13 JSP Standard Tag Library

Bibliotek med nya JSP-kommandon

JSP är inte begränsat till den uppsättning JSP-kommandon som ingår initialt.

Det är möjligt att utveckla *kommandobibliotek* med nya JSP-kommandon.

(Engelska termen är *custom tag libraries*.)

Utveckling sker genom deklarationer i XML och programmering i Java.

Exempel på tänkbara kommandobibliotek:

- Utbyggnad med nya språkelement
- Förenklad åtkomst till JDBC och JNDI
- Inkapsling av åtkomst till affärssystem
- Integration med modellklasser i Swing
- Diagramgrafik

Vi har redan visat hur JSP-kommandon gör det lättare att integrera JSP med JavaBeans-komponenter. JSP är emellertid inte begränsat till den uppsättning av JSP-kommandon som ingår initialt, utan det är möjligt att utveckla *kommandobibliotek* (den engelska termen är *custom tag libraries*) bestående av nya JSP-kommandon med gemensamt prefix för biblioteket. Sådana JSP-kommandon beskrivs deklarativt i en XML-beskrivning, och kan stödjas av exekverbar kod utvecklad i Java. Utveckling av sådana kommandobibliotek ligger emellertid helt utanför ramen för denna kurs.

Det finns många tänkbara tillämpningar av kommandobibliotek. Några exempel är:

- utbyggnad med nya språkelement för att reducera kodvolymen, t ex iteration över datastrukturer
- förenklad åtkomst till ofta använda delar av standardbiblioteket, som exempelvis **JDBC** och **JNDI**
- inkapsling av åtkomst till affärssystem som inte JSP-utvecklare bör ha obegränsad tillgång till
- integration av JSP med de mer avancerade modellklasserna i Javas användargränssnittsarkitektur **Swing**, t ex `ListModel` och `TableModel`
- stöd för att skapa diagramgrafik

JSP Standard Tag Library (JSTL)

Fr o m **JSP 1.2** finns ett standardbibliotek, **JSP Standard Tag Library** (JSTL):

- Innehåller JSP-kommandon för styrstrukturer, nyttofunktioner m m
- JSP-direktivet `taglib` binder ett lokalt prefix till varje bibliotek
- Varje bibliotek har ett standardprefix; kan ändras för att undvika kollisioner
- Attributvärden kan innehålla JSP-uttryck

dice.jsp

```
<%-- Dice JSP - Lennart Månsson - 2020-01-06 --%>
<%@ page info="Dice Throw JSP - Version 1.1" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

...
      int nDice = 0;
      if (count != null && count.length() > 0)
          nDice = Integer.parseInt(count);
    %>
    <hr><%= player %>, dina slag:
    <c:forEach var="i" begin="1" end="<%= nDice %>">
       <%= 1 + new Random().nextInt(6) %>
    </c:forEach>
  </body>
</html>
```

Att förstärka JSP med bibliotek av nya JSP-kommandon är en så central idé att det fr o m JSP 1.2 finns ett standardbibliotek av JSP-kommandon, **JSP Standard Tag Library** (**JSTL**). Syftet med JSTL är att bygga ut JSP inom ett antal centrala områden, för att på det sättet bredda den gemensamma bas som alla JSP-utvecklare har att utgå ifrån.

Bilden visar en variant av det tidigare exemplet med en JSP-sida för tärningsslag där Java-koden för en `for`-slinga som genererar upprepade tärningsslag har ersatts av JSP-kommandot `c:forEach` i JSTL. Det webbinnehåll som innesluts av märkorden `<c:forEach>` resp `</c:forEach>` kommer att genereras upprepade gånger. Antalet gånger styrs av kommandots attribut, där attributen `begin` och `end` anger start- resp slutvärde för en variabel som räknas upp med ett för varje nytt varv i slingan. Variabeln är åtkomlig i JSP under det variabelnamn som anges av attributet `var`:

```
<c:forEach var="i" begin="1" end="<%= nDice %>">
     <%= 1 + new Random().nextInt(6) %>
</c:forEach>
```

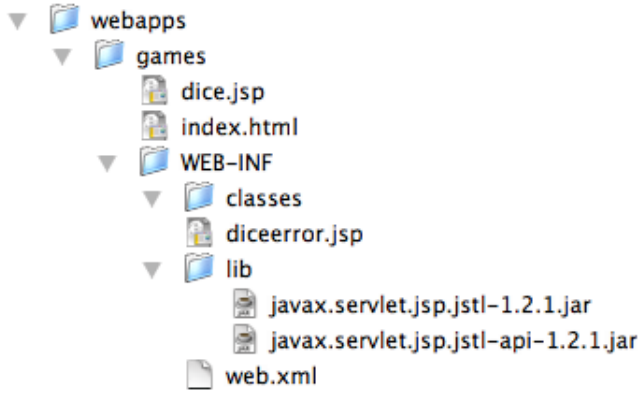
Notera att attributvärden precis som i tidigare fall kan vara dynamiska och anges av JSP-uttryck.

När ett kommandobibliotek ska användas – även de som ingår i JSTL – måste det först identifieras med hjälp av JSP-direktivet `<%@ taglib ... %>`, t ex:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

där attributet `uri` anger en adress som unikt identifierar biblioteket, medan attributet `prefix` visar vilket lokalt prefix som ska användas för bibliotekets samtliga JSP-kommandon i den aktuella JSP-filen.

Filstruktur för webbapplikationer med JSTL



- JSTL består av JAR-filerna `javax.servlet.jsp.jstl-1.2.1.jar` och `javax.servlet.jsp.jstl-api-1.2.1.jar`
- JAR-filer för bibliotek placeras i underkatalogen `lib` till katalogen `WEB-INF`
- Vissa webbservrar tillhandahåller automatiskt JAR-filerna centralt lagrade
- Webbservern **GlassFish** kräver inte att vi själv tar med JAR-filerna
- Däremot krävs de av **NetBeans IDE** om vi vill förkompilera JSP-filerna

En webbapplikation som utnyttjar JSP med JSTL följer samma övergripande filstruktur som vanliga webbapplikationer som utnyttjar enbart HTML och/eller JSP:

- Varje webbapplikation ges en egen katalog under den gemensamma rotkatalogen, med samma namn som webbapplikationen ges i URL:er. I bildens exempel har webbapplikationen namnet `games`.
- Filer med statiskt webbinnehåll som ska servas ut direkt till klienter utan ytterligare bearbetning, t ex HTML-filer, rena textfiler och bildfiler, placeras på den fysiska plats i filstrukturen som motsvarar den URL de adresseras med. Filen `index.html` placeras alltså direkt i webbapplikationens rotkatalog. Även JSP-filer som t ex `dice.jsp` placeras på detta sätt, trots att de inte servas ut direkt till klienter.
- Alla övriga filer som enbart är till för användning internt inom webbservern samlas i en underkatalog till webbapplikationens rotkatalog med namnet `WEB-INF`. Dit hör bl a ev installationsbeskrivningar.
- Fristående objektkodsfiler för Java-klasser placeras relativt underkatalogen `classes` till katalogen `WEB-INF`, så att katalogen `classes` fungerar som rotkatalog för objektkod inom webbapplikationen. I vårt exempel fanns inga sådana klasser, så därför är katalogen tom.
- JSTL 1.2.1 består av två JAR-filer: `javax.servlet.jsp.jstl-api-1.2.1.jar` och `javax.servlet.jsp.jstl-1.2.1.jar`. (JSTL 1.1 bestod av JAR-filerna `jstl.jar` och `standard.jar`, medan JSTL 1.2 bestod av JAR-filerna `jstl-api-1.2.jar` och `jstl-impl-1.2.jar`.). Dessa placeras i underkatalogen `lib` till katalogen `WEB-INF`.

Det är inte alltid nödvändigt att ta med JAR-filerna för JSTL i webbapplikationen. Många webbservrar med stöd för JSP, däribland den i **GlassFish**, har redan JAR-filerna installerade i en central katalog inom webbservern, och tar automatiskt med dem i rotkataloglistan för objektkod under exekvering. Däremot måste de tas med i projektet om vi vill att förkompileringen av JAR-filer i **NetBeans IDE** ska fungera.

Deklaration av JSP-kommandon

På något sätt måste webbservern få tillgång till deklarationer av de JSP-kommandon som ingår i kommandobiblioteket, så att det går att avgöra om vi använder korrekta attribut, etc. Hur sker detta?

En förutsättning för att vi ska kunna använda ett JSP-kommando som `<c:forEach>` är att det lokala prefixet (`c`) knutits till ett faktiskt kommandobibliotek. Det är vad vi gör med JSP-direktivet `taglib`:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

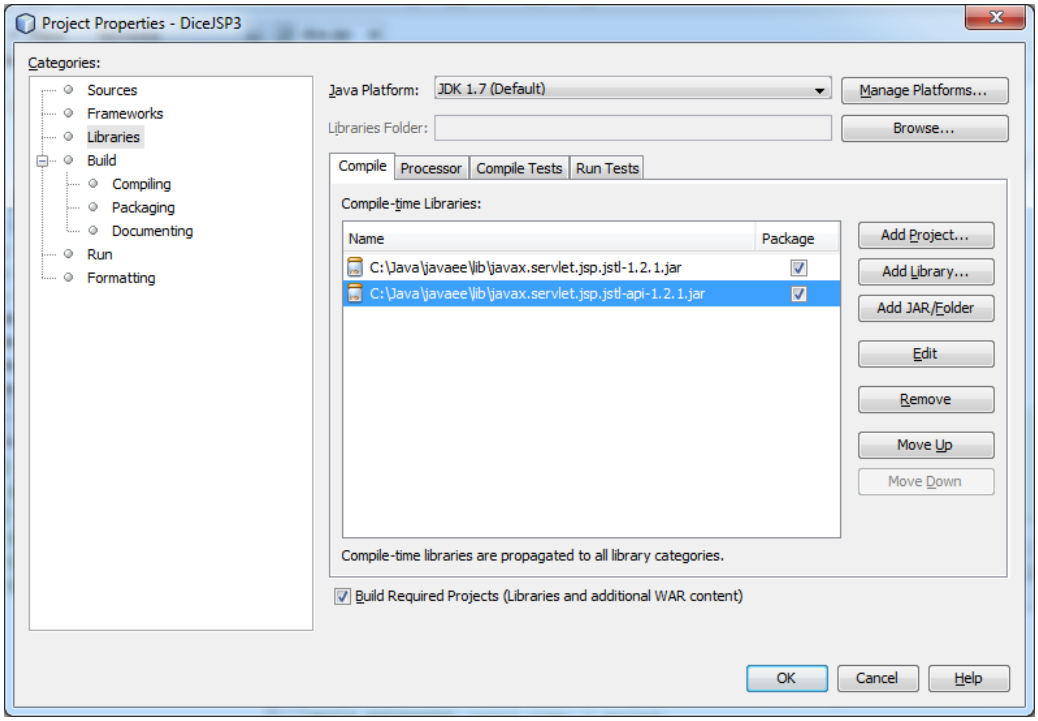
Med detta direktiv har vi knutit det lokala prefixet `c` till en specifik URI (resursidentifierare) som unikt identifierar biblioteket. Eftersom denna URI ser ut som en del av en URL till en resurs på Internet, är det lätt att tro att det är där definitionen av kommandobiblioteket finns, men så är inte fallet. I stället finns beskrivningen av kommandobiblioteket i en serie *TLD-filer* (*tag library descriptor-filer*) som distribueras med biblioteket. I dessa TLD-filer finns deklarationer för samtliga kommandon i biblioteket. För JSTL 1.2.1 finns de i katalogen `META-INF` i JAR-filen `javax.servlet.jsp.jstl-1.2.1.jar`.

JSTL är ett stort bibliotek. De JSP-kommandon som ingår i biblioteket är indelade i underbibliotek som knyts till varsitt prefix och har varsin TLD-fil. I vårt fall är deklarationerna för **Core**-biblioteket i JSTL samlade i en TLD-fil med namnet `c.tld` i JAR-filen `javax.servlet.jsp.jstl-1.2.1.jar`. (Filens namn stämmer överens med att kommandon i biblioteket konventionsmässigt ges prefixet `c`.)

Det som gör det möjligt för webbservern att knyta alla JSP-kommandon med ett visst lokalt prefix i JSP-sidan till rätt deklarationer, dvs rätt TLD-fil, är den URI som förekom i JSP-direktivet `taglib`. Exakt samma URI ska återfinnas i TLD-filen som värde för elementet `uri`, och denna URI ska vara unik för TLD-filen. Eftersom TLD-filer finns på ett förbestämt ställe i JAR-filerna, kan webbservern leta bland rotkataloglistans JAR-filer och på så vis lokalisera rätt TLD-fil.

JSP-utveckling i NetBeans IDE

Lägga till biblioteksfiler för JSTL



Vi har redan nämnt att det inte är nödvändigt att lägga till JSTL-biblioteket i projektet för att kunna exekvera webbapplikationen i webbservern som hör till **GlassFish**. För vissa andra webbservrar kan detta emellertid krävas, och i verktyget **NetBeans IDE** krävs det om vi vill kunna förkompilera JSP-sidor för att verifiera syntaxen i dem. Därför är det ett frivilligt steg att lägga till JSTL-biblioteket i projektet:

För att lägga till biblioteket JSTL, högerklicka på symbolen för projektet i vänsterspalten i verktyget och välj **Properties**.

Under kategorin *Libraries* visas vilka bibliotek och andra moduler som utnyttjas av projektet:

- Klicka på knappen **Add JAR/Folder**.
- Navigera till katalogen `C:\Java\javaee\lib` i kursens underlag. Markera filen med namnet `javax.servlet.jsp.jstl-1.2.1.jar`, välj alternativet *Absolute Path* och klicka på knappen **Öppna**. Upprepa förfarandet för JAR-filen `javax.servlet.jsp.jstl-api-1.2.1.jar`. Bibliotekets båda JAR-filer ska nu visas i listan i fönstret **Properties**.
- Klicka **OK** för att bekräfta de ändrade inställningarna.

(Verktyget NetBeans i nuvarande version tillhandahåller en äldre version, JSTL 1.1, som ett färdigt val genom att i stället klicka på knappen **Add Library**, markera biblioteket **JSTL 1.1** och klicka på knappen **Add Library**. JSTL 1.2.1 finns däremot inte som ett färdigt val, varför vi i stället hämtat biblioteket från `http://jstl.java.net` och placerat det i kursens underlag.)

De två JAR-filerna som ingår i JSTL ska nu synas under alternativet **Libraries** i vänsterspalten (dubbelklicka **Libraries** under projektets symbol om listan inte syns). Därmed kommer dessa biblioteksfiler automatiskt att tas med i WAR-filen för webbapplikationen.

Kommandobibliotek i JSTL 1.2

Bibliotek	*Lokalt prefix*	*Omfattar*
Core	`c`	styrstrukturer för villkorlig och upprepad generering av webbinnehåll, felhantering, hantering av variabler, JavaBeans-komponenter och webbresurser
I18N & Formatting	`fmt`	stöd för nationell anpassning i form av formatering och resurshantering
Database Access	`sql`	stöd för databasåtkomst via JDBC
XML Processing	`x`	stöd för XML-bearbetning
Functions	`fn`	vanliga nyttofunktioner, t ex stränghantering

- JSTL är en obligatorisk del av JSP fr o m **JSP 2.0**
- JSTL 1.0 var ett frivilligt tillägg till **JSP 1.2**
- Kommandobiblioteket **Functions** ingick inte i JSTL 1.0.

JSTL består idag av fem kommandobibliotek:

- **Core** (kommandoprefix `c`, URI `http://java.sun.com/jsp/jstl/core`)
  – stöd för bl a hantering av JSP-variabler och JavaBeans-komponenter, hantering av URL-adresserade webbresurser, styrstrukturer och felhantering
- **I18N & Formatting** (kommandoprefix `fmt`, URI `http://java.sun.com/jsp/jstl/fmt`)
  – stöd för nationell anpassning i form av formatering och resurshantering (den kryptiska förkortningen *I18N* står för *internationalization*)
- **Database Access** (kommandoprefix `sql`, URI `http://java.sun.com/jsp/jstl/sql`)
  – stöd för relationsdatabasåtkomst via JDBC
- **XML Processing** (kommandoprefix `x`, URI `http://java.sun.com/jsp/jstl/xml`)
  – stöd för XML-bearbetning
- **Functions** (kommandoprefix `fn`, URI `http://java.sun.com/jsp/jstl/functions`)
  – stöd för vanliga nyttofunktioner, t ex för stränghantering

Kommandobiblioteket **Functions** skiljer sig från de övriga, eftersom dess beståndsdelar uteslutande används i EL-uttryck (presenteras senare i kapitlet).

Den första versionen av JSTL, version 1.0, var ett frivilligt tillägg till JSP 1.2. I den ingick samtliga ovan nämnda kommandobibliotek utom **Functions**. Fr o m **JSP 2.0** ingår JSTL som en obligatorisk del i JSP. Den version som ingick i JSP 2.0 var **JSTL 1.1**. Den nuvarande versionen, **JSTL 1.2**, är en obligatorisk del fr o m **JSP 2.1**.