04-2013 | Winter | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java aktuell

Java ist nicht zu bremsen

Mobile statt Cloud
Java Enterprise Edition 7, Seite 8

Morphia, Spring Data & Co.
Java-Persistenz-Frameworks
für MongoDB, Seite 20

Starke Performance
Java ohne Schwankungen, Seite 50

Java und Oracle
Dynamische Reports innerhalb
der Datenbank, Seite 53

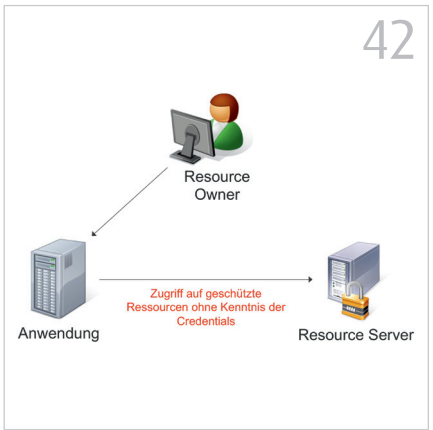
D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

iJUG Verbund

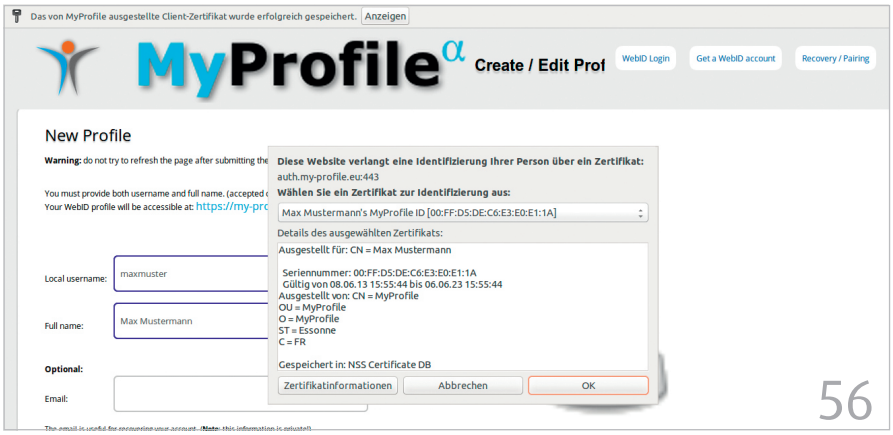
Sonderdruck

Java-Persistenz-Frameworks für MongoDB, Seite 20

Hibernate und Extra-Lazy Initialization, Seite 38



OAuth 2.0 – ein Standard, Seite 42



Universelles Ein-Klick-Log-in mit WebID, Seite 56

Version 1.2.x von Play einen wirklich runden Eindruck hinterließ. Es gab für alle Belange einer Web-Anwendung exakt eine Best-Practice-Implementierung. Diesen Rundumschlag lässt Play 2 leider vermissen.

Aus rein technologischer Sicht mag Play 2 mit der Umstellung auf Scala seine Berechtigung haben. Betrachtet man das Framework unter dem Gesichtspunkt der Nachhaltigkeit und strategischen Ausrichtung bei der Implementierung von Unternehmensanwendungen, so hat das Framework leider noch nicht die notwendige Reife. Zudem bleibt fraglich, ob es sich langfristig auszahlt, auf ein Framework in der Java-Welt jenseits des Java-EE-Standards zu setzen. Für kritische, nachhaltige Unternehmensanwendungen bietet beispielsweise Oracle ADF eine auf Standards basierende Alternative, in der neueste Technologien wie CSS3 und HTML5 integriert sind – mit der Essentials Edition sogar lizenzkostenfrei.

Trotz dieser kritischen Punkte bleibt das Play-2-Framework interessant und unter der Federführung der Typesafe Inc. eines mit möglicherweise rosiger Zukunft. Es bleibt spannend in der Welt der Web-Frameworks.

Weitere Informationen und Verweise

- „Webapps mit Play! entwickeln – Nichts leichter als das", JavaAktuell 2/2013. Es empfiehlt sich, diesen Artikel im Vorfeld gelesen zu haben. Dort finden sich alle notwendigen Verweise zum Play-Framework. Der Einstieg ist http://www.playframework.org

[1] Ehcache: http://ehcache.org
[2] SecureSocial: http://securesocial.ws
[3] CoffeeScript: http://coffeescript.org
[4] LESS: http://www.lesscss.de
[5] WebJars: http://www.webjars.org
[6] Heroku, Cloud: http://www.heroku.com
[7] Git, SCM: http://git-scm.com
[8] Yalp-Framework: https://github.com/yalp-framework/yalp
[9] Play 1 Fork Umfrage: https://www.surveymonkey.com/sr.aspx?sm=uRhTJLButShLupD_2bknn5YKhStcjfFZ1eHw4i7wg5X9w_3d

Andreas Koop
andreas.koop@enpit.de

Dipl.-Inf. Andreas Koop ist Geschäftsführer des Beratungsunternehmens enpit consulting OHG. Sein beruflicher Schwerpunkt liegt in der Beratung, dem Training und der Architektur für Oracle-ADF- und WebLogic-Server-Projekte. Daneben beschäftigt er sich mit der Entwicklung von mobilen Apps auf Basis von HTML5 sowie Oracle ADF Mobile und evaluiert aktuelle Web-Technologien und -Frameworks.

Flexible Suche mit Lucene

Florian Hopf, Freiberuflicher Software-Entwickler

Um große Textmengen zu durchsuchen, reichen herkömmliche Mechanismen oft nicht mehr aus. Eine indexbasierte Suche auf Basis der Open-Source-Bibliothek Apache Lucene bietet dann viele Vorteile.

Wenn große Mengen semistrukturierter Daten vorliegen, ergibt sich häufig das Problem, bestimmte Informationen aufzufinden. Während eine SQL-Datenbank gute Zugriffsmöglichkeiten auf strukturierte Daten bietet, kommt man beim Suchen in Texten schnell an die Grenzen. Unter anderem ergeben sich Probleme hinsichtlich der Skalierbarkeit und bei großen Datenmengen ist ein Durchsuchen der Daten zum Abfragezeitpunkt, beispielsweise über eine LIKE-Query, nicht mehr sinnvoll. Zu-

sätzlich fehlen Features, die bei modernen Such-Lösungen erwartet werden: Die Suche soll tolerant hinsichtlich der Schreibweise des Suchbegriffs sein und die Ergebnisse sollen sortiert nach Relevanzkriterien zurückgeliefert werden. Eine dafür besser geeignete Technologie ist die indexbasierte Suche, wie sie zum Beispiel von der Bibliothek Lucene und den darauf aufbauenden Suchservern Solr und ElasticSearch zur Verfügung gestellt wird.

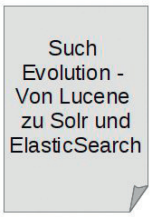
Indexbasierte Suche

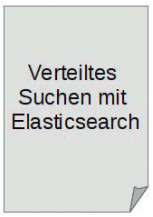
Das Grundprinzip einer indexbasierten Suche ist die Aufbereitung der zu durchsuchenden Inhalte und die Ablage in einem für die Suche optimierten Format in einer separaten Datenstruktur, dem invertierten Index. Der Index ähnelt im Kern einer Map, in der für einzelne Teile der Texte, die sogenannten „Terme" als Referenzen auf die Dokumente gespeichert werden, in denen sie auftauchen. Eine Suche besteht dann nur noch aus einem Lookup auf den Term und ist dadurch sehr schnell.

Der komplexe Vorgang, den Eingabetext zu parsen, wird der Suche vorgelagert und beim Indizieren, dem Aufbau der Datenstruktur, ausgeführt. Da der Index eine zusätzliche Ablage darstellt, ist es auch gleichgültig, woher die Daten ursprünglich stammen. Es ist möglich, Daten aus unterschiedlichen Quellen, beispielsweise aus einer Datenbank und aus Dateien im Filesystem, gemeinsam in einem Index abzulegen und zu durchsuchen.

Welche Einheiten eines Textes die Terme bilden, ist anwendungsabhängig, das Aufsplitten erfolgt beim sogenannten „Analyzing". Meist werden dabei mehrere Verarbeitungsschritte durchgeführt, wie wir im folgenden Beispiel sehen werden. Als Eingabe verwenden wir zwei Dokumente, die jeweils aus einem kurzen Vortragstitel bestehen: „Such-Evolution – von Lucene zu Solr und ElasticSearch" und „Verteiltes Suchen mit ElasticSearch". Jedes indizierte Dokument bekommt eine Id zugewiesen, die als Verweis verwendet werden kann.

In einem ersten Schritt werden die Wörter des Textes extrahiert sowie Leer- und Satzzeichen entfernt. Abbildung 1 zeigt den Inhalt des Index nach der Verarbeitung. Um unabhängig von der Groß- und Kleinschreibung zu bleiben, können alle Terme in Kleinbuchstaben umgewandelt werden. Danach sieht unser Index dann wie in Abbildung 2 aus.

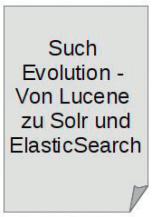


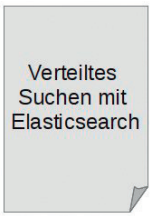


1. Tokenization

Term	Document Id
Such	1
Evolution	1
Von	1
Lucene	1
zu	1
Solr	1
und	1
ElasticSearch	1
Verteiltes	2
Suchen	2
mit	2
Elasticsearch	2

Abbildung 1: Index nach dem Aufsplitten der Sätze in Wörter (Tokenization)





1. Tokenization

2. Lowercasing

Term	Document Id
such	1
evolution	1
von	1
lucene	1
zu	1
solr	1
und	1
elasticsearch	1,2
verteiltes	2
suchen	2
mit	2

Abbildung 2: Index nach dem Umwandeln in Kleinbuchstaben (Lowercasing)

Wenn man sich den Prozess beim Auffinden eines Dokuments vor Augen führt, wird man feststellen, dass eine Verarbeitung nur beim Indizieren nicht ausreichend ist. Zwar ist es jetzt möglich, kleingeschriebene Suchbegriffe zu matchen, großgeschriebene Begriffe führen allerdings aufgrund des Lowercasing während der Indizierung zu keinen Treffern. Um die Modifikationen an den Termen beim Indizieren zu kompensieren, sind die gleichen Schritte auch für den Suchbegriff durchzuführen. Dadurch ist garantiert, dass gleiche Wörter beim Indizieren und Suchen auf denselben Term zurückgeführt werden.

Eine einfache Suche mit exakten Treffern ist damit bereits möglich, der Eindruck einer intelligenten Suche entsteht allerdings erst durch weitere Schritte. Der oft verwendete sprachabhängige Prozess „Stemming" führt Begriffe auf ihren Wortstamm zurück. So werden die Begriffe „Such" und „Suchen" beide in den Term „such" überführt. Dadurch sind Abweichungen wie Einzahl oder Mehrzahl nicht mehr relevant, da sie auf denselben Stamm zurückgeführt werden. Unser Index könnte dann nach Durchführung des Stemming wie in Abbildung 3 aussehen.

Es wurde bereits erwähnt, dass der Analyzing-Prozess anwendungsabhängig ist. Die meisten der Schritte sind verlustbehaftet, das heißt, es gehen potenziell Unterschiede zwischen Wörtern verloren. Groß- und Kleinschreibung kann für einzelne Anwendungen eine Rolle spielen, außerdem kann das Stemming, da es sich um einen algorithmischen Prozess handelt, Be-

griffe, die nicht nach den entsprechenden Regeln aufgebaut sind, falsch modifizieren.

Generell gilt es bei diesen Überlegungen abzuwägen, ob möglichst viele eventuell passende Ergebnisse zurückgeliefert werden oder ob die zurückgelieferten Ergebnisse auf jeden Fall passen sollen, mit der Gefahr, dass nicht alle passenden Dokumente gefunden werden. Im Information Retrieval beschreiben diese Aspekte die Begriffe „Recall" (möglichst viele gute Ergebnisse zurückliefern) und „Precision" (möglichst wenige schlechte Ergebnisse zurückliefern).

Neben den hier gezeigten Schritten sind noch viele weitere denkbar: Terme können automatisch mit Synonymen angereichert werden, häufig vorkommende Füllwörter können entfernt oder Sonderzeichen normalisiert werden. Wie der Analyzing-Prozess einer Suchanwendung aussieht, muss sorgfältig mit den eigenen Anforderungen abgeglichen werden, da er die Qualität der Suchergebnisse maßgeblich beeinflusst.

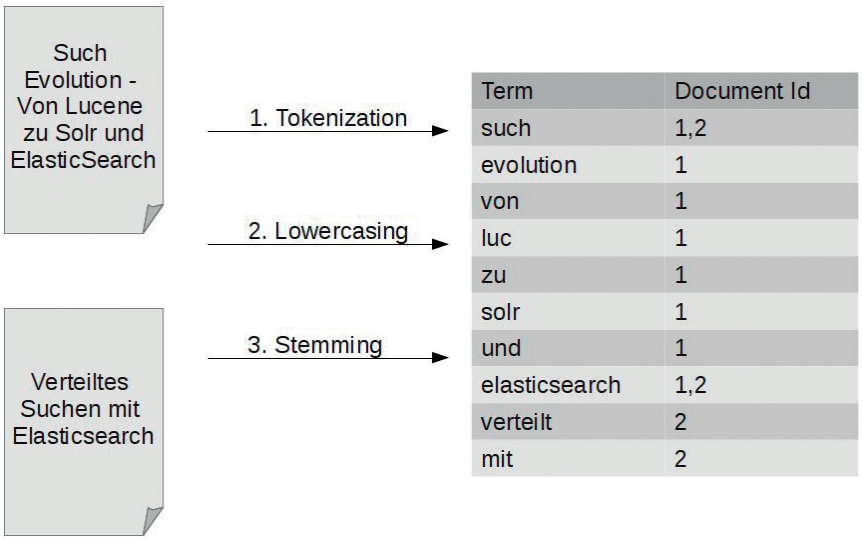


Abbildung 3: Index nach Durchführen des Stemming

Lucene

In der Java-Welt ist die Bibliothek Lucene für Such-Anwendungen seit Jahren fest etabliert. Sie bietet die Implementierung eines invertierten Index mit unterstützenden Datenstrukturen und Möglichkeiten, diese Strukturen zu schreiben, zu lesen und zu durchsuchen. Im Kern und als zusätzliche Module sind zahlreiche für das Analyzing notwendige Klassen verfügbar, insbesondere „Tokenizer", die Texte aufsplitten, „TokenFilter", die Änderungen wie das Lowercasing und Stemming durchführen, und „Analyzer", die aus Tokenizern und TokenFiltern bestehen und für häufige Einsatzzwecke schon fertig zur Verfügung stehen. Für die Suche können unterschiedliche Queries verwendet werden, die entweder in Textform verarbeitet oder programmatisch konstruiert werden können. Häufige Anwendungsfälle wie Keyword-Suche (Term-Query), Ähnlichkeitssuche (Fuzzy-Query) oder Phrase-Queries werden direkt unterstützt.

Um einen ersten Eindruck vom Indizieren und Suchen mit Lucene zu bekommen, betrachten wir die Verarbeitung von Vortragsankündigungen. Diese bestehen teils aus strukturierten Daten wie Sprecher und Datum, aber auch aus textuellen Daten, die durchsucht werden sollen, wie dem Titel. Ein ausführlicheres Beispiel in Form einer Webanwendung findet sich auch auf GitHub unter https://github.com/fhopf/lucene-solr-talk.

Die Ein- und Ausgabe erfolgt in Lucene jeweils durch Instanzen der Klasse „Document". Ein Document ist eine Sammlung von Feldern, die jeweils aus einem Namen und einem Wert bestehen. Welche Felder verfügbar sind, wird von der Anwendung bestimmt; Lucene ist intern schemafrei und daher vergleichbar mit Document-Stores aus der NoSQL-Welt. Zur Bestimmung, wie ein Feld verarbeitet wird, können weitere Eigenschaften konfiguriert werden:

- *Indexed*
  Bestimmt, ob ein Feld in den Index geschrieben wird und ob es vom Analyzer verarbeitet werden soll
- *Stored*
  Bestimmt, ob der Originalinhalt zusätzlich gespeichert werden soll, um ihn bei Anzeige der Suchergebnisse verfügbar zu haben

Seit Lucene 4 müssen diese Eigenschaften nicht mehr direkt gesetzt werden; für häufige Anwendungsfälle stehen unterschiedliche Field-Implementierungen zur Verfügung, beispielsweise „TextField", wenn der Inhalt indiziert und „analyzed" sein soll, oder „StringField", wenn zwar die Indizierung, nicht aber das Analyzing durchgeführt werden soll. Für unser Beispiel entscheiden wir, dass alle Felder bei der Darstellung der Suchergebnisse zur Verfügung stehen sollen. Titel und Datum werden in den Index geschrieben, allerdings soll lediglich der Titel vom Analyzer verarbeitet werden (siehe Listing 1).

Zum Indizieren benötigen wir einen „IndexWriter", der Daten in den Index schreiben kann. Der Speicherort des Index ist durch die Klasse „Directory" abstrahiert, von der es unterschiedliche Implementierungen gibt, die die Daten beispielsweise optimiert für unterschiedliche Betriebssysteme auf der Festplatte ablegen oder die Struktur direkt im Speicher halten. Der „IndexWriter" wird unter anderem mit dem „Analyzer", der für das Aufsplitten der Inhalte verantwortlich ist, und dem „Directory" konfiguriert. Anschließend können eines oder mehrere Dokumente über „addDocument" zum Index hinzugefügt werden. In diesem Moment stehen die Dokumente allerdings noch nicht zur Suche zur Verfügung. Lucene speichert den Index in Segmenten, die jeweils einen Teil des Index enthalten. Erst nach einem „commit" kann dieses Segment durchsucht werden (siehe Listing 2).

Für den lesenden Zugriff auf den Index steht der „IndexSearcher" zur Verfügung. Dieser bietet Methoden, um für übergebene Query-Instanzen die zugehörigen Ergebnisse aus dem Index zu lesen. Eine Query-Instanz kann entweder über einen „QueryParser" aus einem String erzeugt oder programmatisch zusammengebaut werden. Die vom Standard-QueryParser an-

gebotene Lucene-Syntax bietet viele Möglichkeiten, etwa um Teilbegriffe als optional zu markieren, Begriffe auszuschließen oder Phrase- und Fuzzy-Queries durchzuführen. Entscheidet man sich dafür, die Query programmatisch selbst aufzubauen, muss man beachten, dass man das Analyzing selbst durchzuführen hat, da dieses ansonsten im „QueryParser" stattfindet (siehe Listing 3).

Zur Ausgabe der Suchergebnisse werden ebenfalls wieder Instanzen von „Document" verwendet, die über die „TopDocs" identifiziert werden können und über den „IndexSearcher" auslesbar sind. Im Gegensatz zu den beim Indizieren übergebenen Dokumenten sind nun allerdings nur noch die als „stored" markierten Felder verfügbar (siehe Listing 4).

Die Reihenfolge der zurückgelieferten Dokumente ist durch den Relevanzmechanismus in Lucene bestimmt, standardmäßig kommt hier der TF/IDF-Algorithmus zum Einsatz. Dieser basiert auf der Annahme, dass Dokumente, die einen Begriff häufiger enthalten, eine höhere Relevanz für den User haben. Gleichzeitig werden Begriffe, die im Gesamt-Index sehr häufig vorkommen, als weniger wichtig angesehen. Die Relevanz ist durch Mechanismen wie „Boosting" zur Index- oder Suchzeit auch beeinflussbar. Dabei werden bestimmte Dokumente, Felder oder Terme mit einem Boost-Faktor versehen, der den Scoring-Wert eines Dokuments und damit die Sortierung beeinflusst. Zusätzlich ist es auch möglich, den verwendeten Algorithmus auszutauschen, entweder durch eine der weiteren mitgelieferten Implementierungen oder durch einen eigenen Mechanismus.

Die Sortierung nach Relevanz ist einer der großen Vorteile bei der Implementierung einer Anwendung mit Lucene. Es ist bei Bedarf jedoch auch möglich, die Ergebnisse nach einem bestimmten Feld zu sortieren, beispielsweise bei Indizierung des Datumsfeldes die Anzeige sortiert nach der Aktualität.

```
Document doc = new Document();
doc.add(new TextField("title", "Such Evolution", Field.Store.YES));
doc.add(new StoredField("speaker", "Florian Hopf"));
doc.add(new StringField("date", "20130704", Field.Store.YES));
```

Listing 1

```
Directory directory = FSDirectory.open(new File("/tmp/talk-index"));
IndexWriterConfig config = new IndexWriterConfig(Version.LUCENE_43, new
        GermanAnalyzer(Version.LUCENE_43));
try (IndexWriter writer = new IndexWriter(directory, config)) {
        writer.addDocument(doc);
        writer.commit();
}
```

Listing 2

```
IndexReader reader = DirectoryReader.open(directory);
IndexSearcher searcher = new IndexSearcher(reader);
QueryParser parser = new QueryParser(Version.LUCENE_43, "title", new
        GermanAnalyzer(Version.LUCENE_43));
Query query = parser.parse("suchen");
TopDocs topDocs = searcher.search(query, 5);
assertEquals(1, topDocs.totalHits);
```

Listing 3

```
ScoreDoc scoreDoc = topDocs.scoreDocs[0];
Document result = searcher.doc(scoreDoc.doc);
assertEquals("Florian Hopf", result.get("speaker"));
```

Listing 4

Ausblick

Neben den hier angesprochenen Features von Lucene sind noch sehr viele weitere Funktionen verfügbar. Sehr populär ist beispielsweise das in der nächsten Ausgabe beschriebene „Faceting", mit dem Suchergebnisse kategorisiert werden können. „Highlighting" ermöglicht das Hervorheben der Fundstelle im Text, um den Kontext schnell zu erfassen. „Result Grouping" dient dem Zusammenfassen von Suchergebnissen anhand bestimmter Kriterien, über „Suggester" können Vorschläge zu sinnvollen Suchen gegeben werden.

Mit Lucene kann mit relativ wenig Code das Grundgerüst einer Such-Anwendung implementiert werden. Die wenigen Zeilen, die wir gesehen haben, ermöglichen uns bereits das Indizieren von Dokumenten und die flexible Suche darin. Geht es jedoch darum, eine Anwendung in Produktion zu nehmen, sind noch weitere Faktoren zu beachten, die eine höhere Komplexität bringen können. Beispielsweise sollten wir vermeiden, den „IndexReader" bei jeder Anfrage zu erzeugen, da es sich dabei um eine sehr teure Operation handelt.

Zur Implementierung fortgeschrittener Features ist teilweise ein tieferes Verständnis der Interna von Lucene notwendig. Schließlich kann es noch passieren, dass unsere Datenmenge ansteigt und wir die Dokumente auf mehrere Indizes zu verteilen haben, oder wir müssen uns um Ausfallsicherheit Gedanken machen. Diese Anforderungen sind mit den auf Lucene aufbauenden Suchservern Apache Solr und ElasticSearch oftmals einfacher umzusetzen. In der nächsten Ausgabe werden wir deshalb einen Blick auf diese beiden Systeme werfen und sehen, dass sie uns einiges an Arbeit abnehmen können.

Links

- http://lucene.apache.org
- https://github.com/fhopf/lucene-solr-talk

Florian Hopf
mail@florian-hopf.de

Florian Hopf arbeitet als freiberuflicher Software-Entwickler mit den Schwerpunkten „Content Management" und „Suchlösungen" in Karlsruhe. Er setzt Lucene und Solr seit Jahren in unterschiedlichen Projekten ein und ist einer der Organisatoren der Java User Group Karlsruhe.

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG *News* und vier Ausgaben im Jahr Business News zusammen für 70 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN
0700 11 36 24 39

ODER BESTELLEN SIE ONLINE
go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr
☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:
Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.