendung für Hebräisch ist durch Übersetzung der Inhalte noch lange nicht abgeschlossen. Sie müssen zusätzlich das Layout Ihrer Anwendung so anpassen, dass alle Inhalte rechtsbündig sind. Auch eine Navigation, die Sie üblicherweise auf dem linken Seitenrand erwarten würden, müssten Sie rechtsbündig ausrichten.

Wenn Tapestry ein Template einer Seite zum Erzeugen des Markups benötigt, versucht es, zunächst eine lokalisierte Version dieser Seite zu finden. Ein lokalisiertes Template enthält als Suffix ein `Locale` in ihrem Namen. So ist in Listing 6.6 eine hebräische Version des Templates der Seite `MyPage` zu sehen. An den Namen dieser Datei ist das `Locale` `iw` angefügt. Diese Version unterscheidet sich von der englischen (Listing 6.7) nur minimal: Die Schrift ist rechtsbündig. Falls keine lokalisierte Version eines Templates gefunden wird, nimmt Tapestry das Standard-Template, d. h. das Template ohne Suffix.

*Listing 6.4: MyPage.properties*

```
hello-world=Hello, World!
```

*Listing 6.5: MyPage\_iw.properties*

```
hello-world=\u05E2\u05D5\u05DC\u05DD\u002C\u05E9\u05DC\u05D5\u05DD
```

*Listing 6.6: MyPage\_iw.tml (hebräische Version)*

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_1_0.xsd">
  <body style="text-align:right">
    ${message:hello-world}
  </body>
</html>
```

*Listing 6.7: MyPage.tml*

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_1_0.xsd">
  <body>
    ${message:hello-world}
  </body>
</html>
```

In der Abbildung 6.1 ist das Ergebnis einer Anfrage an die Seite `MyPage` dargestellt, die zwei Benutzer mit unterschiedlichen Spracheinstellungen im Browser sehen würden.

## 6.5 Zugreifen auf den Nachrichtenkatalog

Oft muss man auf die übersetzten Nachrichten nicht nur aus dem Template heraus zugreifen, sondern auch aus einer Seiten- oder Komponentenklasse. Dafür stellt Tapestry den Dienst `Messages` bereit, der Ihnen das manuelle Laden eines `ResourceBundles` abnimmt. Sie können diesen Dienst wie in Listing 6.8 mittels `@Inject` injizieren und auf die Nachrichten über zwei Methoden zugreifen. Die Methode `get()` sucht in dem *Nachrichtenkatalog* nach einem Wert für den angegebenen Schlüssel. Bei der Verwendung der Methode `format()` wird die Nachricht als ein Format-String mit Argumenten, wie er von `java.util.Formatter` verlangt wird, interpretiert.

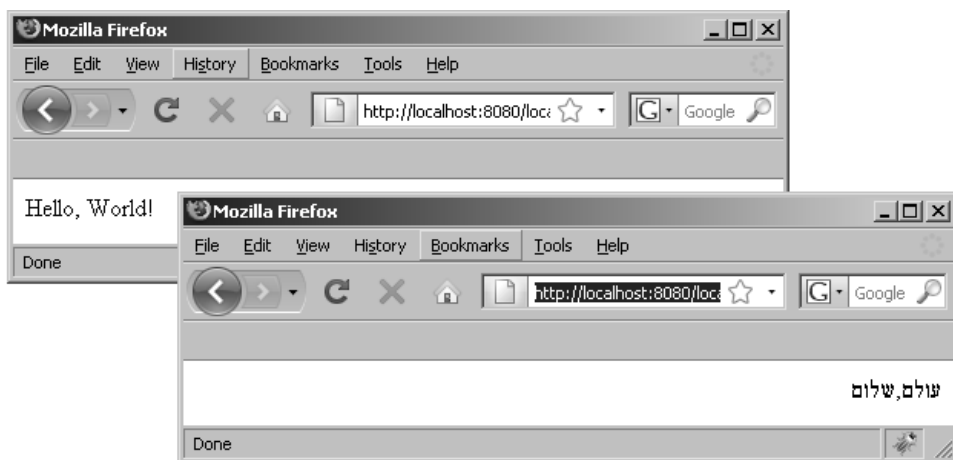


Abbildung 6.1: Hebräische und englische Versionen der Seite MyPage

Anders als bei Standardmechanismen von Java zum Auslesen von lokalisierten Nachrichten, müssen Sie ein `ResourceBundle` nicht manuell erzeugen. Außerdem ist es nicht notwendig, den Namen der Ressourcendatei anzugeben. Tapestry verknüpft eine Seite bzw. Komponente mit dem dazugehörigen *Nachrichtenkatalog* über den Namen.

Listing 6.8: Zugreifen auf den Nachrichtenkatalog aus der Seite MyPage heraus

```
public class MyPage {
    ...

    @Inject
    private Messages messages;

    String getShoppingCartSummary(){
        if(shoppingCart.isEmpty()){
            return messages.get("cart-empty");
        }
        return messages.format("cart-size", shoppingCart.size());
    }
}
```

Listing 6.9: MyPage\_de.properties

```
cart-empty=Ihr Warenkorb ist leer.
cart-size=Sie haben %d Waren in Ihrem Korb.
```

## 6.6 Unterstützte Sprachen

Die Suche nach einer passenden Übersetzung eines Schlüssels innerhalb eines *Nachrichtenkatalogs* sowie die Suche nach einer lokalisierten Version eines Templates ist in Tapestry auf die Menge der Sprachen eingeschränkt, die innerhalb des IoC-Containers konfiguriert sind. Zurzeit sind in Tapestry folgende `Locales` vorkonfiguriert:

### Listing 6.10: Unterstützte Locales

```
en, it, es, zh_CN, pt_PT, de, ru, hr,
fi_FI, sv_SE, fr_FR, da, pt_BR, ja, el
```

Für jede dieser `Locale` besitzt das Framework Übersetzungen für alle Nachrichten, die Tapestry einem Benutzer präsentiert. Darunter sind beispielsweise Texte, die von Komponenten präsentiert werden, und Validierungsnachrichten von Formularen. Falls ein Benutzer mit einer `Locale`, die nicht in dieser Liste vertreten ist, eine Anfrage abschickt, versucht Tapestry, die nächste passende Sprache auszuwählen. Wird beispielsweise eine Anfrage einem Benutzer mit der Spracheinstellung `de_CH` gestellt, wird die Sprache `de` ausgewählt. Falls keine passende `Locale` gefunden wird, werden ihm die Übersetzungen der Standardsprache präsentiert. Die Standardsprache ist die erste `Locale` in dieser Liste, also `en`.

Sie können die Liste der unterstützten Sprachen überschreiben, indem Sie eine *Contribute-Methode* innerhalb Ihres Applikationsmoduls wie in Listing 6.11 schreiben. Kopieren Sie diesen Quellcode in Ihr *Modul*, und geben Sie die Liste Ihrer gewünschten Sprachen in Form einer durch Komma separierten Liste von `Locales` an.

### Listing 6.11: Überschreiben der Menge der unterstützten Locales

```
public class AppModule {
    public static void contributeApplicationDefaults(
        MappedConfiguration<String, String> configuration) {

        configuration.add(SymbolConstants.SUPPORTED_LOCALES,
            "en,de,ru,iw");
    }
}
```

## 6.7 Lokalisierung statischer Ressourcen

Jede Webanwendung besitzt statische Ressourcen wie Bilder, Cascading Style Sheets (CSS) und JavaScript-Bibliotheken, die in Form von Referenzen auf die jeweiligen Dateien in den HTML-Code eingebunden werden können (siehe Listing 6.12). Dabei gibt es zwei Möglichkeiten, eine Referenz anzugeben: durch relative oder absolute URLs. Bei der Angabe von absoluten Pfaden besteht das Problem, dass Sie die

Adresse einer Webseite in einem Template hart kodieren oder zur Laufzeit herausfinden müssen. Eine relative Pfadangabe ist mit der Gefahr verbunden, dass die Referenzen auf die Ressourcen bei einem Verschieben der Seiten nicht mehr funktionieren.

#### Listing 6.12: Einbinden statischer Ressourcen

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_1_0.xsd">
  <head>
    <title>Tapestry 5 Buch</title>
    <link rel="stylesheet" type="text/css" href="/css/main.css">
    <script type="text/javascript" src="/js/lib.js"/>
  </head>
  <body>
    
  </body>
</html>
```

Um dieses Problem umzugehen, gibt es in Tapestry sogenannte *Assets*. Ein *Asset* ist ein Java-Objekt, das eine Referenz auf eine Ressource darstellt und in Seiten oder Komponenten mittels `@Inject` injiziert werden kann. Die Annotation `@Path` dient der Angabe des Pfades eines *Assets*. In Listing 6.13 werden zwei *Assets* injiziert: das Logo der Anwendung und eine CSS-Datei. Das Präfix `context` informiert Tapestry, dass der Pfad `images/logo.gif` im Wurzelverzeichnis der Anwendung zu finden ist. Das Präfix `classpath` wird dagegen eingesetzt, um eine Ressource aus dem Klassenpfad (beispielsweise aus einem Java-Archiv) zu referenzieren. Falls kein Präfix angegeben wird, wird das Standardpräfix `classpath` verwendet, wobei der Wert innerhalb der Annotation `@Path` als ein relativer Pfad zu der jeweiligen Klasse interpretiert wird.

#### Listing 6.13: Injizieren von Assets

```
public class Index{
    @Inject
    @Path("context:images/logo.gif")
    @Property
    private Asset logo;

    @Inject
    @Path("classpath:/css/main.css ")
    @Property
    private Asset css;

    @Inject
    @Path("/js/lib.js")
    @Property
    private Asset jsLib;
}
```

Um ein `Asset` in ein Template einzubinden, benutzt man *Expansions*. Dazu muss entweder eine Getter-Methode bereitgestellt oder die jeweilige Eigenschaft mit `@Property` annotiert werden. Wenn Tapestry diese *Expansion* auswertet, wird das Framework erkennen, dass es sich um ein `Asset` handelt, und eine entsprechende URL generieren. Ein Beispiel ist in Listing 6.14 zu sehen.

*Listing 6.14: Einbinden von Assets in ein Template*

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_1_0.xsd">
  <head>
    <title>Tapestry 5 Buch</title>
    <link rel="stylesheet" type="text/css" href="\${css}">
    <script type="text/javascript" src="\${jsLib}"/>
  </head>
  <body>
    
  </body>
</html>
```

Falls Sie die `Assets` nicht in Eigenschaften Ihrer Seiten oder Komponente injizieren möchten, können Sie die beiden Annotationen `@IncludeJavaScriptLibrary` und `@IncludeStyleSheet` wie in Listing 6.15 verwenden. In diesem Fall müssen Sie die jeweiligen Ressourcen im Template nicht explizit einbinden. Tapestry wird dies für Sie übernehmen. Sollten mehrere Komponenten innerhalb derselben Seite die gleiche JavaScript-Bibliothek oder CSS-Datei mittels dieser Annotationen einfügen, wird Tapestry dafür sorgen, dass die jeweiligen Referenzen nur einmalig im erzeugten Markup auftauchen.

*Listing 6.15: Alternative Möglichkeit zum Einbinden von Assets*

```
@IncludeJavaScriptLibrary("js/lib.js")
@IncludeStyleSheet("classpath:/css/main.css")
public class Index{
    ...
}
```

Einer der großen Vorteile von `Assets` ist, dass sie wie die Inhalte eines *Nachrichtenkatalogs* lokalisiert werden können. Damit Tapestry eine passende Grafik für die Sprache des Benutzers darstellt, müssen Sie lediglich die Namenskonvention für die Benennung von `Assets` befolgen, die der der Ressourcendateien eines *Nachrichtenkatalogs* entspricht. In der Abbildung 6.2 ist zu sehen, dass die Seite *WikiLogo* (Listing 6.16) ein Bild enthält. Einem Benutzer mit der Spracheinstellung `de` wird das Wiki-Logo mit dem deutschen Untertitel präsentiert (*wiki\_de.png*). Ein Benutzer mit der `Locale ru` sieht dagegen ein Logo mit einem russischen Untertitel (*wiki\_ru.png*).

*Listing 6.16: Lokalisierung der Assets*

```
public class WikiLogo {
    @Inject
    @Path("wiki.png")
    @Property
    private Asset wikiLogo;
}
```

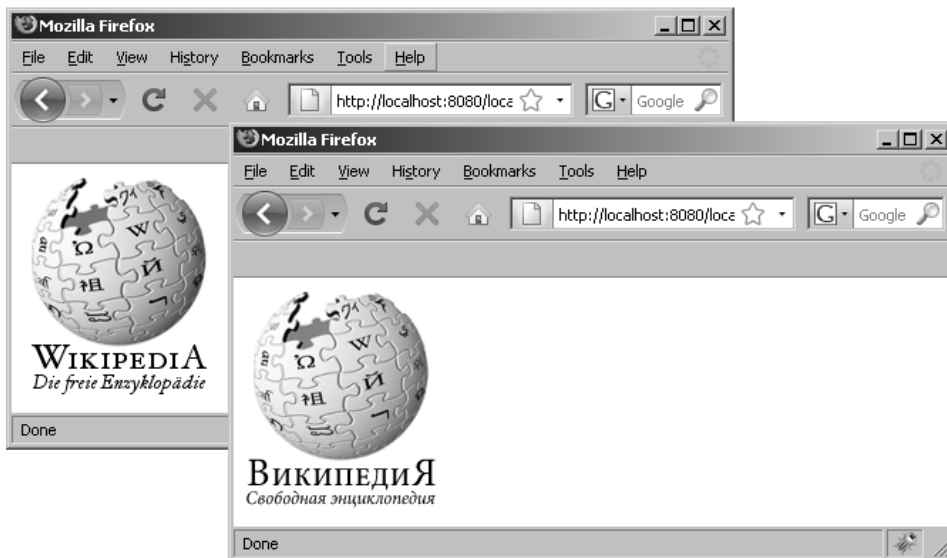


Abbildung 6.2: Lokalisierung von Assets

## 6.8 Umschalten zwischen unterstützten Sprachen einer Anwendung

Die Auswahl der Sprache einer Tapestry-Anwendung erfolgt, indem das `Locale` der jeweiligen Sprache in der angefragten URL kodiert wird. In jeder URL an Tapestry wird zwischen dem Kontextpfad und dem Namen der angefragten Seite nach einer String-Repräsentation einer Sprache gesucht. So würde Tapestry in den folgenden URLs die Sprache `de` erkennen.

```
http://localhost:8080/app/de
```

```
http://localhost:8080/app/de/mypage
```

Falls dieses erkannte `Locale` in der Liste der unterstützten Sprachen enthalten ist (siehe Abschnitt 6.6), wird es mithilfe des Dienstes `PersistentLocale` dauerhaft gespeichert. Die Methode `set()` des Dienstes sorgt dafür, dass das `Locale` für den aktuellen Thread zwischengespeichert wird. Wenn Tapestry URLs für `PageLink`, `ActionLink` usw.

generiert, wird das in `PersistentLocale` gespeicherte `Locale` in der URL wieder kodiert. Damit wird sichergestellt, dass eine einmal in einer URL kodierte Sprache bei der Navigation zwischen Seiten einer Anwendung von einer Seite zur nächsten übertragen wird.

Falls Sie Ihren Benutzern die Möglichkeit geben wollen, die Sprache der Anwendung durch eine Aktion zu ändern, können Sie ein `Locale` programmatisch im Dienst `PersistentLocale` setzen. In Listing 6.17 ist das Template einer Seite zu sehen, in dem zwei `ActionLink` zur Auswahl der Sprachen Deutsch und Englisch zu sehen sind. Durch einen Klick auf einen der beiden Links der jeweiligen Handler-Methoden wird für das Ereignis `action` (Listing 6.18) entweder `Locale.GERMAN` oder `Locale.ENGLISH` im Dienst `PersistentLocale` gespeichert. Dies hat zur Folge, dass in allen Links auf die Seiten der Anwendung die gesetzte Sprache kodiert wird, sodass sie immer wieder ausgelesen werden kann. Sie können das nachvollziehen, wenn Sie nach der Auswahl einer Sprache nun auf den Link zur Startseite klicken. Sie werden sehen, dass eine der beiden Sprachen an die URL der Startseite angehängt wurde.

#### Listing 6.17: *ChooseLanguage.tml*

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_1_0.xsd">
  <body>
    Bitte wählen Sie Ihre Sprache aus
    und gehen Sie anschließend zur Startseite.
    <p>
      <t:actionlink t:id="de">Deutsch</t:actionlink> |
      <t:actionlink t:id="en">Englisch</t:actionlink>
    </p>
    <p>
      <t:pagelink page="Index">Gehe zur Startseite</t:pagelink>
    </p>
  </body>
</html>
```

#### Listing 6.18: *ChooseLanguage.java*

```
public class ChooseLanguage {
    @Inject
    private PersistentLocale persistentLocale;

    void onActionFromDe(){
        persistentLocale.set(Locale.GERMAN);
    }

    void onActionFromEn(){
        persistentLocale.set(Locale.ENGLISH);
    }
}
```

Zusätzlich wird das in der angefragten URL erkannte `Locale` im Dienst `ThreadLocale` gespeichert, der für jede ankommende Anfrage neu erzeugt wird. Jede Funktionalität in Tapestry, die das `Locale` der aktuellen Anfrage zur Internationalisierung benötigt, bezieht dieses `Locale` über den Dienst `ThreadLocale`. Auch Sie können diesen Dienst in Ihre Seiten bzw. Komponenten injizieren, um das `Locale` der aktuellen Anfrage herauszufinden.

Wenn in einer angefragten URL kein `Locale` erkannt wurde oder das erkannte `Locale` nicht in der Liste der unterstützten Sprachen enthalten ist, wird die in den Browsereinstellungen des Benutzers ausgewählte Sprache ermittelt. Die im Browser ausgewählte Sprache wird bei jeder Anfrage in dem HTTP-Header *Accept-Language* mitgeliefert, sodass Tapestry mithilfe der Servlet-API das `Locale` des Clients ermitteln und im Dienst `ThreadLocale` setzen kann.



### PersistentLocale vs. ThreadLocale

Die beiden Dienste `PersistentLocale` und `ThreadLocale` werden zur Speicherung des `Locale` des aktuellen Threads eingesetzt, es besteht jedoch ein wichtiger Unterschied.

Der Dienst `PersistentLocale` dient der Speicherung des `Locale` in einer URL. Falls in der URL einer Anfrage kein `Locale` erkannt wurde, wird keine Standardsprache in diesem Dienst gesetzt. Sie sollten sich nicht auf diesen Dienst verlassen, wenn Sie ein `Locale` benötigen.

Der Dienst `ThreadLocale` besitzt dagegen immer ein `Locale`, entweder das aus der URL oder aus dem HTTP-Header *Accept-Language*.

Für den Zugriff auf ein `Locale` gibt es also mehrere Alternativen:

- Für den Zugriff auf das `Locale` einer Anfrage kann der Dienst `Request` injiziert werden, über dessen Methode `getLocale()` das `Locale` bezogen werden kann.
- Falls gesetzt, kann das gespeicherte `Locale` mithilfe des Dienstes `PersistentLocale` ausgelesen werden. Dazu kann die Methode `get()` aufgerufen werden.
- Das `Locale` des aktuellen Thread kann beim Dienst `ThreadLocale` über die Methode `getLocale()` abgefragt werden.
- Man kann aber auch eine Instanz von `Locale` in eine Seite injizieren lassen, indem eine Eigenschaft vom Typ `java.util.Locale` mit `@Inject` annotiert wird. Das injizierte `Locale` kommt aus dem Dienst `ThreadLocale`.

In Listing 6.19 sind alle Möglichkeiten zum Zugriff auf die Spracheinstellungen des aktuellen Benutzers zu sehen.

Listing 6.19: Unterschiedliche Arten des Zugriffs auf Locale

```
public class LocaleExample {
    @Inject
    private Request request;

    @Inject
    private Locale locale;

    @Inject
    private ThreadLocale threadLocale;

    @Inject
    private PersistentLocale persistentLocale;

    void onActivate() {
        System.err.println("request.getLocale(): "
            + request.getLocale());
        System.err.println("locale: " + locale);
        System.err.println("threadLocale: "
            + threadLocale.getLocale());
        System.err.println("persistentLocale.get(): "
            + persistentLocale.get());
    }
}
```

## 6.9 Zusammenfassung

In Tapestry werden die Übersetzungen der darzustellenden Texte in sogenannten *Nachrichtenkatalogen* verwaltet. Es wird zwischen einem *anwendungsweiten-* und *Komponenten-Nachrichtenkatalog* unterschieden. Ein *anwendungsweiter Nachrichtenkatalog* umfasst alle Nachrichten, die in vielen Seiten oder Komponenten benötigt werden. Ein *Komponenten-Nachrichtenkatalog* gehört dagegen exklusiv einer Seite oder Komponente.

Neben einzelnen Texten können auch ganze Templates lokalisiert werden, um beispielsweise das Layout einer Anwendung an andere Schreibrichtungen anzupassen. Weiterhin können statische Ressourcen durch `Asset` repräsentiert werden, die zum einen gecached, zum anderen lokalisiert werden können.

Das Zuordnen der lokalisierten Ressourcen (*Nachrichtenkatalog*, *Template* oder *Asset*) zur Sprache eines Benutzers erfolgt über das Suffix der jeweiligen Datei.

Zum Umschalten der Sprache einer Anwendung kann das `Locale` der jeweiligen Sprache in die Anfrage-URL kodiert werden. Tapestry wird das `Locale` erkennen und die Inhalte in der gewünschten Sprache darstellen. Dazu muss die angefragte Sprache in der Liste der unterstützten Sprachen enthalten sein.