Ανώτατο Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Φοιτητές που εργάστηκαν για την παρούσα πτυχιακή εργασία:

• • •

Πτυχιακή Εργασία:

Μετατροπή του δικτυακού τόπου του Μουσικού Εργαστηρίου «Λαβύρινθοι» του Ross Dally από γλώσσα προγραμματισμού διαδικτύου **PHP** σε **JSP** και κατασκευή της ανάλογης Βάσης Δεδομένων.

• • •

Επώνυμο	Όνομα	Πατρώνυμο	Αριθμός Μητρώου	Εξάμηνο
Βούρβαχης	Αντώνιος	Γεώργιος	1086	ΠΓ’
Γιάνναρος	Παναγιώτης	Μιχαήλ-Στυλιανός	1148	ΠΒ’

Υπεύθυνος Καθηγητής:

Αιβαλής	Κωνσταντίνος	Καθηγητής Τομέα Πληροφορικής

Πίνακας Περιεχομένων

1. Πρόλογος

Στην παρούσα πτυχιακή εργασία καλούμαστε να μετατρέψουμε τον δικτυακό τόπο του Μουσικού Εργαστηρίου «Λαβύρινθοι» του Ross Daily το οποίο είναι κατασκευασμένο σε γλώσσα PHP σε γλώσσα JSP καθώς επίσης, και την βάση δεδομένων που κρατάει τα δυναμικά στοιχεία των σελίδων του.

Στις παραγράφους που ακολουθούν ο αναγνώστης μπορεί να ενημερωθεί τόσο για την χρησιμότητα και τις δραστηριότητες του Μουσικού Εργαστηρίου, όσο επίσης, και για τεχνικές λεπτομέρειες που σχετίζονται με τον τρόπο κατασκευής του δικτυακού τόπου καθώς επίσης, παρέχεται και μια σύντομη αναφορά στις λειτουργίες της γλώσσας η οποία χρησιμοποιήθηκε.

2. Μουσικό Εργαστήριο «Λαβύρινθοι»

Το Μουσικό εργαστήριο «Λαβύρινθοι» ιδρύθηκε το 1982 από τον Ross Daly, με κύριο αντικείμενο τη μύηση νέων ανθρώπων σε μια δημιουργική προσέγγιση μουσικών παραδόσεων σχεδόν από ολόκληρο τον κόσμο. Από το 2002 το Μουσικό Εργαστήρι «Λαβύρινθοι» σε συνεργασία με το Δήμο «Νίκος Καζαντζάκης», στεγάζει τις δραστηριότητες του σε ένα παλιό κτίριο, καλόγουστα διαμορφωμένο, στο χωριό Χουδέτσι, το οποίο απέχει περίπου 20 χλμ. από το Ηράκλειο, υπό την καλλιτεχνική διεύθυνση του Ross Daly.

Ο σκοπός του Μουσικού Εργαστηρίου «Λαβύρινθοι» είναι: Η συλλογή, διάσωση και προβολή παραδοσιακών μουσικών οργάνων από όλο τον κόσμο. Τα εκθέματα του μουσείου αποτελούν τεκμήρια των κοινωνικών, ιστορικών και πολιτισμικών εκφάνσεων των χωρών προέλευσης. Είναι το μοναδικό μουσείο του είδους του στην Ελλάδα και περιλαμβάνει πλούσιες συλλογές που διαφωτίζουν τις αντιπροσωπευτικότερες παραδοσιακές μουσικές, από την αρχαιότητα έως σήμερα. Το Μουσικό Εργαστήρι «Λαβύρινθοι» είναι κέντρο έρευνας κι εκμάθησης των οργάνων της παραδοσιακής μουσικής ανά τον κόσμο και στοχεύει στην διατήρηση της παγκόσμιας μουσικής παράδοσης. Καλλιεργεί τη μουσική γνώση και συμβάλλει στην προβολή νέων μουσικών δημιουργών.

Στα πλαίσια αυτής της προσπάθειας διοργανώνονται συναυλίες που προάγουν τους σκοπούς και το αντικείμενό του κι επεκτείνονται σε ολόκληρη τη χώρα αλλά κι εκτός συνόρων. Η Κρήτη ήταν ανέκαθεν σημείο συνάντησης πολλών και διαφόρων πολιτισμών, τόσο λόγω της γεωγραφικής της θέσης, όσο και λόγω του υπέροχου κλίματός της. Το εγχείρημα του Μουσικού Εργαστηρίου «Λαβύρινθοι» στηρίζεται ακριβώς στο γεγονός αυτό. Πιστεύουμε ακράδαντα ότι η Κρήτη είναι σήμερα το ιδανικότερο μέρος της Μεσογείου για να αποτελέσει εκπαιδευτικό και πολιτιστικό κέντρο για πολλούς γειτονικούς και συγγενικούς λαούς. Είναι παρατηρημένο δε, ότι τις τελευταίες δεκαετίες υπάρχει στην Ελλάδα και στην Ευρώπη αυξημένο ενδιαφέρον για τις μουσικές αυτές παραδόσεις, γεγονός που επιβάλλει την ύπαρξη ενός τέτοιου πολυπολιτισμικού εκπαιδευτικού κέντρου. Το Μουσικό Εργαστήρι «Λαβύρινθοι» είναι ένα πρωτοποριακό σε παγκόσμιο επίπεδο εγχείρημα.

Έκθεση Μουσικών Οργάνων του Κόσμου: Η συλλογή, ταξινομημένη ειδολογικά, αποτελεί στυλοβάτη των ερευνητικών κι εκθεσιακών δραστηριοτήτων του εργαστηρίου.

Οργανοποιείο: Στο μουσείο «Λαβύρινθοι» λειτουργεί κατάλληλα εξοπλισμένο εργαστήριο όπου πραγματοποιούνται εργασίες κατασκευής και συντήρησης των μουσικών οργάνων. Ιδιαίτερη μέριμνα επιδεικνύεται για την προστασία των αντικειμένων της έκθεσης, καθώς και για τη διαμόρφωση των χώρων φύλαξης των συλλογών.

3. Η γλώσσα προγραμματισμού διαδικτύου JSP (Java Server Pages)

Η JavaServer τεχνολογία σελίδων (JSP) μας επιτρέπει να αναμίξουμε το κανονικό, το στατικό HTML με το δυναμικά παραγμένο περιεχόμενο από τα servlets. Πολλές ιστοσελίδες που χτίζονται από τα προγράμματα της CGI είναι πρώτιστα στατικές, με τα μέρη που η αλλαγή περιόρισε σε μερικές μικρές θέσεις. Παραδείγματος χάριν, η αρχική σελίδα στα πιο on-line καταστήματα είναι η ίδια για όλους τους επισκέπτες, εκτός από ένα μικρό ευπρόσδεκτο μήνυμα που δίνει το όνομα του επισκέπτη εάν αυτό είναι γνωστό. Αλλά οι περισσότερες παραλλαγές της CGI, συμπεριλαμβάνων και των servlets, μας κάνουν να παράγουμε ολόκληρη την σελίδα μέσω του προγράμματός μας, ακόμα κι αν

το μεγαλύτερο μέρος της είναι πάντα η ίδια. Η JSP μας επιτρέπει να δημιουργήσουμε αυτά τα δύο μέρη χωριστά. Το μεγαλύτερο μέρος της σελίδας αποτελείται από το κανονικό HTML, το οποίο περνάει στον επισκέπτη αμετάβλητο. Τα μέρη που παράγονται δυναμικά είναι μαρκαρισμένα με ειδικές HTML-like ετικέτες και μικτά δεξιά στη σελίδα.

3.1. Τα πλεονεκτήματα της JSP

Η JSP έχει διάφορα πλεονεκτήματα πέρα από τις πολλές εναλλακτικές λύσεις της. Παρακάτω παρουσιάζονται μερικά από αυτά και τα οποία είναι:

Σε σύγκριση με τις Active Server Pages (ASP)

Η ASP είναι μια ανταγωνιστική τεχνολογία προερχόμενη από τη Microsoft. Τα πλεονεκτήματα της JSP είναι διπλά. Κατ' αρχάς, το δυναμικό μέρος γράφεται σε Java, κι όχι σε VBScript ή μια άλλη ASP-specific γλώσσα, κι έτσι αυτό την κάνει να είναι ισχυρότερη και καλύτερη που ταιριάζει στις σύνθετες εφαρμογές που απαιτούν τα επαναχρησιμοποιήσιμα συστατικά. Δεύτερον, η JSP είναι φορητή σε άλλα λειτουργικά συστήματα και Servers δικτύου στα οποία δεν είμαστε κλειδωμένοι στα Windows NT/2000 και IIS. Θα μπορούσαμε να προβάλουμε το ίδιο επιχείρημα κατά τη σύγκριση της JSP με ColdFusion με JSP που μπορούμε να χρησιμοποιήσουμε την Java και δεν είμαστε δεμένοι σε ένα κατάλληλο προϊόν Server.

Σε σύγκριση με την PHP

Η PHP είναι ελεύθερη, ανοικτού-κώδικα HTML-embedded scripting γλώσσα που είναι κάπως παρόμοια τόσο με την ASP, όσο επίσης, και με την JSP. Το πλεονέκτημα της JSP είναι ότι το δυναμικό μέρος γράφεται σε Java, για την οποία πιθανώς ήδη είμαστε ενήμεροι, το οποίο έχει ήδη ένα εκτενές API για τη δικτύωση, την πρόσβαση των βάσεων δεδομένων, τα διανεμημένα αντικείμενα, και τους ομοίους, ενώ η PHP απαιτεί μια εξ'ολοκλήρου νέα γλώσσα.

Σε σύγκριση με τα καθαρά Servlets

Η JSP δεν παρέχει οποιεσδήποτε ικανότητες που δεν θα μπορούσαν να ολοκληρωθούν σε γενικές γραμμές με ένα servlet. Στην πραγματικότητα, τα έγγραφα JSP είναι αυτόματα μεταφρασμένα στα servlets πίσω από τις σκηνές. Αλλά είναι καταλληλότερη να γράψει, αλλά και για να τροποποιήσει κανονικό HTML από το να έχει τεράστιο πλήθος δηλώσεων println που παράγουν το HTML. Επιπλέον, με το χωρισμό της παρουσίασης από το περιεχόμενο, μπορούμε να βάλουμε διαφορετικούς ανθρώπους σε διαφορετικούς στόχους: οι δικοί μας εμπειρογνώμονες σχεδίου ιστοσελίδας, μπορούν να χτίσουν το HTML χρησιμοποιώντας τα εξοικειωμένα εργαλεία και να αφήσουν τις θέσεις για τους προγραμματιστές μας των servlet για να παρεμβάλουν το δυναμικό περιεχόμενο.

Σε σύγκριση με τα Server-Side Includes (SSI)

Τα SSI είναι μια ευρέως υποστηριγμένη τεχνολογία για την είσοδο των εξωτερικά καθορισμένων κομματιών στατικής ιστοσελίδας. Η JSP είναι καλύτερη επειδή έχουμε ένα πλουσιότερο σύνολο εργαλείων για το χτίσιμο αυτού του εξωτερικού κομματιού κι έχουμε περισσότερες επιλογές σχετικά με το στάδιο της απάντησης HTTP στην οποία το κομμάτι εισέρχεται πραγματικά. Εκτός αυτού, τα SSI προορίζονται πραγματικά μόνο για τους απλούς συνυπολογισμούς, κι όχι για τα "πραγματικά" προγράμματα τα οποία χρησιμοποιούν form data, κάνουν τις συνδέσεις των βάσεων δεδομένων, και τους ομοίους.

Σε σύγκριση με την JavaScript

Η JavaScript, που είναι απολύτως ευδιάκριτη από τη γλώσσα προγραμματισμού της Java, χρησιμοποιείται κανονικά για να παράγει το HTML δυναμικά στον client, που χτίζει τα μέρη ιστοσελίδας δεδομένου ότι ο browser «φορτώνει» το έγγραφο. Αυτή είναι μια χρήσιμη ικανότητα αλλά μόνο στην περίπτωση όταν χειρίζεται τις καταστάσεις όπου οι δυναμικές πληροφορίες είναι βασισμένες στο περιβάλλον του client. Με εξαίρεση τα Cookies, το στοιχείο αιτήματος HTTP δεν είναι διαθέσιμο στις client-side ρουτίνες JavaScript. Κι, από τις ρουτίνες ελλείψεων JavaScript για τον προγραμματισμό δικτύων, ο κώδικας JavaScript στον client δεν μπορεί να προσπελάσει server-side πόρους όπως τις βάσεις δεδομένων, τους καταλόγους, τις πληροφορίες τιμολόγησης, και τους ομοίους. Η JavaScript μπορεί επίσης, να χρησιμοποιηθεί στον Server, κι ειδικότερα στους Servers Netscape κι ως scripting γλώσσα για IIS. Η Java είναι με μεγάλη διαφορά πιο ισχυρή, εύκαμπτη, αξιόπιστη, και φορητή.

Σε σύγκριση με την στατική HTML

Το κανονικό HTML, φυσικά, δεν μπορεί να περιέχει τις δυναμικές πληροφορίες, κι έτσι οι στατικές σελίδες HTML δεν μπορούν να βασιστούν στις πηγές στοιχείων εισόδου ή στα server-side δεδομένα πηγών. Η JSP είναι τόσο εύκολη και κατάλληλη που είναι αρκετά λογικό να αυξηθούν οι σελίδες HTML που ωφελούνται μόνο ελαφρώς από την εισαγωγή των δυναμικών στοιχείων. Προηγουμένως, η δυσκολία της χρησιμοποίησης των δυναμικών στοιχείων απέκλεισε τη χρήση της σε όλες εκτός από τις πολυτιμότερες περιπτώσεις.

3.2. Στοιχεία JSP Scripting

Τα scripting στοιχεία JSP μας αφήνουν να εισάγουμε τον κώδικα στο servlet που θα παραχθεί από τη σελίδα JSP. Υπάρχουν τρεις μορφές:

- Εκφράσεις της έκφρασης μορφής `<%= expression %>`, οι οποίες αξιολογούνται και παρεμβάλλονται στην έξοδο του servlet.
- Scriptlets της μορφής `<% code %>`, τα οποία παρεμβάλλονται στη μέθοδο *_jspService* του servlet, τα οποία καλούνται από την μέθοδο *service*.
- Δηλώσεις της μορφής `<%! code %>`, που παρεμβάλλονται στο σώμα της κλάσης Servlet, έξω από οποιεσδήποτε υπάρχουσες μεθόδους.

Template Text

Σε πολλές περιπτώσεις, ένα μεγάλο ποσοστό της σελίδας JSP μας αποτελείται ακριβώς από το στατικό HTML, γνωστό και σαν *template text*. Από σχεδόν κάθε άποψη, αυτό το HTML ακριβώς όπως το κανονικό HTML, ακολουθεί όλους τους ίδιους κανόνες σύνταξης, κι απλά "περνά μέσω" στον client από το servlet που δημιουργείται, με σκοπό να χειριστεί τη σελίδα. Όχι μόνο το HTML φαίνεται κανονικό, αλλά μπορεί να δημιουργηθεί από οτιδήποτε εργαλεία τα οποία ήδη χρησιμοποιούμε για την οικοδόμηση μιας ιστοσελίδας.

Υπάρχουν δύο δευτερεύουσες εξαιρέσεις στο "template text που περνάει κατ' ευθείαν μέσω" του κανόνα. Κατ' αρχάς, εάν θέλουμε να έχουμε `<%` στην έξοδο, πρέπει να βάλουμε `<\%` στο template text. Δεύτερον, εάν θέλουμε ένα σχόλιο για να εμφανίσουμε στη σελίδα JSP αλλά όχι στο επακόλουθο έγγραφο, χρησιμοποιούμε `<%-- JSP Comment --%>` HTML της φόρμας `<!-- HTML Comment -->` που περνούν μέσω στο επακόλουθο HTML κανονικά.

3.3. Εκφράσεις της JSP

Μια έκφραση JSP χρησιμοποιείται για να εισάγει τις τιμές άμεσα στην έξοδο. Έχει την ακόλουθη μορφή:

```
<%= Java Expression %>
```

Η έκφραση αξιολογείται, μετατρέπεται σε μια συμβολοσειρά, κι εισάγεται στη σελίδα. Αυτή η αξιολόγηση εκτελείται στο χρόνο τρεξίματος, όταν ζητείται η σελίδα, κι έχει έτσι πλήρη πρόσβαση στις πληροφορίες που αφορούν το αίτημα. Παραδείγματος χάριν, το ακόλουθο παράδειγμα παρουσιάζει τα date/time στα οποία η σελίδα ζητήθηκε:

```
Current time: <%= new java.util.Date() %>
```

Προκαθορισμένες μεταβλητές

Για να απλοποιήσουμε αυτές τις εκφράσεις, μπορούμε να χρησιμοποιήσουμε διάφορες προκαθορισμένες μεταβλητές. Οι σημαντικότερες εκφράσεις είναι:

- request, την *HttpServletRequest*
- response, την *HttpServletResponse*
- session, την *HttpSession* που συνδέεται με το αίτημα, εκτός αν τεθεί εκτός λειτουργίας με την ιδιότητα *session* της οδηγίας *page*.
- out, το *PrintWriter*, μια αποθηκευμένη έκδοση αποκαλούμενη *JspWriter*, που χρησιμοποιείται για να στείλει την έξοδο στον client. Παρακάτω ακολουθεί ένα παράδειγμα:

```
Your hostname: <%= request.getRemoteHost() %>
```

Σύνταξη XML για εκφράσεις

Οι συντάκτες XML μπορούν να χρησιμοποιήσουν την ακόλουθη εναλλακτική σύνταξη για τις εκφράσεις της JSP:

```
<jsp:expression>

Java Expression

</jsp:expression>
```

Σημειώνουμε ότι τα στοιχεία XML, αντίθετα από αυτά της HTML, είναι case sensitive, κι έτσι είμαστε σίγουροι να χρησιμοποιήσουμε `jsp:expression` σε lower case.

Χρησιμοποίηση εκφράσεων ως τιμές ιδιοτήτων

Όπως θα δούμε κι αργότερα, η JSP περιλαμβάνει διάφορα στοιχεία που χρησιμοποιούν τη σύνταξη XML για να διευκρινίσουν τις διάφορες παραμέτρους. Παραδείγματος χάριν, το ακόλουθο παράδειγμα περνά "Marty" στη μέθοδο `setFirstName` του αντικειμένου που δεσμεύεται στη μεταβλητή `author`. Δεν ανησυχούμε στην περίπτωση την οποία δεν καταλαβαίνουμε τις λεπτομέρειες αυτού του κώδικα. Σκοπός μας εδώ είναι απλά να επισημάνουμε τη χρήση των ιδιοτήτων του *name*, *property*, και του *value*.

```
<jsp:setProperty name="author"

property="firstName"

value="Marty" />
```

Οι περισσότερες ιδιότητες απαιτούν την τιμή να είναι μια σταθερή συμβολοσειρά που εσωκλείεται είτε στα ενιαία είτε στα διπλά αποσπάσματα, όπως είδαμε και στο προηγούμενο

παράδειγμα. Μερικές ιδιότητες εντούτοις, μας επιτρέπουν να χρησιμοποιήσουμε μια έκφραση της JSP που είναι υπολογισμένη κατά τον χρόνο του αιτήματος. Η ιδιότητα *value* της `jsp:setProperty` είναι ένα τέτοιο παράδειγμα, κι έτσι ο ακόλουθος κώδικας είναι τέλεια επιτρεπτός:

```
<jsp:setProperty name="user"

property="id"

value='<%= "UserID" + Math.random() %>' />
```

3.4. Δηλώσεις της JSP

Μια δήλωση JSP μας αφήνει να καθορίσουμε τις μεθόδους ή τα πεδία τα οποία εισέρχονται στο κύριο σώμα της κλάσης Servlet, έξω από τη μέθοδο *_jspService* που καλείται από την *service* για να επεξεργαστεί το αίτημα. Μια δήλωση έχει την ακόλουθη μορφή:
`<%! Java Code %>`

Δεδομένου ότι οι δηλώσεις δεν παράγουν οποιαδήποτε έξοδο, χρησιμοποιούνται κανονικά από κοινού με τις εκφράσεις JSP ή τα scriptlets. Παραδείγματος χάριν, είναι εδώ ένα JSP fragment που τυπώνει τον αριθμό των φορών που η τρέχουσα σελίδα έχει ζητηθεί δεδομένου ότι ο Server τέθηκε σε έναρξη, ή την κλάση Servlet που άλλαξε και «ξαναφορτώθηκε». Ξανακαλούμε αυτά τα πολλαπλά αιτήματα του client στο ίδιο αποτέλεσμα servlet μόνο στα πολλαπλά νήματα που καλούν τη μέθοδο *service* μιας ενιαίας περίπτωσης servlet. Δεν οδηγούν τα αποτελέσματα στη δημιουργία πολλαπλών περιπτώσεων servlet εκτός από το ενδεχόμενο, όταν εφαρμόζει το servlet *SingleThreadModel*. Κατά συνέπεια, οι μεταβλητές περίπτωσης ή πεδία ενός servlet μοιράζονται από τα πολλαπλά αιτήματα κι η *accessCount* δεν είναι απαραίτητο να δηλωθεί ως `static` κάτωθι.

```
<%! private int accessCount = 0; %>

Accesses to page since server reboot:

<%= ++accessCount %>
```

Σύνταξη ειδικών δηλώσεων

Όπως και με τα scriptlets, εάν θέλουμε να χρησιμοποιήσουμε τους χαρακτήρες `%>`, εισάγουμε `%\>` αντ' αυτού. Τέλος, πρέπει να σημειώσουμε ότι η ισοδύναμη έκφραση XML της εντολής `<%! Code %>` είναι η εξής ακόλουθη:

```
<jsp:declaration>

Code

</jsp:declaration>
```

3.5. Προκαθορισμένες μεταβλητές

Για να απλοποιήσουμε τον κώδικα στις εκφράσεις JSP και τα scriptlets, εφοδιαζόμαστε με οκτώ αυτόματα καθορισμένες μεταβλητές, αποκαλούμενες μερικές φορές ως *implicit objects*. Δεδομένου ότι οι δηλώσεις JSP οδηγούν στον κώδικα που εμφανίζεται έξω από τη μέθοδο *_jspService*, αυτές οι μεταβλητές δεν είναι προσβάσιμες στις δηλώσεις. Οι διαθέσιμες μεταβλητές είναι οι *request*, *response*, *out*, *session*, *application*, *config*, *pageContext*, και *page*. Οι λεπτομέρειες για κάθε μια δίνονται κατωτέρω.

request

Αυτή η μεταβλητή είναι η *HttpServletRequest* που συνδέεται με το αίτημα που μας δίνει την πρόσβαση στις παραμέτρους αιτήματος, τον τύπο αιτήματος (π.χ., *GET* ή *POST*), και τα εισερχόμενα headers HTTP (π.χ., Cookies). Για να κυριολεκτήσουμε, εάν το πρωτόκολλο στο αίτημα είναι κάτι εκτός από αυτό του HTTP, η *request* επιτρέπεται για να είναι μια υποκλάση της *ServletRequest* εκτός από την *HttpServletRequest*. Εντούτοις, λίγοι, ενδεχομένως, JSP Servers υποστηρίζουν αυτά τα non-HTTP servlets.

response

Αυτή η μεταβλητή είναι η *HttpServletResponse* που συνδέεται με την απάντηση στον client. Σε αυτό το σημείο πρέπει να σημειώσουμε ότι αφού το ρεύμα εξόδου αποθηκεύεται κανονικά, είναι επιτρεπτό να τεθούν τα HTTP Status Codes και τα headers απάντησης στις σελίδες JSP, ακόμα κι αν ο καθορισμός των headers ή των status codes δεν επιτρέπεται στα servlets μόλις σταλεί οποιαδήποτε έξοδος στον client.

out

Αυτό είναι το *PrintWriter* που χρησιμοποιείται για να στείλει την έξοδο στον client. Εντούτοις, για να καταστήσει το αντικείμενο *response* χρήσιμο, αυτό είναι μια αποθηκευμένη έκδοση *Print-Writer* αποκαλούμενη *JspWriter*. Μπορούμε να ρυθμίσουμε το μέγεθος buffer μέσω της χρήσης της ιδιότητας *buffer* της οδηγίας *page*. Επίσης, σημειώνουμε ότι η *out* χρησιμοποιείται σχεδόν αποκλειστικά στα scriptlets, δεδομένου ότι οι εκφράσεις JSP τοποθετούνται αυτόματα στο ρεύμα εξόδου κι έτσι σπάνια πρέπει να αναφερθεί ρητά η *out*.

session

Αυτή η μεταβλητή είναι το αντικείμενο *HttpSession* που συνδέεται με το αίτημα. Υπενθυμίζουμε ότι οι σύνοδοι δημιουργούνται αυτόματα, κι έτσι αυτή η μεταβλητή είναι συνδεδεμένη ακόμα κι αν δεν υπάρχει καμία εισερχόμενη αναφορά συνόδου. Η μια εξαίρεση είναι εάν χρησιμοποιούμε την ιδιότητα *session* της οδηγίας *page* για να κλείσουμε τις συνόδους. Σε εκείνη την περίπτωση, προσπαθούμε να παραπέμψουμε τα μεταβλητά λάθη αιτίας *session* στο χρόνο κατά τον οποίο η σελίδα JSP μεταφράζεται σε ένα servlet.

application

Αυτή η μεταβλητή είναι το *ServletContext* όπως λαμβάνεται μέσω του `getServletConfig().getContext()`. Τα servlets κι οι σελίδες JSP μπορούν να αποθηκεύσουν τα επίμονα στοιχεία στο αντικείμενο *ServletContext* παρά στις μεταβλητές περίπτωσης. Το *ServletContext* έχει τις μεθόδους `setAttribute` και `getAttribute` που μας αφήνουν να αποθηκεύσουμε τα αυθαίρετα στοιχεία που συνδέονται με τα διευκρινισμένα κλειδιά. Η διαφορά μεταξύ της αποθήκευσης των στοιχείων στις μεταβλητές περίπτωσης και της αποθήκευσης τους στο *Servlet-Context* είναι ότι το *ServletContext* μοιράζεται από όλα τα servlets στη μηχανή servlet, ή στην Web εφαρμογή, εάν ο Server μας υποστηρίζει μια τέτοια ικανότητα.

config

Αυτή η μεταβλητή είναι το αντικείμενο *ServletConfig* για αυτήν την σελίδα.

pageContext

Η JSP εισήγαγε μια νέα κλάση αποκαλούμενη PageContext για να δώσει ένα ενιαίο σημείο της πρόσβασης σε πολλές από τις ιδιότητες των σελίδων και για να παρέχει μια κατάλληλη θέση για να αποθηκεύσει τα κοινά στοιχεία. Η μεταβλητή `pageContext` αποθηκεύει την τιμή του αντικειμένου *PageContext* που συνδέεται με την τρέχουσα σελίδα.

page

Αυτή η μεταβλητή είναι απλά ένα συνώνυμο του *this* και δεν είναι πολύ χρήσιμη στη γλώσσα προγραμματισμού της Java. Δημιουργήθηκε σαν κάτοχος θέσεων για το χρόνο όταν μπόρεσε να είναι η scripting γλώσσα κάτι εκτός από την Java.

3.6. Η οδηγία σελίδων JSP: Δόμηση παραχθέντων Servlets

3.6.1. Η ιδιότητα *import*

Η ιδιότητα *import* της οδηγίας *page* μας επιτρέπει να διευκρινίσουμε τα packages που πρέπει να εισαχθούν από το servlet στις σελίδες JSP οι οποίες μεταφράζονται. Εάν δεν διευκρινίζουμε ρητά οποιεσδήποτε κλάσεις που εισάγουν, το servlet εισάγει `java.lang.*`, `javax.servlet.*`, `javax.servlet.jsp.*`, `javax.servlet.http.*`, κι ενδεχομένως κάποιο αριθμό server-specific δηλωτικών. Θα ήταν λάθος να δοκιμάσουμε να γράψουμε τον κώδικα JSP που στηρίζεται σε οποιεσδήποτε server-specific κλάσεις που εισάγονται αυτόματα.
Η χρήση των ιδιοτήτων *import* λαμβάνει μια από τις ακόλουθες δύο μορφές:

```
<%@ page import="package.class" %>

<%@ page import="package.class1,...,package.classN" %>
```

Παραδείγματος χάριν, η ακόλουθη οδηγία δηλώνει ότι όλες οι κλάσεις στο package `java.util` πρέπει να είναι διαθέσιμες στη χρήση χωρίς ρητά προσδιοριστικά package.

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

Η ιδιότητα *import* είναι η μόνη ιδιότητα *page* που επιτρέπεται για να εμφανιστούν πολλαπλοί χρόνοι μέσα στο ίδιο έγγραφο. Αν κι οι οδηγίες *page* μπορούν να εμφανιστούν οπουδήποτε μέσα στο έγγραφο, είναι παραδοσιακό να τοποθετηθούν οι δηλώσεις `import` είτε κοντά στην κορυφή του εγγράφου είτε αμέσως πριν από την πρώτη θέση ότι το παραπεφθέν package χρησιμοποιείται.

Κατάλογοι για τις συνηθισμένες κλάσεις

Εάν εισάγουμε τις κλάσεις οι οποίες δεν είναι οποιεσδήποτε από τις τυποποιημένες της Java ή τα packages `javax.servlet`, πρέπει να είμαστε βέβαιοι ότι εκείνες οι κλάσεις έχουν εγκατασταθεί κατάλληλα στον Server μας. Ειδικότερα, οι περισσότεροι Servers που υποστηρίζουν το αυτόματο servlet «ξαναφορτώνοντας» δεν επιτρέπουν τις κλάσεις που είναι στους καταλόγους αυτόματου-ξαναφορτώματος που παραπέμπονται από τις JSP σελίδες. Αυτές οι κατάλληλες θέσεις που χρησιμοποιούνται για τις κλάσεις Servlet ποικίλλουν από Server σε Server, κι έτσι πρέπει να συμβουλευθούμε το documentation του Server μας για την οριστική καθοδήγηση.

3.6.2. Η ιδιότητα *contentType*

Η ιδιότητα *contentType* θέτει το header απάντησης *Content-Type*, δείχνοντας τον τύπο MIME του εγγράφου που στέλνεται στον client. Η χρήση των ιδιοτήτων *contentType* λαμβάνει μια από τις ακόλουθες δύο μορφές:

```
<%@ page contentType="MIME-Type" %>

<%@ page contentType="MIME-Type; charset=Character-Set" %>
```

Παραδείγματος χάριν, η οδηγία

```
<%@ page contentType="text/plain" %>
```

έχει την ίδια επίδραση με το scriptlet

```
<% response.setContentType("text/plain"); %>
```

Αντίθετα από τα κανονικά servlets, όπου ο προεπιλεγμένος τύπος MIME είναι text/plain, η προεπιλεγμένη τιμή για τις σελίδες JSP είναι text/html, με ένα σύνολο προεπιλεγμένων χαρακτήρων ISO-8859-1).

3.6.3. Η ιδιότητα *session*

Η ιδιότητα *session* ελέγχει εάν η σελίδα συμμετέχει ή όχι στις συνόδους HTTP. Η χρήση αυτής της ιδιότητας λαμβάνει μιας από τις ακόλουθες δύο μορφές:

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

```
<%@ page session="false" %>
```

Μια τιμή *true*, η οποία είναι κι η προεπιλεγμένη δείχνει ότι η προκαθορισμένη μεταβλητή *session*, του τύπου *HttpSession* πρέπει να δεσμευθεί στην υπάρχουσα σύνοδο εάν το ένα υπάρχει ειδάλλως, μια νέα σύνοδος πρέπει να δημιουργηθεί και να δεσμευθεί στη *session*. Μια τιμή *false* σημαίνει ότι καμία σύνοδος δεν θα χρησιμοποιηθεί αυτόματα και προσπαθεί να έχει πρόσβαση στη μεταβλητή *session* που θα οδηγήσει στα λάθη στο χρόνο κατά τον οποίο η σελίδα JSP μεταφράζεται μέσα σε ένα servlet.

3.6.4. Η ιδιότητα *buffer*

Η ιδιότητα *buffer* διευκρινίζει το μέγεθος του buffer που χρησιμοποιείται από την μεταβλητή *out*, η οποία είναι του τύπου *JspWriter*, που είναι μια υποκλάση του *PrintWriter*. Η χρήση αυτής της ιδιότητας λαμβάνει τη μια από δύο μορφές:

```
<%@ page buffer="sizekb" %>
```

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

Οι Servers μπορούν να χρησιμοποιήσουν ένα μεγαλύτερο buffer από αυτό που διευκρινίζουμε, αλλά όχι μικρότερο. Παραδείγματος χάριν, η εντολή *<%@ page buffer="32kb" %>* σημαίνει ότι η περιεκτικότητα σε έγγραφα πρέπει να αποθηκευθεί και να μην σταλεί στον client έως ότου έχουν συσσωρευτεί τουλάχιστον 32 kilobytes ή η σελίδα να έχει ολοκληρωθεί. Το προεπιλεγμένο μέγεθος του buffer καθορίζεται από τον Server, αλλά πρέπει να είναι τουλάχιστον 8 kilobytes. Πρέπει να είμαστε προσεκτικοί κατά την διάρκεια του τερματισμού της αποθήκευσης, κάνοντας έτσι να απαιτεί τα λήμματα JSP τα οποία θέτουν τα headers ή τα status codes για να εμφανιστούν στην κορυφή του αρχείου, πριν από οποιοδήποτε περιεχόμενο HTML.

3.6.5. Η ιδιότητα *autoflush*

Η ιδιότητα *autoflush* ελέγχει εάν ο buffer εξόδου πρέπει να καθαριστεί αυτόματα όταν είναι πλήρης ή εάν μια εξαίρεση πρέπει να αυξηθεί όταν το buffer υπερχειλίζει. Η χρήση αυτής της ιδιότητας λαμβάνει μιας από τις ακόλουθες δύο μορφές:

```
<%@ page autoflush="true" %>
```

```
<%-- Default --%>
```

```
<%@ page autoflush="false" %>
```

Μια τιμή *false* είναι μη-επιτρεπτή όταν επίσης, χρησιμοποιούμε *buffer="none"*.

3.6.6. Η ιδιότητα *extends*

Η ιδιότητα *extends* δείχνει την υπερκλάση του servlet που θα παραχθεί για τη σελίδα JSP και λαμβάνει την ακόλουθη μορφή:

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

Χρησιμοποιούμε αυτήν την ιδιότητα με την ακραία προσοχή δεδομένου ότι ο Server μπορεί να χρησιμοποιείται από μια υπερκλάση ήδη.

3.6.7. Η ιδιότητα *info*

Η ιδιότητα *info* καθορίζει μια συμβολοσειρά που μπορεί να ανακτηθεί από το servlet με τη βοήθεια της μεθόδου `getServletInfo`. Η χρήση της *info* λαμβάνει την ακόλουθη μορφή:

```
<%@ page info="Some Message" %>
```

3.6.8. Η ιδιότητα *errorPage*

Η ιδιότητα *errorPage* διευκρινίζει μια σελίδα JSP η οποία πρέπει να επεξεργαστεί οποιεσδήποτε εξαιρέσεις, όπως για παράδειγμα κάτι του τύπου `Throwable` που πετιούνται αλλά που δεν πιάνονται στην τρέχουσα σελίδα. Χρησιμοποιείται ως εξής:

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

Η εξαίρεση που ρίχνεται θα είναι αυτόματα διαθέσιμη στην οριζόμενη σελίδα λάθους με τη βοήθεια της μεταβλητής `exception`.

3.6.9. Η ιδιότητα *isErrorPage*

Η ιδιότητα *isErrorPage* δείχνει εάν η τρέχουσα σελίδα μπορεί ή όχι να ενεργήσει σαν σελίδα λάθους για μια άλλη σελίδα JSP. Η χρήση της *isErrorPage* λαμβάνει μια από τις ακόλουθες δύο μορφές:

```
<%@ page isErrorPage="true" %>
```

```
<%@ page isErrorPage="false" %> <%!-- Default --%>
```

3.6.10. Η ιδιότητα *language*

Σε κάποιο σημείο, η *language* ιδιότητα προορίζεται να διευκρινίσει την βαθύτερη γλώσσα προγραμματισμού που χρησιμοποιείται, όπως στο ακόλουθο παράδειγμα, το οποίο είναι:

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

Για τώρα, δεν συντρέχει κάποιος ιδιαίτερος λόγος να ενοχληθούμε με αυτήν την ιδιότητα δεδομένου ότι η Java είναι συγχρόνως κι η προεπιλογή κι η μόνη επιτρεπτή επιλογή.

3.6.11. Σύνταξη XML για τις οδηγίες

Η JSP μας επιτρέπει να χρησιμοποιήσουμε μια εναλλακτική XML-συμβατή σύνταξη για τις οδηγίες. Αυτά τα κατασκευάσματα λαμβάνουν την ακόλουθη μορφή:

```
<jsp:directive.directiveType attribute="value" />
```

Παραδείγματος χάριν, η ισοδύναμη σύνταξη της XML για την παρακάτω εντολή:

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

είναι

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

3.7. Συμπεριλαμβάνοντας αρχεία και Applets στα έγγραφα JSP

3.7.1. Συμπεριλαμβάνοντας αρχεία κατά τον χρόνο μετάφρασης της σελίδας

Χρησιμοποιούμε την οδηγία *include* για να συμπεριλάβουμε ένα αρχείο στο κύριο έγγραφο JSP στο χρόνο κατά τον οποίο το έγγραφο μεταφράζεται μέσα σε ένα servlet, που είναι χαρακτηριστικά η πρώτη φορά που προσπελαύνεται. Η σύνταξη της είναι η ακόλουθη:

```
<%@ include file="Relative URL" %>
```

Υπάρχουν δύο διακλαδώσεις του γεγονότος ότι το συμπεριλαμβανόμενο αρχείο έχει εισαχθεί στο χρόνο μετάφρασης της σελίδας, όχι σαν αίτημα του χρόνου, αλλά σαν *jsp:include*.

Κατ' αρχάς, συμπεριλαμβάνουμε το ίδιο το πραγματικό αρχείο, αντίθετα από με την *jsp:include*, όπου ο Server «τρέχει» τη σελίδα και παρεμβάλλει την έξοδό του. Αυτή η προσέγγιση σημαίνει ότι το συμπεριλαμβανόμενο αρχείο μπορεί να περιέχει τα κατασκευάσματα JSP, όπως κι οι δηλώσεις των πεδίων ή μεθόδων, που έχουν επιπτώσεις στην κύρια σελίδα συνολικά.

Δεύτερον, εάν το συμπεριλαμβανόμενο αρχείο αλλάζει, τότε όλα τα αρχεία JSP που το χρησιμοποιούν χρειάζεται να ενημερωθούν. Δυστυχώς, αν κι οι Servers επιτρέπονται για να υποστηρίξουν ένα μηχανισμό για την ανίχνευση όταν ένα συμπεριλαμβανόμενο αρχείο έχει αλλάξει, κι έπειτα επαναμεταγλωττίζεται το servlet, δεν απαιτούνται για να κάνουν κάτι τέτοιο. Στην πράξη, λίγοι Servers υποστηρίζουν αυτήν την ικανότητα.

Επιπλέον, δεν υπάρχει μια απλή και φορητή εντολή η οποία να "ξαναμεταφράζει αυτήν την σελίδα JSP τώρα". Αντ' αυτού, πρέπει να ενημερώσουμε την ημερομηνία τροποποίησης της σελίδας JSP. Μερικά λειτουργικά συστήματα έχουν τις εντολές οι οποίες ενημερώνουν την ημερομηνία τροποποίησης χωρίς να χρειαστεί να επεξεργαστούμε το αρχείο (π.χ., η εντολή *touch* σε Unix), αλλά μια απλή φορητή εναλλακτική λύση είναι να συμπεριληφθεί ένα σχόλιο JSP στην κύρια σελίδα. Ενημερώνουμε το σχόλιο όποτε το συμπεριλαμβανόμενο αρχείο αλλάζει. Παραδείγματος χάριν, να βάλουμε την ημερομηνία τροποποίησης του συμπεριλαμβανόμενου αρχείου σε σχόλιο, όπως στο παράδειγμα που ακολουθεί.

```
<%-- Navbar.jsp modified 3/1/00 --%>

<%@ include file="Navbar.jsp" %>
```

3.7.2. Συμπεριλαμβάνοντας αρχεία στο χρόνο αίτησης

Η οδηγία *include* μας επιτρέπει να συμπεριλάβουμε τα έγγραφα τα οποία περιέχουν τον κώδικα JSP στις πολλαπλές διαφορετικές σελίδες. Συμπεριλαμβάνοντας το περιεχόμενο της JSP είναι μια αρκετά χρήσιμη ικανότητα, αλλά η *include* οδηγία απαιτεί από εμάς να ενημερώσουμε την ημερομηνία τροποποίησης της σελίδας όποτε το συμπεριλαμβανόμενο αρχείο αλλάζει, το οποίο είναι μια σημαντική δυσχέρεια. Η δράση της *jsp:include* περιλαμβάνει τα αρχεία στον χρόνο κατά τον οποίο ο client αιτείται κι έτσι δεν απαιτεί από εμάς να ενημερώσουμε το κύριο αρχείο όταν αλλάζει ένα συμπεριλαμβανόμενο αρχείο. Αφ' ετέρου, η σελίδα έχει μεταφραστεί ήδη σε ένα servlet μέχρι το χρόνο αιτήματος, κι έτσι τα συμπεριλαμβανόμενα αρχεία δεν μπορούν να περιέχουν JSP.

Αν και τα συμπεριλαμβανόμενα αρχεία δεν μπορούν να περιέχουν JSP, μπορούν όμως να είναι το αποτέλεσμα των πόρων οι οποίοι χρησιμοποιούν JSP για να δημιουργήσουν την έξοδο. Δηλαδή το URL που αναφέρεται στο συμπεριλαμβανόμενο πόρο ερμηνεύεται με τον

κανονικό τρόπο από τον Server και μπορεί έτσι να είναι ένα servlet ή μια σελίδα JSP. Αυτή ακριβώς είναι η συμπεριφορά της μεθόδου *include* της κλάσης RequestDispatcher, η οποία είναι αυτό που τα servlets που χρησιμοποιούν εάν θέλουν να κάνουν αυτόν τον τύπο συνυπολογισμού αρχείων. Το στοιχείο `jsp:include` έχει δύο απαραίτητες ιδιότητες, όπως φαίνεται και στο παρακάτω παράδειγμα: *page*, ένα σχετικό URL που παραπέμπει στο αρχείο που θα συμπεριληφθεί και *flush*, που πρέπει να έχει την τιμή *true*.

```
<jsp:include page="Relative URL" flush="true" />
```

Αν συμπεριλαμβάνουμε χαρακτηριστικά HTML ή plain text έγγραφα, δεν υπάρχει καμία απαίτηση ότι τα συμπεριλαμβανόμενα αρχεία έχουν οποιαδήποτε κατάλληλη επέκταση αρχείων. Εντούτοις, ο Java Web Server 2.0 έχει ένα bug το οποίο τον προκαλεί να τερματίσει την επεξεργασία των σελίδων όταν προσπαθεί να συμπεριλάβει ένα αρχείο που δεν έχει μια .html ή .htm επέκταση, όπως για παράδειγμα somefile.txt. Το Tomcat, το JSWDK, κι οι περισσότεροι εμπορικοί Servers δεν έχουν κανέναν τέτοιο περιορισμό.

Οι υπεύθυνοι για την ανάπτυξη σελίδων μπορούν να αλλάξουν τα νέα αντικείμενα στα αρχεία Item1.html μέσω του Item4.html χωρίς να πρέπει να ενημερωθεί η κύρια σελίδα ειδήσεων.

3.7.3. Συμπεριλαμβάνοντας Applets για το Java Plug-In

Με την JSP, δεν χρειαζόμαστε οποιαδήποτε ειδική σύνταξη για να συμπεριλάβουμε τα συνηθισμένα applets: απλώς χρησιμοποιούμε την κανονική ετικέτα HTML *APPLET*. Εντούτοις, αυτά τα applets πρέπει να χρησιμοποιήσουν την έκδοση JDK 1.1 ή την έκδοση JDK 1.02, δεδομένου ότι ούτε οι εκδόσεις Netscape 4.x, αλλά ούτε κι οι εκδόσεις του Internet Explorer 5.x υποστηρίζουν την Java 2 platform, δηλαδή την έκδοση JDK 1.2. Αυτή η έλλειψη υποστήριξης επιβάλλει διάφορους περιορισμούς στα applets:

- Προκειμένου να χρησιμοποιήσουμε την *Swing*, πρέπει να στείλουμε τα αρχεία Swing πέρα από το δίκτυο. Αυτή η διαδικασία είναι μεγάλη σπατάλη χρόνου κι αποτυγχάνει στον Internet Explorer 3 και τις εκδόσεις του Netscape 3.x και 4.01-4.05, που υποστηρίζουν μόνο την JDK 1.02, δεδομένου ότι η *Swing* εξαρτάται από την JDK 1.1.

- Δεν μπορούμε να χρησιμοποιήσουμε την Java 2D.

- Δεν μπορούμε να χρησιμοποιήσουμε την Java 2 συλλογή package.

- Ο κώδικάς μας εκτελείται πιο αργά, δεδομένου ότι οι περισσότεροι μεταγλωττιστές για την Java 2 platform βελτιώνονται σημαντικά πέρα από τους 1.1 προκατόχους τους.

Επιπλέον, οι πρόωρες καταβολές των browser είχαν διάφορες ασυνέπειες με τον τρόπο που υποστήριξαν τα διάφορα συστατικά AWT, κάνοντας τη δοκιμή και την παράδοση των σύνθετων διασυνδέσεων των χρηστών περισσότερο φορτική διαδικασία από ότι οφείλει να είναι. Για να εξετάσει αυτό το πρόβλημα, η Sun ανέπτυξε ένα plug-in browser για τον Netscape και τον Internet Explorer που μας επιτρέπει να χρησιμοποιήσουμε την Java 2 platform για τα applets σε ποικίλους browsers. Αυτό το plug-in είναι διαθέσιμο στην ηλεκτρονική διεύθυνση http://java.sun.com/products /plugin/, κι έρχεται επίσης, συσσωρευμένο με την JDK 1.2.2 κι αργότερα. Δεδομένου ότι το plug-in είναι αρκετά μεγάλο, κάποια MBytes, δεν είναι λογικό να αναμένει από τους χρήστες στο WWW σε μεγάλο για να το μεταφορτώσει και να το εγκαταστήσει, μόνο και μόνο για να εκτελέσουμε τα applets μας. Αφ' ετέρου, είναι μια λογική εναλλακτική λύση για τα γρήγορα εταιρικά intranets, ειδικά δεδομένου των applets τα οποία μπορούν αυτόματα να προτρέψουν τους browsers που στερούνται αυτό το plug-in να το μεταφορτώσουν.

Δυστυχώς εντούτοις, η κανονική ετικέτα *APPLET* δεν θα λειτουργήσει με το plug-in, δεδομένου ότι οι browsers έχουν σαν σκοπό συγκεκριμένα να χρησιμοποιήσουν μόνο την ενσωματωμένη εικονική μηχανή τους όταν βλέπουν *APPLET*. Αντ' αυτού, πρέπει να χρησιμοποιήσουμε μια μακρινή κι ακατάστατη ετικέτα *OBJECT* για τον Internet Explorer κι εξίσου μια μακρινή ετικέτα *EMBED* για τον Netscape. Επιπλέον, δεδομένου ότι χαρακτηριστικά δεν ξέρουμε ποιος τύπος browser θα έχει πρόσβαση στη σελίδα μας, πρέπει είτε να συμπεριλάβουμε και *OBJECT* και *EMBED*, τοποθετώντας την *EMBED* μέσα στο τμήμα *COMMENT* του *OBJECT*, είτε να προσδιορίσουμε τον τύπο του browser στον χρόνο του αιτήματος και να χτίσουμε υπό όρους τη σωστή ετικέτα. Αυτή η διαδικασία είναι απλή, αλλά κουραστική και σπατάλη χρόνου. Το στοιχείο *jsp:plugin* καθοδηγεί τον Server να χτίσει μια ετικέτα κατάλληλη για τα applets που χρησιμοποιούν το plug-in. Οι Servers επιτρέπονται κάποια παρέκκλιση ακριβώς πώς εφαρμόζουν αυτήν την υποστήριξη, αλλά οι περισσότεροι απλώς συμπεριλαμβάνουν και *OBJECT* και *EMBED*.

Το στοιχείο *jsp:plugin*

Ο απλούστερος τρόπος για να χρησιμοποιηθεί η *jsp:plugin* είναι να παρασχεθούν τέσσερις ιδιότητες: η *type*, η *code*, η *width*, κι η *height*. Παρέχουμε μια τιμή του applet για την ιδιότητα *type* και χρησιμοποιούμε τις άλλες τρεις ιδιότητες με ακριβώς τον ίδιο τρόπο όπως με το στοιχείο *APPLET*, με δύο εξαιρέσεις: τα ονόματα των ιδιοτήτων είναι case sensitive, και τα ενιαία ή τα διπλά αποσπάσματα απαιτούνται πάντα γύρω από τις τιμές των ιδιοτήτων. Έτσι, παραδείγματος χάριν, θα μπορούσαμε να αντικαταστήσουμε τον παρακάτω κώδικα:

```
<APPLET CODE="MyApplet.class"

WIDTH=475 HEIGHT=350>

</APPLET>
```

με τον ακόλουθο κώδικα:

```
<jsp:plugin type="applet"

code="MyApplet.class"

width="475" height="350">

</jsp:plugin>
```

Το στοιχείο *jsp:plugin* έχει διάφορες άλλες προαιρετικές ιδιότητες. Οι περισσότερες, αλλά όχι όλες, παράλληλες ιδιότητες του στοιχείου *APPLET*. Παρακάτω παρουσιάζεται ένας πλήρης κατάλογος.

type

Για τα applets, αυτή η ιδιότητα πρέπει να έχει μια τιμή του *applet*. Εντούτοις, το Java Plug-In μας επιτρέπει επίσης, να ενσωματώσουμε τα στοιχεία *JavaBeans* στην ιστοσελίδα. Σε αυτήν την περίπτωση χρησιμοποιούμε μια τιμή *bean*.

code

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην ιδιότητα *CODE* του *APPLET*, διευκρινίζοντας το κορυφαίο αρχείο κλάσης του *applet* το οποίο επεκτείνει *Applet* ή *JApplet*. Σε αυτό το σημείο ακριβώς πρέπει να θυμηθούμε ότι το όνομα *code* πρέπει να είναι σε lower case με την *jsp:plugin*, δεδομένου ότι ακολουθεί τη σύνταξη XML, ενώ με την *APPLET*, η περίπτωση δεν ενόχλησε, δεδομένου ότι τα ονόματα ιδιοτήτων της HTML δεν είναι ποτέ case sensitive.

width

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην ιδιότητα *WIDTH* του *APPLET*, διευκρινίζοντας το πλάτος σε pixels που διατηρούνται για το applet. Σε αυτό το σημείο πρέπει απλώς να θυμηθούμε ότι πρέπει να εσωκλείσουμε την τιμή στα ενιαία ή στα διπλά αποσπάσματα.

height

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην ιδιότητα *HEIGHT* του *APPLET*, διευκρινίζοντας το ύψος σε pixels που διατηρούνται για το applet. Σε αυτό το σημείο ακριβώς είναι αξιοσημείωτο να θυμηθούμε ότι πρέπει να εσωκλείουμε την τιμή στα ενιαία ή στα διπλά αποσπάσματα.

codebase

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην ιδιότητα *CODEBASE* του *APPLET*, διευκρινίζοντας τον κατάλογο βάσεων για τα applets. Η ιδιότητα *code* ερμηνεύεται σχετικά με αυτόν τον κατάλογο. Όπως με το στοιχείο *APPLET*, εάν παραλείψουμε αυτήν την ιδιότητα, ο κατάλογος της τρέχουσας σελίδας χρησιμοποιείται σαν προεπιλογή. Στην περίπτωση της JSP, αυτή η προεπιλεγμένη θέση είναι ο κατάλογος όπου το αρχικό αρχείο JSP κατοίκησε, κι όχι η system-specific θέση του servlet που προκύπτει από το αρχείο JSP.

align

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην *ALIGN* ιδιότητα του *APPLET* και του *IMG*, διευκρινίζοντας την ευθυγράμμιση του applet μέσα στην ιστοσελίδα. Οι επιτρεπτές τιμές είναι η *left*, η *right*, η *top*, η *bottom*, κι η *middle*. Με την *jsp:plugin*, σε καμία περίπτωση δεν πρέπει να ξεχάσουμε να συμπεριλάβουμε αυτές τις τιμές στα ενιαία ή στα διπλά αποσπάσματα, ακόμα κι αν τα αποσπάσματα είναι προαιρετικά για την *APPLET* και την *IMG*.

hspace

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην ιδιότητα *HSPACE* του *APPLET*, διευκρινίζοντας το κενό διάστημα στα pixels που διατηρούνται στο αριστερό και το δεξιό τμήμα του applet. Σε αυτό το σημείο ακριβώς είναι αξιοσημείωτο ότι πρέπει να θυμηθούμε να εσωκλείουμε την τιμή στα ενιαία ή στα διπλά αποσπάσματα.

vspace

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην ιδιότητα *VSPACE* του *APPLET*, διευκρινίζοντας το κενό διάστημα στα pixels τα οποία διατηρούνται στην κορυφή και το κατώτατο σημείο του applet. Σε αυτό το σημείο ακριβώς είναι αξιοσημείωτο ότι πρέπει να θυμηθούμε να εσωκλείουμε την τιμή στα ενιαία ή στα διπλά αποσπάσματα.

archive

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην ιδιότητα *ARCHIVE* του *APPLET*, διευκρινίζοντας ένα αρχείο JAR από το οποίο οι κλάσεις κι οι εικόνες πρέπει να «φορτωθούν».

name

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην ιδιότητα *NAME* του *APPLET*, διευκρινίζοντας ένα όνομα που χρησιμοποιείται για την διά-applet επικοινωνία ή για τον προσδιορισμό του applet στις scripting γλώσσες όπως η JavaScript.

title

Αυτή η ιδιότητα χρησιμοποιείται όμοια στην πολύ σπάνια χρησιμοποιούμενη ιδιότητα *TITLE* του *APPLET*, κι ουσιαστικά όλα τα άλλα στοιχεία της HTML στην έκδοση HTML 4.0, που

διευκρινίζει ένα τίτλο που θα μπορούσε να χρησιμοποιηθεί για ένα tool-tip ή για την ευρετηρίαση.

jreversion

Αυτή η ιδιότητα προσδιορίζει την έκδοση του Runtime Environment (JRE) της Java που απαιτείται. Η προεπιλεγμένη έκδοση είναι η 1.1.

iepluginurl

Αυτή η ιδιότητα υποδεικνύει ένα URL από το οποίο η σύνδεση για τον Internet Explorer μπορεί να μεταφορτωθεί. Οι χρήστες που δεν έχουν ήδη εγκατεστημένη τη σύνδεση θα προτραπούν να το μεταφορτώσουν από αυτήν την θέση. Η προκαθορισμένη τιμή θα κατευθύνει το χρήστη στο Site της Sun, αλλά για τη χρήση ενδοδικτύου που ίσως θελήσουμε για να κατευθύνουμε το χρήστη σε ένα τοπικό αντίγραφο.

nspluginurl

Αυτή η ιδιότητα υποδεικνύει ένα URL από το οποίο η σύνδεση για τον Netscape μπορεί να μεταφορτωθεί. Η προκαθορισμένη τιμή θα κατευθύνει το χρήστη στο Site της Sun, αλλά για τη χρήση ενδοδικτύου που ίσως θελήσουμε για να κατευθύνουμε το χρήστη σε ένα τοπικό αντίγραφο.

Τα στοιχεία *jsp:param* και *jsp:params*

Το στοιχείο *jsp:param* χρησιμοποιείται με την *jsp:plugin* κατά τρόπο παρόμοιο με τον τρόπο ότι η *PARAM* χρησιμοποιείται με την *APPLET*, διευκρινίζοντας ένα όνομα και μια τιμή που προσεγγίζονται από μέσα από το applet από την *getParameter*. Υπάρχουν δύο κύριες διαφορές, εντούτοις. Κατ' αρχάς, δεδομένου ότι η *jsp:param* ακολουθεί τη σύνταξη XML, τα ονόματα ιδιοτήτων πρέπει να είναι lower case, οι τιμές ιδιοτήτων πρέπει να εσωκλείονται στα ενιαία ή στα διπλά αποσπάσματα, και το στοιχείο πρέπει να τελειώσει με */>*, κι όχι μόνο με *>*. Δεύτερον, όλες οι καταχωρήσεις *jsp:param* πρέπει να εσωκλειστούν μέσα σε ένα στοιχείο *jsp:params*. Έτσι, παραδείγματος χάριν, θα αντικαθιστούσαμε τον παρακάτω κώδικα:

```
<APPLET CODE="MyApplet.class"

WIDTH=475 HEIGHT=350>

<PARAM NAME="PARAM1" VALUE="VALUE1">

<PARAM NAME="PARAM2" VALUE="VALUE2">

</APPLET>
```

με τον ακόλουθο κώδικα:

```
<jsp:plugin type="applet"

code="MyApplet.class"

width="475" height="350">

<jsp:params>

<jsp:param name="PARAM1" value="VALUE1" />

<jsp:param name="PARAM2" value="VALUE2" />

</jsp:params>

</jsp:plugin>
```

Το στοιχείο *jsp:fallback*

Το στοιχείο *jsp:fallback* παρέχει το εναλλακτικό κείμενο στους browsers που δεν υποστηρίζουν την *OBJECT* ή την *EMBED*. Χρησιμοποιούμε αυτό το στοιχείο με σχεδόν τον ίδιο τρόπο όπως θα χρησιμοποιούσαμε το εναλλακτικό κείμενο το οποίο τοποθετείται μέσα σε ένα στοιχείο *APPLET*. Έτσι, παραδείγματος χάριν, θα αντικαθιστούσαμε τον παρακάτω κώδικα:

```
<APPLET CODE="MyApplet.class"

WIDTH=475 HEIGHT=350>

<B>Error: this example requires Java.</B>

</APPLET>
```

με τον ακόλουθο κώδικα:

```
<jsp:plugin type="applet"

code="MyApplet.class"

width="475" height="350">

<jsp:fallback>

<B>Error: this example requires Java.</B>

</jsp:fallback>

</jsp:plugin>
```

Εντούτοις, πρέπει να σημειώσουμε ότι ο Java Web Server 2.0 έχει ένα bug που τον αναγκάζει να αποτύχει κατά το μετάφραση των σελίδων που περιλαμβάνουν τα στοιχεία *jsp:fallback*. Το Tomcat, το JSWDK, κι οι περισσότεροι εμπορικοί Servers χειρίζονται την *jsp:fallback* κατάλληλα.

3.8. Χρησιμοποιώντας JavaBeans με την JSP

3.8.1. Βασική χρήση των Beans

Η δράση της *jsp:useBean* μας επιτρέπει να «φορτώσουμε» ένα bean που χρησιμοποιείται στη σελίδα JSP. Τα beans παρέχουν μια πολύ χρήσιμη ικανότητα επειδή μας αφήνουν να εκμεταλλευτούμε την ικανότητα επαναχρησιμοποίησης των κλάσεων της Java χωρίς θυσία της ευκολίας που η JSP προσθέτει πέρα από τα servlets μόνο.
Η απλούστερη σύνταξη για τη διευκρίνιση ότι ένα bean πρέπει να χρησιμοποιηθεί είναι:

```
<jsp:useBean id="name" class="package.Class" />
```

Αυτό σημαίνει συνήθως "κάνουμε τρέχον ένα αντικείμενο της κλάσης που διευκρινίζεται από την *Class*, και το δεσμεύουμε σε μια μεταβλητή με το όνομα που διευκρινίζεται από το *id*". Έτσι, παραδείγματος χάριν, η δράση της JSP:

```
<jsp:useBean id="book1" class="coreservlets.Book" />
```

μπορεί κανονικά να θεωρηθεί όπως ισοδύναμη με το scriptlet:

```
<% coreservlets.Book book1 = new coreservlets.Book(); %>
```

Αν κι είναι κατάλληλο να σκεφτούμε *jsp:useBean* σαν ισοδύναμο με την οικοδόμηση ενός αντικειμένου, η *jsp:useBean* έχει τις πρόσθετες επιλογές που τη καθιστούν ισχυρότερη. Εάν τα beans μπορούν να μοιραστούν, είναι χρήσιμο να ληφθούν οι αναφορές στα υπάρχοντα beans, κι έτσι η δράση της *jsp:useBean* διευκρινίζει ότι ένα νέο αντικείμενο επείγεται μόνο εάν δεν υπάρχει κανένας υπάρχων με το ίδιο *id* και *scope*. Παρά τη χρησιμοποίηση της ιδιότητας *class*, επιτρεπόμαστε να χρησιμοποιήσουμε *beanName* αντ' αυτού. Η διαφορά είναι ότι η *beanName* μπορεί να αναφερθεί είτε σε μια κλάση είτε σε ένα αρχείο που περιέχει ένα δημοσιευμένο σε συνέχειες αντικείμενο *bean*. Η τιμή της ιδιότητας *beanName* περνάει στην μέθοδο *instantiate* του *java.beans.Bean*.

Στις περισσότερες περιπτώσεις, επιθυμούμε την τοπική μεταβλητή για να έχουμε τον ίδιο τύπο με το αντικείμενο που δημιουργείται. Σε μερικές περιπτώσεις εντούτοις, που ίσως θελήσουμε τη μεταβλητή για να δηλωθεί για να έχουμε ένα τύπο που είναι υπερκλάση του πραγματικού τύπου *bean* ή είναι μια διεπαφή που το bean εφαρμόζει. Σε αυτό το σημείο δύναται να χρησιμοποιήσουμε την ιδιότητα *type* για να το ελέγξουμε, όπως στο ακόλουθο παράδειγμα:

```
<jsp:useBean id="thread1" class="MyClass" type="Runnable" />
```

Αυτή η χρήση οδηγεί σε παρόμοιο κώδικα με τα εξής που παρεμβάλλονται στη μέθοδο *_jspService*:

```
Runnable thread1 = new MyClass();
```

Πρέπει να σημειώσουμε ότι αφού η *jsp:useBean* χρησιμοποιεί τη σύνταξη XML, το format διαφέρει με τρεις τρόπους από τη σύνταξη HTML: τα ονόματα των ιδιοτήτων είναι case sensitive, είτε τα ενιαία είτε τα διπλά αποσπάσματα μπορούν να χρησιμοποιηθούν, αλλά το ένα ή το άλλο πρέπει να χρησιμοποιηθεί, και το τέλος της ετικέτας είναι χαρακτηρισμένο με */>*, κι όχι μόνο *>*. Οι πρώτες δύο συντακτικές διαφορές ισχύουν για όλα τα στοιχεία JSP που μοιάζουν με το *jsp:xxx*. Η τρίτη διαφορά ισχύει εκτός αν το στοιχείο είναι ένας container με μια χωριστή ετικέτα έναρξης και τέλους.

Υπάρχουν επίσης, μερικές ακολουθίες χαρακτήρων που απαιτούν τον ειδικό χειρισμό προκειμένου να εμφανιστούν οι εσωτερικές τιμές ιδιοτήτων:

- Για να πάρουμε *'* μέσα σε μια τιμή ιδιότητας, χρησιμοποιούμε *\'*
- Για να πάρουμε " μέσα σε μια τιμή ιδιότητας, χρησιμοποιούμε \"
- Για να πάρουμε \ μέσα σε μια τιμή ιδιότητας, χρησιμοποιούμε \\
- Για να πάρουμε %> μέσα σε μια τιμή ιδιότητας, χρησιμοποιούμε %\>
- Για να πάρουμε <% μέσα σε μια τιμή ιδιότητας, χρησιμοποιούμε <\%

Πρόσβαση των ιδιοτήτων beans

Μόλις έχουμε ένα bean, μπορούμε να έχουμε πρόσβαση στις ιδιότητές του με την *jsp:getProperty*, η οποία παίρνει μια ιδιότητα *name* που πρέπει να ταιριάζει με το *id* που δίνεται στην *jsp:useBean* κι μια ιδιότητα *property* η οποία ονομάζει την ιδιότητα ενδιαφέροντος. Εναλλακτικά, θα μπορούσαμε να χρησιμοποιήσουμε μια έκφραση JSP και να καλέσουμε ρητά μια μέθοδο στο αντικείμενο που διευκρινίζει το μεταβλητό όνομα με τις ιδιότητες *id*. Παραδείγματος χάριν, ας υποθέσουμε ότι η κλάση Book έχει μια ιδιότητα *String* η οποία ονομάζεται *title* κι ότι έχουμε δημιουργήσει μια περίπτωση που καλείται *book1* με τη χρησιμοποίηση του παραδείγματος *jsp:useBean* που μόλις δόθηκε, θα μπορούσαμε να

εισάγουμε την τιμή της ιδιότητας *title* μέσα στη σελίδα JSP με καθέναν των ακόλουθων δύο τρόπων:

```
<jsp:getProperty name="book1" property="title" />
```

```
<%= book1.getTitle() %>
```

Η πρώτη προσέγγιση είναι προτιμητέα σε αυτήν την περίπτωση, δεδομένου ότι η σύνταξη είναι πιο προσιτή στους σχεδιαστές ιστοσελίδας που δεν εξοικειώνονται με τη γλώσσα προγραμματισμού της Java. Εντούτοις, η άμεση πρόσβαση στη μεταβλητή είναι χρήσιμη όταν χρησιμοποιούμε τους βρόχους, τις υπό όρους δηλώσεις, και τις μεθόδους που δεν αντιπροσωπεύονται ως ιδιότητες.

Εάν δεν εξοικειωνόμαστε με την έννοια των ιδιοτήτων του bean, η τυποποιημένη ερμηνεία της δήλωσης "αυτό το bean έχει μια ιδιότητα του τύπου *T* που ονομάζεται *foo*" είναι "αυτή η κλάση έχει μια μέθοδο που ονομάζεται *getFoo* η οποία επιστρέφει κάτι του τύπου *T* και καλεί μια άλλη μέθοδο *setFoo* που παίρνει ένα *T* σαν όρισμα και το αποθηκεύει για την πιο πρόσφατη πρόσβαση από την *getFoo*".

Θέτοντας ιδιότητες των beans: Απλή περίπτωση

Για να τροποποιήσουμε τις ιδιότητες του bean, χρησιμοποιούμε κανονικά την `jsp:setProperty`. Αυτή η δράση έχει πολλές και διαφορετικές φόρμες, αλλά με την απλούστερη μορφή παρέχουμε ακριβώς τρεις ιδιότητες: την *name*, που πρέπει να ταιριάζει με το *id* που δίνεται από την `jsp:useBean`, την *property*, όπου το όνομα της ιδιότητας για αλλαγή, και την *value*, που είναι η νέα τιμή.

Μια εναλλακτική λύση της χρησιμοποίησης της δράσης `jsp:setProperty` είναι να χρησιμοποιηθεί ένα scriptlet που να καλεί ρητά τις μεθόδους στο αντικείμενο bean. Παραδείγματος χάριν, λαμβάνοντας υπόψη το αντικείμενο *book1* που παρουσιάστηκε παραπάνω, θα μπορούσαμε να χρησιμοποιήσουμε μια από τις ακόλουθες δύο μορφές για να τροποποιήσουμε την ιδιότητα *title*:

```
<jsp:setProperty name="book1"

property="title"

value="Core Servlets and JavaServer Pages" />
```

```
<% book1.setTitle("Core Servlets and JavaServer Pages"); %>
```

Η χρησιμοποίηση της `jsp:setProperty` έχει το πλεονέκτημα ότι είναι πιο προσιτή στον μη-προγραμματιστή, αλλά η άμεση πρόσβαση στο αντικείμενο μας επιτρέπει να εκτελέσουμε τις πιο σύνθετες διαδικασίες όπως η ρύθμιση της τιμής υπό όρους ή η κλήση των μεθόδων εκτός από τις `getXxx` ή τις `setXxx` στο αντικείμενο.

3.8.2. Θέτοντας ιδιότητες των Beans

Χρησιμοποιούμε κανονικά την `jsp:setProperty` για να ρυθμίσουμε τις ιδιότητες του bean. Η απλούστερη μορφή αυτής της δράσης παίρνει τρεις ιδιότητες: την *name*, που πρέπει να ταιριάζει με το *id* που δίνεται από την `jsp:useBean`, την *property*, που είναι το όνομα της ιδιότητας στην αλλαγή, και την *value* που είναι η νέα τιμή.

Μόλις το bean επείγεται, η χρησιμοποίηση μιας παραμέτρου εισόδου για να ρυθμίσει το *itemID* είναι απλή, όπως παρουσιάζεται και στο ακόλουθο παράδειγμα:

```
<jsp:setProperty
```

```
name="entry"

property="itemID"

value='<%= request.getParameter("itemID") %>' />
```

Σε αυτό το σημείο πρέπει να παρατηρηθεί ότι χρησιμοποιήθηκε μια έκφραση JSP για την παράμετρο *value*. Οι περισσότερες τιμές των ιδιοτήτων της JSP πρέπει να είναι σταθερές συμβολοσειρές, αλλά οι ιδιότητες *value* και *name* της `jsp:setProperty` επιτρέπονται για να είναι αίτημα-χρονικές εκφράσεις. Εάν η έκφραση χρησιμοποιεί τα διπλά αποσπάσματα εσωτερικά, υπενθυμίζουμε ότι τα ενιαία αποσπάσματα μπορούν να χρησιμοποιηθούν αντί των διπλών αποσπασμάτων γύρω από τις τιμές ιδιοτήτων κι ότι το `\'` καθώς επίσης, και το `\"` μπορούν να χρησιμοποιηθούν για να αντιπροσωπεύσουν τα ενιαία ή τα διπλά αποσπάσματα μέσα σε μια τιμή μιας ιδιότητας.

Ένωση των μεμονωμένων ιδιοτήτων με τις παραμέτρους εισόδου

Ο καθορισμός της ιδιότητας *itemID* ήταν εύκολος δεδομένου ότι η τιμή της είναι μια συμβολοσειρά. Ο καθορισμός των ιδιοτήτων *numItems* και *discountCode* είναι λίγο πιο προβληματικός δεδομένου ότι οι τιμές τους πρέπει να είναι αριθμοί κι η *getParameter* επιστρέφει μια συμβολοσειρά. Εδώ είναι ο κάπως δυσκίνητος κώδικας που απαιτείται για να θέσει την *numItems*:

```
<%

int numItemsOrdered = 1;

try {

numItemsOrdered =

Integer.parseInt(request.getParameter("numItems"));

} catch(NumberFormatException nfe) {}

%>

<jsp:setProperty

name="entry"

property="numItems"

value="<%= numItemsOrdered %>" />
```

Ευτυχώς, η JSP έχει μια συμπαθητική λύση σε αυτό το πρόβλημα που μας αφήνει να συνδέσουμε μια ιδιότητα με μια παράμετρο αιτήματος και που εκτελεί αυτόματα τη μετατροπή τύπων από τις συμβολοσειρές στους αριθμούς, τους χαρακτήρες, και τις Boolean τιμές. Αντί της χρησιμοποίησης των ιδιοτήτων *value*, χρησιμοποιούμε την *param* για να ονομάσουμε μια παράμετρο εισόδου. Η τιμή αυτής της παραμέτρου χρησιμοποιείται αυτόματα σαν τιμή της ιδιότητας, κι οι απλές μετατροπές τύπων εκτελούνται αυτόματα. Εάν η διευκρινισμένη παράμετρος εισόδου λείπει από το αίτημα, κανένα μέτρο δεν λαμβάνεται, όπου το σύστημα δεν περνά `null` στη σχετική ιδιότητα. Έτσι, παραδείγματος χάριν, θέτοντας την ιδιότητα `numItems` μπορεί να απλοποιηθεί σε:

```
<jsp:setProperty

name="entry"

property="numItems"
```

```
param="numItems" />
```

Ένωση όλων των ιδιοτήτων με τις παραμέτρους εισόδου

Η ένωση μιας ιδιότητας με μια παράμετρο εισόδου μας σώζει από την ενόχληση της εκτέλεσης των μετατροπών για πολλούς από τους απλούς ενσωματωμένους τύπους. Η JSP μας αφήνει να πάρουμε τη διαδικασία ένα βήμα περαιτέρω με την ένωση όλων των ιδιοτήτων με τις όμοια ονομασμένες παραμέτρους εισόδου. Το μόνο που πρέπει να κάνουμε είναι να παράσχουμε *"*"* για την παράμετρο *property*.

```
<jsp:setProperty name="entry" property="*" />
```

Αν κι αυτή η προσέγγιση είναι απλή, τέσσερις μικρές προειδοποιήσεις είναι στη διαταγή. Κατ' αρχάς, όπως με τις χωριστά σχετικές ιδιότητες, κανένα μέτρο δεν λαμβάνεται όταν λείπει μια παράμετρος εισόδου. Ειδικότερα, το σύστημα δεν παρέχει *null* σαν τιμή ιδιότητας.

Δεύτερον, το JSWDK κι ο Java Web Server κι οι δύο αποτυγχάνουν για τις μετατροπές στις ιδιότητες που αναμένουν *double* τιμές. Τρίτον, η αυτόματη μετατροπή τύπων δεν φρουρεί ενάντια στις παράνομες τιμές τόσο αποτελεσματικά όπως κάνει ο χειροκίνητος τρόπος μετατροπή τύπων. Έτσι, ίσως πρέπει να εξετάσουμε τις σελίδες λάθους κατά τη χρησιμοποίηση της αυτόματης μετατροπής τύπων. Τέταρτο, δεδομένου ότι και τα ονόματα ιδιότητας κι οι παράμετροι εισόδου είναι case sensitive, το όνομα της ιδιότητας κι η παράμετρος εισόδου πρέπει να ταιριάζουν ακριβώς.

3.8.3. Διανεμόμενα Beans

Μέχρι αυτό το σημείο, έχουμε μεταχειριστεί τα αντικείμενα που δημιουργήθηκαν με την *jsp:use-Bean* σαν αυτά να δεσμεύτηκαν απλά στις τοπικές μεταβλητές στη μέθοδο *_jspService*, που καλείται από τη μέθοδο *service* του servlet που παράγεται από τη σελίδα. Αν και τα beans είναι πράγματι συνδεδεμένα στις τοπικές μεταβλητές, αυτή δεν είναι η μόνη συμπεριφορά. Αποθηκεύονται επίσης, στη μια από τις τέσσερις διαφορετικές θέσεις, ανάλογα με την τιμή των προαιρετικών ιδιοτήτων *scope* της *jsp:useBean*. Η ιδιότητα *scope* έχει τις ακόλουθες πιθανές τιμές:

page

Αυτή είναι η προκαθορισμένη τιμή. Δείχνει ότι, εκτός από τη δέσμευση σε μια τοπική μεταβλητή, το αντικείμενο bean πρέπει να τοποθετηθεί στο αντικείμενο *PageContext* κατά τη διάρκεια του τρέχοντος αιτήματος. Σε γενικές γραμμές, η αποθήκευση του αντικειμένου εκεί σημαίνει ότι ο κώδικας του servlet μπορεί να έχει πρόσβαση με το να καλέσει την *getAttribute* στην προκαθορισμένη μεταβλητή *pageContext*. Στην πράξη, τα beans δημιουργούνται με το πεδίο *page* και σχεδόν πάντα προσπελαύνονται από την *jsp:getProperty*, την *jsp:setProperty*, τα scriptlets, ή τις εκφράσεις που θα παρατηρήσουμε παρακάτω αργότερα.

application

Αυτή η πολύ χρήσιμη τιμή σημαίνει ότι, εκτός από τη δέσμευση σε μια τοπική μεταβλητή, το αντικείμενο bean θα αποθηκευτεί στο κοινό *ServletContext* το οποίο είναι διαθέσιμο μέσω της προκαθορισμένης μεταβλητής `application` ή από μια κλήση της `getServletContext()`.

Το *ServletContext* μοιράζεται από όλα τα servlets στην ίδια Web εφαρμογή, ή όλα τα servlets στον ίδιο Server ή μηχανή servlet εάν καμία από τις ρητές Web εφαρμογές δεν καθορίζονται. Οι τιμές στο *ServletContext* μπορούν να ανακτηθούν από τη μέθοδο *getAttribute*. Αυτή η διανομή έχει μερικές διακλαδώσεις. Κατ' αρχάς, παρέχει έναν απλό μηχανισμό για πολλαπλά servlets και τις σελίδες JSP για να έχει πρόσβαση στο ίδιο αντικείμενο. Δεύτερον, αφήνει ένα servlet να δημιουργήσει ένα αντικείμενο bean το οποίο θα χρησιμοποιηθεί στις JSP σελίδες, όχι μόνο προσπελάζοντας μια η οποία δημιουργήθηκε προηγουμένως. Αυτή η προσέγγιση αφήνει ένα servlet να χειριστεί πολύπλοκα αιτήματα των χρηστών με τη σύσταση των beans, που τα αποθηκεύουν στο *ServletContext*, κατόπιν διαβιβάζοντας το αίτημα σε μια από διάφορες πιθανές σελίδες JSP για να παρουσιάσει τα αποτελέσματα κατάλληλα για τα στοιχεία αιτήματος.

session

Αυτή η τιμή σημαίνει ότι, εκτός από τη δέσμευση σε μια τοπική μεταβλητή, το αντικείμενο *bean* θα αποθηκευτεί στο αντικείμενο *HttpSession* που συνδέεται με το τρέχον αίτημα, όπου μπορεί να ανακτηθεί με την βοήθεια της *getValue*. Η προσπάθεια να χρησιμοποιηθεί `scope="session"` προκαλεί ένα λάθος στο χρόνο μεταφράσεων σελίδων όταν η οδηγία *page* ορίζει, ότι η τρέχουσα σελίδα δεν συμμετέχει σε συνόδους.

request

Αυτή η τιμή δηλώνει ότι, εκτός από τη δέσμευση σε μια τοπική μεταβλητή, το αντικείμενο *bean* πρέπει να τοποθετηθεί στο αντικείμενο *ServletRequest* κατά τη διάρκεια του τρέχοντος αιτήματος, όπου είναι διαθέσιμο με τη βοήθεια της μεθόδου *getAttribute*. Αυτή η τιμή είναι μόνο μια μικρή παραλλαγή του πεδίου ανά-αιτήματος που παρέχεται από `scope="page"` (ή εξ'ορισμού όταν δεν διευκρινίζεται κανένα *scope*).

3.9. Σύνδεση JDBC και Βάσεων Δεδομένων

3.9.1. Βασικά βήματα στην χρήση JDBC

Υπάρχουν επτά τυποποιημένα βήματα στη συζήτηση των βάσεων δεδομένων:
1. Φόρτωση του οδηγού JDBC

2. Καθορισμός της σύνδεσης URL

3. Εγκατάσταση της σύνδεσης

4. Δημιουργία ενός Statement αντικειμένου

5. Εκτέλεση ενός ερωτήματος ή ενός update

6. Επεξεργασία των αποτελεσμάτων

7. Κλείσιμο της σύνδεσης

Παρακάτω παρουσιάζονται μερικές λεπτομέρειες της διαδικασίας.

«Φόρτωση» του οδηγού

Ο οδηγός είναι το κομμάτι του λογισμικού που ξέρει πώς να μιλήσει στον πραγματικό Server των βάσεων δεδομένων. Για να «φορτώσουμε» τον οδηγό, όλα αυτά τα οποία

χρειαζόμαστε είναι να «φορτωθεί» η κατάλληλη κλάση που είναι ένα static block στην ίδια στην κλάση όπου αυτόματα κάνει μια περίπτωση οδηγών και την καταχωρεί με το διευθυντή οδηγών JDBC. Για να καταστήσει τον κώδικά μας όσο το δυνατόν πιο εύκαμπτο, είναι καλύτερο να αποφευχθεί η hard-coding στην αναφορά στο όνομα κλάσης.

Αυτές οι απαιτήσεις θέτουν δύο ενδιαφέρουσες ερωτήσεις. Κατ' αρχάς, πώς να «φορτώνουμε» μια κλάση χωρίς παραγωγή μιας περίπτωσης από αυτήν και δεύτερον, πώς μπορούμε να αναφερθούμε σε μια κλάση της οποίας το όνομα δεν είναι γνωστό όταν συντάσσεται ο κώδικας. Η απάντηση και στις δύο ερωτήσεις είναι η χρήση της *`Class.forName`*. Αυτή η μέθοδος παίρνει μια συμβολοσειρά που αντιπροσωπεύει ένα πλήρως κατάλληλο όνομα κλάσης, δηλαδή, ένα που περιλαμβάνει τα package ονόματα και «φορτώνει» την αντίστοιχη κλάση. Αυτή η κλήση θα μπορούσε να «πετάξει» ένα *`ClassNotFound-Exception`*, κι έτσι πρέπει να είναι μέσα σε ένα block *`try/catch`*. Παρακάτω ακολουθεί ένα σχετικό παράδειγμα:

```
try {

Class.forName("connect.microsoft.MicrosoftDriver");

Class.forName("oracle.jdbc.driver.OracleDriver");

Class.forName("com.sybase.jdbc.SybDriver");

} catch(ClassNotFoundException cnfe) {

System.err.println("Error loading driver: " + cnfe);

}
```

Μια από τις ομορφιές της προσέγγισης JDBC είναι ότι ο Server των βάσεων δεδομένων δεν απαιτεί καμία αλλαγή. Αντ' αυτού, ο οδηγός JDBC που βρίσκεται στον client μεταφράζει τις κλήσεις που γράφονται στη γλώσσα προγραμματισμού της Java σε συγκεκριμένο format που απαιτείται από τον Server. Αυτή η προσέγγιση σημαίνει ότι πρέπει να λάβουμε ένα οδηγό JDBC συγκεκριμένο για την βάση δεδομένων που εμείς χρησιμοποιούμε κι ότι θα πρέπει να ελέγξουμε την τεκμηρίωσή της για το πλήρως κατάλληλο όνομα κλάσης στη χρήση. Οι περισσότεροι προμηθευτές βάσεων δεδομένων παρέχουν τους ελεύθερους οδηγούς JDBC για τις βάσεις δεδομένων τους, αλλά υπάρχουν πολλοί προμηθευτές τρίτων των οδηγών για τις παλαιότερες βάσεις δεδομένων. Για έναν ενημερωμένο κατάλογο, μπορούμε να επισκεφτούμε την ηλεκτρονική διεύθυνση http://java.sun.com/products/jdbc/drivers.html. Πολλοί από αυτούς τους προμηθευτές οδηγών παρέχουν τις εκδόσεις ελεύθερης δοκιμής, συνήθως με μια ημερομηνία λήξης ή με μερικούς περιορισμούς στον αριθμό ταυτόχρονων συνδέσεων, κι έτσι είναι εύκολο να μαθευτεί ο JDBC χωρίς πληρωμή για έναν οδηγό.

Σε γενικές γραμμές, μπορούμε να χρησιμοποιήσουμε την *`Class.forName`* για οποιαδήποτε κλάση *`CLASSPATH`* μας. Στην πράξη εντούτοις, οι περισσότεροι προμηθευτές οδηγών JDBC διανέμουν τους οδηγούς τους μέσα στα αρχεία JAR. Έτσι, πρέπει να είμαστε βέβαιοι να περιλάβουμε την πορεία στο αρχείο JAR στη ρύθμιση *`CLASSPATH`* μας.

Καθορισμός της σύνδεσης URL

Μόλις «φορτώσουμε» τον οδηγό JDBC, πρέπει να διευκρινίσουμε τη θέση του Server των βάσεων δεδομένων. Τα URLs τα οποία αναφέρονται στις βάσεις δεδομένων χρησιμοποιούν το jdbc: το πρωτόκολλο κι έχει το server host, το port, και το database name, ή την αναφορά, που ενσωματώνεται μέσα στο URL. Το ακριβές format θα καθοριστεί στην τεκμηρίωση που έρχεται με τον κατάλληλο οδηγό, αλλά εδώ είναι δύο αντιπροσωπευτικά παραδείγματα:

```
String host = "dbhost.yourcompany.com";
```

```
String dbName = "someName";

int port = 1234;

String oracleURL = "jdbc:oracle:thin:@" + host +

":" + port + ":" + dbName;

String sybaseURL = "jdbc:sybase:Tds:" + host +

":" + port + ":" + "?SERVICENAME=" + dbName;
```

Ο JDBC χρησιμοποιείται συχνά από τα servlets ή τις κανονικές εφαρμογές υπολογιστών γραφείου αλλά υιοθετείται επίσης, μερικές φορές από τα applets. Εάν χρησιμοποιούμε JDBC από ένα applet, πρέπει να θυμηθούμε ότι, για να αποτρέψουν τα εχθρικά applets από το ξεφύλλισμα πίσω από τα εταιρικά firewalls, οι browsers αποτρέπουν τα applets από το να κάνουν τις συνδέσεις δικτύων οπουδήποτε εκτός από τον Server κι από τον οποίο φορτώθηκαν. Συνεπώς, στη χρήση JDBC από τα applets, είτε ο Server των βάσεων δεδομένων πρέπει να κατοικήσει στην ίδια μηχανή με τον Server HTTP ή πρέπει να χρησιμοποιήσουμε ένα proxy Server που επαναδρομολογεί τα ερωτήματα της βάσης δεδομένων σε ένα πραγματικό Server.

Εγκατάσταση της σύνδεσης

Για να κάνουμε την πραγματική σύνδεση δικτύων, πρέπει να περάσουμε το URL, το username της βάσης δεδομένων, και το password στη μέθοδο *`getConnection`* της κλάσης Driver-Manager, όπως διευκρινίζεται και στο ακόλουθο παράδειγμα. Πρέπει να σημειώσουμε ότι η *`getConnection`* «πετάει» ένα *`SQLException`*, κι έτσι πρέπει να χρησιμοποιήσουμε ένα block *`try/catch`*. Στο παράδειγμα που ακολουθεί παραλείπουμε αυτό το block, δεδομένου ότι οι μέθοδοι στα ακόλουθα βήματα «πετάνε» την ίδια εξαίρεση, κι έτσι χρησιμοποιούμε χαρακτηριστικά ένα ενιαίο block *`try/catch`* για όλες τους.

```
String username = "jay_debesee";

String password = "secret";

Connection connection =

DriverManager.getConnection(oracleURL, username, password);
```

Ένα προαιρετικό μέρος αυτού του βήματος είναι να παρατηρήσουμε επάνω στις πληροφορίες για την βάση δεδομένων με τη χρησιμοποίηση της *`getMetaData`* μεθόδου της *`Connection`*. Αυτή η μέθοδος επιστρέφει ένα αντικείμενο *DatabaseMetaData* που έχει τις μεθόδους για να μας επιτρέψει να ανακαλύψουμε το όνομα καθώς επίσης, και την έκδοση της ίδιας της βάσης δεδομένων (*`getDatabaseProductName`*, *`getDatabaseProductVersion`*) ή του οδηγού JDBC (*`getDriverName`*, *`get-DriverVersion`*). Εδώ είναι ένα παράδειγμα:

```
DatabaseMetaData dbMetaData = connection.getMetaData();

String productName =

dbMetaData.getDatabaseProductName();

System.out.println("Database: " + productName);

String productVersion =dbMetaData.getDatabaseProductVersion();

System.out.println("Version: " + productVersion);
```

Άλλες χρήσιμες μέθοδοι στην κλάση *Connection* περιλαμβάνουν *prepareStatement*, *prepare-Call* με την οποία δημιουργούμε ένα *CallableStatement*, *rollback* με την οποία πραγματοποιείται αναίρεση των δηλώσεων προτού του τελευταίου commit, *commit* με το οποίο γίνεται η οριστικοποίηση των διαδικασιών προτού του τελευταίου commit, *close* με την οποία ολοκληρώνεται η σύνδεση, και *isClosed* όπου είτε η σύνδεση είναι εκτός χρόνου, είτε κλείνει ρητά.

Δημιουργία ενός Statement

Ένα αντικείμενο Statement χρησιμοποιείται για να στείλει τις ερωτήσεις και τις εντολές στην βάση δεδομένων και δημιουργείται από τη *Connection* ως εξής:

```
Statement statement = connection.createStatement();
```

Εκτέλεση ενός ερωτήματος (Query)

Μόλις έχουμε ένα αντικείμενο Statement, μπορούμε να το χρησιμοποιήσουμε για να στείλουμε τα SQL ερωτήματα με τη χρησιμοποίηση της μεθόδου *executeQuery*, η οποία επιστρέφει ένα αντικείμενο του τύπου *Result-Set*. Εδώ είναι ένα παράδειγμα:

```
String query = "SELECT col1, col2, col3 FROM sometable";

ResultSet resultSet = statement.executeQuery(query);
```

Για να τροποποιήσουμε την βάση δεδομένων, χρησιμοποιούμε *executeUpdate* αντί του *executeQuery*, και παρέχουμε έτσι μια συμβολοσειρά η οποία χρησιμοποιεί *UPDATE*, *INSERT*, ή *DELETE*. Άλλες χρήσιμες μέθοδοι στην κλάση Statement περιλαμβάνουν *execute* δηλαδή, την εκτέλεση μιας αυθαίρετης εντολής και *setQueryTimeout* σαν ο ορισμός μιας μέγιστης καθυστέρησης για να περιμένουμε τα αποτελέσματα. Μπορούμε επίσης, να δημιουργήσουμε τα παραμετροποιήσιμα ερωτήματα (parameterized queries), όπου οι τιμές παρέχονται σε ένα προσυλλεγμένο fixed-format ερώτημα.

Επεξεργασία των αποτελεσμάτων

Ο απλούστερος τρόπος να αντιμετωπιστούν τα αποτελέσματα είναι να υποβληθούν σε επεξεργασία μια γραμμή τη φορά, που χρησιμοποιεί την *ResultSet's next* μέθοδο για να κινήσει μέσω του πίνακα μια γραμμή σε ένα χρόνο. Μέσα σε μια γραμμή, η *ResultSet* παρέχει ότι διάφορες *getXxx* μέθοδοι που παίρνουν ένα δείκτη στηλών ή ένα όνομα στηλών σαν όρισμα κι επιστρέφουν το αποτέλεσμα σαν ποικίλους διαφορετικούς τύπους της Java. Παραδείγματος χάριν, η χρήση της *getInt* εάν η τιμή της ήταν ένας ακέραιος αριθμός, *getString* για μια συμβολοσειρά, και τόσο επάνω για τους περισσότερους άλλους τύπους στοιχείων.

Εάν θέλουμε ακριβώς να εμφανίσουμε τα αποτελέσματα, μπορούμε να χρησιμοποιήσουμε την *getString* ανεξάρτητα από τον πραγματικό τύπο των στηλών. Εντούτοις, εάν χρησιμοποιούμε την έκδοση που παίρνει ένα δείκτη στηλών, πρέπει να σημειώσουμε ότι οι στήλες συντάσσονται αρχικά σε 1, έπειτα από τη σύμβαση SQL, κι όχι σε 0 όπως με τις συμβολοσειρές, τα διανύσματα, και τις περισσότερες άλλες δομές δεδομένων στη γλώσσα προγραμματισμού της Java.

Εδώ είναι ένα παράδειγμα που τυπώνει τις τιμές των πρώτων τριών στηλών σε όλες τις γραμμές ενός *ResultSet*.

```
while(resultSet.next()) {

System.out.println(results.getString(1) + " " +

results.getString(2) + " " +
```

```
results.getString(3));

}
```

Εκτός από τις *getXxx* και τις *next* μεθόδους, άλλες χρήσιμες μέθοδοι στην κλάση ResultSet περιλαμβάνουν *findColumn*, *wasNull*, και *getMetaData*.

Η μέθοδος *getMetaData* είναι ιδιαίτερα χρήσιμη. Λαμβάνοντας υπόψη μόνο ένα ResultSet, πρέπει να είμαστε ενήμεροι για το όνομα, τον αριθμό, και τον τύπο των στηλών για να είμαστε σε θέση να επεξεργαστούμε τον πίνακα κατάλληλα. Για τις ειδικές ερωτήσεις εντούτοις, είναι χρήσιμο να είναι σε θέση να ανακαλύψει δυναμικά τις υψηλού επιπέδου πληροφορίες για το αποτέλεσμα. Αυτός είναι ο ρόλος της κλάσης ResultSetMetaData, όπου μας επιτρέπει να καθορίσουμε τον αριθμό, τα ονόματα, και τους τύπους των στηλών στο ResultSet. Οι χρήσιμες μέθοδοι *ResultSetMetaData* περιλαμβάνουν επίσης, *getColumnCount* που είναι ο αριθμός των στηλών, *getColumn-Name* (*colNumber*, ένα όνομα στηλών), *getColumnType* ένα *int* για να συγκρίνει ενάντια στις καταχωρήσεις *java.sql.Types*, *isReadOnly* όπου είναι η είσοδος μιας μόνο τιμής ανάγνωσης, *isSearchable* όπου μπορεί να χρησιμοποιηθεί σε μια *WHERE* πρόταση, *isNullable* που είναι μια επιτρεπόμενη *null* τιμή, και διάφοροι άλλοι που δίνουν τις λεπτομέρειες στον τύπο και την ακρίβεια της στήλης. Το *ResultSetMetaData* δεν περιλαμβάνει τον αριθμό των γραμμών, εντούτοις όμως, ο μόνος τρόπος για να καθορίσει ποιός είναι, είναι να καλέσει επανειλημμένα την *next* στο ResultSet έως ότου επιστρέψει *false*.

Κλείσιμο της σύνδεσης

Για να κλείσουμε τη σύνδεση ρητά, θα κάνουμε:

```
connection.close();
```

Πρέπει να αναβάλουμε αυτό το βήμα εάν αναμένουμε να εκτελέσουμε τις πρόσθετες διαδικασίες των βάσεων δεδομένων, δεδομένου ότι τα γενικά έξοδα του ανοίγματος μιας σύνδεσης είναι συνήθως μεγάλα.

3.10. Servlets

Τα servlets είναι η απάντηση της τεχνολογίας της Java στο Common Gateway Interface (CGI) προγραμματισμό. Είναι προγράμματα τα οποία εκτελούνται σε ένα Web Server, ενεργώντας ως μέσο στρώμα μεταξύ ενός αιτήματος που προέρχεται από ένα Web Browser ή άλλου client HTTP και των βάσεων δεδομένων ή των εφαρμογών (Application) στον Server HTTP. Οι εργασίες που έχουν να επιτελέσουν είναι οι εξής:

- Να διαβάσουν οποιαδήποτε στοιχεία που στέλνονται από το χρήστη

  Αυτά τα στοιχεία εισάγονται συνήθως σε μια φόρμα μιας ιστοσελίδας, αλλά θα μπορούσαν επίσης, να προέλθουν κι από ένα Java Applet, ή από ένα πρόγραμμα client HTTP.

- Να ανατρέξουν οποιεσδήποτε πληροφορίες για το αίτημα που ενσωματώνεται στο αίτημα HTTP

  Αυτές οι πληροφορίες περιλαμβάνουν τις λεπτομέρειες για τις ικανότητες ενός browser, των cookies, το host name του αιτούντος client, κι ούτω καθεξής.

- Να παράγουν τα αποτελέσματα

  Αυτή η διαδικασία μπορεί να απαιτήσει σε μια βάση δεδομένων, μια κλήση RMI ή CORBA, μια εφαρμογή κληρονομιών, ή την απάντηση άμεσα.

- ❖ Να σχηματοποιήσουν τα αποτελέσματα μέσα σε ένα έγγραφο

  Στις περισσότερες περιπτώσεις, αυτό περιλαμβάνει την ενσωμάτωση των πληροφοριών μέσα σε μια σελίδα HTML.

- ❖ Να θέσουν τις κατάλληλες παραμέτρους απάντησης HTTP

  Αυτό σημαίνει να πει στον browser ποιος τύπος εγγράφου επιστρέφεται (π.χ., HTML), τα Cookies ρύθμισης και τις παραμέτρους, κι άλλοι τέτοιοι στόχοι.

- ❖ Να στείλουν το έγγραφο πίσω στον client

  Το παρόν έγγραφο μπορεί να σταλεί με το format κειμένων (HTML), το binary format (εικόνες GIF), ή ακόμα και με ένα συμπιεσμένο format όπως το gzip που είναι βαλμένο σε στρώσεις πάνω από κάποιο άλλο ελλοχεύον format. Πολλά αιτήματα του client μπορούν να ικανοποιηθούν με την επιστροφή των προ-χτισμένων εγγράφων, κι αυτά τα αιτήματα θα αντιμετωπίζονταν από τον Server χωρίς επίκληση των servlets.

Σε πολλές περιπτώσεις εντούτοις, ένα στατικό αποτέλεσμα δεν είναι ικανοποιητικό, και μια σελίδα πρέπει να παραχθεί για κάθε αίτημα. Υπάρχουν διάφοροι λόγοι για τους οποίους οι ιστοσελίδες χρειάζονται να χτιστούν on-the-fly όπως οι παρακάτω ιστοσελίδες:

- ✓ Η ιστοσελίδα είναι βασισμένη στα στοιχεία τα οποία υποβάλλονται από το χρήστη

  Παραδείγματος χάριν, η σελίδα αποτελεσμάτων από τις μηχανές αναζήτησης κι οι σελίδες διαταγή-επιβεβαίωσης στα on-line καταστήματα είναι συγκεκριμένες για τα κατάλληλα αιτήματα χρηστών.

- ✓ Η ιστοσελίδα προέρχεται από στοιχεία τα οποία αλλάζουν συχνά

  Παραδείγματος χάριν, μια καιρική έκθεση ή μια σελίδα τίτλων ειδήσεων να χτίσει τη σελίδα δυναμικά, ίσως επιστρέφοντας μια προηγουμένως χτισμένη σελίδα εάν είναι ακόμα ενημερωμένη.

- ✓ Η ιστοσελίδα χρησιμοποιεί τις πληροφορίες από τις εταιρικές βάσεις δεδομένων ή άλλες server-side πηγές.

  Παραδείγματος χάριν, ένα e-commerce Site θα μπορούσε να χρησιμοποιήσει ένα servlet για να χτίσει μια ιστοσελίδα η οποία απαριθμεί την τρέχουσα τιμή και τη διαθεσιμότητα κάθε στοιχείου που είναι για την πώληση.

Σε γενικές γραμμές, τα servlets δεν είναι περιορισμένα στους Servers δικτύου ή εφαρμογής που χειρίζονται τα αιτήματα HTTP, αλλά μπορούν να χρησιμοποιηθούν για άλλους τύπους Servers επίσης. Παραδείγματος χάριν, τα servlets θα μπορούσαν να ενσωματωθούν στους Servers ταχυδρομείου ή FTP για να επεκτείνουν τη λειτουργία τους. Στην πράξη εντούτοις, αυτή η χρήση των servlets δεν έχει πιάσει επάνω, και θα συζητήσουμε μόνο τα servlets HTTP.

3.10.1. Τα πλεονεκτήματα των Servlets πέραν της "παραδοσιακής" CGI

Τα servlets της Java είναι αποδοτικότερα, ευκολότερα να χρησιμοποιηθούν, ισχυρότερα, πιο φορητά, ασφαλέστερα, και φτηνότερα από την παραδοσιακή CGI κι από τις πολύ εναλλακτικές CGI-like τεχνολογίες.

Αποδοτικότερα

Με την παραδοσιακή CGI, μια νέα διαδικασία αρχίζει για κάθε ένα αίτημα HTTP. Εάν το ίδιο το πρόγραμμα της CGI είναι σχετικά σύντομο, τα γενικά έξοδα της έναρξης της

διαδικασίας μπορούν να εξουσιάσουν το χρόνο εκτέλεσης. Με τα servlets, η Java Virtual Machine μένει τρέχοντας και χειρίζεται κάθε αίτημα χρησιμοποιώντας ένα ελαφρύ νήμα της Java, κι όχι μια βαρέων βαρών διαδικασία λειτουργικών συστημάτων. Ομοίως, στην παραδοσιακή CGI, εάν υπάρχουν ταυτόχρονα αιτήματα N στο ίδιο πρόγραμμα της CGI, ο κώδικας για το πρόγραμμα της CGI φορτώνεται στη μνήμη N χρόνους. Με τα servlets εντούτοις, θα υπήρχαν N νήματα αλλά μόνο ένα ενιαίο αντίγραφο της κλάσης Servlet. Τέλος, όταν τελειώνει ένα πρόγραμμα της CGI ένα αίτημα, το πρόγραμμα τερματίζεται. Αυτό το καθιστά δύσκολο να εναποθηκεύσει τους υπολογισμούς, να κρατήσει τις συνδέσεις των βάσεων δεδομένων ανοικτές, και να εκτελέσει άλλες βελτιστοποιήσεις που στηρίζονται στα επίμονα στοιχεία. Τα servlets εντούτοις, παραμένουν στη μνήμη ακόμα κι αφού ολοκληρώσουν μια απάντηση, κι έτσι είναι απλό για να αποθηκεύσουν τα αυθαίρετα σύνθετα στοιχεία μεταξύ των αιτημάτων.

Καταλληλότερα

Τα servlets έχουν μια εκτενή υποδομή για αυτόματη ανάλυση κι αποκωδικοποίηση ενός HTML form data, ανάγνωση και ρύθμιση των headers HTTP, τον χειρισμό των Cookies, τις συνόδους καταδίωξης, και πολλές άλλες τέτοιες υψηλού επιπέδου χρησιμότητες. Εκτός αυτού, ξέρουμε ήδη τη γλώσσα προγραμματισμού Java. Γιατί να συντρέχει λόγος να μάθουμε επίσης, και την γλώσσα προγραμματισμού Perl; Είμαστε ήδη πεπεισμένοι ότι εκείνη η τεχνολογία της Java κάνει για τον πιο αξιόπιστο κι επαναχρησιμοποιήσιμο κώδικα από ότι κάνει η C++. Γιατί να επιστρέψουμε στην C++ για server-side προγραμματισμό;

Ισχυρότερα

Τα servlets υποστηρίζουν διάφορες ικανότητες που είναι δύσκολο ή αδύνατο να ολοκληρωθούν με την κανονική CGI. Τα servlets μπορούν να μιλήσουν άμεσα στον Server δικτύου, ενώ τα κανονικά προγράμματα της CGI δεν μπορούν, τουλάχιστον χωρίς την χρησιμοποίηση ενός server-specific API. Η επικοινωνία με τον Server δικτύου το καθιστά ευκολότερο να μεταφράσει σχετικά URLs στα συγκεκριμένα ονόματα πορειών, παραδείγματος χάριν. Τα πολλαπλά servlets μπορούν επίσης, να μοιραστούν τα στοιχεία, που το καθιστούν εύκολο να εφαρμόσουν τη συγκέντρωση σύνδεσης των βάσεων δεδομένων και τις παρόμοιες βελτιστοποιήσεις κοινής εκμετάλλευσης πόρων. Τα servlets μπορούν επίσης, να διατηρήσουν τις πληροφορίες από αίτημα σε αίτημα, που απλοποιεί τις τεχνικές όπως τη σύνοδο που ακολουθεί και που εναποθηκεύει τους προηγούμενους υπολογισμούς.

Φορητότερα

Τα servlets γράφονται στη γλώσσα προγραμματισμού της Java κι ακολουθεί ένα τυποποιημένο API. Συνεπώς, τα servlets που γράφονται για παράδειγμα, τον I-Planet Enterprise Server μπορούν να «τρέξουν» ουσιαστικά αμετάβλητα σε Apache, στον Microsoft Internet Information Server (IIS), την IBM WebSphere, ή StarNine WebStar. Παραδείγματος χάριν, ουσιαστικά όλα τα servlets κι οι σελίδες JSP σε αυτό το έγγραφο εκτελέστηκαν στον Sun's Java Web Server, Apache Tomcat και Sun's JavaServer Web Development Kit (JSWDK) χωρίς τις αλλαγές στον κώδικα. Πολλά servlets εξετάστηκαν σε BEA WebLogic και στην IBM WebSphere επίσης. Στην πραγματικότητα, τα servlets υποστηρίζονται άμεσα ή από μια σύνδεση σε ουσιαστικά κάθε σημαντικό Server δικτύου. Είναι τώρα μέρος της Java 2 Platform, Enterprise Edition (J2EE βλέπουμε http://java.sun.com/j2ee/), κι έτσι η υποστήριξη βιομηχανίας για τα servlets γίνεται ακόμα πιο κυρίαρχη.

Ασφαλέστερα

Μια από τις κύριες πηγές ευπαθειών στα παραδοσιακά προγράμματα της CGI προέρχεται από το γεγονός ότι εκτελούνται συχνά από τα γενικής χρήσης κοχύλια (shells) λειτουργικών συστημάτων. Έτσι, ο προγραμματιστής της CGI πρέπει να είναι πολύ προσεκτικός στο φίλτρο έξω χαρακτήρες όπως τα backquotes κι οι άνω τελείες που αντιμετωπίζονται ειδικά από το κοχύλι. Αυτό είναι σκληρότερο από κάποιος να σκεφτεί, κι οι αδυναμίες που προέρχονται από αυτό το πρόβλημα αποκαλύπτονται συνεχώς στην ευρέως χρησιμοποιημένη CGI βιβλιοθήκες. Μια δεύτερη πηγή προβλημάτων είναι το γεγονός ότι μερικά προγράμματα της CGI υποβάλλονται σε επεξεργασία από τις γλώσσες που δεν ελέγχουν αυτόματα τα όρια ενός πίνακα ή μιας συμβολοσειράς. Παραδείγματος χάριν, στην C και την C++ είναι τέλεια επιτρεπτό να διατεθεί το 100-στοιχείο του πίνακα, τότε γράφει έπειτα στο 999th "στοιχείο," που είναι πραγματικά κάποιο τυχαίο μέρος της μνήμης προγράμματος. Έτσι, οι προγραμματιστές που ξεχνούν να κάνουν αυτόν τον έλεγχο οι ίδιοι ανοίγουν το σύστημά τους μέχρι το σκόπιμο ή τυχαίο buffer επιθέσεις υπερχείλισης. Τα servlets υποφέρουν από κανένα από αυτά τα προβλήματα. Ακόμα κι αν ένα servlet εκτελεί μια μακρινή κλήση συστημάτων για να επικαλεσθεί ένα πρόγραμμα για το τοπικό λειτουργικό σύστημα, δεν χρησιμοποιεί ένα κοχύλι για να κάνει έτσι. Και φυσικά τα όρια του πίνακα ελέγχονται κι άλλα χαρακτηριστικά γνωρίσματα προστασίας μνήμης είναι ένα κεντρικό μέρος της γλώσσας προγραμματισμού της Java.

Ανέξοδα

Υπάρχουν διάφοροι ελεύθεροι ή πολύ ανέξοδοι Servers δικτύου διαθέσιμοι που είναι καλοί για τη "προσωπική" χρήση ή τους χαμηλής έντασης ιστοχώρους. Εντούτοις, με τη σημαντικότερη εξαίρεση του Apache, η οποία είναι ελεύθερη, οι περισσότεροι commercial-quality Servers δικτύου είναι σχετικά ακριβοί. Εντούτοις, μόλις έχουμε ένα Server δικτύου, δεν τίθεται πλέον θέμα κόστους, που προσθέτει την υποστήριξη του servlet στο γεγονός ότι κοστίζει πολύ λίγα επιπλέον. Αυτό είναι σε αντίθεση με πολλές από τις άλλες εναλλακτικές λύσεις της CGI, οι οποίες απαιτούν μια σημαντική αρχική επένδυση για να αγοράσουν ένα ιδιόκτητο package.

3.10.2. Βασική δομή ενός Servlet

Ένας browser παράγει αυτό το αίτημα όταν ο χρήστης πληκτρολογήσει ένα URL στη γραμμή διευθύνσεων, κι ακολουθεί μια σύνδεση από μια ιστοσελίδα, ή υποβάλλει μια φόρμα HTML που δεν διευκρινίζει μια `METHOD`. Τα servlets μπορούν επίσης, πολύ εύκολα να χειριστούν `POST` αιτήματα, τα οποία παράγονται όταν υποβάλλει κάποιος μια φόρμα HTML που διευκρινίζει `METHOD=" POST "`.

Για να είναι ένα servlet, μια κλάση πρέπει να επεκτείνει την *HttpServlet* και να αγνοήσει την `doGet` ή την `doPost`, ανάλογα με εάν το στοιχείο στέλνεται από `GET` ή `POST` αιτήματα. Εάν θέλουμε το ίδιο servlet για να χειριστούμε και `GET` και `POST` αιτήματα και για να λάβουμε ίδια μέτρα για κάθε ένα, μπορούμε απλά να έχουμε `doGet` την κλήση `doPost`, ή αντίστροφα.

Κι οι δύο μέθοδοι παίρνουν δύο ορίσματα: ένα *HttpServletRequest* κι ένα *HttpServletResponse*. Η *HttpServletRequest* έχει τις μεθόδους με τις οποίες μπορούμε να ανακαλύψουμε για τις εισερχόμενες πληροφορίες όπως τα form data, τα headers αιτήματος HTTP, και το hostname του client. Η *HttpServletResponse* μας αφήνει να διευκρινίσουμε τις εξερχόμενες πληροφορίες όπως τα HTTP Status Codes (200, 404, κ.λπ.), τα headers απάντησης (*Content-Type*, *Set-Cookie*, κ.λπ.), κι επιπλέον, μας αφήνει να λάβουμε ένα *PrintWriter* που χρησιμοποιείται για να στείλει την περιεκτικότητα σε έγγραφα πίσω στον client. Για τα απλά servlets, το μεγαλύτερο μέρος της προσπάθειας ξοδεύεται στις δηλώσεις `println` που παράγουν την επιθυμητή σελίδα. Τα form data, τα headers αιτήματος HTTP, οι απαντήσεις HTTP, και τα Cookies όλα θα συζητηθούν λεπτομερώς παρακάτω. Δεδομένου ότι τόσο η `doGet`, όσο επίσης, κι η `doPost`

«πετάνε» δύο εξαιρέσεις, είμαστε υποχρεωμένοι να τις περιλάβουμε στη δήλωση. Τέλος, πρέπει να εισάγουμε τις κλάσεις `Java.io` (για *PrintWriter*, κ.λπ.), `javax.servlet` (για *HttpServlet*, κ.λπ.), και `javax.servlet.http` (για *HttpServletRequest* και *HttpServlet-Response*).

Για να κυριολεκτήσουμε, η *HttpServlet* δεν είναι η μόνη αφετηρία για τα servlets, δεδομένου ότι τα servlets μπόρεσαν, σε γενικές γραμμές, να επεκτείνουν το ταχυδρομείο, το FTP, ή άλλους τύπους Servers. Τα servlets για αυτά τα περιβάλλοντα θα επέκτειναν μια κλάση συνήθειας που προέρχεται από την Generic-Servlet, που είναι η γονική κλάση της *HttpServlet*. Στην πράξη εντούτοις, τα servlets χρησιμοποιούνται σχεδόν αποκλειστικά για τους Servers που επικοινωνούν μέσω του HTTP (δηλ., Servers δικτύου κι εφαρμογής).

3.10.3. Ο κύκλος ζωής ενός Servlet

Παραπάνω, αναφερθήκαμε αόριστα στο γεγονός ότι μόνο μια ενιαία περίπτωση ενός servlet δημιουργείται, με κάθε αίτημα χρηστών με συνέπεια ένα νέο νήμα που δίνεται μακριά στην `doGet` ή στην `doPost` ανάλογα με την περίπτωση. Σκοπός μας είναι τώρα να γίνουμε πιο συγκεκριμένοι για το πώς τα servlets δημιουργούνται και καταστρέφονται, και πώς και πότε οι διάφορες μέθοδοι επικαλούνται.

Αρχικά, όταν το servlet δημιουργείται, η μέθοδος της *init* επικαλείται, έτσι ώστε που να τοποθετήσουμε τον one-time κώδικα οργάνωσης. Μετά από αυτό, κάθε αίτημα των χρηστών οδηγεί σε ένα νήμα το οποίο καλεί τη μέθοδο *service* της προηγουμένως δημιουργημένης περίπτωσης.

Τα πολλαπλά ταυτόχρονα αιτήματα οδηγούν κανονικά στα πολλαπλά νήματα που καλούν τη *service* ταυτόχρονα, αν και το servlet μας μπορεί να εφαρμόσει μια ειδική διεπαφή που ορίζει ότι μόνο ένα ενιαίο νήμα επιτρέπεται για να «τρέξει» σε οποιοδήποτε χρόνο. Η μέθοδος *service* καλεί στη συνέχεια τη `doGet`, `doPost`, ή άλλη `doXxx` μέθοδο, ανάλογα με τον τύπο αιτήματος HTTP το οποίο έλαβε. Τέλος, όταν αποφασίζει να «ξεφορτώσει» ο Server ένα servlet, καλεί αρχικά τη μέθοδο *destroy* του servlet.

Η μέθοδος *init*

Η μέθοδος *init* καλείται όταν δημιουργείται αρχικά το servlet και δεν απαιτείται πάλι κάθε αίτημα χρηστών. Έτσι, χρησιμοποιείται για τις one-time ενάρξεις, ακριβώς όπως και με τη μέθοδο *init* των applets. Το servlet μπορεί να δημιουργηθεί όταν επικαλείται αρχικά ένας χρήστης ένα URL αντίστοιχο στο servlet ή όταν αρχίζει αρχικά ο Server, ανάλογα με το πώς έχουμε καταχωρήσει το servlet με τον Server δικτύου. Θα δημιουργηθεί για το πρώτο αίτημα χρηστών εάν δεν καταχωρείται ρητά αλλά αντ' αυτού ακριβώς τοποθετείται σε ένα από τους τυποποιημένους Servers καταλόγους.

Υπάρχουν δύο εκδόσεις της μεθόδου *init*: Μια που δεν παίρνει κανένα όρισμα και μια η οποία παίρνει ένα αντικείμενο *ServletConfig* σαν όρισμα. Η πρώτη έκδοση χρησιμοποιείται όταν δεν πρέπει να διαβάσει το servlet οποιεσδήποτε τοποθετήσεις που ποικίλλουν από Server σε Server. Ο καθορισμός της μεθόδου μοιάζει με αυτό το παράδειγμα:

```
public void init() throws ServletException {

//Initialization code...

}
```

Η δεύτερη έκδοση της μεθόδου *init* χρησιμοποιείται όταν πρέπει να διαβάσει το servlet τις server-specific τοποθετήσεις προτού να μπορέσει να ολοκληρώσει την έναρξη. Παραδείγματος χάριν, το servlet να πρέπει να ξέρει για τις τοποθετήσεις των βάσεων δεδομένων, τα αρχεία κωδικού πρόσβασης, τις server-specific παραμέτρους απόδοσης, τα

αρχεία αρίθμησης χτυπήματος, ή τα δημοσιευμένα στοιχεία Cookies από τα προηγούμενα αιτήματα. Η δεύτερη έκδοση της μεθόδου *init* μοιάζει με αυτό το παράδειγμα:

```
public void init(ServletConfig config) throws ServletException {

      super.init(config);

      //Initialization code...

}
```

Για αυτόν τον κώδικα έχουμε να παρατηρήσουμε δύο πράγματα. Κατ' αρχάς, η μέθοδος *init* παίρνει ένα `Servlet-Config` σαν όρισμα. Το `ServletConfig` έχει μια μέθοδο `getInitParameter` με την οποία μπορούμε να κοιτάξουμε επάνω στις παραμέτρους έναρξης που συνδέονται με το servlet.

Ακριβώς όπως με τη μέθοδο `getParameter` που χρησιμοποιείται στη μέθοδο *init* των applets, τόσο η είσοδος (το όνομα παραμέτρου), όσο κι η έξοδος (η τιμή παραμέτρου) είναι συμβολοσειρές. Πρέπει να σημειώσουμε ότι αν και κοιτάζουμε επάνω στις παραμέτρους κατά τρόπο φορητό, τους θέτουμε με έναν server-specific τρόπο. Παραδείγματος χάριν, με Tomcat, ενσωματώνουμε τις ιδιότητες servlet σε ένα αρχείο αποκαλούμενο web.xml, με το JSWDK χρησιμοποιούμε servlets.properties, με τον Server εφαρμογής WebLogic χρησιμοποιούμε weblogic.properties, και με το Java Web Server θέτουμε τις ιδιότητες αμφίδρομα μέσω της κονσόλας διοίκησης.

Το δεύτερο πράγμα που παρατηρείται για τη δεύτερη έκδοση της μεθόδου *init* είναι ότι η πρώτη γραμμή του σώματος της μεθόδου είναι μια κλήση της μεθόδου *super.init*. Αυτή η κλήση είναι ιδιαιτέρως κρίσιμη! Το αντικείμενο *ServletConfig* χρησιμοποιείται αλλού στο servlet, κι η μέθοδος *init* των υπερκλάσεων το καταχωρεί όπου το servlet μπορεί να το βρει αργότερα. Έτσι, μπορεί να προκαλέσουμε στους εαυτούς μας τεράστιους πονοκέφαλους αργότερα εάν παραλείψουμε αυτή την κλήση *super.init*.

Η μέθοδος *service*

Κάθε φορά που λαμβάνει ο Server ένα αίτημα για ένα servlet, ο Server «γεννάει» ένα νέο νήμα και καλεί την μέθοδο *service*. Η μέθοδος *service* ελέγχει τον τύπο του HTTP αιτήματος (*GET*, *POST*, *PUT*, *DELETE*, κ.λπ.) και καλεί ανάλογα με την περίπτωση τις μεθόδους `doGet`, `doPost`, `doPut`, `doDelete`, κ.λπ. Τώρα, εάν έχουμε ένα servlet που πρέπει να χειριστεί και τόσο τα *POST*, επίσης, και τα *GET* αιτήματα όμοια, μπορούμε να μπούμε στον πειρασμό και να αγνοήσουμε την μέθοδο *service* άμεσα όπως παρουσιάζεται και στο παρακάτω παράδειγμα, παρά την εφαρμογή και `doGet` και `doPost`.

```
public void service(HttpServletRequest request,HttpServletResponse response)throws
ServletException, IOException {

   // Servlet Code

}
```

Αυτό δεν είναι μια καλή ιδέα. Αντ' αυτού, ακριβώς έχει `doPost` την κλήση `doGet` (ή αντίστροφα), όπως στο παράδειγμα που ακολουθεί:

```
public void doGet(HttpServletRequest request,
HttpServletResponse response)throws ServletException, IOException {

  // Servlet Code

}
```

```
public void doPost(HttpServletRequest request,
HttpServletResponse response)throws ServletException, IOException {

  doGet(request, response);

}
```

Αν κι αυτή η προσέγγιση παίρνει μερικές πρόσθετες γραμμές κώδικα, έχει πέντε πλεονεκτήματα πέρα από την αγνόηση της μεθόδου *service*:

- Μπορούμε να προσθέσουμε την υποστήριξη για άλλες υπηρεσίες αργότερα με την προσθήκη `doPut`, `doTrace`, κ.λπ., ίσως σε μια υποκλάση. Η εξαιρετικά σημαντική μέθοδος *service* αποκλείει άμεσα αυτήν την δυνατότητα.

- Μπορούμε να προσθέσουμε την υποστήριξη για τις ημερομηνίες τροποποίησης με την προσθήκη μιας μεθόδου *get-LastModified*. Εάν χρησιμοποιούμε `doGet`, η τυποποιημένη μέθοδος *service* χρησιμοποιεί τη μέθοδο *getLastModified* για να θέσει τα *Last-Modified* headers και για να αποκριθούν κατάλληλα σε υπό όρους *GET* αιτημάτων (αυτά περιλαμβάνουν ένα *If-Modified-Since* header).

- Παίρνουμε την αυτόματη υποστήριξη για *HEAD* αιτήματα. Το σύστημα επιστρέφει ακριβώς οτιδήποτε headers και status codes `doGet` θέτει, αλλά παραλείπει το σώμα των σελίδων. Το *HEAD* είναι μια χρήσιμη μέθοδος αιτήματος για τους συνηθισμένους HTTP clients. Παραδείγματος χάριν, οι ελεγκτικοί μηχανισμοί αξιοπιστίας των συνδέσεων που ελέγχουν μια σελίδα για τις νεκρές συνδέσεις υπερκειμένων χρησιμοποιούν συχνά *HEAD* αντί *GET* προκειμένου να μειώσουν το φόρτο εργασίας των Servers.

- Παίρνουμε την αυτόματη υποστήριξη για τα αιτήματα *OPTIONS*. Εάν μια μέθοδος `doGet` υπάρχει, η τυποποιημένη *service* μέθοδος απαντάει με *OPTIONS* αιτήματα, επιστρέφοντας ένα *Allow* header που δείχνει ότι τα *GET*, τα *HEAD*, τα *OPTIONS*, και τα *TRACE* αιτήματα υποστηρίζονται.

- Παίρνουμε την αυτόματη υποστήριξη για τα αιτήματα *TRACE*. Το *TRACE* είναι μια μέθοδος αιτήματος που χρησιμοποιείται για τη διόρθωση client: επιστρέφει ακριβώς τα headers ενός HTTP αιτήματος πίσω στον client.

Οι μέθοδοι doGet, doPost και doXXX

Αυτές οι μέθοδοι περιέχουν το πραγματικό μέρος του servlet μας. Ενενήντα εννέα τοις εκατό (99%) του χρόνου, εμείς φροντίζουμε μόνο για *GET* ή/και *POST* αιτήματα, κι έτσι αγνοούμε τις μεθόδους `doGet` ή/και `doPost`. Εντούτοις, εάν θέλουμε, μπορούμε επίσης, να αγνοήσουμε την μέθοδο `doDelete` για τα αιτήματα *DELETE*, την `doPut` για *PUT*, την `doOptions` για *OPTIONS*, και την `doTrace` για *TRACE* αιτήματα αντίστοιχα. Θα ξανακαλέσουμε εντούτοις, όπου έχουμε την αυτόματη υποστήριξη για *OPTIONS* και *TRACE*, όπως περιγράφτηκε και στο προηγούμενο τμήμα που αφορούσε τη μεθόδου *service*. Σε αυτό το σημείο πρέπει να σημειώσουμε ότι δεν υπάρχει καμία μέθοδος *doHead*. Κι αυτό επειδή το σύστημα χρησιμοποιεί αυτόματα τις τοποθετήσεις γραμμών και headers θέσης για την `doGet` για να απαντήσει στα *HEAD* αιτήματα.

Η μέθοδος *destroy*

Ο Server μπορεί να αποφασίσει να αφαιρέσει μια προηγουμένως φορτωμένη περίπτωση servlet, ίσως επειδή καλείται ρητά να κάνει έτσι από τον Server administrator, ή ίσως επειδή το servlet είναι αδρανές για ένα μεγάλο χρονικό διάστημα. Προτού το πραγματοποιήσει εντούτοις, καλεί την μέθοδο *destroy* του servlet. Αυτή η μέθοδος δίνει στο servlet μας, μια

πιθανότητα να κλείσει τις συνδέσεις των βάσεων δεδομένων, να σταματήσει τα νήματα υποβάθρου, να γράψει στους καταλόγους Cookies ή να χτυπήσει τις αριθμήσεις στο δίσκο, και να εκτελέσει άλλες τέτοιες δραστηριότητες καθαρισμού. Είναι αξιοσημείωτο να γνωρίσουμε εντούτοις, ότι είναι εφικτό για τον Server του δικτύου να συντριφθεί.

Μετά από αυτά τελικά, δεν γράφονται όλοι οι Servers δικτύου στις αξιόπιστες γλώσσες προγραμματισμού όπως την Java, που μερικές γράφονται στις γλώσσες (όπως αυτές που ονομάζονται μετά από τα γράμματα της αλφαβήτου) όπου είναι εύκολο να διαβαστούν ή να γραφτούν μακριά οι άκρες των πινάκων, να γίνουν τα παράνομα typecasts, ή να υπάρξουν οι ταλαντευόμενοι δείκτες λόγω των λαθών αποκατάστασης της μνήμης.

Εκτός αυτού, ούτε η τεχνολογία της Java δεν θα αποτρέψει κάποια από την τρικλοποδιά πάνω από το καλώδιο δύναμης που περνά στον υπολογιστή. Έτσι, δεν μετράμε στην *destroy* ως τον μοναδικό μηχανισμό για την αποθήκευση των καταστάσεων στο δίσκο. Οι δραστηριότητες όπως ο υπολογισμός του χτυπήματος ή των συσσωρευμένων καταλόγων των τιμών των Cookies που δείχνουν την ειδική πρόσβαση που πρέπει επίσης, φιλενεργά να γράψουν την κατάστασή τους στο δίσκο περιοδικά.

3.11. Χειρισμός των αιτημάτων των Clients: Form Data

Κάθε φορά που θα έχουμε χρησιμοποιήσει μια μηχανή αναζήτησης, αν επισκεφτούμε ένα on-line βιβλιοπωλείο, ακολουθημένα αποθέματα στον Ιστό, ή ρωτήσουμε μια βασισμένη στο WEB Site για τα αποσπάσματα στα αεροπορικά εισιτήρια, θα έχουμε πιθανώς παρατηρήσει αστεία URLs όπως για παράδειγμα το URL http://host/path?user=Marty+Hall&origin=bwi&dest=lax. Το μέρος μετά από το σημάδι ερώτησης (δηλ., user=Marty+Hall&origin=bwi&dest=lax) είναι γνωστό σαν *form data* (ή *query data*) κι είναι ο πιο κοινός τρόπος να παρθούν οι πληροφορίες από μια ιστοσελίδα σε ένα server-side πρόγραμμα. Τα form data μπορούν να συνδεθούν με το τέλος του URL μετά από ένα σημάδι ερώτησης (όπως παραπάνω), για *GET* τα αιτήματα, ή να σταλούν στον Server σε μια χωριστή γραμμή, για *POST* αιτήματα.

Η εξαγωγή των αναγκαίων πληροφοριών από αυτά τα form data είναι παραδοσιακά ένα από τα πιο κουραστικά μέρη του προγραμματισμού της CGI. Καταρχήν, πρέπει να διαβάσουμε τα στοιχεία με ένα τρόπο για τα *GET* αιτήματα (στην παραδοσιακή CGI, αυτό είναι συνήθως μέσω της μεταβλητής περιβάλλοντος *QUERY_STRING*) κι ένα διαφορετικό τρόπο για τα *POST* αιτήματα (με την ανάγνωση της τυποποιημένης εισαγωγής στην παραδοσιακή CGI). Δεύτερον, πρέπει να τεμαχίσουμε τα ζευγάρια στα &, να χωρίσουμε έπειτα τα ονόματα των παραμέτρων (αριστερά των ίσων σημαδιών) από τις τιμές των παραμέτρων (δεξιά των ίσων σημαδιών).

Τρίτον, πρέπει να αποκωδικοποιήσουμε τις τιμές του URL. Στους alphanumeric χαρακτήρες στέλνεται αμετάβλητο, αλλά τα διαστήματα μετατρέπονται προστιθέμενων των σημαδιών κι άλλων χαρακτήρων τα οποία μετατρέπονται σε *%XX* όπου *XX* είναι η τιμή ASCII (ή ISO Latin-1) του χαρακτήρα, στο δεκαεξαδικό σύστημα. Κατόπιν, το server-side πρόγραμμα πρέπει να αντιστρέψει τη διαδικασία. Παραδείγματος χάριν, εάν κάποιο εισάγει μια τιμή "*~hall*, *~gates*, και *~mcneal*y" σε ένα πεδίο κειμένου μαζί με το όνομα users σε μια HTML φόρμα, τα δεδομένα στέλνονται ως "*users=%7Ehall%2C+%7Egates%2C+and+%7Emcneal*y", και το server-side πρόγραμμα πρέπει να ανασυγκροτήσει την αρχική συμβολοσειρά. Τέλος, ο τέταρτος λόγος για τον οποίο η ανάλυση των form data είναι κουραστική είναι ότι οι τιμές μπορούν να παραλειφθούν (π.χ., "*param1=val1&param2=&param3=val3*") ή μια παράμετρος μπορεί να έχει περισσότερες από μια τιμές (π.χ., "*param1=val1&param2=val2&param1=val3*"), κι έτσι ο αναλυμένος κώδικάς μας χρειάζεται ειδικές περιπτώσεις για αυτές τις καταστάσεις.

3.11.1. Ανάγνωση των Form Data από τα Servlets

Ένα από τα συμπαθητικά χαρακτηριστικά γνωρίσματα των servlets είναι ότι όλη αυτή η ανάλυση της φόρμας αντιμετωπίζεται αυτόματα. Καλούμε απλά τη μέθοδο *getParameter* του HTTP-ServletRequest, παρέχοντας το case-sensitive όνομα της παραμέτρου σαν όρισμα. Χρησιμοποιούμε την *getParameter* ακριβώς με τον ίδιο τρόπο όταν τα δεδομένα στέλνονται από *GET* όπως ακριβώς κάνουμε κι όταν στέλνονται από *POST*. Το servlet γνωρίζει ποια μέθοδος αιτήματος χρησιμοποιήθηκε και κάνει αυτόματα την σωστή κίνηση στο παρασκήνιο. Η επιστρεφόμενη τιμή είναι μια συμβολοσειρά που αντιστοιχεί στην URL-αποκωδικοποιημένη τιμή του πρώτου εκείνου περιστατικού του ονόματος της παραμέτρου. Μια κενή συμβολοσειρά επιστρέφεται εάν η παράμετρος υπάρχει αλλά δεν έχει καμία τιμή, και `null` επιστρέφεται εάν δεν υπήρξε καμία τέτοια παράμετρος. Εάν η παράμετρος θα μπορούσε ενδεχομένως να έχει περισσότερες από μια τιμές, πρέπει να καλέσουμε την *getParameterValues* (που επιστρέφει ένα πίνακα συμβολοσειρών) αντί της *getParameter* (που επιστρέφει μια ενιαία συμβολοσειρά). Η επιστρεφόμενη τιμή της *getParameterValues* είναι `null` για τα ανύπαρκτα ονόματα παραμέτρων κι είναι ένας πίνακας ενός-στοιχείου όταν έχει η παράμετρος μόνο μια ενιαία τιμή. Τα ονόματα των παραμέτρων είναι case sensitive, κι έτσι παραδείγματος χάριν, η `request.getParameter("Param1")` κι η `request.getParameter("param1")` δεν είναι ανταλλάξιμες.

Τέλος, αν και τα περισσότερα πραγματικά servlets ψάχνουν ένα συγκεκριμένο σύνολο ονομάτων των παραμέτρων, για λόγους διόρθωσης είναι μερικές φορές χρήσιμο να αποκτηθεί ένας πλήρης κατάλογος. Χρησιμοποιούμε την *getParameterNames* για να πάρουμε αυτόν τον κατάλογο υπό μορφή Enumeration, κάθε είσοδος της οποίας μπορεί να γίνει προσαρμογή τύπου σε μια συμβολοσειρά και να χρησιμοποιηθεί σε μια κλήση *getParameter* ή *getParameterValues*. Σε αυτό το σημείο ακριβώς πρέπει να σημειώσουμε ότι το HttpServletRequest API δεν διευκρινίζει τη διαταγή στην οποία τα ονόματα εμφανίζονται μέσα σε εκείνη την Enumeration.

3.12. Χειρισμός των αιτημάτων των Clients: Headers HTTP αιτήματος

3.12.1. Ανάγνωση των αιτημάτων των Headers από τα Servlets

Η ανάγνωση των headers είναι απλή καλώντας ακριβώς τη μέθοδο *getHeader* της *HttpServletRequest*, η οποία επιστρέφει μια συμβολοσειρά εάν το διευκρινισμένο header παρέχηκε με αυτό το αίτημα, `null` διαφορετικά. Τα ονόματα των headers δεν είναι case sensitive. Έτσι, παραδείγματος χάριν, οι εντολές `request.getHeader("Connection")` και `request.getHeader("connection")` είναι ανταλλάξιμες.

Αν κι η *getHeader* είναι ο γενικής χρήσης τρόπος να διαβαστούν τα εισερχόμενα headers, υπάρχουν μερικά headers που τόσο συνήθως χρησιμοποιούνται ότι έχουν τις ειδικές μεθόδους προσπέλασης στην *HttpServletRequest*.

getCookies

Η μέθοδος *getCookies* επιστρέφει το περιεχόμενο του header Cookies, που αναλύεται και που αποθηκεύεται σε έναν πίνακα αντικειμένων *Cookies*.

getAuthType και *getRemoteUser*

Οι μέθοδοι *getAuthType* και *getRemoteUser* σπάζουν το header *Authorization* στα συστατικά κομμάτια της.

getContentLength

Η μέθοδος *getContentLength* επιστρέφει την τιμή του header *Content-Length* (σαν `int`).

getContentType

Η μέθοδος *getContentType* επιστρέφει την τιμή του header *Content-Type* (σαν συμβολοσειρά).

getDateHeader* και *getIntHeader

Οι μέθοδοι *getDateHeader* και *getIntHeader* διαβάζουν το διευκρινισμένο header και μετατρέπουν έπειτα τις `date` και τις τιμές `int`, αντίστοιχα.

getHeaderNames

Παρά να ανατρέξουμε ένα κατάλληλο header, μπορούμε να χρησιμοποιήσουμε τη μέθοδο *getHeaderNames* για να πάρουμε ένα Enumeration όλων των ονομάτων headers που παραλαμβάνονται με αυτό το κατάλληλο αίτημα.

getHeaders

Στις περισσότερες περιπτώσεις, κάθε όνομα header εμφανίζεται μόνο μια φορά στο αίτημα. Περιστασιακά εντούτοις, ένα header μπορεί να εμφανιστεί σε πολλαπλούς χρόνους, με κάθε περιστατικό απαριθμώντας μια χωριστή τιμή. Η *Accept-Language* είναι ένα τέτοιο παράδειγμα. Εάν ένα όνομα header επαναλαμβάνεται στο αίτημα, τα servlets έκδοσης 2.1 δεν μπορούν να έχουν πρόσβαση στις πιο πρόσφατες τιμές χωρίς ανάγνωση του ακατέργαστου ρεύματος εισόδου, δεδομένου ότι η *getHeader* επιστρέφει την τιμή του πρώτου περιστατικού του header μόνο. Στην έκδοση 2.2 εντούτοις, η *getHeaders* επιστρέφει ένα Enumeration των τιμών όλων των περιστατικών του header. Τέλος, εκτός από να ανατρέξουμε τα headers αιτήματος, μπορούμε να πάρουμε τις πληροφορίες για η ίδια την κύρια γραμμή αιτήματος, επίσης, με τη βοήθεια των μεθόδων σε HTTP-ServletRequest.

getMethod

Η μέθοδος *getMethod* επιστρέφει την κύρια μέθοδο αιτήματος (κανονικά *GET* ή *POST*, αλλά πράγματα όπως *HEAD*, *PUT*, και *DELETE* είναι εφικτά).

getRequestURI

Η μέθοδος *getRequestURI* επιστρέφει το μέρος του URL που έρχεται μετά από το host και το port αλλά πριν από τα form data. Παραδείγματος χάριν, για ένα URL http://randomhost.com/servlet/search.BookSearch,getRequestURI θα επέστρεφε /servlet/search.BookSearch.

getProtocol

Τελικά, η μέθοδος *getProtocol* επιστρέφει το τρίτο μέρος της γραμμής αιτήματος, η οποία είναι γενικά HTTP / 1.0 ή HTTP / 1.1. Τα servlets πρέπει συνήθως να ελέγξουν την *getProtocol* πριν διευκρινίσουν τα headers απάντησης που είναι συγκεκριμένες για το HTTP 1.1.

3.12.2. Headers αιτήματος HTTP 1.1

Η πρόσβαση στα headers αιτήματος επιτρέπουν στα servlets να εκτελέσουν διάφορες βελτιστοποιήσεις και να παρέχουν διάφορα χαρακτηριστικά γνωρίσματα όχι πιθανά. Αυτό το τμήμα παρουσιάζει κάθε ένα από τα πιθανά headers αιτήματος HTTP 1.1 μαζί με μια συνοπτική περίληψη για το πώς τα servlets μπορούν να τις χρησιμοποιήσουν. Τα εξής τμήματα δίνουν τα πιο λεπτομερή παραδείγματα. Πρέπει να σημειώσουμε ότι το HTTP 1.1 υποστηρίζει superset των headers που επιτρέπονται στο HTTP 1.0. Υπάρχουν διάφορες

θέσεις που τα επίσημα RFCs είναι αρχειοθετημένα on-line, όπου το καλύτερο στοίχημά είναι να αρχίσουν στην ηλεκτρονική διεύθυνση http://www.rfc-editor.org/ και να πάρουν ένα τρέχοντα κατάλογο των Sites αρχείων.

Accept

Αυτό το header διευκρινίζει τους τύπους MIME που ο browser ή ο άλλος client μπορεί να χειριστεί. Ένα servlet που μπορεί να επιστρέψει ένα πόρο με περισσότερα από ένα format μπορεί να εξετάσει το *Accept* header για να αποφασίσει ποιο format θα χρησιμοποιήσει. Παραδείγματος χάριν, οι εικόνες με το format PNG έχουν μερικά πλεονεκτήματα συμπίεσης πέρα από εκείνους στο GIF, αλλά μόνο μερικοί browsers υποστηρίζουν το PNG. Εάν είχαμε εικόνες και με τα δύο formats, ένα servlet θα μπορούσε να καλέσει `request.getHeader("Accept")`, να ελέγχει για image/png, κι εάν βρει, να χρησιμοποιήσει xxx.png ονόματα αρχείου σε όλα τα στοιχεία IMG που παράγει. Διαφορετικά θα χρησιμοποιούσε xxx.gif.

Accept-Charset

Αυτό το header δείχνει τα σύνολα χαρακτήρα (π.χ., ISO-8859-1) που ο browser μπορεί να χρησιμοποιήσει.

Accept-Encoding

Αυτό το header υποδεικνύει τους τύπους κωδικοποιήσεων που ο client ξέρει πώς να χειριστεί. Εάν λαμβάνει αυτό το header, ο Server είναι ελεύθερος να κωδικοποιήσει τη σελίδα με τη χρησιμοποίηση συγκεκριμένου format (συνήθως για να μειώσει το χρόνο μετάδοσης), στέλνοντας το *Content-Encoding* header απάντησης για να δείξει ότι έτσι το έχει κάνει. Αυτός ο τύπος κωδικοποίησης είναι απολύτως ευδιάκριτος από τον τύπο MIME του πραγματικού εγγράφου (όπως διευκρινίζεται στο header απάντησης *Content-Type*), δεδομένου ότι αυτή η κωδικοποίηση αντιστρέφεται προτού να αποφασίσει ο browser τι να κάνει με το περιεχόμενο. Αφ' ετέρου, χρησιμοποιώντας μια κωδικοποίηση ο browser δεν καταλαβαίνει τα αποτελέσματα στις συνολικά ακατανόητες σελίδες. Συνεπώς, είναι κρίσιμο ότι ελέγχουμε ρητά το *Accept-Encoding* header πριν χρησιμοποιούμε οποιοδήποτε τύπο ικανοποιημένης κωδικοποίησης. Οι τιμές του gzip ή compress είναι οι δύο τυποποιημένες δυνατότητες. Η συμπίεση των σελίδων πριν τις επιστραφούν είναι πολύ πολύτιμη υπηρεσία επειδή ο χρόνος αποκωδικοποίησης είναι πιθανό να είναι μικρός έναντι της αποταμίευσης στο χρόνο μετάδοσης.

Accept-Language

Αυτό το header διευκρινίζει τις προτιμημένες γλώσσες του client,σε περίπτωση που το servlet μπορεί να παράγει τα αποτελέσματα σε περισσότερες από μια γλώσσες. Η τιμή του header πρέπει να είναι ένας από τους τυποποιημένους γλωσσικούς κώδικες όπως en, en-us, da, κ.λπ.

Authorization

Αυτό το header χρησιμοποιείται από τους clients για να προσδιορίσουν τους εαυτούς τους κατά την πρόσβαση σε ιστοσελίδες που προστατεύονται από κωδικό.

Cache-Control

Αυτό το header μπορεί να χρησιμοποιηθεί από τον client για να διευκρινίσει διάφορες επιλογές για το πώς οι σελίδες πρέπει να είναι εναποθηκευμένες από proxy Servers. Το header αιτήματος αγνοείται συνήθως από τα servlets, αλλά το header απάντησης *Cache-Control*

μπορεί να είναι χρήσιμη να δείξει ότι μια σελίδα αλλάζει συνεχώς και δεν πρέπει να εναποθηκευθεί.

Connection

Αυτό το header λέει εάν ο client μπορεί ή όχι να χειριστεί τις επίμονες συνδέσεις HTTP. Αυτές αφήνουν τον client ή άλλο browser να ανακτήσει πολλαπλά αρχεία (π.χ., ένα αρχείο HTML και διάφορες σχετικές εικόνες) με μια ενιαία σύνδεση υποδοχής, που σώζει τα γενικά έξοδα της διαπραγμάτευσης διάφορων ανεξάρτητων συνδέσεων. Με ένα αίτημα HTTP 1.1, προεπιλεγμένες είναι οι επίμονες συνδέσεις, κι ο client πρέπει να διευκρινίσει μια τιμή του *close* για αυτό το header για να χρησιμοποιήσει τις παλιές συνδέσεις. Στην έκδοση HTTP 1.0, μια τιμή των keep-alive σημαίνει ότι οι επίμονες συνδέσεις πρέπει να χρησιμοποιηθούν.

Κάθε αίτημα HTTP οδηγεί σε μια νέα επίκληση ενός servlet, ανεξάρτητα από εάν το αίτημα είναι μια χωριστή σύνδεση. Δηλαδή ο Server επικαλείται το servlet μόνο αφού έχει διαβάσει ήδη ο Server το αίτημα HTTP. Αυτό σημαίνει ότι τα servlets χρειάζονται τη βοήθεια από τον Server για να χειριστούν τις επίμονες συνδέσεις. Συνεπώς, η εργασία του servlet είναι ακριβώς να καταστεθεί πιθανό για τον Server να χρησιμοποιηθούν οι επίμονες συνδέσεις, το οποίο γίνεται με την αποστολή ενός header απάντησης *Content-Length*.

Content-Length

Αυτό το header ισχύει μόνο στα *POST* αιτήματα και δίνει το μέγεθος των *POST* στοιχείων στα Bytes. Παρά την κλήση της `request.getIntHeader("Content-Length")`, μπορούμε απλά να χρησιμοποιήσουμε την `request.getContentLength()`. Εντούτοις, δεδομένου ότι τα servlets φροντίζουν την ανάγνωση των form data για μας, είναι απίθανο να χρησιμοποιήσουμε αυτό το header ρητά.

Content-Type

Αν κι αυτό το header χρησιμοποιείται συνήθως στις απαντήσεις από τον Server, μπορεί επίσης, να είναι μέρος των αιτημάτων του client όταν επισυνάπτει ο client ένα έγγραφο σαν *POST* δεδομένα ή υποβάλλοντας *PUT* αιτήματα. Μπορούμε να έχουμε πρόσβαση σε αυτό το header με τη μέθοδο στενογραφίας *getContentType* της *HttpServletRequest*.

Cookie

Αυτό το header χρησιμοποιείται για να επιστρέψει τα Cookies στους Servers που τα έστειλαν προηγουμένως στον browser. Τεχνικά, το Cookie δεν είναι μέρος της έκδοσης HTTP 1.1. Ήταν αρχικά μια επέκταση του Netscape αλλά τώρα υποστηρίζεται ευρέως, και συμπεριλαμβάνεται και στον Netscape και στον εξερευνητή Διαδικτύου(Internet Explorer).

Expect

Αυτό το σπάνια χρησιμοποιημένο header αφήνει τον Client να πει στον Server ποια είδη συμπεριφορών αναμένει. Μια τυποποιημένη τιμή για αυτό το header, 100-continue, στέλνεται από ένα browser που θα στέλνει ένα συνημμένο έγγραφο και θέλει να ξέρει εάν ο Server θα το δεχτεί. Ο Server πρέπει να στείλει ένα status code είτε 100 (Continue) είτε 417 (Expectation Failed) σε κάθε τέτοια την περίπτωση.

From

Αυτό το header δίνει τη διεύθυνση ηλεκτρονικού ταχυδρομείου του ατόμου που είναι αρμόδιος για το αίτημα HTTP. Οι browsers δεν στέλνουν αυτό το header, αλλά οι Web spiders (ρομπότ) συχνά το θέτουν «ως ευγένεια» για να βοηθήσουν να προσδιορίσουν την πηγή υπερφόρτωσης Servers ή να επαναλάβουν τα ανάρμοστα αιτήματα.

Host

Οι browsers απαιτούνται για να διευκρινίσουν αυτό το header, το οποίο δείχνει το host και το port όπως δίνεται στο αρχικό URL. Λόγω του αιτήματος για την αποστολή και τις μηχανές που έχουν τα πολλαπλά hostnames, είναι αρκετά δυνατό ότι ο Server δεν θα μπορούσε ειδάλλως να καθορίσει αυτές τις πληροφορίες. Αυτό το header δεν είναι νέο στην έκδοση HTTP 1.1, αλλά στην έκδοση HTTP 1.0 ήταν προαιρετικό, όχι απαιτούμενο.

If-Match

Αυτό το σπάνια χρησιμοποιημένο header ισχύει πρώτιστα στα *PUT* αιτήματα. Ο Client μπορεί να παρέχει ένα κατάλογο ετικετών οντοτήτων όπως επιστρέφεται από το header απάντησης *ETag*, κι η λειτουργία εκτελείται μόνο εάν ένα από αυτά ταιριάζει.

If-Modified-Since

Αυτό το header δείχνει ότι ο client θέλει τη σελίδα μόνο εάν την έχουν αλλάξει μετά από την καθορισμένη ημερομηνία. Αυτή η επιλογή είναι πολύ χρήσιμη επειδή αφήνει τα cache έγγραφα των browsers και τα «ξαναφορτώνει» στο δίκτυο μόνο όταν αυτά έχουν αλλάξει. Εντούτοις, τα servlets δεν πρέπει να ασχοληθούν άμεσα με αυτό το header. Αντ' αυτού, πρέπει ακριβώς να εφαρμόσουν τη μέθοδο *getLastModified* για να έχουν τις ημερομηνίες τροποποίησης συστημάτων αυτόματα.

If-None-Match

Αυτό το header είναι όπως το *If-Match*, εκτός από το ότι η λειτουργία πρέπει να εκτελεσθεί μόνο εάν καμία ετικέτα οντοτήτων δεν ταιριάζει.

If-Range

Αυτό το σπάνια χρησιμοποιημένο header αφήνει ένα client που έχει ένα μερικό αντίγραφο ενός εγγράφου να ζητήσει είτε τα μέρη που λείπουν (εάν είναι αμετάβλητο) ή ένα ολόκληρο νέο έγγραφο (εάν έχει αλλάξει από μια καθορισμένη ημερομηνία).

If-Unmodified-Since

Αυτό το header είναι όπως το *If-Modified-Since* στην αντιστροφή, που δείχνει ότι η λειτουργία πρέπει να πετύχει μόνο εάν το έγγραφο είναι παλαιότερο από την καθορισμένη ημερομηνία. Χαρακτηριστικά, η *If-Modified-Since* χρησιμοποιείται για τα *GET* αιτήματα ("δώστε μου το έγγραφο μόνο εάν είναι νεότερο από την εναποθηκευμένη έκδοσή μου"), ενώ η *If-Unmodified-Since* χρησιμοποιείται για τα *PUT* αιτήματα ("ενημερώστε το παρόν έγγραφο μόνο εάν κανένας άλλος δεν το έχει αλλάξει από τότε που παρήχθει").

Pragma

Ένα header *Pragma* με μια τιμή `no-cache` δείχνει ότι ένα servlet που ενεργεί ως proxy πρέπει να διαβιβάσει το αίτημα ακόμα κι αν έχει ένα τοπικό αντίγραφο. Η μόνη τυποποιημένη τιμή για αυτό το header είναι `no-cache`.

Proxy-Authorization

Αυτό το header αφήνει τους clients να προσδιοριστούν proxies που την απαιτούν. Τα servlets αγνοούν χαρακτηριστικά αυτό το header, χρησιμοποιώντας την *Authorization* αντ' αυτού.

Range

Αυτό το σπάνια χρησιμοποιημένο header αφήνει έναν client που έχει ένα μερικό αντίγραφο ενός εγγράφου να ζητήσει μόνο τα μέρη που λείπουν.

Referer

Αυτό το header δείχνει το URL της αναφερόμενης ιστοσελίδας. Παραδείγματος χάριν, εάν είμαστε στην ιστοσελίδας 1 και χτυπάμε σε μια σύνδεση της ιστοσελίδας 2, το URL της ιστοσελίδας 1 συμπεριλαμβάνεται στο header *Referer* όταν ο browser ζητά την ιστοσελίδας 2. Όλοι οι σημαντικοί browsers θέτουν αυτό το header, κι έτσι είναι ένας χρήσιμος τρόπος για να ξέρουμε από που προήλθαν τα αιτήματα. Αυτή η ικανότητα είναι χρήσιμη για τους διαφημιστές που παραπέμπουν τους ανθρώπους στο Site μας, για να αλλάξουν ελαφρώς το περιεχόμενο ανάλογα με το αναφερόμενο Site, ή απλά για παρακολούθηση. Στην τελευταία περίπτωση, οι περισσότεροι άνθρωποι στηρίζονται απλά στα αρχεία ημερολογίου Servers δικτύου, δεδομένου ότι το *Referer* καταγράφεται χαρακτηριστικά εκεί. Αν κι είναι χρήσιμο, μην στηριχθούμε σε μεγάλο ποσοστό στο header *Referer* δεδομένου ότι μπορεί να είναι εύκολα μια απάτη από έναν client. Τέλος, είναι αξιοσημείωτο το γεγονός ότι αυτό το header είναι *Referer,* όχι *Referrer*, λόγω ενός ορθογραφικού λάθους από έναν από τους αρχικούς συντάκτες HTTP.

Upgrade

Το header *Upgrade* αφήνει τον browser ή άλλο client να διευκρινίσει ένα πρωτόκολλο επικοινωνίας που προτιμά άνω της έκδοσης HTTP 1.1. Εάν επίσης, ο Server υποστηρίζει εκείνο το πρωτόκολλο, τότε κι ο client κι ο Server μπορούν να μεταστρέψουν τα πρωτόκολλα. Αυτός ο τύπος διαπραγμάτευσης πρωτοκόλλου σχεδόν πάντα εκτελείται προτού να επικαλεσθεί το servlet. Κατά συνέπεια, τα servlets φροντίζουν σπάνια για αυτό το header.

User-Agent

Αυτό το header προσδιορίζει τον browser ή άλλο client που υποβάλλει το αίτημα και μπορεί να χρησιμοποιηθεί για να επιστρέψει το διαφορετικό περιεχόμενο στους διαφορετικούς τύπους browsers. Να είμαστε προσεκτικοί σ' αυτής τη χρήση εντούτοις η στήριξη σε ένα σκληρό-κωδικοποιημένο κατάλογο εκδόσεων browser και σχετικών χαρακτηριστικών γνωρισμάτων μπορεί να κάνει αναξιόπιστο και hard-to-modify τον κώδικα servlet. Όποτε είναι εφικτό, χρησιμοποιούμε κάτι συγκεκριμένο στα headers HTTP αντ' αυτού. Παραδείγματος χάριν, αντί της προσπάθειας να αναφερθεί ποιοι browsers υποστηρίζουν gzip σε ποιες πλατφόρμες, ελέγχουμε απλά το *Accept-Encoding* header. Κατά γενική ομολογία, αυτό είναι όχι πάντα δυνατό, αλλά όταν δεν είναι, πρέπει να αναρωτηθούμε εάν το browser-specific χαρακτηριστικό γνώρισμα που χρησιμοποιούμε πραγματικά προσθέτει αρκετή αξία για να αξίζει το κόστος συντήρησης.

Οι περισσότερες εκδόσεις εξερευνητών Διαδικτύου απαριθμούν μια έκδοση "Mozilla" (Netscape) πρώτα στη γραμμή User-Agent τους, με την πραγματική έκδοση browser που απαριθμείται parenthetically. Αυτό γίνεται για τη συμβατότητα με JavaScript, όπου το header *User-Agent* χρησιμοποιείται μερικές φορές για να καθορίσει ποια χαρακτηριστικά γνωρίσματα JavaScript υποστηρίζονται. Επίσης, σε αυτό το σημείο κρατάμε το γεγονός ότι αυτό το header μπορεί να είναι εύκολα απάτη, ένα γεγονός που καλείται σε ερώτημα για την αξιοπιστία των Sites που χρησιμοποιούν αυτό το header για "να παρουσιάσουν" διείσδυση στην αγορά των διάφορων εκδόσεων browser. Εκατομμύρια δολάρια στο μάρκετινγκ των χρημάτων που οδηγούν στις στατιστικές θα μπορούσαν να διαστραφούν από ένα συγκεκριμένο client που γράφτηκε σε λιγότερο από μια ώρα, και θα έπρεπε να πάρουν εκείνους τους αριθμούς ως ακριβείς;

Via

Αυτό το header τίθεται σαν στόχος από gateways και proxies για να παρουσιάσει τα ενδιάμεσα Sites που πέρασε διαμέσω το αίτημα.

Warning

Αυτό το σπάνια χρησιμοποιημένο catchall header αφήνει τους clients να προειδοποιήσουν για την εναποθήκευση ή τα ικανοποιημένα λάθη μετασχηματισμού.

3.13. Παραγωγή των απαντήσεων των Servers: HTTP Status Codes

3.13.1. Διευκρίνιση των Status Codes

Όπως περιγράφεται ακριβώς, η γραμμή θέσης απάντησης HTTP αποτελείται από μια έκδοση HTTP, ένα status code, κι ένα σχετικό μήνυμα. Δεδομένου ότι το μήνυμα συνδέεται άμεσα με το status code κι η έκδοση HTTP καθορίζεται από τον Server, όλα όσα πρέπει να κάνει ένα servlet είναι να θέσει το status code. Ο τρόπος να γίνει αυτό είναι με τη μέθοδο *setStatus* της *HttpServletResponse*. Εάν η απάντησή μας περιλαμβάνει ένα ειδικό status code κι ένα έγγραφο, πρέπει να βεβαιωθούμε να καλέσουμε την *setStatus* πριν από το να επιστρέψει οτιδήποτε από το περιεχόμενο μέσω της *PrintWriter*. Αυτό επειδή μια απάντηση HTTP αποτελείται από τη γραμμή θέσης, ένα ή περισσότερα headers, μια κενή γραμμή, και το πραγματικό έγγραφο, σε εκείνη την διαταγή.

Τα headers μπορούν να εμφανιστούν σε οποιαδήποτε διαταγή, και τα servlets αποθηκεύουν τα headers και τα στέλνουν εντελώς ξαφνικά, κι έτσι είναι επιτρεπτό να τεθεί ο κώδικας επιτρεπτού καθεστώτος (μέρος της πρώτης γραμμής επιστρεφόμενης) ακόμα και μετά από τη ρύθμιση των headers. Αλλά τα servlets δεν αποθηκεύουν απαραιτήτως το ίδιο το έγγραφο, δεδομένου ότι οι χρήστες ίσως θελήσουν να δουν τα μερικά αποτελέσματα για τις μακροχρόνιες σελίδες. Στην έκδοση 2.1 της προδιαγραφής servlet, η έξοδος *PrintWriter* δεν αποθηκεύεται προσωρινά, κι έτσι την πρώτη φορά που χρησιμοποιείς την *PrintWriter*, είναι πάρα πολύ αργά να επιστρέψει και να θέσει τα headers. Στην έκδοση 2.2, οι μηχανές servlet επιτρέπουν να αποθηκεύσουν προσωρινά την έξοδο, αλλά το μέγεθος του buffer αφήνεται απροσδιόριστο. Μπορούμε να χρησιμοποιήσουμε τη μέθοδο *getBufferSize* της *HttpServletResponse* για να αποφασίσουμε το μέγεθος, ή τη *setBufferSize* για να το διευκρινίσουμε (μέγεθος). Στην έκδοση 2.2 με την αποθήκευση που επιτρέπεται, μπορούμε να θέσουμε τα status codes έως ότου γεμίζει ο buffer και έπειτα στέλνεται πραγματικά στον client. Εάν δεν είμαστε βέβαιοι εάν ο buffer έχει σταλεί, μπορούμε να χρησιμοποιήσουμε τη μέθοδο *isCommitted* που ελέγχει.

Η μέθοδος *setStatus* παίρνει ένα `int` (το status code) σαν όρισμα, αλλά αντί της χρησιμοποίησης των ρητών αριθμών, είναι σαφέστερο και πιο αξιόπιστο να χρησιμοποιήσει τις σταθερές που καθορίζονται στην *HttpServletResponse*. Το όνομα κάθε σταθεράς προέρχεται από το πρότυπο μήνυμα HTTP 1.1 για κάθε μια σταθερά, όλα κεφαλαία με ένα πρόθεμα του SC (για το status code) και τα διαστήματα αλλάζουν για να υπογραμμιστούν.

Κατά συνέπεια, δεδομένου ότι το μήνυμα για 404 " Not Found", η ισοδύναμη σταθερά στην *HttpServletResponse* είναι SC_NOT_FOUND. Στην έκδοση 2.1 της προδιαγραφής servlet, υπάρχουν τρεις εξαιρέσεις. Η σταθερά για τον κώδικα 302 προέρχεται από το μήνυμα HTTP 1.0 (Moved Temporarily), όχι το μήνυμα HTTP 1.1 (Found), κι οι σταθερές για τους κώδικες 307 (Temporary Redirect) και 416 (Requested Range Not Satisfiable) λείπουν συνολικά.

Η έκδοση 2.2 πρόσθεσε τη σταθερά για 416, αλλά οι ασυνέπειες για 307 και 302 παραμένουν.

Αν κι η γενική μέθοδος των status codes είναι απλά να κληθεί η `response.setStatus(int)`, υπάρχουν δύο κοινές περιπτώσεις όπου μια συντομότερη μέθοδος στην *HttpServletResponse* παρέχεται. Γνωρίζουμε ότι κι οι δύο μέθοδοι «πετάνε» `IOException`, ενώ η *setStatus* όχι.

public void sendError(int code, String message)

Η μέθοδος *sendError* στέλνει ένα status code (συνήθως 404) μαζί με ένα σύντομο μήνυμα που είναι αυτόματα σχηματοποιημένο μέσα σε ένα έγγραφο HTML και σταλμένο στον client.

public void sendRedirect(String url)

Η μέθοδος *sendRedirect* παράγει μια απάντηση 302 μαζί με ένα header *Location* που δίνει το URL του νέου εγγράφου. Με την έκδοση servlets 2.1, αυτό πρέπει να είναι ένα απόλυτο URL. Στην έκδοση 2.2, είτε ένα απόλυτο είτε ένα σχετικό URL επιτρέπεται και το σύστημα μεταφράζει αυτόματα σχετικά URLs σε απόλυτα πριν τοποθετήσει αυτά στο header *Location*. Ο καθορισμός ενός Status Code δεν σημαίνει απαραιτήτως ότι δεν χρειάζεται να επιστρέψουμε ένα έγγραφο. Παραδείγματος χάριν, αν κι οι περισσότεροι Servers παράγουν αυτόματα ένα μικρό "File Not Found" μήνυμα για 404 απαντήσεις, ένα servlet ίσως να θελήσει να προσαρμόσει αυτήν την απάντηση. Σε αυτό το σημείο δεν πρέπει να παραλείψουμε το γεγονός ότι εάν στέλνουμε την έξοδο, τότε πρέπει να καλέσουμε την *setStatus* ή την *sendError* πρώτα.

3.13.2. HTTP 1.1 Status Codes κι ο σκοπός τους

Τα εξής τμήματα περιγράφουν κάθε έναν από τα διαθέσιμα status codes για τη χρήση στα servlets που μιλούν στους clients HTTP 1.1, μαζί με το πρότυπο μήνυμα που συνδέεται με κάθε κώδικα. Μια καλή κατανόηση αυτών των κωδίκων μπορεί εντυπωσιακά να αυξήσει τις ικανότητες των servlets μας, κι έτσι πρέπει τουλάχιστον να ξαφρίσουμε τις περιγραφές για να δούμε ποιες επιλογές είναι στη διάθεσή μας. Μπορούμε να επιστρέψουμε για να πάρουμε τις λεπτομέρειες όταν είμαστε έτοιμοι να χρησιμοποιήσουμε μερικές από τις ικανότητες.

Η πλήρης προδιαγραφή HTTP 1.1 δίνεται σε RFC 2616, στο οποίο μπορούμε να έχουμε πρόσβαση on-line με τη μετάβαση στην ηλεκτρονική διεύθυνση http://www.rfc-editor.org/ και μετά από τις συνδέσεις με τα πιο πρόσφατα Sites αρχείων RFC. Οι κώδικες που είναι νέοι στην έκδοση HTTP 1.1 σημειώνονται, δεδομένου ότι πολλοί browsers υποστηρίζουν μόνο την έκδοση HTTP 1.0. Πρέπει μόνο να στείλουμε τους νέους κώδικες στους clients που υποστηρίζουν την έκδοση HTTP 1.1, όπως ελέγχεται με τον έλεγχο της *request.getRequestProtocol*. Το υπόλοιπο αυτού του τμήματος περιγράφει τα συγκεκριμένα διαθέσιμα status codes στην έκδοση HTTP 1.1. Αυτοί οι κώδικες εμπίπτουν σε πέντε γενικές κλάσεις:

100-199

Οι κώδικες 100s είναι ενημερωτικοί, δείχνοντας ότι ο client πρέπει να αποκριθεί με κάποια άλλη δράση.

200-299

Οι τιμές 200s δηλώνουν ότι το αίτημα ήταν επιτυχές.

300-399

Οι τιμές 300s χρησιμοποιούνται για τα αρχεία που έχουν κινήσει και περιλαμβάνουν συνήθως ένα header *Location* που δείχνει τη νέα διεύθυνση.

400-499

Οι τιμές 400s δείχνουν ένα λάθος από τον client.

500-599

Οι κώδικες 500s δηλώνουν ένα λάθος από τον Server.

Οι σταθερές στη *HttpServletResponse* που αντιπροσωπεύουν τους διάφορους κώδικες προέρχονται από τα πρότυπα μηνύματα που συνδέονται με τους κώδικες. Στα servlets, αναφερόμαστε συνήθως στα Status Codes μόνο με τη βοήθεια αυτών των σταθερών. Παραδείγματος χάριν, θα χρησιμοποιούσαμε την εντολή `response.setStatus(response.SC_NO_CONTENT)` παρά την εντολή `response.setStatus(204)`, δεδομένου ότι η τελευταία εντολή είναι ασαφής στους αναγνώστες κι είναι επιρρεπή σε τυπογραφικά λάθη. Εντούτοις, πρέπει να σημειώσουμε ότι οι Servers επιτρέπουν να ποικίλουν τα μηνύματα ελαφρώς, κι οι clients δίνουν προσοχή μόνο στην αριθμητική τιμή. Έτσι, παραδείγματος χάριν, ίσως να δούμε ένα Server να επιστρέψει μια γραμμή θέσης του HTTP/1.1 200 Document Follows αντί του HTTP/1.1 200 OK.

100 (Continue)

Εάν ο Server λαμβάνει ένα *Expect* header αιτήματος τιμής 100-continue, σημαίνει ότι ο client ρωτά εάν μπορεί να στείλει ένα συνημμένο έγγραφο σε μια αίτηση παρακολούθησης. Σε αυτή την περίπτωση, ο Server πρέπει είτε να αποκριθεί με τη θέση 100 (SC_CONTINUE) για να πει στον client να προχωρήσει είτε να χρησιμοποιήσει 417 (Expectation Failed) για να πει του browser ότι δεν θα δεχτεί το έγγραφο. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

101 (Switching Protocols)

Μια θέση 101 (SC_SWITCHING_PROTOCOLS) δείχνει ότι ο Server θα συμμορφωθεί με το header και την αλλαγή *Upgrade* σε ένα διαφορετικό πρωτόκολλο. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

200 (OK)

Μια τιμή 200 (SC_OK) σημαίνει ότι όλα είναι καλά. Το έγγραφο ακολουθεί για *GET* και *POST* αιτήματα. Αυτό το status είναι η προεπιλογή για τα servlets εάν δεν χρησιμοποιούμε την *setStatus*, θα πάρουμε 200.

201 (Created)

Ένα status code 201 (SC_CREATED) δηλώνει ότι ο Server δημιούργησε ένα νέο έγγραφο σε απάντηση στο αίτημα που το header *Location* πρέπει να δώσει το URL της.

202 (Accepted)

Μια τιμή 202 (SC_ACCEPTED) λέει στον client ότι το αίτημα ενεργεί, αλλά η επεξεργασία δεν είναι ακόμα πλήρης.

203 (Non-Authoritative Information)

Μια θέση 203 (SC_NON_AUTHORITATIVE_INFORMATION) δηλώνει ότι το έγγραφο επιστρέφεται κανονικά, αλλά μερικά από τα headers απάντησης είναι ανακριβείς δεδομένου ότι ένα αντίγραφο εγγράφων χρησιμοποιείται. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

204 (No Content)

Ένα status code 204 (SC_NO_CONTENT) ορίζει ότι ο browser πρέπει να συνεχίσει να επιδεικνύει το προηγούμενο έγγραφο επειδή κανένα νέο έγγραφο δεν είναι διαθέσιμο. Αυτή η συμπεριφορά είναι χρήσιμη εάν ο χρήστης «ξαναφορτώνει» περιοδικά μια σελίδα με τη πίεση του κουμπιού "Reload", και μπορούμε να καθορίσουμε ότι η προηγούμενη σελίδα είναι ήδη ενημερωμένη. Παραδείγματος χάριν, ένα servlet να κάνει κάτι παρεμφερή:

```
int pageVersion =

Integer.parseInt(request.getParameter("pageVersion"));

if (pageVersion >= currentVersion) {

response.setStatus(response.SC_NO_CONTENT);

} else {

// Create regular page

}
```

Εντούτοις, αυτή η προσέγγιση δεν λειτουργεί για τις σελίδες που «ξαναφορτώνονται» αυτόματα μέσω του *Refresh* header απάντησης ή της ισοδύναμης εντολής *<META HTTP-EQUIV="Refresh" ...>* εισόδου HTML, από την επιστροφή ενός μελλοντικού «ξαναφορτώματος» 204 θέσης στάσεων κώδικα. Στην JavaScript εντούτοις, το βασισμένο αυτόματο «ξαναφόρτωμα» θα μπορούσε ακόμα να λειτουργήσει σε αυτή την περίπτωση.

205 (Reset Content)

Μια τιμή 205 (SC_RESET_CONTENT) σημαίνει ότι δεν υπάρχει κανένα νέο έγγραφο, αλλά ο browser πρέπει να επαναρυθμίσει την εξέταση εγγράφων. Αυτό το Status Code χρησιμοποιείται στους browsers δύναμης για να καθαρίσει τα πεδία της φόρμας. Είναι νέο στην έκδοση HTTP 1.1.

206 (Partial Content)

Ένα status code 206 (SC_PARTIAL_CONTENT) στέλνεται όταν εκπληρώνει ο Server ένα μερικό αίτημα που περιλαμβάνει ένα header *Range*. Αυτή η τιμή είναι νέα στην έκδοση HTTP 1.1.

300 (Multiple Choices)

Μια τιμή 300 (SC_MULTIPLE_CHOICES) δηλώνει ότι το ζητούμενο έγγραφο μπορεί να βρεθεί σε διάφορες θέσεις, οι οποίες θα απαριθμηθούν στο επιστρεφόμενο έγγραφο. Εάν ο Server έχει μια προτιμημένη επιλογή, θα πρέπει να απαριθμηθεί στο header απάντησης *Location*.

301 (Moved Permanently)

Η θέση 301 (SC_MOVED_PERMANENTLY) δείχνει ότι το ζητούμενο έγγραφο είναι αλλού. Το νέο URL για το έγγραφο δίνεται στο header απάντησης *Location*. Οι browsers πρέπει αυτόματα να ακολουθήσουν σύνδεση με το νέο URL.

302 (Found)

Αυτή η τιμή είναι παρόμοια με την 301, εκτός από το ότι το URL που δίνεται από το header *Location* πρέπει να ερμηνευθεί ως προσωρινή αντικατάσταση, όχι μόνιμη. Στο σημείο αυτό πρέπει να παρατηρήσουμε το γεγονός ότι στην έκδοση HTTP 1.0, το μήνυμα κινήθηκε

προσωρινά αντί της τιμής Found, κι η σταθερά στην *HttpServletResponse* είναι SC_MOVED_TEMPORARILY, κι όχι η αναμενόμενη τιμή SC_FOUND.

Το status code 302 είναι πολύ χρήσιμο επειδή οι browsers ακολουθούν αυτόματα την αναφορά στο νέο URL που δίνεται στο header απάντησης *Location*. Είναι τόσο χρήσιμο, στην πραγματικότητα, που υπάρχει μια ειδική μέθοδος για αυτό, η *sendRedirect*.

Η χρησιμοποίηση της `response.sendRedirect (url)` έχει μερικά πλεονεκτήματα πέρα από τη χρησιμοποίηση της `response.setStatus(response.SC_MOVED_TEMPORARILY)` και της `response.setHeader("Location", url)`. Κατ' αρχάς, είναι μικρότερη κι ευκολότερη. Δεύτερον, με την *sendRedirect*, το servlet χτίζει αυτόματα μια σελίδα που περιέχει τη σύνδεση που παρουσιάζει στους παλαιότερους browsers ότι δεν ακολουθούν αυτόματα επαναπροσανατολισμούς. Τέλος, με την έκδοση 2.2 των servlets (η έκδοση J2EE), η *sendRedirect* μπορεί να χειριστεί σχετικά URLs, μεταφράζοντας τα αυτόματα σε απόλυτα. Εντούτοις, πρέπει να χρησιμοποιήσουμε ένα απόλυτο URL στην έκδοση 2.1.

Εάν επαναπροσανατολίζουμε το χρήστη σε μια άλλη σελίδα μέσα στο Site μας, πρέπει να περάσουμε το URL μέσω της μεθόδου encodeURL της *HttpServletResponse*. Κάνοντας το αυτό είναι μια απλή προφύλαξη σε περίπτωση που χρησιμοποιούμε πάντα την καταδίωξη συνόδου βασισμένη στο URL-ξαναγράψιμο. Το URL-ξαναγράψιμο είναι ένας τρόπος να ακολουθούμε τους χρήστες που θέτουν εκτός λειτουργίας τα Cookies ενώ είναι επί του site μας. Εφαρμόζεται με την προσθήκη των πρόσθετων πληροφοριών πορειών στο τέλος κάθε URL, αλλά το servlet σύνοδος-που ακολουθεί το API φροντίζει τις λεπτομέρειες αυτόματα.

Αυτό το status code χρησιμοποιείται μερικές φορές εναλλακτικά με το 301. Παραδείγματος χάριν, εάν ζητήσουμε λανθασμένα `http://host/~user`, μερικοί Servers θα απαντήσουν με κωδικό 301 ενώ άλλοι θα χρησιμοποιήσουν το 302.

Τεχνικά, οι browsers είναι μόνο υποτιθέμενοι για να ακολουθήσουν αυτόματα τον επαναπροσανατολισμό εάν το αρχικό αίτημα ήταν *GET*.

303 (See Other)

Η θέση 303 (SC_SEE_OTHER) είναι παρόμοια με την 301 και την 302, εκτός από το ότι εάν το αρχικό αίτημα ήταν *POST*, το νέο έγγραφο (που δίνεται στο *Location* header) πρέπει να ανακτηθεί με *GET*. Αυτός ο κώδικας είναι νέος στην έκδοση HTTP 1.1.

304 (Not Modified)

Όταν ένας client έχει ένα εναποθηκευμένο έγγραφο, μπορεί να εκτελέσει ένα υπό όρους αίτημα με την παροχή ενός *If-Modified-Since* header για να δείξει ότι θέλει μόνο το έγγραφο εάν το έχουν αλλάξει από την καθορισμένη ημερομηνία. Μια τιμή 304 (SC_NOT_MODIFIED) σημαίνει ότι η εναποθηκευμένη έκδοση είναι ενημερωμένη κι ο client πρέπει να το χρησιμοποιήσει (έγγραφο). Διαφορετικά, ο Server πρέπει να επιστρέψει το ζητούμενο έγγραφο με το κανονικό (200) status code. Τα servlets δεν πρέπει κανονικά να θέσουν αυτό το Status Code άμεσα. Αντ' αυτού, πρέπει να εφαρμόσουν τη μέθοδο *getLastModified* και να αφήσουν την προεπιλεγμένη μέθοδο *service* να χειριστεί τα υπό όρους αιτήματα που βασίζονται σε αυτήν την ημερομηνία τροποποίησης.

305 (Use Proxy)

Μια τιμή 305 (SC_USE_PROXY) δηλώνει ότι το ζητούμενο έγγραφο πρέπει να ανακτηθεί μέσω του πληρεξούσιου που απαριθμείται στο header *Location*. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

307 (Temporary Redirect)

Οι κανόνες για το πώς ένας browser πρέπει να χειριστεί μια θέση 307 είναι ίδιοι με εκείνους για τη 302. Η τιμή 307 προστέθηκε στην έκδοση HTTP 1.1 δεδομένου ότι πολλοί browsers ακολουθούν λανθασμένα τον επαναπροσανατολισμό σε μια απάντηση 302 ακόμα κι αν το αρχικό μήνυμα είναι ένα *POST*. Οι browsers υποτίθεται ότι πρέπει να ακολουθήσουν τον επαναπροσανατολισμό ενός *POST* αιτήματος μόνο όταν λαμβάνουν μια θέση απάντησης 303. Αυτή η νέα θέση προορίζεται να είναι σαφής: ακολουθούμε τα επαναπροσανατολισμένα *GET* και *POST* αιτήματα στην περίπτωση 303 απαντήσεων ακολουθούμε τα επαναπροσανατολισμένα *GET* αλλά όχι *POST* αιτήματα στην περίπτωση 307 απαντήσεων. Εδώ ακριβώς πρέπει να συγκρατήσουμε το γεγονός ότι για κάποιους λόγους δεν υπάρχει καμία σταθερά στην *HttpServletResponse* που να αντιστοιχεί σε αυτό το Status Code. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

400 (Bad Request)

Μια θέση 400 (SC_BAD_REQUEST) δείχνει την κακή σύνταξη στο αίτημα client.

401 (Unauthorized)

Μια τιμή 401 (SC_UNAUTHORIZED) δηλώνει ότι ο client προσπάθησε να έχει πρόσβαση σε μια προστατευμένη με κωδικό σελίδα χωρίς κατάλληλες προσδιοριστικές πληροφορίες στο header *Authorization*. Η απάντηση πρέπει να περιλάβει το *WWW-Authenticate* header.

403 (Forbidden)

Ένα status code 403 (SC_FORBIDDEN) σημαίνει ότι ο Server αρνείται να παρέχει τον πόρο, ανεξάρτητα από την έγκριση. Αυτή η θέση είναι συχνά το αποτέλεσμα των κακών άδειων αρχείων ή καταλόγου στον Server.

404 (Not Found)

Η κακόφημη θέση 404 (SC_NOT_FOUND) λέει στον client ότι κανένας πόρος δεν θα μπορούσε να βρεθεί σε εκείνη την διεύθυνση. Αυτή η τιμή είναι η τυποποιημένη απάντηση "καμιά τέτοια σελίδα". Είναι μια κοινή και χρήσιμη απάντηση ότι υπάρχει μια ειδική μέθοδος για αυτήν στην κλάση HttpServletResponse: *sendError("message")*. Το πλεονέκτημα της *sendError* άνω της *setStatus* είναι ότι, με την *sendError*, ο Server παράγει αυτόματα μια σελίδα λάθους που παρουσιάζει μήνυμα λάθους. Δυστυχώς εντούτοις, η συμπεριφορά προεπιλογής του Internet Explorer 5 πρόκειται να αγνοήσει τη σελίδα λάθους που στέλνουμε πίσω κι εκτείθεται ακόμα κι αν έρχεται σε αντίθεση με την προδιαγραφή HTTP. Για να απενεργοποιήσουμε αυτήν την ρύθμιση, πηγαίνουμε στις επιλογές των εργαλείων, επιλέγουμε τις επιλογές Διαδικτύου, έπειτα επιλέγουμε το Advanced tab, και σιγουρευόμαστε ότι η επιλογή “Show friendly HTTP error messages” δεν είναι τσεκαρισμένη. Δυστυχώς εντούτοις, μερικοί χρήστες δεν γνωρίζουν αυτήν την ρύθμιση, κι έτσι αυτό το "χαρακτηριστικό γνώρισμα" αποτρέπει τους περισσότερους χρήστες της έκδοσης Internet Explorer 5 από τη θέα οποιωνδήποτε πληροφοριακών μηνυμάτων που επιστρέφουμε. Άλλοι σημαντικοί browsers κι ο Internet Explorer 4 του επιδεικνύουν κατάλληλα τις server-generated σελίδες λάθους.

405 (Method Not Allowed)

Μια τιμή 405 (SC_METHOD_NOT_ALLOWED) δείχνει ότι η μέθοδος αιτήματος (*GET*, *POST*, *HEAD*, *PUT*, *DELETE*, κ.λπ.) δεν επιτράπηκε για αυτόν τον κατάλληλο πόρο. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

406 (Not Acceptable)

Μια τιμή 406 (SC_NOT_ACCEPTABLE) δηλώνει ότι ο ζητούμενος πόρος έχει ένα τύπο MIME ασυμβίβαστο με τους τύπους που διευκρινίζονται από τον client *Accept* του στο header. Η τιμή 406 είναι νέα στην έκδοση HTTP 1.1.

407 (Proxy Authentication Required)

Η τιμή 407 (SC_PROXY_AUTHENTICATION_REQUIRED) είναι παρόμοια με την 401, αλλά χρησιμοποιείται από proxy Servers. Αυτό δείχνει ότι ο client πρέπει να επικυρωθεί με τον proxy Server. Ο proxy Server επιστρέφει ένα *Proxy-Authenticate* header απάντησης στον client, η οποία οδηγεί στον browser επανασυνδέοντας με ένα header αιτήματος *Proxy-Authorization*. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

408 (Request Timeout)

Ο κώδικας 408 (SC_REQUEST_TIMEOUT) σημαίνει ότι πήρε πάρα πολύ ώρα στον client για να τελειώσει το αίτημα. Είναι νέο στην έκδοση HTTP 1.1.

409 (Conflict)

Συνήθως συνδεμένη με *PUT* αιτήματα, η θέση 409 (SC_CONFLICT) χρησιμοποιείται για τις καταστάσεις όπως μια προσπάθεια να «φορτωθεί» μια ανακριβής έκδοση ενός αρχείου. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

410 (Gone)

Μια τιμή 410 (SC_GONE) λέει στον client ότι το ζητούμενο έγγραφο πηγαίνει και καμία διεύθυνση αποστολής δεν είναι γνωστή. Η θέση 410 διαφέρει από 404 στο ότι το έγγραφο είναι γνωστό μόνιμα, όχι μόνο μη διαθέσιμος για άγνωστους λόγους, όπως με την 404. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

411 (Length Required)

Μια θέση 411 (SC_LENGTH_REQUIRED) δηλώνει ότι ο Server δεν μπορεί να επεξεργαστεί το αίτημα (assumedly ένα *POST* αίτημα με ένα συνημμένο έγγραφο) εκτός αν ο client στείλει ένα *Content-Length* header που δείχνει πόσα στοιχεία στέλνονται στον Server. Αυτή η τιμή είναι νέα στην έκδοση HTTP 1.1.

412 (Precondition Failed)

Η θέση 412 (SC_PRECONDITION_FAILED) δείχνει ότι κάποια προϋπόθεση που διευκρινίστηκε στα headers αιτήματος ήταν `false`. Είναι νέο στην έκδοση HTTP 1.1.

413 (Request Entity Too Large)

Ένα status code 413 (SC_REQUEST_ENTITY_TOO_LARGE) λέει στον client ότι το ζητούμενο έγγραφο είναι μεγαλύτερο από αυτό που ο Server θέλει να χειριστεί τώρα. Εάν ο Server σκέφτεται ότι μπορεί να το χειριστεί αργότερα, θα πρέπει να συμπεριλάβει ένα *Retry-After* header απάντησης. Αυτή η τιμή είναι νέα στην έκδοση HTTP 1.1.

414 (Request URI Too Long)

Η θέση 414 (SC_REQUEST_URI_TOO_LONG) χρησιμοποιείται όταν το URI είναι πάρα πολύ μακρύ. Σε αυτό το πλαίσιο, "URI " σημαίνει το μέρος του URL που ήρθε μετά από τον host και το port στο URL. Παραδείγματος χάριν, στην ηλεκτρονική διεύθυνση http://www.y2k-

disaster.com:8080/we/look/silly/now/, το URI είναι /we/look/silly/now/. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

415 (Unsupported Media Type)

Μια τιμή 415 (SC_UNSUPPORTED_MEDIA_TYPE) σημαίνει ότι το αίτημα είχε ένα συνημμένο έγγραφο ενός τύπου που ο Server δεν ξέρει πώς να χειριστεί. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

416 (Requested Range Not Satisfiable)

Ένα status code 416 δηλώνει ότι ο client συμπεριέλαβε ένα ανικανοποίητο header *Range* στο αίτημα. Αυτή η τιμή είναι νέα στην έκδοση HTTP 1.1. Εκπληκτικά, η σταθερά που αντιστοιχεί σε αυτήν την τιμή παραλείφθηκε από την *HttpServletResponse* στην έκδοση 2.1 του servlet API.

417 (Expectation Failed)

Εάν ο Server λαμβάνει ένα *Expect* header αιτήματος τιμής 100-continue, σημαίνει ότι ο client ρωτά εάν μπορεί να στείλει ένα συνημμένο έγγραφο σε μια αίτηση παρακολούθησης. Σε αυτή την περίπτωση, ο Server πρέπει είτε να αποκριθεί με αυτήν την θέση (417) για να πει στον browser ότι δεν θα δεχτεί το έγγραφο ή θα χρησιμοποιήσει το 100 (SC_CONTINUE) για να πει στον client να προχωρήσει. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

500 (Internal Server Error)

Το 500 (SC_INTERNAL_SERVER_ERROR) είναι το γενικό “server is confused” status code. Προκύπτει συχνά από τα προγράμματα της CGI ή (heaven forbid!) servlets που συντρίβουν ή επιστρέφουν τα εσφαλμένα σχηματοποιημένα headers.

501 (Not Implemented)

Η θέση 501 (SC_NOT_IMPLEMENTED) ειδοποιεί τον client ότι ο Server δεν υποστηρίζει τη λειτουργία για να εκπληρώσει το αίτημα. Χρησιμοποιείται, παραδείγματος χάριν, όταν εκδίδει ο client μια εντολή όπως *PUT* που ο Server δεν υποστηρίζει.

502 (Bad Gateway)

Μια τιμή 502 (SC_BAD_GATEWAY) χρησιμοποιείται από τους Servers που ενεργούν ως proxies ή gateways που δείχνει ότι ο αρχικός Server πήρε μια κακή απάντηση από το μακρινό Server.

503 (Service Unavailable)

Ένα status code 503 (SC_SERVICE_UNAVAILABLE) δηλώνει ότι ο Server δεν μπορεί να αποκριθεί λόγω της συντήρησης ή της υπερφόρτωσης. Παραδείγματος χάριν, ένα servlet ίσως επιστρέψει αυτό το header εάν κάποια διαθέσιμη σύνδεση νημάτων ή βάση δεδομένων είναι αυτήν την περίοδο πλήρης. Ο Server μπορεί να παρέχει ένα *Retry-After* header για να πει στον client πότε να προσπαθήσει πάλι.

504 (Gateway Timeout)

Μια τιμή 504 (SC_GATEWAY_TIMEOUT) χρησιμοποιείται από τους Servers που ενεργούν ως proxies ή gateways που δείχνει ότι ο αρχικός Server δεν πήρε μια έγκαιρη απάντηση από το μακρινό Server. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

505 (HTTP Version Not Supported)

Ο κώδικας 505 (SC_HTTP_VERSION_NOT_SUPPORTED) σημαίνει ότι ο Server δεν υποστηρίζει την έκδοση του HTTP που ονομάζεται στη γραμμή αιτήματος. Αυτό το Status Code είναι νέο στην έκδοση HTTP 1.1.

3.14. Παραγωγή των απαντήσεων των Servers: Headers HTTP απαντήσεων

3.14.1. Ρύθμιση των Headers απάντησης από τα Servlets

Ο γενικότερος τρόπος να διευκρινιστούν τα headers είναι να χρησιμοποιηθεί η μέθοδος *setHeader* της *HttpServletResponse*. Αυτή η μέθοδος παίρνει δύο συμβολοσειρές: το όνομα των headers και την τιμή των headers. Όπως με τον καθορισμό των κωδίκων θέσης, πρέπει να διευκρινίσουμε τα headers πριν επιστρέψουμε το πραγματικό έγγραφο. Με την έκδοση 2.1 των servlets, σημαίνει ότι πρέπει να θέσουμε τα headers πριν από την πρώτη χρήση της *PrintWriter* ή της ακατέργαστης *OutputStream* που διαβιβάζει την περιεκτικότητα σε έγγραφα. Με την έκδοση servlets 2.2 (η έκδοση J2EE), η *PrintWriter* μπορεί να χρησιμοποιήσει ένα buffer, κι έτσι μπορούμε να θέσουμε τα headers μέχρι την πρώτη φορά που ο buffer γεμίζει.

Εκτός από τη γενικής χρήσης μέθοδο *setHeader*, η *HttpServlet-Response* έχει επίσης, δύο ειδικευμένες μεθόδους για να θέσει τα headers που περιέχουν τις ημερομηνίες και τους ακέραιους αριθμούς:

setDateHeader(String header, long milliseconds)

Αυτή η μέθοδος μας σώζει από το πρόβλημα της μετάφρασης μιας ημερομηνίας της Java στα χιλιοστά του δευτερολέπτου από το 1970 (όπως επιστρέφεται από την `System.currentTimeMillis`, την `Date.getTime`, ή την `Calendar.getTimeInMillis`) σε μια GMT χρονική συμβολοσειρά.

setIntHeader(String header, int headerValue)

Αυτή η μέθοδος μας διαθέτει τη δευτερεύουσα δυσχέρεια της μετατροπής ενός `int` σε μια συμβολοσειρά πριν παρεμβληθεί σε ένα header. Το HTTP επιτρέπει τα πολλαπλά περιστατικά του ίδιου ονόματος headers, και θέλουμε μερικές φορές να προσθέσουμε ένα νέο header παρά να αντικαταστήσουμε οποιαδήποτε υπάρχον header με το ίδιο όνομα. Παραδείγματος χάριν, είναι αρκετά κοινό να υπάρξουν πολλαπλά *Accept* και headers *Set-Cookie* που διευκρινίζουν τους διαφορετικούς υποστηριγμένους τύπους MIME και τα διαφορετικά Cookies, αντίστοιχα. Με την έκδοση servlets 2.1, η *set-Header*, η *setDateHeader* κι η *setIntHeader* προσθέτουν πάντα τα νέα headers, κι έτσι δεν υπάρχει κανένας τρόπος στα headers "unset" που τέθηκαν προηγουμένως(π.χ., με μια κληρονομημένη μέθοδο). Με την έκδοση servlets 2.2, η *setHeader*, η *setDateHeader*, κι η *setIntHeader* αντικαθιστούν τα οποιαδήποτε υπάρχοντα headers του ίδιου ονόματος, ενώ η *addHeader*, η *addDateHeader*, κι η *addIntHeader* προσθέτουν ένα header ανεξάρτητα από το εάν ένα header με εκείνο το όνομα υπάρχει ήδη. Εάν τυχόν μας ενοχλεί ένα συγκεκριμένο header το οποίο έχει τεθεί ήδη, τότε χρησιμοποιούμε την *containsHeader* στον έλεγχο.

Τέλος, η *HttpServletResponse* παρέχει επίσης, διάφορες μεθόδους ευκολίας για τα κοινά headers. Αυτές οι μέθοδοι συνοψίζονται ως εξής.

setContentType

Αυτή η μέθοδος θέτει το header *Content-Type* και χρησιμοποιείται από την πλειοψηφία των servlets.

setContentLength

Αυτή η μέθοδος θέτει το header *Content-Length*, η οποία είναι χρήσιμη εάν ο browser υποστηρίζει τις επίμονες (keep-alive) συνδέσεις HTTP.

addCookie

Αυτή η μέθοδος παρεμβάλλει ένα Cookie στο header *Set-Cookie*. Δεν υπάρχει καμία αντίστοιχη μέθοδος *setCookie*, δεδομένου ότι είναι φυσιολογικό να υπάρξουν πολλαπλές γραμμές *Set-Cookie*.

sendRedirect

Η μέθοδος *sendRedirect* θέτει το header *Location* καθώς επίσης, και το status code σε 302.

3.14.2. Headers απάντησης HTTP 1.1 κι η έννοια τους

Παρακάτω παρουσιάζεται μια περίληψη των headers απάντησης HTTP 1.1. Μια καλή κατανόηση αυτών των headers μπορεί να αυξήσει την αποτελεσματικότητα των servlets μας, κι έτσι πρέπει τουλάχιστον να ξαφρίσουμε τις περιγραφές για να δούμε ποιες επιλογές είναι στη διάθεσή μας. Μπορούμε να επιστρέψουμε για να πάρουμε τις λεπτομέρειες όταν είμαστε έτοιμοι να χρησιμοποιήσουμε τις ικανότητες. Αυτά τα headers είναι superset εκείνων που επιτρέπονται στην έκδοση HTTP 1.0. Τα ονόματα των headers δεν είναι case sensitive, αλλά γράφονται παραδοσιακά με το πρώτο γράμμα κάθε λέξης που κεφαλαιοποιείται.

Να είμαστε προσεκτικοί στα servlets γραψίματος των οποίων η συμπεριφορά εξαρτάται από τα headers απάντησης που είναι μόνο διαθέσιμα στην έκδοση HTTP 1.1, ειδικά εάν το servlet μας πρέπει να «τρέξει» στο WWW "σε μεγάλο," παρά σε ένα ενδοδίκτυο-πολλοί παλαιότεροι browsers υποστηρίζουν μόνο την έκδοση HTTP 1.0. Είναι καλύτερο να ελεγχθεί ρητά η έκδοση HTTP με την χρήση της *request.getRequestProtocol* πριν χρησιμοποιήσει τα νέα headers.

Accept-Ranges

Αυτό το header, που είναι νέο στην έκδοση HTTP 1.1, λέει στον client εάν δεχόμαστε ή όχι τα headers αιτήματος *Range*. Διευκρινίζουμε χαρακτηριστικά μια τιμή από Bytes για να δείξουμε ότι δεχόμαστε τα αιτήματα *Range*, και μια τιμή *none* για να δείξουμε ότι δεν τα δεχόμαστε.

Age

Αυτό το header χρησιμοποιείται από proxies για να δείξει πόσο πολύ καιρό πριν το έγγραφο παρήχθη από τον αρχικό Server. Είναι νέο στην έκδοση HTTP 1.1 και χρησιμοποιείται σπάνια από τα servlets.

Allow

Το *Allow* header διευκρινίζει τις μεθόδους αιτήματος (*GET*, *POST*, κ.λπ.) όπου ο Server υποστηρίζει. Απαιτείται για τις απαντήσεις με τιμή την 405 (Method Not Allowed). Η προεπιλεγμένη μέθοδος *service* των servlets παράγει αυτόματα αυτό το header για τα αιτήματα *OPTIONS*.

Cache-Control

Αυτό το χρήσιμο header λέει στον browser ή σε άλλο client τις περιστάσεις στις οποίες το έγγραφο απάντησης μπορεί ακίνδυνα να εναποθηκευθεί. Έχει τις ακόλουθες πιθανές τιμές:

• public: Το έγγραφο είναι cacheable, ακόμα κι αν οι κανονικοί κανόνες (π.χ., για τις password-protected σελίδες) δείχνουν ότι δεν πρέπει να είναι.

• private: Το έγγραφο είναι για έναν ενιαίο χρήστη και μπορεί μόνο να αποθηκευτεί σε ιδιωτικές (μη-μοιραζόμενες) κρυψώνες.

• no-cache: Το έγγραφο δεν πρέπει ποτέ να εναποθηκευθεί (δηλ., χρησιμοποιείται για να ικανοποιήσει ένα πιο πρόσφατο αίτημα). Ο Server μπορεί επίσης, να διευκρινίσει "*no-cache="header1,header2,...,headerN"*" για να δείξει τα headers που πρέπει να παραλειφθούν εάν μια εναποθηκευμένη απάντηση χρησιμοποιείται αργότερα. Οι browsers κανονικά δεν εναποθηκεύουν τα έγγραφα τα οποία ανακτήθηκαν από τα αιτήματα που περιλαμβάνουν form data.

Εντούτοις, εάν ένα servlet παράγει διαφορετικό περιεχόμενο για διαφορετικά αιτήματα ακόμα κι όταν τα ερωτήματα δεν περιέχουν κανένα form data, είναι κρίσιμο να πει στον browser να μην εναποθηκεύσει την απάντηση. Δεδομένου ότι οι παλαιότεροι browsers χρησιμοποιούσαν το header *Pragma* για αυτόν το λόγο, η χαρακτηριστική προσέγγιση ενός servlet είναι να τεθούν και τα δύο headers, όπως στο ακόλουθο παράδειγμα.

```
response.setHeader("Cache-Control", "no-cache");

response.setHeader("Pragma", "no-cache");
```

• no-store: Το έγγραφο δεν πρέπει ποτέ να εναποθηκευθεί και δεν πρέπει να αποθηκευτεί ακόμη και σε μια προσωρινή θέση στον δίσκο. Αυτό το header προορίζεται να αποτρέψει τα αμελή αντίγραφα των ευαίσθητων πληροφοριών.

• must-revalidate: Ο client πρέπει να τεκμηριώσει το έγγραφο με τον αρχικό Server, όχι μόνο ενδιάμεσοι proxies, κάθε φορά που χρησιμοποιείται.

• proxy-revalidate: Αυτό είναι το ίδιο με το header *must-revalidate*, εκτός από το ότι ισχύει μόνο για τις κοινές κρυψώνες.

• max-age=xxx: Το έγγραφο πρέπει να θεωρηθεί πολυδιατηρημένο μετά από xxx δευτερόλεπτα. Αυτό είναι μια κατάλληλη εναλλακτική λύση για το *Expires* header, αλλά συνεργάζεται μόνο με τους HTTP 1.1 clients. Εάν κι η *max-age* κι η *Expires* είναι παρούσες στην απάντηση, η τιμή της *max-age* παίρνει την προτεραιότητα.

• s-max-age=xxx: Οι κοινές κρυψώνες πρέπει να θεωρήσουν το έγγραφο πολυδιατηρημένο μετά από xxx δευτερόλεπτα. Το header *Cache-Control* είναι νέο στην έκδοση HTTP 1.1.

Connection

Μια τιμή `close` για αυτό το header απάντησης καθοδηγεί τον browser να μην χρησιμοποιήσει τις επίμονες συνδέσεις HTTP. Τεχνικά, οι επίμονες συνδέσεις είναι η προεπιλογή όταν υποστηρίζει ο client την έκδοση HTTP 1.1 και δεν διευκρινίζει ένα "*Connection: close*" header αιτήματος, ή όταν διευκρινίζει ένας client HTTP 1.0 "*Connection: keep-alive*". Εντούτοις, δεδομένου ότι οι επίμονες συνδέσεις απαιτούν ένα header απάντησης *Content-Length*, δεν υπάρχει κανένας λόγος για ένα servlet να χρησιμοποιήσει ρητά το header *Connection*. Απλώς παραλείπουμε το header *Content-Length* εάν δεν χρησιμοποιούμε τις επίμονες συνδέσεις.

Content-Encoding

Αυτό το header δείχνει τον τρόπο με τον οποίο η σελίδα κωδικοποιήθηκε κατά τη διάρκεια της μετάδοσης. Ο browser πρέπει να αντιστρέψει την κωδικοποίηση πριν αποφασίζει τι να κάνει με το έγγραφο. Η συμπίεση του εγγράφου με το gzip μπορεί να αποδειχτεί για πολλές φορές σωτήρια κατά τον χρόνο μετάδοσης.

Content-Language

Το *Content-Language* header δηλώνει τη γλώσσα στην οποία το έγγραφο γράφεται. Η τιμή του header πρέπει να είναι ένας από τους τυποποιημένους γλωσσικούς κώδικες όπως για παράδειγμα en, en-us, da, κ.λπ.

Content-Length

Αυτό το header δείχνει τον αριθμό των Bytes στην απάντηση. Αυτή η πληροφορία χρειάζεται μόνο εάν ο browser χρησιμοποιεί μια επίμονη (keep-alive) σύνδεση HTTP. Βλέπει το header *Connection* για τον καθορισμό όταν υποστηρίζει ο browser τις επίμονες συνδέσεις. Εάν θέλουμε το servlet μας για να εκμεταλλευθούμε τις επίμονες συνδέσεις όταν το υποστηρίζει ο browser, το servlet μας πρέπει να γράψει το έγγραφο μέσα σε ένα *ByteArrayOutput-Stream*, να ανατρέξει το μέγεθός του όταν γίνεται, να το τοποθετήσουμε αυτό μέσα σε ένα *Content-Length* πεδίο με την εντολή `response.setContentLength`, και κατόπιν να στείλουμε το περιεχόμενο μέσω της εντολής `byteArrayStream.writeTo(response.getOutputStream ())`.

Content-Location

Αυτό το header παρέχει μια εναλλακτική διεύθυνση για το ζητούμενο έγγραφο. Η *Content-Location* είναι ενημερωτικές απαντήσεις που περιλαμβάνουν αυτό το header και περιλαμβάνουν επίσης, το ζητούμενο έγγραφο, αντίθετα από την περίπτωση με το header *Location*. Αυτό το header είναι νέο στην έκδοση HTTP 1.1.

Content-MD5

Το *Content-MD5* header απάντησης παρέχει μια MD5 αφομοίωση για το επόμενο έγγραφο. Αυτή η αφομοίωση παρέχει έναν έλεγχο ακεραιότητας μηνυμάτων για τους clients που θέλουν να επιβεβαιώσουν ότι έλαβαν το πλήρες, κι αμετάβλητο έγγραφο. Αυτό το header είναι νέο στην έκδοση HTTP 1.1.

Content-Range

Αυτό το νέο header HTTP 1.1 στέλνεται με τις partial-document απαντήσεις και διευκρινίζει πόσο του συνολικού εγγράφου εστάλη. Παραδείγματος χάριν, μια τιμή "Bytes 500-999/2345" σημαίνει ότι η τρέχουσα απάντηση περιλαμβάνει τα Bytes 500 μέχρι 999 ενός εγγράφου που περιέχει 2345 Bytes στο σύνολο.

Content-Type

Το header *Content-Type* δίνει το τύπο MIME (Multipurpose Internet Mail Extension) του εγγράφου απάντησης. Η ρύθμιση αυτού του header είναι τόσο κοινή που υπάρχει μια ειδική μέθοδος στην *HttpServletResponse* για την *setContentType*. Οι τύποι MIME είναι της μορφής maintype/subtype για τους επίσημα καταχωρημένους τύπους, και της μορφής maintype/x-subtype για τους μη καταγεγραμμένους τύπους. Ο προεπιλεγμένος τύπος MIME για τα servlets είναι text/plain, αλλά τα servlets συνήθως ρητά διευκρινίζουν text/html. Μπορούν εντούτοις, να διευκρινίσουν άλλους τύπους αντ' αυτού.

Ο πίνακας 7.1 απαριθμεί μερικούς από τους πιο κοινούς τύπους MIME που χρησιμοποιούνται από τα servlets. Εντούτοις, οι νέοι τύποι MIME καταχωρούνται όλη την ώρα, κι έτσι ένας δυναμικός κατάλογος είναι μια καλύτερη θέση για να κοιτάξει. Οι επίσημα καταχωρημένοι τύποι παρατίθενται στην ηλεκτρονική διεύθυνση http://www.isi.edu/in-notes/iana/assignments/media-types/media-types. Για τους κοινούς μη καταγεγραμμένους τύπους, η ηλεκτρονική διεύθυνση http://www.ltsw.se/knbase/internet/mime.htp είναι μια καλή πηγή.

Έννοια τύπων

application/msword Microsoft Word document

application/octet-stream Unrecognized or binary data

application/pdf Acrobat (.pdf) file

application/postscript PostScript file

application/vnd.lotus-notes Lotus Notes file

application/vnd.ms-excel Excel spreadsheet

application/vnd.ms-powerpoint Powerpoint presentation

application/x-gzip Gzip archive

application/x-java-archive JAR file

application/x-java-serialized-object Serialized Java object

application/x-java-vm Java bytecode (.class) file

application/zip Zip archive

audio/basic Sound file in .au or .snd format

audio/x-aiff AIFF sound file

audio/x-wav Microsoft Windows sound file

audio/midi MIDI sound file

text/css HTML cascading style sheet

text/html HTML document

text/plain Plain text

image/gif GIF image

image/jpeg JPEG image

image/png PNG image

image/tiff TIFF image

image/x-xbitmap X Window bitmap image

video/mpeg MPEG video clip

video/quicktime QuickTime video clip

Date

Αυτό το header διευκρινίζει την τρέχουσα ημερομηνία με GMT format. Εάν θέλουμε να ρυθμίσουμε την ημερομηνία από ένα servlet, χρησιμοποιούμε τη μέθοδο *setDateHeader* για να την διευκρινίσουμε. Εκείνη η μέθοδος μας σώζει από το πρόβλημα της σχηματοποίησης της συμβολοσειράς της ημερομηνίας κατάλληλα, όπως θα ήταν απαραίτητο με την εντολή `response.setHeader("Date", "...")`. Εντούτοις, οι περισσότεροι Servers θέτουν αυτό το header αυτόματα, κι έτσι τα servlets δεν χρειάζονται συνήθως να την ρυθμίσουν.

ETag

Αυτό το νέο header HTTP 1.1 δίνει τα ονόματα στα επιστρεφόμενα έγγραφα έτσι ώστε να μπορούν να αναφερθούν από τον client αργότερα, όπως συμβαίνει και με το *If-Match* header αίτησης.

Expires

Αυτό το header ορίζει το χρόνο στον οποίο το περιεχόμενο πρέπει να θεωρηθεί ξεπερασμένο κι έτσι πλέον να εναποθηκευθεί. Ένα servlet ίσως χρησιμοποιήσει αυτό για ένα έγγραφο που αλλάζει σχετικά συχνά, για να αποτρέψει τον browser από την επίδειξη μιας πολυδιατηρημένης εναποθηκευμένης τιμής. Παραδείγματος χάριν, ο ακόλουθος κώδικας θα καθοδηγούσε τον browser για να μην εναποθηκεύσει το έγγραφο για περισσότερο από 10 λεπτά.

```
long currentTime = System.currentTimeMillis();

long tenMinutes = 10*60*1000; // In milliseconds

response.setDateHeader("Expires",

currentTime + tenMinutes);
```

Επίσης, πρέπει να παρατηρήσουμε την τιμή `max-age` του header *Cache-Control*.

Last-Modified

Αυτό το πολύ χρήσιμο header δείχνει πότε το έγγραφο τροποποιήθηκε για τελευταία φορά. Ο client μπορεί έπειτα να εναποθηκεύσει το έγγραφο και να παρέχει μια ημερομηνία από ένα *If-Modified-Since* αίτημα header στα πιο πρόσφατα αιτήματα. Αυτό το αίτημα αντιμετωπίζεται ως ένα υπό όρους *GET*, με το έγγραφο που επιστρέφεται μόνο εάν η *Last-Modified* ημερομηνία είναι πιο αργά από αυτή που διευκρινίζεται την για *If-Modified-Since*. Διαφορετικά, μια (Not Modified) γραμμή θέσης 304 επιστρέφεται, κι ο client χρησιμοποιεί το εναποθηκευμένο έγγραφο. Εάν θέσουμε αυτό το header ρητά, χρησιμοποιούμε τη μέθοδο *setDateHeader* για να σωθούμε από την ενόχληση της μορφοποίησης GMT των date strings. Εντούτοις, στις περισσότερες περιπτώσεις εφαρμόζουμε απλά τη μέθοδο *getLastModified* κι αφήνουμε την τυποποιημένη μέθοδο *service* να χειριστεί *If-Modified-Since* αιτήματα.

Location

Αυτό το header, που πρέπει να συμπεριληφθεί με όλες τις απαντήσεις που έχουν ένα status code στα 300s, δηλώνει τον browser για τη διεύθυνση εγγράφων. Ο browser επανασυνδέει αυτόματα σε αυτήν την θέση κι ανακτά το νέο έγγραφο. Αυτό το header συνήθως θέτει έμμεσα, μαζί με ένα status code 302, με τη μέθοδο *sendRedirect* της *HttpServletResponse*.

Pragma

Η παροχή αυτού του header με τιμή `no-cache` καθοδηγεί τους clients HTTP 1.0 για να μην εναποθηκεύσουν το έγγραφο. Εντούτοις, η υποστήριξη για αυτό το header ήταν ασυμβίβαστη με τους browsers HTTP 1.0. Στην έκδοση HTTP 1.1, "Cache-Control:no-cache" είναι μια πιο αξιόπιστη αντικατάσταση.

Refresh

Αυτό το header προσδιορίζει πόσο σύντομα (σε δευτερόλεπτα) ο browser πρέπει να ζητήσει μια ενημερωμένη σελίδα. Παραδείγματος χάριν, για να πούμε στον browser για να ζητήσουμε ένα νέο αντίγραφο σε 30 δευτερόλεπτα, θα διευκρινίζαμε μια τιμή του 30 με την εντολή `response.setIntHeader("Refresh", 30)`. Σημείωση ότι η *Refresh* δεν ορίζει τις συνεχείς αναπροσαρμογές που διευκρινίζει ακριβώς πότε η επόμενη αναπροσαρμογή πρέπει να είναι. Έτσι, πρέπει να συνεχίσουμε να παρέχουμε την *Refresh* σε όλες τις επόμενες απαντήσεις, κι η αποστολή ενός (No Content) status code 204 σταματά τον browser από το να «ξαναφορτώσει» περαιτέρω.

Αντί της κατοχής του browser «ξαναφορτώνουμε» ακριβώς την τρέχουσα σελίδα, μπορούμε να διευκρινίσουμε τη σελίδα που «φορτώνει». Κάνουμε αυτό με την παροχή μιας άνω τελείας κι ενός URL αφότου αναζωογονήσουμε το χρόνο. Παραδείγματος χάριν, για να πούμε στον browser για να μεταβούμε στην ηλεκτρονική διεύθυνση http://host/path μετά από 5 δευτερόλεπτα, θα γράφαμε την ακόλουθη εντολή `response.setHeader("Refresh", "5 URL=http://host/path")`. Αυτή η ρύθμιση είναι χρήσιμη για τις "splash screens", όπου μια εισαγωγική εικόνα ή ένα μήνυμα επιδεικνύεται εν συντομία προτού «φορτωθεί» η πραγματική σελίδα. Σημειώνουμε ότι αυτό το header ρυθμίζεται συνήθως από την `<META HTTP-EQUIV="Refresh" CONTENT="5; URL=http://host/path">` στο *HEAD* τμήμα της σελίδας HTML, παρά σαν ρητό header από τον Server. Εκείνη η χρήση ήρθε περίπου επειδή το αυτόματο ξαναφόρτωμα ή η αποστολή είναι κάτι που επιδιώκεται συχνά από τους συντάκτες των στατικών σελίδων HTML.

Για τα servlets εντούτοις, που θέτουν το header άμεσα είναι ευκολότερα και σαφέστερα. Αυτό το header δεν είναι επίσημα μέρος του HTTP 1.1 αλλά είναι μια επέκταση που υποστηρίζεται κι από τον Netscape κι από τον Internet Explorer.

Retry-After

Αυτό το header μπορεί να χρησιμοποιηθεί από κοινού με μια (Service Unavailable) απάντηση 503 για να πει στον client πόσο σύντομα μπορεί να επαναλάβει το αίτημά της.

Server

Αυτό το header προσδιορίζει τον Server δικτύου. Τα servlets δεν θέτουν συνήθως τον Web Server που το κάνει μόνος του.

Set-Cookie

Το header *Set-Cookie* διευκρινίζει ένα Cookie που συνδέεται με τη σελίδα. Κάθε Cookie απαιτεί ένα χωριστό header *Set-Cookie*. Τα servlets δεν πρέπει να χρησιμοποιήσουν την εντολή `response.setHeader("Set-Cookie", ...),` αλλά αντ' αυτού πρέπει να χρησιμοποιήσουν την ειδικής χρήσης μέθοδο *addCookie* της *HttpServletResponse*. Τεχνικά, το *Set-Cookie* δεν είναι μέρος της έκδοσης HTTP 1.1. Ήταν αρχικά μια επέκταση του Netscape αλλά τώρα είναι πολύ ευρέως υποστηριζόμενο, συμπεριλαμβάνων και του Netscape και του Internet Explorer.

Trailer

Αυτό το νέο και σπάνιο χρησιμοποιημένο header HTTP 1.1 προσδιορίζει τα πεδία των headers που είναι παρόντα στο ρυμουλκό ενός μηνύματος που στέλνεται με "chunked" η μεταφορά-κωδικοποίηση. Υπενθυμίζουμε ότι η ηλεκτρονική διεύθυνση http://www.rfc-editor.org/ διατηρεί έναν ενημερωμένο κατάλογο Sites αρχείων RFC.

Transfer-Encoding

Η παροχή αυτού του header με τιμή *chunked* δείχνει την " chunked" μεταφορά-κωδικοποίηση.

Upgrade

Αυτό το header χρησιμοποιείται όταν χρησιμοποιεί ο client πρώτος το header αιτήματος *Upgrade* για να ζητήσει από τον Server να μεταπηδήσει σε ένα από διάφορα πιθανά νέα πρωτόκολλα. Εάν ο Server συμφωνεί, στέλνει ένα (Switching Protocols) status code 101 και περιλαμβάνει ένα header απάντησης *Upgrade* με το συγκεκριμένο πρωτόκολλο που μεταπηδά. Αυτή η διαπραγμάτευση πρωτοκόλλου συνεχίζεται συνήθως από τον ίδιο τον Server, κι όχι από ένα servlet.

Vary

Αυτό το σπάνια χρησιμοποιημένο νέο header HTTP 1.1 λέει στον client ποια headers μπορούν να χρησιμοποιηθούν για να καθορίσουν εάν το έγγραφο απάντησης μπορεί να εναποθηκευθεί.

Via

Αυτό το header χρησιμοποιείται από gateways και proxies για να απαριθμήσει τα ενδιάμεσα Sites που το αίτημα πέρασε μέσω. Είναι νέο στην έκδοση HTTP 1.1.

Warning

Αυτό το νέο και σπάνια χρησιμοποιημένο catchall header μας αφήνει να προειδοποιήσουμε τους clients για την εναποθήκευση ή τα ικανοποιημένα λάθη μετασχηματισμού.

WWW-Authenticate

Αυτό το header συμπεριλαμβάνεται πάντα με ένα αναρμόδιο status code 401. Λέει στον browser ποιους εγκεκριμένους τύπους και πολιτικές ο client πρέπει να παρέχει στο header *Authorization* του. Συχνά, τα servlets αφήνουν password-protected ιστοσελίδες να αντιμετωπιστούν από τους ειδικευμένους μηχανισμούς του Server δικτύου (π.χ. .htaccess) παρά να τους χειριστεί άμεσα.