ΕΠΙΣΚΟΠΗΣΗ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ JSP

Θέματα σε αυτό το Κεφάλαιο

- Κατανόηση της ανάγκης για την JSP
- Εκτίμηση των ωφελειών από την JSP
- Σύγκριση της JSP και άλλων τεχνολογιών
- Αποφυγή παρανοήσεων σχετικά με την JSP
- Εγκατάσταση σελίδων JSP
- Επισκόπηση της σύνταξης JSP

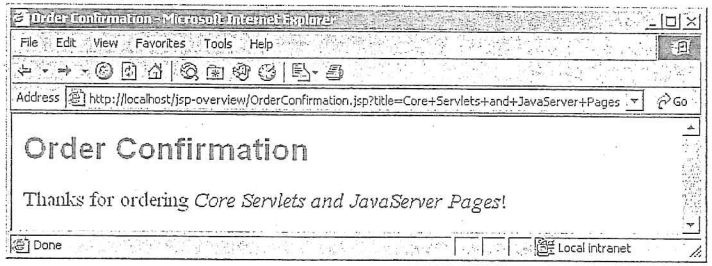
Κεφάλαιο **10**

Η τεχνολογία JavaServer Pages (JSP, σελίδες διακομιστή Java) σας δίνει τη δυνατότητα να αναμίξετε κανονικό στατικό κώδικα HTML με περιεχόμενο που παράγεται δυναμικά. Γράφετε απλώς τον κανονικό κώδικα HTML με το συνηθισμένο τρόπο, χρησιμοποιώντας οικεία εργαλεία κατασκευής ιστοσελίδων. Κατόπιν περικλείετε τον κώδικα για τα δυναμικά τμήματα σε ειδικές ετικέτες, οι περισσότερες από τις οποίες ξεκινούν με <% και τελειώνουν με %>.

Για παράδειγμα, η Λίστα 10.1 (Εικόνα 10-1) παρουσιάζει μια πολύ μικρή σελίδα JSP η οποία χρησιμοποιεί μια παράμετρο αίτησης για να εμφανίσει τον τίτλο ενός βιβλίου. Παρατηρήστε ότι η λίστα είναι κατά κύριο λόγο τυπικός κώδικας HTML· ο δυναμικός κώδικας συνίσταται μόνο στο τμήμα που φαίνεται με έντονη γραφή.

Λίστα 10.1 OrderConfirmation.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Order Confirmation</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.CSS"
      TYPE="text/css">
</HEAD>
<BODY>
<H2>Order Confirmation</H2>
Thanks for ordering <I><%= request.getParameter("title") %> </I>!
   </BODY></HTML>
```



Εικόνα 10-1 Αποτέλεσμα της σελίδας `OrderConfirmation.jsp`.

Μπορείτε να θεωρήσετε τις μικροϋπηρεσίες σαν κώδικα Java με HTML στο εσωτερικό τους· μπορείτε να θεωρήσετε τις σελίδες JSP σαν σελίδες HTML με κώδικα Java στο εσωτερικό τους. Ούτε οι μικροϋπηρεσίες ούτε οι σελίδες JSP περιορίζονται υποχρεωτικά στη γλώσσα HTML, αν και συνήθως αυτό κάνουν, και αυτή η υπεραπλουστευμένη περιγραφή είναι ένας συνηθισμένος τρόπος αντιμετώπισης αυτών των τεχνολογιών.

Βέβαια, παρά τις φαινομενικά μεγάλες διαφορές μεταξύ των σελίδων JSP και των μικροϋπηρεσιών, παρασκηνιακά είναι το ίδιο πράγμα. Οι σελίδες JSP μεταφράζονται σε μικροϋπηρεσίες, οι μικροϋπηρεσίες μεταγλωττίζονται, και κατά το χρόνο της αίτησης εκτελούνται οι μεταγλωττισμένες μικροϋπηρεσίες. Έτσι η συγγραφή σελίδων JSP είναι απλώς ένας άλλος τρόπος συγγραφής μικροϋπηρεσιών.

Αν και παρασκηνιακά οι μικροϋπηρεσίες και οι σελίδες JSP είναι ισοδύναμες, δεν είναι το ίδιο χρήσιμες σε όλες τις περιπτώσεις. Ο διαχωρισμός της στατικής HTML από το δυναμικό περιεχόμενο παρέχει μια σειρά από οφέλη σε σχέση με τις μικροϋπηρεσίες, και η προσέγγιση που χρησιμοποιείται στις σελίδες JSP προσφέρει αρκετά πλεονεκτήματα σε σχέση με ανταγωνιστικές τεχνολογίες. Αυτό το κεφάλαιο εξηγεί τις αιτίες για τη χρήση των σελίδων JSP, εξετάζει τα πλεονεκτήματα, ξεδιαλύνει μερικές παρανοήσεις, παρουσιάζει τον τρόπο εγκατάστασης και εκτέλεσης των σελίδων JSP, και συνοψίζει τη σύνταξη της JSP που εξετάζεται στο υπόλοιπο μέρος του βιβλίου.

10.1 Η ανάγκη για την τεχνολογία JSP

"Για σταμάτα", θα μου πείτε, "Μόλις ξόδεψες αρκετά κεφάλαια στα οποία εκθειάζεις τις αρετές των μικροϋπηρεσιών. Μας αρέσουν οι μικροϋπηρεσίες. Είναι εύκολες στη συγγραφή και αποδοτικές στην εκτέλεση. Κάνουν εύκολη την ανάγνωση των παραμέτρων αιτήσεων και τη διευθέτηση προσαρμοσμένου κώδικα για το χειρισμό ελλιπών δεδομένων και δεδομένων σε λανθασμένη μορφή. Μπορούν εύκολα να χρησιμοποιήσουν τις κεφαλίδες αιτήσεων HTTP και μπορούν με ευελιξία να χειριστούν δεδομένα απαντήσεων HTTP. Μπορούν να προσαρμόσουν τη συμπεριφορά τους με βάση μπισκότα, να παρακολουθούν ειδικά δεδομένα του χρήστη με την API παρακολούθησης συνεδριών, και να επικοινωνήσουν με βάσεις δεδομένων μέσω της JDBC. Τι παραπάνω χρειάζομαι;"

Αυτή είναι μια πολύ καλή παρατήρηση. Οι μικροϋπηρεσίες είναι πράγματι χρήσιμες, και η τεχνολογία JSP με κανέναν τρόπο δεν τις αχρηστεύει. Κοιτάξτε όμως τη λίστα με τα θέματα στα οποία διαπρέπουν οι μικροϋπηρεσίες. Όλα αφορούν εργασίες που σχετίζονται με τον *προγραμματισμό* ή την *επεξεργασία* δεδομένων. Οι μικροϋπηρεσίες, όμως, δεν είναι τόσο καλές στην *παρουσίαση* και έχουν τις ακόλουθες ελλείψεις όσον αφορά την παραγωγή εξόδου:

- **Είναι δύσκολη η συγγραφή και συντήρηση του κώδικα HTML.** Χρησιμοποιείτε εντολές `print` για την παραγωγή HTML; Δύσκολα αυτό θα μπορούσε να χαρακτηριστεί εύχρηστο: πρέπει να χρησιμοποιείτε παρενθέσεις και ελληνικά ερωτηματικά, πρέπει να εισάγετε ανάποδες καθέτους μπροστά από τα ενσωματωμένα εισαγωγικά, και πρέπει να χρησιμοποιείτε συνένωση αλφαριθμητικών για να συγκεντρώσετε το περιεχόμενο. Εκτός απ' όλα αυτά, το αποτέλεσμα δεν μοιάζει με κώδικα HTML και έτσι είναι δύσκολο να το κατανοήσει κανείς με μια ματιά. Συγκρίνετε αυτό το στυλ των μικροϋπηρεσιών με τη Λίστα 10.1, όπου με δυσκολία μπορείτε να παρατηρήσετε ότι η σελίδα δεν είναι ένα συνηθισμένο έγγραφο HTML.
- **Δεν μπορείτε να χρησιμοποιήσετε τα τυπικά εργαλεία κατασκευής σελίδων HTML.** Όλα αυτά τα θαυμάσια εργαλεία ανάπτυξης τοποθεσιών Ιστού που διαθέτετε είναι ελάχιστα χρήσιμα όταν γράφετε κώδικα Java.
- **Η HTML είναι απροσπέλαστη για τους προγραμματιστές που δεν γνωρίζουν Java.** Αν η HTML είναι ενσωματωμένη στον κώδικα Java, έναν ειδικός στην ανάπτυξη εφαρμογών για τον Ιστό που δεν γνωρίζει τη γλώσσα προγραμματισμού Java θα δυσκολευτεί να εξετάσει και να τροποποιήσει τον κώδικα HTML.

10.2 Οφέλη της τεχνολογίας JSP

Οι σελίδες JSP μεταφράζονται σε μικροϋπηρεσίες. Έτσι, σε θεμελιώδες επίπεδο, οποιαδήποτε εργασία που μπορεί να πραγματοποιηθεί με σελίδες JSP μπορεί επίσης να επιτευχθεί και με τις μικροϋπηρεσίες. Ωστόσο, αυτή η εσωτερική ισοδυναμία δεν σημαίνει ότι οι μικροϋπηρεσίες και οι σελίδες JSP είναι εξίσου κατάλληλες σε όλες τις περιπτώσεις. Το θέμα δεν είναι η ισχύς της τεχνολογίας, αλλά η ευκολία, η παραγωγικότητα, και η δυνατότητα συντήρησης που προσφέρουν η μία και η άλλη τεχνολογία. Έτσι κι αλλιώς, οτιδήποτε μπορείτε να κάνετε σε μια πλατφόρμα υπολογιστή με τη γλώσσα προγραμματισμού Java θα μπορούσατε να το κάνετε και στη γλώσσα assembly. Είναι όμως σημαντικό ζήτημα η γλώσσα που θα επιλέξετε.

Η τεχνολογία JSP παρέχει τα ακόλουθα οφέλη σε σχέση με τη χρήση μόνο μικροϋπηρεσιών:

- **Είναι ευκολότερη η συγγραφή και συντήρηση του κώδικα HTML.** Ο στατικός σας κώδικας είναι κανονικός κώδικας HTML: χωρίς επιπλέον ανάποδες καθέτους, διπλά εισαγωγικά, και περίεργη σύνταξη Java.
- **Μπορείτε να χρησιμοποιήσετε τυπικά εργαλεία ανάπτυξης τοποθεσιών Ιστού.** Για παράδειγμα, για τις πιο πολλές σελίδες JSP του βιβλίου χρησιμοποιούμε το Macromedia Dreamweaver. Μπορούν ακόμα και να χρησιμοποιηθούν εργαλεία HTML που

δεν γνωρίζουν τίποτα για σελίδες JSP, επειδή απλούστατα δεν λαμβάνουν υπόψη τους τις ετικέτες JSP.

- **Μπορείτε να διαιρέσετε την ομάδα ανάπτυξης.** Οι προγραμματιστές Java μπορούν να δουλεύουν με το δυναμικό κώδικα. Οι προγραμματιστές Ιστού μπορούν να επικεντρωθούν στο επίπεδο της παρουσίασης. Στα μεγάλα έργα αυτός ο διαχωρισμός είναι πολύ σημαντικός. Ανάλογα με το μέγεθος της ομάδας σας και την πολυπλοκότητα του έργου σας, μπορείτε να επιβάλετε έναν πιο χαλαρό ή πιο ισχυρό διαχωρισμό μεταξύ του στατικού κώδικα HTML και του δυναμικού περιεχομένου.

Με αυτή τη συζήτηση δεν θέλουμε να πούμε ότι πρέπει να σταματήσετε να χρησιμοποιείτε μικροϋπηρεσίες και θα πρέπει να χρησιμοποιείτε μόνο σελίδες JSP. Καθόλου. Σχεδόν όλα τα έργα θα χρησιμοποιούν και τις δύο. Για ορισμένες αιτήσεις στο έργο σας θα χρησιμοποιήσετε μικροϋπηρεσίες. Για άλλες θα χρησιμοποιήσετε σελίδες JSP. Και για άλλες θα τις συνδυάσετε με την αρχιτεκτονική MVC (Κεφάλαιο 15). Θέλετε απλώς το κατάλληλο εργαλείο για την κάθε δουλειά, και οι μικροϋπηρεσίες δεν αποτελούν από μόνες τους πανάκεια.

10.3 Πλεονεκτήματα της JSP σε σχέση με ανταγωνιστικές τεχνολογίες

Μερικά χρόνια πριν, ο βασικός συγγραφέας του βιβλίου (ο Marty) είχε προσκληθεί να συμμετάσχει σε μία μικρή συζήτηση στρογγυλής τραπέζης, των 20 ατόμων, σχετικά με την τεχνολογία λογισμικού. Δίπλα από τον Marty καθόταν ο James Gosling, ο δημιουργός της γλώσσας προγραμματισμού Java. Μερικές θέσεις παραδίπλα καθόταν ένα υψηλόβαθμο στέλεχος ενός πολύ μεγάλου οίκου λογισμικού, με έδρα την πόλη Redmond της Washington. Κατά τη διάρκεια της συζήτησης ο συντονιστής έφερε το θέμα της Jiny, η οποία εκείνη την εποχή ήταν μια νέα τεχνολογία Java, και ρώτησε το συγκεκριμένο στέλεχος ποιες ήταν οι σκέψεις του γι' αυτήν. Αυτός απάντησε ότι ήταν πολύ νωρίς για να πει κάτι, αλλά ότι φαινόταν ως εξαίρετη ιδέα. Συνεχίζοντας είπε ότι θα την παρακολουθούσαν, και αν φαινόταν να τυγχάνει αποδοχής θα ακολουθούσαν τη στρατηγική της εταιρείας που ήταν "αγκαλιάζω και επεκτείνω" (embrace and extend). Σε εκείνο το σημείο παρενέβη ο Gosling και είπε: "Εννοείς ντροπιάζω και φουσκώνω." (*dis*grace and *dis*tend).

Η αιτία για τα παράπονα του Gosling ήταν το ότι θεωρούσε πως η συγκεκριμένη εταιρεία θα έπαιρνε την τεχνολογία από άλλες εταιρείες και θα την αλλοίωνε για τους δικούς της σκοπούς. Συνέβη όμως το ακριβώς αντίθετο. Δεν ήταν η κοινότητα της Java που επινόησε την ιδέα της σχεδίασης σελίδων ως ένα μίγμα στατικής HTML και δυναμικού κώδικα επισημασμένου με ειδικές ετικέτες. Για παράδειγμα, το ColdFusion το έκανε αυτό χρόνια πριν. Ακόμα και η ASP (ένα προϊόν από την εταιρεία του στελέχους που αναφέραμε προηγουμένως) διαφήμιζε αυτή την προσέγγιση πριν παρουσιαστεί η τεχνολογία JSP και αποφασίσει να ακολουθήσει τον ίδιο δρόμο. Στην πραγματικότητα η JSP όχι μόνο υιοθέτησε τη γενική ιδέα, αλλά χρησιμοποίησε πολλές από τις ειδικές ετικέτες που χρησιμοποιούσε η τεχνολογία ASP.

Έτσι το ερώτημα είναι το εξής: γιατί να χρησιμοποιήσουμε JSP και όχι κάποια από αυτές τις τεχνολογίες; Μια πρώτη απάντηση είναι ότι δεν υποστηρίζουμε ότι όλοι θα πρέπει να το κάνουν.

Αρκετές από αυτές τις τεχνολογίες είναι αρκετά καλές και αποτελούν εύλογες επιλογές σε ορισμένες περιπτώσεις. Σε άλλες περιπτώσεις, όμως, η τεχνολογία JSP είναι εμφανώς ανώτερη. Μερικοί από τους λόγους για τους οποίους ισχύει αυτό είναι οι ακόλουθοι.

Σύγκριση με τις τεχνολογίες .NET και ASP (Active Server Pages)

Η .NET είναι μια καλά σχεδιασμένη τεχνολογία της Microsoft. Η ASP.NET είναι το τμήμα που ανταγωνίζεται άμεσα τις μικροϋπηρεσίες και τις σελίδες JSP. Τα πλεονεκτήματα των JSP είναι δύο.

Πρώτον, η JSP είναι φορητή σε πολλά λειτουργικά συστήματα και διακομιστές Ιστού· δεν είστε δεσμευμένοι να κάνετε εκδίπλωση της εφαρμογής σας μόνο σε συστήματα με Windows και IIS. Αν και ο πυρήνας της πλατφόρμας .NET εκτελείται και σε ορισμένα άλλα λειτουργικά συστήματα πλην των Windows, δεν ισχύει το ίδιο και για την τεχνολογία ASP. Δεν μπορείτε να περιμένετε να εκδιπλώσετε σοβαρές εφαρμογές ASP σε πολλούς διακομιστές και λειτουργικά συστήματα. Για ορισμένες εφαρμογές, αυτή η διαφορά δεν παίζει ρόλο. Σε άλλες όμως είναι σημαντική.

Δεύτερον, για κάποιες εφαρμογές η επιλογή της υποκείμενης γλώσσας παίζει μεγάλο ρόλο. Για παράδειγμα, αν και η γλώσσα C# της .NET είναι πολύ καλά σχεδιασμένη και μοιάζει με την Java, υπάρχουν πολύ λιγότεροι προγραμματιστές που είναι εξοικειωμένοι με τον πυρήνα της σύνταξης C# ή με τις βοηθητικές βιβλιοθήκες της. Επιπλέον, πολλοί προγραμματιστές εξακολουθούν να χρησιμοποιούν την αρχική έκδοση της τεχνολογίας ASP. Ως προς αυτή την έκδοση η JSP έχει ένα ξεκάθαρο προβάδισμα όσον αφορά το δυναμικό κώδικα. Με την τεχνολογία JSP το δυναμικό μέρος γράφεται σε Java, και όχι σε VBScript ή κάποια άλλη γλώσσα, ειδικά για την ASP. Έτσι η JSP είναι ισχυρότερη και πιο κατάλληλη για πολύπλοκες εφαρμογές που χρειάζονται επαναχρησιμοποιούμενα στοιχεία.

Τα ίδια επιχειρήματα θα μπορούσατε να χρησιμοποιήσετε και όταν συγκρίνετε τη JSP με την προηγούμενη έκδοση του ColdFusion· με την τεχνολογία JSP μπορείτε να χρησιμοποιήσετε την Java για τον "πραγματικό κώδικα", και δεν είστε δεσμευμένοι με κάποιο συγκεκριμένο διακομιστή. Η τρέχουσα έκδοση, όμως, του ColdFusion χρησιμοποιεί διακομιστή J2EE, οπότε επιτρέπει στους προγραμματιστές την ανάμιξη κώδικα ColdFusion και κώδικα μικροϋπηρεσιών/JSP.

Σύγκριση με την PHP

Η PHP (αναδρομικό ακρωνύμιο του "PHP: Hypertext Preprocessor", δηλαδή "προεπεξεργαστής υπερ-κειμένου") είναι μία δωρεάν, ανοικτού κώδικα, γλώσσα σεναρίων με ενσωματωμένη HTML, που κατά κάποιον τρόπο μοιάζει και με την ASP και με την JSP. Ένα πλεονέκτημα της JSP είναι ότι το δυναμικό μέρος γράφεται σε Java, η οποία διαθέτει ήδη μια εκτενή API για δίκτυα, προσπέλαση βάσεων δεδομένων, κατανεμημένα αντικείμενα, και άλλα τέτοια στοιχεία, ενώ η PHP απαιτεί την εκμάθηση μιας εντελώς νέας και λιγότερο χρησιμοποιούμενης γλώσσας. Ένα δεύτερο πλεονέκτημα της JSP είναι ότι διαθέτει ευρύτερη υποστήριξη από τους οίκους κατασκευής εργαλείων και διακομιστών, σε σύγκριση με την PHP.

Σύγκριση με τις αμιγείς μικροϋπηρεσίες

Η JSP δεν παρέχει καμία δυνατότητα που, θεωρητικά, δεν είναι εφικτή με τις μικροϋπηρεσίες. Στην πραγματικότητα, παρασκηνιακά τα έγγραφα JSP μεταφράζονται αυτόματα σε μικροϋπηρεσίες. Είναι όμως πιο εύχρηστο να γράψετε (και να τροποποιήσετε!) κανονικό κώδικα HTML από το να χρησιμοποιήσετε εκατοντάδες εντολές `println` για την παραγωγή κώδικα HTML. Επιπλέον, ο διαχωρισμός της παρουσίασης από το περιεχόμενο σας επιτρέπει να αναθέσετε σε διαφορετικά άτομα διαφορετικές εργασίες: οι σχεδιαστές ιστοσελίδων μπορούν να χρησιμοποιήσουν οικεία εργαλεία για την παραγωγή της HTML, και είτε να αφήσουν χώρο για τους προγραμματιστές των μικροϋπηρεσιών, ώστε αυτοί να εισαγάγουν το δυναμικό περιεχόμενο, είτε να καλούν έμμεσα το δυναμικό περιεχόμενο μέσω ετικετών XML.

Σημαίνει αυτό ότι μπορείτε απλώς να μάθετε JSP και να ξεχάσετε τις μικροϋπηρεσίες; Εκατό τοις εκατό όχι! Οι προγραμματιστές JSP θα πρέπει να γνωρίζουν μικροϋπηρεσίες για τέσσερις λόγους:

1. Οι σελίδες JSP μεταφράζονται σε μικροϋπηρεσίες. Δεν μπορείτε να καταλάβετε πώς λειτουργούν οι σελίδες JSP αν δεν καταλαβαίνετε πώς λειτουργούν οι μικροϋπηρεσίες.
2. Οι σελίδες JSP αποτελούνται από στατική HTML, ειδικές ετικέτες JSP, και κώδικα Java. Τι είδους κώδικα Java; Κώδικα μικροϋπηρεσιών! Δεν μπορείτε να γράψετε αυτόν τον κώδικα αν δεν καταλαβαίνετε από προγραμματισμό μικροϋπηρεσιών.
3. Ορισμένες εργασίες υλοποιούνται καλύτερα από μικροϋπηρεσίες παρά με σελίδες JSP. Η JSP είναι καλή στην παραγωγή σελίδων που περιλαμβάνουν μεγάλες ενότητες από καλά δομημένο κώδικα HTML ή άλλο παρόμοιο κείμενο. Οι μικροϋπηρεσίες είναι καλύτερες στην παραγωγή δυαδικών δεδομένων, στη δόμηση σελίδων με εξαιρετικά μεταβαλλόμενη δομή, και στην εκτέλεση εργασιών (όπως η ανακατεύθυνση) που περιλαμβάνουν ελάχιστη ή καθόλου έξοδο.
4. Ορισμένες εργασίες εκτελούνται καλύτερα με το *συνδυασμό* μικροϋπηρεσιών και JSP, παρά με τη χρήση *μόνο* μικροϋπηρεσιών ή *μόνο* σελίδων JSP. Για λεπτομέρειες δείτε το Κεφάλαιο 15.

Σύγκριση με την JavaScript

Η JavaScript, μια γλώσσα προγραμματισμού που είναι εντελώς ξεχωριστή από την Java, κανονικά χρησιμοποιείται για τη δυναμική παραγωγή HTML στον *πελάτη*, κατασκευάζοντας τμήματα της ιστοσελίδας καθώς ο φυλλομετρητής φορτώνει το έγγραφο. Είναι μία χρήσιμη δυνατότητα και κανονικά δεν επικαλύπτεται με τις δυνατότητες της JSP (που εκτελείται στο *διακομιστή*). Οι σελίδες JSP εξακολουθούν να περιλαμβάνουν ετικέτες `SCRIPT` για κώδικα JavaScript, όπως γίνεται και στις κανονικές σελίδες HTML. Στην πραγματικότητα, η JSP μπορεί να χρησιμοποιηθεί για τη δυναμική παραγωγή του κώδικα JavaScript που θα σταλεί στον πελάτη. Έτσι η JavaScript δεν είναι ανταγωνιστική τεχνολογία, αλλά συμπληρωματική.

Είναι επίσης δυνατή η χρήση της JavaScript στο διακομιστή, όπως στους διακομιστές Sun ONE (παλιότερα ονομαζόταν iPlanet), IIS, και BroadVision. Ωστόσο, η Java είναι πιο ισχυρή, ευέλικτη, αξιόπιστη, και φορητή.

Σύγκριση με τη WebMacro ή τη Velocity

Κατά κανέναν τρόπο η τεχνολογία JSP δεν είναι τέλεια. Πολλοί έχουν υποδείξει λειτουργίες που θα μπορούσαν να βελτιωθούν. Αυτό είναι κάτι καλό, και ένα από τα πλεονεκτήματα της JSP είναι ότι οι προδιαγραφές της καθορίζονται από μία κοινότητα που προέρχεται από πολλές διαφορετικές εταιρείες. Έτσι η τεχνολογία μπορεί να ενσωματώσει βελτιώσεις σε επόμενες εκδόσεις της.

Ωστόσο, στην προσπάθειά τους να αντιμετωπίσουν αυτές τις ελλείψεις, κάποιες ομάδες έχουν αναπτύξει εναλλακτικές τεχνολογίες με βάση τη Java. Κάτι τέτοιο, κατά τη γνώμη μας, είναι λανθασμένο. Η χρήση ενός εργαλείου τρίτου κατασκευαστή, όπως το Apache Struts (δείτε το βιβλίο More Servlets and JavaServer Pages), που *επαυξάνει* την τεχνολογία μικροϋπηρεσιών και JSP, είναι καλή ιδέα όταν το εργαλείο προσφέρει ικανοποιητικό όφελος σε σχέση με την αυξημένη πολυπλοκότητα. Η χρήση, όμως, ενός μη καθιερωμένου εργαλείου το οποίο προσπαθεί να *αντικαταστήσει* την τεχνολογία JSP δεν είναι καλή ιδέα. Όταν επιλέγετε κάποια τεχνολογία, θα πρέπει να σταθμίζετε πολλούς παράγοντες: όπως τυποποίηση, φορητότητα, υποστήριξη από τη βιομηχανία, και τεχνικά χαρακτηριστικά. Τα επιχειρήματα υπέρ των εναλλακτικών επιλογών σε σχέση με την JSP εστιάζονται σχεδόν αποκλειστικά στα τεχνικά χαρακτηριστικά. Εξίσου σημαντικά θέματα είναι όμως η φορητότητα, η τυποποίηση, και η ολοκλήρωση. Για παράδειγμα, όπως αναφέρθηκε στην Ενότητα 2.11, οι προδιαγραφές των μικροϋπηρεσιών και των σελίδων JSP ορίζουν μια τυποποιημένη δομή καταλόγου για τις εφαρμογές Ιστού και παρέχουν πρότυπα αρχεία (αρχεία .war) για την εκδίπλωση των εφαρμογών Ιστού. Όλοι οι διακομιστές που είναι συμβατοί με την τεχνολογία JSP θα πρέπει να υποστηρίζουν αυτά τα πρότυπα. Μπορείτε να διευθετήσετε φίλτρα (δείτε το βιβλίο More Servlets and JavaServer Pages) τα οποία θα ισχύουν για οποιονδήποτε αριθμό μικροϋπηρεσιών και σελίδων JSP, αλλά όχι και σε μη τυποποιημένους πόρους. Το ίδιο ισχύει και για τις ρυθμίσεις ασφαλείας των εφαρμογών Ιστού (επίσης δείτε το βιβλίο More Servlets and JavaServer Pages).

Εκτός από αυτά τα ζητήματα, η τρομερή υποστήριξη της τεχνολογίας μικροϋπηρεσιών και JSP έχει αποτέλεσμα να παρουσιάζονται βελτιώσεις που τείνουν να αναιρέσουν πολλές από τις κριτικές κατά της JSP. Για παράδειγμα, η JSP Standard Tag Library (Τυπική βιβλιοθήκη ετικετών JSP) (More Servlets and JavaServer Pages) και η γλώσσα παραστάσεων JSP 2.0 (Κεφάλαιο 16) έρχονται να αντιμετωπίσουν δύο από τις πιο καλά θεμελιωμένες κριτικές: την έλλειψη καλών δομών επανάληψης και τη δυσκολία προσπέλασης δυναμικών αποτελεσμάτων χωρίς τη χρήση είτε κώδικα Java, είτε στοιχείων `jsp:useBean`.

10.4 Παρανοήσεις σχετικά με την τεχνολογία JSP

Σε αυτή την ενότητα θα εξετάσουμε μερικές από τις πιο συνηθισμένες παρανοήσεις σχετικά με την τεχνολογία JSP.

Πολλοί ξεχνούν ότι η JSP είναι τεχνολογία για το διακομιστή

Στην τοποθεσία Ιστού του βιβλίου αναφέρεται η διεύθυνση ηλεκτρονικού ταχυδρομείου του βασικού συγγραφέα: hall@coreservlets.com. Επιπλέον, ο Marty διδάσκει μικροϋπηρεσίες και JSP σε εκπαιδευτικά σεμινάρια σε διάφορες εταιρείες και δημόσιους οργανισμούς. Έτσι λαμβάνει πολλά μηνύματα με ερωτήσεις για τις μικροϋπηρεσίες και την τεχνολογία JSP. Ακολουθούν μερικές τυπικές ερωτήσεις που έχει λάβει (τις περισσότερες από αυτές περισσότερες από μία φορές).

- Ο διακομιστής μας εκτελεί το JDK 1.4, οπότε πώς θα βάλω ένα στοιχείο Swing σε μια σελίδα JSP;
- Πώς μπορώ να τοποθετήσω μια εικόνα σε μια σελίδα JSP; Δεν γνωρίζω τις κατάλληλες εντολές εισόδου/εξόδου της Java για την ανάγνωση αρχείων εικόνων.
- Επειδή ο Tomcat δεν υποστηρίζει την JavaScript, πώς θα κάνω τις εικόνες μου να φωτίζονται όταν ο χρήστης περνά το ποντίκι του επάνω από αυτές;
- Οι πελάτες μας χρησιμοποιούν παλαιότερους φυλλομετρητές που δεν καταλαβαίνουν JSP. Τι θα πρέπει να κάνουμε;
- Πώς μπορώ να εμποδίσω τους πελάτες να βλέπουν τις ετικέτες JSP όταν χρησιμοποιούν τη διαταγή "View Source" (Προβολή κώδικα) στο φυλλομετρητή τους;

Όλες αυτές οι ερωτήσεις βασίζονται στην υπόθεση ότι οι φυλλομετρητές γνωρίζουν σχετικά με τις διεργασίες που εκτελούνται στο διακομιστή. Κάτι τέτοιο όμως δεν ισχύει. Έτσι:

- Για την τοποθέτηση μικροεφαρμογών με στοιχεία Swing σε ιστοσελίδες, αυτό που παίζει ρόλο είναι η έκδοση Java του φυλλομετρητή — η έκδοση του διακομιστή δεν παίζει κανένα ρόλο. Αν ο φυλλομετρητής υποστηρίζει την πλατφόρμα Java 2 τότε χρησιμοποιείτε κανονικές ετικέτες `APPLET` (ή συνδεόμενες υπομονάδες Java), και το κάνετε αυτό ακόμα και αν ο διακομιστής χρησιμοποιεί κάποια τεχνολογία που δεν βασίζεται στη Java.
- Δεν χρειάζεστε τις εντολές εισόδου/εξόδου της Java για να διαβάσετε αρχεία εικόνων· τοποθετείτε απλώς την εικόνα στον κατάλογο των πόρων Ιστού (δηλαδή δύο επίπεδα επάνω από τον κατάλογο WEB-INF/classes) και δίνετε ως έξοδο μια κανονική ετικέτα `IMG`.
- Για να δημιουργήσετε εικόνες που αλλάζουν όταν περνά το ποντίκι επάνω από αυτές χρησιμοποιείτε κώδικα JavaScript ο οποίος εκτελείται στον πελάτη, τον οποίο δηλώνετε με την ετικέτα `SCRIPT`· αυτό δεν αλλάζει επειδή ο διακομιστής χρησιμοποιεί JSP.
- Οι φυλλομετρητές δεν "υποστηρίζουν" τη JSP — βλέπουν απλώς την έξοδο της σελίδας JSP. Έτσι θα πρέπει απλώς να φροντίσετε η σελίδα σας JSP να παράγει έξοδο HTML που να είναι συμβατή με το φυλλομετρητή, όπως θα κάνατε και με τις στατικές σελίδες HTML.

- Τέλος, φυσικά δεν χρειάζεται να κάνετε τίποτα για να εμποδίσετε τους πελάτες να δουν τις ετικέτες JSP· η επεξεργασία αυτών των ετικετών γίνεται στο διακομιστή, και δεν αποτελεί τμήμα της εξόδου που αποστέλλεται στον πελάτη.

Σύγχυση μεταξύ του χρόνου μετάφρασης και του χρόνου αίτησης

Η σελίδα JSP μετατρέπεται σε μικροϋπηρεσία. Η μικροϋπηρεσία μεταγλωττίζεται, φορτώνεται στη μνήμη του διακομιστή, παίρνει αρχικές τιμές, και εκτελείται. Πότε όμως συμβαίνει το κάθε βήμα; Για να απαντήσετε σε αυτή την ερώτηση, θυμηθείτε δύο σημεία:

- Η σελίδα JSP μεταφράζεται σε μικροϋπηρεσία και μεταγλωττίζεται μόνο την πρώτη φορά που προσπελάζεται, μετά από την τελευταία τροποποίησή της.
- Η φόρτωση στη μνήμη, η απόδοση αρχικών τιμών, και η εκτέλεση ακολουθεί τους κανονικούς κανόνες των μικροϋπηρεσιών.

Ο Πίνακας 10.1 παραθέτει μερικά συνηθισμένα σενάρια και αναφέρει πότε συμβαίνει ή δεν συμβαίνει το κάθε βήμα στο συγκεκριμένο σενάριο. Έχουν επισημανθεί με έντονη γραφή οι καταχωρίσεις για τις οποίες συμβαίνουν συχνότερα παρανοήσεις. Όταν συμβουλεύεστε τον πίνακα, προσέξτε ότι οι μικροϋπηρεσίες που προκύπτουν από σελίδες JSP χρησιμοποιούν τη μέθοδο `_jspService` (η οποία καλείται και για τις αιτήσεις `GET` και για τις αιτήσεις `POST`), και όχι τις μεθόδους `doGet` ή `doPost`. Επίσης, για την απόδοση αρχικών τιμών χρησιμοποιούν τη μέθοδο `jspInit`, και όχι τη μέθοδο `Init`.

Πίνακας 10.1 Λειτουργίες JSP σε διάφορα σενάρια

	Μετάφραση της σελίδας JSP σε μικροϋπηρεσία	Μεταγλώττιση της μικροϋπηρεσίας	Φόρτωση της μικροϋπηρεσίας στη μνήμη του διακομιστή	Κλήση της `jspInit`	Κλήση της `_jspService`
Συγγραφή της σελίδας για πρώτη φορά					
Αίτηση 1	Ναι	Ναι	Ναι	Ναι	Ναι
Αίτηση 2	Όχι	Όχι	Όχι	Όχι	Ναι
Επανεκκίνηση διακομιστή					
Αίτηση 3	Όχι	Όχι	Ναι	Ναι	Ναι
Αίτηση 4	Όχι	Όχι	Όχι	Όχι	Ναι
Τροποποίηση σελίδας					
Αίτηση 5	Ναι	Ναι	Ναι	Ναι	Ναι
Αίτηση 6	Όχι	Όχι	Όχι	Όχι	Ναι

Θεώρηση ότι η JSP είναι από μόνη της αρκετή

Υπάρχει μια μικρή κοινότητα προγραμματιστών που είναι τόσο ερωτευμένοι με την τεχνολογία JSP ώστε τη χρησιμοποιούν σχεδόν για τα πάντα. Οι περισσότεροι από αυτούς τους προγραμματιστές δεν χρησιμοποιούν *ποτέ* μικροϋπηρεσίες· πολλοί δεν χρησιμοποιούν καν τις βοηθητικές κλάσεις — δημιουργούν απλώς μεγάλες περίπλοκες σελίδες JSP για όλες τις εργασίες.

Κάτι τέτοιο, όμως, είναι λανθασμένο. Η JSP είναι ένα εξαιρετικό εργαλείο. Όμως το θεμελιώδες πρόβλημα που έρχεται να αντιμετωπίσει είναι η *παρουσίαση*: η δυσκολία δημιουργίας και συντήρησης του κώδικα HTML για την παρουσίαση των αποτελεσμάτων μιας αίτησης. Η JSP αποτελεί καλή επιλογή για σελίδες με σχετικά σταθερή μορφή και μεγάλη ποσότητα στατικού κειμένου. Από μόνη της, η JSP είναι λιγότερο καλή για εφαρμογές που έχουν μεταβλητή δομή και παράγουν έξοδο δυαδικών δεδομένων ή χειρίζονται το πρωτόκολλο HTTP χωρίς ρητή έξοδο (όπως στη μικροϋπηρεσία της Ενότητας 6.4 για τις μηχανές αναζήτησης). Από την άλλη πλευρά, άλλες εφαρμογές μπορούν να υλοποιηθούν καλύτερα με ένα συνδυασμό των μικροϋπηρεσιών και των σελίδων JSP (δείτε το Κεφάλαιο 15), και όχι μόνο με τις μικροϋπηρεσίες ή μόνο με σελίδες JSP.

Η JSP είναι ένα ισχυρό και ευρέως χρησιμοποιούμενο εργαλείο. Παρόλα αυτά, άλλα εργαλεία είναι μερικές φορές καλύτερα. Επιλέξτε το κατάλληλο εργαλείο για την κάθε εργασία.

Θεώρηση ότι μόνο οι μικροϋπηρεσίες είναι αρκετές

Από την άλλη μεριά του φάσματος, απέναντι στο στρατόπεδο των οπαδών της χρήσης μόνο σελίδων JSP βρίσκεται το στρατόπεδο των οπαδών της χρήσης μόνο μικροϋπηρεσιών. Οι οπαδοί αυτού του στρατοπέδου θεωρούν, αρκετά σωστά, ότι οι σελίδες JSP δεν είναι τίποτα άλλο παρά μεταμφιεσμένες μικροϋπηρεσίες, οπότε δεν μπορούν να επιτύχουν κάτι που δεν θα μπορούσε να επιτευχθεί με τις μικροϋπηρεσίες. Από αυτό συμπεραίνουν ότι θα πρέπει να παραμείνει κανείς στις μικροϋπηρεσίες, όπου έχει πρόσβαση σε ολόκληρη την εσωτερική ισχύ, διαθέτει ολοκληρωμένο έλεγχο, και μπορεί να δει τις ακριβώς συμβαίνει. Για σταθείτε, μήπως έχετε ακούσει ξανά αυτό το επιχείρημα; Δεν θυμίζει τις κραυγές των φανατικών: "μην είστε χαζοί· γράψτε όλο τον κώδικά σας σε γλώσσα assembly";

Πράγματι, θα μπορούσατε να χρησιμοποιήσετε μικροϋπηρεσίες για οποιαδήποτε εργασία στην οποία χρησιμοποιείται η τεχνολογία JSP. Όμως δεν είναι πάντοτε εξίσου βολικό να κάνετε κάτι τέτοιο. Για εργασίες που περιλαμβάνουν μεγάλο τμήμα στατικού περιεχομένου HTML, η χρήση της τεχνολογίας JSP (ή ενός συνδυασμού JSP και μικροϋπηρεσιών) απλοποιεί τη δημιουργία και τη συντήρηση του κώδικα HTML, σας δίνει τη δυνατότητα να χρησιμοποιήσετε καθιερωμένα εργαλεία δημιουργίας τοποθεσιών Ιστού, και σας επιτρέπει να χρησιμοποιήσετε την τακτική του "διαίρει και βασίλευε", με το διαχωρισμό του έργου μεταξύ προγραμματιστών Java και προγραμματιστών Ιστού.

Οι μικροϋπηρεσίες είναι ισχυρά και ευρέως χρησιμοποιούμενα εργαλεία. Παρόλα αυτά, άλλα εργαλεία είναι μερικές φορές καλύτερα. Επιλέξτε το κατάλληλο εργαλείο για την κάθε εργασία.

10.5 Εγκατάσταση των σελίδων JSP

Οι μικροϋπηρεσίες απαιτούν τη ρύθμιση της μεταβλητής `CLASSPATH`, τη χρήση πακέτων για την αποφυγή διενέξεων ονομάτων, την εγκατάσταση των αρχείων κλάσεων σε συγκεκριμένες θέσεις στο διακομιστή, και τη χρήση ειδικών URL. Κάτι τέτοιο δεν ισχύει για τις σελίδες JSP. Οι σελίδες JSP μπορούν να τοποθετηθούν στους ίδιους καταλόγους που τοποθετούνται οι κανονικές σελίδες HTML, οι εικόνες, και τα φύλλα στυλ· μπορούν επίσης να προσπελαστούν μέσω URL της ίδιας μορφής με εκείνα που χρησιμοποιούνται για σελίδες HTML, εικόνες, και φύλλα στυλ. Θα παραθέσουμε εδώ μερικά παραδείγματα προεπιλεγμένων θέσεων εγκατάστασης (δηλαδή, θέσεις που ισχύουν όταν δεν χρησιμοποιούνται προσαρμοσμένες εφαρμογές Ιστού) και τα αντίστοιχα URL. Όπου αναφέρουμε *ΚάποιοςΚατάλογος* μπορείτε να χρησιμοποιήσετε οποιοδήποτε όνομα καταλόγου θέλετε, εκτός από WEB-INF και META-INF που δεν μπορούν να χρησιμοποιηθούν ως ονόματα καταλόγων. Όταν χρησιμοποιείτε την προεπιλεγμένη εφαρμογή Ιστού, θα πρέπει επίσης να αποφεύγετε ονόματα καταλόγων που ταιριάζουν με το πρόθεμα URL οποιασδήποτε εφαρμογής Ιστού. Για πληροφορίες σχετικά με τον ορισμό των δικών σας εφαρμογών Ιστού δείτε την Ενότητα 2.11.

Κατάλογοι JSP για τον Tomcat (Προεπιλεγμένη εφαρμογή Ιστού)

- **Κύρια θέση**
  κατάλογος_εγκατάστασης/webapps/ROOT
- **Αντίστοιχο URL**
  http://*υπολογιστήςΥπηρεσίας*/*ΚάποιοΑρχείο.jsp*
- **Πιο ειδική θέση (Τυχαίος υποκατάλογος)**
  κατάλογος_εγκατάστασης/webapps/ROOT/*ΚάποιοςΚατάλογος*
- **Αντίστοιχο URL**
  http://*υπολογιστήςΥπηρεσίας*/*ΚάποιοςΚατάλογος*/*ΚάποιοΑρχείο.jsp*

Κατάλογοι JSP για το JRun (Προεπιλεγμένη εφαρμογή Ιστού)

- **Κύρια θέση**
  κατάλογος_εγκατάστασης/doc
- **Αντίστοιχο URL**
  http://*υπολογιστήςΥπηρεσίας*/*ΚάποιοΑρχείο.jsp*
- **Πιο ειδική θέση (Τυχαίος υποκατάλογος)**
  κατάλογος_εγκατάστασης/doc/*ΚάποιοςΚατάλογος*
- **Αντίστοιχο URL**
  http://*υπολογιστήςΥπηρεσίας*/*ΚάποιοςΚατάλογος*/*ΚάποιοΑρχείο.jsp*

Κατάλογοι JSP για το Resin (Προεπιλεγμένη εφαρμογή Ιστού)

- **Κύρια θέση**
  κατάλογος_εγκατάστασης/servers/default-ear/default-war
- **Αντίστοιχο URL**
  http://*υπολογιστήςΥπηρεσίας*/*ΚάποιοΑρχείο.jsp*
- **Πιο ειδική θέση (Τυχαίος υποκατάλογος)**
  κατάλογος_εγκατάστασης/servers/default-ear/default-war/*ΚάποιοςΚατάλογος*
- **Αντίστοιχο URL**
  http://*υπολογιστήςΥπηρεσίας*/*ΚάποιοςΚατάλογος*/*ΚάποιοΑρχείο.jsp*

Προσέξτε ότι, αν και οι *ίδιες* οι σελίδες JSP δεν χρειάζονται κάποιον ειδικό κατάλογο εγκατάστασης, οι κλάσεις Java που καλούνται *από* σελίδες JSP θα πρέπει να βρίσκονται στις καθιερωμένες θέσεις που χρησιμοποιούνται από τις κλάσεις των μικροϋπηρεσιών (για παράδειγμα, /Web-INF/classes/*ΚατάλογοςΠουΤαιριάζειΜεΤοΌνομαΤουΠακέτου*· δείτε σχετικά την Ενότητα 2.10). Σημειώστε επίσης ότι οι κλάσεις Java που χρησιμοποιούνται από σελίδες JSP θα πρέπει *πάντοτε* να ανήκουν σε πακέτα· θα αναλύσουμε περισσότερο το θέμα αυτό σε επόμενα κεφάλαια.

10.6 Βασική σύνταξη

Ακολουθεί μια σύντομη σύνοψη των διαφόρων δομών JSP που θα συναντήσετε στο βιβλίο.

Κείμενο HTML

- **Περιγραφή:**
  Περιεχόμενο HTML που μεταβιβάζεται αναλλοίωτο στον πελάτη
- **Παράδειγμα:**
  `<H1>Κάτι</H1>`
- **Εξετάζεται στην:**
  Ενότητα 11.1

Σχόλια HTML

- **Περιγραφή:**
  Σχόλιο HTML που αποστέλλεται στον πελάτη αλλά δεν εμφανίζεται από το φυλλομετρητή
- **Παράδειγμα:**
  `<!-- Κάτι -->`
- **Εξετάζεται στην:**
  Ενότητα 11.1

Κείμενο Προτύπου

- **Περιγραφή:**
  Περιεχόμενο HTML που μεταβιβάζεται αναλλοίωτο στον πελάτη
- **Παράδειγμα:**
  Οτιδήποτε άλλο εκτός από τη σύνταξη των υποενοτήτων που ακολουθούν
- **Εξετάζεται στην:**
  Ενότητα 11.1

Σχόλια JSP

- **Περιγραφή:**
  Σχόλιο του προγραμματιστή που δεν αποστέλλεται στον πελάτη
- **Παράδειγμα:**
  `<%-- Κάτι --%>`
- **Εξετάζεται στην:**
  Ενότητα 11.1

Παράσταση JSP

- **Περιγραφή:**
  Παράσταση που υπολογίζεται και αποστέλλεται στον πελάτη κάθε φορά που γίνεται αίτηση της σελίδας
- **Παράδειγμα:**
  `<%= Τιμή Java %>`
- **Εξετάζεται στην:**
  Ενότητα 11.4

Μικροσενάριο JSP

- **Περιγραφή:**
  Εντολή ή εντολές που εκτελούνται κάθε φορά που γίνεται αίτηση της σελίδας
- **Παράδειγμα:**
  `<%= Εντολή Java %>`
- **Εξετάζεται στην:**
  Ενότητα 11.7

Δήλωση JSP

- **Περιγραφή:**
  Πεδίο ή μέθοδος που γίνεται τμήμα του ορισμού της κλάσης όταν η σελίδα μεταφράζεται σε μικροϋπηρεσία
- **Παραδείγματα:**
  `<%! Ορισμός Πεδίου %>`
  `<%! Ορισμός Μεθόδου %>`
- **Εξετάζεται στην:**
  Ενότητα 11.10

Οδηγία JSP

- **Περιγραφή:**
  Πληροφορίες υψηλού επιπέδου σχετικά με τη δομή του κώδικα της μικροϋπηρεσίας (`page`), του κώδικα που περιλαμβάνεται κατά το χρόνο μετάφρασης (`include`), ή τις βιβλιοθήκες προσαρμοσμένων ετικετών που χρησιμοποιούνται (`taglib`).
- **Παράδειγμα:**
  `<%@ directive att="val" %>`
- **Εξετάζονται:**
  `page`: Κεφάλαιο 12
  `include`: Κεφάλαιο 13
  `taglib` και `tag`: βιβλίο More Servlets and JavaServer Pages

Ενέργεια JSP

- **Περιγραφή:**
  Ενέργεια που πραγματοποιείται όταν γίνεται αίτηση της σελίδας
- **Παράδειγμα:**
  `<jsp:blah>...</jsp:blah>`
- **Εξετάζονται:**
  `jsp:include` και τα σχετικά: Κεφάλαιο 13
  `jsp:useBean` και τα σχετικά: Κεφάλαιο 14
  `jsp:invoke` και τα σχετικά: βιβλίο More Servlets and JavaServer Pages

Στοιχείο γλώσσας παραστάσεων JSP

- **Περιγραφή:**
  Συντομευμένη παράσταση JSP
- **Παράδειγμα:**
  `${ EL Παράσταση }`
- **Εξετάζεται στο:**
  Κεφάλαιο 16

Προσαρμοσμένη ετικέτα (Προσαρμοσμένη ενέργεια)

- **Περιγραφή:**
  Ενεργοποίηση προσαρμοσμένης ετικέτας
- **Παράδειγμα:**
  `<prefix:όνομα>`
  `Σώμα`
  `</prefix:όνομα>`
- **Εξετάζεται στο:**
  βιβλίο More Servlets and JavaServer Pages

Κείμενο προτύπου με ακολουθία διαφυγής

- **Περιγραφή:**
  Κείμενο που διαφορετικά θα ερμηνευόταν με ειδικό τρόπο. Αφαιρείται η κάθετος και το κείμενο που απομένει αποστέλλεται στον πελάτη
- **Παραδείγματα:**
  `<\%`
  `%\>`
- **Εξετάζεται στην:**
  Ενότητα 11.1

ΚΛΗΣΗ ΚΩΔΙΚΑ JAVA ΜΕ ΣΤΟΙΧΕΙΑ ΣΕΝΑΡΙΟΥ JSP

Θέματα σε αυτό το Κεφάλαιο

- Σύγκριση στατικού και δυναμικού κειμένου
- Δυναμικός κώδικας και καλός σχεδιασμός σελίδων JSP
- Η σημασία των πακέτων για τις βοηθητικές κλάσεις JSP
- Παραστάσεις JSP
- Μικροσενάρια JSP
- Δηλώσεις JSP
- Κώδικας μικροϋπηρεσίας που προκύπτει από στοιχεία σεναρίου JSP
- Μικροσενάρια και κείμενο υπό συνθήκη
- Προκαθορισμένες μεταβλητές
- Σύγκριση μικροϋπηρεσιών και σελίδων JSP για παρόμοιες εργασίες

Κεφάλαιο 11

Αυτό το κεφάλαιο εξετάζει την "κλασική" προσέγγιση όσον αφορά την κλήση κώδικα Java από το εσωτερικό σελίδων JSP. Αυτή η προσέγγιση λειτουργεί τόσο στην JSP 1 (δηλαδή, JSP 1.2 και προηγούμενες εκδόσεις) όσο και στην JSP 2.0. Στο Κεφάλαιο 16 θα εξετάσουμε τη γλώσσα παραστάσεων JSP, η οποία παρέχει ένα συμπαγή μηχανισμό για την έμμεση κλήση κώδικα Java αλλά μόνο για την έκδοση JSP 2.0 ή μεταγενέστερη.

11.1 Δημιουργία κειμένου προτύπου

Στις περισσότερες περιπτώσεις, ένα μεγάλο ποσοστό του εγγράφου σας JSP αποτελείται από στατικό κείμενο (συνήθως κώδικας HTML), που είναι γνωστό ως *κείμενο προτύπου* (template text). Σχεδόν από όλες τις απόψεις αυτός ο κώδικας HTML μοιάζει με κανονική γλώσσα HTML, ακολουθεί τους ίδιους κανόνες σύνταξης, και απλώς "μεταβιβάζεται" στον πελάτη από τη μικροϋπηρεσία που δημιουργείται για το χειρισμό της σελίδας. Όχι μόνο μοιάζει με κανονικός κώδικας HTML, αλλά μπορεί και να δημιουργηθεί με όποιο εργαλεία χρησιμοποιείτε ήδη για την κατασκευή ιστοσελίδων. Για παράδειγμα, εμείς χρησιμοποιήσαμε το Macromedia Dreamweaver για πολλές σελίδες JSP αυτού του βιβλίου.

Υπάρχουν δύο μικρές εξαιρέσεις στον κανόνα "το κείμενο προτύπου μεταβιβάζεται ως έχει". Πρώτον, αν θέλετε στην έξοδο να εμφανιστεί το κείμενο <% ή %>, θα πρέπει να τοποθετήσετε το αλφαριθμητικό <\% ή %\> στο κείμενο προτύπου. Δεύτερον, αν θέλετε κάποιο σχόλιο να εμφανιστεί στη σελίδα JSP αλλά όχι στο έγγραφο που θα προκύψει, χρησιμοποιήστε την εντολή

```
<%-- Σχόλιο JSP --%>
```

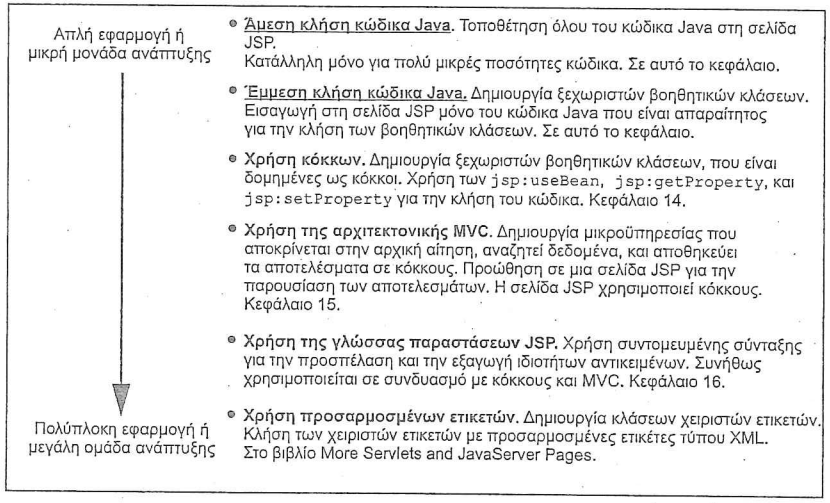
Τα σχόλια HTML της μορφής

```
<!-- Σχόλιο HTML -->
```

μεταβιβάζονται κανονικά στον πελάτη.

11.2 Κλήση κώδικα Java από σελίδες JSP

Υπάρχουν αρκετοί διαφορετικοί τρόποι παραγωγής δυναμικού περιεχομένου από σελίδες JSP, όπως φαίνεται στην Εικόνα 11-1. Κάθε μία από αυτές τις προσεγγίσεις έχει τη δική της θέση· το μέγεθος και η πολυπλοκότητα του έργου αποτελούν τον πιο σημαντικό παράγοντα για να αποφασίσετε ποια προσέγγιση είναι η πιο κατάλληλη. Ωστόσο, θα πρέπει να θυμάστε ότι το πιο συχνό σφάλμα είναι η τοποθέτηση πολύ κώδικα κατευθείαν στη σελίδα, παρά το αντίθετο. Αν και η τοποθέτηση λίγου κώδικα Java κατευθείαν στις σελίδες JSP λειτουργεί πολύ καλά για τις απλές εφαρμογές, η χρήση μεγάλων και πολύπλοκων μπλοκ κώδικα Java σε σελίδες JSP δημιουργεί ένα αποτέλεσμα που είναι δύσκολο στη συντήρηση, την αποσφαλμάτωση, την επαναχρησιμοποίηση, και την κατανομή της εργασίας μεταξύ των διαφόρων μελών της ομάδας ανάπτυξης. Για λεπτομέρειες δείτε την Ενότητα 11.3 (Περιορισμός της ποσότητας κώδικα Java σε σελίδες JSP). Παρόλα αυτά, πολλές σελίδες είναι αρκετά απλές, και οι δύο πρώτες προσεγγίσεις της Εικόνας 11-1 (ρητή τοποθέτηση κώδικα Java κατευθείαν στη σελίδα) λειτουργούν πολύ καλά. Σε αυτό το κεφάλαιο θα εξετάσουμε αυτές τις προσεγγίσεις.



Εικόνα 11-1 Στρατηγικές κλήσης κώδικα Java από σελίδες JSP.

Τύποι στοιχείων σεναρίου JSP

Τα στοιχεία σεναρίου JSP (JSP scripting elements) σας επιτρέπουν την εισαγωγή κώδικα Java στη μικροϋπηρεσία που θα παραχθεί από τη σελίδα JSP. Υπάρχουν τρεις μορφές τέτοιων στοιχείων:

1. **Παραστάσεις** (expressions) της μορφής <%= *Παράσταση Java* %>, που υπολογίζονται και εισάγονται στην έξοδο της μικροϋπηρεσίας.
2. **Μικροσενάρια** (scriplets) της μορφής <% *Κώδικας Java* %>, που εισάγονται στη μέθοδο `_jspService` της μικροϋπηρεσίας (η οποία καλείται από τη `service`).
3. **Δηλώσεις** (declarations) της μορφής <%! *Δήλωση Πεδίου/Μεθόδου* %>, που εισάγονται στο σώμα της κλάσης της μικροϋπηρεσίας, έξω από τις υπάρχουσες μεθόδους.

Όλα αυτά τα στοιχεία σεναρίου περιγράφονται με περισσότερες λεπτομέρειες στις ενότητες που ακολουθούν.

11.3 Περιορισμός της ποσότητας κώδικα Java σε σελίδες JSP

Έχετε 25 γραμμές κώδικα Java που θα πρέπει να καλέσετε. Έχετε δύο επιλογές: (1) τοποθέτηση των 25 γραμμών κατευθείαν στη σελίδα JSP, ή (2) τοποθέτηση των 25 γραμμών κώδικα σε μια ξεχωριστή κλάση Java, τοποθέτηση της κλάσης στον κατάλογο WEB-INF/classes/*ΚατάλογοςΠουΤαιριάζειΜεΤοΌνομαΤουΠακέτου*, και χρήση μίας ή δύο γραμμών κώδικα Java στη σελίδα JSP για την πραγματοποίηση της κλήσης. Ποια λύση είναι καλύτερη; Η δεύτερη. Η δεύτερη. Η δεύτερη! Και όσο περισσότερες γραμμές κώδικα έχετε — 50, 100, 500 κ.λπ — τόσο καλύτερη είναι. Οι λόγοι είναι οι εξής:

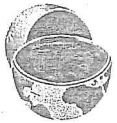
- **Ανάπτυξη.** Γενικά θα γράφετε τις κανονικές κλάσεις σε ένα περιβάλλον ανάπτυξης για Java (για παράδειγμα, ένα IDE όπως τα JBuilder ή Eclipse ή κάποιο διορθωτή κώδικα όπως οι UltraEdit ή emacs), ενώ θα γράφετε τις σελίδες JSP σε ένα περιβάλλον ανάπτυξης για HTML, όπως το Dreamweaver. Το περιβάλλον για τη Java είναι κατά κανόνα καλύτερο σε θέματα όπως το κλείσιμο παρενθέσεων, η παροχή επιτόπιας βοήθειας, ο έλεγχος της σύνταξης, ο χρωματισμός του κώδικα, κ.ο.κ.
- **Μεταγλώττιση.** Για να μεταγλωττίσετε μια κανονική κλάση Java πατάτε στο κουμπί Build (Δόμηση) του IDE ή καλείτε το μεταγλωττιστή `javac`. Για να μεταγλωττίσετε μια σελίδα JSP πρέπει να τη μεταφέρετε στον κατάλληλο κατάλογο, να ξεκινήσετε το διακομιστή, να ανοίξετε το φυλλομετρητή, και να πληκτρολογήσετε το κατάλληλο URL.
- **Αποσφαλμάτωση.** Ξέρουμε ότι ποτέ δε συμβαίνει σε σας, αλλά όταν *εμείς* γράφουμε κλάσεις Java ή σελίδες JSP, μερικές φορές κάνουμε συντακτικά λάθη. Αν υπάρχει συντακτικό λάθος στον ορισμό μιας κανονικής κλάσης, ο μεταγλωττιστής θα σας ενημερώσει αμέσως και θα επισημάνει επίσης τη γραμμή του κώδικα όπου έχει συμβεί το

σφάλμα. Αν υπάρχει συντακτικό λάθος σε μια σελίδα JSP, ο διακομιστής συνήθως θα σας ενημερώσει για τη γραμμή της *μικροϋπηρεσίας* (δηλαδή, τη μικροϋπηρεσία στην οποία μεταφράστηκε η σελίδα JSP) που περιέχει το σφάλμα. Για την παρακολούθηση της εξόδου κατά το χρόνο εκτέλεσης, με τις κανονικές κλάσεις μπορείτε να χρησιμοποιήσετε απλές εντολές `System.out.println`, στην περίπτωση που το IDE που χρησιμοποιείτε δεν προσφέρει κάτι καλύτερο. Στις σελίδες JSP μπορείτε μερικές φορές να χρησιμοποιήσετε εντολές εκτύπωσης, αλλά η θέση όπου θα εμφανιστεί η έξοδος διαφέρει από διακομιστή σε διακομιστή.

- **Κατανομή της εργασίας.** Πολλές μεγάλες ομάδες ανάπτυξης αποτελούνται από άτομα που είναι ειδικευμένα στη γλώσσα Java και άλλα άτομα που έχουν εξειδίκευση στην HTML αλλά έχουν ελάχιστη ή και καθόλου γνώση της Java. Όσο περισσότερος κώδικας Java βρίσκεται απευθείας στη σελίδα, τόσο δυσκολότερο θα είναι για τους σχεδιαστές Ιστού (που έχουν ειδίκευση στην HTML) να χειριστούν τη σελίδα.
- **Έλεγχος.** Υποθέσετε ότι θέλετε να δημιουργήσετε μια σελίδα JSP που να εμφανίζει τυχαίους ακέραιους αριθμούς από το 1 μέχρι κάποιο όριο (συμπεριλαμβανομένου του ορίου). Χρησιμοποιείτε τη μέθοδο `Math.random`, πολλαπλασιάζετε επί το εύρος, μετατρέπετε το αποτέλεσμα σε `int`, και προσθέτετε τη μονάδα. Μέχρι εδώ καλά. Είστε όμως σίγουροι; Αν το κάνετε κατευθείαν σε μια σελίδα JSP, θα πρέπει να καλείτε συνεχώς τη σελίδα για να δείτε αν θα πάρετε τους αριθμούς στο καθορισμένο εύρος, και όχι έξω από αυτό. Αφού πατήσετε μερικές φορές στο κουμπί Ανανέωσης, στο τέλος θα βαρεθείτε. Αν όμως το κάνετε αυτό σε μια στατική μέθοδο μιας κανονικής κλάσης Java, μπορείτε να γράψετε μια ρουτίνα ελέγχου η οποία θα καλεί τη μέθοδο από το εσωτερικό ενός βρόχου (δείτε τη Λίστα 11.13) και στη συνέχεια θα μπορείτε να εκτελέσετε εκατοντάδες ή και χιλιάδες ελέγχους χωρίς κανένα πρόβλημα. Για περισσότερο πολύπλοκες μεθόδους, μπορείτε να αποθηκεύσετε την έξοδο και, κάθε φορά που αλλάζετε τη μέθοδο, να συγκρίνετε τα νέα αποτελέσματα με αυτά που έχετε αποθηκεύσει.
- **Επαναχρησιμοποίηση.** Τοποθετείτε κάποιο τμήμα κώδικα σε μια σελίδα JSP. Αργότερα ανακαλύπτετε ότι πρέπει να κάνετε το ίδιο πράγμα σε μία διαφορετική σελίδα JSP. Τι θα κάνετε; Αποκοπή και επικόλληση; Μπα! Η επανάληψη κώδικα κατ' αυτόν τον τρόπο αποτελεί "έγκλημα καθοσιώσεως", επειδή αν (όταν!) αλλάξετε την προσέγγισή σας θα πρέπει να κάνετε αλλαγές στον κώδικα σε πολλές διαφορετικές θέσεις. Η αντιμετώπιση του προβλήματος της επαναχρησιμοποίησης κώδικα είναι η ουσία του αντικειμενοστρεφούς προγραμματισμού. Δεν πρέπει να ξεχνάτε αυτές τις αρχές μόνο και μόνο επειδή χρησιμοποιείτε σελίδες JSP για να απλοποιήσετε την παραγωγή σελίδων HTML.

"Για στάσου", θα πείτε. "Έχω ένα IDE που κάνει ευκολότερη την ανάπτυξη, αποσφαλμάτωση, και μεταγλώττιση σελίδων JSP." Σύμφωνοι, αυτή είναι μια καλή παρατήρηση. Δεν υπάρχει κάποιος απαράβατος κανόνας για το πόσος κώδικας Java θα πρέπει να βρίσκεται σε μια σελίδα. Κανένα IDE, όμως, δεν θα λύσει το πρόβλημα του ελέγχου και της επαναχρησιμοποίησης, και η στρατηγική του γενικού σχεδιασμού θα πρέπει να επικεντρώνεται στην τοποθέτηση του πολύ-

πλοκου κώδικα σε κανονικές κλάσεις Java, και τη δημιουργία όσο το δυνατόν απλούστερων σελίδων JSP.



Η προσέγγιση του βιβλίου

Περιορίστε την ποσότητα του κώδικα Java που βρίσκεται στις σελίδες JSP. Τουλάχιστον, χρησιμοποιήστε βοηθητικές κλάσεις που καλούνται από τις σελίδες JSP. Αφού αποκτήσετε αρκετή πείρα, δοκιμάστε τους κόκκους, την αρχιτεκτονική MVC, και τις προσαρμοσμένες ετικέτες.

Σχεδόν όλοι οι έμπειροι προγραμματιστές έχουν συναντήσει υπερβολές: σελίδες JSP που αποτελούνται από πολλές γραμμές κώδικα Java με ψήγματα κώδικα HTML. Αυτό είναι προφανώς κακό: κάνει δυσκολότερη την ανάπτυξη, τη μεταγλώττιση, την αποσφαλμάτωση, το διαχωρισμό του έργου μεταξύ των μελών της ομάδας, τον έλεγχο, και την επαναχρησιμοποίηση. Μια μικροϋπηρεσία θα ήταν πολύ καλύτερη λύση. Ωστόσο, ορισμένοι από αυτούς τους προγραμματιστές έχουν φτάσει στα άκρα λέγοντας ότι είναι *πάντοτε* λάθος να υπάρχει *οποιοσδήποτε* κώδικας Java κατευθείαν σε μια σελίδα JSP. Σίγουρα, για κάποια έργα αξίζει τον κόπο να υπάρχει αυστηρός διαχωρισμός μεταξύ περιεχομένου και παρουσίασης και να επιβάλλεται ένα στυλ όπου δεν υπάρχει σύνταξη Java μέσα στις σελίδες JSP. Όμως κάτι τέτοιο δεν είναι πάντοτε αναγκαίο, ούτε καν ωφέλιμο.

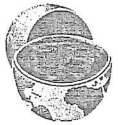
Ορισμένοι προχωρούν ακόμη πιο μακριά, δηλώνοντας ότι *όλες* οι σελίδες *όλων* των εφαρμογών θα πρέπει να χρησιμοποιούν την αρχιτεκτονική Model-View-Controller (MVC), κατά προτίμηση με το πλαίσιο εργασιών Apache Struts. Αυτό, κατά τη γνώμη μας, είναι υπερβολικό. Ναι, η αρχιτεκτονική MVC (Κεφάλαιο 15) είναι μια καλή ιδέα, και τη χρησιμοποιούμε συνεχώς στα πραγματικά έργα. Και πράγματι το Struts (δείτε το βιβλίο More Servlets and JavaServer Pages) είναι ένα ωραίο πλαίσιο εργασιών· καθώς τυπωνόταν αυτό το βιβλίο, το χρησιμοποιούσαμε σε ένα μεγάλο έργο. Αυτές οι προσεγγίσεις είναι σπουδαίες όταν η κατάσταση έχει μέτρια (γενικά MVC) ή μεγάλη πολυπλοκότητα (Struts).

Τα απλά προβλήματα, όμως, θέλουν απλές λύσεις. Κατά τη γνώμη μας, όλες οι προσεγγίσεις της Εικόνας 11-1 έχουν κάποια αξία· η επιλογή εξαρτάται κυρίως από την πολυπλοκότητα της εφαρμογής και από το μέγεθος της ομάδας ανάπτυξης. Παρόλα αυτά, θα πρέπει να θυμάστε το εξής: οι αρχάριοι είναι πιο πιθανό να πέσουν στο λάθος της δημιουργίας σελίδων που είναι δύσκολες στη συντήρηση και γεμάτες κώδικα Java, παρά να χρησιμοποιήσουν χωρίς λόγο μεγάλα και περίπλοκα πλαίσια εργασιών.

Η σημασία της χρήσης πακέτων

Όποτε γράφετε κλάσεις Java, τα αρχεία των κλάσεων θα πρέπει να τοποθετούνται στον κατάλογο WEB-INF/classes/*ΚατάλογοςΠουΤαιριάζειΜεΤοΌνομαΤουΠακέτου* (ή σε ένα αρχείο JAR που τοποθετείται στον κατάλογο WEB-INF/lib). Αυτό ισχύει ανεξάρτητα από το αν η κλάση είναι σε μια μικροϋπηρεσία, σε μια κανονική βοηθητική κλάση, σε κόκκο, σε χειριστή προσαρμοσμένων ετικετών, ή σε κάτι άλλο. Όλος ο κώδικας τοποθετείται στην ίδια θέση.

Με τις κανονικές μικροϋπηρεσίες, όμως, είναι λογικό μερικές φορές να χρησιμοποιείτε το προεπιλεγμένο πακέτο, αφού μπορείτε να χρησιμοποιήσετε ξεχωριστές εφαρμογές Ιστού (δείτε την Ενότητα 2.11) για να αποφύγετε τη διένεξη ονομάτων με τις μικροϋπηρεσίες των άλλων έργων. Στην περίπτωση όμως του κώδικα που καλείται από σελίδες JSP, θα πρέπει πάντοτε να χρησιμοποιείτε πακέτα. Άλλωστε, αφού όταν γράφετε κάποια βοηθητική κλάση για να τη χρησιμοποιήσετε από μια μικροϋπηρεσία δεν γνωρίζετε αν αργότερα θα χρησιμοποιηθεί και από κάποια σελίδα JSP, η στρατηγική αυτή σημαίνει ότι θα πρέπει *πάντοτε* να χρησιμοποιείτε πακέτα για *όλες* τις κλάσεις, είτε αυτές χρησιμοποιούνται από μικροϋπηρεσίες, είτε χρησιμοποιούνται από σελίδες JSP.



Η προσέγγιση του βιβλίου

Τοποθετήστε όλες τις κλάσεις σας σε πακέτα.

Γιατί; Για να απαντηθεί αυτή η ερώτηση, δείτε τον κώδικα που ακολουθεί. Ο κώδικας μπορεί να περιέχει ή να μην περιέχει μια δήλωση `package`, αλλά δεν περιέχει εντολές `import`.

```
...
public class SomeClass {
  public String someMethod(...) {
    SomeHelperClass test = new SomeHelperClass(...);
    String someString = SomeUtilityClass.someStaticMethod(...);
    ...
  }
}
```

Το ερώτημα είναι τώρα σε ποιο πακέτο θα θεωρήσει το σύστημα ότι βρίσκονται οι κλάσεις `SomeHelperClass` και `SomeUtilityClass`; Η απάντηση είναι: σε όποιο πακέτο βρίσκεται η `SomeClass`. Και ποιο πακέτο είναι αυτό; Ότι έχει δοθεί στη δήλωση `package`. Όλα καλά. Αυτή είναι στοιχειώδης σύνταξη Java — κανένα πρόβλημα. Ωραία λοιπόν, δείτε τον ακόλουθο κώδικα σελίδας JSP:

```
...
<%
  SomeHelperClass test = new SomeHelperClass(...);
  String someString = SomeUtilityClass.someStaticMethod(...);
%>
```

Η ίδια ερώτηση τώρα: σε ποιο πακέτο θα θεωρήσει το σύστημα ότι βρίσκονται οι κλάσεις `SomeHelperClass` και `SomeUtilityClass`; Ίδια απάντηση: σε όποιο πακέτο βρίσκεται η τρέχουσα κλάση (η μικροϋπηρεσία στην οποία έχει μεταφραστεί η σελίδα JSP). Ποιο πακέτο είναι αυτό; Μμμ, καλή ερώτηση. Κανείς δεν ξέρει! Το πακέτο δεν είναι κάτι που είναι τυποποιημένο από τις προδιαγραφές της τεχνολογίας JSP. Οπότε, όταν χρησιμοποιούνται κατ' αυτόν τον

τρόπο βοηθητικές κλάσεις χωρίς πακέτα (packageless), αυτές θα λειτουργήσουν μόνο αν το σύστημα δομήσει μια μικροϋπηρεσία χωρίς πακέτα. Όμως αυτό δεν συμβαίνει πάντοτε, οπότε κώδικας JSP όπως αυτός του παραδείγματος μπορεί να αστοχήσει. Για να γίνουν ακόμη χειρότερα τα πράγματα, οι διακομιστές μερικές φορές *δομούν* μικροϋπηρεσίες χωρίς πακέτα από σελίδες JSP. Για παράδειγμα, οι περισσότερες εκδόσεις του Tomcat δομούν μικροϋπηρεσίες χωρίς πακέτα από σελίδες JSP που βρίσκονται στον κατάλογο ανώτατου επιπέδου της εφαρμογής Ιστού. Το πρόβλημα είναι ότι δεν υπάρχει απολύτως κανένα πρότυπο που να μας καθοδηγεί για το πότε συμβαίνει ή δεν συμβαίνει αυτό. Θα ήταν πολύ καλύτερα αν ο κώδικας JSP, όπως αυτός του παραδείγματος, αστοχούσε πάντοτε. Αντίθετα, μερικές φορές λειτουργεί και μερικές όχι, ανάλογα με το διακομιστή ή με τον κατάλογο στον οποίο βρίσκεται η σελίδα JSP. Πολύ μεγάλο μπέρδεμα!

Πάντα να έχετε το νου σας στραμμένο στην ασφάλεια, τη φορητότητα, και τον προσχεδιασμό. Να χρησιμοποιείτε πάντοτε πακέτα!

11.4 Χρήση παραστάσεων JSP

Η παράσταση JSP (JSP expression) χρησιμοποιείται για την εισαγωγή τιμών κατευθείαν στην έξοδο. Έχει την ακόλουθη μορφή:

```
<%= Παράσταση Java %>
```

Η παράσταση υπολογίζεται, μετατρέπεται σε αλφαριθμητικό, και εισάγεται στη σελίδα. Αυτός ο υπολογισμός πραγματοποιείται κατά το χρόνο εκτέλεσης (όταν γίνεται η αίτηση της σελίδας), και έτσι έχει πλήρη πρόσβαση στις πληροφορίες σχετικά με την αίτηση. Για παράδειγμα, το ακόλουθο παράδειγμα παρουσιάζει την ημερομηνία και την ώρα που έγινε η αίτηση της σελίδας:

```
Τρέχουσα ώρα: <%= new java.util.Date() %>
```

Προκαθορισμένες μεταβλητές

Για να απλοποιήσετε αυτές τις παραστάσεις, μπορείτε να χρησιμοποιήσετε μια σειρά από προκαθορισμένες μεταβλητές (ή "έμμεσα αντικείμενα", implicit objects). Δεν υπάρχει τίποτα μαγικό σχετικά με αυτές τις μεταβλητές· απλώς το σύστημα σας λέει ποια ονόματα θα χρησιμοποιήσει για τις τοπικές μεταβλητές στην `_jspService` (τη μέθοδο που αντικαθιστά την `doGet` στις μικροϋπηρεσίες που προκύπτουν από σελίδες JSP). Τα έμμεσα αντικείμενα εξετάζονται λεπτομερέστερα στην Ενότητα 11.12, αλλά για τους σκοπούς των παραστάσεων τα πιο σημαντικά από αυτά είναι τα εξής:

- **request**, το αντικείμενο `HttpServletRequest`.
- **response**, το αντικείμενο `HttpServletResponse`.

- **session**, το αντικείμενο `HttpSession` που σχετίζεται με την αίτηση (εκτός και αν έχει απενεργοποιηθεί με την ιδιότητα `session` της οδηγίας `page` — δείτε την Ενότητα 12.4).
- **out**, το αντικείμενο `Writer` (μια εκδοχή του τύπου `JspWriter` με χώρο προσωρινής αποθήκευσης), που χρησιμοποιείται για την αποστολή εξόδου στον πελάτη
- **application**, το αντικείμενο `ServiceContext`. Πρόκειται για τη δομή δεδομένων που είναι κοινή για όλες τις μικροϋπηρεσίες και τις σελίδες JSP της εφαρμογής Ιστού, και είναι κατάλληλη για την αποθήκευση κοινόχρηστων δεδομένων. Λεπτομερής εξέταση αυτού του αντικειμένου γίνεται στα κεφάλαια για τους κόκκους (Κεφάλαιο 14) και την αρχιτεκτονική MVC (Κεφάλαιο 15).

Ας δούμε ένα παράδειγμα:

```
Ο υπολογιστής σας: <%= request.getRemoteHost() %>
```

Αντιστοιχία σελίδων JSP και μικροϋπηρεσιών

Μόλις τώρα αναφέραμε ότι η παράσταση JSP υπολογίζεται και εισάγεται στην έξοδο της σελίδας. Αν και αυτό είναι σωστό, μερικές φορές είναι χρήσιμο να γνωρίζετε τι συμβαίνει στα παρασκήνια.

Στην πραγματικότητα τα πράγματα είναι αρκετά απλά: οι παραστάσεις JSP κατά βάση μετατρέπονται σε εντολές `print` (ή `write`) στη μικροϋπηρεσία που προκύπτει από τη σελίδα JSP. Ενώ ο κανονικός κώδικας HTML μετατρέπεται σε εντολές `print` με διπλά εισαγωγικά γύρω από το κείμενο, οι παραστάσεις JSP γίνονται εντολές `print` χωρίς εισαγωγικά. Αντί να τοποθετηθούν στη μέθοδο `doGet`, αυτές οι εντολές `print` τοποθετούνται σε μια νέα μέθοδο που ονομάζεται `_jspService`, η οποία καλείται από τη `service` τόσο για τις αιτήσεις `GET` όσο και για τις αιτήσεις `POST`. Για παράδειγμα, η Λίστα 11.1 δείχνει ένα μικρό δείγμα σελίδας JSP που περιλαμβάνει στατική HTML και μια παράσταση JSP. Η Λίστα 11.2 παρουσιάζει μια μέθοδο `_jspService` που είναι δυνατόν να προκύψει από αυτόν τον κώδικα. Φυσικά, τα διάφορα προϊόντα παράγουν κώδικα με ελαφρώς διαφορετικούς τρόπους, και είναι αρκετά συνηθισμένες διάφορες βελτιστοποιήσεις, όπως η ανάγνωση της HTML από ένα στατικό πίνακα byte.

Έχουμε επίσης υπεραπλουστεύσει τον ορισμό της μεταβλητής `out`· η `out` σε μια σελίδα JSP είναι ένα αντικείμενο `JspWriter`, οπότε θα πρέπει να τροποποιήσετε το κάπως απλούστερο αντικείμενο `PrintWriter` το οποίο προκύπτει άμεσα από μια κλήση της `getWriter`. Έτσι, μην περιμένετε ότι ο κώδικας που θα παραγάγει ο διακομιστής σας θα είναι *ακριβώς* ίδιος με αυτόν που ακολουθεί.

Λίστα 11.1 Δείγμα παράστασης JSP: Random Number

```
<H1> A Random Number</H1>
  <%= Math.random() %>
```


Λίστα 11.2 Αντιπροσωπευτικός κώδικας προκύπτουσας μικροϋπηρεσίας: Random Number

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  HttpSession session = request.getSession();
  JspWriter out = response.getWriter();
  out.println("<H1 A Random Number</H1>);
  out.println(Math.random());
  ...
  }
```

Αν θέλετε να δείτε τον ακριβή κώδικα που παράγει ο διακομιστής σας, θα πρέπει να ψάξετε λίγο για να τον βρείτε. Στην πραγματικότητα μερικοί διακομιστές διαγράφουν τα πηγαία αρχεία κώδικα στην περίπτωση που η μεταγλώττιση είναι επιτυχής. Εδώ όμως παραθέτουμε μια σύνοψη των θέσεων που χρησιμοποιούνται από τους τρεις συνηθισμένους δωρεάν διακομιστές ανάπτυξης.

Πηγαίος κώδικας μικροϋπηρεσιών που παράγονται αυτόματα στον Tomcat

κατάλογος_εγκατάστασης/work/Standalone/localhost/_

(Ο τελικός κατάλογος είναι ένας χαρακτήρας υπογράμμισης. Γενικότερα, ο κώδικας βρίσκεται στον κατάλογο *κατάλογος_εγκατάστασης*/work/Standalone/localhost/*ΌνομαΕφαρμογήςΙστού*. Η θέση διαφέρει λίγο μεταξύ των διαφόρων εκδόσεων του Tomcat.)

Πηγαίος κώδικας μικροϋπηρεσιών που παράγονται αυτόματα στο JRun

κατάλογος_εγκατάστασης/servers/default/default-ear/default-war/WEB-INF/jsp

(Γενικότερα, ο κώδικας βρίσκεται στον κατάλογο WEB-INF/jsp της εφαρμογής Ιστού στην οποία ανήκει η σελίδα JSP. Σημειώστε, όμως, ότι το JRun δεν αποθηκεύει τα αρχεία .java, εκτός κι αν αλλάξετε από `false` σε `true` το στοιχείο `keepGenerated` του αρχείου *κατάλογος_εγκατάστασης*/servers/default/SERVER-INF/default-web.xml.)

Πηγαίος κώδικας μικροϋπηρεσιών που παράγονται αυτόματα στο Resin

κατάλογος_εγκατάστασης/WEB-INF/work

(Γενικότερα, ο κώδικας βρίσκεται στον κατάλογο WEB-INF/work της εφαρμογής Ιστού στην οποία ανήκει η σελίδα JSP.)

Σύνταξη XML για παραστάσεις

Αυτοί που χρησιμοποιούν τη γλώσσα XML μπορούν να χρησιμοποιήσουν την ακόλουθη εναλλακτική σύνταξη για τις παραστάσεις JSP:

```
<jsp:expression>Παράσταση Java</jsp:expression>
```

Από την JSP 1.2 και μετά οι διακομιστές είναι υποχρεωμένοι να υποστηρίζουν αυτή τη σύνταξη εφόσον οι συγγραφείς δεν αναμιγνύουν στην ίδια σελίδα τη σύνταξη XML και την καθιερωμένη μορφή JSP (<%= ... %>). Αυτό σημαίνει ότι, για να χρησιμοποιήσετε τη σύνταξη XML, θα πρέπει να τη χρησιμοποιήσετε για *ολόκληρη* τη σελίδα. Στη JSP 1.2 (όχι όμως και στη 2.0) αυτή η απαίτηση σημαίνει ότι θα πρέπει ολόκληρη η σελίδα να περικλείεται σε ένα στοιχείο `jsp:root`. Κατά συνέπεια, οι περισσότεροι προγραμματιστές ακολουθούν την κλασική σύνταξη εκτός από τις περιπτώσεις που είτε παράγουν έγγραφα XML (π.χ., xhtml ή SOAP), ή όταν η ίδια η σελίδα JSP είναι έξοδος κάποιας διαδικασίας XML (π.χ., XSLT).

Προσέξτε ότι τα στοιχεία XML, σε αντίθεση με αυτά της HTML, κάνουν διάκριση πεζών και κεφαλαίων χαρακτήρων. Γι' αυτό φροντίστε να γράφετε το κείμενο `jsp:expression` με πεζούς χαρακτήρες.

11.5 Παράδειγμα: Παραστάσεις JSP

Η Λίστα 11.3 δείχνει ένα παράδειγμα σελίδας JSP με το όνομα Expressions.jsp. Τοποθετήσαμε το αρχείο σε έναν κατάλογο με όνομα jsp-scripting, αντιγράψαμε ολόκληρο τον κατάλογο από τον κατάλογο ανάπτυξης στο ανώτατο επίπεδο της προεπιλεγμένης εφαρμογής Ιστού (γενικά, στον κατάλογο ανώτατου επιπέδου της εφαρμογής Ιστού — ένα επίπεδο επάνω από τον κατάλογο WEB-INF), και χρησιμοποιήσαμε το URL http://*υπολογιστήςΥπηρεσίας*/jsp-scripting/Expressions.jsp. Οι Εικόνες 11-2 και 11-3 δείχνουν μερικά τυπικά αποτελέσματα.

Παρατηρήστε ότι στην ενότητα `HEAD` της σελίδας JSP έχουμε ενσωματώσει ετικέτες `META` και ένα σύνδεσμο φύλλου στυλ. Αποτελεί καλή πρακτική η ενσωμάτωση αυτών των στοιχείων, αλλά υπάρχουν δύο λόγοι για τους οποίους τα στοιχεία αυτά συχνά παραλείπονται από σελίδες που παράγονται με τις κανονικές μικροϋπηρεσίες.

Πρώτον, με τις μικροϋπηρεσίες είναι επίπονη η παραγωγή των απαραίτητων εντολών `println`. Με τις σελίδες JSP, όμως, η μορφή είναι απλούστερη και μπορείτε να κάνετε χρήση των επιλογών επαναχρησιμοποίησης κώδικα που παρέχουν τα εργαλεία κατασκευής κανονικών σελίδων HTML. Αυτή η ευκολία είναι ένας σημαντικός λόγος για τη χρήση σελίδων JSP. Οι σελίδες JSP δεν είναι πιο *ισχυρές* από τις μικροϋπηρεσίες (παρασκηνιακά *είναι* μικροϋπηρεσίες), αλλά μερικές φορές είναι πιο *εύχρηστες* από τις μικροϋπηρεσίες.

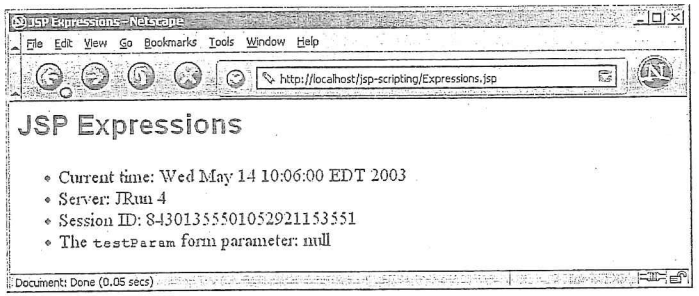
Δεύτερον, οι μικροϋπηρεσίες δεν μπορούν να χρησιμοποιήσουν την απλούστερη μορφή των σχετικών URL (αυτών που αναφέρονται σε αρχεία τα οποία βρίσκονται στον ίδιο κατάλογο με την τρέχουσα σελίδα), αφού οι κατάλογοι των μικροϋπηρεσιών δεν αντιστοιχίζονται σε URL με τον ίδιο τρόπο που αντιστοιχίζονται τα URL των κανονικών ιστοσελίδων. Επιπλέον, οι διακομιστές απαγορεύουν ρητά την άμεση προσπέλαση του περιεχομένου του καταλόγου WEB-INF/classes (ή οπουδήποτε στον κατάλογο WEB-INF) από τους πελάτες. Έτσι είναι αδύνατη η τοποθέτηση, στον ίδιο κατάλογο, φύλλων στυλ και αρχείων κλάσεων μικροϋπηρεσιών, ακόμα και αν χρησιμοποιήσετε τα στοιχεία `servlet` και `servlet-mapping` του αρχείου web.xml (δείτε την Ενότητα 2.12, "Εφαρμογές Ιστού: Μια προεπισκόπηση") για να προσαρμόσετε τα URL των μικροϋπηρεσιών. Από την άλλη πλευρά, οι σελίδες JSP εγκαθίστανται στην κανονική ιεραρχία

ιστοσελίδων του διακομιστή, και τα σχετικά URL μπορούν να αναλυθούν επειδή η σελίδα JSP προσπελάζεται άμεσα από τον πελάτη, και όχι έμμεσα μέσω μιας κλήσης `RequestDispatcher` (δείτε σχετικά το Κεφάλαιο 15, "Ολοκλήρωση μικροϋπηρεσιών και JSP: Η αρχιτεκτονική Model View Controller (MVC)").

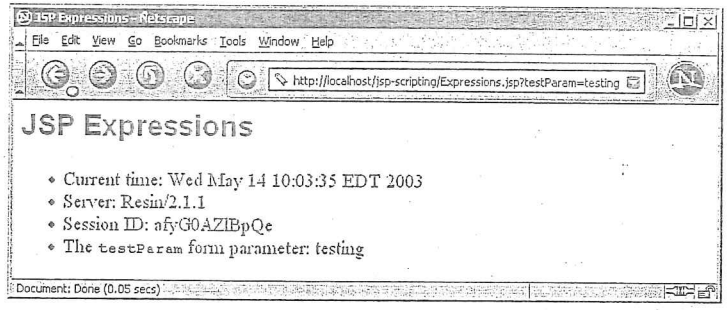
Έτσι, στις περισσότερες περιπτώσεις τα φύλλα στυλ και οι σελίδες JSP μπορούν να βρίσκονται στον ίδιο κατάλογο. Τον πηγαίο κώδικα του φύλλου στυλ, όπως και όλο τον κώδικα που φαίνεται ή αναφέρεται στο βιβλίο, μπορείτε να τον βρείτε στη διεύθυνση http://www.coreservlets.com.

Λίστα 11.3 Expressions.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>JSP Expressions</TITLE>
<META NAME="keywords"
      CONTENT="JSP,expressions,JavaServer Pages,servlets">
<META NAME="description"
      CONTENT="A quick example of JSP expressions.">
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Server: <%= application.getServerInfo() %>
  <LI>Session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
   </BODY></HTML>
```



Εικόνα 11-2 Αποτέλεσμα της σελίδας Expressions.jsp με χρήση του Macromedia JRun και παράλειψη της παραμέτρου αίτησης `testParam`.



Εικόνα 11-3 Αποτέλεσμα της **Expressions.jsp** με χρήση του Caucho Resin και τιμή `testing` για την παράμετρο αίτησης `testParam`.

11.6 Σύγκριση μικροϋπηρεσιών και σελίδων JSP

Στην Ενότητα 4.3 παρουσιάσαμε ένα παράδειγμα μικροϋπηρεσίας που παρέχει στην έξοδο τις τιμές τριών συγκεκριμένων παραμέτρων φόρμας. Ο κώδικας αυτής της μικροϋπηρεσίας επαναλαμβάνεται και εδώ, στη Λίστα 11.4. Η Λίστα 11.5 (Εικόνα 11-4) παρουσιάζει μια εκδοχή που έχει γραφτεί σε μορφή JSP, με χρήση παραστάσεων JSP για την προσπέλαση των παραμέτρων της φόρμας. Η εκδοχή JSP είναι σαφώς ανώτερη: συντομότερη, απλούστερη, και ευκολότερη στη συντήρηση.

Φυσικά με αυτό δεν εννοούμε ότι όλες οι μικροϋπηρεσίες θα μετατρέπονται σε σελίδες JSP με τόσο ξεκάθαρο τρόπο. Η JSP λειτουργεί καλύτερα όταν η *δομή* της σελίδας HTML είναι σταθερή αλλά οι *τιμές* στις διάφορες θέσεις πρέπει να υπολογίζονται δυναμικά. Αν η δομή της σελίδας είναι δυναμική, η JSP είναι λιγότερο ωφέλιμη. Σε μια τέτοια περίπτωση οι μικροϋπηρεσίες είναι καλύτερες. Προφανώς, αν η σελίδα αποτελείται από δυαδικά δεδομένα ή έχει λίγο στατικό περιεχόμενο, οι μικροϋπηρεσίες είναι σαφώς ανώτερες. Επιπλέον, μερικές φορές η απάντηση δεν βρίσκεται ούτε μόνο στις μικροϋπηρεσίες ούτε μόνο στις σελίδες JSP, αλλά σε ένα συνδυασμό και των δύο. Για λεπτομέρειες δείτε το Κεφάλαιο 15("Ολοκλήρωση μικροϋπηρεσιών και JSP: Η αρχιτεκτονική Model View Controller (MVC)").

Λίστα 11.4 ThreeParams.java

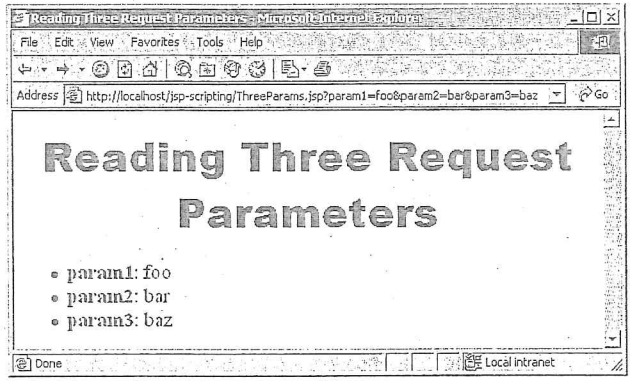
```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
```


Λίστα 11.4 ThreeParams.java *(συνέχεια)*

```
    PrintWriter out = response.getWriter();
    String title = "Reading Three Request Parameters";
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>param1</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>param2</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>param3</B>: "
                + request.getParameter("param3") + "\n" +
                "</UL>\n" +
                "</BODY></HTML>");

  }
}
```

Λίστα 11.5 ThreeParams.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Reading Three Request Parameters</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1>reading three request parameters</H1>
<UL>
  <LI><B>param1</B>: <%= request.getparameter("param1") %>
  <LI><B>param2</B>: <%= request.getparameter("param2") %>
  <LI><B>param3</B>: <%= request.getparameter("param3") %>
</UL>
  </BODY></HTML>
```



Εικόνα 11-4 Αποτέλεσμα της σελίδας **ThreeParams.jsp**.

11.7 Συγγραφή μικροσεναρίων

Αν θέλετε να κάνετε κάτι πιο περίπλοκο από την εμφάνιση της τιμής μιας απλής παράστασης, τα μικροσενάρια (scriplets) JSP σας επιτρέπουν να εισάγετε κώδικα στη μέθοδο `_jspService` της μικροϋπηρεσίας (που καλείται από τη `service`). Τα μικροσενάρια έχουν την ακόλουθη μορφή:

```
<% Κώδικας Java %>
```

Τα μικροσενάρια έχουν πρόσβαση στις ίδιες "αυτόματες" μεταβλητές (`request`, `session`, `out`, κ.λπ.) όπως συμβαίνει και με τις παραστάσεις. Έτσι, για παράδειγμα, αν θέλετε να στείλετε ρητά έξοδο στη σελίδα που θα προκύψει, θα μπορούσατε να χρησιμοποιήσετε τη μεταβλητή `out`, όπως στο παράδειγμα που ακολουθεί.

```
<%
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
%>
```

Στη συγκεκριμένη περίπτωση, θα μπορούσατε να επιτύχετε το ίδιο αποτέλεσμα πιο εύκολα με τη χρήση ενός συνδυασμού μικροσεναρίου και παράστασης JSP, όπως στο ακόλουθο παράδειγμα:

```
<% String queryData = request.getQueryString(); %>
Attached GET data: <%= queryData %>
```

Ή, θα μπορούσατε να χρησιμοποιήσετε μόνο μια παράσταση JSP, όπως εδώ:

```
Attached GET data: <%= request.getQueryString() %>
```


Γενικά όμως, τα μικροσενάρια μπορούν να εκτελέσουν μια σειρά από εργασίες που δεν μπορούν να υλοποιηθούν μόνο με παραστάσεις. Σε αυτές τις εργασίες περιλαμβάνονται οι κεφαλίδες απάντησης και οι κωδικοί κατάστασης, η εκτέλεση παράπλευρων ενεργειών (όπως η καταχώριση στο ημερολόγιο του διακομιστή ή η ενημέρωση μιας βάσης δεδομένων), ή η εκτέλεση κώδικα που περιέχει βρόχους, συνθήκες, ή άλλες πολύπλοκες δομές. Για παράδειγμα, το τμήμα κώδικα που ακολουθεί, καθορίζει ότι η τρέχουσα σελίδα θα αποσταλεί στον πελάτη ως έγγραφο Microsoft Word και όχι ως HTML (που είναι η προεπιλογή). Επειδή το Microsoft Word μπορεί να εισαγάγει έγγραφα HTML, αυτή η τεχνική είναι αρκετά χρήσιμη σε πραγματικές εφαρμογές.

```
<% response.setContentType("application/msword"); %>
```

Είναι σημαντικό να σημειώσετε ότι δεν χρειάζεται να ορίσετε τις κεφαλίδες απάντησης ή τους κωδικούς κατάστασης στην *αρχή* της σελίδας JSP, παρά το ότι αυτή η δυνατότητα φαίνεται να παραβιάζει τον κανόνα που ορίζει ότι τα δεδομένα απάντησης θα πρέπει να καθορίζονται πριν από την αποστολή οποιουδήποτε περιεχομένου στον πελάτη. Είναι έγκυρο να ορίσετε κεφαλίδες και κωδικούς κατάστασης μετά από μια μικρή ποσότητα περιεχομένου εγγράφου, επειδή οι μικροϋπηρεσίες που προκύπτουν από σελίδες JSP χρησιμοποιούν μια ειδική παραλλαγή του `Writer` (τύπου `JspWriter`) που αποθηκεύει προσωρινά το έγγραφο. Όμως αυτή η συμπεριφορά της προσωρινής αποθήκευσης μπορεί να αλλάξει· δείτε σχετικά το Κεφάλαιο 12 (Έλεγχος της δομής των μικροϋπηρεσιών: Η οδηγία page) για μια περιγραφή των ιδιοτήτων `buffer` και `autoflush` της οδηγίας `page`.

Αντιστοιχία JSP και μικροϋπηρεσιών

Είναι εύκολο να καταλάβετε ποια είναι η αντιστοιχία μεταξύ των μικροσεναρίων JSP και του κώδικα των μικροϋπηρεσιών: ο κώδικας των μικροσεναρίων εισάγεται άμεσα στη μέθοδο `_jspService`: ούτε αλφαριθμητικά, ούτε εντολές `print`, ούτε άλλες αλλαγές. Για παράδειγμα, η Λίστα 11.6 παρουσιάζει ένα μικρό δείγμα σελίδας JSP που περιλαμβάνει στατικό κώδικα HTML, μια παράσταση JSP, και ένα μικροσενάριο JSP. Η Λίστα 11.7 παρουσιάζει μια μέθοδο `_jspService` που μπορεί να προκύψει για αυτή τη σελίδα. Προσέξτε ότι η κλήση της `bar` (η παράσταση JSP) δεν συνοδεύεται από ελληνικό ερωτηματικό, ενώ η κλήση της `baz` (το μικροσενάριο JSP) συνοδεύεται από ελληνικό ερωτηματικό. Να θυμάστε ότι οι παραστάσεις Java περιέχουν *τιμές* Java (που δεν τελειώνουν σε ελληνικό ερωτηματικό), ενώ τα περισσότερα μικροσενάρια JSP περιέχουν *εντολές* Java (που τελειώνουν σε ελληνικό ερωτηματικό). Για να θυμάστε πιο εύκολα πότε πρέπει να χρησιμοποιείτε ελληνικό ερωτηματικό, θα πρέπει απλώς να θυμάστε ότι οι παραστάσεις τοποθετούνται στο εσωτερικό εντολών `print` ή `write`, και φυσικά η εντολή `out.print(blah;);` είναι εμφανές λάθος.

Και πάλι τα προϊόντα άλλων κατασκευαστών θα παραγάγουν κώδικα με λίγο διαφορετικούς τρόπους, και επίσης υπεραπλουστεύσαμε τη μεταβλητή `out` (που είναι τύπου `JspWriter`, και όχι το κάπως απλούστερο `PrintWriter` που προκύπτει από μια κλήση `getWriter`). Έτσι δεν θα πρέπει να περιμένετε ότι ο κώδικας που θα δημιουργήσει ο διακομιστής σας θα είναι *ακριβώς* ίδιος.

Λίστα 11.6 Δείγμα παράστασης/μικροσεναρίου JSP

```
<H2>foo</H2>
<%= bar() %>
   <%= baz(); %>
```

Λίστα 11.7 Αντιπροσωπευτικός κώδικας μικροϋπηρεσίας που προκύπτει: Παράσταση/Μικροσενάριο

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  HttpSession session = request.getSession();
  JspWriter out = response.getWriter();
  out.println("<H2>foo</H2>");
  out.println(bar());
  baz();
  ...
   }
```

Σύνταξη XML για μικροσενάρια

Το ισοδύναμο σε XML για την εντολή <% *Κώδικας Java* %> είναι

```
<jsp:scriplet>Κώδικας Java</jsp:scriplet>
```

Από την JSP 1.2 και μετά οι διακομιστές είναι υποχρεωμένοι να υποστηρίζουν αυτή τη σύνταξη, εφόσον οι συγγραφείς των σελίδων δεν αναμιγνύουν την εκδοχή XML (<jsp:scriplet>...</jsp:scriplet>) και την εκδοχή τύπου ASP (<% ... %>) στην ίδια σελίδα· αν χρησιμοποιήσετε την εκδοχή XML, θα πρέπει να τη χρησιμοποιήσετε με συνέπεια για ολόκληρη τη σελίδα. Να θυμάστε ότι στα στοιχεία XML γίνεται διάκριση πεζών και κεφαλαίων γραμμάτων· φροντίστε το `jsp:scriplet` να είναι γραμμένο με πεζούς χαρακτήρες.

11.8 Παράδειγμα μικροσεναρίου

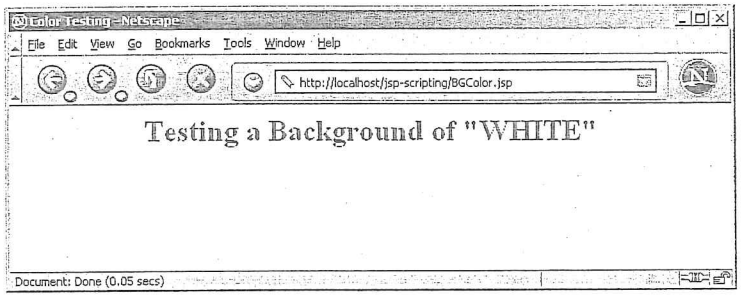
Ως παράδειγμα κώδικα που είναι πολύ περίπλοκος για μια παράσταση JSP, η Λίστα 11.8 παρουσιάζει μια σελίδα JSP που χρησιμοποιεί την παράμετρο αίτησης `bgcolor` για να ορίσει το χρώμα παρασκηνίου της σελίδας. Η απλή χρήση της εντολής

```
<BODY BGCOLOR=<% request.getParameter("bgColor") %>">
```

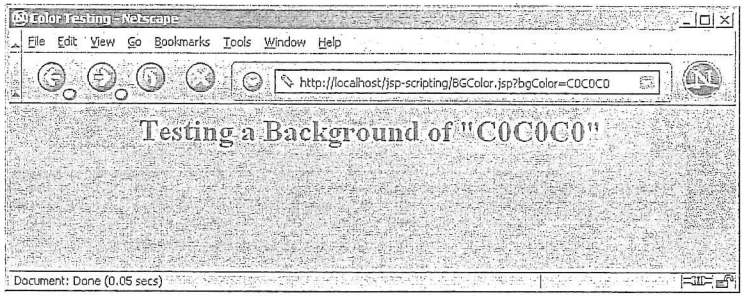

θα παραβίαζε το βασικό κανόνα της ανάγνωσης δεδομένων: να γίνεται πάντοτε έλεγχος για ελλιπή δεδομένα ή δεδομένα με λανθασμένη μορφή. Γι' αυτό θα χρησιμοποιήσουμε ένα μικροσενάριο. Παραλείψαμε το φύλλο στυλ JSP-Styles.css έτσι ώστε να μην υποσκελίσει το χρώμα παρασκηνίου. Οι Εικόνες 11-5, 11-6, και 11.7 παρουσιάζουν αντίστοιχα το προεπιλεγμένο αποτέλεσμα, το αποτέλεσμα για χρώμα φόντου `C0C0C0`, και το αποτέλεσμα για το φόντο `papayawhip` (ένα από τα περίεργα ονόματα χρωμάτων των X11, που εξακολουθεί να υποστηρίζεται από πολλούς μεταγλωττιστές για ιστορικούς λόγους).

Λίστα 11.8 BGColor.jsp

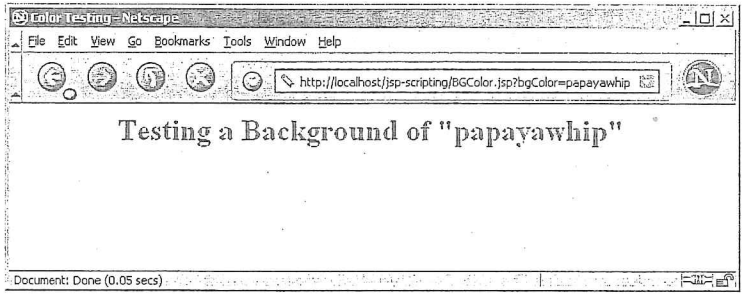
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor = request.getParameter("bgColor");
if ((bgColor == null) || (bgColor.trim().equals(""))) {
  bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>"
<H2 ALIGN="CENTER">Testing a Background of "<%= bgColor %>"</H2>
   </BODY></HTML>
```



Εικόνα 11-5 Προεπιλεγμένο αποτέλεσμα της σελίδας BGColor.jsp.



Εικόνα 11-6 Αποτέλεσμα της σελίδας BGColor.jsp όταν προσπελάζεται με την παράμετρο `bgColor` να έχει την τιμή RGB `C0C0C0`.



Εικόνα 11-7 Αποτέλεσμα της BGColor.jsp όταν προσπελάζεται με την παράμετρο `bgColor` να έχει ως τιμή το όνομα χρώματος `papayawhip` των X11.

11.9 Μετατροπή τμημάτων σελίδων JSP σε παραστάσεις υπό συνθήκη

Μια άλλη χρήση των μικροσεναρίων είναι η υπό συνθήκη έξοδος HTML ή άλλου περιεχομένου που *δεν* βρίσκεται σε ετικέτα JSP. Το κλειδί σε αυτή την προσέγγιση είναι τα εξής γεγονότα: (α) ο κώδικας στο εσωτερικό ενός μικροσεναρίου εισάγεται στη μέθοδο `_jspService` (που καλείται από τη `service`) της μικροϋπηρεσίας που προκύπτει, και (β) η στατική HTML (το κείμενο προτύπου) πριν ή μετά το μικροσενάριο μετατρέπεται σε εντολές `print`. Αυτή η συμπεριφορά σημαίνει ότι τα μικροσενάρια δεν χρειάζεται να περιέχουν ολοκληρωμένες εντολές Java, και τα μπλοκ κώδικα που παραμένουν ημιτελή μπορούν να επηρεάσουν τη στατική HTML ή τη σελίδα JSP εκτός του μικροσεναρίου. Για παράδειγμα, δείτε το παρακάτω τμήμα JSP της Λίστα 11.9, που περιέχει κείμενο προτύπου και μικροσενάρια.

Λίστα 11.9 DayWish.jsp

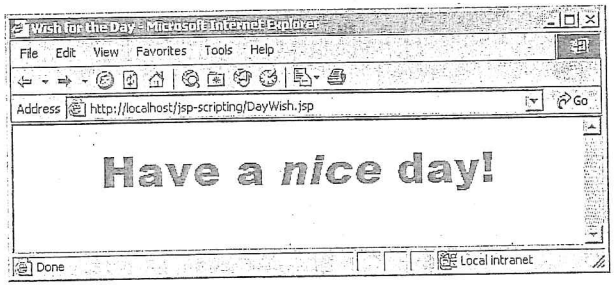
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Wish for the Day</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<% if (Math.random() < 0.5) { %>
<H1>Have a <I>nice</I> day!</H1>
<% } else { %>
<H1>Have a <I>lousy</I> day!</H1>
<% } %>
   </BODY></HTML>
```

Πιθανόν να θεωρείτε το έντονο τμήμα κάπως μπερδεμένο. Εμάς μας μπέρδεψε τις πρώτες φορές που είδαμε δομές αυτής της μορφής. Ούτε η γραμμή "have a nice day" ούτε η γραμμή "have a lousy day" περιέχονται σε ετικέτα JSP, οπότε φαίνεται παράξενο που μόνο μία από τις δύο γίνεται τμήμα της εξόδου για οποιαδήποτε αίτηση. Δείτε τις Εικόνες 11-8 και 11-9.

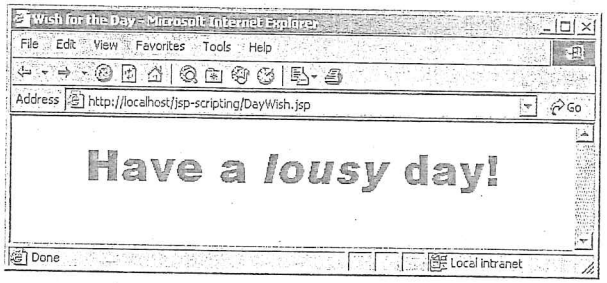
Μην πανικοβάλλεστε! Ακολουθήστε απλώς τους κανόνες που ορίζουν πώς γίνεται η μετατροπή του κώδικα JSP σε κώδικα μικροϋπηρεσίας. Αφού σκεφθείτε πώς θα μετατραπεί το συγκεκριμένο παράδειγμα, από τη μηχανή JSP, σε κώδικα μικροϋπηρεσίας, θα πάρετε το ακόλουθο αποτέλεσμα, που γίνεται εύκολα κατανοητό.

```
if (Math.random() < 0.5) {
   out.println("<H1>Have a <I>nice</I> day!</H1>");
} else {
   out.println("<H1>Have a <I>lousy</I> day!</H1>");
}
```

Το κλειδί είναι ότι τα δύο πρώτα μικροσενάρια δεν περιέχουν ολοκληρωμένες εντολές, αλλά μάλλον μερικές εντολές που έχουν "αιωρούμενες" αγκύλες. Αυτό επιτρέπει τη "σύλληψη" του επακόλουθου κώδικα HTML μέσα από τις προτάσεις `if` ή `else`.



Εικόνα 11-8 Ένα πιθανό αποτέλεσμα της σελίδας DayWish.jsp.



Εικόνα 11-9 Ένα άλλο πιθανό αποτέλεσμα της σελίδας DayWish.jsp.

Η κατάχρηση αυτής της προσέγγισης μπορεί να οδηγήσει σε κώδικα JSP που είναι δυσνόητος και δύσκολος στη συντήρηση. Καλό είναι να αποφύγετε τη μετατροπή μεγάλων ενοτήτων HTML σε παραστάσεις υπό συνθήκη, και να προσπαθήσετε να κρατήσετε όσο το δυνατόν περισσότερο τις σελίδες JSP εστιασμένες σε εργασίες παρουσίασης (έξοδο HTML). Παρόλα αυτά, υπάρχουν περιστάσεις στις οποίες οι εναλλακτικές λύσεις δεν είναι ελκυστικές. Βασικό παράδειγμα είναι η παραγωγή καταλόγων ή πινάκων που περιέχουν απροσδιόριστο αριθμό καταχωρίσεων. Αυτό συμβαίνει αρκετά συχνά όταν παρουσιάζετε δεδομένα που είναι αποτέλεσμα ενός ερωτήματος σε κάποια βάση δεδομένων. Για πληροφορίες σχετικά με την προσπέλαση βάσεων δεδομένων από κώδικα Java δείτε το Κεφάλαιο 17 (Προσπέλαση βάσεων δεδομένων με την JDBC). Επίσης, ακόμα και αν *εσείς* δε χρησιμοποιείτε αυτή την προσέγγιση, είναι αναπόφευκτο να δείτε παραδείγματα αυτής της προσέγγισης στα έργα σας, και θα πρέπει να καταλαβαίνετε πώς και γιατί λειτουργεί με τον τρόπο που λειτουργεί.

11.10 Χρήση δηλώσεων

Η δήλωση (declaration) JSP σας επιτρέπει να ορίσετε μεθόδους ή πεδία που εισάγονται στο κυρίως σώμα της κλάσης της μικροϋπηρεσίας (*έξω* από τη μέθοδο `_jspService`, που καλείται από τη `service` για την επεξεργασία της αίτησης). Η δήλωση έχει την ακόλουθη μορφή:

```
<% Ορισμός μεθόδου ή πεδίου %>
```

Επειδή οι δηλώσεις δεν παράγουν έξοδο, κανονικά χρησιμοποιούνται σε συνδυασμό με παραστάσεις ή μικροσενάρια JSP. Θεωρητικά, οι δηλώσεις JSP μπορούν να περιέχουν ορισμούς πεδίων (μεταβλητές στιγμιοτύπων), ορισμούς μεθόδων, ορισμούς εσωτερικών κλάσεων, ή και στατικά μπλοκ απόδοσης αρχικών τιμών: οτιδήποτε είναι έγκυρο στο εσωτερικό του ορισμού μιας κλάσης, αλλά έξω από τις υπάρχουσες μεθόδους. Στην πράξη, όμως, οι δηλώσεις σχεδόν πάντοτε περιέχουν ορισμούς πεδίων ή μεθόδων.

Θα πρέπει όμως να σας επιστήσουμε την προσοχή σε ένα πράγμα: μη χρησιμοποιείτε δηλώσεις JSP για να υποσκελίσετε (override) τις τυπικές μεθόδους (`service`, `doGet`, `init`, κ.λπ) του κύκλου ζωής της μικροϋπηρεσίας. Η μικροϋπηρεσία στην οποία μεταφράζεται η σελίδα JSP κάνει ήδη χρήση αυτών των μεθόδων. Δεν υπάρχει ανάγκη να αποκτήσουν οι δηλώσεις πρόσβαση στις `service`, `doGet`, ή `doPost`, αφού οι κλήσεις της `service` μεταβιβάζονται αυτόματα στη `_jspService`, που είναι η θέση όπου τοποθετείται ο κώδικας ο οποίος προκύπτει από τις παραστάσεις και τα μικροσενάρια. Ωστόσο, για το σκοπό της απόδοσης αρχικών τιμών και καθαρισμού τιμών, μπορείτε να χρησιμοποιήσετε τις `jspInit` και `jspDestroy` — οι τυπικές μέθοδοι `init` και `destroy` είναι σίγουρο ότι θα καλέσουν αυτές τις δύο μεθόδους των μικροϋπηρεσιών που προκύπτουν από σελίδες JSP.

Η προσέγγιση του βιβλίου

Για την απόδοση αρχικών τιμών και για καθαρισμό τιμών στις σελίδες JSP, χρησιμοποιήστε δηλώσεις JSP που υποσκελίζουν τις `jspInit` και `jspDestroy`, και όχι τις `init` και `destroy`.

Εκτός από την υποσκέλιση τυπικών μεθόδων, όπως οι `jspInit` και `jspDestroy`, η χρησιμότητα των δηλώσεων JSP για τον ορισμό μεθόδων είναι σε κάποιο βαθμό αμφισβητούμενη. Η μεταφορά των μεθόδων σε ξεχωριστές κλάσεις (πιθανόν με μορφή στατικών μεθόδων) κάνει ευκολότερη τη συγγραφή τους (αφού χρησιμοποιείτε ένα περιβάλλον Java, και όχι κάποιο περιβάλλον HTML), ευκολότερο τον έλεγχο (δεν χρειάζεται να εκτελέσετε το διακομιστή), ευκολότερη την αποσφαλμάτωση (οι προειδοποιήσεις μεταγλώττισης σας δίνουν τον αριθμό της γραμμής — δεν χρειάζεστε "κόλπα" για να δείτε την τυπική έξοδο), και ευκολότερη την επαναχρησιμοποίηση (πολλές σελίδες JSP μπορούν να χρησιμοποιήσουν την ίδια βοηθητική κλάση). Ωστόσο, η χρήση δηλώσεων JSP για τον ορισμό μεταβλητών στιγμιοτύπων (πεδίων), όπως θα δούμε σε λίγο, σας προσφέρει κάτι που δεν αναπαράγεται εύκολα με τις ξεχωριστές βοηθητικές κλάσεις: μια θέση για την αποθήκευση δεδομένων που διατηρούνται μεταξύ των αιτήσεων.

Η προσέγγιση του βιβλίου

Ορίστε τις περισσότερες μεθόδους με ξεχωριστές κλάσεις Java, και όχι με δηλώσεις JSP.

Αντιστοιχία JSP και μικροϋπηρεσιών

Οι δηλώσεις JSP δημιουργούν κώδικα που τοποθετείται στο εσωτερικό του ορισμού της κλάσης της μικροϋπηρεσίας, αλλά έξω από τη μέθοδο `_jspService`. Επειδή τα πεδία και οι μέθοδοι μπορούν να δηλωθούν με οποιαδήποτε σειρά, δεν παίζει ρόλο αν ο κώδικας από τις δηλώσεις βρίσκεται στην αρχή ή στο τέλος της μικροϋπηρεσίας. Για παράδειγμα, η Λίστα 11.10 παρουσιάζει ένα μικρό παράδειγμα σελίδας JSP που περιλαμβάνει στατικό κώδικα HTML, μία δήλωση JSP, και μία παράσταση JSP. Η Λίστα 11.11 παρουσιάζει τη μικροϋπηρεσία που μπορεί να προκύψει από αυτή τη σελίδα. Πρέπει να σημειώσουμε ότι το συγκεκριμένο όνομα της μικροϋπηρεσίας που προκύπτει δεν καθορίζεται από τις προδιαγραφές JSP, και στην πράξη οι διάφοροι διακομιστές χρησιμοποιούν διαφορετικές συμβάσεις. Επίσης, όπως έχει ήδη αναφερθεί, τα διάφορα

προϊόντα θα παραγάγουν αυτόν τον κώδικα με λίγο διαφορετικούς τρόπους, και υπεραπλουστεύσαμε τη μεταβλητή out (που είναι τύπου JspWriter, και όχι του ελαφρώς απλούστερου τύπου PrintWriter που προκύπτει από μια κλήση της getWriter). Τέλος, η μικροϋπηρεσία δεν θα υλοποιήσει ποτέ άμεσα την HttpJspPage, αλλά μάλλον θα επεκτείνει κάποια ειδική κλάση του προϊόντος, που ήδη υλοποιεί την HttpJspPage. Γι' αυτό μην περιμένετε ότι ο κώδικας που θα παραγάγει ο διακομιστής σας θα είναι *ακριβώς* ίδιος με αυτόν που βλέπετε εδώ.

Λίστα 11.10 Δείγμα δήλωσης JSP

```
<H1>Some Heading</H1>
<%!
  private String randomHeading() {
    return ("<H2>" + Math.random() + "</H2>");
  }
%>
   <%= randomHeading() %>
```

Λίστα 11.11 Αντιπροσωπευτικός κώδικας της μικροϋπηρεσίας που προκύπτει: Δήλωση

```
public class xxxx implements HttpJspPage {
 private String randomHeading {
   return("<H2>" + Math.random() + "</H2>");
}
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
   throws ServletException, IOException {
  response.setContentType('text/html");
  HttpSession session = request.getSession();
  JspWriter out = response.getWriter();
  out.println("<H1>Some Heading</H1>");
  out.println(randomHeading());
  ...
  }
...
   }
```

Σύνταξη XML για δηλώσεις

Η ισοδύναμη σύνταξη σε γλώσσα XML για το <%! *Ορισμός πεδίου ή μεθόδου* %> είναι η εξής

```
<jsp:declaration>Ορισμός πεδίου ή μεθόδου</jsp:declaration>
```

Από την JSP 1.2 και μετά οι διακομιστές είναι υποχρεωμένοι να υποστηρίζουν αυτή τη σύνταξη, εφόσον οι δημιουργοί σελίδων δεν αναμιγνύουν τη σύνταξη XML (<jsp:declaration> ...</jsp:declaration>) και τη σύνταξη τύπου ASP (<% ... %>) στην ίδια σελίδα. Αν πρόκειται να χρησιμοποιήσετε τη μορφή XML, τότε ολόκληρη η σελίδα θα πρέπει να ακολουθεί τη σύνταξη XML· γι' αυτό οι περισσότεροι προγραμματιστές ακολουθούν την κλασική σύνταξη, εκτός από την περίπτωση που για κάποιο λόγο χρησιμοποιούν XML. Να θυμάστε ότι στα στοιχεία XML γίνεται διάκριση πεζών και κεφαλαίων χαρακτήρων· φροντίστε η παράσταση jsp:declaration να είναι γραμμένη με πεζούς χαρακτήρες.

11.11 Παράδειγμα δήλωσης

Σε αυτό το παράδειγμα, το ακόλουθο απόσπασμα JSP εμφανίζει στην οθόνη πόσες φορές έχει γίνει αίτηση της σελίδας από τη στιγμή της εκκίνησης του διακομιστή (ή από την αλλαγή και την επαναφόρτωση της κλάσης της μικροϋπηρεσίας). Ένας μετρητής επισκέψεων σε δύο γραμμές κώδικα!

```
<%! private int accessCount = 0; %>:
Access to page since server reboot:
<%= ++accessCount %>
```

Θυμηθείτε ότι οι διάφορες αιτήσεις ενός πελάτη για την ίδια μικροϋπηρεσία δημιουργούν διαφορετικά νήματα που καλούν τη μέθοδο service ενός μόνο στιγμιότυπου της μικροϋπηρεσίας. *Δεν* οδηγούν στη δημιουργία πολλαπλών στιγμιοτύπων της μικροϋπηρεσίας, εκτός πιθανόν από την περίπτωση που η μικροϋπηρεσία υλοποιεί τη διασύνδεση SingleThreadModel, η οποία τώρα έχει καταργηθεί (δείτε την Ενότητα 3.7). Έτσι οι μεταβλητές στιγμιοτύπου (τα πεδία) μιας κανονικής μικροϋπηρεσίας είναι κοινά στις διάφορες αιτήσεις, και η accessCount δεν χρειάζεται να δηλωθεί ως static. Βέβαια, οι προχωρημένοι αναγνώστες μπορεί να αναρωτηθούν αν αυτός ο κώδικας είναι ασφαλής ως προς τη χρήση σε νήματα· εγγυάται κανείς ο κώδικας ότι ο κάθε χρήστης θα έχει μια μοναδική μέτρηση; Η απάντηση είναι όχι· σε ασυνήθιστες καταστάσεις πολλοί χρήστες θα μπορούσαν, θεωρητικά, να δουν την ίδια τιμή. Για τον αριθμό των επισκέψεων, στο βαθμό που η μέτρηση είναι σωστή μακροπρόθεσμα, δεν πειράζει αν περιστασιακά δύο διαφορετικοί χρήστες δουν την ίδια τιμή. Για τιμές, όμως, όπως τα αναγνωριστικά συνεδριών, είναι κρίσιμο αυτά τα αναγνωριστικά να έχουν μοναδικές τιμές. Αν ενδιαφέρεστε για ένα παράδειγμα που είναι παρόμοιο με το προηγούμενο απόσπασμα αλλά χρησιμοποιεί μπλοκ κώδικα synchronized για να εγγυηθεί την ασφάλεια των νημάτων, δείτε την περιγραφή για την ιδιότητα isThreadSafe της οδηγίας page στο Κεφάλαιο 12.

Η Λίστα 11.12 παρουσιάζει την πλήρη σελίδα JSP· η Εικόνα 11-10 δείχνει ένα αντιπροσωπευτικό αποτέλεσμα. Βέβαια, πριν σπεύσετε να χρησιμοποιήσετε αυτή την προσέγγιση για την παρακολούθηση της επισκέψεων σε όλες τις σελίδες σας, θα πρέπει να σας επιστήσουμε την προσοχή σε ορισμένα σημεία.

Πρώτον, δεν θα μπορούσατε να τη χρησιμοποιήσετε για έναν πραγματικό μετρητή επισκέψεων, αφού η μέτρηση ξεκινά από την αρχή κάθε φορά που επανεκκινείτε το διακομιστή. Έτσι, για έναν πραγματικό μετρητή θα έπρεπε να χρησιμοποιήσετε τις jspInit και jspDestroy για να διαβάσετε την προηγούμενη μέτρηση κατά την εκκίνηση, και να την αποθηκεύσετε κατά τον τερματισμό του διακομιστή.

Δεύτερον, ακόμα και αν χρησιμοποιήσετε την `jspDestroy`, και πάλι μπορεί να συμβεί κάποια κατάρρευση του διακομιστή (π.χ., λόγω διακοπής ρεύματος), οπότε περιοδικά θα έπρεπε να γράφετε τη μέτρηση στο δίσκο.

Τέλος, μερικοί προχωρημένοι διακομιστές υποστηρίζουν κατανεμημένες εφαρμογές όπου μια συστοιχία διακομιστών εμφανίζεται στον πελάτη σαν ένας διακομιστής. Αν οι μικροϋπηρεσίες σας ή οι σελίδες JSP πρέπει να υποστηρίζουν αυτού του είδους την κατανεμημένη λειτουργία, φροντίστε να το λάβετε υπόψη από πριν και αποφύγετε τη χρήση πεδίων για διατηρούμενα δεδομένα. Αντίθετα, χρησιμοποιήσετε μια βάση δεδομένων. (Σημειώστε ότι τα αντικείμενα συνεδριών διαμοιράζονται αυτόματα μεταξύ των κατανεμημένων εφαρμογών, εφόσον οι τιμές είναι `serializable`. Όμως οι τιμές συνεδριών είναι ειδικές για κάθε χρήστη, ενώ εμείς σε αυτή την περίπτωση χρειαζόμαστε δεδομένα ανεξάρτητα από τον πελάτη.)

Λίστα 11.12 AccessCount.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>JSP Declarations</title>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css"></head>
<BODY>
<H1>JSP Declarations</H1>
<%! private int accessCount = 0; %>
<H2>Accesses to page since server reboot:
<%= ++accessCount %></H2>
   </BODY></HTML>
```

Εικόνα 11-10 Επίσκεψη της σελίδας AccessCounts.jsp μετά από εννιά προηγούμενες επισκέψεις από τον ίδιο ή από διαφορετικούς πελάτες.

11.12 Χρήση προκαθορισμένων μεταβλητών

Κατά τη συγγραφή μιας μεθόδου doGet για κάποια μικροϋπηρεσία, το πιο πιθανό είναι ότι θα γράφατε κάτι σαν το ακόλουθο:

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  HttpSession session = request.getSession();
  PrintWrtiter out = response.getWriter();
  ...
}
```

Η API μικροϋπηρεσιών σάς είπε τους τύπους των ορισμάτων για την doGet, τις μεθόδους που θα καλέσετε για να πάρετε τα αντικείμενα συνεδρίας και εγγραφής, και τους τύπους τους. Η JSP αλλάζει το όνομα της μεθόδου από doGet σε `_jspService` και χρησιμοποιεί ένα αντικείμενο `JspWriter`, αντί του `PrintWriter`. Όμως η ιδέα είναι η ίδια. Το ερώτημα είναι, ποιος σας είπε ποια ονόματα μεταβλητών να χρησιμοποιήσετε; Η απάντηση είναι "κανείς". Επιλέξατε όποια ονόματα θέλατε.

Για να είναι χρήσιμες οι παραστάσεις και τα μικροσενάρια JSP, θα πρέπει να γνωρίζετε ποια ονόματα μεταβλητών χρησιμοποιεί η μικροϋπηρεσία που δημιουργήθηκε αυτόματα. Αυτό σας το λένε οι προδιαγραφές. Στη μέθοδο `_jspService` έχετε στη διάθεσή σας οκτώ τοπικές μεταβλητές, που ορίζονται αυτόματα και οι οποίες μερικές φορές ονομάζονται "έμμεσα αντικείμενα" (implicit objects). Αυτές οι μεταβλητές δεν έχουν τίποτα το εξαιρετικό· είναι απλώς τα ονόματα τοπικών μεταβλητών. *Τοπικών* μεταβλητών. Όχι σταθερών. Όχι δεσμευμένων λέξεων της JSP. Τίποτα μαγικό. Έτσι, αν γράφετε κώδικα που δεν αποτελεί τμήμα της μεθόδου `_jspService`, αυτές οι μεταβλητές δεν θα είναι διαθέσιμες. Συγκεκριμένα, επειδή οι δηλώσεις JSP δημιουργούν κώδικα που εμφανίζεται έξω από την `_jspService`, αυτές οι μεταβλητές δεν είναι προσπελάσιμες στις δηλώσεις. Παρομοίως, δεν είναι διαθέσιμες στις βοηθητικές κλάσεις που καλούνται από σελίδες JSP. Αν κάποια ξεχωριστή μέθοδος πρέπει να έχει πρόσβαση σε μια από αυτές τις μεταβλητές, κάντε αυτό που κάνετε πάντα στα προγράμματα Java: μεταβιβάστε τη μεταβλητή.

Οι διαθέσιμες μεταβλητές είναι οι: `request`, `response`, `out`, `session`, `application`, `config`, `pageContext`, και `page`. Λεπτομέρειες για κάθε μία από αυτές δίνονται στη συνέχεια. Υπάρχει και μία ακόμα διαθέσιμη μεταβλητή, η `exception`, αλλά αφορά μόνο τις σελίδες σφαλμάτων. Αυτή η μεταβλητή περιγράφεται στο Κεφάλαιο 12 (Έλεγχος της δομής των μικροϋπηρεσιών: Η οδηγία page), στις ενότητες για τις ιδιότητες `errorPage` και `isErrorPage`.

- **`request`**
  Αυτή η μεταβλητή είναι το αντικείμενο `HttpServletRequest` που σχετίζεται με την αίτηση· σας παρέχει πρόσβαση στις παραμέτρους αίτησης, τον τύπο της αίτησης (π.χ., `GET` ή `POST`), και τις εισερχόμενες κεφαλίδες HTTP (π.χ., τα μπισκότα).

- `response`
  Αυτή η μεταβλητή είναι το αντικείμενο `HttpServletResponse` που σχετίζεται με την απάντηση στον πελάτη. Επειδή το ρεύμα εξόδου (δείτε σχετικά με τη μεταβλητή `out`) κανονικά αποθηκεύεται προσωρινά, είναι συνήθως έγκυρος ο ορισμός κωδικών κατάστασης HTTP και κεφαλίδων απάντησης στο σώμα των σελίδων JSP, παρά το ότι στις μικροϋπηρεσίες δεν επιτρέπεται ο ορισμός των κωδικών κατάστασης και των κεφαλίδων απάντησης όταν έχει σταλεί κάποια έξοδος στον πελάτη. Αν όμως απενεργοποιήσετε την προσωρινή αποθήκευση (δείτε την ιδιότητα `buffer`, στο Κεφάλαιο 12), θα πρέπει να ορίσετε τους κωδικούς κατάστασης και τις κεφαλίδες πριν από την αποστολή οποιουδήποτε περιεχομένου.
- `out`
  Αυτή η μεταβλητή είναι το αντικείμενο `Writer` που χρησιμοποιείται για την αποστολή εξόδου στον πελάτη. Ωστόσο, για να διευκολυνθεί ο ορισμός κεφαλίδων απάντησης σε διάφορες θέσεις στη σελίδα JSP, η μεταβλητή `out` δεν είναι το καθιερωμένο αντικείμενο `PrintWriter`, αλλά μια εκδοχή του αντικειμένου `Writer` με δυνατότητα προσωρινής αποθήκευσης, η οποία ονομάζεται `JspWriter`. Μπορείτε να προσαρμόσετε το μέγεθος του χώρου προσωρινής αποθήκευσης μέσω της ιδιότητας `buffer` της οδηγίας `page` (δείτε το Κεφάλαιο 12). Η μεταβλητή `out` χρησιμοποιείται σχεδόν αποκλειστικά σε μικροσενάρια, αφού οι παραστάσεις JSP τοποθετούνται αυτόματα στο ρεύμα εξόδου και έτσι σπανίως υπάρχει ανάγκη ρητής αναφοράς στην `out`.
- `session`
  Αυτή η μεταβλητή είναι το αντικείμενο `HttpSession` που σχετίζεται με την αίτηση. Θυμηθείτε ότι οι συνεδρίες δημιουργούνται αυτόματα στην τεχνολογία JSP, έτσι αυτή η μεταβλητή είναι δεσμευμένη ακόμα και αν δεν υπάρχει εισερχόμενη αναφορά συνεδρίας. Η μόνη εξαίρεση είναι η χρήση της ιδιότητας `session` της οδηγίας `page` (Κεφάλαιο 12) για την απενεργοποίηση της αυτόματης παρακολούθησης συνεδριών. Σε αυτή την περίπτωση οι προσπάθειες αναφοράς στη μεταβλητή `session` προκαλούν σφάλματα κατά το χρόνο μετάφρασης της σελίδας JSP σε μικροϋπηρεσία. Για γενικές πληροφορίες σχετικά με την παρακολούθηση συνεδριών και την κλάση `HttpSession` δείτε το Κεφάλαιο 9.
- `application`
  Αυτή η μεταβλητή είναι το αντικείμενο `ServletContext` που λαμβάνεται από την `getServletContext`. Οι μικροϋπηρεσίες και οι σελίδες JSP μπορούν να αποθηκεύουν διατηρούμενα δεδομένα στο αντικείμενο `ServletContext`, αντί σε μεταβλητές στιγμιοτύπου. Η κλάση `ServletContext` διαθέτει τις μεθόδους `setAttribute` και `getAttribute`, οι οποίες σας επιτρέπουν να αποθηκεύσετε τυχαία δεδομένα που σχετίζονται με καθορισμένα κλειδιά. Η διαφορά μεταξύ της αποθήκευσης δεδομένων σε μεταβλητές στιγμιοτύπου και της αποθήκευσης σε ένα αντικείμενο `ServletContext` είναι ότι το αντικείμενο `ServletContext` είναι κοινόχρηστο στις μικροϋπηρεσίες και τις σελίδες JSP της εφαρμογής Ιστού, ενώ οι μεταβλητές στιγμιοτύπου είναι διαθέσιμες μόνο στη μικροϋπηρεσία που αποθήκευσε τα δεδομένα.

- `config`
  Αυτή η μεταβλητή είναι το αντικείμενο `ServletConfig` της σελίδας. Θεωρητικά μπορείτε να τη χρησιμοποιήσετε για να διαβάσετε παραμέτρους απόδοσης αρχικών τιμών, αλλά στην πράξη οι παράμετροι απόδοσης αρχικών τιμών διαβάζονται από την `jspInit`, και όχι από την `_jspService`.
- `pageContext`
  Η JSP εισήγαγε μια κλάση με όνομα `PageContext` για να δώσει ένα μοναδικό σημείο πρόσβασης σε πολλές από τις ιδιότητες της σελίδας. Η κλάση `PageContext` διαθέτει μεθόδους όπως οι `getRequest`, `getResponse`, `getOut`, `getSession`, κ.ο.κ. Η μεταβλητή `pageContext` αποθηκεύει την τιμή του αντικειμένου `PageContext` που σχετίζεται με την τρέχουσα σελίδα. Αν κάποια μέθοδος ή συνάρτηση δόμησης (constructor) πρέπει να προσπελάσει πολλά αντικείμενα που σχετίζονται με μια σελίδα, τότε η μεταβίβαση της `pageContext` είναι ευκολότερη λύση από τη μεταβίβαση πολλών ξεχωριστών αναφορών στις `request`, `response`, `out` κ.ο.κ.
- `page`
  Αυτή η μεταβλητή είναι απλώς ένα συνώνυμο του αντικειμένου `this`, και δεν είναι ιδιαίτερα χρήσιμη. Δημιουργήθηκε ως δεσμευτικό θέσης για την περίπτωση που η γλώσσα σεναρίου θα μπορούσε να είναι κάτι άλλο εκτός από Java.

11.13 Σύγκριση παραστάσεων, μικροσεναρίων, και δηλώσεων JSP

Αυτή η ενότητα περιέχει αρκετά παρόμοια παραδείγματα, καθένα από τα οποία παράγει τυχαίους ακεραίους από το 1 μέχρι το 10. Τα παραδείγματα αυτά δείχνουν τις διαφορές στον τρόπο χρήσης των τριών στοιχείων σεναρίου της JSP. Όλες οι σελίδες χρησιμοποιούν τη μέθοδο `randomInt` που ορίζεται στη Λίστα 11.13.

Λίστα 11.13 RanUtilities.java

```
package coreservlets; // Πάντοτε να χρησιμοποιείτε πακέτα!!

/** Απλή βοηθητική κλάση για την παραγωγή τυχαίων ακεραίων. */

public class RanUtilities {

  /** Τυχαίος int από 1 μέχρι (και το) range. */

  public static int randomInt(int range) {
    return(1 + ((int)(Math.random() * range)));
  }

  /** Ρουτίνα ελέγχου. Κλήση από τη γραμμή διαταγών με
```

Λίστα 11.13 RanUtilities.java *(συνέχεια)*

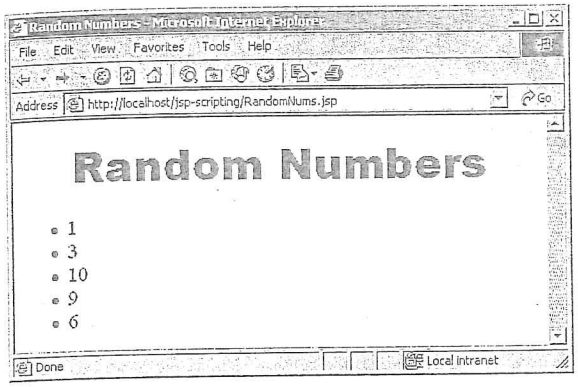
```
   *  το επιθυμητό εύρος. Θα τυπώσει 100 τιμές.
   *  Επαλήθευση ότι βλέπετε τιμές από 1 μέχρι (και) range
   *  και καμία τιμή έξω από αυτό το διάστημα.
   */

  public static void main(String[] args) {
    int range = 10;
    try {
     range = Integer.parseInt(args[0]);
    } catch(Exception e) { // Αριθμοδείκτης πίνακα ή μορφή αριθμού
      // Μην κάνεις τίποτα: η range έχει ήδη προεπιλεγμένη τιμή
    }
    for(int i=0; i<100; i++) {
      System.out.println(randomInt(range));
    }
  }
   }
```

Παράδειγμα 1: Παραστάσεις JSP

Στο πρώτο παράδειγμα ο στόχος είναι η εμφάνιση πέντε τυχαίων ακεραίων από το 1 μέχρι το 10, σε μορφή λίστας με κουκκίδες. Επειδή η δομή αυτής της σελίδας είναι σταθερή και χρησιμοποιούμε μια ξεχωριστή βοηθητική κλάση για τη μέθοδο `randomInt`, οι παραστάσεις JSP είναι το μόνο που χρειαζόμαστε. Η Λίστα 11.14 παρουσιάζει τον κώδικα και η Εικόνα 11-11 δείχνει ένα τυπικό αποτέλεσμα.

Λίστα 11.14 RandomNums.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Random Numbers</title>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css"></head>
<BODY>
<H1>Random Numbers</H1>
<UL>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
  <LI><%= coreservlets.RanUtilities.randomInt(10) %>
</UL>
   </BODY></HTML>
```

Εικόνα 11-11 Αποτέλεσμα της σελίδας **RandomNums.jsp.** Όποτε ξαναφορτώνεται η σελίδα εμφανίζονται διαφορετικές τιμές.

Παράδειγμα 2: Μικροσενάρια JSP

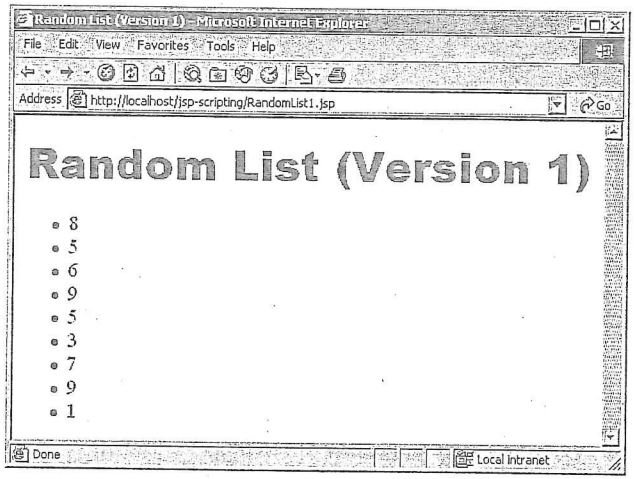
Στο δεύτερο παράδειγμα ο στόχος είναι η παραγωγή μιας λίστας με καταχωρίσεις από 1 μέχρι 10 (τυχαία επιλεγμένες), όπου κάθε καταχώριση έχει έναν αριθμό από το 1 μέχρι το 10. Επειδή ο αριθμός των καταχωρίσεων της λίστας είναι δυναμικός, είναι απαραίτητο ένα μικροσενάριο JSP. Όμως θα πρέπει να υπάρχει ένα μόνο μικροσενάριο το οποίο θα περιέχει ένα βρόχο που θα εμφανίζει στην έξοδο τους αριθμούς, ή θα πρέπει να χρησιμοποιήσουμε την προσέγγιση των αιωρούμενων αγκυλών που περιγράφηκε στην Ενότητα 11.9 (Μετατροπή τμημάτων σελίδων JSP σε παραστάσεις υπό συνθήκη); Η επιλογή εδώ δεν είναι ξεκάθαρη: η πρώτη προσέγγιση παρέχει ένα πιο συμπαγές αποτέλεσμα, αλλά η δεύτερη προσέγγιση κάνει διαθέσιμο το στοιχείο `<LI>` στον προγραμματιστή Ιστού, ο οποίος μπορεί να θέλει να τροποποιήσει τον τύπο της κουκκίδας ή να προσθέσει επιπλέον στοιχεία μορφοποίησης. Γι' αυτό θα παρουσιάσουμε και τις δύο προσεγγίσεις. Η Λίστα 11.15 παρουσιάζει την πρώτη προσέγγιση (ένα βρόχο που χρησιμοποιεί την προκαθορισμένη μεταβλητή `out`)· η Λίστα 11.16 παρουσιάζει τη δεύτερη προσέγγιση (σύλληψη της "στατικής" HTML σε ένα βρόχο). Οι Εικόνες 11-12 και 11-13 δείχνουν μερικά τυπικά αποτελέσματα.

Λίστα 11.15 RandomList1.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Random List (Version 1)</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css"></head>
<BODY>
<H1>Random List (Version 1)</H1>
```

Λίστα 11.15 RandomList1.jsp *(συνέχεια)*

```
<UL>
<%
int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numentries; i++) {
  out.println=""><LI>" + coreservlets.RanUtilities.randomInt(10));
}
%>
</UL>
</BODY></HTML>
```



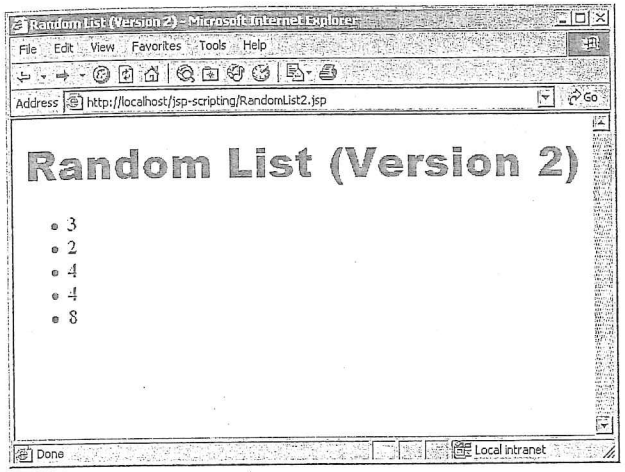
Εικόνα 11-12 Αποτέλεσμα της σελίδας **ReandomList1.jsp.** Όταν επαναφορτώνεται η σελίδα εμφανίζονται διαφορετικές τιμές (και διαφορετικός αριθμός στοιχείων στη λίστα).

Λίστα 11.16 RandomList2.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Random List (Version 2)</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css"></head>
<BODY>
<H1>Random List (Version 1)</H1>
<UL>
```


Λίστα 11.16 RandomList2.jsp *(συνέχεια)*

```
<%
int numEntries = coreservlets.RanUtilities.randomInt(10);
for(int i=0; i<numentries; i++) {
>%
<LI><%= coreservlets.RanUtilities.randomInt(10) %>
<% } %>
</UL>
</BODY></HTML>
```



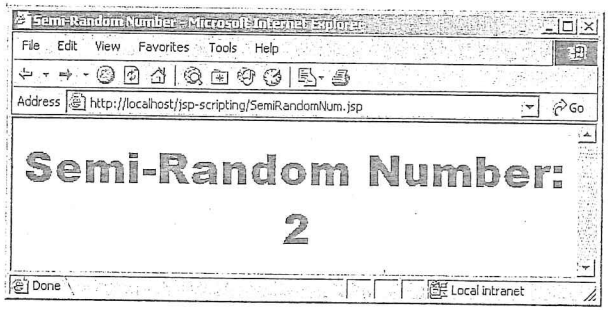
Εικόνα 11-13 Αποτέλεσμα της σελίδας ReandomList2.jsp. Όταν επαναφορτώνεται η σελίδα εμφανίζονται διαφορετικές τιμές (και διαφορετικός αριθμός στοιχείων στη λίστα).

Παράδειγμα 3: Δηλώσεις JSP

Σε αυτό το τρίτο παράδειγμα ο στόχος είναι η παραγωγή ενός τυχαίου αριθμού κατά την πρώτη αίτηση, και μετά η εμφάνιση του *ίδιου* αριθμού μέχρι να γίνει επανεκκίνηση του διακομιστή. Ο φυσικός τρόπος για την επίτευξη αυτής της διατήρησης είναι οι μεταβλητές στιγμιοτύπου (πεδία). Ο λόγος είναι ότι οι μεταβλητές στιγμιοτύπου παίρνουν αρχικές τιμές μόνο όταν δομείται το αντικείμενο, και οι μικροϋπηρεσίες δομούνται μία φορά και παραμένουν στη μνήμη μεταξύ των αιτήσεων: δεν εκχωρείται νέο στιγμιότυπο για κάθε νέα αίτηση. Οι παραστάσεις και τα μικροσενάρια JSP χειρίζονται μόνο τον κώδικα που βρίσκεται στο εσωτερικό της μεθόδου `_jspService`, οπότε δεν είναι κατάλληλα στην περίπτωση αυτή. Αυτό που χρειαζόμαστε είναι μια δήλωση JSP. Η Λίστα 11.17 παρουσιάζει τον κώδικα και η Εικόνα 11-14 δείχνει ένα τυπικό αποτέλεσμα.

Λίστα 11.17 SemiRandomNumber.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Semi-Random Number</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css"></head>
<BODY>
<%!
private int randomNum = coreservlets.RanUtilities.randomInt(10);
%>
<H1>Semi-Random Number:<BR><%= randomNum %></H1>
   </BODY></HTML>
```



Εικόνα 11-14 Αποτέλεσμα της σελίδας SemiRandomNumber.jsp. Μέχρι να γίνει επανεκκίνηση του διακομιστή, όλοι οι πελάτες θα βλέπουν το ίδιο αποτέλεσμα.

ΕΛΕΓΧΟΣ ΤΗΣ ΔΟΜΗΣ ΤΩΝ ΜΙΚΡΟΫΠΗΡΕΣΙΩΝ: Η ΟΔΗΓΙΑ PAGE

Θέματα σε αυτό το Κεφάλαιο

- Κατανόηση του σκοπού της οδηγίας `page`
- Προσδιορισμός των κλάσεων που θα εισαχθούν
- Καθορισμός του τύπου MIME της σελίδας
- Παραγωγή φύλλων εργασίας Excel
- Συμμετοχή σε συνεδρίες
- Καθορισμός του μεγέθους και της συμπεριφοράς του χώρου προσωρινής αποθήκευσης εξόδου
- Καθορισμός των σελίδων που θα χειριστούν σφάλματα JSP
- Έλεγχος της νηματικής συμπεριφοράς
- Χρήση συμβατής με XML σύνταξης για τις οδηγίες

Κεφάλαιο 12

Η οδηγία (directive) JSP επηρεάζει τη συνολική δομή της μικροϋπηρεσίας που προκύπτει από τη σελίδα JSP. Τα παρακάτω πρότυπα δείχνουν τις δύο δυνατές μορφές για τις οδηγίες. Μπορούν να χρησιμοποιηθούν απλά εισαγωγικά αντί για διπλά εισαγωγικά γύρω από τις τιμές ιδιοτήτων, αλλά δεν μπορούν να παραλειφθούν τελείως τα εισαγωγικά. Για να έχετε εισαγωγικά μέσα στην τιμή μιας ιδιότητας, θα πρέπει να γράψετε πριν από αυτά μια κάθετο: \' για ' και \" για ".

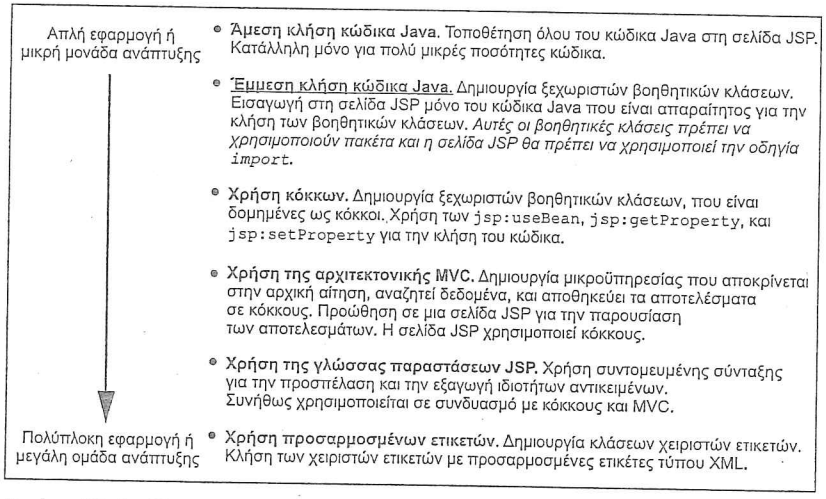
```
<%@ directive ιδιότητα="τιμή" %>
<%@ directive ιδιότητα1="τιμή1" %>
              ιδιότητα2="τιμή2"
              ...
              ιδιότηταN="τιμήN"
```

Στη JSP υπάρχουν τρεις κύριοι τύποι οδηγιών: οι `page`, `include`, και `taglib`. Η οδηγία `page` σας επιτρέπει να ρυθμίσετε τη δομή της μικροϋπηρεσίας με την εισαγωγή κλάσεων, την προσαρμογή της υπερκλάσης της μικροϋπηρεσίας, τον ορισμό του τύπου του περιεχομένου, και άλλα παρόμοια θέματα. Η οδηγία `page` μπορεί να τοποθετηθεί σε οποιοδήποτε σημείο στο εσωτερικό του εγγράφου· η χρήση της είναι το θέμα αυτού του κεφαλαίου. Η δεύτερη οδηγία, η `include`, σας επιτρέπει να ενσωματώσετε στη σελίδα JSP ένα αρχείο κατά τη στιγμή της μετάφρασης του αρχείου JSP σε μικροϋπηρεσία. Η οδηγία `include` θα πρέπει να τοποθετηθεί στο έγγραφο στο σημείο όπου θέλετε να γίνει η εισαγωγή του αρχείου· θα την περιγράψουμε στο Κεφάλαιο 13. Η τρίτη οδηγία, η `taglib`, ορίζει προσαρμοσμένες ετικέτες σήμανσης· περιγράφεται λεπτομερώς στο βιβλίο More Servlets and JavaServer Pages, όπου υπάρχουν αρκετά κεφάλαια σχετικά με βιβλιοθήκες προσαρμοσμένων ετικετών.

Η οδηγία page σάς επιτρέπει να ορίσετε μία ή περισσότερες από τις ακόλουθες ιδιότητες (στις ιδιότητες αυτές γίνεται διάκριση πεζών και κεφαλαίων χαρακτήρων, και τις παρουσιάζουμε ανάλογα με τη συχνότητα χρήσης τους): import, contentType, pageEncoding, session, isELIgnored (μόνο για την JSP 2.0), buffer, autoFlush, info, errorPage, isErrorPage, isThreadSafe, language, και extends. Αυτές οι ιδιότητες επεξηγούνται στις ενότητες που ακολουθούν.

12.1 Η ιδιότητα import

Η ιδιότητα import της οδηγίας page σάς επιτρέπει να καθορίσετε τα πακέτα που θα πρέπει να εισαχθούν από τη μικροϋπηρεσία στην οποία μεταφράζεται η σελίδα JSP. Όπως περιγράψαμε στην Ενότητα 11.3 (Περιορισμός της ποσότητας κώδικα Java σε σελίδες JSP) και δείξαμε στην Εικόνα 12-1, η χρήση ξεχωριστών βοηθητικών (helper) κλάσεων κάνει το δυναμικό σας κώδικα πιο εύκολο στη συγγραφή, τη συντήρηση, τον έλεγχο, και την επαναχρησιμοποίηση.



Εικόνα 12-1 Στρατηγικές κλήσης δυναμικού κώδικα Java από σελίδες JSP.

Όταν χρησιμοποιείτε βοηθητικές κλάσεις, να θυμάστε ότι θα πρέπει *πάντοτε* να βρίσκονται σε πακέτα. Ένας λόγος είναι ότι τα πακέτα είναι μια καλή στρατηγική για όλα τα μεγάλα έργα, επειδή παρέχουν προστασία όσον αφορά τις διενέξεις ονομάτων. Με τις σελίδες JSP, όμως, τα πακέτα είναι απολύτως αναγκαία. Η αιτία είναι ότι, αν απουσιάζουν τα πακέτα, οι κλάσεις στις οποίες κάνετε αναφορά θεωρούνται ότι βρίσκονται στο ίδιο πακέτο με την τρέχουσα κλάση. Για παράδειγμα, υποθέστε ότι μια σελίδα JSP περιέχει το παρακάτω μικροσενάριο:

```
<% Test t = new Test(); %>
```


Τώρα, αν η Test είναι ένα εισαγόμενο πακέτο, δεν υπάρχει καμία ασάφεια. Αν όμως η Test δεν ανήκει στο πακέτο, ή δεν έχει εισαχθεί ρητά το πακέτο στο οποίο ανήκει η Test, τότε το σύστημα θα υποθέσει ότι η Test βρίσκεται στο ίδιο πακέτο με τη μικροϋπηρεσία που έχει παραχθεί αυτόματα. Το πρόβλημα είναι ότι το πακέτο της μικροϋπηρεσίας που παράγεται αυτόματα δεν είναι γνωστό! Είναι αρκετά συνηθισμένο για τους διακομιστές να δημιουργούν μικροϋπηρεσίες των οποίων τα πακέτα καθορίζονται από τον κατάλογο στον οποίο βρίσκεται η σελίδα JSP. Άλλοι διακομιστές χρησιμοποιούν διαφορετικές προσεγγίσεις. Έτσι δεν μπορείτε να θεωρείτε απλώς ότι οι κλάσεις χωρίς πακέτα θα λειτουργούν σωστά. Το ίδιο επιχείρημα ισχύει και για τους κόκκους (Κεφάλαιο 14), αφού οι κόκκοι είναι απλώς κλάσεις που ακολουθούν απλές συμβάσεις ονομασίας και δομής.

Η προσέγγιση του βιβλίου

Πάντοτε να τοποθετείτε σε πακέτα τις βοηθητικές κλάσεις και τους κόκκους που δημιουργείτε.

Εξ ορισμού, η μικροϋπηρεσία εισάγει τις κλάσεις java.lang.*, javax.servlet.*, javax.servlet.jsp.*, javax.servlet.http.*, και πιθανόν και ορισμένες άλλες κλάσεις ειδικά για τον κάθε διακομιστή. Ποτέ μη γράφετε κώδικα JSP που βασίζεται στην αυτόματη εισαγωγή κλάσεων που αφορούν ειδικά ένα συγκεκριμένο διακομιστή· κάτι τέτοιο θα κάνει τον κώδικά σας μη φορητό.

Η χρήση της ιδιότητας import μπορεί να πάρει μια από τις ακόλουθες μορφές:

```
<%@ page import="πακέτο.κλάση" %>
<%@ page import="πακέτο.κλάση1, ... πακέτο.κλάσηN" %>
```

Για παράδειγμα, η ακόλουθη οδηγία σηματοδοτεί ότι όλες οι κλάσεις του πακέτου java.util θα πρέπει να είναι διαθέσιμες για χρήση χωρίς ρητά αναγνωριστικά πακέτου.

```
<%@ page import="java.util.*" %>
```

Η ιδιότητα import είναι η μόνη ιδιότητα της οδηγίας page που μπορεί να εμφανίζεται πολλές φορές στο ίδιο έγγραφο. Αν και οι οδηγίες page μπορούν να βρίσκονται σε οποιοδήποτε σημείο στο έγγραφο, κατά παράδοση οι εντολές import τοποθετούνται είτε κοντά στην αρχή του εγγράφου, είτε ακριβώς πριν από τη θέση όπου χρησιμοποιείται για πρώτη φορά το αναφερόμενο πακέτο.

Ας σημειωθεί ότι, αν και οι κανονικές σελίδες JSP μπορούν να τοποθετηθούν στους κανονικούς καταλόγους HTML του διακομιστή, οι κλάσεις που γράφετε και οι οποίες χρησιμοποιούνται από τις σελίδες JSP θα πρέπει να τοποθετούνται στους ειδικούς καταλόγους για τον κώδικα Java (για παράδειγμα, .../WEB-INF/classes/*καταλόγοςΠουΤαιριάζειΜεΤοΌνομαΤουΠακέτου*). Για πληροφορίες σχετικά με αυτούς τους καταλόγους δείτε τις Ενότητες 2.10 (Κατάλογοι εκδίπλωσης για την προεπιλεγμένη εφαρμογή Ιστού: Σύνοψη) και 2.11 (Εφαρμογές Ιστού: Μια προεπισκόπηση).

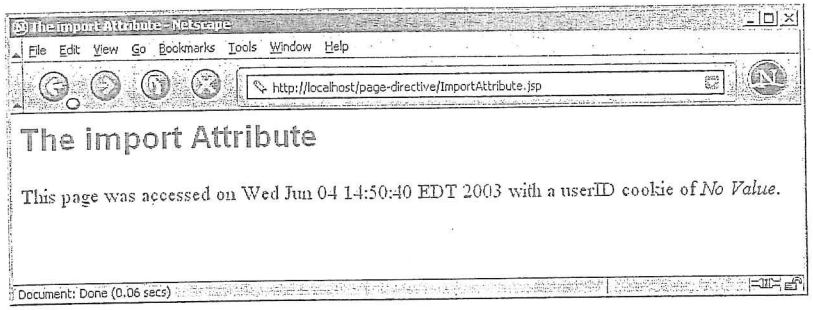
Για παράδειγμα, η Λίστα 12.1 παρουσιάζει μια σελίδα η οποία δείχνει και τα τρία στοιχεία σεναρίου του προηγούμενου κεφαλαίου. Η σελίδα χρησιμοποιεί τρεις κλάσεις που δεν βρίσκονται στην τυπική λίστα εισαγωγής της JSP: τις κλάσεις `java.util.Date`, `coreservlets.CookieUtilities` (δείτε τη Λίστα 8.3), και `coreservlets.LongLivedCookie` (δείτε τη Λίστα 8.4). Έτσι, για απλοποίηση των αναφορών σε αυτές τις κλάσεις, η σελίδα JSP χρησιμοποιεί την οδηγία

```
<%@ page import="java.util.*, coreservlets.*" %>
```

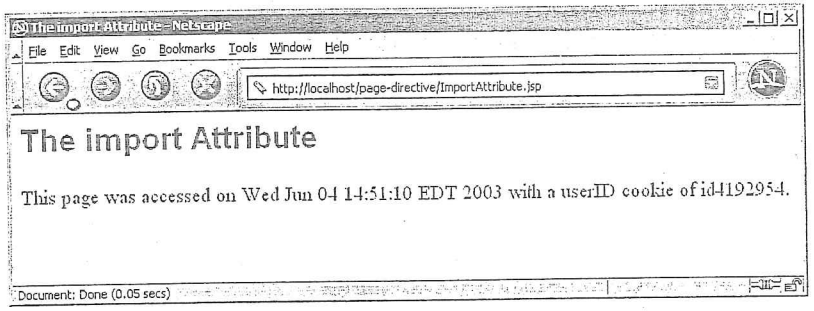
Οι Εικόνες 12-2 και 12-3 δείχνουν μερικά τυπικά αποτελέσματα.

Λίστα 12.1 ImportAttribute.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>The import Attribute</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H2>The import Attribute</H2>
<%-- JSP page Directive --%>
<%@ page import="java.util.*,coreservlets.*" %>
<%-- JSP Declaration --%>
<%!
private String randomID() {
  int num = (int)(Math.random()*10000000.0);
  return("id" + num);
}
private final String NO_VALUE = "<I>No Value</I>";
%>
<%-- JSP Scriptlet --%>
<%
String oldID =
  CookieUtilities.getCookieValue(request, "userID", NO_VALUE);
if (oldID.equals(NO_VALUE)) {
  String newID = randomID();
  Cookie cookie = new LongLivedCookie("userID", newID);
  response.addCookie(cookie);
}
%>
<%-- JSP Expressions --%>
This page was accessed on <%= new Date() %> with a userID
cookie of <%= oldID %>.
   </BODY></HTML>
```

Εικόνα 12-2 Όταν προσπελάζουμε για πρώτη φορά τη σελίδα ImportAttribute.jsp.



Εικόνα 12-3 Όταν προσπελάζεται σε επόμενη αίτηση η σελίδα ImportAttribute.jsp.

12.2 Οι ιδιότητες contentType και pageEncoding

Η ιδιότητα `contentType` ορίζει την κεφαλίδα απάντησης `Content-Type`, η οποία προσδιορίζει τον τύπο MIME (Multiple Internet Mail Extension, γενικές επεκτάσεις ταχυδρομείου Internet) του εγγράφου που αποστέλλεται στον πελάτη. Για περισσότερες πληροφορίες σχετικά με τους τύπους MIME δείτε τον Πίνακα 7.1 (Συνηθισμένοι τύποι MIME) στην Ενότητα 7.2 (Κατανόηση των κεφαλίδων απάντησης του πρωτοκόλλου HTTP 1.1).

Η χρήση της ιδιότητας `contentType` παίρνει μια από τις ακόλουθες δύο μορφές:

```
<%@ page contentType="Τύπος-MIME" %>
<%@ page contentType="Τύπος-MIME; charset=Σύνολο-Χαρακτήρων" %>
```

Για παράδειγμα, η οδηγία

```
<%@ page contentType="application/vnd.ms-excel" %>
```

έχει το ίδιο αποτέλεσμα με το μικροσενάριο

```
<% response.setContentType("application/vnd.ms-excel"); %>
```

Η πρώτη διαφορά μεταξύ των δύο μορφών είναι ότι η εντολή `response.setContentType` κάνει ρητή χρήση κώδικα Java (μια προσέγγιση που ορισμένοι προγραμματιστές προσπαθούν να αποφύγουν), ενώ η οδηγία `page` χρησιμοποιεί μόνο σύνταξη JSP. Η δεύτερη διαφορά είναι ότι οι οδηγίες αναλύονται συντακτικά (parse) με ειδικό τρόπο· δεν μετατρέπονται άμεσα σε κώδικα `_jspService` στη θέση στην οποία εμφανίζονται. Αυτό σημαίνει ότι η μέθοδος `response.setContentType` μπορεί να κληθεί μέσα από παραστάσεις υπό συνθήκη, ενώ η οδηγία `page` δεν μπορεί. Ο καθορισμός του τύπου του περιεχομένου με βάση μια παράσταση υπό συνθήκη είναι κάτι χρήσιμο όταν το ίδιο το περιεχόμενο μπορεί να εμφανιστεί με διαφορετικές μορφές — για ένα σχετικό παράδειγμα δείτε την επόμενη ενότητα "Παραγωγή φύλλων εργασίας Excel υπό συνθήκη".

Σε αντίθεση με τις κανονικές μικροϋπηρεσίες, για τις οποίες ο προεπιλεγμένος τύπος MIME είναι ο `text/plain`, ο προεπιλεγμένος τύπος των σελίδων JSP είναι `text/html` (με προεπιλεγμένο σύνολο χαρακτήρων `ISO-8859-1`). Έτσι οι σελίδες JSP που παράγουν έξοδο HTML στο σύνολο χαρακτήρων Latin δεν χρειάζεται να χρησιμοποιήσουν την ιδιότητα `contentType`. Αν θέλετε να αλλάξετε τόσο τον τύπο περιεχομένου, όσο και το σύνολο χαρακτήρων, μπορείτε να το κάνετε με την ακόλουθη οδηγία:

```
<%@ page contentType="κάποιοςΤύποςMIME"; charset="κάποιοΣύνολοΧαρακτήρων" %>
```

Αν όμως θέλετε να αλλάξετε μόνο το σύνολο χαρακτήρων, είναι πιο απλό να χρησιμοποιήσετε την ιδιότητα `pageEncoding`. Για παράδειγμα, οι Ιαπωνικές σελίδες JSP μπορούν να χρησιμοποιήσουν την παρακάτω οδηγία:

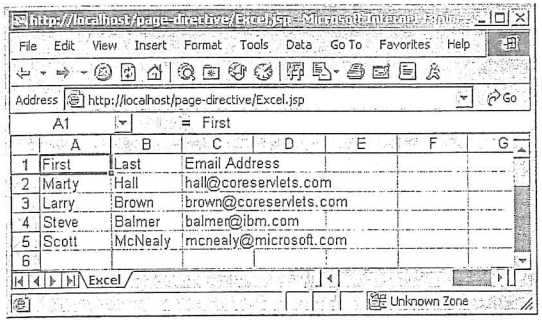
```
<%@ page pageEncoding="Shift_JIS" %>
```

Παραγωγή φύλλων εργασίας Excel

Η Λίστα 12.2 παρουσιάζει μια σελίδα JSP που χρησιμοποιεί την ιδιότητα `contentType` σε συνδυασμό με δεδομένα που διαχωρίζονται από στηλοθέτες για την παραγωγή εξόδου Excel. Παρατηρήστε ότι η οδηγία `page` και το σχόλιο βρίσκονται στο τέλος της σελίδας, έτσι ώστε οι επαναφορές κεφαλής (carriage return) που βρίσκονται στο τέλος των γραμμών να μην εμφανιστούν στο έγγραφο Excel. (Σημείωση: η JSP δεν παραβλέπει τα λευκά διαστήματα — συνήθως η σελίδα JSP παράγει κώδικα HTML όπου τα περισσότερα λευκά διαστήματα δεν λαμβάνονται υπόψη από το φυλλομετρητή, αλλά η ίδια η JSP διατηρεί τα διαστήματα και τα στέλνει στον πελάτη). Η Εικόνα 12-4 δείχνει το αποτέλεσμα στον Internet Explorer σε ένα σύστημα όπου είναι εγκατεστημένο το Microsoft Office.

Λίστα 12.2 Excel.jsp

```
First  Last      Email Address
Marty  Hall      hall@coreservlets.com
Larry  Brown     brown@coreservlets.com
Steve  Balmer    balmer@ibm.com
Scott  McNealy   mcnealy@microsoft.com
<%@ page contentType="application/vnd.ms-excel" %>
   <%-- Υπάρχουν στηλοθέτες, και όχι κενά, ανάμεσα στις στήλες. --%>
```



Εικόνα 12-4 Έγγραφο Excel (Excel.jsp) στον Internet Explorer.

12.3 Παραγωγή φύλλων εργασίας Excel υπό συνθήκη

Στις περισσότερες περιπτώσεις όπου δεν παράγετε περιοχόμενο HTML από τις σελίδες JSP, θα γνωρίζετε από πριν τον τύπο του περιεχομένου. Σε αυτές τις περιπτώσεις η ιδιότητα `contentType` της οδηγίας `page` είναι η κατάλληλη λύση: δεν χρειάζεται ρητή σύνταξη Java και μπορεί να τοποθετηθεί οπουδήποτε στη σελίδα.

Περιστασιακά, όμως, μπορεί να θέλετε να κατασκευάσετε το ίδιο περιεχόμενο αλλά να αλλάζετε τον τύπο του περιεχομένου, ανάλογα με την περίσταση. Για παράδειγμα, πολλά συστήματα επεξεργασίας κειμένου και επιτραπέζιας έκδοσης εντύπων έχουν δυνατότητα εισαγωγής σελίδων HTML. Έτσι θα μπορούσατε να κανονίσετε να εμφανίζεται η σελίδα είτε στο σύστημα σελιδοποίησης είτε στο φυλλομετρητή, ανάλογα με τον τύπο του περιεχομένου που στέλνετε. Παρομοίως, το Microsoft Excel μπορεί να εισαγάγει πίνακες που προσδιορίζονται με την ετικέτα `TABLE` της HTML. Αυτή η δυνατότητα μας παρέχει μια απλή μέθοδο για την επιστροφή περιεχομένου είτε σε μορφή HTML είτε σε μορφή Excel, ανάλογα με τις προτιμήσεις του χρήστη: χρησιμοποιείτε απλώς έναν πίνακα HTML και ορίζετε τον τύπο του περιεχομένου σε `application/vnd.ms-excel` μόνον όταν ο χρήστης ζητά τα αποτελέσματα σε μορφή Excel.

Δυστυχώς, αυτή η προσέγγιση φέρνει στην επιφάνεια μια μικρή αδυναμία της οδηγίας `page`: οι τιμές των ιδιοτήτων δεν μπορούν να υπολογιστούν κατά το χρόνο εκτέλεσης, ούτε μπορεί να γίνει εισαγωγή οδηγιών `page` υπό συνθήκη, όπως γίνεται με το κείμενο προτύπου. Έτσι η παρα-

κάτω προσπάθεια έχει ως αποτέλεσμα περιεχόμενο Excel, ανεξάρτητα από το αποτέλεσμα της μεθόδου checkUserRequest.

```
<% boolean usingExcel = checkUserRequest(request); %>
<% if (usingExcel) { %>
<%@ page contentType="application/vnd.ms-excel" %>
<% } %>
```

Για παράδειγμα, κάποτε εργαστήκαμε σε ένα έργο που εμφάνιζε λογιστικές πληροφορίες (στοιχεία προϋπολογισμού) σε εξουσιοδοτημένους χρήστες. Τα δεδομένα μπορούσαν να εμφανιστούν σε έναν πίνακα κανονικής ιστοσελίδας, αν ο χρήστης ήθελε απλώς να τα εξετάσει, ή να τοποθετηθούν σε ένα φύλλο εργασίας Excel αν ο χρήστης ήθελε να τα μεταφέρει σε κάποια αναφορά. Όταν ήρθαμε σε επαφή με το έργο για πρώτη φορά, υπήρχαν δύο εντελώς διαφορετικά τμήματα κώδικα για την κάθε εργασία. Το αλλάξαμε αυτό έτσι ώστε να κατασκευάζεται ο ίδιος πίνακας HTML και στις δύο περιπτώσεις, και απλώς αλλάξαμε τον τύπο του περιεχομένου. Voila!

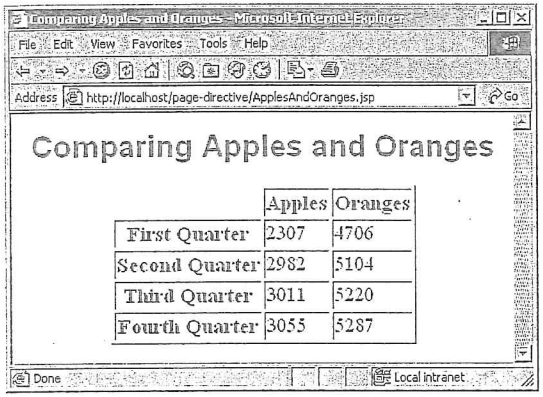
Η Λίστα 12.3 παρουσιάζει μια σελίδα που χρησιμοποιεί αυτή την προσέγγιση· οι Εικόνες 12-5 και 12-6 δείχνουν τα αποτελέσματα. Φυσικά, σε μια πραγματική εφαρμογή, τα δεδομένα είναι σχεδόν σίγουρο ότι θα προέρχονται από κάποια βάση δεδομένων. Για λόγους απλότητας εδώ χρησιμοποιούμε στατικές τιμές, αλλά μπορείτε να δείτε το Κεφάλαιο 17 (Προσπέλαση βάσεων δεδομένων με την JDBC) για πληροφορίες σχετικά με την επικοινωνία με βάσεις δεδομένων από μικροϋπηρεσίες και σελίδες JSP.

Λίστα 12.13 ApplesAndOranges.jsp

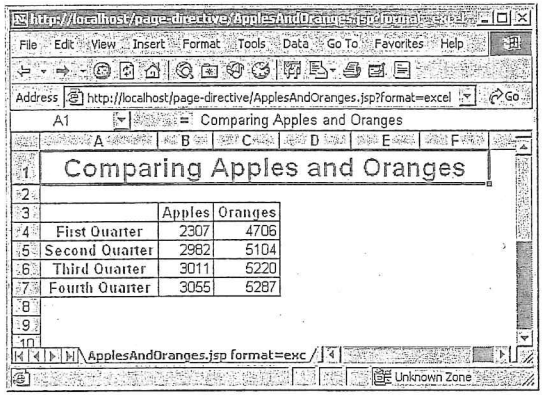
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Comparing Apples and Oranges</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<H2>Comparing Apples and Oranges</H2>
<%
String format = request.getParameter("format");
if ((format != null) &amp;&amp; (format.equals("excel"))) {
  response.setContentType("application/vnd.ms-excel");
}
%>
<TABLE BORDER="1">
  <TR><TH></TH>          <TH>Apples<TH>Oranges
  <TR><TH>First Quarter <TD>2307  <TD>4706
```


Λίστα 12.13 ApplesAndOranges.jsp *(συνέχεια)*

```
  <TR><TH>Second Quarter<TD>2982  <TD>5104
  <TR><TH>Third Quarter <TD>3011  <TD>5220
  <TR><TH>Fourth Quarter<TD>3055  <TD>5287
</TABLE>
   </CENTER></BODY></HTML>
```



Εικόνα 12-5 Το προεπιλεγμένο αποτέλεσμα της σελίδας ApplesAndOranges.jsp είναι περιεχόμενο HTML.



Εικόνα 12-6 Αν καθορίσουμε `format=excel` στη σελίδα ApplesAndOranges.jsp, θα προκύψει περιεχόμενο σε μορφή Excel.

12.4 Η ιδιότητα session

Η ιδιότητα `session` ελέγχει αν η σελίδα συμμετέχει σε συνεδρίες HTTP. Η χρήση αυτής της ιδιότητας παίρνει μια από τις ακόλουθες μορφές:

```
<%@ page session="true" %> <%-- Default --%>
<%@ page session="false" %>
```

Η τιμή `true` (προεπιλεγμένη τιμή) σηματοδοτεί ότι η προκαθορισμένη μεταβλητή `session` (τύπου `HttpSession`) θα πρέπει να συνδεθεί με την υπάρχουσα συνεδρία, αν αυτή υπάρχει· διαφορετικά θα πρέπει να δημιουργηθεί μια νέα συνεδρία και να αποδοθεί στη `session`. Η τιμή `false` σημαίνει ότι δεν θα δημιουργηθούν αυτόματα συνεδρίες και ότι οι προσπάθειες να προσπελαστεί η μεταβλητή `session` θα οδηγήσουν σε σφάλματα κατά το χρόνο μετάφρασης της σελίδας JSP σε μικροϋπηρεσία.

Η χρήση της οδηγίας `session="false"` μπορεί να εξοικονομήσει μεγάλες ποσότητες μνήμης του διακομιστή σε τοποθεσίες με μεγάλη κυκλοφορία. Ωστόσο, θα πρέπει να σημειώσετε ότι η χρήση της οδηγίας `session="false"` δεν *απενεργοποιεί* την παρακολούθηση συνεδριών — απλώς εμποδίζει τις σελίδες JSP να δημιουργήσουν νέες συνεδρίες για χρήστες που δεν έχουν ήδη. Έτσι, επειδή οι συνεδρίες είναι *ειδικές για κάθε χρήστη* (user-specific), και όχι *ειδικές για κάθε σελίδα* (page-specific), δεν υπάρχει όφελος από την απενεργοποίηση της παρακολούθησης συνεδριών για μια σελίδα εκτός και αν απενεργοποιήσετε επίσης την επιλογή αυτή για τις σχετικές σελίδες που είναι πιθανόν να δεχτούν επίσκεψη στην ίδια συνεδρία του πελάτη.

12.5 Η ιδιότητα isELIgnored

Η ιδιότητα `isELIgnored` ελέγχει αν θα παραβλέπεται η Γλώσσα Παραστάσεων (Expression Language, EL) της JSP 2.0 (`true`) ή θα υπολογίζεται κανονικά (`false`). Αυτή η ιδιότητα είναι καινούργια στη JSP 2.0, και δεν είναι έγκυρη η χρήση της στην JSP 1.2 ή τις προγενέστερες εκδόσεις. Η προεπιλεγμένη τιμή της ιδιότητας εξαρτάται από την έκδοση του αρχείου web.xml που χρησιμοποιείται από την εφαρμογή Ιστού. Αν το αρχείο web.xml καθορίζει μικροϋπηρεσίες 2.3 (που αντιστοιχούν σε JSP 1.2) ή κάποια προηγούμενη έκδοση, η προεπιλεγμένη τιμή είναι `true` (αλλά εξακολουθεί να είναι έγκυρη η αλλαγή της προεπιλεγμένης τιμής — επιτρέπεται να χρησιμοποιήσετε αυτή την ιδιότητα σε διακομιστή συμβατό με την JSP 2.0 ανεξάρτητα από την έκδοση στο αρχείο web.xml). Αν το αρχείο web.xml καθορίζει μικροϋπηρεσίες 2.4 (που αντιστοιχούν σε JSP 2.0) ή μεταγενέστερη έκδοση, η προεπιλεγμένη τιμή είναι `false`. Η χρήση αυτής της ιδιότητας παίρνει μια από τις εξής μορφές:

```
<%@ page isELIgnored="false" %>
<%@ page isELIgnored="true" %>
```

Η JSP 2.0 εισήγαγε μια συμπαγή γλώσσα παραστάσεων για την προσπέλαση των παραμέτρων αιτήσεων, των μπισκότων, των κεφαλίδων HTTP, των ιδιοτήτων κόκκων, και των στοιχείων `Collection` από το εσωτερικό μιας σελίδας JSP. Για λεπτομέρειες δείτε το Κεφάλαιο 16 (Απλοποίηση της προσπέλασης κώδικα Java: Η γλώσσα παραστάσεων της JSP 2.0). Οι παραστάσεις στη Γλώσσα Παραστάσεων της JSP 2.0 έχουν τη μορφή `${`*παράσταση*`}`. Κανονικά,

αυτές οι παραστάσεις είναι χρήσιμες. Τι θα συμβεί, όμως, αν έχετε μια σελίδα JSP 1.2 η οποία, εντελώς τυχαία, περιέχει ένα αλφαριθμητικό της μορφής `${...}`; Στη JSP 2.0 κάτι τέτοιο θα μπορούσε να δημιουργήσει προβλήματα. Η χρήση της επιλογής `isELIgnored="false"` εμποδίζει την εμφάνιση αυτών των προβλημάτων.

12.6 Οι ιδιότητες buffer και autoFlush

Η ιδιότητα `buffer` καθορίζει το μέγεθος του χώρου προσωρινής αποθήκευσης που χρησιμοποιείται από τη μεταβλητή `out`, η οποία είναι τύπου `JspWriter`. Η χρήση αυτής της ιδιότητας παίρνει μια από τις δύο παρακάτω μορφές:

```
<%@ page buffer="μέγεθοςΣεkb" %>
<%@ page buffer="none" %>
```

Οι διακομιστές μπορούν να χρησιμοποιήσουν μεγαλύτερο χώρο προσωρινής αποθήκευσης από αυτόν που καθορίζετε, αλλά όχι μικρότερο. Για παράδειγμα, η οδηγία `<%@ page buffer=32kb" %>` σημαίνει ότι το έγγραφο θα πρέπει να αποθηκευτεί προσωρινά και δεν θα σταλεί στον πελάτη αν δεν συγκεντρωθούν τουλάχιστον 32 kilobyte, ή αν δεν ολοκληρωθεί η σελίδα, ή αν δεν αδειάσει ρητά (flush) η έξοδος (π.χ., με την εντολή `response.flushBuffer`). Το προεπιλεγμένο μέγεθος του χώρου προσωρινής αποθήκευσης εξαρτάται από το διακομιστή, αλλά πρέπει να είναι τουλάχιστον 8 kilobyte. Να είστε προσεκτικοί όταν απενεργοποιείτε τη χρήση προσωρινής αποθήκευσης: αν το κάνετε αυτό, τα στοιχεία JSP που ορίζουν κεφαλίδες ή κωδικούς κατάστασης θα πρέπει να βρίσκονται στην αρχή του αρχείου, πριν από οποιοδήποτε περιεχόμενο HTML. Από την άλλη πλευρά, η απενεργοποίηση της χρήσης προσωρινής αποθήκευσης, ή η χρήση ενός μικρού χώρου προσωρινής αποθήκευσης, είναι μερικές φορές χρήσιμη όταν χρειάζεται πολύς χρόνος για την παραγωγή της κάθε γραμμής της εξόδου: σε αυτή την περίπτωση οι χρήστες θα βλέπουν κάθε γραμμή τη στιγμή που είναι έτοιμη, και δεν θα περιμένουν για να δουν ομάδες γραμμών.

Η ιδιότητα `autoFlush` ελέγχει αν ο χώρος προσωρινής αποθήκευσης εξόδου θα πρέπει να αδειάζει αυτόματα όταν είναι γεμάτος (η προεπιλεγμένη τιμή), ή αν θα πρέπει να δημιουργείται εξαίρεση όταν συμβαίνει υπερχείλιση του χώρου προσωρινής αποθήκευσης (`autoFlush="false"`). Η χρήση αυτής της ιδιότητας παίρνει μια από τις ακόλουθες δύο μορφές:

```
<%@ page autoFlush="true" %> <%-- προεπιλογή --%>
<%@ page autoFlush="false" %>
```

Η τιμή `false` δεν είναι έγκυρη όταν χρησιμοποιείται και η ιδιότητα `buffer="none"`. Η χρήση της οδηγίας `autoFlush="false"` είναι εξαιρετικά σπάνια όταν ο πελάτης είναι ένας κανονικός φυλλομετρητής Ιστού. Ωστόσο, αν ο πελάτης είναι μια προσαρμοσμένη εφαρμογή, μπορεί να θέλετε να εξασφαλίσετε ότι η εφαρμογή είτε θα λαμβάνει ένα πλήρες μήνυμα, είτε δεν θα λαμβάνει καθόλου το μήνυμα. Η τιμή `false` θα μπορούσε επίσης να χρησιμοποιηθεί για τη σύλληψη ερωτημάτων σε βάσεις δεδομένων που παράγουν πάρα πολλά δεδομένα, αλλά γενικά είναι καλύτερη η τοποθέτηση μιας τέτοιας λογικής στον κώδικα προσπέλασης των δεδομένων, και όχι στον κώδικα παρουσίασης.

12.7 Η ιδιότητα info

Η ιδιότητα `info` ορίζει ένα αλφαριθμητικό το οποίο μπορεί να ανακτηθεί από μια μικροϋπηρεσία μέσω της μεθόδου `getServletInfo`. Η χρήση της οδηγίας `info` παίρνει την ακόλουθη μορφή:

```
<%@ page info="Κάποιο μήνυμα" %>
```

12.8 Οι ιδιότητες errorPage και isErrorPage

Η ιδιότητα `errorPage` καθορίζει μια σελίδα JSP που θα πρέπει να επεξεργαστεί κάθε εξαίρεση (δηλαδή, κάτι που είναι τύπου `Throwable`) η οποία μεταβιβάζεται αλλά δεν συλλαμβάνεται από την τρέχουσα σελίδα. Χρησιμοποιείται με τον εξής τρόπο:

```
<%@ page errorPage="Σχετικό URL" %>
```

Η εξαίρεση που μεταβιβάζεται θα είναι αυτόματα διαθέσιμη στην καθορισμένη σελίδα σφάλματος μέσω της μεταβλητής `exception`.

Η ιδιότητα `isErrorPage` υποδεικνύει αν μπορεί ή όχι η τρέχουσα σελίδα να λειτουργήσει ως σελίδα σφάλματος για κάποια άλλη σελίδα JSP. Η χρήση της οδηγίας `isErrorPage` παίρνει μια από τις ακόλουθες δύο μορφές:

```
<%@ page isErrorPage="true" %>
<%@ page isErrorPage="false" %> <%-- Προεπιλογή --%>
```

Για παράδειγμα, η Λίστα 12.4 παρουσιάζει μια σελίδα JSP που υπολογίζει την ταχύτητα με βάση τις παραμέτρους απόστασης και χρόνου. Επειδή η σελίδα δεν ελέγχει αν λείπουν ή αν έχουν λανθασμένη μορφή οι παράμετροι εισόδου, θα μπορούσε πολύ εύκολα να συμβεί κάποιο σφάλμα κατά το χρόνο εκτέλεσης. Ωστόσο, η σελίδα καθορίζει τη σελίδα SpeedErrors.jsp (Λίστα 12.5) για το χειρισμό των σφαλμάτων που προκύπτουν στη σελίδα ComputeSpeed.jsp, οπότε οι χρήστες δεν θα λαμβάνουν τα τυπικά, αυστηρά μηνύματα σφαλμάτων της JSP. Προσέξτε ότι η σελίδα SpeedErrors.jsp τοποθετείται στον κατάλογο WEB-INF. Επειδή οι διακομιστές δεν επιτρέπουν στους χρήστες την άμεση προσπέλαση του καταλόγου WEB-INF, αυτή η διευθέτηση εμποδίζει τους χρήστες να προσπελάσουν κατά λάθος τη σελίδα SpeedErrors.jsp. Όταν συμβεί κάποιο σφάλμα, η σελίδα SpeedErrors.jsp προσπελάζεται από το *διακομιστή*, και όχι από τον *πελάτη*: οι σελίδες σφαλμάτων αυτού του τύπου δεν καταλήγουν σε κλήσεις `response.sendRedirect`, και ο πελάτης βλέπει μόνο το URL της σελίδας για την οποία έγινε η αρχική αίτηση, και όχι το URL της σελίδας σφάλματος.

Οι Εικόνες 12-7 και 12-8 δείχνουν τα αποτελέσματα όταν παρέχονται σωστές και λανθασμένες παράμετροι εισόδου, αντίστοιχα.

Σημειώστε ότι η ιδιότητα `errorPage` καθορίζει σελίδες σφαλμάτων *για συγκεκριμένες σελίδες*. Για να καθορίσετε σελίδες σφαλμάτων που θα ισχύουν για ολόκληρη την εφαρμογή Ιστού ή για διάφορες κατηγορίες σε μια εφαρμογή, χρησιμοποιήστε το στοιχείο `error-page` του αρχείου web.xml. Για λεπτομέρειες δείτε το βιβλίο More Servlets and JavaServer Pages.

Λίστα 12.4 ComputeSpeed.jsp

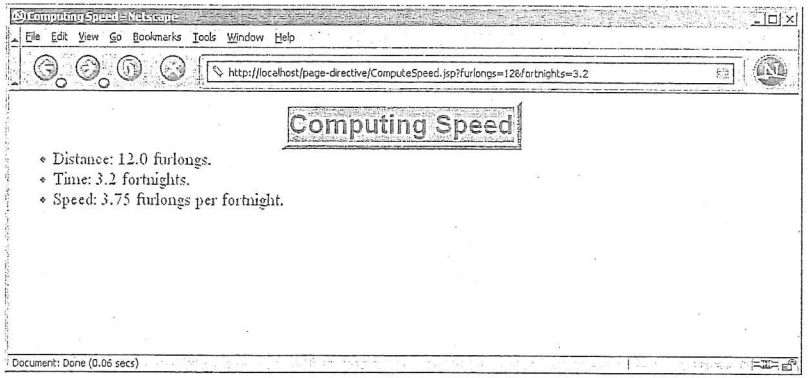
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Computing Speed</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<%@ page errorPage="/WEB-INF/SpeedErrors.jsp" %>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Computing Speed</TABLE>
<%!
// Παρατηρήστε την απουσία του μπλοκ try/catch για την NumberFormatException
// αν η τιμή είναι null ή έχει λανθασμένη μορφή.

private double toDouble(String value) {
  return(Double.parseDouble(value));
}
%>
<;%
double furlongs = toDouble(request.getParameter("furlongs"));
double fortnights = toDouble(request.getParameter("fortnights"));
double speed = furlongs/fortnights;
%>
<UL>
  <LI>Distance: <%= furlongs %> furlongs.
  <LI>Time: <%= fortnights %> fortnights.
  <LI>Speed: <%= speed %> furlongs per fortnight.
</UL>
   </BODY></HTML>
```

Λίστα 12.5 SpeedErrors.jsp

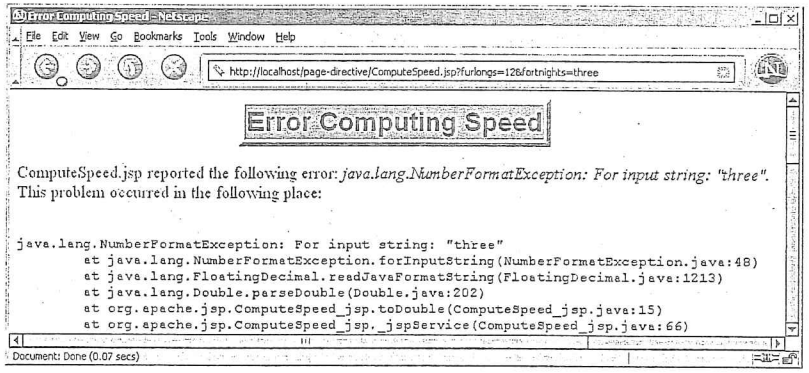
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Error Computing Speed</title>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<%@ page isErrorPage="true" %>
<TABLE BORDER=5 ALIGN="CENTER" >
  <TR><TH CLASS="TITLE">
      Error Computing Speed</TABLE>
<P>
```

Λίστα 12.5 SpeedErrors.jsp *(συνέχεια)*

```
ComputeSpeed.jsp reported the following error:
<I><%= exception %></I>. This problem occurred in the
following place:
<PRE><%@ page import="java.io.*" %>
<% exception.printStackTrace(new PrintWriter(out)); %>
</PRE>
   </BODY></HTML>
```



Εικόνα 12-7 Η σελίδα ComputeSpeed.jsp όταν δίνονται έγκυρες τιμές.



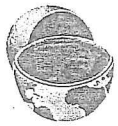
Εικόνα 12-8 Η σελίδα ComputeSpeed.jsp όταν δίνονται λανθασμένες τιμές. Προσέξτε ότι η γραμμή διευθύνσεων δείχνει το URL της σελίδας ComputeSpeed.jsp, και όχι της SpeedErrors.jsp.

12.9 Η ιδιότητα isThreadSafe

Η ιδιότητα `isThreadSafe` ελέγχει αν η μικροϋπηρεσία που προκύπτει από τη σελίδα JSP θα επιτρέπει την ταυτόχρονη προσπέλαση (η προεπιλεγμένη τιμή) ή θα εγγυάται ότι κανένα στιγμιότυπο μικροϋπηρεσίας δεν θα επεξεργάζεται περισσότερες από μία αιτήσεις κάθε φορά (`isThreadSafe="false"`). Η χρήση της ιδιότητας `isThreadSafe` παίρνει μια από τις παρακάτω δύο μορφές:

```
<%@ page isThreadSafe="true" %> <%-- Προεπιλογή --%>
<%@ page isThreadSafe="false" %>
```

Δυστυχώς, ο καθιερωμένος μηχανισμός για την παρεμπόδιση της ταυτόχρονης πρόσβασης είναι η υλοποίηση της διασύνδεσης `SingleThreadModel` (Ενότητα 3.7). Αν και στην αρχή προτεινόταν η χρήση των `SingleThreadModel` και `isThreadSafe="false"`, οι πρόσφατες εμπειρίες έδειξαν ότι η διασύνδεση `SingleThreadModel` είναι τόσο άσχημα σχεδιασμένη που είναι ουσιαστικά άχρηστη. Έτσι καλό είναι να αποφεύγετε τη χρήση της ιδιότητας `isThreadSafe="false"`, και να προτιμάτε να χρησιμοποιείτε ρητό συγχρονισμό.



Προειδοποίηση

Μη χρησιμοποιείτε την `isThreadSafe`. Χρησιμοποιήστε ρητό συγχρονισμό.

Για να καταλάβετε γιατί η `isThreadSafe="false"` είναι κακή ιδέα, δείτε το παρακάτω τμήμα κώδικα που υπολογίζει το αναγνωριστικό ενός χρήστη, χωρίς εγγύηση νηματικής ασφάλειας. Ο κώδικας αυτός δεν παρουσιάζει νηματική ασφάλεια επειδή ένα νήμα θα μπορούσε να εκτελεστεί ξανά αφού διαβάσει την `idNum` αλλά πριν την ενημερώσει, με αποτέλεσμα να έχουμε δύο χρήστες με το ίδιο αναγνωριστικό χρήστη.

```
<%! private int idNum = 0; %>
<%
String userID = "userID" + idNum;
out.println("Your ID is " + userID = ".");
idNum = idNum + 1;
%>
```

Ο κώδικας θα έπρεπε να έχει χρησιμοποιήσει ένα μπλοκ `synchronized`. Αυτή η δομή γράφεται ως

```
synchronized(κάποιοΑντικείμενο) {...}
```

και σημαίνει ότι από τη στιγμή που ένα νήμα εισέρχεται στο μπλοκ του κώδικα, κανένα άλλο νήμα δεν μπορεί να εισέρθει στο ίδιο μπλοκ (ούτε και σε κάποιο άλλο μπλοκ με την ίδια αναφορά στο αντικείμενο) μέχρι να ολοκληρώσει τη δουλειά του το πρώτο νήμα. Έτσι το προηγούμενο τμήμα κώδικα θα έπρεπε να έχει γραφτεί με τον ακόλουθο τρόπο:

```
<%! private int idNum = 0; %>
<%
synchronized(this) {
  String userID = "userID" + idNum;
  out.println("Your ID is " + userID = ".");
  idNum = idNum + 1;
}
%>
```

Υπάρχουν δύο λόγοι για τους οποίους αυτή η εκδοχή ρητού συγχρονισμού είναι ανώτερη από την αρχική εκδοχή με την προσθήκη της οδηγίας `<%@ page isThreadSafe="false" %>`.

Πρώτον, η εκδοχή ρητού συγχρονισμού πιθανότατα θα παρουσιάζει πολύ καλύτερη απόδοση αν η σελίδα προσπελάζεται συχνά. Η αιτία είναι ότι οι περισσότερες σελίδες JSP δεν περιορίζονται από τις δυνατότητες της CPU (Κεντρική μονάδα επεξεργασίας) αλλά από την είσοδο/έξοδο. Έτσι, καθώς το σύστημα περιμένει είσοδο/έξοδο (π.χ., την απόκριση από μια βάση δεδομένων, το αποτέλεσμα μιας κλήσης EJB, την έξοδο που αποστέλλεται στο χρήστη μέσω δικτύου), το σύστημα θα μπορούσε να κάνει κάτι άλλο. Επειδή οι περισσότεροι διακομιστές υλοποιούν τη διασύνδεση `SingleThreadModel` με τοποθέτηση των αιτήσεων στην ουρά και χειρισμό μίας αίτησης κάθε φορά, οι σελίδες JSP με μεγάλη κυκλοφορία μπορεί να είναι *πολύ* βραδύτερες με αυτή την προσέγγιση.

Ακόμη χειρότερα, η εκδοχή που χρησιμοποιεί την ιδιότητα `SingleThreadModel` μπορεί να μη δώσει καν τη σωστή απάντηση! Στους διακομιστές επιτρέπεται να υλοποιήσουν τη `SingleThreadModel` όχι με τοποθέτηση των αιτήσεων στην ουρά, αλλά με τη δημιουργία μιας δεξαμενής στιγμιοτύπων μικροϋπηρεσιών, υπό την προϋπόθεση ότι κανένα στιγμιότυπο δεν θα καλείται ταυτόχρονα. Έτσι, φυσικά, αναιρείται πλήρως ο σκοπός της χρήσης πεδίων για διατήρηση δεδομένων, αφού κάθε στιγμιότυπο θα έχει διαφορετικό πεδίο (μεταβλητή στιγμιοτύπου), ενώ πολλοί χρήστες θα μπορούν να πάρουν το ίδιο αναγνωριστικό χρήστη. Ούτε ο ορισμός του πεδίου `idNum` ως `static` λύνει το πρόβλημα· η αναφορά `this` θα είναι διαφορετική για κάθε στιγμιότυπο μικροϋπηρεσίας, οπότε η προστασία δεν θα έχει κανένα αποτέλεσμα.

Αυτά τα προβλήματα είναι κατά βάση ανεπίλυτα. Αποδεχθείτε το. Ξεχάστε τη διασύνδεση `SingleThreadModel` και την ιδιότητα `isThreadSafe="false"` και χρησιμοποιήστε στον κώδικά σας ρητό συγχρονισμό.

12.10 Η ιδιότητα extends

Η ιδιότητα `extends` προσδιορίζει την υπερκλάση της μικροϋπηρεσίας που θα παραχθεί από τη σελίδα JSP. Παίρνει την ακόλουθη μορφή:

```
<%@ page extends="πακέτο.class" %>
```

Αυτή η ιδιότητα κανονικά είναι δεσμευμένη για τους προγραμματιστές ή τους κατασκευαστές που υλοποιούν θεμελιώδεις αλλαγές στον τρόπο με τον οποίο λειτουργεί μία σελίδα (π.χ., για την προσθήκη λειτουργιών προσωπικού χαρακτήρα). Οι κοινοί θνητοί θα πρέπει να αποφεύγουν αυτή την ιδιότητα, εκτός και αν αναφέρονται σε κλάσεις που παρέχονται ειδικά γι' αυτόν το σκοπό από τον κατασκευαστή του διακομιστή.

12.11 Η ιδιότητα language

Κάποια στιγμή η ιδιότητα `language` προοριζόταν για τον καθορισμό της γλώσσας σεναρίου που χρησιμοποιείται, όπως παρακάτω:

```
<%@ page language="cobol" %>
```

Για την ώρα, μην ασχολείστε με αυτή την ιδιότητα αφού η `java` είναι αφενός η προεπιλεγμένη τιμή, και αφετέρου η μοναδική έγκυρη επιλογή.

12.12 Σύνταξη XML για οδηγίες

Αν γράφετε σελίδες JSP που είναι συμβατές με XML, μπορείτε να χρησιμοποιήσετε για τις οδηγίες μια εναλλακτική σύνταξη, που είναι συμβατή με την XML, υπό την προϋπόθεση ότι δεν θα αναμιγνύετε στην ίδια σελίδα σύνταξη XML και κλασική σύνταξη. Αυτές οι δομές έχουν την ακόλουθη μορφή:

```
<jsp:directive.τύποςΟδηγίας οδηγία="τιμή" />
```

Για παράδειγμα, η ισοδύναμη σύνταξη σε XML για την οδηγία

```
<%@ page import=java.util.*" %>
```

είναι

```
<jsp:directive.page import="java.util.*" />
```