



# Spring's Remoting and Web Services

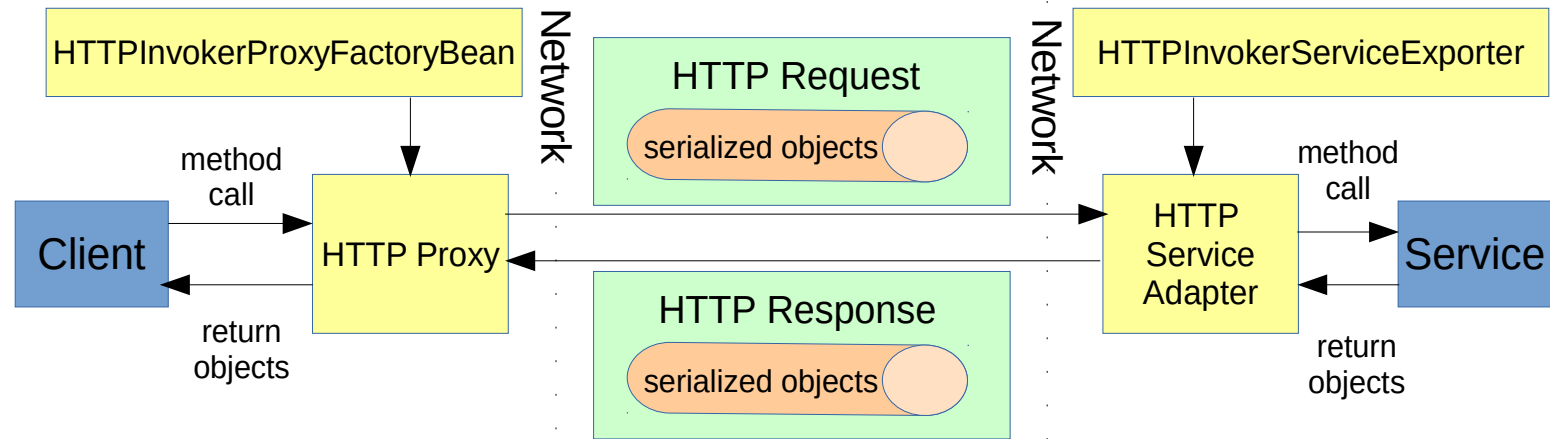
---

- Remote Method Invocation (RMI)
  - `RmiProxyFactoryBean` and `RmiServiceExporter`
- JMS
  - `JmsInvokerServiceExporter` and `JmsInvokerProxyFactoryBean`
- Hessian
  - Caucho's lightweight binary HTTP-based protocol
- Spring's HTTP invoker
  - `HttpInvokerProxyFactoryBean` and `HttpInvokerServiceExporter`
- JAX-WS
  - Java Web Services Development Pack
- Spring REST
  - Relies on Spring MVC

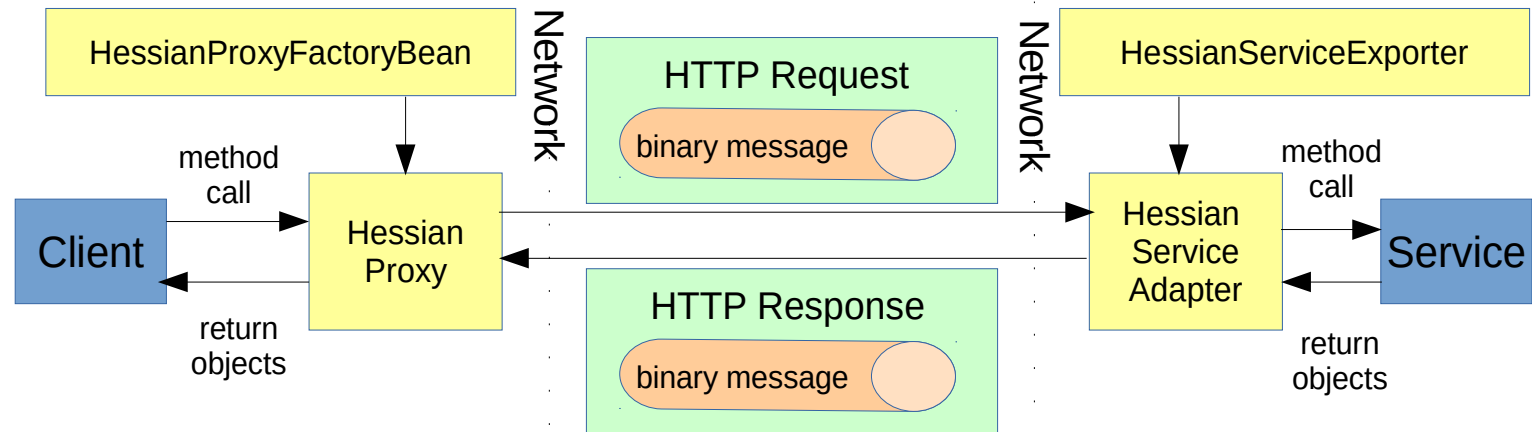


# Types of Web Services

## HTTP Invoker



## Hessian





# Hessian Server Sample

## Hello Service

```
package example;

public class BasicService implements Basic {
    private String _greeting = "Hello, world";

    public void setGreeting(String greeting)
    {
        _greeting = greeting;
    }

    public String hello()
    {
        return _greeting;
    }
}
```

## web.xml

```
<web-app>
  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>com.caucho.hessian.server.HessianServlet</servlet-class>
    <init-param>
      <param-name>home-class</param-name>
      <param-value>example.BasicService</param-value>
    </init-param>
    <init-param>
      <param-name>home-api</param-name>
      <param-value>example.Basic</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <url-pattern>/hello</url-pattern>
    <servlet-name>hello</servlet-name>
  </servlet-mapping>
</web-app>
```



# Hessian Client Sample

---

## API for Basic service

```
package example;

public interface Basic {
    public String hello();
}
```

## Hessian Client for Basic service

```
package example;

import com.caucho.hessian.client.HessianProxyFactory;

public class BasicClient {
    public static void main(String []args)
        throws Exception
    {
        String url = "http://www.caucho.com/hessian/test/basic";

        HessianProxyFactory factory = new HessianProxyFactory();
        Basic basic = (Basic) factory.create(Basic.class, url);

        System.out.println("Hello: " + basic.hello());
    }
}
```



Spring Web Services:

# **SPRING REST WEBSERVICES**



# Java Implementation Service

```
21 @Controller
22 public class StudentController {
23
24     @RequestMapping(value = "/rest/student/{id}", method = RequestMethod.GET)
25     public @ResponseBody Student getStudent(@PathVariable("id") int id) {
26         return data.get(id);
27     }
28
29     @RequestMapping(value = "/rest/students", method = RequestMethod.GET)
30     public @ResponseBody List<Student> getAllStudents() {
31         List<Student> studentList = new ArrayList<Student>();
32         Set<Integer> studentIdKeys = data.keySet();
33         for(Integer i : studentIdKeys){
34             studentList.add(data.get(i));
35         }
36         return studentList;
37     }
38
39     @RequestMapping(value = "/rest/student", method = RequestMethod.POST)
40     @ResponseStatus( HttpStatus.CREATED )
41     public @ResponseBody Student createStudent(@RequestBody Student student) {
42         student.setCreatedDate(new Date());
43         data.put(student.getId(), student);
44         return student;
45     }
46
47     @RequestMapping(value = "/rest/student/{id}", method = RequestMethod.DELETE)
48     @ResponseStatus( HttpStatus.OK )
49     public void deleteStudent(@PathVariable("id") int id) {
50         data.remove(id);
51     }
52 }
```



# web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
5
6     <!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
7     <context-param>
8         <param-name>contextConfigLocation</param-name>
9         <param-value>/WEB-INF/spring/root-context.xml</param-value>
10    </context-param>
11
12    <!-- Creates the Spring Container shared by all Servlets and Filters -->
13    <listener>
14        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
15    </listener>
16
17    <!-- Processes application requests -->
18    <servlet>
19        <servlet-name>appServlet</servlet-name>
20        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
21        <init-param>
22            <param-name>contextConfigLocation</param-name>
23            <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
24        </init-param>
25        <load-on-startup>1</load-on-startup>
26    </servlet>
27
28    <servlet-mapping>
29        <servlet-name>appServlet</servlet-name>
30        <url-pattern>/</url-pattern>
31    </servlet-mapping>
32 </web-app>
```



# servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the ${webappRoot}/resources directory -->
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <!-- Configure to plugin JSON as request and response in method handler -->
    <beans:bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
        <beans:property name="messageConverters">
            <beans:list>
                <beans:ref bean="jsonMessageConverter"/>
            </beans:list>
        </beans:property>
    </beans:bean>

    <!-- Configure bean to convert JSON to POJO and vice versa -->
    <beans:bean id="jsonMessageConverter" class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
    </beans:bean>

    <context:component-scan base-package="edu.mum.cs544.controller" />

</beans:beans>
```





# Java Implementation Client

```
public class TestRestClient {

    public static final String SERVER_URI = "http://localhost:8080/RestSample";

    public static void main(String args[]) {
        testCreateStudent();
        System.out.println("*****");
        testGetStudent();
        System.out.println("*****");
        testGetAllStudent();
    }

    private static void testGetAllStudent() {
        RestTemplate restTemplate = new RestTemplate();
        //we can't get List<Employee> because JSON convertor doesn't know the type of
        //object in the list and hence convert it to default JSON object type LinkedHashMap
        List<LinkedHashMap> studentList = restTemplate.getForObject(SERVER_URI+"/rest/students", List.class);
        System.out.println(studentList.size());
        for(LinkedHashMap map : studentList){
            System.out.println("ID="+map.get("id")+",Name="+map.get("name")+",CreatedDate="+map.get("createdDate"));;
        }
    }
}
```



# Java Implementation Client

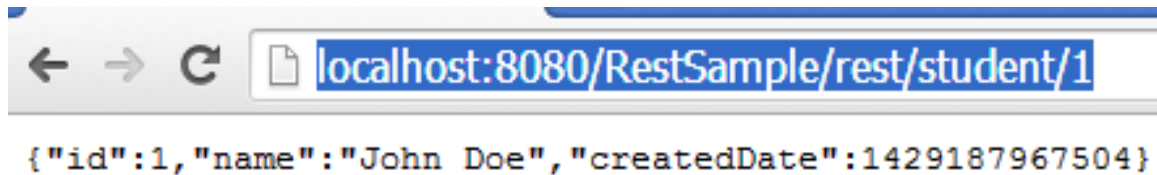
```
32 private static void testCreateStudent() {  
33     RestTemplate restTemplate = new RestTemplate();  
34     Student student = new Student();  
35     student.setId(3);  
36     student.setName("Lisa Trump");  
37     Student response = restTemplate.postForObject(SERVER_URI+"/rest/student", student, Student.class);  
38     printEmpData(response);  
39 }  
40  
41 private static void testGetStudent() {  
42     RestTemplate restTemplate = new RestTemplate();  
43     Student student = restTemplate.getForObject(SERVER_URI+"/rest/student/1", Student.class);  
44     printEmpData(student);  
45 }  
46  
47 public static void printEmpData(Student student){  
48     System.out.println("ID="+student.getId()+" ,Name="+student.getName()+" ,CreatedDate="+student.getCreatedDate());  
49 }  
50 }
```



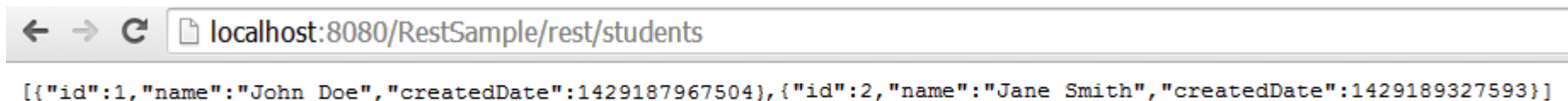
# REST Output

---

- Easy to see with a browser



A screenshot of a web browser window. The address bar shows the URL `localhost:8080/RestSample/rest/student/1`. Below the address bar, the JSON response is displayed: `{"id":1,"name":"John Doe","createdDate":1429187967504}`.



A screenshot of a web browser window. The address bar shows the URL `localhost:8080/RestSample/rest/students`. Below the address bar, the JSON response is displayed: `[{"id":1,"name":"John Doe","createdDate":1429187967504}, {"id":2,"name":"Jane Smith","createdDate":1429189327593}]`.



# Web Uniform Interface Semantics

---

RFC 2616 specifies the intended method behavior for HTTP methods

Method	CRUD	Operation	Safe	Idempotent
POST	CREATE	Create a new resource as subordinate	No	No
GET	READ	Retrieve a resource representation	Yes	Yes
PUT	UPDATE CREATE	Modify a resource or Create a resource at the id	No	Yes
DELETE	DELETE	Remove a resource	No	Yes



# SOAP/REST?

---

- WS\* SOAP → Transport level protocol
  - Middleware, interoperability standards
  - WS-AtomicTransactions
  - WS-ReliableMessaging
  - WS-Security
- REST → Application level protocol
  - Web architecture
  - Easy to build
  - Lightweight
  - Low complexity
  - Certificates and LDAP can delegate authentication and authorization
  - Secure data should not be sent as URI parameters