ГЛАВА 15

Мониторинг приложений Spring

Типичное JEE-приложение содержит несколько уровней и компонентов, таких как уровень презентаций, уровень обслуживания, уровень постоянства и источник данных серверной части. На протяжении стадии разработки либо после развертывания приложения в среде контроля качества (quality assurance — QA) или в производственной среде нам нужно удостовериться, что приложение находится в работоспособном состоянии и не содержит никаких потенциальных проблем либо узких мест.

В Java-приложении существует много аспектов, которые могут вызвать проблемы с производительностью или перегрузкой серверных ресурсов (наподобие центрального процессора, памяти и подсистемы ввода-вывода). В качестве примеров следует упомянуть неэффективный Java-код, утечку памяти (когда в Java-коде выделяются новые объекты без освобождения ссылок на них, что препятствует очистке памяти JVM-машиной во время процесса сборки мусора), параметры JVM, параметры пула потоков, конфигурация источников данных (например, количество разрешенных одновременных подключений к базе данных), настройка базы данных и наличие длительно выполняющихся SQL-запросов.

Следовательно, нам необходимо понимать поведение приложения во время выполнения и идентифицировать любые потенциально узкие места либо имеющиеся проблемы. В мире Java доступно множество инструментов, которые помогают проводить мониторинг поведения JEE-приложений во время выполнения. Большинство из них построены на основе технологии JMX (Java Management Extensions — управляющие расширения Java). В этой главе мы представим несколько общих приемов мониторинга JEE-приложений, основанных на Spring. В частности, в главе будут рассматриваться следующие темы.

- **Поддержка JMX в Spring.** Мы обсудим комплексную поддержку JMX в Spring и покажем, как сделать бины Spring открытыми для мониторинга с помощью инструментов JMX. В качестве инструмента мониторинга приложений мы будем использовать VisualVM (`http://visualvm.java.net/index.html`).
- **Мониторинг статистики Hibernate.** Многие макеты, в том числе Hibernate, предоставляют поддерживающие классы и инфраструктуру для получения доступа к рабочему состоянию и метрикам производительности с применением JMX. Мы покажем, как включить мониторинг посредством JMX для часто используемых компонентов JEE-приложений Spring.

Имейте в виду, что настоящая глава не предназначена служить введением в JMX, поэтому предполагается наличие у вас базового понимания этой технологии. За детальными сведениями по данной теме обращайтесь к онлайновому ресурсу Oracle по адресу `www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html`.

Поддержка JMX в Spring

В технологии JMX классы, которые открыты для мониторинга и управления, называются *управляемыми бинами* (managed bean), или *MBean-компонентами*. Платформа Spring Framework поддерживает несколько механизмов для открытия MBean-компонентов. Основное внимание в этой главе будет уделено открытию бинов Spring (разработанных в виде простых объектов POJO) как MBean-компонентов для проведения мониторинга JMX.

В последующих разделах мы обсудим процедуру открытия бина, содержащего связанную с приложением статистику, в качестве MBean-компонента для мониторинга JMX. Темы включают реализацию бина Spring, открытие бина Spring как MBean-компонента в `ApplicationContext` и применение инструмента VisualVM для мониторинга MBean-компонента.

Экспортирование бина Spring в JMX

В качестве примера мы будем использовать веб-службу REST, построенную в главе 12. Еще раз просмотрите код примера приложения в указанной главе или обратитесь непосредственно к коду примеров для этой книги, где доступен исходный код, от которого мы и будем отталкиваться. За счет добавления JMX мы хотим открыть количество контактов в базе данных в целях мониторинга JMX. Итак, давайте реализуем интерфейс и класс, как показано в листингах 15.1 и 15.2.

Листинг 15.1. Интерфейс `AppStatistics`

```
package com.apress.prospring4.ch15;
public interface AppStatistics {
    int getTotalContactCount();
}
```

Листинг 15.2. Класс `AppStatisticsImpl`

```
package com.apress.prospring4.ch15;

import org.springframework.beans.factory.annotation.Autowired;

public class AppStatisticsImpl implements AppStatistics {
    @Autowired
    private ContactService contactService;

    @Override
    public int getTotalContactCount() {
        return contactService.findAll().size();
    }
}
```

В этом примере определен метод для извлечения общего количества записей контактов в базе данных. Чтобы открыть бин Spring для JMX, понадобится добавить конфигурацию в `ApplicationContext`. В листинге 15.3 показан код, необходимый для открытия бина Spring для JMX в конфигурационном файле (`rest-context.xml`).

Листинг 15.3. Открытие бина Spring для JMX

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:mvc="http://www.springframework.org/schema/mvc"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc.xsd">

   <mvc:annotation-driven>
      <mvc:message-converters>
         <bean class="org.springframework.http.converter.json.
MappingJackson2HttpMessageConverter"/>
         <bean class="org.springframework.http.converter.xml.
MarshallingHttpMessageConverter">
            <property name="marshaller" ref="castorMarshaller"/>
            <property name="unmarshaller" ref="castorMarshaller"/>
         </bean>
      </mvc:message-converters>
   </mvc:annotation-driven>

   <context:component-scan base-package="com.apress.prospring4.ch15"/>

   <bean id="castorMarshaller"
         class="org.springframework.oxm.castor.CastorMarshaller">
      <property name="mappingLocation"
                value="classpath:META-INF/spring/oxm-mapping.xml"/>
   </bean>

   <bean id="appStatisticsBean"
         class="com.apress.prospring4.ch15.AppStatisticsImpl"/>
   <bean id="jmxExporter"
         class="org.springframework.jmx.export.MBeanExporter">
      <property name="beans">
         <map>
            <entry key="bean:name=ProSpring4ContactApp"
                   value-ref="appStatisticsBean"/>
         </map>
      </property>
   </bean>
</beans>
```

Во-первых, в коде объявляется бин для POJO со статистикой (`AppStatisticsImpl`), который мы хотим открыть. Во-вторых, объявляется бин `jmxExporter` с классом реализации `MBeanExporter`.

Класс `MBeanExporter` — это ключевой класс в рамках поддержки JMX платформой Spring Framework. Он отвечает за регистрацию бинов Spring на MBean-сервере JMX (сервер, реализующий JDK-интерфейс `javax.management.MBeanServer`, который существует в большинстве распространенных веб- и JEE-контейнеров, таких как Tomcat и WebSphere). При открытии бина Spring как MBean-компонента среда Spring попытается найти на сервере выполняющийся экземпляр `MBeanServer` и с его помощью зарегистрировать этот MBean-компонент. Например, для сервера Tomcat экземпляр `MBeanServer` будет создаваться автоматически, поэтому никакого дополнительного конфигурирования не требуется.

Внутри бина `jmxExporter` свойство `beans` определяет бины Spring, которые необходимо открыть. Это свойство типа `Map`, поэтому в нем можно указать любое количество MBean-компонентов. В нашем случае нужно открыть бин `appStatisticsBean`, содержащий информацию о приложении контактов, которую мы хотим показать администраторам. В определении MBean-компонента ключ будет использоваться как `ObjectName` (класс `javax.management.ObjectName` в JDK) для бина Spring, на который ссылается соответствующее значение записи. В приведенной выше конфигурации MBean-компонент `appStatisticsBean` будет открыт с `ObjectName` вида `bean:name=Prospring4ContactApp`. По умолчанию все открытые свойства бина видны как атрибуты, а все открытые методы — как операции.

Теперь MBean-компонент доступен для мониторинга через JMX. А теперь займемся настройкой VisualVM и применением клиента JMX в целях мониторинга.

Настройка VisualVM для мониторинга JMX

VisualVM — очень полезный инструмент, помогающий проводить мониторинг разнообразных аспектов Java-приложений. Это бесплатный инструмент, который находится в папке `bin` внутри папки установки JDK. На веб-сайте проекта также доступна для загрузки автономная версия VisualVM (`http://visualvm.java.net/download.html`). В этой главе мы будем пользоваться автономной версией, которой на момент написания книги была 1.3.8.

Для поддержки различных функций мониторинга в VisualVM применяется система подключаемых модулей. Чтобы обеспечить мониторинг MBean-компонентов Java-приложений, потребуется установить подключаемый модуль VisualVM-MBeans. Выполните следующие шаги.

1. Выберите в меню VisualVM пункт Tools⇨Plugins (Сервис⇨Подключаемые модули).
2. В открывшемся диалоговом окне Plugins (Подключаемые модули) перейдите на вкладку Available Plugins (Доступные подключаемые модули).
3. Щелкните на кнопке Check for Newest (Проверить наличие самых новых).
4. Отметьте флажок возле подключаемого модуля VisualVM-MBeans и щелкните на кнопке Install (Установить).

По завершении установки удостоверьтесь, что сервер Tomcat функционирует, а пример приложения запущен. Затем в представлении Applications (Приложения) в левой части экрана VisualVM вы должны увидеть выполняющийся процесс Tomcat.

По умолчанию VisualVM ищет Java-приложения, которые функционируют под управлением платформы JDK. Двойной щелчок на желаемом узле приводит к отображению экрана мониторинга.

После установки подключаемого модуля VisualVM-MBeans появляется вкладка MBeans (MBean-компоненты). Перейдя на нее, можно просмотреть доступные MBean-компоненты. Слева должен быть виден узел под названием bean (бин). Если развернуть его, появится MBean-компонент `Prospring4ContactApp`, который был открыт.

В правой части показан метод, реализованный в бине, с атрибутом `TotalContactCount` (который был автоматически выведен методом `getTotalContactCount()` внутри бина). Значением является 3, что соответствует количеству записей, добавленных в базу данных при запуске приложения. В обычном приложении это число изменялось бы на основе количества контактов, которые добавляются во время выполнения приложения.

Мониторинг статистики Hibernate

В Hibernate также поддерживается управление и открытие для JMX метрик, связанных с постоянством. Чтобы сделать это доступным, понадобится добавить в конфигурацию JPA (файл `datasource-tx-jpa.xml`) два дополнительных свойства Hibernate, как показано в листинге 15.4.

Листинг 15.4. Включение статистики Hibernate

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:jdbc="http://www.springframework.org/schema/jdbc"
      xmlns:jpa="http://www.springframework.org/schema/data/jpa"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/jdbc
       http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
       http://www.springframework.org/schema/data/jpa
       http://www.springframework.org/schema/data/jpa/spring-jpa.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd">

   <jdbc:embedded-database id="dataSource" type="H2">
      <jdbc:script location="classpath:META-INF/config/schema.sql"/>
      <jdbc:script location="classpath:META-INF/config/test-data.sql"/>
   </jdbc:embedded-database>

   <bean id="transactionManager"
         class="org.springframework.orm.jpa.JpaTransactionManager">
      <property name="entityManagerFactory" ref="emf"/>
   </bean>

   <tx:annotation-driven transaction-manager="transactionManager" />

   <bean id="emf"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
      <property name="dataSource" ref="dataSource" />
```

```
      <property name="jpaVendorAdapter">
         <bean
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
      </property>
      <property name="packagesToScan" value="com.apress.prospring4.ch15"/>
      <property name="jpaProperties">
         <props>
           <prop key="hibernate.dialect">org.hibernate.dialect.H2Dialect
           </prop>
           <prop key="hibernate.max_fetch_depth">3</prop>
           <prop key="hibernate.jdbc.fetch_size">50</prop>
           <prop key="hibernate.jdbc.batch_size">10</prop>
           <prop key="hibernate.show_sql">true</prop>
           <!-- Свойства статистики Hibernate -->
           <prop key="hibernate.generate_statistics">true</prop>
           <prop key="hibernate.session_factory_name">sessionFactory</prop>
         </props>
      </property>
   </bean>

   <context:annotation-config/>

   <jpa:repositories base-package="com.apress.prospring4.ch15"
                     entity-manager-factory-ref="emf"
                     transaction-manager-ref="transactionManager"/>
</beans>
```

Свойство `hibernate.generate_statistics` инструктирует Hibernate о необходимости генерации статистики для его поставщика постоянства JPA, тогда как свойство `hibernate.session_factory_name` определяет имя фабрики сеансов, требуемое MBean-компонентом статистики Hibernate. Оба свойства находятся ниже комментария “Свойства статистики Hibernate”.

Наконец, этот MBean-компонент нужно добавить в конфигурацию `MBeanExporter` из Spring. В листинге 15.5 приведена модифицированная конфигурация, которая была создана ранее в файле `rest-context.xml`.

Листинг 15.5. MBean-компонент для статистики Hibernate

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:mvc="http://www.springframework.org/schema/mvc"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc.xsd">

   <mvc:annotation-driven>
      <mvc:message-converters>
         <bean class="org.springframework.http.converter.json.
MappingJackson2HttpMessageConverter"/>
```

```
        <bean class="org.springframework.http.converter.xml.
MarshallingHttpMessageConverter">
            <property name="marshaller" ref="castorMarshaller"/>
            <property name="unmarshaller" ref="castorMarshaller"/>
        </bean>
    </mvc:message-converters>
</mvc:annotation-driven>

<context:component-scan base-package="com.apress.prospring4.ch15"/>

<bean id="castorMarshaller"
      class="org.springframework.oxm.castor.CastorMarshaller">
    <property name="mappingLocation"
              value="classpath:META-INF/spring/oxm-mapping.xml"/>
</bean>
<bean id="appStatisticsBean"
      class="com.apress.prospring4.ch15.AppStatisticsImpl"/>
<bean id="jmxExporter"
      class="org.springframework.jmx.export.MBeanExporter">
    <property name="beans">
        <map>
            <entry key="bean:name=ProSpring4ContactApp"
                   value-ref="appStatisticsBean"/>
            <entry key="bean:name=Prospring4ContactApp-hibernate"
                   value-ref="statisticsBean"/>
        </map>
    </property>
</bean>
<bean id="statisticsBean" class="org.hibernate.jmx.StatisticsService">
    <property name="statisticsEnabled" value="true"/>
    <property name="sessionFactoryJNDIName" value="sessionFactory"/>
</bean>
</beans>
```

Здесь объявлен новый бин `statisticsBean` с классом `StatisticsService` из Hibernate в качестве реализации. С помощью этого класса Hibernate поддерживает открытие статистики для JMX.

Обратите внимание на свойство `sessionFactoryJNDIName`, которое должно соответствовать свойству, определенному в листинге 15.4 (`hibernate.session_factory_name`). Внутри бина `jmxExporter` объявлен еще один бин с `ObjectName` вида `bean:name=Prospring4ContactApp-hibernate`, который ссылается на бин `statisticsBean`.

Теперь статистика Hibernate включена и доступна через JMX. Перезагрузите приложение, и после обновления VisualVM можно будет видеть MBean-компонент статистики Hibernate. Щелчок на узле `ProSpring3ContactApp-hibernate` приводит к отображению подробной статистики с правой стороны. На полях с информацией, не относящейся к элементарным типам Java (например, `List`), можно щелкать, чтобы раскрывать их для просмотра содержимого.

В VisualVM можно видеть множество других метрик, в том числе `EntityNames`, `SessionOpenCount`, `SecondCloseCount` и `QueryExecutionMaxTime`. Их значения полезны для понимания поведения постоянства внутри приложения и могут помочь при поиске и устранении неполадок, а также при настройке производительности.

Резюме

В этой главе были раскрыты высокоуровневые темы, связанные с мониторингом JEE-приложений на основе на Spring.

Прежде всего, мы обсудили поддержку JMX в Spring (JMX — это стандарт для мониторинга Java-приложений). Затем мы продемонстрировали реализацию специальных MBean-компонентов для открытия информации, связанной с приложением, а также открытие статистических данных по общим компонентам, таким как Hibernate.