Chapter 15 JMX 연동 지원

: JMX(Java Management Extensions) API는 Java의 표준 API로서 Java 객체를 관리하고 모니터링 하는 데 사용된다. SPRING은 빈 객체가 JMX API를 구현했느냐

안 했느냐의 여부에 상관없이 빈 객체를 JMX MBean으로 등록해 주는 기능을 제공하고 있다. 따라서, SPRING을 사용하면 간단한 설정만으로 기존의 POJO 객체를

JMX를 통해 관리하고 모니터링 할 수 있다.

1. SPRING 빈을 MBean으로 등록하기

: 다음 Code는 단순한 Class로서 게 개의 프로퍼티(viewCount, sumOfResponse, Time, averageResponseTime)와 데이터 처리를 위한 두 개의 메서드

(increaseViewCount, reset)를 제공하고 있다.

package madvirus.spring.chap15.monitor;

public class PerformanceMonitor {

private int viewCount;

private long sumOfResponseTime;

private double averageResponseTime;

public int getViewCount() {

return viewCount;

}

public void setViewCount(int pageView) {

this.viewCount = pageView;

}

public long getSumOfResponseTime() {

return sumOfResponseTime;

}

public void setSumOfResponseTime(long sumOfResponseTime) {

this.sumOfResponseTime = sumOfResponseTime;

}

public double getAverageResponseTime() {

return averageResponseTime;

}

public void setAverageResponseTime(double averageResponseTime) {

this.averageResponseTime = averageResponseTime;

}

public void increaseViewCount(long responseTime) {

viewCount++;

sumOfResponseTime += responseTime;

averageResponseTime = (double) sumOfResponseTime / (double) viewCount;

}

public void reset() {

viewCount = 0;

sumOfResponseTime = 0;

averageResponseTime = 0.0;

}

}

SPRING을 사용해서 앞의 Source Code의 PerformanceMonitor Class를 JMX MBean으로 등록하려면 MBeanExporter만 사용하면 된다. 아래 Code는 설정 예이다.

<bean id="monitor" class="madvirus.spring.chap15.monitor.PerformanceMonitor" />

<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter">

<property name="beans">

<map>

<entry key="madvirus:name=monitor" value-ref="monitor" />

</map>

</property>

</bean>

MBeanExporter의 beans 프로퍼티는 MBean으로 등록할 SPRING 빈을 Map으로 전달 받는다. 이때 Map에 포함된 각 항목의 키는 MBeanServer에 SPRING이 생성한

MBean을 등록할 때 사용할 ObjectName으로 사용되며, 각 항목의 값으로부터 MBean 객체를 생성하게 된다. 예를 들어, 위 Code의 경우 "monitor" 빈 객체로부터

MBean 객체를 생성하게 되며, 생성된 MBean 객체를 MBeanServer에 등록할 때 사용되는 ObjectName은 "madvirus:name=monitor"가 된다.

MBeanExporter는 기본적으로 실행 중인 MBeanServer를 사용한다. 따라서, 톰캣이나 JBoss와 같이 MBeanServer를 제공하는 Application 서버를 사용할 경우,

생성한 MBean을 해당 MBeanServer에 등록한다. Java5 이상 버전의 경우에도 기본적으로 MBeanServer를 제공하고 있기 때문에, Java5나 6버전을 사용하는

경우 MBeanExporter가 생성한 MBean 객체는 Java 플랫폼이 제공하는 기본 MBeanServer에 등록한다.

Java6의 jconsolde을 이용해서 SPRING이 등록한 MBean의 정보를 조회한 결과 화면을 제공한다. MBeanExporter를 통해서 등록한 MBean 객체가 존재하는 것을

확인할 수 있다.

2. MBeanServer 설정

: Java 1.4 이하 버전을 사용하거나 Application 서버에서 MBeanServer를 제공하지 않는 경우, 또는 기존에 제공되는 MBeanServer가 아닌 새로운 MBeanServer를

생성하고 싶은 경우에는 SPRING이 제공하는 MBeanServerFactoryBean을 이용해서 MBeanServer를 직접 생성할 수 있다.

<bean id="mbeanServer" class="org.springframework.jmx.support.MBeanServerFactoryBean"

p:locateExistingServerIfPossible="false" p:defaultDomain="madvirus" />

<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter">

<property name="server" ref="mbeanServer" />

<property name="beans">

<map>

<entry key="madvirus:name=monitor" value-ref="monitor" />

</map>

</property>

</bean>

MBeanServerFactoryBean은 개존에 동작하는 MBeanServer의 존재 여부에 상관 없이 항상 새로운 MBeanServer 객체를 생성한다. 새롭게 생성한 MBeanServer에

SPRING 빈 객체를 MBean으로 등록하고 싶다면 MBeanExporter의 server 프로퍼티에 생성한 MBeanServer 객체를 설정하면 된다.

만약 기존에 동작 중인 MBeanServer가 존재할 경우, MBeanServerFactoryBean이 새로운 MBeanServer를 생성하지 않고 기존 MBeanServer를 사용하도록 하려면

아래 Code와 같이 locateExistingServerIfPossible 프로퍼티의 값을 true로 설정하면 된다.

<bean id="mbeanServer" class="org.springframework.jmx.support.MBeanServerFactoryBean"

p:locateExistingServerIfPossible="true" p:defaultDomain="madvirus" />

2.1 MBeanServerFactoryBean과 MBeanExporter의 설정 순서

: MBeanExporter는 생성한 MBean을 MBeanServer에 등록할 때, 다음과 같이 server 프로퍼티를 이용해서 MBean을 등록할 MBeanServer를 지정할 수 있다.

<bean id="mbeanServer" class="org.springframework.jmx.support.MBeanServerFactoryBean"

p:locateExistingServerIfPossible="false" p:defaultDomain="madvirus" />

<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter">

<property name="server" ref="mbeanServer" />

<property name="beans">

<map>

<entry key="madvirus:name=monitor" value-ref="monitor" />

</map>

</property>

</bean>

MBeanExporter의 server 속성을 지정하지 않을 경우, 현재 존재하는 MBeanServer에 MBean을 등록하게 된다. 만약 다음과 같이 MBeanServerFactoryBean의

설정이 MBeanExporter 설정보다 앞에 위치한다면, MBeanExporter는 MBeanServerFactoryBean이 생성한 MBeanServer에 MBean을 등록한다.

<bean id="mbeanServer" class="org.springframework.jmx.support.MBeanServerFactoryBean"

p:locateExistingServerIfPossible="false" p:defaultDomain="madvirus" />

<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter">

<property name="beans">

<map>

<entry key="madvirus:name=monitor" value-ref="monitor" />

</map>

</property>

</bean>

그런데, 다음과 같이 MBeanExporter 설정보다 MBeanServerFactoryBean 설정이 뒤에 위치하면, MBeanExporter 빈 객체가 생성된 다음에 MBeanServerFactoryBean이

생성한 MBeanServer가 생성되기 때문에, MBeanExporter는 MBeanServerFactoryBean이 생성한 MBeanServer가 아닌 플랫폼의 기본 MBeanServer를 사용하게 된다.

<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter">

<property name="beans">

<map>

<entry key="madvirus:name=monitor" value-ref="monitor" />

</map>

</property>

</bean>

<bean id="mbeanServer" class="org.springframework.jmx.support.MBeanServerFactoryBean"

p:locateExistingServerIfPossible="false" p:defaultDomain="madvirus" />

따라서, MBeanExporter와 MBeanServerFactoryBean을 함께 사용할 경우에는 성정의 순서를 주의해야 한다. 그렇지 않을 경우 MBeanExporter가 생성한 MBean 객체가

원하지 않는 MBeanServer에 등록될 수 있다. 따라서, MBeanServerFactoryBean을 이용해서 생성한 MBeanServer를 사용하고 싶은 경우에는 MBeanExporter의

server 프로퍼티를 사용해서 명시적으로 MBeanServer를 지정해주는 것이 좋다.

3. MBean Interface 결정

: 앞에서 PerformanceMonitor Class의 모든 public get/set 메서드가 속성으로 등록되고 모든 public 메서드가 오퍼레이션으로 등록된 것을 확인할 수 있다. 따라서,

등록된 MBean 객체의 모든 속성을 읽기/쓰기 가능하고 모든 오퍼레이션을 호출할 수 있다. 하지만, 경우에 따라서 읽기 전용으로 속성을 제공하고 싶을때도 있고,

모든 public 메서드가 아닌 특정한 메서드만 오퍼레이션으로 공개하고 싶을때가 있을 것이다.

MBeanExporter는 빈 객체를 MBean으로 등록할 때 어떤 속성과 어떤 오퍼레이션을 공개할 지에 대한 여부를 결정할 때 MBeanInfoAssembler 구현 객체를 사용한다.

기본 구현 Class인 SimpleReflectiveMBeanInfoAssembler는 앞서 설명했듯이 모든 public 메서드를 MBean의 프로퍼티와 오퍼레이션으로 등록한다.

SPRING은 기본 방식 외에도 메서드 이름을 지정하는 방식, Java Interface를 지정하는 방식, 그리고 Annotation을 이용해서 지정하는 방식을 지원하고 있다.

본 절에서는 이들 방식에 대해서 차례대로 살펴보도록 하겠다.

3.1 메서드 이름을 이용한 MBean 객체 설정

: MethodNameBasedMBeanInfoAssembler를 사용하면 지정한 메서드만 MBean에 의해 관리되는 Interface로 설정할 수 있다. 아래 Code와 같이 MBeanExporter의

assembler 프로퍼티에 MethodNameBasedMBeanInfoAssembler 객체를 전달해 주면 되는데, 이때 managedMethods 프로퍼티를 이용해서 MBean Interface로

사용될 메서드 이름 목록을 입력한다.

<bean id="monitorByName" class="madvirus.spring.chap15.monitor.PerformanceMonitor" />

<bean id="assembler"

class="org.springframework.jmx.export.assembler.MethodNameBasedMBeanInfoAssembler">

<property name="managedMethods">

<list>

<value>getViewCount</value>

<value>getSumOfResponseTime</value>

<value>getAverageResponseTime</value>

<value>reset</value>

</list>

</property>

</bean>

<bean id="exporterByName" class="org.springframework.jmx.export.MBeanExporter">

<property name="assembler" ref="assembler" />

<property name="beans">

<map>

<entry key="madvirus:name=monitorByMethod" value-ref="monitorByName" />

</map>

</property>

</bean>

위 Code는 get 메서드와 reset 메서드를 managedMethods의 값으로 지정했다. 따라서 ViewCount, SumOfResponseTime, AverageResponseTime 속성은 읽기 전용

프로퍼티가 되며, 그 외 나머지 메서드들 중에서 reset() 메서드만 오퍼레이션으로 제공된다.

3.2 Java Interface를 이용한 MBean 객체 설정

: InterfaceBasedMBeanInfoAssembler Class를 이용하면 지정한 Java Interface에 정의된 메서드에 따라 MBean 객체의 Interface가 결정된다. 예를 들어, 앞서

작성했던 PerformanceMonitor를 MBean으로 등록할 때 increaseViewCount() 오퍼레이션과 ViewCount 속성만 MBean Interface로 제공하고 싶다고 하자. 이 경우

아래 Source와 같이 제공하고 싶은 메서드만을 선언하고 있는 Interface를 작성한다.

package madvirus.spring.chap15.monitor;

public interface ManagerPerformanceMonitor {

void increaseViewCount(long responseTime);

int getViewCount();

void setViewCount();

}

MBean Interface로 제공하고 싶은 Interface를 작성했다면, InterfaceBasedMBeanInfoAssembler 객체를 생성한 뒤 managedInterfaces 프로퍼티로 Interface

목록을 지정하면 된다.

<bean id="assembler"

class="org.springframework.jmx.export.assembler.InterfaceBasedMBeanInfoAssembler">

<property name="managedInterfaces">

<list>

<value>madvirus.spring.chap15.monitor.ManagerPerformanceMonitor</value>

</list>

</property>

</bean>

<bean id="exporterByIF" class="org.springframework.jmx.export.MBeanExporter"

assembler-ref="assemlberByIF">

<property name="beans">

<map>

<entry key="madvirus:name=monitorByIF" value-ref="monitorByIF" />

</map>

</property>

</bean>

InterfaceBasedMBeanInfoAssembler가 Interface로 생성한 MBean 객체를 생성한다.

(Java Interface를 사용하면 MBean Interface를 설정할 수 있다.)

ViewCount 속성과 increaseViewCount() 오퍼레이션만 MBean Interface로 제공된 것을 확인할 수 있다.

InterfaceBasedMBeanInfoAssembler와 관련해서 주목할 점은 MBean으로 등록된 SPRING 빈이 지정한 Interface를 구현하지 않아도 된다는 점이다. 예제에서도

PerformanceMonitor Class가 ManagedPerformanceMonitor Interface를 구현하지 않은 것을 알 수 있다.

3.3 Annotation을 이용한 MBean 객체 설정

: Annotation을 이용해서 Source Code에 직접 MBean 설정을 명시할 수도 있다. SPRING은 Java Code에 MBean 관련 정보를 명시할 수 있는 Annotation을 제공하고 있는데,

이 중에서 @ManagedResource, @ManagedAttribute, @ManagedOperation Annotation이 주로 시용된다. 아래 Code는 Annotation을 이용하여 MBean 정보를 Code에

설정한 예를 보여 주고 있다.

package madvirus.spring.chap15.monitor;

import org.springframework.jmx.export.annotation.ManagedAttribute;

import org.springframework.jmx.export.annotation.ManagedOperation;

import org.springframework.jmx.export.annotation.ManagedResource;

@ManagedResource(objectName = "madvirus:name=AnnotMonitor", description = "Annotation 이용")

public class AnnotPerformanceMonitor {

private int viewCount;

private long sumOfResponseTime;

private double averageResponseTime;

@ManagedAttribute

public int getViewCount() {

return viewCount;

}

@ManagedAttribute

public void setViewCount(int pageView) {

this.viewCount = pageView;

}

@ManagedAttribute

public double getAverageResponseTime() {

return averageResponseTime;

}

public void setAverageResponseTime(double averageResponseTime) {

this.averageResponseTime = averageResponseTime;

}

@ManagedOperation

public void increaseViewCount(long responseTime) {

viewCount++;

sumOfResponseTime += responseTime;

averageResponseTime = (double) sumOfResponseTime / (double) viewCount;

}

@ManagedOperation

public void reset() {

viewCount = 0;

sumOfResponseTime = 0;

averageResponseTime = 0.0;

}

}

@ManagedResource Annotation은 Class에 적용되며 objectName 멜리먼트를 이용해서 MBean의 ObjectName을 설정한다.

@ManagedAttribute Annotation은 속성으로 제공하고 싶은 메서드에 적용된다. @ManagedAttribute Annotation을 get 메서드에만 적용하면 읽기 전용이

되고, set 메서드에만 적용하면 쓰기 전용이 되며, get/set 두 메서드에 함께 적용하면 읽기/쓰기가 모두 되는 속성이 된다.

@ManagedOperation Annotation은 오퍼레이션으로 제공할 메서드에 적용한다.

Annotation을 이용해서 MBean Interface 정보를 설정했다면, 아래 Code와 같이 MBeanExporter를 설정함으로써 Annotation이 적용된 빈 객체를 MBean으로

등록할 수 있게 된다.

<bean id="annotMonitor" class="madvirus.spring.chap15.monitor.AnnotPerformanceMonitor" />

<context:mbean-export server="mbeanServer"/>

<bean id="attributeSource"-->

class="org.springframework.jmx.export.annotation.AnnotationJmxAttributeSource" />

<bean id="annotAsselbler"-->

class="org.springframework.jmx.export.assembler.MetadataMBeanInfoAssembler"

p:attributeSource-ref="attributeSource" />

<bean id="namingStrategy"

class="org.springframework.jmx.export.naming.MetadataNamingStrategy"

p:attributeSource-ref="attributeSource" />

<bean id="exporterByAnnot" class="org.springframework.jmx.export.MBeanExporter"

p:server-ref="mbeanServer" p:assembler-ref="annotAsselbler"

p:namingStrategy-ref="namingStrategy" p:autodetect="true" />

Annotation을 이용해서 MBean Interface를 설정하려면 다음의 세 가지 빈 객체를 등록해 주어야 한다.

- AnnotationJmxAttributeSource : Annotation에서 MBean 정보를 읽어 온다.

- MetadataMBeanInfoAssembler : 지정한 JmxAttributeSource로 부터 메타 정보를 읽어와 MBean Interface를 생성한다.

- MetadataNamingStrategy : 지정한 JmxAttributeSource를 통해서 구한 메타 정보를 이용해서 ObjectName을 생성한다.

각 빈을 설정했다면 MBeanExporter의 assembler 프로퍼티와 namingStrategy 프로퍼티의 값으로 각각 MetadataMBeanInfoAssembler 빈 객체와 MetadataNamingStrategy

빈 객체를 전달해 주면 된다.

만약, Annotation이 적용된 빈 객체를 자동으로 지정하지 않고 MBeanExpoerter에 직접 설정하고 싶다면, autodetect 프로퍼티를 설정하지 않고 bean 프로퍼티를

그대로 사용하면 된다.

Annotation이 적용된 SPRING 빈 객체를 MBean 객체로 등록하려면 세 개의 빈 객체(AnnotationJmxAttributeSource, MetadataMBeanInfoAssembler, MetadataNamingStrategy)

를 필요로 하는데, SPRING 2.5버전 부터 추가된 <context:mbean-export> 태그를 이용하면 다음과 같이 한 줄로 설정을 끝낼 수 있다.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

...

<context:mbean-export server="mbeanServer"/>

...

</beans>

<context:mbean-export> 태그는 내부적으로 Annotation 기반의 빈 객체를 MBean 객체로 등록하는 필요한 모즌 작업을 처리해 준다. 따라서, <context:mbean-export>

태그를 사용하면 다른 복잡한 설정 없이 Annotation이 적용된 빈 객체를 Mbean 객체로 등록할 수 있다.

<context:mbean-export> 태그는 플랫폼에서 기본적으로 제공하는 MBeanServer를 사용한다. 플랫폼 MBeanServer가 아닌 직접 생성한 MBeanServer를 사용하고 싶다면

위 Code처럼 server 프로퍼티를 이용해서 MBeanServer를 설정하면 된다.

4. MBeanServer 원격으로 제공

: MBeanServer는 원격지 프로세스(로컬의 다른 프로세스 포함)에서 직접 MBeanServer에 접근할 수 있는 방법을 제공하지 않는다. 대신 JSR 160(JMX Remote API)에서

정의한 커넥터 서버를 통해 원격지 프로세스에서 MBeanServer에 접근할 수 있도록 하였다.

SPRING은 ConnectorServerFactoryBean Class를 사용해서 원격지에서 MBeanServer에 접근할 수 있도록 해 주는 JMXConnectorServer를 생성해 준다.

아래 Code는 ConnectorServerFactoryBean의 설정 예이다.

<bean id="connectorServer"

class="org.springframework.jmx.support.ConnectorServerFactoryBean"

p:serviceUrl="service:jmx:rmi://localhost/jndi/rmi://localhost:1099/madvirus"

p:objectName="connector:name=rmi" />

ConnectorServerFactoryBean의 serviceUrl 프로퍼티는 원격지에서 커넥터 서버에 연결할 때 사용할 URL을 입력한다. serviceUrl 프로퍼티의 기본 값은

"service:jmx:jmxmp://localhost:9875"인데, Java5나 6버전은 JMXMP 프로토콜을 지원하고 있지 않다. 그래서 위 Code에서는 RMI 기반의 URL을 입력하였다.

RMI 기반의 프로토콜을 사용할 경우 RMI Registry가 실행 중이어야 한다. 만약, 별도로 RMI Registry를 실행하지 않는다면, SPRING이 제공하는 RmiRegistryFactoryBean을

사용해서 RMI Registry를 실행할 수 있다.

<bean id="connectorServer"

class="org.springframework.jmx.support.ConnectorServerFactoryBean"

depends-on="registry" p:server-ref="mbeanServer"

p:serviceUrl="service:jmx:rmi://localhost/jndi/rmi://localhost:1099/madvirus"

p:objectName="connector:name=rmi" />

<bean id="registry" class="org.springframework.remoting.rmi.RmiRegistryFactoryBean"

p:port="1099" />

ConnectorServerFactoryBean의 server 프로퍼티는 커넥터 서버가 연결할 MBeanServer를 설정한다. server 프로퍼티를 설정하지 않으면 실행중인 MBeanServer를

선택한다.

objectName 프로퍼티는 ConnectorServerFactoryBean가 생성한 JMXConnector를 MBeanServer에 등록할 때 사용될 ObjectName을 입력한다.

ConnectorServerFactoryBean를 사용해서 MBeanServer와 연결된 커넥터 서버를 생성하면, 원격지에서 지정한 URL을 통해서 MBeanServer에 접근할 수 있게 된다.

5. 원격지 MBeanServer 접근 및 사용

: 원격지의 MBeanServer에 연결하면 MBean 객체의 속성을 조작하거나 오퍼레이션을 호출할 수 있다. SPRING은 원격지의 MBeanServer에 접근할 수 있는 두 가지 방법을

제공하고 있다. 첫 번째 방법은 MBeanServerConnectionFactoryBean을 이용해서 MBeanServer에 연결된 MBeanServerConnection을 생성해서 MBeanServer에 접근하는

것이고, 두 번째 방법은 MBeanProxyFactoryBean을 이용해서 생성한 프록시 객체를 이용해서 MBeanServer 서버에 접근하는 것이다.

5.1 MBeanProxyFactoryBean을 이용한 접근

: javax.management.MBeanServerConnection Interface는 원격지 MBeanServer에 연결해서 원격지 MBeanServer에 등록된 MBean을 관리하고 모니터링 하는 용도로

사용된다. SPRING의 MBeanServerConnectionFactoryBean Class는 원격지의 MBeanServer의 커넥터에 접속하는 MBeanServerConnection을 생성해주는 기능을

제공하며, 이를 통해 원격지의 MBeanServer를 관리할 수 있다. 아래 Code는 MBeanServerConnectionFactory Class의 설정 예이다.

<bean id="mbeanServerlClient"

class="org.springframework.jmx.support.MBeanServerConnectionFactoryBean"

p:serviceUrl="service:jmx:rmi://localhost/jndi/rmi://localhost:1099/madvirus" />

serviceUrl 프로퍼티는 원격지 MBeanServer의 커넥터 서버에 연결할 때 사용되는 서비스 URL이다. (앞서 ConnectionServerFactoryNeam의 serviceUrl 프로퍼티의 값과

동일한 값을 갖는다.)

MBeanServerConnectionFactoryBean은 MBeanServerConnection 객체를 생성한다. 따라서, 원격지 MBeanServer에 접근해야 하는 객체는 MBeanServerConnection을

이용해서 필요한 작업을 수행할 수 있다. 아래 Code는 MBeanServerConnection을 이용해서 원격지 MBeanServer에 등록된 MBean 객체의 속성 정보를 읽어오는

예를 보여주고 있다.

package madvirus.spring.chap15.monitor;

import java.io.IOException;

import javax.management.AttributeNotFoundException;

import javax.management.InstanceNotFoundException;

import javax.management.MBeanException;

import javax.management.MBeanServerConnection;

import javax.management.MalformedObjectNameException;

import javax.management.ObjectName;

import javax.management.ReflectionException;

public class MonitorDataReader {

private MBeanServerConnection connection;

public void setConnection(MBeanServerConnection connection) {

this.connection = connection;

}

public int getViewCount() {

ObjectName mbeanName = null;

try {

mbeanName = new ObjectName("madvirus:name=monitor");

} catch (MalformedObjectNameException e) {

throw new RuntimeException("invalid name : " + e.getMessage());

}

Integer viewCount;

try {

viewCount = (Integer) connection.getAttribute(mbeanName,

"ViewCount");

return viewCount.intValue();

} catch (AttributeNotFoundException e) {

throw new RuntimeException(e);

} catch (InstanceNotFoundException e) {

throw new RuntimeException(e);

} catch (MBeanException e) {

throw new RuntimeException(e);

} catch (ReflectionException e) {

throw new RuntimeException(e);

} catch (IOException e) {

throw new RuntimeException(e);

}

}

}

MBeanServerConnection은 MBean 객체를 조회하는 기능뿐만 아니라 MBean 객체를 등록하고 리스너 관리 기능도 제공하고 있기 때문에 여러 MBean 객체를

다루어야 할때 적당하다.

5.2 MBeanServerConnectionFactoryBean을 이용한 접근

: javax.management.MBeanServerConnection Interface를 이용해서 직접 MBeanServer에 접근하고 관련된 기능을 사용할 수 있지만, MBeanServerConnection을

사용하면 앞서 Code에서 보듯이 꽤 많은 양의 try~catch 블럭을 작성해 주어야만 한다.

만약, 특정한 MBean 객체에 접근해야 하는데 위 Code 같이 성가신 try~catch 블럭을 처리하고 싶지 않다면 MBeanProxyFactoryBean을 이용해서 MBeanServerConnection과

연결된 프록시 기반의 객체를 생성하면 된다. 아래 Code는 설정 예이다.

<bean id="performanceMonitor"

class="org.springframework.jmx.access.MBeanProxyFactoryBean"

p:objectName="madvirus:name=monitor" p:server-ref="mbeanServerlClient"

p:proxyInterface="madvirus.spring.chap15.monitor.PerformanceMonitorInterface" />

<bean id="mbeanServerlClient"

class="org.springframework.jmx.support.MBeanServerConnectionFactoryBean"

p:serviceUrl="service:jmx:rmi://localhost/jndi/rmi://localhost:1099/madvirus" />

MBeanProxyFactoryBean은 proxyInterface 프로퍼티로 지정한 Interface를 구현한 프록시 객체를 생성한다. 이 프록시 객체는 내부적으로 MBeanServerConnection을

이용해서 원격지 MBeanServer의 MBean 객체에 접근하게 된다. server프로퍼티를 이용해서 원격지 MBeanServerConnection을 전달받고, objectName 프로퍼티를

이용해서 사용할 원격지 MBeanServer의 MBean 객체 이름을 설정한다.

proxyInterface 프로퍼티로 지정한 Interface는 원격지 MBeanServer의 MBean 객체가 제공하는 속성 및 오퍼레이션과 Mapping되는 메서드를 선언해야 한다. 예를 들어,

위 Code에서 설정한 Interface는 다음과 같이 정의되어 있다.

package madvirus.spring.chap15.monitor;

public interface PerformanceMonitorInterface {

int getViewCount();

long getSumOfResponseTime();

double getAverageResponseTime();

void increaseViewCount(long responseTime);

void reset();

}

MBeanProxyFactoryBean가 생성한 프록시 객체는 proxyInterface 프로퍼티로 지정한 Interface를 구현하고 있으므로 다음과 같이 Interface를 사용해서 원격지의

MBean 객체를 사용할 수 있게 된다.

private PerformanceMonitorInterface monitor = null;

public void setMonitor(PerformanceMonitorInterface monitor){

this.monitor = monitor;

}

public int getViewCount(){

return monitor.getViewCount();

}

프록시 객체는 원격지 MBeanServer와 연동하는 과정에서 예외가 발생할 경우 알맞은 org.springframework.jmx.JmxException으로 변환해서 발생시킨다.

JmxException은 RuntimeException이기 때문에 try~catch 블럭을 필요로 하지 않으며, 필요한 예외에 대해서만 catch 블록에서 처리해 주면 된다. 따라서,

성가신 try~catch 블럭 Code를 작성하지 않고도 원격지 MBean 객체를 사용할 수 있게 된다.