

## 1.服务器启动

```
new Hetty().start();
```

## 2.服务器配置

服务器默认配置文件为类路径下的 server.properties.相关配置属性包括:

1) server.devmod

此属性配置开发模式,默认为 false。设置为 true 会有比较详细的调试信息,但是会影响性能,建议在生产环境中关闭。

2) server.key

此属性结合 server.secret 配置服务器的认证信息。

3) server.secret

同上

4) server.port

此属性配置服务器端口号,默认为 8080。

5) properties.file

此属性配置 service 的配置文件,默认为 config.xml,可以配置多个文件,只需要用,分开即可。

6) 线程池参数的配置

server.thread.corePoolSize、server.thread.maxPoolSize、server.thread.keepAliveTime 这三个参数用于配置 hetty 线程池大小,建议根据系统环境和压力测试结果配置。

## 3.服务器端接口的配置

1) 接口

```
public interface Hello {
    String hello();
    String hello(String name);
    String hello(String name1,String name2);
    User getUser(int id);
    String getAppSecret(String key);
}
```

2) 实现类

```
public class Hello2Impl implements Hello {

    @Override
    public String hello(String name) {
        return "Hello1, "+name+"!";
    }

    @Override
    public User getUser(int id) {
        {
            User u=new User();
            u.setAge(1);
            u.setEmail("zhuzhsh@163.com");
            u.setId(1);
            u.setName("zhuzhsh");
            for(int i=0;i<5;i++){
                Role r=new Role();
                r.setName("role"+i);
                r.setDescription("role"+i);
                u.addRole(r);
            }
            System.out.println(u);
            return u;
        }
    }

    @Override
    public String getAppSecret(String key) {
        // TODO Auto-generated method stub
    }
}
```

```

        return "111";
    }

    @Override
    public String hello() {
        // TODO Auto-generated method stub
        return "Hello";
    }

    @Override
    public String hello(String name1, String name2) {
        // TODO Auto-generated method stub
        return name1+" say Hello to "+name2;
    }
}

```

### 3) 配置文件配置

```

<?xml version="1.0" encoding="UTF-8"?>
<deployment>
    <applications>
        <application user="client1" password="client1"/>
    </applications>

    <services>
        <service name="hello" interface="test.example.Hello" overload="true">
            <provider version="1" class="test.example.HelloImpl"/>
            <provider version="2" class="test.example.Hello2Impl" default="true"/>
        </service>
    </services>

    <security-settings>
        <security-setting user="client1" service="hello" version="1" />
    </security-settings>
</deployment>

```

上面的配置文件我分开来介绍，首先是 `applications` 节点，这个节点用于配置客户端的认证，比如我们在例子中就配置了一个用户（用户名和密码都是client1），这样客户端在调用时就可以使用client1来完成认证请求。在 `services` 节点中，需要配置service的名字、接口类、是否重载以及实现类的具体信息。上面的例子中，我配置了一个hello接口，接口中有重载的方法，且此接口有两个版本，版本2为默认调用的版本。在 `Security-settings` 节点中，可以配置用户对应的接口版本，比如我们的例子中，我配置了client1在调用hello service时使用版本1。

### 4. 客户端使用

首先在客户端中加入接口类，然后我们的调用代码如下：

```

String url = "http://localhost:8081/apis/hello/";

HessianProxyFactory factory = new HessianProxyFactory();

factory.setUser("client1");

factory.setPassword("client1");

factory.setOverloadEnabled(false);

final Hello basic = (Hello) factory.create(Hello.class, url);

System.out.println(basic.hello());

```

setUser和 setPassword分别用于设置用户名和密码，用于服务器端认证， setOverloadEnabled用于设置是否有方法重载（稍后详细解释）。

## 5. 注意点

在我们的service配置时，有这样的属性：overload，在客户端调用时也需要设置 OverloadEnabled，在hessian中，为了提高调用性能，默认是不支持service中的方法重载的。因为hessian客户端向服务器发送请求时，如果没有方法重载，只需要告诉服务器此次请求调用的方法名，而如果有方法重载时，需要发送方法的参数信息，哪个好，一想便知。