* **我们的代码如何读取属性文件**
* 在纯java代码配置方式中，推荐用注解方式来读取properties文件最方便，使用@value和@PropertySource。

示例：

|  |
| --- |
| @Configuration//通过该注解来表明该类是一个Spring的配置，相当于一个xml文件  @PropertySource(value= {"classpath:jdbc.properties"})//指定属性文件  **public class** SpringConfig {  @Value("${jdbc.url}")//从属性文件中读这个量  **private** String jdbcUrl;  @Bean// 通过该注解来表明是一个Bean对象，相当于xml中的<bean>  **public** UserDAO getUserDAO(){  **return** **new** UserDAO(jdbcUrl); // 直接new对象做演示//此bean就取得了这个量  }  } |

* 多个properties文件的引入的写法

|  |
| --- |
| @PropertySource(value= {"classpath:jdbc.properties","…","…","…"}) |

* 解决如果找不到值怎么办

|  |
| --- |
| @PropertySource(value= {"classpath:jdbc.properties",ignoreResourceNotFound=true) |

* 如果把@PropertySource放在应用入口类修饰，这样应用一开启就载入properties文件了。

如下示例

|  |
| --- |
| //应用入口类  @SpringBootApplication  @RestController  @PropertySource(value={"classpath:names.properties"})  **public** **class** DemoApplication {  @RequestMapping("/")  **public** String home() {  **return** "hello," + name;  }  @Value("${m.n}")  String name;    **public** **static** **void** main(String[] args) {  SpringApplication.*run*(DemoApplication.**class**, args);  }  } |
| //name.properties文件  m.n=tom  x.y=kitty |

* **数据库连接的配置读取properties文件的写法**
* 这里展一下数据源的配置，它从properties文件提取数据的写法。该写法也能用于将来的集成框架中。

|  |
| --- |
| //properties文件  db.driver=com.mysql.jdbc.Driver  db.url=jdbc:mysql://localhost:3306/abc?characterEncoding=utf-8  db.username=root  db.password=123456 |
| //以前的xml方式的写法(如果你用其它的数据源,该文件写法会有些不同)  <bean id=*"dataSource"* class=*"org.apache.ibatis.datasource.pooled.PooledDataSource"*>  <property name=*"driver"* value=*"${driver}"* />  <property name=*"url"* value=*"${url}"* />  <property name=*"username"* value=*"${username}"* />  <property name=*"password"* value=*"${password}"* />  </bean> |
| //现在的用java配置文件的方式(如果你用其它的数据源,该文件写法会有些不同)  @Configuration  @PropertySource(value={"classpath:jdbc.properties"})  **public** **class** SomeConfig {    @Value("${db.driver}")  String driver;  @Value("${db.url}")  String url;  @Value("${db.username}")  String username;  @Value("${db.password}")  String password;    @Bean  PooledDataSource pooledDataSource(){//方法名等价于xml方式的id值  PooledDataSource pooledDataSource = **new** PooledDataSource();  pooledDataSource.setDriver(driver);  pooledDataSource.setUrl(url);  pooledDataSource.setUsername(username);  pooledDataSource.setPassword(password);    //打印测试下各分量//还可以再打印下getUrl()、getUsername()、getPassword();  System.***out***.println(pooledDataSource.getDriver());    **return** pooledDataSource;  }    } |
| //主程序，测试一下纯java代码配置  **public** **class** Main {  **public** **static** **void** main(String arg[]) {  // 实例化容器  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(SomeConfig.**class**);  // 获取Bean对象  PooledDataSource con = context.getBean(PooledDataSource.**class**);  **try** {  con.getConnection();//调用一下获取连接这样才能真正测试正确性  } **catch** (SQLException e) {  e.printStackTrace();  }    context.destroy();  context.close();  }  } |

|  |  |  |
| --- | --- | --- |
| 注意：  参见<https://blog.csdn.net/q1054261752/article/details/52032699> （此人也遇到了同样的问题）  上面的   |  | | --- | | @Value("${db.username}") 和 db.username=root |   不要写成了   |  | | --- | | @Value("${username}") 和 username=root |   因为后者会居然提取到windows的用户Administrator。 |

总结：

但是，从这个例子可以看出，纯java代码配置方案并不一定比xml方式更简洁，只是看起来更不容易出错而已。（因为调用setXXX()方法时IDE能够提示）。