一旦应用已经部署并且正在运行，单独是哟共DI冰崩你改变应用的配置。如果要在运行时改变应员工的配置，需要使用Java管理扩展(java Management Extensions， JMX)。

JMX能过够管理、监视和配置应用。使用JMX管理应用的核心足迹教案是托管bean(managed bean，MBean)。即：暴露特定方法的JavaBean

JMX规范定义的MBean：

- **标准MBean**：标准的MBean的管理接口时通过在固定的几口上执行反射确定的，bean类会实现这个接口
- **动态MBean**：动态MBean的管理接口实在运行时通过调用DynamicMBean接口的方法来确定。应为管理接口不是通过静态接口定义的，因此可以在运行时改变
- **开放MBean**：开放MBean是一种特殊的动态MBean，其属性和方法只限定于原始类型。原始类型的包装类以及可以分解为原始类型或原始类型包装类的任意类型
- **模型MBean**：模型MBean也是一种特殊的动态MBean，用于充当管理接口与受管理资源的中介。模型MBean并不像他们所声明的那样来编写。他们通常通过工厂生成，工厂会使用元信息来组装管理接口。

Spring的JMX木块可以将Spring】 bean导出为模型MBean，这样就可以查看应用程序的内容不情况并且能够更改配置--甚至在应用的运行期。

1 将Spring Bean导出为MBean

```
public class ExportController {

    private static final String MAX_LONG_AS_STRING = "9223372036854775807";

    public static final int DEFAULT_SPITTLES_PER_PAGE = 25;

    public int spittlesPerPage = DEFAULT_SPITTLES_PER_PAGE;

    public int getSpittlesPerPage() {
        return spittlesPerPage;
    }

    public void setSpittlesPerPage(int spittlesPerPage) {
        this.spittlesPerPage = spittlesPerPage;
    }
}
```

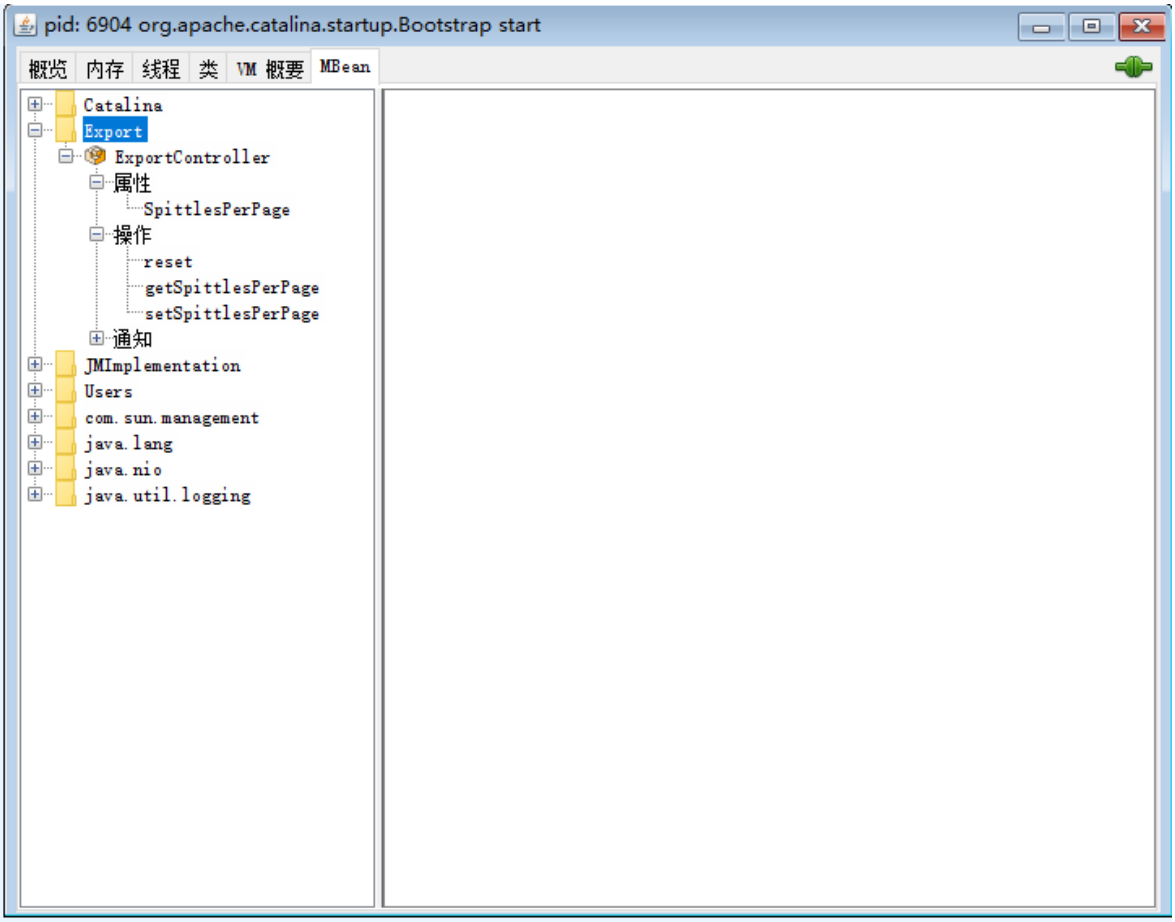
```
@Configuration
public class ExportConfig {

    @Bean
    public ExportController exportController() {
        return new ExportController();
    }

    @Bean
    public MBeanExporter mBeanExporter(ExportController exportController) {
        MBeanExporter exporter = new MBeanExporter();
        Map<String, Object> beans = new HashMap<>(1);
        beans.put("Export:name=ExportController", exportController);
```

```
        exporter.setBeans(beans);
        return exporter;
    }
}
```

下一步将ExportController暴露为MBean，而spittlesPerPage属性将成为MBean的托管属性。这样就可以在运行时改变该属性的值。

MBeanExporter可以把一个或多个Spring bean导出为MBean服务器内的模型MB而安。MBean服务器(也被陈伟MBean代理)是MBean身存的容器。对MBean的访问，也是通过MBean服务器来实现的。

将Springbean导出为JMX mBean之后，可以使用基于JMX的管理工具(JConsole、VisualVM)查看正在运行的应用程序，显式bean的星星并调用bean的方法



ExportController所有public成员都被导出为MBean的操作或属性。

为了对MBean的属性和操作获得更细粒度的控制;

- **通过名称来声明需要暴露或忽略的bean方法**
- **通过为bean增加接口来选择要暴露的方法**
- **通过注解标注bean来标示托管的属性和操作**

MBean服务从何而来

根据以上配置，MB而安Exporter会假设它正在一个应用服务器中(Tomcat)或提供MBean服务器的其他上下文中运行。但是，如果Spring应用程序时独立的应用或运行的容器没有提供MBean服务器，需要在Spring上下文中共配置一个MBean服务器。

- XML配置，context:mbean-server元素可以实现该功能
- Java配置，配置类型为MBeanServerFactoryBean的ban。

MBeanServerFactoryBean会创建一个MBean服务器，将其作为Spring应用上下文中的bean。默认这个bena的ID时mbeanSErver。3

1.1 通过名称暴露方法

MBean信息装配其时限制方法和属性在MBean上暴露的关键。

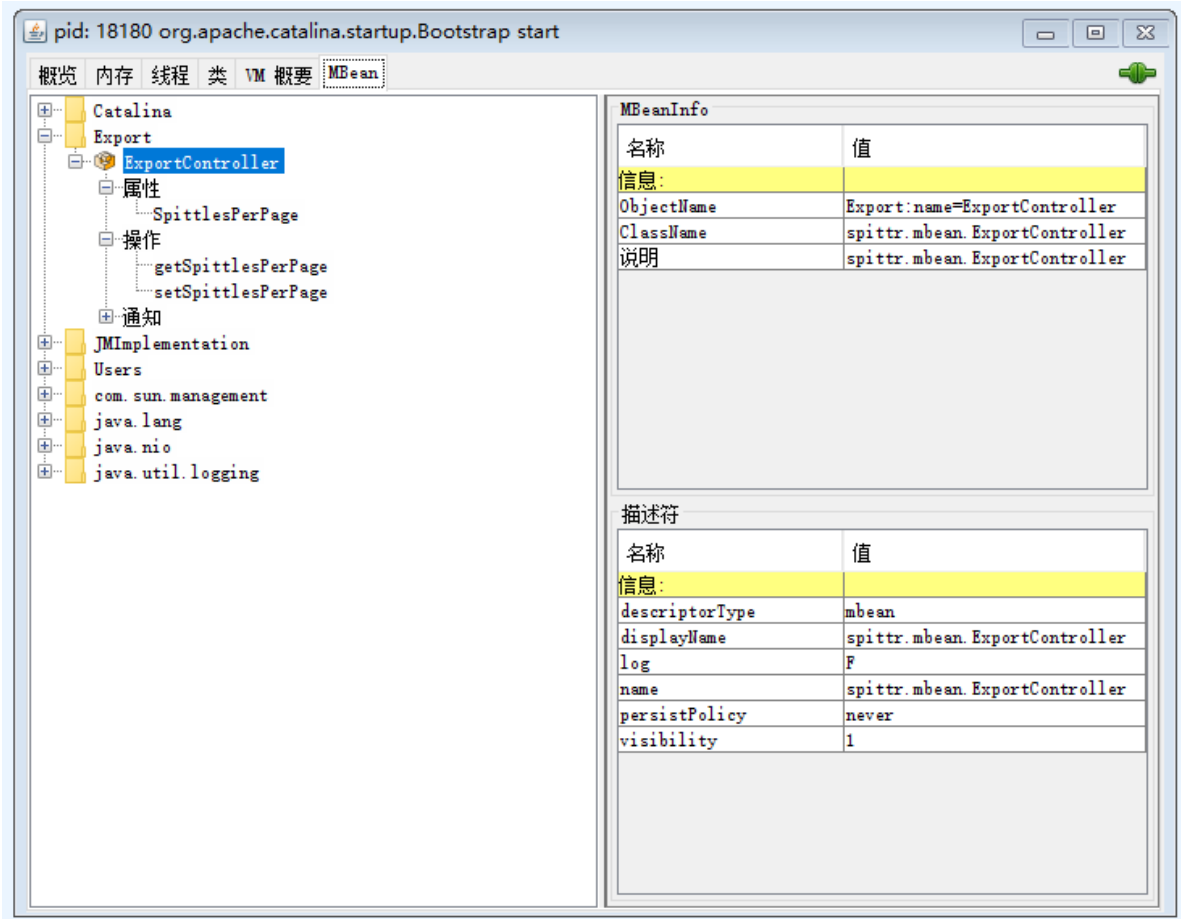
- **MethodNameBaseMBeanInfoAssembler。这个装配器指定了需要暴露为MBean操作的方法名称列表。**

```
@Bean
public MethodNameBasedMBeanInfoAssembler assembler()
{
    MethodNameBasedMBeanInfoAssembler assembler = new
MethodNameBasedMBeanInfoAssembler();
    assembler.setManagedMethods(new String[]
{"getSpittlesPerPage","setSpittlesPerPage"});
    return assembler;
}
```

setManagedMethods指定了哪些方法将暴露为MBean的操作。

```
@Bean
public MBeanExporter mBeanExporter(ExportController exportController,
MBeanInfoAssembler assembler) {
    MBeanExporter exporter = new MBeanExporter();
    Map<String, Object> beans = new HashMap<>(1);
    beans.put("Export:name=ExportController", exportController);
    exporter.setBeans(beans);
    exporter.setAssembler(assembler;
    return exporter;
}
```

为了让这个装配其生效，需要将其装配进MBeanExporter中



reset方法并不会暴露为MBean的托管操作。

- **MethodExclusionMBeanInfoAssembler。这个MBean信息装配器是**
  MethodNameBaseMBeanInfoAssembler的反操作。指定了哪些方法不需要暴露为MBean托管操作的方法名称列表。

```
@Bean
public MethodExclusionMBeanInfoAssembler exclusionMBeanInfoAssembler()
{
    //设置忽略的方法
    MethodExclusionMBeanInfoAssembler assembler = new
MethodExclusionMBeanInfoAssembler();
    assembler.setIgnoredMethods(new String[]{"reset"});
}
```

当导出多个MBean时候，基于方法名称的方式并不能很好满足此场景。

1.2 使用接口定义MBean的操作和属性

InterfaceBasedMBeanInfoAssembler，可以通过使用接口来选择bean的哪些方法需要暴露为MBean的托管操作。

```
public interface ExportManagedOpertions {
    int getSpittlesPerPage();

    void setSpittlesPerPage(int spittlesPerPage);
}
```

```
@Bean
public InterfaceBasedMBeanInfoAssembler interfaceBasedMBeanInfoAssembler() {
    InterfaceBasedMBeanInfoAssembler assembler = new
InterfaceBasedMBeanInfoAssembler();
    assembler.setManagedInterfaces(ExportManagedOpertions.class);
    return assembler;
}
```

ExportController并没有显式实现ExportManagedOpertions接口。这个接口这是为了标示导出的内容，但不需要再代码中直接实现该接口。最好是实现这个接口，从而在MBean和实现类之间保持一个一致的协议

如果通过接口来选择MBean操作，可以把很多方法放在少量的接口中，从而确保InterfaceBasedMBeanInfoAssembler的配置尽量简洁。这些操作必须在某处声明，无论是在Spring配置中还是在某个接口中。此外，托管操作的声明是一种重复----在看i额口中或Spring上下文中生ing的方法名称与实现中所声明的方法名称存在重复。之所以存在这种冯弘福，仅仅是为了满足MBeanExporter的需要而产生的。

1.3 使用注解去当的MBean

MetadataMBeanInfoAssembler，这种装配器可以是哟共注解标识哪些bean的方法需要暴露为MBean的托管操作和属性。

使用springcontext配置命名空间中的context:mbean-export元素，该元素装配了MBean导出器以及为了在Spring中启用注解驱动的MBean所需要的装配器。需要做的局势用它来替换MBeanExporter bean；

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

现在要把任意一个Spring bean转换为MBean，需要做的仅仅是**使用@ManagedResource注解标注bean并使用@ManagedOperation或@ManagedAttribure注解标注的bean的方法**

```
@ManagedResource(objectName = "Export:name=ExportNameController")
public class ExportNameController {

    private static final String MAX_LONG_AS_STRING = "9223372036854775807";

    public static final int DEFAULT_SPITTLES_PER_PAGE = 25;

    public int spittlesPerPage = DEFAULT_SPITTLES_PER_PAGE;

    @ManagedAttribute
    public int getSpittlesPerPage() {
        return spittlesPerPage;
    }

    @ManagedAttribute
    public void setSpittlesPerPage(int spittlesPerPage) {
        this.spittlesPerPage = spittlesPerPage;
    }
}
```

也可以使用@ManagedOperation替换@ManagedAttribute。

这回将方法暴露为MBean的托管操作，但是并不会把spittlesPerPage属性暴露为MBean的托管属性。因为，在暴露MBean功能时，使用@ManagedOperation注解方法是严格显式方法的，并不会把它作为JavaBean的存取方法。因此，@ManagedOperation可以用来把bean的方法暴露为MBean托管操作，而使用@ManagedAttribute可以把bean的属性暴露为MBean托管属性

1.4 处理MBean冲突

MBean服务器中如果名字冲突时，MBeanExporter会抛出InstanceAlreadyExistException异常。可以通过MBeanExporter的setRegistrationPolicy属性或者context:mbean-export的registration属性指定冲突处理机制来改变默认行为：

- FAIL_ON_EXISTING 如果已存在相同名字的MBean则失败
- IGNORE_EXISTING忽略冲突，同时不注册新的MBean
- REPLACE_EXISTING用新的MBean覆盖已存在的MBean

2 远程MBean

JSR-160：Java管理扩展远程访问API规范。定义了JMX远程访问标准，该标准至少需要绑定RMI和可选的JMX消息协议(JMX Messaging Protocol, JMXMP)。

2.1 暴露远程MBean

通过配置Spring的ConnectorServerFactoryBean:

```
@Bean
public ConnectorServerFactoryBean connectorServerFactoryBean() {
    return new ConnectorServerFactoryBean();
}
```

ConnectorServerFactoryBean会创建和启动JSR-160 JMXConnectorServer。默认情况下，服务器使用JMXMP协议并监听端口9875，因此，将绑定“service:jmx:jmxmp://localhost:9875”。但导出MBean的可选方案并不局限于JMXMP。

根据不同JMX实现，有多种远程访问协议可供选择，包括原车给方法调用(RMI),SOAP,Hessian/Burlap和IIOP(Internet InterORB Protocol)。**绑定不同的远程访问协议，仅需要设置ConnectorServerFactoryBean的serviceUrl属性**

```
@Bean
public ConnectorServerFactoryBean connectorServerFactoryBean() {
    ConnectorServerFactoryBean csfb = new ConnectorServerFactoryBean();

 csfb.setServiceUrl("service:jmx:rmi://localhost/jndi/rmi://localhost:1099/spitter");
    return csfb;
}
```

ConnectorServerFactoryBean绑定到一个RMI注册表，该注册表监听1099.同时需要一个RMI注册表运行时，并监听该端口。

```
@Bean
public RmiRegistryFactoryBean rmiRegistryFactoryBean() {
    RmiRegistryFactoryBean factoryBean = new RmiRegistryFactoryBean();
    factoryBean.setPort(1099);
    return factoryBean;
}
```

2.2 访问远程MBean

```
@Bean
public MBeanServerConnectionFactoryBean mBeanServerConnectionFactoryBean()
throws MalformedURLException {
    MBeanServerConnectionFactoryBean factoryBean = new
MBeanServerConnectionFactoryBean();

 factoryBean.setServiceUrl("service:jmx:rmi://localhost/jndi/rmi://localhost:109
9/spitter");
    return factoryBean;
}
```

2.2.1 访问MBean属性

MBeanServerConnectionFactoryBean所涩会给你成的MBeanServerConnection实际上是作为远程MBean服务器的本地代理。它能够以MBeanServerConnectionFactoryBean的形式注入到其他bean属性中

```
public void getAndSetAttribute(MBeanServerConnection connection){
        connection.getAttribute(new
ObjectName("Export:name=ExportController"),"spittlesPerPage");
        connection.setAttribute(new ObjectName("Export:name=ExportController"),
new Attribute("spittlesPerPage",10));
    }
```

为了访问MBean属性，可以是哟共getAttribute()和setAttribute()方法

2.2.2 调用MBean操作

需要使用invoke()方法。

```
public void invokeMethod() throws MalformedObjectNameException {
    connection.invoke(new ObjectName("Export:name=ExportController"),
"setSpittlesPerPage", new Object[]{100}, new String[]{"int"});
}
```

2.3 代理MBean

MBeanProxyFactoryBean是一个代理工厂bean，可以让让我们直接访问远程的MBean(如同在本地的其他bean一样)

```
@Bean
public MBeanProxyFactoryBean remoteExportControllerMBean(MBeanServerConnection
connection) throws MalformedObjectNameException {
    MBeanProxyFactoryBean proxyFactoryBean = new MBeanProxyFactoryBean();
    proxyFactoryBean.setObjectName("");
    proxyFactoryBean.setServer(connection);
    proxyFactoryBean.setProxyInterface(ExportManagedOpertions.class);
    return proxyFactoryBean;
}
```

setServer属性引用了MBeanServerConnection，通过它实现MBean所有通讯的路由

3 处理通知

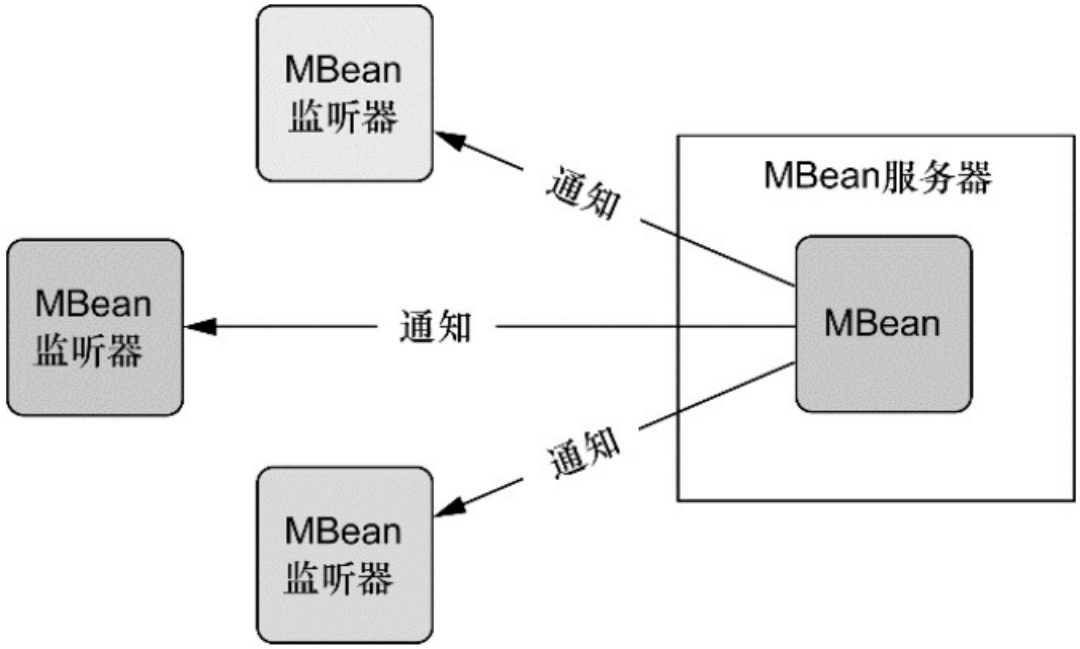


图20.5 JMX通知使MBean与外部世界进行主动通信

Spring通过NotificationPublisherAware接口提供了发送通知的支持。

```
@Component
@ManagedResource("spitter:name=SpittleNotifier")
@ManagedNotification(notificationTypes = "SpittleNotifier.OneMillionSpittles",
name = "TODO")
public class SpittleNotifierImpl implements NotificationPublisherAware {

    private NotificationPublisher notificationPublisher;

    @Override
    public void setNotificationPublisher(NotificationPublisher
notificationPublisher) {
        this.notificationPublisher = notificationPublisher;
    }

    public void millionthSpittlePosted() {
        notificationPublisher.sendNotification(new
Notification("SpittleNotifier.OneMillionSpittles", this, 0));
```

```
    }
}
```

3.1 监听通知

```
class PagingNotification implements NotificationListener {

    @Override
    public void handleNotification(Notification notification, Object handback) {

    }
}
```

```
@Bean
public MBeanExporter mBeanExporter()
{
    MBeanExporter exporter = new MBeanExporter();
    Map<String, PagingNotification> mappings = new HashMap<>();
    mappings.put("Spitter:name=PagingNotificationListener", new
PagingNotification());
    exporter.setNotificationListenerMappings(mappings);
    return exporter;
}
```

setNotificationListenerMappings属性鱼鱼在监听器和监听器所希望监听的MBean之间建立映射