Prof. Dr. Arnd Poetzsch-Heffter
Dipl.-Inf. Jan Schäfer
Dipl.-Inf. Jean-Marie Gaillourdet
Dipl.-Inform. Markus Reitz

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Aufgabenblatt 2: Praktikum Komponententechnik (WS 06/07)

Ausgabe: 31. Oktober 2006

Aufgabe 1 Entwicklung eines Webservers

In den nächsten zwei Wochen geht es nun darum einen funktionsfähigen Webserver zu implementieren. Die erste Version soll nur die Basisfunktionalität eines Webservers besitzen, d.h. er soll auf GET-Requests eines Browsers eine HTML-Datei von der Festplatte lesen und entsprechend dem Browser zurücksenden. Die genaue Definition des GET-Requests finden Sie im HTTP/1.0 RFC 1945 (`http://www.w3.org/Protocols/rfc1945/rfc1945`).

Detaillierte Anforderung

a) Der Server soll mit einem Shell-Skript `<script>` `<port>` gestartet werden können und auf dem übergebenen Port `<port>` auf Anfragen von Clients warten.

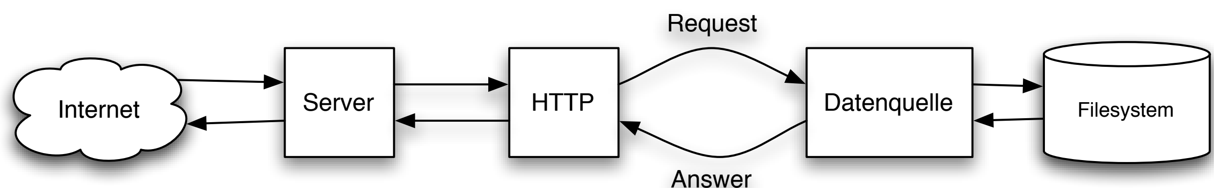
b) Der Server soll HTTP-GET-Befehle verstehen können, auf alle nicht implementierten HTTP-Features soll der Server mit der HTTP-Fehlermeldung „501 Not Implemented" antworten. Falls die HTTP-Request-Nachricht fehlerhaft ist, soll mit „400 Bad Request" geantwortet werden.

c) Der Server soll, abhängig von den Fähigkeiten des Clients, seine Antworten mit gzip packen.

d) Als Datenquellen soll der Server Dateien von der Festplatte lesen. Es soll dabei möglich sein in einer Konfigurationsdatei festzulegen in welchen Verzeichnissen der Server nach Dateien suchen soll. Dabei sollen auch entsprechende Fehlermeldungen wie „403 Forbidden" und „404 Not Found" zurückgegeben werden, falls bei einer Datei keine Rechte bestehen bzw. wenn sie nicht gefunden wurde.

e) Es soll möglich sein Umleitungen (redirects) zu konfigurieren. D.h. der Server soll bei bestimmten URLs mit einer Redirect-Nachricht (3xx) antworten.

f) Der Server soll verschiedene Debug-Stufen haben. D.h. er soll konfigurierbar Debugging-Information, entweder auf der Standardausgabe oder in bestimmte Dateien schreiben. Die Genauigkeit der Debug-Informationen soll ebenfalls konfigurierbar sein.



Implementierungsvorgaben

Für die Implementierung des Servers geben wir ein paar eingeschränkte Vorgaben, die eingehalten werden müssen. Es sollen alle Tools eingesetzt werden, die Sie auf dem letzten Übungsblatt verwendet haben. D.h. Java 5, Ant, JavaDoc und JUnit.

Build-Infrastruktur

Sie sollen eine komplette Build-Infrastruktur mit Ant aufbauen. Es soll möglich sein mit Ant den Quellcode zu übersetzen, JavaDoc-Dokumentation zu erzeugen und JUnit-Tests auszuführen. Es soll dabei folgende Top-Level-Verzeichnisse geben:

- `src` - Enthält die Quellen des Servers.
- `bin` - Enthält den Bytecode des Servers.
- `doc` - Enthält die Dokumentation, sowohl JavaDoc, als auch sonstige Dokumentation.
- `test` - Enthält Quellen und Bytecode zum Testen des Servers.
- `lib` - Enthält Bibliotheken die zum Übersetzen, Testen und/oder Aufrufen des Servers benötigt werden. Dies stellt sicher, dass alle Entwickler die gleichen Versionen verwenden.

Interface-Oriented Programming

Es soll bei der Implementierung darauf geachtet werden, dass verschiedene Komponenten des Webservers nur über Java-Interfaces miteinander kommunizieren. D.h. jede Komponente stellt eine Menge von Java-Interfaces zur Verfügung, die andere Komponenten verwenden können. Die Implementierung der Interfaces sollte aber für andere Komponenten nicht sichtbar sein. Um dies sicherzustellen, soll folgendes Implementierungsschema verwendet werden. Jede Komponente muss sein eigenes Java-Paket haben. Das Paket selbst soll nur die Interface-Dateien, die die Schnittstelle der Komponente darstellen enthalten. Die Implementierung der Komponente soll in ein Unterpaket `impl`. Idealerweise sollten alle Klassen die in dem `impl`-Paket sind als package-private (default) deklariert sein, sodass sie unsichtbar sind für andere Komponenten.

Unit-Testing

Alle Teile des Webservers sollen, soweit möglich, durch Unit-Tests getestet werden. Die Tests sollten schon während der Entwicklung geschrieben werden, nicht erst hinterher.

Dokumentation

Alle öffentlichen Schnittstellen müssen mit JavaDoc dokumentiert werden. D.h. es muss angegeben werden was die einzelnen Parameter bedeuten und welche Werte sie annehmen können, welche Werte eine Methode zurückgeben kann, welche Seiteneffekte eine Methode hat und welche Exception unter welchen Umständen geworfen werden. Private Methoden und Felder von Implementierungs-Klassen sollten auch dokumentiert werden, deren Dokumentation kann aber knapper ausfallen.

Neben der Schnittstellenbeschreibung soll außerdem noch ein Dokument erstellt werden, dass die grobe Architektur des Webservers beschreibt. D.h. hier sollen die einzelnen Komponenten erklärt werden und wie diese untereinander in Beziehung stehen.

Best-Practice-Hinweise

Hier geben wir kurz Hinweise die sich als gut bei der Softwareentwicklung herausgestellt haben. Dies muss nicht eingehalten werden, sollte aber.

Java-Compiler

Man sollte den Java-Compiler so einstellen, dass er Debug-Informationen in den Bytecode generiert, sodass bei Exceptions die Fehlerursache leichter gefunden werden kann.

Versionskontrolle

Es ist sehr wichtig regelmäßig Änderungen einzuchecken, damit alle Entwickler auf einem möglichst aktuellen Stand sind. Entsprechend sollte man auch möglichst häufig ein Update vornehmen, um die Änderungen der anderen Entwickler mitzubekommen. Generell sollte man vor jedem Commit ein Update machen, um Konflikten vorzubeugen.

Eine wichtige Regel sollte beim Einchecken eingehalten werden: Nur Code einchecken der auch kompiliert! Auf keinen Fall solltet ihr Code einchecken der nicht kompilierbar ist, da sonst andere Entwickler nicht mehr vernünftig weiterarbeiten können. Es kann bedenkenlos Code eingecheckt werden, der noch nicht richtig funktioniert oder noch nicht fertig implementiert ist, solange er kompiliert. Dies bedeutet auch, dass nicht alle Unit-Tests durchlaufen müssen bevor man Code eincheckt.

Geben Sie ihren Änderungen eine sinnvolle kurze Beschreibung. Beschreiben Sie die Änderung so, dass man sie später im Log schnell wiederfinden kann und dass andere Entwickler ungefähr wissen was gemacht wurde. Eventuell sollte man auch kurz erklären warum eine Änderung vorgenommen wurde.

Legen Sie keine generierten Dateien ins Repositry. Z.B. sollte die API-Dokumentation die mit JavaDoc generiert wird nicht eingecheckt werden.

Unit-Testing

Es ist sehr hilfreich während dem Entwickeln direkt zu testen. Dies bedeutet, dass bei jeder neuen Klasser oder Methode, die Ihr schreibt, direkt einen Unit-Test zu schreiben, der diese Methode testet. Dies hat zwei Vorteile: Erstens findet ihr direkt Fehler in eurer Implementierung und Zweitens schreibt ihr euren Code direkt so, dass er auch testbar ist. Dadurch verhindert ihr, dass ihr später den Code umstrukturieren müsst, damit ihr ihn testen könnt.

API-Dokumentation

Häufig ist es sinnvoll, noch vor der Implementierung einer Methode die JavaDoc-Dokumentation dazu zu schreiben. Dadurch macht man sich direkt Gedanken darüber, welche Werte die Methode annehmen kann, welche Rückgabewerte sie hat und wann Exceptions geworfen werden sollen. Dies ist besonders wichtig, bei öffentlichen Schnittstellen-Methoden, die von anderen Komponenten verwendet werden sollen.

Refactoring

Under dem Begriff *Refactoring* bezeichnet man die Restruktierung des Codes unter Beibehaltung der ursprünglichen Funktionalität. Es ist fast immer der Fall, dass man im Laufe der Entwicklung feststellt, dass bestimmte Design-Entscheidungen die am Anfang gefällt wurden, nicht optimal waren. Häufig implementiert man auch erstmal drauf los und hat später einen relativ unübersichtlichen Code produziert. Deswegen ist es wichtig immer mal wieder „Aufzuräumen". Wenn Sie merken, dass Ihr Code schlecht „riecht" sollten Sie anfangen ihn umzustruktieren. D.h. z.B. große Code-Blöcke in kleinere Methoden zerlegen, Methoden, Felder und/oder Klassen umzubenennen, zu verschieben, oder sogar zu löschen. Eclipse hat hier eine Vielzahl an vordefinierten Refactoring-Tools, die semi-automatisch den Code umstrukturieren können.