Federinterne Funktionsweise

Lass uns in die Welt von Spring und seine internen Arbeitsweisen eintauchen. Spring ist ein robustes Framework zum Bauen von Java-Anwendungen, und ich werde jede Frage Schritt für Schritt mit klaren Erklärungen beantworten.

Wie funktioniert Spring?

Spring arbeitet nach dem Prinzip der **Inversion of Control (IoC)**. Das bedeutet, dass anstatt Ihr Anwendungscode die Erstellung und Verwaltung von Objekten steuert, übernimmt Spring diese Aufgabe. Dies geschieht durch eine Komponente namens **IoC-Container**. Der IoC-Container ist verantwortlich für:

- **Instanziierung** von Objekten (in Spring als Beans bezeichnet).
- **Konfiguration** dieser basierend auf Ihren Spezifikationen.
- **Zusammenstellung** durch Verwaltung ihrer Abhängigkeiten.

Dieser Ansatz reduziert die enge Kopplung in Ihrem Code, macht ihn modularer und leichter zu warten.

Wie verwaltet Spring Beans?

In Spring sind **Beans** die Objekte, die vom IoC-Container verwaltet werden. Hier ist, wie Spring sie handelt:

1. **Definition**: Sie definieren Beans entweder in:
   - **XML-Konfigurationsdateien**.
   - **Java-basierte Konfiguration** mit Annotationen wie `@Bean`, `@Component`, `@Service`, usw.
2. **Erstellung**: Wenn die Anwendung startet, liest der IoC-Container diese Definitionen und erstellt die Beans.
3. **Dependency Injection (DI)**: Spring injiziert automatisch Abhängigkeiten (andere Beans) in einen Bean, wo es benötigt wird, mit:
   - **Konstruktor-Injektion**.
   - **Setter-Injektion**.
   - **Feld-Injektion** (über `@Autowired`).

Der Container verwaltet den gesamten Lebenszyklus dieser Beans –von der Erstellung bis zur Zerstörung – und stellt sicher, dass sie verfügbar sind, wenn sie benötigt werden.

Unterschied zwischen einem Service und einem Controller

Im Kontext von **Spring MVC** (Spring's Web-Framework) dienen diese beiden Komponenten unterschiedlichen Zwecken:

- **Controller**:
  - Verarbeitet **HTTP-Anfragen** von Benutzern.
  - Verarbeitet Eingaben, ruft Geschäftslogik auf und entscheidet, welche **Ansicht** (z.B. eine Webseite) zurückgegeben wird.
  - Mit `@Controller` oder `@RestController` annotiert.
  - Lebt in der **Web-Schicht**.
- **Service**:
  - Kapselt die **Geschäftslogik** der Anwendung.
  - Führt Aufgaben wie Berechnungen, Datenverarbeitung oder Interaktionen mit Datenbanken durch.
  - Mit `@Service` annotiert.
  - Lebt in der **Geschäftsschicht**.

Beispiel: - Ein Controller könnte eine Anfrage erhalten, um das Profil eines Benutzers anzuzeigen und einen Service aufrufen, um die Benutzerdaten abzurufen. - Der Service holt die Daten aus einer Datenbank und gibt sie an den Controller zurück, der sie dann an die Ansicht sendet.

Kurz gesagt: **Controller verwalten Web-Interaktionen**, während **Services die Kernfunktionalität handhaben**.

Was bietet Spring?

Spring ist ein umfassendes Framework, das eine breite Palette von Werkzeugen für Unternehmensanwendungen bietet. Wichtige Funktionen umfassen:

- **Dependency Injection**: Erleichtert die Verwaltung von Objektabhängigkeiten.
- **Aspect-Oriented Programming (AOP)**: Fügt Querkonzernsprobleme wie Protokollierung oder Sicherheit hinzu.
- **Transaktionsverwaltung**: Stellt die Datenkonsistenz über Operationen sicher.
- **Spring MVC**: Baut robuste Webanwendungen.
- **Spring Boot**: Erleichtert die Einrichtung mit vorkonfigurierten Standards und eingebetteten Servern.
- **Spring Data**: Erleichtert den Datenbankzugriff.
- **Sicherheit**: Bietet Authentifizierungs- und Autorisierungsinstrumente.

Das modulare Design von Spring ermöglicht es Ihnen, nur die Funktionen zu wählen, die Sie benötigen.

Wie sucht Spring nach Objekten oder Beans?

Wenn eine Spring-Anwendung startet:

1. Initialisiert der **IoC-Container**.

2. Er scannt nach **Bean-Definitionen** in:
    - XML-Dateien.
    - Annotierten Klassen (z.B. `@Component`, `@Bean`).
3. Der Container erstellt die Beans und speichert sie in einer **Map**, die nach Name oder Typ indiziert ist.
4. Wenn ein Bean benötigt wird (z.B. für Injektion oder direkte Abfrage), stellt der Container ihn bereit mit:
    - **Nach-Namen-Suche**.
    - **Nach-Typ-Suche** (z.B. über `@Autowired`).

Dieser Prozess stellt sicher, dass Beans während der gesamten Anwendung verfügbar sind.

Wie verwendet man Tomcat oder Netty als Server?

Spring Boot, eine Erweiterung von Spring, erleichtert die Server-Einrichtung, indem es Server direkt in Ihre Anwendung einbettet. Hier ist, wie es mit **Tomcat** und **Netty** funktioniert:

- **Tomcat** (Servlet-basiert):
    - Standardserver in der `spring-boot-starter-web`-Abhängigkeit von Spring Boot.
    - Fügen Sie dies zu Ihrer `pom.xml` hinzu:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

    - Spring Boot integriert Tomcat, und Ihre App läuft als eigenständige ausführbare JAR-Datei.
- **Netty** (Nicht-Servlet-basiert):
    - Wird für reaktive Webanwendungen (z.B. mit Spring WebFlux) verwendet.
    - Ersetzen Sie Tomcat, indem Sie es ausschließen und Netty hinzufügen:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

    - Spring Boot integriert Netty anstelle von Tomcat, unterstützt asynchronen, nicht blockierenden I/O.

Sie konfigurieren den Server über `application.properties` (z.B. Port setzen mit `server.port=8080`). Keine externe Serverbereitstellung ist erforderlich –Spring Boot erledigt alles.

Zusammenfassung

- **Spring-Kern**: Verwendet einen IoC-Container zur Objektverwaltung.
- **Bean-Verwaltung**: Definiert über XML oder Annotationen, mit automatisch injizierten Abhängigkeiten.
- **Service vs. Controller**: Services handhaben Geschäftslogik; Controller verwalten Webanfragen.
- **Funktionen**: Bietet DI, AOP, Web-Unterstützung und mehr.
- **Bean-Suche**: Container findet und stellt Beans aus Konfigurationen bereit.
- **Server**: Spring Boot integriert Tomcat oder Netty mit einfachen Abhängigkeitsänderungen.