13.3. AutoConfig的使用 —— 开发者指南

13.3.1. 建立AutoConfig目录结构

和Maven Filtering不同的是，AutoConfig是针对目标文件的配置工具。因此AutoConfig关心的目录结构是*目标文件的目录结构*。不同的build工具，创建同一目标目录结构所需要的源文件的目录结构会各有不同。本文仅以maven标准目录结构为例，来说明源文件的目录结构编排。

13.3.1.1. WAR包的目录结构

这里所说的war包，可以是一个以zip方式打包的文件，也可以是一个展开的目录。下面以maven标准目录为例，说明项目源文件和目标文件的目录结构的对比：

例 13.3. WAR包的源文件和目标文件目录结构

```
war-project（源目录结构）                    -> war-project.war（目标目录结构）
  |  pom.xml
  |
  └─src
      └─main
          ├─java
          ├─resources                        -> /WEB-INF/classes
          |      file1.xml                          file1.xml
          |      file2.xml                          file2.xml
          |
          └─webapp                           -> /
              ├─META-INF                     -> /META-INF
              |  └─autoconf                  -> /META-INF/autoconf ❶
              |       auto-config.xml                 auto-config.xml ❷
              |
              └─WEB-INF                      -> /WEB-INF
                   web.xml                           web.xml
                   file3.xml                         file3.xml
```

❶ /META-INF/autoconf目录用来存放AutoConfig的描述文件，以及可选的模板文件。

❷ auto-config.xml是用来指导AutoConfig行为的关键描述文件。

创建war包的AutoConfig机制，关键在于创建war目标文件中的/META-INF/autoconf/auto-config.xml描述文件。该描述文件对应的maven项目源文件为：/src/main/webapp/META-INF/autoconf/auto-config.xml。

13.3.1.2. JAR包的目录结构

这里所说的jar包，可以是一个以zip方式打包的文件，也可以是一个展开的目录。下面以maven标准目录为例，说明项目源文件和目标文件的目录结构的对比：

例 13.4. JAR包的源文件和目标文件目录结构

```
jar-project（源目录结构）                    -> jar-project.jar（目标目录结构）
  |   pom.xml
  |
  └─src
      └─main
          ├─java
          └─resources                        -> /
              |   file1.xml                          file1.xml
              |   file2.xml                          file2.xml
              |
              └─META-INF                     -> /META-INF
                  └─autoconf                 -> /META-INF/autoconf ❶
                        auto-config.xml              auto-config.xml ❷
```

❶ /META-INF/autoconf目录用来存放AutoConfig的描述文件，以及可选的模板文件。

❷ auto-config.xml是用来指导AutoConfig行为的关键描述文件。

创建jar包的AutoConfig机制，关键在于创建jar目标文件中的/META-INF/autoconf/auto-config.xml描述文件。该描述文件对应的maven项目源文件为：/src/main/resources/META-INF/autoconf/auto-config.xml。

13.3.1.3. 普通目录

AutoConfig也支持对普通文件目录进行配置。

例 13.5. 对普通的目录执行AutoConfig

```
directory
  |   file1.xml
  |   file2.xml
  |
  └─conf ❶
      auto-config.xml ❷
```

❶ 默认情况下，AutoConfig在/conf目录中寻找AutoConfig的描述文件，以及可选的模板文件。

❷ auto-config.xml是用来指导AutoConfig行为的关键描述文件。

13.3.2. 建立auto-config.xml描述文件

AutoConfig系统的核心就是auto-config.xml描述文件。该描述文件中包含两部分内容：

1. 定义properties：properties的名称、描述、默认值、约束条件等信息；

2. 指定包含placeholders的模板文件。

下面是auto-config.xml文件的样子：（以petstore应用为例）

例 13.6. AutoConfig描述文件示例

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <group>

        <property name="petstore.work"
                  description="应用程序的工作目录" /> ❶

        <property name="petstore.loggingRoot"
                  defaultValue="${petstore.work}/logs"
                  description="日志文件目录" /> ❷

        <property name="petstore.upload"
                  defaultValue="${petstore.work}/upload"
                  description="上传文件的目录" /> ❸

        <property name="petstore.loggingLevel"
                  defaultValue="warn"
                  description="日志文件级别"> ❹

            <validator name="choice"
                       choice="trace, debug, info, warn, error" /> ❺

        </property>

    </group>
    <script>
        <generate template="WEB-INF/web.xml" /> ❻
        <generate template="WEB-INF/common/resources.xml" />
    </script>
</config>
```

❶ 定义properties

❷

❸

❹

❺ 定义property的验证规则（可选）

❻ 生成配置文件的指令。

13.3.2.1. 定义properties

定义一个property的完整格式如下：

例 13.7. 定义一个property

```
<property
    name="..."
    [defaultValue="..."]
    [description="..."]
    [required="true|false"]
>
```

```
    <validator name="..." />
    <validator name="..." />
    ...
</property>
```

可用的property参数包括：

表 13.8. 定义property时可用的参数

参数名	说明
name	Property名称。
defaultValue（可选）	默认值。默认值中可包含对其它property的引用，如${petstore.work}/logs。
description（可选）	对字段的描述，这个描述会显示给deployer，这对他理解该property非常重要。
required（可选）	是否“必填”，默认为true。如果deployer未提供必填项的值，就会报错。

13.3.2.2. 定义property的验证规则

目前，有以下几种验证器：

表 13.9. 可用的property验证规则

验证规则	说明
<validator name="boolean" />	Property值必须为true或false。
<validator name="choice" choice="trace, debug, info, warn, error" />	Property值必须为choice所定义的值之一。
<validator name="email" />	Property值必须为合法的email格式。
<validator name="fileExist" [file="WEB-INF/web.xml"] />	Property值必须为某个存在的文件或目录。 如果指定了file，那就意味着property值所指的目录下，必须存在file所指的文件或子目录。

验证规则	说明
`<validator name="hostExist" />`	Property值必须为合法的IP地址，或者可以解析得到的域名。
`<validator name="keyword" />`	Property值必须为字母、数字、下划线的组合。
`<validator name="number" />`	Property值必须为数字的组合。
`<validator name="regexp" regexp="..." [mode="exact|prefix|contain"] />`	Property值必须符合regexp所指的正则表达式。 其中，mode为匹配的方法： • 完全匹配exact • 前缀匹配prefix • 包含contain 如未指定mode，默认mode为contain。
`<validator name="url" [checkHostExist="false"] [protocols="http,https"] [endsWithSlash="true"] />`	Property值必须是合法URL。 假如指定了checkHostExist=true，那么还会检查域名或IP的正确性； 假如指定了protocols，那么URL的协议必须为其中之一； 假如指定了endsWithSlash=true，那么URL必须以/结尾。

13.3.2.3. 生成配置文件的指令

描述文件中，每个<generate>标签指定了一个包含placeholders的配置文件模板，具体格式为：

例 13.8. 生成配置文件的指令

```
<generate
    template="..."
    [destfile="..."]
    [charset="..."]
    [outputCharset="..."]
>
```

下面是参数的说明：

表 13.10. 生成配置文件的指令参数

参数名	说明
template	需要配置的模板名。 模板名为相对路径，相对于当前jar/war/ear包的根目录。
destfile（可选）	目标文件。 如不指定，表示目标文件和模板文件相同。
charset（可选）	模板的字符集编码。 XML文件不需要指定charset，因为AutoConfig可以自动取得XML文件的字符集编码； 对其它文件必须指定charset。
outputCharset（可选）	目标文件的输出字符集编码。 如不指定，表示和模板charset相同。

13.3.3. 建立模板文件

13.3.3.1. 模板文件的位置

定义完auto-config.xml描述文件以后，就可以创建模板了。模板放在哪里呢？举例说明。

例 13.9. 模板文件的位置

假设在一个典型的WEB应用中，你的auto-config.xml中包含指定了如下模板：

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <group>
        ...
    </group>
    <script>
        <generate template="WEB-INF/classes/file1.xml" />
        <generate template="WEB-INF/classes/file2.xml" />
        <generate template="WEB-INF/file3.xml" />
    </script>
</config>
```

那么，你可以把file1.xml、file2.xml、file3.xml放在下面的位置：

```
war-project（源目录结构）                    -> war-project.war（目标目录结构）
  |   pom.xml
  |
  |--src
      |--main
          |--java
          |--resources                      -> /WEB-INF/classes
          |     file1.xml                         file1.xml - 建议放在这里
          |     file2.xml                         file2.xml - 建议放在这里
          |
          |--webapp
              |--META-INF
              |   |--autoconf
              |          |  auto-config.xml
              |          |
              |          |--WEB-INF           -> /WEB-INF
              |              | file3.xml             file3.xml - 也可以放在这里
              |              |
              |              |--classes       -> /WEB-INF/classes
              |                    file1.xml         file1.xml - 也可以放在这里
              |                    file2.xml         file2.xml - 也可以放在这里
              |
              |--WEB-INF                      -> /WEB-INF
                   file3.xml                        file3.xml - 建议放在这里
```

AutoConfig的寻找模板的逻辑是：

- 如果在auto-config.xml所在的目录下发现模板文件，就使用它；
- 否则在包的根目录中查找模板文件；如果两处均未找到，则报错。

13.3.3.2. 模板的写法

书写模板是很简单的事，你只要：

- 把需要配置的点替换成placeholder：“${property.name}”。当然，你得确保property.name被定义在auto-config.xml中。
- 假如模板中包含*不希望被替换的*运行时的placeholder“*$*{...}”，需要更改成“*${D}*{...}”。

例 13.10. 模板示例

```
...
<context-param>
    <param-name>loggingRoot</param-name>
    <param-value>${petstore.loggingRoot}</param-value>
</context-param>
<context-param>
    <param-name>loggingLevel</param-name>
    <param-value>${petstore.loggingLevel}</param-value>
</context-param>
...
${D}{runtime.placeholder}
```

此外，AutoConfig模板其实是由Velocity模板引擎来渲染的。因此，所有的placeholder必须能够通过velocity的语法。

例 13.11. 使用不符合velocity语法的placeholders

例如，下面的placeholder被velocity看作非法：

```
${my.property.2}
```

解决的办法是，改写成如下样式：

```
${my_property_2}
```

13.4. AutoConfig的使用 —— 部署者指南

部署者有两种方法可以使用AutoConfig：

- 在命令行上直接运行。
- 在maven中调用AutoConfig plugin。

13.4.1. 在命令行中使用AutoConfig

13.4.1.1. 取得可执行文件

AutoConfig提供了Windows以及Unix-like（Linux、Mac OS等）等平台上均可使用的native可执行程序。可执行程序文件被发布在Maven repository中。

如果你已经配置好了maven，那么可以让maven来帮你下载目标文件。

例 13.12. 让maven帮忙下载AutoConfig可执行文件

请创建一个临时文件：pom.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <parent>
        <groupId>com.alibaba.citrus.tool</groupId>
        <artifactId>antx-parent</artifactId>
        <version>1.2</version> ❶
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>temp</artifactId>
</project>
```

❶ 文件中的parent pom的版本号（1.2）决定了你要取得的AutoConfig的版本号。

然后在命令行上执行如下命令：

```
mvn dependency:copy
```

这样就取得了两个文件：

- autoconfig-1.2.tgz
- autoexpand-1.2.tgz - AutoExpand是另一个小工具。它是用来展开war、jar、ear包的。关于AutoExpand的详情，请见第 14 章 *AutoExpand工具使用指南*。

你也可以直接去maven repository中手工下载以上两个包：

- http://repo1.maven.org/maven2/com/alibaba/citrus/tool/antx-autoconfig/1.2/antx-autoconfig-1.2.tgz
- http://repo1.maven.org/maven2/com/alibaba/citrus/tool/antx-autoexpand/1.2/antx-autoexpand-1.2.tgz

取得压缩包以后，可以用下面的命令来展开并安装工具。

表 13.11. 展开并安装工具

Unix-like系统	Windows系统
tar zxvf autoconfig-1.2.tgz tar zxvf autoexpand-1.2.tgz cp autoconfig /usr/local/bin cp autoexpand /usr/local/bin	tar zxvf autoconfig-1.2.tgz tar zxvf autoexpand-1.2.tgz copy autoconfig.exe c:\windows\system32 copy autoexpand.exe c:\windows\system32

13.4.1.2. 执行AutoConfig命令

取得可执行文件以后，就可以试用一下：在命令行上输入**autoconfig**。不带参数的**autoconfig**命令会显示出如下帮助信息。

例 13.13. AutoConfig的帮助信息

```
$ autoconfig
Detected system charset encoding: UTF-8
If your can't read the following text, specify correct one like this:
  autoconfig -c mycharset

使用方法：autoconfig [可选参数] [目录名|包文件名]

可选参数：
 -c,--charset                  输入/输出编码字符集
 -d,--include-descriptors
                               包含哪些配置描述文件，例如：conf/auto-config.xml，
可使用*、**、?通配符，如有多项，用逗号分隔
 -D,--exclude-descriptors      排除哪些配置描述文件，可使用*、**、?通配符，如有多
项，用逗号分隔
```

```
 -g,--gui                          图形用户界面（交互模式）
 -h,--help                         显示帮助信息
 -i,--interactive                  交互模式：auto|on|off，默认为auto，无参数表示on
 -I,--non-interactive              非交互模式，相当于--interactive=off
 -n,--shared-props-name            共享的属性文件的名称
 -o,--output                       输出文件名或目录名
 -P,--exclude-packages             排除哪些打包文件，可使用*、**、?通配符，如有多项，用
逗号分隔
 -p,--include-packages
                                   包含哪些打包文件，例如：target/*.war，可使用*、**、?
通配符，如有多项，用逗号分隔
 -s,--shared-props                 共享的属性文件URL列表，以逗号分隔
 -T,--type                         文件类型，例如：war，jar，ear等
 -t,--text                         文本用户界面（交互模式）
 -u,--userprop                     用户属性文件
 -v,--verbose                      显示更多信息

总耗费时间：546毫秒
```

最简单的AutoConfig命令如下：

例 13.14. 最简单的AutoConfig命令

```
autoconfig petstore.war
```

无论petstore.war是一个zip包还是目录，AutoConfig都会正确地生成其中的配置文件。

13.4.2. 在maven中使用AutoConfig

AutoConfig也可以通过maven plugin来执行。

这种方式使用方式，方便了开发者试运行并测试应用程序。开发者可以在build项目的同时，把AutoConfig也配置好。然而对于非开发的应用测试人员、发布应用的系统管理员来说，最好的方法是使用独立可执行的AutoConfig来配置应用的二进制目标文件。

为了使用maven插件，你需要修改项目的pom.xml来设定它。请注意，一般来说，不要在parent pom.xml中设定AutoConfig，因为这个设置会作用在每个子项目上，导致不必要的AutoConfig执行。只在生成最终目标文件的子项目pom.xml中设定AutoConfig就可以了。例如，对于一个web项目，你可以在生成war包的子项目上设置AutoConfig plugin。

例 13.15. 在pom.xml中设定AutoConfig plugin

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    ...
    <properties>
        ...
        <!-- 定义autoconfig的版本，建议将此行写在parent pom.xml中。 -->
        <autoconfig-plugin-version>1.2</autoconfig-plugin-version>
    </properties>
    ...
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>com.alibaba.citrus.tool</groupId>
                <artifactId>autoconfig-maven-plugin</artifactId>
                <version>${autoconfig-plugin-version}</version>
                <configuration>
                    <!-- 要进行AutoConfig的目标文件，默认为
${project.artifact.file}。
                    <dest>${project.artifact.file}</dest>
                    -->
                    <!-- 配置后，是否展开目标文件，默认为false，不展开。
                    <exploding>true</exploding>
                    -->
                    <!-- 展开到指定目录，默认为
${project.build.directory}/${project.build.finalName}。
                    <explodedDirectory>
                        ${project.build.directory}/${project.build.finalName}
                    </explodedDirectory>
                    -->
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>autoconfig</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

这样，每次执行mvn package或者mvn install时，都会激活AutoConfig，对package目标文件进行配置。

想要避免AutoConfig，只需要一个额外的命令行参数：

例 13.16. 避免执行AutoConfig

```
mvn install -Dautoconfig.skip
```

13.4.3. 运行并观察AutoConfig的结果

第一次执行AutoConfig，无论通过何种方式（独立命令行或maven插件），AutoConfig都会提示你修改user properties文件，以提供所需要的properties值。AutoConfig提供了一套基于文本的交互式界面来编辑这些properties。

例 13.17. 交互式编辑properties

```
您的配置文件需要被更新：
```

```
 file:/.../antx.properties

 这个文件包括了您个人的特殊设置，
 包括服务器端口、您的邮件地址等内容。


 如果不更新此文件，可能会导致配置文件的内容不完整。
 您需要现在更新此文件吗? [Yes][No] y
```

当你通过交互式界面填写了所有properties的值，并通过了AutoConfig的验证以后，AutoConfig就开始生成配置文件：

```
即将保存到文件"file:/.../antx.properties"中, 确定? [Yes][No] y

 保存文件 file:/.../antx.properties...
petstore.loggingLevel  = warn
petstore.loggingRoot   = ${petstore.work}/logs
petstore.upload        = ${petstore.work}/upload
petstore.work          = /tmp
 已保存至文件: file:/.../antx.properties
Loading file:/.../antx.properties
<jar:file:/.../Work/my/apps/petstore-webx3/target/petstore.war!/>
    Generating WEB-INF/web.xml [UTF-8] => WEB-INF/web.xml [UTF-8]

<jar:file:/.../Work/my/apps/petstore-webx3/target/petstore.war!/>
    Generating WEB-INF/common/resources.xml [UTF-8] => WEB-
INF/common/resources.xml [UTF-8]

<jar:file:/.../Work/my/apps/petstore-webx3/target/petstore.war!/>
    Generating log file: META-INF/autoconf/auto-config.xml.log

Expanding: /.../Work/my/apps/petstore-webx3/target/petstore.war
       To: /.../Work/my/apps/petstore-webx3/target/petstore
done.
```

假如发现模板中某个placeholder，并未在auto-config.xml中定义，就会出现以下错误：

```
ERROR - Undefined placeholders found in template:
- Template:   META-INF/autoconf/WEB-INF/web.xml
- Descriptor: META-INF/autoconf/auto-config.xml
- Base URL:   file:/.../Work/my/apps/petstore-webx3/target/petstore/
----------------------------------------------------------------
-> petstore.loggingRoot
----------------------------------------------------------------
```

出现错误以后，Maven会报错，并停止build过程。假如你不希望maven停止，可以用下面的命令来执行maven：

例 13.18. 避免maven因为placeholder未定义而停止

```
mvn ... -Dautoconfig.strict=false
```

AutoConfig会生成一个日志文件，就在auto-config.xml所在的目录下，名字为：auto-config.xml.log。

例 13.19. AutoConfig所生成的日志文件

```
Last Configured at: Fri Jun 18 13:54:22 CST 2010

Base URL: file:/.../Work/my/apps/petstore-webx3/target/petstore/
Descriptor: META-INF/autoconf/auto-config.xml

Generating META-INF/autoconf/WEB-INF/web.xml [UTF-8] => WEB-INF/web.xml [UTF-8]
Generating META-INF/autoconf/WEB-INF/common/resources.xml [UTF-8] => WEB-INF/common/resources.xml [UTF-8]
```

最后，让我们查看一下AutoConfig所生成的文件，其中所有的placeholders应当被替换成你所提供的值了。

例 13.20. AutoConfig生成的结果

```
...
<context-param>
    <param-name>loggingRoot</param-name>
    <param-value>/tmp/logs</param-value>
</context-param>
<context-param>
    <param-name>loggingLevel</param-name>
    <param-value>warn</param-value>
</context-param>
...
${runtime.placeholder}
```

13.4.4. 共享properties文件

当需要配置的内容越来越多时，即使使用AutoConfig这样的机制，也会变得不胜其烦。

假如你的项目包含了好几个模块，而你只负责其中的一个模块。一般来说，你对其它模块的配置是什么并不清楚，事实上你也懒得去关心。但是你为了运行这个项目，你不得不去配置这些模块。假如模块A就是一个你不想关心的模块，但为了运行它，你需要告诉模块A一些参数：数据库连接的参数、域名、端口、文件目录、搜索引擎……可你并不清楚这些参数应该取什么值。

好在AutoConfig提供了一个共享properties文件的方法。

13.4.4.1. 共享的properties文件

你可以创建一系列文件：module-a-db.properites，module-a-searchengine.properties等等。每个文件中都包含了某个运行环境中的关于module A模块的配置参数。

现在，你可以不关心module A了！你只要使用下面的命令：

例 13.21. 指定共享的properties文件

```
autoconfig -s module-a-db.properties,module-a-searchengine.properties❶ ……
```

❶ -s参数代表“共享的properties文件”。

同时，你的antx.properties也被简化了，因为这里只会保存你定义的配置项，而不会包含共享的配置项。

13.4.4.2. 共享整个目录

假如共享的文件很多的话，AutoConfig还有一个贴心的功能，你可以把这些文件按目录来组织：

例 13.22. 按目录组织要被共享的properties文件

```
shared-properties/
  ├─test/                                    // 测试环境的共享配置
  │    module-a-db.properties
  │    module-a-searchengine.properties
  │    module-b.properties
  └─prod/                                    // 生产环境的共享配置
       module-a-db.properties
       module-a-searchengine.properties
       module-b.properties
```

然后，你可以直接在AutoConfig中引用目录：

例 13.23. 共享指定目录中的所有properties文件

```
autoconfig -s shared-propertes/test/ ……
```

AutoConfig就会为你装载这个目录下的所有共享配置文件。（注意，*目录必须以斜杠“/”结尾*）

13.4.4.3. 将共享目录放在http、https或ssh服务器上

AutoConfig还支持从http、https或ssh服务器上取得共享配置文件，只需要将前面例子中的文件名改成http或ssh的URI就可以了：

例 13.24. 共享远程服务器上的properties文件或目录

```
autoconfig -s http://share.alibaba.com/shared-propertes/test/❶ ……
autoconfig -s http://myname@share.alibaba.com/shared-propertes/test/❷ ……
autoconfig -s https://share.alibaba.com/shared-propertes/test/❸ ……
autoconfig -s https://myname@share.alibaba.com/shared-propertes/test/❹ ……
autoconfig -s ssh://myname@share.alibaba.com/shared-propertes/test/❺ ……
```

❶ 共享远程http服务器上的properties文件。

❷ 共享远程http服务器上的properties文件，指定登录用户名。

❸ 共享远程https服务器上的properties文件。

❹ 共享远程https服务器上的properties文件，指定登录用户名。

❺ 共享远程ssh服务器上的properties文件，必须指定用户名。

由于Subversion、Git服务器是支持HTTP/HTTPS协议的，因此将properties文件存放在Subversion或Git服务器上，也是一个极好的办法。由于采用了Subversion或Git，properties文件的版本管理问题也被一举解决了。

需要注意的是，访问http和ssh有可能需要验证用户和密码。当需要验证时，AutoConfig会提示你输入用户名和密码。输入以后，密码将被保存在*$HOME/passwd.autoconfig*文件中，以后就不需要重复提问了。

13.4.4.4. 在多种配置项中切换

当你使用前文所述的autoconfig -s命令来生成antx.properties文件时，你会发现antx.properties中增加了几行特别的内容：

例 13.25. 包含共享文件、目录信息的antx.properties文件

```
antx.properties.default  = http://share.alibaba.com/shared-propertes/test/
```

如果你在-s参数中指定了多项共享properties文件或目录，那么antx.properties中将会这样：

```
antx.properties.default.1  = http://share.alibaba.com/shared-propertes/test/
antx.properties.default.2  = file:/shared-properties/test/my-1.properites
antx.properties.default.3  = file:/shared-properties/test/my-2.properites
```

事实上，AutoConfig还支持多组共享配置，请试用下面的命令：

例 13.26. 使用多组共享配置

```
autoconfig -s http://share.alibaba.com/shared-propertes/test/ -n test❶ ……
```

❶ 为当前共享配置定义一个名字，以后可以用这个名字来简化命令。

这时，antx.properties就会是这个样子：

```
antx.properties = test
antx.properties.test = http://share.alibaba.com/shared-propertes/test/
```

再执行：

```
autoconfig -s http://share.alibaba.com/shared-propertes/prod/ -n prod ……
```

antx.properties就会变成这个样子：

```
antx.properties = prod
antx.properties.test = http://share.alibaba.com/shared-propertes/test/
antx.properties.prod = http://share.alibaba.com/shared-propertes/prod/
```

以后再执行，就不需要再指定-s参数了，只需用-n参数选择一组共享properties文件即可。例如：

```
autoconfig -n prod❶ ……                    // 使用prod生产环境的参数
autoconfig -n test❷ ……                    // 使用test测试环境的参数
autoconfig ❸ ……                           // 不指定，则使用最近一次所选择的共享文件
```

13.4.5. AutoConfig常用命令

下面罗列了AutoConfig的常用的命令及参数：

13.4.5.1. 指定交互式界面的charset

一般不需要特别指定charset，除非AutoConfig自动识别系统编码出错，导致显示乱码。

运行AutoConfig独立可执行程序	`autoconfig ... -c GBK`
运行AutoConfig maven插件	`mvn ... -Dautoconfig.charset=GBK`

13.4.5.2. 指定交互模式

默认情况下，交互模式为自动（auto）。仅当user properties中的值不满足auto-config.xml中的定义时，才会交互式地引导用户提供properties值。

但你可以强制打开交互模式：

运行AutoConfig独立可执行程序	`autoconfig ... -i` `autoconfig ... -i on`
运行AutoConfig maven插件	`mvn ... -Dautoconfig.interactive` `mvn ... -Dautoconfig.interactive=true`

或强制关闭交互模式：

运行AutoConfig独立可执行程序	`autoconfig ... -I` `autoconfig ... -i off`
运行AutoConfig maven插件	`mvn ... -Dautoconfig.interactive=false`

13.4.5.3. 指定user properties

默认情况下，AutoConfig会按下列顺序查找user properties：

1. 当前目录/antx.properties

2. 当前用户HOME目录/antx.properties

但你可以指定一个自己的properties文件，用下面的命令：

运行AutoConfig独立可执行程序	`autoconfig ... -u my.props`
运行AutoConfig maven插件	`mvn ... -Dautoconfig.userProperties=my.props`

13.4.5.4. 显示详细的信息

默认情况下，AutoConfig只输出重要的信息，但有时你想了解更多内部的情况，只需要用下面的命令：

运行AutoConfig独立可执行程序	`autoconfig ... -v`
运行AutoConfig maven插件	不适用

13.4.5.5. 指定输出文件

默认情况下，AutoConfig所生成的配置文件以及日志信息会直接输出到当前包文件或目录中。例如以下命令会改变petstore.war的内容：

```
autoconfig petstore.war
```

但你可以指定另一个输出文件或目录，这样，原来的文件或目录就不会被修改：

运行AutoConfig独立可执行程序	`autoconfig petstore.war -o petstore-configured.war`
运行AutoConfig maven插件	不适用

13.4.5.6. 避免执行AutoConfig

将AutoConfig和maven package phase绑定以后，每次build都会激活AutoConfig。假如你想跳过这一步，只需要下面的命令：

运行AutoConfig独立可执行程序	不适用
运行AutoConfig maven插件	`mvn ... -Dautoconfig.skip`

13.4.5.7. 避免中断maven build

默认情况下，假如发现有未定义的placeholders，AutoConfig会报错并中止maven的执行。假如你不想中断maven build，可以这样做：

运行AutoConfig独立可执行程序	不适用
运行AutoConfig maven插件	`mvn ... -Dautoconfig.strict=false`