Spring的资源管理

本章目标

- 1、了解资源表示形式
- 2、了解资源访问

Spring资源管理

- Spring的资源管理在JDK的基础功能上进行了强大的扩展：
  - 隐藏底层实现
  - 新增资源存在判断、资源操作权限相关的功能，相对于java.net.URL资源不存在则设置为null更友好
  - 支持通配符来获取资源

Spring资源管理

- Spring管理如下资源：
  - UrlResource
  - ClassPathResource
  - FileSystemResource
  - ServletContextResource
  - InputStreamResource
  - ByteArrayResource

Spring资源管理

- 资源协议与路径：

协议	例子	说明
classpath:	classpath:res/extend.properties	从当前jvm的classpath根路径开始获取资源。
file:	file:///tmp/myfile.data	从操作系统(文件路径)的路径获取资源。
http(s):	http(s)://www.chinasofti.com/	从互联网获取资源。
(无标记)	/data/extend.data	根据应用上下文获取资源。

classpath:、file:、http(s):这三个协议都很明确的指明了获取资源的路径，但是没有声明协议的情况就比较特殊，需要根据上下文来判定适用的路径。

Spring资源管理

- Spring使用Resouce接口，访问底层资源：

```
public interface Resource extends InputStreamSource {

    boolean exists();

    boolean isOpen();

    URL getURL() throws IOException;

    File getFile() throws IOException;

    Resource createRelative(String relativePath) throws IOException;

    String getFilename();

    String getDescription();

}
```

```
public interface InputStreamSource {

    InputStream getInputStream() throws IOException;

}
```

资源访问

- 使用ResourceLoader接口加载资源：

```
public interface ResourceLoader {

    Resource getResource(String location);

}
```

IOC容器实现了ResourceLoader接口，因此可以随时加载资源

```
public interface ApplicationContext extends EnvironmentCapable, ListableBeanFactory, HierarchicalBeanFactory,
        MessageSource, ApplicationEventPublisher, ResourcePatternResolver {
```

```
public interface ResourcePatternResolver extends ResourceLoader {
```

```
Resource template = ctx.getResource("some/resource/path/myTemplate.txt");
Resource template = ctx.getResource("classpath:some/resource/path/myTemplate.txt");
Resource template = ctx.getResource("file:///some/resource/path/myTemplate.txt");
Resource template = ctx.getResource("http://myhost.com/resource/path/myTemplate.txt");
```

资源访问

- 从配置中获取资源

不同的IOC环境，分别使用ClassPathResource, FileSystemResource, 或 ServletContextResource来接收资源

```
<bean id="myBean" class="...">
  <property name="template" value="some/resource/path/myTemplate.txt"/>
</bean>
```

如需强制指明类型，也可以使用前缀

```
<property name="template" value="classpath:some/resource/path/myTemplate.txt">
<property name="template" value="file:///some/resource/path/myTemplate.txt"/>
```

应用上下文与资源

- 应用上下文与资源
  - 从classpath中加载资源

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("conf/appContext.xml");
ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");
```

  - 从当前程序运行的工作目录，用相对路径加载资源
  - 使用指定前缀，从classpath中加载资源
  - 使用通配符 * 加载资源

MyBatis资源参考

- MyBaits配置映射文件时，使用的资源串与Spring类似：

```
<!-- Using classpath relative resources -->
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>
```

```
<!-- Register all interfaces in a package as mappers -->
<mappers>
  <package name="org.mybatis.builder"/>
</mappers>
```

```
<!-- Using url fully qualified paths -->
<mappers>
  <mapper url="file:///var/mappers/AuthorMapper.xml"/>
  <mapper url="file:///var/mappers/BlogMapper.xml"/>
  <mapper url="file:///var/mappers/PostMapper.xml"/>
</mappers>
```

```
<!-- Using mapper interface classes -->
<mappers>
  <mapper class="org.mybatis.builder.AuthorMapper"/>
  <mapper class="org.mybatis.builder.BlogMapper"/>
  <mapper class="org.mybatis.builder.PostMapper"/>
</mappers>
```