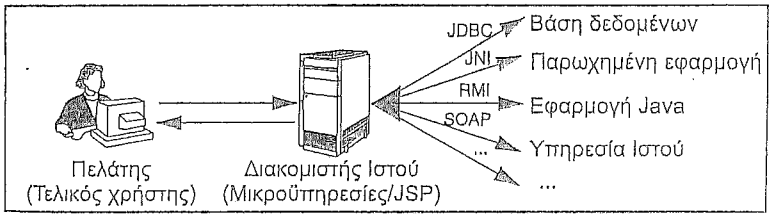
ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΜΙΚΡΟΫΠΗΡΕΣΙΩΝ

Θέματα σε αυτό το Κεφάλαιο

- Η βασική δομή των μικροϋπηρεσιών
- Μια απλή μικροϋπηρεσία που παράγει μόνο κείμενο
- Μικροϋπηρεσία που παράγει κώδικα HTML
- Μικροϋπηρεσίες και πακέτα
- Μερικές βοηθητικές κλάσεις που βοηθούν στη δόμηση κώδικα HTML. Ο κύκλος ζωής μιας μικροϋπηρεσίας
- Πώς γίνεται ο χειρισμός πολυνηματικών προβλημάτων
- Εργαλεία για την αλληλεπιδραστική επικοινωνία με τις μικροϋπηρεσίες
- Στρατηγικές αποσφαλμάτωσης μικροϋπηρεσιών

Κεφάλαιο 3

Όπως αναφέρθηκε στο Κεφάλαιο 1, οι μικροϋπηρεσίες είναι προγράμματα που εκτελούνται σε ένα διακομιστή Ιστού ή εφαρμογών, και λειτουργούν σαν "ενδιάμεσο επίπεδο" μεταξύ αφενός μιας αίτησης που προέρχεται από ένα φυλλομετρητή Ιστού ή κάποιον άλλο πελάτη HTTP, και αφετέρου των βάσεων δεδομένων ή των εφαρμογών στο διακομιστή HTTP. Η δουλειά τους είναι η εκτέλεση εργασιών όπως αυτές που φαίνονται στην Εικόνα 3-1.



Εικόνα 3-1 Ο ρόλος του ενδιάμεσου λογισμικού Ιστού.

1. Ανάγνωση των ρητών δεδομένων (explicit data) που αποστέλλονται από τον πελάτη.

   Ο τελικός χρήστης κανονικά εισάγει αυτά τα δεδομένα σε κάποια φόρμα HTML μιας ιστοσελίδας. Τα δεδομένα, όμως, θα μπορούσαν να προέρχονται και από μια μικροεφαρμογή (applet) ή ένα προσαρμοσμένο πρόγραμμα πελάτη HTTP.

2. **Ανάγνωση των έμμεσων δεδομένων αίτησης HTTP που αποστέλλονται από το φυλλομετρητή.**
   Η Εικόνα 3-1 δείχνει ένα βέλος από τον πελάτη στο διακομιστή Ιστού (το επίπεδο στο οποίο εκτελούνται οι μικροϋπηρεσίες και οι σελίδες JSP), αλλά στην πραγματικότητα υπάρχουν *δύο* είδη τέτοιων δεδομένων: τα ρητά δεδομένα που εισάγει ο τελικός χρήστης σε μια φόρμα, και οι πληροφορίες HTTP που υπάρχουν στο παρασκήνιο. Και οι δύο τύποι δεδομένων έχουν κρίσιμο χαρακτήρα για την αποτελεσματική ανάπτυξη εφαρμογών. Στις πληροφορίες HTTP περιλαμβάνονται τα μπισκότα (cookies), οι τύποι δεδομένων και οι μέθοδοι συμπίεσης που αντιλαμβάνεται ο φυλλομετρητής, κ.ο.κ.· θα εξετάσουμε αυτά τα ζητήματα στο Κεφάλαιο 5.
3. **Παραγωγή των αποτελεσμάτων.**
   Αυτή η διαδικασία μπορεί να χρειάζεται την επικοινωνία με μια βάση δεδομένων, την εκτέλεση μιας κλήσης RMI ή CORBA, την κλήση μιας υπηρεσίας Ιστού, ή τον άμεσο υπολογισμό της απάντησης. Τα πραγματικά σας δεδομένα μπορεί να βρίσκονται σε μια σχεσιακή βάση δεδομένων. Πολύ ωραία. Πιθανόν όμως η βάση δεδομένων να μην καταλαβαίνει το πρωτόκολλο HTTP ή να μην επιστρέφει αποτελέσματα σε κώδικα HTML, οπότε ο φυλλομετρητής Ιστού δεν μπορεί να επικοινωνήσει άμεσα με τη βάση δεδομένων. Το ίδιο ισχύει και για πολλές άλλες εφαρμογές. Χρειάζεστε το ενδιάμεσο επίπεδο του Ιστού για να πάρετε τα εισερχόμενα δεδομένα από το ρεύμα HTTP, να επικοινωνήσετε με την εφαρμογή, και να ενσωματώσετε τα αποτελέσματα στο εσωτερικό ενός εγγράφου.
4. **Αποστολή των ρητών δεδομένων (δηλαδή του εγγράφου) στον πελάτη.**
   Αυτό το έγγραφο μπορεί να αποσταλεί σε μια ποικιλία μορφών, όπως μορφή κειμένου (HTML ή XML), δυαδική μορφή (εικόνες GIF), αρχείο Excel, ή ακόμα και σε συμπιεσμένη μορφή, όπως gzip, που έχει διαστρωματωθεί επάνω από κάποια υποκείμενη μορφή.
5. **Αποστολή έμμεσων δεδομένων απάντησης HTTP.**
   Η Εικόνα 3-1 παρουσιάζει ένα βέλος από το επίπεδο ενδιάμεσου λογισμικού Ιστού (τη μικροϋπηρεσία ή τη σελίδα JSP) προς τον πελάτη, αλλά στην πραγματικότητα, υπάρχουν *δύο* είδη δεδομένων που αποστέλλονται: από τη μία πλευρά το ίδιο το έγγραφο, και από την άλλη οι πληροφορίες HTTP που υπάρχουν στο παρασκήνιο. Και οι δύο τύποι δεδομένων έχουν κρίσιμο χαρακτήρα προκειμένου να έχουμε αποτελεσματική ανάπτυξη εφαρμογών. Η αποστολή δεδομένων απάντησης HTTP περιλαμβάνει την ενημέρωση του φυλλομετρητή ή του όποιου άλλου πελάτη για τον τύπο του εγγράφου που επιστρέφεται (π.χ. HTML), τον ορισμό παραμέτρων για τα μπισκότα και την κρυφή μνήμη προσωρινής αποθήκευσης, και άλλες τέτοιες εργασίες. Αυτές οι εργασίες εξετάζονται στα Κεφάλαια 6 μέχρι 8.

Θεωρητικά, οι μικροϋπηρεσίες δεν περιορίζονται σε διακομιστές Ιστού ή εφαρμογών που χειρίζονται αιτήσεις HTTP, αλλά μπορούν να χρησιμοποιηθούν και για άλλους τύπους διακομιστών. Για παράδειγμα, οι μικροϋπηρεσίες μπορούν να ενσωματωθούν σε διακομιστές FTP ή διακομιστές ηλεκτρονικού ταχυδρομείου, με στόχο την επέκταση της λειτουργικότητάς τους. Στην πράξη, όμως, αυτή η χρήση των μικροϋπηρεσιών δεν είναι ακόμα διαδεδομένη, και έτσι θα εξετάσουμε μόνο μικροϋπηρεσίες HTTP.

3.1 Βασική δομή μικροϋπηρεσιών

Η Λίστα 3.1 παρουσιάζει μια βασική μικροϋπηρεσία που χειρίζεται αιτήσεις GET. Οι αιτήσεις GET, για όσους δεν είναι εξοικειωμένοι με το πρωτόκολλο HTTP, αποτελούν το συνηθισμένο τύπο αιτήσεων του φυλλομετρητή για ιστοσελίδες. Ο φυλλομετρητής παράγει αυτή την αίτηση όταν ο χρήστης γράφει ένα URL στη γραμμή διευθύνσεων, πατά σε ένα σύνδεσμο σε μια ιστοσελίδα, ή υποβάλλει κάποια φόρμα HTML η οποία είτε δεν καθορίζει μέθοδο, ή καθορίζει ότι METHOD="GET". Οι μικροϋπηρεσίες μπορούν επίσης εύκολα να χειρίζονται αιτήσεις POST, οι οποίες παράγονται όταν κάποιος υποβάλλει μια φόρμα HTML που καθορίζει ότι METHOD="POST". Λεπτομέρειες σχετικά με τη χρήση των φορμών HTML και τις διαφορές μεταξύ των μεθόδων GET και POST θα βρείτε στο Κεφάλαιο 19 (Δημιουργία και επεξεργασία φορμών HTML).

Λίστα 3.1 ServletTemplate.java

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

public class ServletTemplate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    // Χρήση του "request" για ανάγνωση εισερχόμενων κεφαλίδων HTTP
    // (π.χ., μπισκότα) και δεδομένων ερωτημάτων από φόρμες HTML.

    // Χρήση του "response" για καθορισμό του κωδικού κατάστασης και
    // των κεφαλίδων απάντησης HTTP (π.χ., τύπος περιεχομένου, μπισκότα).

    PrintWriter out = response.getWriter();
    // Χρήση του "out" για την αποστολή περιεχομένου στο φυλλομετρητή
  }
}
```

Οι μικροϋπηρεσίες συνήθως επεκτείνουν (extend) τη μέθοδο HttpServlet και υποσκελίζουν (override) τις doGet και doPost, ανάλογα με το αν τα δεδομένα στέλνονται με τη μέθοδο GET ή τη μέθοδο POST. Αν θέλετε η μικροϋπηρεσία σας να εκτελεί την ίδια ενέργεια τόσο για την αίτηση GET όσο και για την αίτηση POST, μπορεί απλώς η doGet να καλεί την doPost, ή το αντίστροφο.

Και η doGet και η doPost δέχονται δύο ορίσματα (arguments): τα HttpServletRequest και HttpServletResponse. Το όρισμα HttpServletRequest σας επιτρέπει να παραλάβετε

όλα τα *εισερχόμενα* δεδομένα· η κλάση αυτή διαθέτει μεθόδους με τις οποίες μπορείτε να βρείτε πληροφορίες όπως τα δεδομένα φόρμας (το "ερώτημα"), οι κεφαλίδες αίτησης HTTP, και το όνομα του υπολογιστή υπηρεσίας του πελάτη. Το όρισμα `HttpServletResponse` σας επιτρέπει να καθορίσετε *εξερχόμενες* πληροφορίες, όπως κωδικούς κατάστασης HTTP (200, 404, κ.λπ.) και κεφαλίδες απάντησης (`Content-Type`, `Set-Cookie`, κ.λπ.). Το πιο σημαντικό, όμως, είναι ότι το όρισμα `HttpServletResponse` σας επιτρέπει να λάβετε ένα αντικείμενο `PrintWriter`, το οποίο μπορείτε να χρησιμοποιήσετε για να στείλετε το περιεχόμενο του εγγράφου πίσω στον πελάτη. Για τις απλές μικροϋπηρεσίες, η περισσότερη δουλειά είναι η σύνταξη εντολών `println` για την παραγωγή της επιθυμητής σελίδας. Θα εξετάσουμε σε επόμενα κεφάλαια τα δεδομένα φορμών, τις κεφαλίδες αίτησης HTTP, τις απαντήσεις HTTP, και τα μπισκότα.

Επειδή οι μέθοδοι `doGet` και `doPost` προκαλούν (throw) δύο εξαιρέσεις (exceptions) — τις `ServletException` και `IOException` — θα πρέπει να τις συμπεριλάβετε στη δήλωση (declaration) της μεθόδου. Τέλος, θα πρέπει να εισαγάγετε τις κλάσεις των πακέτων `java.io` (για την `PrintWriter`, κ.λπ.), `javax.servlet` (για την `HttpServlet`, κ.λπ.) και `javax.servlet.http` (για τις `HttpServletRequest` και `HttpServletResponse`).

Ωστόσο, δεν είναι απαραίτητο να απομνημονεύσετε την υπογραφή της μεθόδου (method signature) και τις εντολές εισαγωγής. Αντίθετα, μπορείτε απλώς να κατεβάσετε το παραπάνω πρότυπο από την αρχειοθήκη πηγαίου κώδικα στη διεύθυνση http://www.coreservlets.com/ και να το χρησιμοποιείτε ως σημείο εκκίνησης για τις δικές σας μικροϋπηρεσίες.

3.2 Μικροϋπηρεσία που παράγει μόνο κείμενο

Η Λίστα 3.2 δείχνει μια απλή μικροϋπηρεσία που έχει ως έξοδο μόνο κείμενο. Η έξοδος αυτή φαίνεται στην Εικόνα 3-2. Πριν προχωρήσουμε, αξίζει τον κόπο να αφιερώσουμε λίγο χρόνο για να κάνουμε μια ανασκόπηση της διαδικασίας εγκατάστασης, μεταγλώττισης, και εκτέλεσης μιας απλής μικροϋπηρεσίας. Για μια πιο λεπτομερή περιγραφή της διαδικασίας δείτε το Κεφάλαιο 2 (Εγκατάσταση και διευθέτηση διακομιστή).

Πρώτον, φροντίστε να επαληθεύσετε τα βασικά:

- Ότι ο διακομιστής έχει εγκατασταθεί σωστά, όπως περιγράφεται στην Ενότητα 2.3 (Διευθέτηση διακομιστή)
- Ότι η μεταβλητή ανάπτυξης `CLASSPATH` περιλαμβάνει τις απαραίτητες τρεις καταχωρίσεις (το αρχείο μικροϋπηρεσιών JAR, το ανώτατο επίπεδο του καταλόγου ανάπτυξης, και τον κατάλογο "."), όπως περιγράφεται στην Ενότητα 2.7 (Διαμόρφωση του δικού σας περιβάλλοντος ανάπτυξης).
- Ότι όλες οι περιπτώσεις ελέγχου της Ενότητας 2.8 (Έλεγχος της εγκατάστασής σας) εκτελούνται με επιτυχία.

Δεύτερον, πληκτρολογήστε `"javac HelloWorld.java"` ή ζητήστε από το περιβάλλον ανάπτυξης που χρησιμοποιείτε να μεταγλωττίσει τη μικροϋπηρεσία — για παράδειγμα, επιλέγοντας τη διαταγή Build (Δόμηση) του IDE ή επιλέγοντας τη διαταγή Compile (Μεταγλώττιση) από του μενού JDE του emacs. Το βήμα αυτό θα μεταγλωττίσει τη μικροϋπηρεσία σας και θα δημιουργήσει το αρχείο HelloWorld.class.

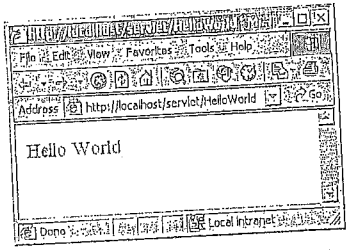
Τρίτον, μεταφέρετε το αρχείο HelloWorld.class στον κατάλογο τον οποίο χρησιμοποιεί ο διακομιστής σας για να αποθηκεύει μικροϋπηρεσίες που βρίσκονται στην προεπιλεγμένη εφαρμογή Ιστού. Η ακριβής τοποθεσία διαφέρει από διακομιστή σε διακομιστή, αλλά κατά κανόνα έχει τη μορφή *κατ_εγκατάστασης*/.../WEB-INF/classes (δείτε την Ενότητα 2.10 για περισσότερες λεπτομέρειες). Για τον Tomcat ο κατάλογος είναι ο *κατ_εγκατάστασης*/webapps/ROOT/WEB-INF/classes, για το JRun είναι ο *κατ_εγκατάστασης*/servers/default/default-ear/default-war/WEB-INF/classes, και για το Resin είναι ο *κατ_εγκατάστασης*/doc/WEB-INF/classes. Εναλλακτικά, μπορείτε να χρησιμοποιήσετε κάποια από τις τεχνικές που περιγράφονται στην Ενότητα 2.9 (Δημιουργία μιας απλοποιημένης μεθόδου εκδίπλωσης) για την αυτόματη τοποθέτηση των αρχείων κλάσεων στην κατάλληλη τοποθεσία.

Τέλος, καλέστε τη μικροϋπηρεσία σας. Αυτό το τελευταίο βήμα περιλαμβάνει τη χρήση είτε του προεπιλεγμένου URL http://*υπολογιστήςΥπηρεσίας*/servlet/*ΌνομαΜικροϋπηρεσίας*, είτε τη χρήση ενός προσαρμοσμένου URL που έχει οριστεί στο αρχείο web.xml, όπως περιγράψαμε στην Ενότητα 2.12 (Εφαρμογές Ιστού: Μια προεπισκόπηση). Στην αρχική φάση ανάπτυξης, σχεδόν σίγουρα θα βρείτε βολική τη χρήση του προεπιλεγμένου URL έτσι ώστε να μη χρειάζεται να διορθώνετε το αρχείο web.xml κάθε φορά που δοκιμάζετε μια νέα μικροϋπηρεσία. Όταν όμως εκδιπλώνετε πραγματικές εφαρμογές, σχεδόν πάντοτε θα απενεργοποιείτε το προεπιλεγμένο URL και θα αποδίδετε ρητά URL μέσω του αρχείου web.xml (δείτε την Ενότητα 2.11, "Εφαρμογές Ιστού: Μια προεπισκόπηση"). Στην πραγματικότητα οι διακομιστές δεν είναι απαραίτητο να υποστηρίζουν προεπιλεγμένες διευθύνσεις URL, και ορισμένοι (όπως ο BEA WebLogic) δεν τις υποστηρίζουν.

Η Εικόνα 3-2 δείχνει τη μικροϋπηρεσία όταν την προσπελάζουμε μέσω του προεπιλεγμένου URL, με το διακομιστή να εκτελείται στο τοπικό μηχάνημα.

Λίστα 3.2 HelloWorld.java

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

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```



Εικόνα 3-2 Αποτέλεσμα του http://localhost/servlet/HelloWorld.

3.3 Μικροϋπηρεσία που παράγει κώδικα HTML

Οι περισσότερες μικροϋπηρεσίες παράγουν κώδικα HTML, και όχι απλό κείμενο όπως συνέβαινε στο προηγούμενο παράδειγμα. Για την παραγωγή κώδικα HTML θα πρέπει να προσθέσετε τρία βήματα στην παραπάνω διαδικασία:

1. Ενημέρωση του διακομιστή ότι του στέλνετε κώδικα HTML.
2. Τροποποίηση των εντολών `println` για τη δόμηση μιας έγκυρης ιστοσελίδας.
3. Έλεγχος του κώδικα HTML με ένα πρόγραμμα επικύρωσης σύνταξης.

Για να επιτύχετε το πρώτο βήμα, θα πρέπει να ορίσετε την κεφαλίδα απάντησης HTTP `Content-Type` σε `text/html`. Γενικά, οι κεφαλίδες ορίζονται με τη μέθοδο `setHeader` της κλάσης `HttpServletResponse`, αλλά ο ορισμός του τύπου περιεχομένου (content type) είναι τόσο συνηθισμένη εργασία που υπάρχει μια ειδική μέθοδος, η `setContentType`, για το σκοπό αυτόν. Για να προσδιορίσετε ότι θα στείλετε κείμενο HTML θα πρέπει να ορίσετε τύπο `text/html`, οπότε ο κώδικας θα μοιάζει με τον ακόλουθο:

```
response.setContentType("text/html");
```

Αν και η HTML είναι το πιο συνηθισμένο είδος εγγράφου που δημιουργούν οι μικροϋπηρεσίες, δεν είναι σπάνιο οι μικροϋπηρεσίες να δημιουργούν και άλλους τύπους εγγράφων. Για παράδειγμα, είναι αρκετά συνηθισμένη η χρήση μικροϋπηρεσιών για την παραγωγή φύλλων εργασίας Excel (τύπος περιεχομένου `application/vnd.ms-excel` — δείτε την Ενότητα 7.3), εικόνες JPEG (τύπος περιεχομένου `image/jpeg` — δείτε την Ενότητα 7.5) και έγγραφα XML (τύπος περιεχομένου `text/xml`). Επιπλέον, πολύ σπάνια θα χρησιμοποιείτε μικροϋπηρεσίες για την παραγωγή σελίδων HTML που έχουν σχετικά σταθερή μορφή (δηλαδή σελίδων που η διάταξή τους αλλάζει πολύ λίγο σε κάθε αίτηση)· σε αυτή την περίπτωση είναι πιο χρήσιμες οι σελίδες JSP. Οι σελίδες JSP περιγράφονται στο Μέρος ΙΙ του βιβλίου (με αρχή στο Κεφάλαιο 10).

Αν δεν είστε εξοικειωμένοι με τις κεφαλίδες απάντησης HTTP, μην ανησυχείτε· θα τις περιγράψουμε στο Κεφάλαιο 7. Ωστόσο, θα πρέπει να σημειώσουμε ότι χρειάζεται να ορίσετε τις κεφαλίδες απάντησης *πριν* πραγματικά επιστρέψετε οποιοδήποτε περιεχόμενο με την εντολή `PrintWriter`. Αυτό συμβαίνει επειδή η απάντηση HTTP αποτελείται από τη γραμμή κατάστα-

σης, μία ή περισσότερες κεφαλίδες, μία κενή γραμμή, και το πραγματικό έγγραφο, *με αυτή τη σειρά*. Οι κεφαλίδες μπορούν να εμφανιστούν με οποιαδήποτε σειρά, και οι μικροϋπηρεσίες αποθηκεύουν προσωρινά τις κεφαλίδες και τις στέλνουν όλες μαζί, οπότε είναι έγκυρο να ορίσετε τον κωδικό κατάστασης (τμήμα της γραμμής που επιστρέφεται) ακόμα και μετά τον ορισμό των κεφαλίδων. Δεν είναι υποχρεωτική, όμως, η προσωρινή αποθήκευση του ίδιου του εγγράφου, αφού οι χρήστες μπορεί να θέλουν να δουν ένα τμήμα των αποτελεσμάτων όταν οι σελίδες έχουν μεγάλο μέγεθος. Οι μηχανές μικροϋπηρεσιών επιτρέπεται να αποθηκεύουν προσωρινά ένα τμήμα της εξόδου, αλλά το μέγεθος αυτού του χώρου προσωρινής αποθήκευσης δεν είναι καθορισμένο. Μπορείτε να χρησιμοποιήσετε τη μέθοδος `getBufferSize` της κλάσης `HttpServletResponse` για να προσδιορίσετε αυτό το μέγεθος, ή μπορείτε να χρησιμοποιήσετε τη μέθοδο `setBufferSize` για να το καθορίσετε. Μπορείτε να ορίζετε τις κεφαλίδες μέχρι να γεμίσει ο χώρος προσωρινής αποθήκευσης και να αποσταλεί πραγματικά στον πελάτη. Αν δεν είστε σίγουροι ότι ο χώρος προσωρινής αποθήκευσης έχει αποσταλεί, μπορείτε να χρησιμοποιήσετε τη μέθοδο `isCommitted` για να το ελέγξετε. Παρόλα αυτά, η καλύτερη προσέγγιση είναι να τοποθετείτε απλώς τη γραμμή `setContentType` πριν από τις γραμμές που χρησιμοποιούν την εντολή `PrintWriter`.

Προειδοποίηση

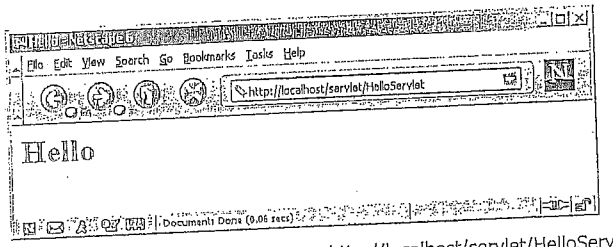
*Πρέπει να ορίσετε τον τύπο περιεχομένου **πριν** από τη μετάδοση του πραγματικού εγγράφου.*

Το δεύτερο βήμα στη συγγραφή μιας μικροϋπηρεσίας που δημιουργεί ένα έγγραφο HTML είναι το να έχουν οι εντολές `println` έξοδο HTML, και όχι απλό κείμενο. Η Λίστα 3.3 παρουσιάζει το αρχείο HelloServlet.java, τη μικροϋπηρεσία που χρησιμοποιήθηκε στην Ενότητα 2.8 για την επαλήθευση της σωστής λειτουργίας του διακομιστή. Όπως δείχνει η Εικόνα 3-3, ο φυλλομετρητής μορφοποιεί το αποτέλεσμα ως HTML, και όχι ως απλό κείμενο.

Λίστα 3.3 HelloServlet.java

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

/** Απλή μικροϋπηρεσία για τον έλεγχο του διακομιστή. */

public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
```

```
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>Hello</H1>\n" +
                "</BODY></HTML>");
  }
}
```



Εικόνα 3.3 Το αποτέλεσμα της http://localhost/servlet/HelloServlet.

Το τελευταίο βήμα είναι να ελέγξετε ότι ο κώδικας HTML που δημιουργήσατε δεν έχει συντακτικά σφάλματα τα οποία θα μπορούσαν να προκαλέσουν απροσδόκητα αποτελέσματα στους διάφορους φυλλομετρητές. Για περισσότερες πληροφορίες σχετικά με τα προγράμματα επικύρωσης HTML δείτε την Ενότητα 3.5 (Απλές βοηθητικές κλάσεις δόμησης κώδικα HTML).

3.4 Πακετάρισμα μικροϋπηρεσιών

Σε ένα περιβάλλον παραγωγής, πολλοί προγραμματιστές μπορεί να αναπτύσσουν μικροϋπηρεσίες για τον ίδιο διακομιστή. Έτσι η τοποθέτηση όλων των μικροϋπηρεσιών στον ίδιο κατάλογο έχει αποτέλεσμα μια ογκώδη και δύσκολη στη διαχείριση συλλογή κλάσεων, καθώς και τον κίνδυνο της διένεξης ονομάτων στην περίπτωση που δύο προγραμματιστές επιλέξουν κατά λάθος το ίδιο όνομα για κάποια μικροϋπηρεσία ή βοηθητική κλάση. Οι εφαρμογές Ιστού, όμως, (δείτε την Ενότητα 2.11) βοηθούν στην αντιμετώπιση αυτού του προβλήματος διαχωρίζοντας τα διάφορα στοιχεία σε ξεχωριστούς καταλόγους, όπου κάθε κατάλογος έχει το δικό του σύνολο μικροϋπηρεσιών, βοηθητικών κλάσεων, σελίδων JSP, και αρχείων HTML. Ωστόσο, επειδή η εφαρμογή Ιστού μπορεί να είναι πολύ μεγάλη, εξακολουθείτε να χρειάζεστε την τυπική λύση της Java για την αποφυγή διενέξεων ονομάτων: τα πακέτα (packages). Εκτός από αυτό, όπως θα δείτε στη συνέχεια, οι προσαρμοσμένες σελίδες JSP θα πρέπει *πάντοτε* να είναι σε πακέτα. Έτσι είναι καλό να αποκτήσετε αυτή τη συνήθεια από νωρίς.

Για να τοποθετήσετε τις μικροϋπηρεσίες σας σε πακέτα, θα πρέπει να εκτελέσετε τα επόμενα δύο επιπλέον βήματα:

1. **Τοποθέτηση των αρχείων σε έναν υποκατάλογο που ταυτίζεται με το όνομα του πακέτου.** Για παράδειγμα, θα χρησιμοποιήσουμε το πακέτο `coreservlets` για τις περισσότερες μικροϋπηρεσίες αυτού του βιβλίου. Έτσι τα αρχεία κλάσεων θα πρέπει να τοποθετηθούν σε έναν υποκατάλογο με το όνομα coreservlets. Να θυμάστε ότι υπάρχει διάκριση μεταξύ πεζών και κεφαλαίων γραμμάτων τόσο για τα ονόματα των πακέτων, όσο και για τα ονόματα των καταλόγων — ανεξάρτητα από το λειτουργικό σύστημα που χρησιμοποιείτε.
2. **Εισαγωγή μιας εντολής πακέτου στο αρχείο της κλάσης.** Για παράδειγμα, αν μία κλάση βρίσκεται στο πακέτο με το όνομα somePackage, τότε η κλάση θα πρέπει να βρίσκεται στον κατάλογο somePackage και η *πρώτη* μη σχολιασμένη γραμμή του αρχείου θα πρέπει να είναι η εξής

   ```
   package somePackage;
   ```

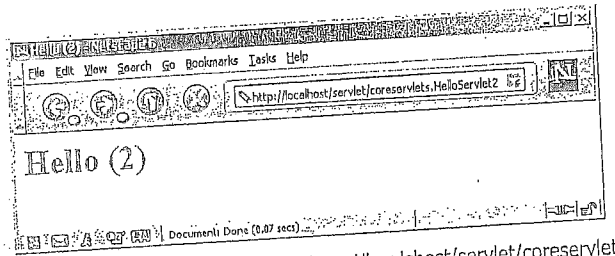
   Για παράδειγμα, η Λίστα 3.4 παρουσιάζει μια παραλλαγή της κλάσης `HelloServlet` που βρίσκεται στο πακέτο `coreservlets`, άρα βρίσκεται στον κατάλογο coreservlets. Όπως περιγράφεται στην Ενότητα 2.8 (Έλεγχος της εγκατάστασής σας), το αρχείο της κλάσης θα πρέπει να τοποθετηθεί στον κατάλογο *κατ_εγκατάστασης*/webapps/ROOT/WEB-INF/classes/coreservlets για τον Tomcat, στον κατάλογο *κατ_εγκατάστασης*/servers/default/default-ear/default-war/WEB-INF/classes/coreservlets για το JRun, ή στον κατάλογο *κατ_εγκατάστασης*/doc/WEB-INF/classes/coreservlets για το Resin. Άλλοι διακομιστές έχουν ανάλογες θέσεις εγκατάστασης.
   Η Εικόνα 3-4 παρουσιάζει τη μικροϋπηρεσία όταν την προσπελάζουμε μέσω του προεπιλεγμένου URL.

Λίστα 3.4 coreservlets/HellopServlet2.java

```
package coreservlets;

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

/** Απλή μικροϋπηρεσία για τον έλεγχο της χρήσης πακέτων. */

public class HelloServlet2 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello (2)</TITLE></HEAD>\n" +
```

Λίστα 3.4 coreservlets/HellopServlet2.java *(συνέχεια)*

```
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1>Hello (2)</H1>\n" +
            "</BODY></HTML>");
  }
}
```



Εικόνα 3-4 Αποτέλεσμα της http://localshost/servlet/coreservlets.HelloServlet2.

3.5 Απλές βοηθητικές κλάσεις δόμησης κώδικα HTML

Όπως πιθανόν να γνωρίζετε ήδη, το έγγραφο HTML είναι δομημένο ως εξής:

```
<!DOCTYPE...>
<HTML>
<HEAD><TITLE>...<</TITLE>...</HEAD>
<BODY...>...</BODY>
  </HTML>
```

Όταν χρησιμοποιείτε μικροϋπηρεσίες για να δομήσετε τη σελίδα HTML, μπορεί να μπείτε στον πειρασμό να παραλείψετε κάποιο τμήμα αυτής της δομής, ιδιαίτερα τη γραμμή DOCTYPE, αφού σχεδόν όλοι οι μεγάλοι φυλλομετρητές δεν τη λαμβάνουν υπόψη τους παρόλο που οι προδιαγραφές της HTML το απαιτούν. Το πλεονέκτημα της γραμμής DOCTYPE είναι ότι ενημερώνει τα προγράμματα επικύρωσης HTML (HTML validators) για την έκδοση της HTML που χρησιμοποιείτε, έτσι ώστε να γνωρίζουν σε σύγκριση με ποιες προδιαγραφές θα κάνουν τον έλεγχο του εγγράφου σας. Αυτά τα προγράμματα επικύρωσης είναι πολύτιμες υπηρεσίες αποσφαλμάτωσης, που σας βοηθούν να εντοπίσετε συντακτικά σφάλματα HTML τα οποία πιθανόν ο δικός σας φυλλομετρητής να τα αντιμετωπίζει σωστά, όμως άλλοι φυλλομετρητές ίσως έχουν πρόβλημα στην εμφάνισή τους.

Τα δύο πιο δημοφιλή προγράμματα ηλεκτρονικής επικύρωσης προέρχονται από τη World Wide Web Consortium (http://validator/w3/org) και από τη Web Design Group (http://www.htmlhelp.com/tools/validator/). Σας επιτρέπουν να υποβάλετε ένα URL, κατόπιν ανακτούν τη σελίδα, ελέγχουν τη σύνταξη σε σχέση με τις τυπικές προδιαγραφές της HTML, και σας αναφέρουν τα σφάλματα. Επειδή για τον πελάτη η μικροϋπηρεσία που παράγει κώδικα HTML μοιάζει με κανονική ιστοσελίδα, μπορεί και αυτή να επικυρωθεί με κανονικό τρόπο εκτός και αν χρειάζεται δεδομένα POST για να επιστρέψει τα αποτελέσματά της. Επειδή τα δεδομένα GET προσαρτώνται στο URL, μπορείτε να αποστείλετε στα προγράμματα επικύρωσης ένα URL που περιέχει δεδομένα GET. Αν η μικροϋπηρεσία σας είναι διαθέσιμη μόνο στο εσωτερικό του εταιρικού τείχους προστασίας, μπορείτε απλώς να την εκτελέσετε, να αποθηκεύσετε τη σελίδα HTML στο δίσκο, και να διαλέξετε την επιλογή File Upload (Ανέβασμα αρχείου) του προγράμματος επικύρωσης.

Η προσέγγιση του βιβλίου

Χρησιμοποιήστε κάποιο πρόγραμμα επικύρωσης HTML για να ελέγξετε τη σύνταξη των σελίδων που παράγουν οι μικροϋπηρεσίες σας.

Ομολογουμένως, μερικές φορές είναι κάπως άβολη η παραγωγή κώδικα HTML με εντολές println, ιδιαίτερα στην περίπτωση των ανιαρών μακροσκελών γραμμών όπως η δήλωση DOCTYPE. Κάποιοι αντιμετωπίζουν αυτό το πρόβλημα με τη δημιουργία βοηθητικών κλάσεων που παράγουν μακροσκελείς εντολές HTML και μετά τις χρησιμοποιούν στις μικροϋπηρεσίες τους. Αντιμετωπίζουμε με σκεπτικισμό τη χρησιμότητα μιας τέτοιας εκτεταμένης βιβλιοθήκης. Πρώτον και σημαντικότερο, ο μπελάς της παραγωγής κώδικα HTML με προγραμματιστικό τρόπο είναι το κύριο πρόβλημα με το οποίο καταπιάνονται οι σελίδες JSP (δείτε το Κεφάλαιο 10, "Επισκόπηση της τεχνολογίας JSP"). Δεύτερον, οι ρουτίνες παραγωγής HTML μπορεί να είναι δύσχρηστες και να μην υποστηρίζουν το πλήρες εύρος των ιδιοτήτων HTML (τις ιδιότητες CLASS και ID των φύλλων στυλ, τους χειριστές συμβάντων JavaScript, τα χρώματα παρασκηνίου στα κελιά πινάκων, κ.ο.κ.).

Παρά την αμφισβητήσιμη αξία μιας πλήρους βιβλιοθήκης παραγωγής κώδικα HTML, αν βρείτε ότι επαναλαμβάνετε τις ίδιες δομές πολλές φορές, μπορείτε να φτιάξετε μία απλή βοηθητική κλάση η οποία θα απλοποιεί αυτές τις δομές. Έτσι κι αλλιώς δουλεύετε με τη γλώσσα προγραμματισμού Java· δεν πρέπει να ξεχνάτε την κύρια αρχή του αντικειμενοστρεφούς προγραμματισμού, που είναι η επαναχρησιμοποίηση (αντί για την επανάληψη) κώδικα. Η επανάληψη παρόμοιου ή σχεδόν παρόμοιου κώδικα σημαίνει ότι θα πρέπει να αλλάξετε τον κώδικα σε πολλές διαφορετικές θέσεις όταν αναπόφευκτα αλλάξετε την προσέγγισή σας.

Για τις τυπικές μικροϋπηρεσίες, δύο τμήματα των ιστοσελίδων (τα DOCTYPE και HEAD) είναι απίθανο να αλλάξουν, και έτσι μπορούμε να ωφεληθούμε από την ενσωμάτωσή τους σε ένα απλό βοηθητικό αρχείο. Αυτά τα δύο μέρη παρουσιάζονται στη Λίστα 3.5, ενώ η Λίστα 3.6 παρουσιάζει μια παραλλαγή της κλάσης HelloServlet που κάνει χρήση αυτού του βοηθήματος. Στην πορεία θα προσθέσουμε μερικές ακόμα βοηθητικές κλάσεις.

Λίστα 3.5 coreservlets/ServerUtilities.java

```
package coreservlets;

import javax.servlet.*;
import javax.servlet.http.*;

/** Μερικές βοηθητικές κλάσεις για εξοικονόμηση χρόνου.
 *  Προσέξτε ότι οι περισσότερες είναι στατικές μέθοδοι. */
```

Λίστα 3.5 coreservlets/ServerUtilities.java *(συνέχεια)*

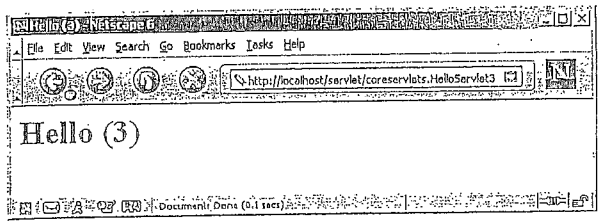
```
public class ServletUtilities {
  public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">";
  public static String headWithTitle(String title) {
    return(DOCTYPE + "\n" +
           "<HTML>\n" +
           "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
  }
...
}
```

Λίστα 3.6 coreservlets/HelloServlet3.java

```
package coreservlets;

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

/** Απλή μικροϋπηρεσία για τον έλεγχο της χρήσης πακέτων και βοηθητικών
 *  κλάσεων από το ίδιο πακέτο.
 */

public class HelloServlet3 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Hello (3)";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>" + title + "</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Αφού μεταγλωττίσετε το αρχείο HelloServlet3.java (που θα έχει αποτέλεσμα την αυτόματη μεταγλώττιση του αρχείου ServerUtilities.java), θα πρέπει να μεταφέρετε τα δύο αρχεία κλάσεων στον υποκατάλογο coreservlets της προεπιλεγμένης θέσης εκδίπλωσης εφαρμογών του διακομιστή (.../WEB-INF/classes· για λεπτομέρειες δείτε την Ενότητα 2.8). Αν παρουσιαστεί το μήνυμα σφάλματος "Unresolved symbol" (μη επιλύσιμο σύμβολο) κατά τη μεταγλώττιση του αρχείου HelloServlet3.java, ελέγξτε τις ρυθμίσεις της μεταβλητής CLASSPATH, όπως περιγράψαμε στην Ενότητα 2.7 (Διευθέτηση του δικού σας περιβάλλοντος ανάπτυξης), και ιδιαίτερα το τμήμα σχετικά με τη συμπερίληψη του ανώτατου επιπέδου του καταλόγου ανάπτυξης στη μεταβλητή

CLASSPATH. Η Εικόνα 3.5 δείχνει το αποτέλεσμα όταν καλούμε τη μικροϋπηρεσία με το προεπιλεγμένο URL.



Εικόνα 3.5 Το αποτέλεσμα της http://localhost/servlet/coreservlets.HelloServlet3.

3.6 Ο κύκλος ζωής μιας μικροϋπηρεσίας

Στην ενότητα 1.4 (Πλεονεκτήματα των μικροϋπηρεσιών σε σχέση με την παραδοσιακή CGI) αναφερθήκαμε στο γεγονός ότι δημιουργείται ένα μόνο στιγμιότυπο (instance) της μικροϋπηρεσίας, ενώ κάθε νέα αίτηση του χρήστη έχει αποτέλεσμα τη δημιουργία ενός νέου νήματος (thread) το οποίο εκχωρείται είτε στην doGet είτε στην doPost, ανάλογα με την περίσταση. Εδώ θα δώσουμε πιο συγκεκριμένες πληροφορίες σχετικά με τη δημιουργία και την καταστροφή των μικροϋπηρεσιών, καθώς και για το πώς και πότε καλούνται οι διάφορες μέθοδοι. Καταρχήν θα παρουσιάσουμε μια σύνοψη των θεμάτων, και στις υποενότητες που ακολουθούν θα τα αναπτύξουμε πιο διεξοδικά.

Όταν δημιουργείται για πρώτη φορά μια μικροϋπηρεσία καλείται η μέθοδός της `init`· έτσι η `init` είναι η θέση όπου θα τοποθετήσετε κώδικα διευθέτησης ο οποίος θα εκτελεστεί μία μόνο φορά. Μετά από αυτό, η κάθε αίτηση χρήστη δημιουργεί ένα νήμα το οποίο καλεί τη μέθοδο `service` του στιγμιοτύπου που έχει δημιουργηθεί προηγουμένως. Αν υπάρχουν πολλές ταυτόχρονες αιτήσεις, αυτές κανονικά δημιουργούν πολλά νήματα που καλούν ταυτόχρονα τη μέθοδο `service`, αν και η μικροϋπηρεσία σας μπορεί να υλοποιήσει μια ειδική διασύνδεση (`SingleThreadMode`) η οποία ορίζει ότι επιτρέπεται να εκτελείται κάθε φορά ένα μόνο νήμα. Κατόπιν η μέθοδος `service` καλεί τις `doGet`, `doPost`, ή κάποια άλλη μέθοδο `doXxx` ανάλογα με τον τύπο της αίτησης HTTP που έχει ληφθεί. Τέλος, αν ο διακομιστής αποφασίσει να απομακρύνει από τη μνήμη (unload) μια μικροϋπηρεσία, καλεί τη μέθοδο `destroy` της μικροϋπηρεσίας.

Η μέθοδος service

Κάθε φορά που ο διακομιστής λαμβάνει μια αίτηση για κάποια μικροϋπηρεσία, ξεκινά ένα νέο νήμα και καλεί τη μέθοδο `service`. Η μέθοδος `service` ελέγχει τον τύπο της αίτησης HTTP (`GET`, `POST`, `PUT`, `DELETE`, κ.λπ.) και καλεί την `doGet`, `doPost`, `doPut`, `doDelete`, κ.λπ., ανάλογα με την περίσταση. Η αίτηση `GET` προκύπτει από μια κανονική αίτηση URL ή από μια φόρ-

μια HTML στην οποία δεν έχει καθοριστεί η παράμετρος METHOD. Η αίτηση POST προκύπτει από μια φόρμα HTML όπου στην παράμετρο METHOD έχει οριστεί η τιμή POST. Οι άλλες αιτήσεις HTTP παράγονται μόνο από προσαρμοσμένους πελάτες. Αν δεν είστε εξοικειωμένοι με τις φόρμες HTML, δείτε το Κεφάλαιο 19 (Δημιουργία και επεξεργασία φορμών HTML).

Τώρα, αν έχετε μία μικροϋπηρεσία που πρέπει να χειριστεί και την αίτηση POST και την αίτηση GET με το ίδιο τρόπο, μπορεί να μπείτε στον πειρασμό να υποσκελίσετε (override) άμεσα τη μέθοδο service, αντί να υλοποιήσετε τόσο τη μέθοδο doGet όσο και τη μέθοδο doPost. Αυτό δεν είναι καλή ιδέα. Αντίθετα, θα πρέπει απλώς η doPost να καλεί την doGet (ή το αντίστροφο), όπως στο επόμενο παράδειγμα:

```
public void doGet(HttpServletResponse request,
                  HttpServletResponse response)
    throws ServletException, IOExcpetion {
  //κώδικας μικροϋπηρεσίας
}

public void doPost(HttpServletResponse request,
                   HttpServletResponse response)
    throws ServletException, IOExcpetion {
  doGet(request, response);
}
```

Αν και αυτή η προσέγγιση χρειάζεται μερικές επιπλέον γραμμές κώδικα, έχει αρκετά πλεονεκτήματα σε σύγκριση με τον άμεσο υποσκελισμό της service. Πρώτον, μπορείτε αργότερα να προσθέσετε υποστήριξη για άλλες μεθόδους αιτήσεων HTTP, προσθέτοντας τις doPut, doTrace, κ.λπ., ίσως σε κάποια δευτερεύουσα κλάση. Ο άμεσος υποσκελισμός της service αποκλείει αυτή τη δυνατότητα. Δεύτερον, μπορείτε να προσθέσετε υποστήριξη για ημερομηνίες τροποποίησης, προσθέτοντας τη μέθοδο getLastModified όπως φαίνεται στη Λίστα 3.7. Επειδή η getLastModified καλείται από την προεπιλεγμένη μέθοδο service, ο υποσκελισμός της service εξαλείφει αυτή την επιλογή. Τέλος, η service σάς παρέχει αυτόματη υποστήριξη των αιτήσεων HEAD, OPTION, και TRACE.

Η προσέγγιση του βιβλίου

Αν η μικροϋπηρεσία σας πρέπει να χειριστεί με παρόμοιο τρόπο τις GET και POST, θα πρέπει η doPost να καλεί την doGet ή το αντίστροφο. Μην υποσκελίσετε τη μέθοδο service.

Οι μέθοδοι doGet, doPost, και doXxx

Αυτές οι μέθοδοι αποτελούν το "ψαχνό" της μικροϋπηρεσίας σας. Ενενήντα εννιά φορές στις εκατό θα ασχολείστε μόνο με αιτήσεις GET ή POST, οπότε θα υποσκελίζετε την doGet και/ή την doPost. Αν θέλετε, όμως, μπορείτε να υποσκελίσετε την doDelete για τις αιτήσεις DELETE, την doPut για τις αιτήσεις PUT, την doOptions για τις αιτήσεις OPTIONS, και την doTrace για

τις αιτήσεις TRACE. Να θυμάστε, όμως, ότι έχετε αυτόματη υποστήριξη για τις αιτήσεις OPTIONS και TRACE.

Κανονικά, δεν είναι απαραίτητο να υλοποιήσετε την doHead για να χειριστείτε αιτήσεις HEAD (οι αιτήσεις HEAD ορίζουν ότι ο διακομιστής θα πρέπει να επιστρέψει τις κανονικές κεφαλίδες HTTP, αλλά όχι το συσχετιζόμενο έγγραφο). Κανονικά δεν χρειάζεται να υλοποιήσετε την doHead, επειδή το σύστημα καλεί αυτόματα την doGet και χρησιμοποιεί τη γραμμή κατάστασης και τις ρυθμίσεις των κεφαλίδων για να απαντήσει σε αιτήσεις HEAD. Μερικές φορές, όμως, είναι χρήσιμη η υλοποίηση της doHead έτσι ώστε να μπορείτε να παράγετε ταχύτερα απαντήσεις σε αιτήσεις HEAD (δηλαδή αιτήσεις από προσαρμοσμένους πελάτες που θέλουν απλώς τις κεφαλίδες HTTP και όχι το πραγματικό έγγραφο) — χωρίς να πρέπει να δομήσετε την έξοδο του πραγματικού αντικειμένου.

Η μέθοδος init

Τον περισσότερο χρόνο οι μικροϋπηρεσίες σας ασχολούνται με τα δεδομένα κάθε αίτησης, και οι doGet και doPost είναι οι μόνες μέθοδοι που χρειάζεστε για ολόκληρη τη ζωή των μικροϋπηρεσιών. Περιστασιακά όμως θα θέλετε να πραγματοποιείτε πολύπλοκες εργασίες διευθέτησης όταν γίνεται η αρχική φόρτωση της μικροϋπηρεσίας, χωρίς να επαναλαμβάνετε αυτές τις εργασίες για κάθε αίτηση. Η μέθοδος init έχει σχεδιαστεί γι' αυτόν το σκοπό· η μέθοδος αυτή καλείται όταν δημιουργείται για πρώτη φορά η μικροϋπηρεσία και *δεν* καλείται πάλι για κάθε αίτηση χρήστη. Έτσι χρησιμοποιείται για την άπαξ απόδοση αρχικών τιμών, ακριβώς όπως και η μέθοδος init των μικροεφαρμογών (applets). Η μικροϋπηρεσία κανονικά δημιουργείται όταν ο χρήστης καλεί τη διεύθυνση URL που αντιστοιχεί σε αυτή, αλλά μπορείτε να καθορίσετε να γίνεται φόρτωση της μικροϋπηρεσίας την πρώτη φορά που θα γίνει εκκίνηση του διακομιστή (δείτε το κεφάλαιο σχετικά με το αρχείο web.xml στο βιβλίο More Servlets and JavaServer Pages).

Ο ορισμός της μεθόδου init είναι ο εξής.

```
public void init throws ServletException, IOException {
  //Κώδικας απόδοσης αρχικών τιμών...
}
```

Η μέθοδος init πραγματοποιεί δύο παραλλαγές απόδοσης αρχικών τιμών: γενικές αποδόσεις αρχικών τιμών, και αποδόσεις αρχικών τιμών που ρυθμίζονται από αντίστοιχες παραμέτρους.

Γενικές αποδόσεις αρχικών τιμών

Με τον πρώτο τύπο απόδοσης αρχικών τιμών (initialization) η μέθοδος init απλώς δημιουργεί ή φορτώνει ορισμένα δεδομένα τα οποία θα χρησιμοποιηθούν σε όλη τη διάρκεια της ζωής της μικροϋπηρεσίας, ή πραγματοποιεί κάποιους υπολογισμούς για μία μόνο φορά. Αν είστε εξοικειωμένοι με τις μικροεφαρμογές, αυτή η λειτουργία είναι ανάλογη με την κλήση της getImage από μια μικροεφαρμογή για τη φόρτωση αρχείων εικόνων από το δίκτυο: η λειτουργία πρέπει να

πραγματοποιηθεί μόνο μία φορά, γι' αυτό και ενεργοποιείται μέσω της init. Παραδείγματα τέτοιων περιπτώσεων για μικροϋπηρεσίες είναι οι συνδέσεις με μια δεξαμενή βάσεων δεδομένων για αιτήσεις που θα χειριστεί η μικροϋπηρεσία ή η φόρτωση ενός αρχείου δεδομένων σε ένα αντικείμενο HashMap.

Η Λίστα 3.7 παρουσιάζει μια μικροϋπηρεσία που χρησιμοποιεί τη μέθοδο init για να κάνει δύο πράγματα.

Πρώτον, κατασκευάζει έναν πίνακα 10 ακεραίων. Επειδή αυτοί οι αριθμοί βασίζονται σε πολύπλοκους υπολογισμούς, δεν θέλουμε να επαναλαμβάνεται ο υπολογισμός με κάθε αίτηση. Έτσι η doGet αναζητεί τους αριθμούς που έχει υπολογίσει η init, αντί να τους παράγει κάθε φορά. Τα αποτελέσματα αυτής της τεχνικής φαίνονται στην Εικόνα 3-6.

Δεύτερον, επειδή η έξοδος της μικροϋπηρεσίας δεν αλλάζει εκτός και αν γίνει επανεκκίνηση του διακομιστή, η init αποθηκεύει μια ημερομηνία τροποποίησης σελίδας η οποία χρησιμοποιείται από τη μέθοδο getLastModified. Η μέθοδος αυτή πρέπει να επιστρέψει το χρόνο τροποποίησης εκφρασμένο σε χιλιοστά του δευτερολέπτου από το 1970, όπως έχει καθιερωθεί για τις ημερομηνίες στη Java. Ο χρόνος μετατρέπεται αυτόματα σε μορφή GMT που είναι κατάλληλη για την κεφαλίδα Last-Modified. Το πιο σημαντικό, όμως, είναι ότι αν ο διακομιστής λάβει μία αίτηση GET υπό συνθήκη (μια αίτηση που καθορίζει ότι ο πελάτης θέλει μόνο τις σελίδες που είναι σημειωμένες ως If-Modified-Since (έχουν τροποποιηθεί μετά από κάποια συγκεκριμένη ημερομηνία), το σύστημα συγκρίνει τη συγκεκριμένη ημερομηνία με αυτή που επιστρέφει η μέθοδος getLastModified και επιστρέφει τη σελίδα μόνο αν η σελίδα έχει τροποποιηθεί μετά τη συγκεκριμένη ημερομηνία. Οι φυλλομετρητές κάνουν συχνή χρήση αυτών των αιτήσεων υπό συνθήκη για σελίδες που έχουν αποθηκεύσει στο χώρο προσωρινής αποθήκευσής τους, οπότε η υποστήριξη των αιτήσεων υπό συνθήκη βοηθά τους χρήστες σας (επειδή λαμβάνουν ταχύτερα τα αποτελέσματα) και μειώνει το φόρτο του διακομιστή (επειδή στέλνετε λιγότερα ολοκληρωμένα έγγραφα). Επειδή οι κεφαλίδες Last-Modified και If-Modified-Since χρησιμοποιούν δευτερόλεπτα, η μέθοδος getLastModified θα πρέπει να στρογγυλοποιεί τους χρόνους σε δευτερόλεπτα.

Λίστα 3.7 coreservlets/LotteryNumbers.java

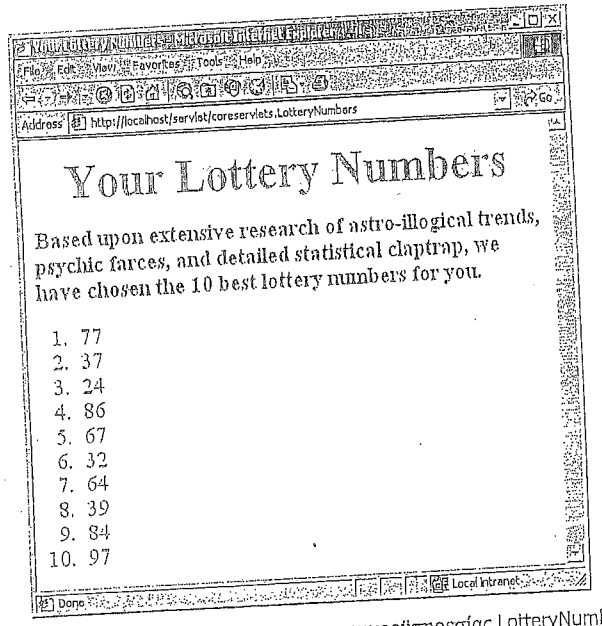
```
package coreservlets;

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

/** Παράδειγμα που χρησιμοποιεί την απόδοση αρχικών τιμών
 *  σε μικροϋπηρεσία και τη μέθοδο getLastModified.
 */

public class LotteryNumbers extends HttpServlet {
  private long modTime;
  private int[] numbers = new int[10];
```


Λίστα 3.7 coreservlets/LotteryNumbers.java *(συνέχεια)*

```
  /** Η μέθοδος init καλείται μόνο όταν φορτώνεται η μικροϋπηρεσία
   *  για πρώτη φορά, πριν γίνει η επεξεργασία της πρώτης αίτησης.
   */

  public void init() throws ServletException {
    // Στρογγυλοποίηση στο πλησιέστερο δευτερόλεπτο (δηλαδή, 1000 msec)
    modTime = System.currentTimeMillis()/1000*1000;
    for(int i=0; i<numbers.length; i++) {
      numbers[i] = randomNum();
    }
  }

/** Επιστροφή της λίστας των αριθμών που υπολόγισε η init. */

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Your Lottery Numbers";
    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" +
                "<B>Based upon extensive research of " +
                "astro-illogical trends, psychic farces, " +
                "and detailed statistical claptrap, " +
                "we have chosen the " + numbers.length +
                " best lottery numbers for you.</B>" +
                "<OL>");
    for(int i=0; i<numbers.length; i++) {
      out.println("  <LI>" + numbers[i]);
    }
    out.println("</OL>" +
                "</BODY></HTML>");
  }

  /** Η τυπική μέθοδος service συγκρίνει αυτή την ημερομηνία
   *  με την ημερομηνία που καθορίζεται στην κεφαλίδα αίτησης
   *  If-Modified-Since.
   *  Αν η ημερομηνία της getLastModified είναι μεταγενέστερη ή δεν
   *  υπάρχει κεφαλίδα If-Modified-Since, η μέθοδος doGet καλείται
   *  κανονικά. Αν όμως η ημερομηνία της getLastModified είναι ίδια ή
```

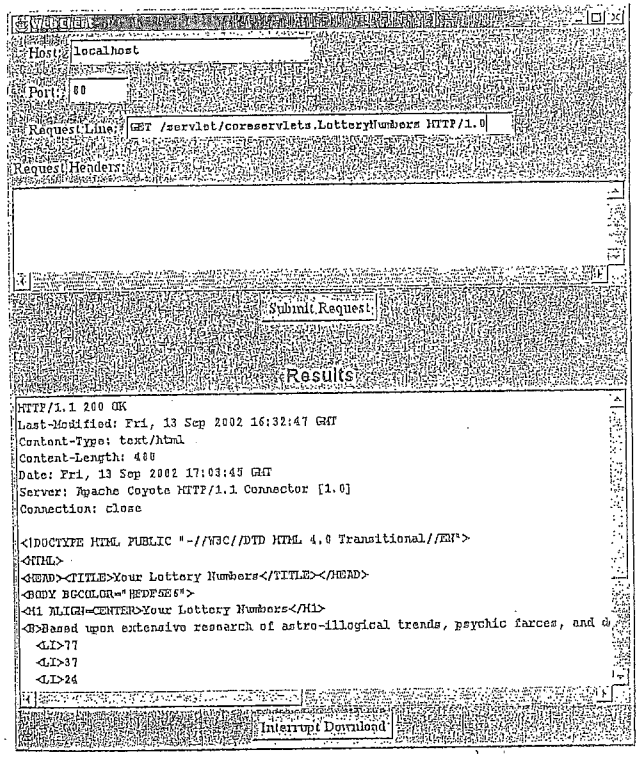
Λίστα 3.7 coreservlets/LotteryNumbers.java *(συνέχεια)*

```
 *  προγενέστερη, η μέθοδος service στέλνει την απάντηση 304
 *  (Δεν έχει τροποποιηθεί) και <B>δεν</B>καλεί την doGet.
 *  Σε αυτή την περίπτωση ο φυλλομετρητής πρέπει να χρησιμοποιήσει
 *  την αποθηκευμένη εκδοχή της σελίδας.
 */

public long getLastModified(HttpServletRequest request) {
  return(modTime);
}

// Ένας τυχαίος ακέραιος από 0 μέχρι 99.

private int randomNum() {
  return((int)(Math.random() * 100));
}
}
```

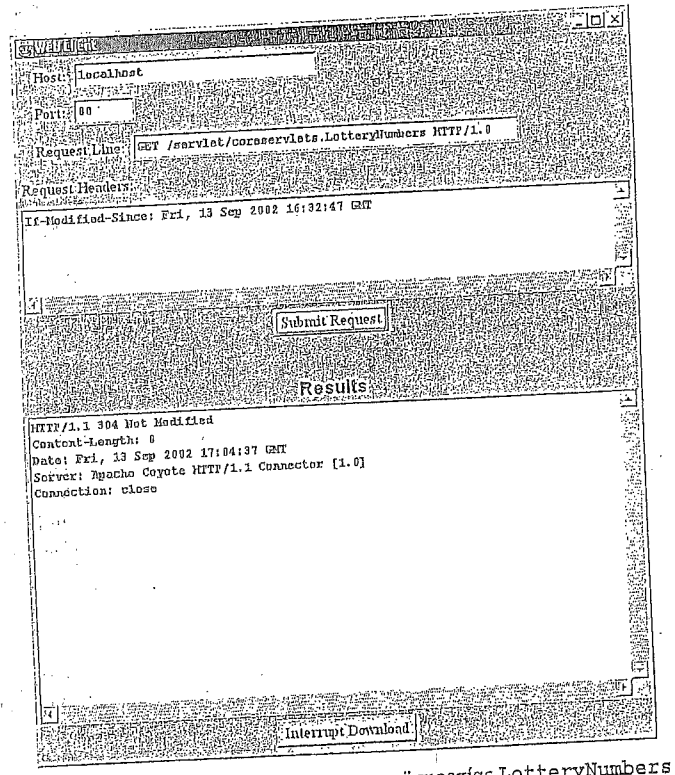


Εικόνα 3-6 Το αποτέλεσμα της μικροϋπηρεσίας LotteryNumbers.

Οι Εικόνες 3-7 και 3-8 παρουσιάζουν το αποτέλεσμα αιτήσεων για την ίδια μικροϋπηρεσία με δύο ελαφρώς διαφορετικές ημερομηνίες `If-Modified-Since`. Για να ορίσουμε τις κεφαλίδες αίτησης και να δούμε τις κεφαλίδες απάντησης χρησιμοποιήσαμε το `WebClient`, μια εφαρμογή Java που σας επιτρέπει να διευθετείτε αλληλεπιδραστικά αιτήσεις HTTP, να τις υποβάλλετε, και να βλέπετε τα "ανεπεξέργαστα" αποτελέσματα. Ο κώδικας της εφαρμογής `WebClient` υπάρχει στην αρχειοθήκη πηγαίου κώδικα, στην ηλεκτρονική ιστοσελίδα του βιβλίου (http://www.coreservlets.com/).



Εικόνα 3-7 Η προσπέλαση της μικροϋπηρεσίας `LotteryNumbers` έχει αποτέλεσμα κανονική απόκριση (όπου το έγγραφο αποστέλλεται στον πελάτη) σε δύο περιπτώσεις: όταν υπάρχει αίτηση GET χωρίς συνθήκη, ή όταν υπάρχει αίτηση υπό συνθήκη στην οποία η ημερομηνία που καθορίζεται είναι προγενέστερη από την ημερομηνία που έγινε η απόδοση αρχικών τιμών στη μικροϋπηρεσία. Ο κώδικας της εφαρμογής `WebClient` (η οποία εδώ χρησιμοποιείται για την αλληλεπιδραστική σύνδεση με το διακομιστή) υπάρχει στην αρχειοθήκη πηγαίου κώδικα του βιβλίου, στη διεύθυνση http://www.coreservlets.com/.



Εικόνα 3-8 Η προσπέλαση της μικροϋπηρεσίας `LotteryNumbers` έχει αποτέλεσμα την απάντηση 304 (Not Modified, δεν έχει τροποποιηθεί) χωρίς πραγματικό έγγραφο σε μία περίπτωση: όταν ληφθεί αίτηση `GET` υπό συνθήκη στην οποία η ημερομηνία που καθορίζεται είναι ίδια ή μεταγενέστερη από την ημερομηνία απόδοσης αρχικών τιμών στη μικροϋπηρεσία.

Αποδόσεις αρχικών τιμών που ρυθμίζονται με παραμέτρους

Στο προηγούμενο παράδειγμα η μέθοδος `init` υπολόγισε κάποια δεδομένα τα οποία χρησιμοποιήθηκαν από τις μεθόδους `doGet` και `getLastModified`. Αν και αυτός ο τύπος γενικής απόδοσης αρχικών τιμών είναι αρκετά συνηθισμένος, εξίσου συνηθισμένος είναι και ο έλεγχος της απόδοσης αρχικών τιμών με τη χρήση παραμέτρων απόδοσης αρχικών τιμών (initialization parameters). Για να καταλάβετε το κίνητρο για τις παραμέτρους απόδοσης αρχικών τιμών, θα πρέπει να σκεφθείτε τις κατηγορίες των ανθρώπων που μπορεί να θέλουν να προσαρμόσουν τον τρόπο με τον οποίο συμπεριφέρεται μια μικροϋπηρεσία ή μια σελίδα JSP. Υπάρχουν τρεις τέτοιες ομάδες ανθρώπων:

1. Προγραμματιστές.
2. Τελικοί χρήστες.
3. Υπεύθυνοι εκδίπλωσης.

Οι προγραμματιστές αλλάζουν τη συμπεριφορά μιας μικροϋπηρεσίας αλλάζοντας τον κώδικα. Οι τελικοί χρήστες αλλάζουν τη συμπεριφορά μιας μικροϋπηρεσίας παρέχοντας δεδομένα σε μια φόρμα HTML (με την προϋπόθεση ότι ο προγραμματιστής έχει γράψει τη μικροϋπηρεσία με τέτοιον τρόπο ώστε να διαβάζει αυτά τα δεδομένα). Τι γίνεται όμως με τους υπευθύνους εκδίπλωσης; Θα πρέπει να υπάρχει κάποιος τρόπος που να επιτρέπει στους διαχειριστές να μεταφέρουν μικροϋπηρεσίες από μηχάνημα σε μηχάνημα και να αλλάζουν ορισμένες παραμέτρους (π.χ., τη διεύθυνση μιας βάσης δεδομένων, το μέγεθος μιας δεξαμενής συνδέσεων, ή τη θέση ενός αρχείου δεδομένων) χωρίς να χρειάζεται να τροποποιηθεί ο πηγαίος κώδικας της μικροϋπηρεσίας. Η παροχή αυτής της δυνατότητας είναι ο σκοπός αυτών των παραμέτρων.

Επειδή η χρήση των παραμέτρων απόδοσης αρχικών τιμών βασίζεται σε πολύ μεγάλο βαθμό στο αρχείο περιγραφής εκδίπλωσης (web.xml), θα αναβάλουμε τις λεπτομέρειες και τα παραδείγματα παραμέτρων απόδοσης αρχικών τιμών μέχρι το κεφάλαιο που εξετάζει το αρχείο αυτό, στο βιβλίο More Servlets and JavaServer Pages. Εδώ θα παρουσιάσουμε απλώς μια σύντομη επισκόπηση:

1. Χρησιμοποιήστε το στοιχείο `sevlet` του αρχείου web.xml για να δώσετε ένα όνομα στη μικροϋπηρεσία σας.
2. Χρησιμοποιήστε το στοιχείο `servlet-mapping` για να εκχωρήσετε μια προσαρμοσμένη διεύθυνση URL στη μικροϋπηρεσία σας. Ποτέ δεν θα χρησιμοποιείτε τα προεπιλεγμένα URL της μορφής http://.../**servlet/***ΌνομαΜικροϋπηρεσίας* όταν χρησιμοποιείτε παραμέτρους απόδοσης αρχικών τιμών. Στην πράξη, αυτά τα προεπιλεγμένα URL, αν και είναι εξαιρετικά χρήσιμα κατά τη διάρκεια της ανάπτυξης, δεν χρησιμοποιούνται σχεδόν ποτέ στα σενάρια εκδίπλωσης.
3. Προσθέστε υποστοιχεία `init_param` στο στοιχείο `servlet` του web.xml για να εκχωρήσετε ονόματα και τιμές στις παραμέτρους απόδοσης αρχικών τιμών.
4. Από το εσωτερικό της μεθόδου `init` της μικροϋπηρεσίας σας καλέστε τη μέθοδο `getServletConfig` για να πάρετε μια αναφορά στο αντικείμενο `ServletConfig`.
5. Καλέστε τη μέθοδο `getInitParameter` του αντικειμένου `ServletConfig` με το όνομα της παραμέτρου απόδοσης αρχικών τιμών. Η επιστρεφόμενη τιμή είναι η τιμή της παραμέτρου απόδοσης αρχικών τιμών, ή η τιμή `null` αν δεν υπάρχει τέτοια παράμετρος στο αρχείο web.xml.

Η μέθοδος destroy

Ο διακομιστής μπορεί να αποφασίσει να αφαιρέσει το στιγμιότυπο μιας μικροϋπηρεσίας που έχει φορτωθεί, πιθανώς επειδή του ζητήθηκε ρητά κάτι τέτοιο από το διαχειριστή του διακομιστή ή επειδή η μικροϋπηρεσία είναι ανενεργός για μεγάλο χρονικό διάστημα. Πριν το κάνει όμως αυτό, καλεί τη μέθοδο `destroy` της μικροϋπηρεσίας. Αυτή η μέθοδος δίνει στη μικροϋπηρεσία σας

την ευκαιρία να κλείσει συνδέσεις βάσεων δεδομένων, να σταματήσει νήματα που λειτουργούν στο παρασκήνιο, να γράψει στο δίσκο λίστες μπισκότων (cookies lists) ή μετρήσεις επισκέψεων (hit counts), και να πραγματοποιήσει εργασίες καθαρισμού. Να θυμάστε, όμως, ότι είναι πιθανόν να καταρρεύσει ο διακομιστής Ιστού (θυμάστε τις διακοπές ρεύματος στην Αθήνα;). Γι' αυτό μη βασίζεστε στη `destroy` ως *μοναδικό* μηχανισμό αποθήκευσης δεδομένων κατάστασης στο δίσκο. Αν η μικροϋπηρεσία σας εκτελεί δραστηριότητες όπως η καταμέτρηση επισκέψεων ή η συγκέντρωση καταλόγων με τιμές μπισκότων που υποδεικνύουν ειδική πρόσβαση, θα πρέπει περιοδικά, για λόγους προφύλαξης, να γράφετε τα δεδομένα στο δίσκο.

3.7 Η διασύνδεση SingleThreadModel

Κανονικά το σύστημα δημιουργεί ένα μόνο στιγμιότυπο της μικροϋπηρεσίας σας και στη συνέχεια δημιουργεί ένα νέο νήμα για κάθε αίτηση χρήστη. Αυτό σημαίνει ότι, αν εμφανιστεί μια νέα αίτηση ενώ γίνεται επεξεργασία μιας προηγούμενης αίτησης, μπορούν ταυτόχρονα πολλά νήματα να προσπελάζουν το ίδιο αντικείμενο μικροϋπηρεσίας. Επομένως οι μέθοδοι `doGet` και `doPost` θα πρέπει να είναι προσεκτικές έτσι ώστε να συγχρονίζουν την πρόσβαση σε πεδία και άλλα κοινόχρηστα δεδομένα (αν υπάρχουν), αφού πολλά νήματα μπορούν να προσπελάσουν ταυτόχρονα τα δεδομένα. Ας σημειωθεί ότι τα διάφορα νήματα δεν μοιράζονται τις τοπικές μεταβλητές, και επομένως αυτές δεν χρειάζονται ειδική προστασία.

Θεωρητικά μπορείτε να εμποδίσετε τη πολυνηματική πρόσβαση αναγκάζοντας τη μικροϋπηρεσία σας να υλοποιήσει τη διασύνδεση `SingleThreadModel`, όπως παρακάτω:

```
public class YourServlet extends HttpServlet
    implements SingleThreadModel {
...
}
```

Αν υλοποιήσετε αυτή τη διασύνδεση, το σύστημα εγγυάται ότι ποτέ δεν θα υπάρχουν περισσότερα από ένα νήματα αιτήσεων που θα προσπελάζουν το στιγμιότυπο της μικροϋπηρεσίας σας. Στις περισσότερες περιπτώσεις αυτό επιτυγχάνεται με την τοποθέτηση όλων των αιτήσεων στην ουρά και τη μεταβίβασή τους, μία κάθε φορά, στο στιγμιότυπο της μικροϋπηρεσίας. Ωστόσο, ο διακομιστής επιτρέπεται να δημιουργήσει μια δεξαμενή πολλαπλών στιγμιοτύπων, όπου το καθένα θα χειρίζεται μία αίτηση κάθε φορά. Και με τους δύο τρόπους, αυτό σημαίνει ότι δεν χρειάζεται να ανησυχείτε για την ταυτόχρονη προσπέλαση κανονικών πεδίων (μεταβλητών στιγμιοτύπων) της μικροϋπηρεσίας. Θα πρέπει, όμως, να συγχρονίσετε την πρόσβαση σε μεταβλητές κλάσεων (πεδία static) ή σε κοινόχρηστα δεδομένα που είναι αποθηκευμένα έξω από τη μικροϋπηρεσία.

Αν και θεωρητικά η διασύνδεση `SingleThreadModel` δεν επιτρέπει την ταυτόχρονη πρόσβαση, στην πράξη υπάρχουν δύο λόγοι για τους οποίους αποτελεί κακή πρακτική.

Πρώτον, η σύγχρονη πρόσβαση στις μικροϋπηρεσίες σας μπορεί να επιβαρύνει σε σημαντικό βαθμό την απόδοση (υπάρχει υστέρηση), αν γίνεται συχνή πρόσβαση σε αυτές. Όταν η μικροϋπηρεσία αναμένει είσοδο/έξοδο, ο διακομιστής δεν μπορεί να χειριστεί αιτήσεις που εκκρεμούν για την ίδια μικροϋπηρεσία. Έτσι θα πρέπει να το σκεφθείτε καλά πριν χρησιμοποιήσετε την

προσέγγιση `SingleThreadModel`. Αντί γι' αυτό, θα πρέπει να εξετάσετε μήπως πρέπει να εφαρμόσετε συγχρονισμό μόνο στο τμήμα του κώδικα που χειρίζεται τα κοινόχρηστα δεδομένα.

Το δεύτερο πρόβλημα της προσέγγισης `SingleThreadModel` οφείλεται στο γεγονός ότι οι προδιαγραφές επιτρέπουν στους διακομιστές να χρησιμοποιούν δεξαμενές στιγμιότυπων, αντί να τοποθετούν στην ουρά ενός στιγμιοτύπου τις αιτήσεις. Στο βαθμό που κάθε στιγμιότυπο χειρίζεται μία μόνο αίτηση κάθε φορά, η προσέγγιση της δεξαμενής στιγμιότυπων ικανοποιεί τις απαιτήσεις των προδιαγραφών. Είναι όμως κακή ιδέα.

Από τη μία πλευρά, υποθέστε ότι χρησιμοποιείτε κανονικές, μη στατικές, μεταβλητές στιγμιοτύπου (πεδία) για να κάνετε αναφορά σε κοινόχρηστα δεδομένα. Είναι σαφές ότι η `SingleThreadModel` εμποδίζει την ταυτόχρονη πρόσβαση, αλλά το καταφέρνει καίγοντας τα χλωρά μαζί με τα ξερά: κάθε στιγμιότυπο μικροϋπηρεσίας έχει ξεχωριστό αντίγραφο των μεταβλητών στιγμιοτύπου, έτσι πλέον η κοινοχρησία των δεδομένων δεν γίνεται σωστά.

Από την άλλη πλευρά, υποθέστε ότι χρησιμοποιείτε στατικές μεταβλητές στιγμιοτύπου για να κάνετε αναφορά σε κοινόχρηστα δεδομένα. Σε αυτή την περίπτωση η προσέγγιση της δεξαμενής στιγμιοτύπων με τη διασύνδεση `SingleThreadModel` δεν παρέχει κανένα πλεονέκτημα: πολλαπλές αιτήσεις (που χρησιμοποιούν διαφορετικά στιγμιότυπα) μπορούν να προσπελάσουν ταυτόχρονα τα στατικά δεδομένα.

Παρόλα αυτά, η `SingleThreadModel` είναι σε μερικές περιπτώσεις χρήσιμη. Για παράδειγμα, μπορεί να χρησιμοποιηθεί όταν οι μεταβλητές στιγμιοτύπου παίρνουν εκ νέου αρχικές τιμές με κάθε αίτηση (π.χ. όταν χρησιμοποιούνται μόνο για την απλοποίηση της επικοινωνίας μεταξύ των μεθόδων). Πάντως, τα προβλήματα με τη `SingleThreadModel` είναι τόσο σοβαρά που καταργήθηκε από τις προδιαγραφές των μικροϋπηρεσιών 2.4 (JSP 2.0). Είναι πολύ καλύτερο να χρησιμοποιήσετε συγχρονισμένα (`synchronized`) μπλοκ κώδικα.

Προειδοποίηση

Αποφύγετε την υλοποίηση της διασύνδεσης `SingleThreadModel` για μικροϋπηρεσίες με μεγάλη κυκλοφορία. Χρησιμοποιήστε τη με προσοχή στις άλλες περιπτώσεις. Για κώδικα επιπέδου παραγωγής, ο ρητός συγχρονισμός κώδικα είναι σχεδόν πάντοτε καλύτερη λύση. Η διασύνδεση `SingleThreadModel` καταργήθηκε στην έκδοση 2.4 των προδιαγραφών μικροϋπηρεσιών.

Για παράδειγμα, εξετάστε τη μικροϋπηρεσία της Λίστας 3.8 που προσπαθεί να εκχωρήσει μοναδικά αναγνωριστικά χρήστη (ID) σε κάθε πελάτη (προφανώς, είναι μοναδικά μέχρι να γίνει επανεκκίνηση του διακομιστή). Η μικροϋπηρεσία χρησιμοποιεί μια μεταβλητή στιγμιοτύπου (πεδίο) με το όνομα `nextID` για να παρακολουθεί ποιο ID θα πρέπει να εκχωρηθεί στη συνέχεια, και χρησιμοποιεί τον παρακάτω κώδικα για την έξοδο του ID: