

# 一、JNDI介绍

## 1、什么是JNDI

---

JNDI (Java Naming and Directory Interface, Java命名和目录接口) 是一种标准的Java命名系统接口, JNDI提供统一的客户端API, 通过不同的访问提供者接口JNDI服务供应接口(SPI)的实现, 由管理者将JNDI API映射为特定的命名服务和目录系统, 使得Java应用程序可以和这些命名服务和目录服务之间进行交互。目录服务是命名服务的一种自然扩展。

JNDI可以访问的目录及服务, 比如: DNS、LDAP、CORBA对象服务、RMI等等。

简单理解: 在前面一节我们学到了RMI, 比如RMI对外提供了服务, 那么JNDI可以通过相关API可以链接处理这些服务。

## 2、JNDI的五个包

---

在JNDI中提供了五个作用不同的包。

- `javax.naming`
- `javax.naming.directory`
- `javax.naming.ldap`
- `javax.naming.event`
- `javax.naming.spi`

### 2.1、`javax.naming`

它包含了命名服务的类和接口。比如其中定义了Context接口, 可以用于查找、绑定/解除绑定、重命名对象以及创建和销毁子上下文等操作。这个也是我们比较关注的一个包。

- 查找

最常用的操作是 `lookup()`。你向`lookup()`提供你想要查找的对象的名称, 它返回与该名称绑定的对象。

- 绑定

`listBindings()` 返回一个名字到对象的绑定的枚举。绑定是一个元组, 包含绑定对象的名称、对象的类的名称和对象本身。

- 列表

`list()` 与 `listBindings()` 类似, 只是它返回一个包含对象名称和对象类名称的名称枚举。`list()` 对于诸如浏览器等想要发现上下文中绑定的对象的信息但又不需要所有实际对象的应用程序来说非常有用。

- 引用

在一个实际的名称服务中，有些对象可能无法直接存储在系统内，这时它们便以引用的形式进行存储。

### 2.1.1、InitialContext类

构造方法：

```
InitialContext()  
    构建一个初始上下文。  
InitialContext(boolean lazy)  
    构造一个初始上下文，并选择不初始化它。  
InitialContext(Hashtable<?, ?> environment)  
    使用提供的环境构建初始上下文。
```

常用方法：

```
bind(Name name, Object obj) 将名称绑定到对象。  
list(String name) 枚举在命名上下文中绑定的名称以及绑定到它们的对象的类名。  
lookup(String name) 检索命名对象。  
rebind(String name, Object obj) 将名称绑定到对象，覆盖任何现有绑定。  
unbind(String name) 取消绑定命名对象。
```

### 2.1.2、Reference类

构造方法：

```
Reference(String className)  
    为类名为“className”的对象构造一个新的引用。  
Reference(String className, RefAddr addr)  
    为类名为“className”的对象和地址构造一个新引用。  
Reference(String className, RefAddr addr, String factory, String  
factoryLocation)  
    为类名为“className”的对象，对象工厂的类名和位置以及对象的地址构造一个新引用。  
  
Reference(String className, String factory, String factoryLocation)  
    为类名为“className”的对象以及对象工厂的类名和位置构造一个新引用。
```

常用方法：

```
void add(int posn, RefAddr addr)  
    将地址添加到索引posn的地址列表中。  
void add(RefAddr addr)  
    将地址添加到地址列表的末尾。  
void clear()  
    从此引用中删除所有地址。  
RefAddr get(int posn)  
    检索索引posn上的地址。  
RefAddr get(String addrType)  
    检索地址类型为“addrType”的第一个地址。  
Enumeration<RefAddr> getAll()  
    检索本参考文献中地址的列举。  
String getClassName()  
    检索引用引用的对象的类名。  
String getFactoryClassLocation()  
    检索此引用引用的对象的工厂位置。  
String getFactoryClassName()  
    检索此引用引用对象的工厂的类名。  
Object remove(int posn)
```

从地址列表中删除索引`posn`上的地址。

`int size()`

检索此引用中的地址数。

`String toString()`

生成此引用的字符串表示形式。

官方详细介绍：

<https://docs.oracle.com/javase/tutorial/jndi/overview/naming.html>

## 2.2、`javax.naming.directory`

继承了`javax.naming`，提供了除命名服务外访问目录服务的功能。

官方详细介绍：

<https://docs.oracle.com/javase/tutorial/jndi/overview/dir.html>

## 2.3、`javax.naming.ldap`

继承了`javax.naming`，提供了访问LDAP的能力。

官方详细介绍：

<https://docs.oracle.com/javase/tutorial/jndi/overview/dir.html>

## 2.4、`javax.naming.event`

包含了用于支持命名和目录服务中的事件通知的类和接口。

官方详细介绍：

<https://docs.oracle.com/javase/tutorial/jndi/overview/event.html>

## 2.5、`javax.naming.spi`

允许动态插入不同实现，为不同命名目录服务供应商的开发人员提供开发和实现的途径，以便应用程序通过JNDI可以访问相关服务。

官方详细介绍：

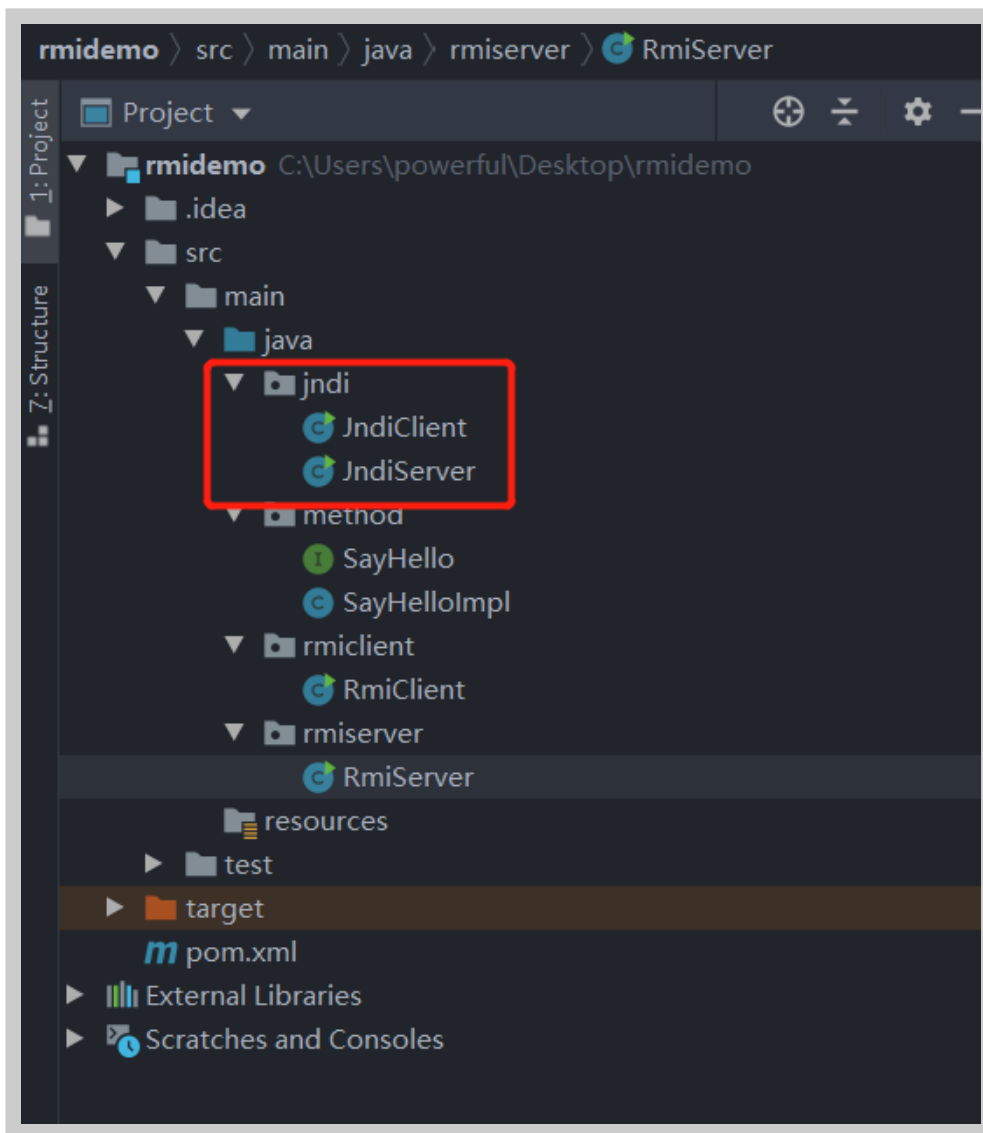
<https://docs.oracle.com/javase/tutorial/jndi/overview/event.html>

# 二、JNDI操作RMI

## 1、创建工程

我们在RMI基础的代码中进行一些修改。

打开`rmidemo`项目，在 `src.main.java` 下新建一个名为 `jndi` 的目录，并在该目录下分别新建两个名为 `JndiClient`，`JndiServer` 的Java Class。如下图所示：



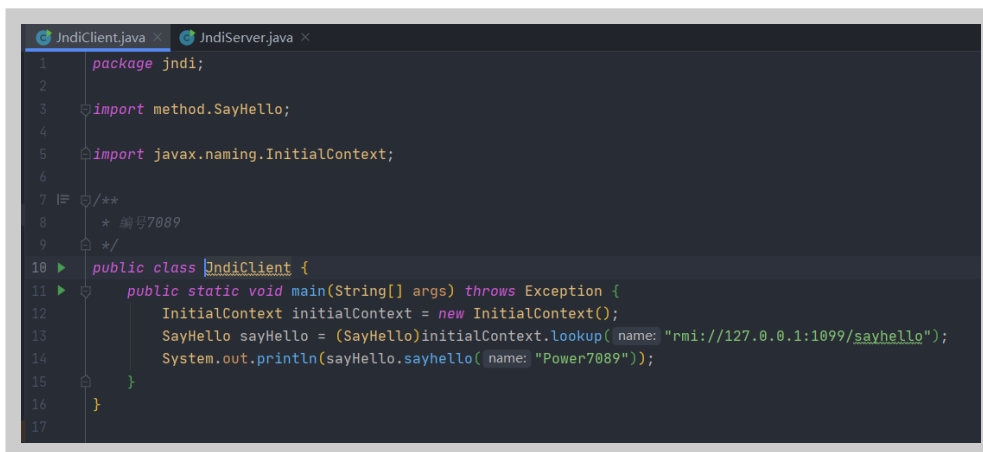
## 2、JndiClient代码

在 `JndiClient` 中键入以下代码，最终如下图所示：

```
package jndi;

import method.SayHello;
import javax.naming.InitialContext;

public class JndiClient {
    public static void main(String[] args) throws Exception {
        InitialContext initialContext = new InitialContext();
        SayHello sayHello =
        (SayHello)initialContext.lookup("rmi://127.0.0.1:1099/sayhello");
        System.out.println(sayHello.sayhello("Power7089"));
    }
}
```



```
1 package jndi;
2
3 import method.SayHello;
4
5 import javax.naming.InitialContext;
6
7 /**
8  * 编号7089
9  */
10 public class JndiClient {
11     public static void main(String[] args) throws Exception {
12         InitialContext initialContext = new InitialContext();
13         SayHello sayHello = (SayHello)initialContext.lookup(name: "rmi://127.0.0.1:1099/sayhello");
14         System.out.println(sayHello.sayhello(name: "Power7089"));
15     }
16 }
17
```

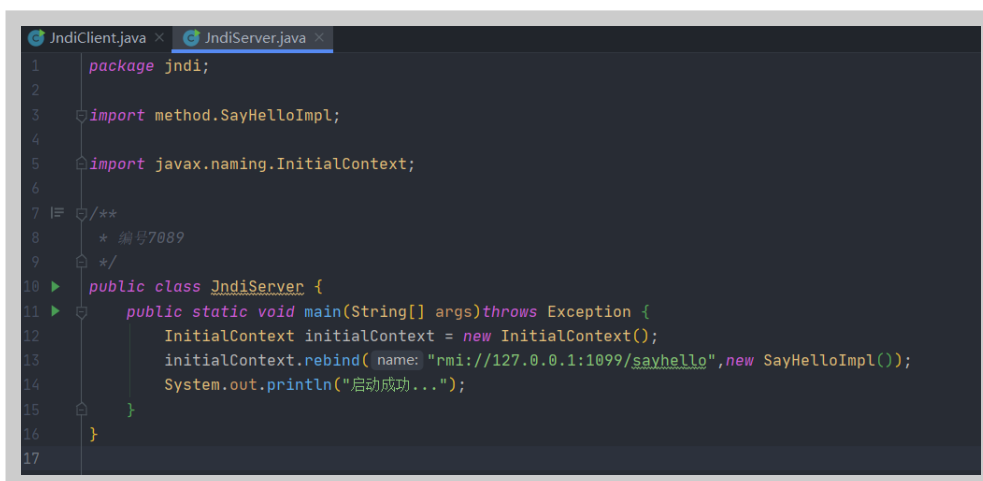
### 3、JndiServer代码

在 **JndiServer** 中键入以下代码，最终如下图所示：

```
package jndi;

import method.SayHelloImpl;
import javax.naming.InitialContext;

public class JndiServer {
    public static void main(String[] args) throws Exception {
        InitialContext initialContext = new InitialContext();
        initialContext.rebind("rmi://127.0.0.1:1099/sayhello", new
SayHelloImpl());
        System.out.println("启动成功...");
    }
}
```



```
1 package jndi;
2
3 import method.SayHelloImpl;
4
5 import javax.naming.InitialContext;
6
7 /**
8  * 编号7089
9  */
10 public class JndiServer {
11     public static void main(String[] args) throws Exception {
12         InitialContext initialContext = new InitialContext();
13         initialContext.rebind(name: "rmi://127.0.0.1:1099/sayhello", new SayHelloImpl());
14         System.out.println("启动成功...");
15     }
16 }
17
```

### 4、运行项目

- ①、先启动 **RmiServer** 。
- ②、再启动 **JndiServer** 。
- ③、最后启动 **JndiClient** 。

观察运行结果。

JNDI基础到此结束，后面实战部分会有JNDI注入，基础薄弱的朋友先补下基础，目前理解即可。