HTML-теги, вы можете вставлять различную текстовую, графическую, аудио, видеоинформацию и интерактивные игры на страницы электронных приложений. Интерактивные кнопки позволяют просматривать каждую страницу приложения в отдельности. При разработке мультимедийных инструментов развиваются творческие качества ученика и повышаются его компьютерные навыки. Изучая компьютерные технологий, студент станет высококвалифицированным специалистом со знанием компьютерной грамотности. Он готовит себя к будущим навыкам работы на компьютере в своей работе в качестве будущего профессионала.

В данной статье приводится результат творческой работы над проектом, которая прошла в рамках курса «Информационные и коммуникационные технологии», является актуальной темой в подготовке специалистов педагогического направления.

Ключевые слова: *графика, цвета, инструмент, нарисованный, Inkscape, HTML, веб-страницы, теги.*

УДК 004.422.81

Salykova, O.S.
c. t. s., associate professor of Software Department,
KSU named after A. Baytursynov
Ivanova, I.V.
c. p. s., docent of Software Department,
KSU named after A. Baytursynov
Zhiyenbayeva, A.A.
2nd year master student of specialty
«6M070400 – Computer Engineering and Software»,
KSU named after A. Baytursynov
Tuletaev, E.S.
2nd year master student of specialty
«6M072400 – Technological Machines and Equipment»,
KSU named after A. Baytursynov,
Kostanay, Kazakhstan

FEATURES OF USING REST-ARCHITECTURE FOR DEVELOPMENT OF CLIENT-SERVER APPLICATIONS

Abstract

This article considers features of using REST-architecture for development of client-server applications. HTTP client libraries for Java programming language and examples of creating GET and POST requests on Java are given. The main principles of working with Restlet framework for development of applications on Java programming language are provided.

Key words: *REST-architecture, client-server application, HTTP, Restlet, Java programming language.*

1 Introduction

Nowadays, there are a lot of ways to create client-server applications, and one of the widespread one is REST API. REST is a style of software architecture for distributed hypermedia systems; that is, systems in which text, graphics, audio, and other media are stored across a network and interconnected through hyperlinks. It stands for REpresentation State Transfer, which requires clarification because the central abstraction in REST – the *resource* – does not occur in the acronym. A resource in the RESTful sense is anything that has an URI; that is, an identifier that satisfies formatting requirements. The formatting requirements are what make URIs *uniform*. Recall, too, that URI stands for Uniform *Resource* Identifier; hence, the notions of *URI* and *resource* are intertwined.

2 Materials and methods

In a RESTful request targeted at a resource, the resource itself remains on the service machine. The requester typically receives a *representation* of the resource if the request succeeds. It is the representation that transfers from the service machine to the requester machine. In different terms, a RESTful client issues a request that involves a resource, for instance, a request to *read* the resource. If this read request succeeds, a typed representation (for instance, text/html) of the resource is transferred from the server that hosts the resource to the client that issued the request. The representation is a good one only if it captures the resource's state in some appropriate way.

In summary, RESTful web services require not just resources to represent but also client-invoked operations on such resources. At the core of the RESTful approach is the insight that HTTP, despite the occurrence of *Transport* in its name, is an API and not simply a transport protocol. HTTP has its well-known *verbs*, officially known as *methods*. Table 1 shows the HTTP verbs that correspond to the *CRUD* (Create, Read, Update, Delete) operations so familiar throughout computing.

Table 1 – HTTP verbs and CRUD operations

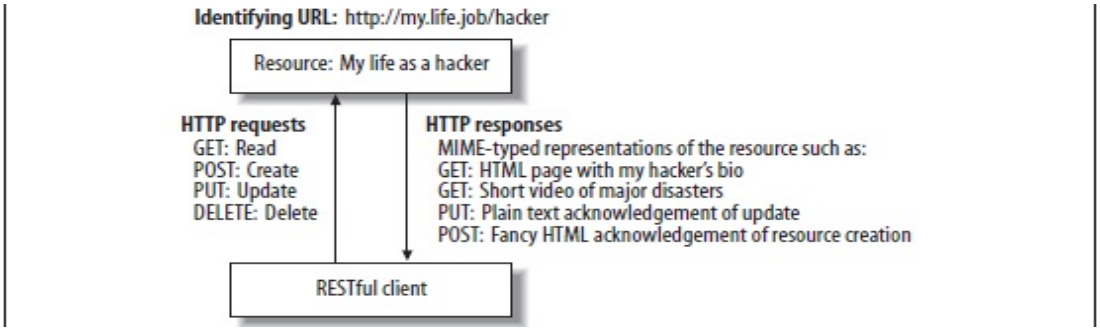
HTTP verb	Meaning in CRUD terms
POST	*Create* a new resource from the request data
GET	*Read* a resource
PUT	*Update* a resource from the request data
DELETE	*Delete* a resource

Although HTTP is not case-sensitive, the HTTP verbs are traditionally written in uppercase. There are additional verbs. For example, the verb HEAD is a variation on GET that requests only the HTTP headers that would be sent to fulfill a GET request. There are also TRACE and INFO verbs.

Picture 1 is a whimsical depiction of a resource with its identifying URI, together with a RESTful client and some typed representations sent as responses to HTTP requests for the resource. Each HTTP request includes a verb to indicate which CRUD operation should be performed on the resource. A good representation is precisely one that matches the requested operation and captures the resource's state in some appropriate way. For example, in this depiction a GET request could return my biography as a hacker as either an HTML document or a short video summary. The video would fail to capture the state of the resource if it depicted, say, only the major disasters in my brother's career rather than those in my own. A typical HTML representation of the resource would include hyperlinks to other resources, which in turn could be the target of HTTP requests with the appropriate CRUD verbs.

HTTP also has standard response codes, such as 404 to signal that the requested resource could not be found, and 200 to signal that the request was handled successfully. In short, HTTP provides request verbs and MIME types for client requests and status codes (and MIME types) for service responses.

Modern browsers generate only GET and POST requests. Moreover, many applications treat these two types of requests interchangeably. For example, Java HttpServlets have callback methods such as doGet and doPost that handle GET and POST requests, respectively. Each callback has the same parameter types, HttpServletRequest (the key/value pairs from the requester) and HttpServletResponse (a typed response to the requester). It is common to have the two callbacks execute the same code (for instance, by having one invoke the other), thereby conflating the original HTTP distinction between *read* and *create*. A key guiding principle of the RESTful style is to respect the original meanings of the HTTP verbs. In particular, any GET request should be side effect-free (or, in jargon, *idempotent*) because a GET is a *read* rather than *create*, *update*, or *delete* operation. A GET as a *read* with no side effects is called a *safe* GET.



Picture 1 – A small slice of a RESTful system

The REST approach does not imply that either resources or the processing needed to generate adequate representations of them are simple. A REST-style web service might be every bit as subtle and complicated as a SOAP-based service. The RESTful approach tries to simplify matters by taking what HTTP, with its MIME type system, already offers: built-in CRUD operations, uniformly identifiable resources, and typed representations that can capture a resource's state. REST as a design philosophy tries to isolate application complexity at the endpoints, that is, at the client and at the service. A service may require lots of logic and computation to maintain resources and to generate adequate representation of resources – for instance, large and subtly formatted XML documents – and a client may require significant XML processing to extract the desired information from the XML representations transferred from the service to the client. Yet the RESTful approach keeps the complexity out of the transport level, as a resource representation is transferred to the client as the body of an HTTP response message. By contrast, a SOAP-based service inevitably complicates the transport level because a SOAP *message* is encapsulated as the body of a transport message; for instance, an HTTP or SMTP message. SOAP requires messages within messages, whereas REST does not [1, p. 123].

3, 4 Results and discussions

HTTP client libraries for Java programming language.

The Java standard library comes with an HTTP client, java.net.HttpURLConnection. You can get an instance by calling open on a java.net.URL object. Though it supports most of the basic features of HTTP, programming to its API is very difficult [2, p. 34]. The Apache Jakarta project has a competing client called HttpClient, which has a better design. There's also Restlet. It's also an HTTP client library. The class org.restlet.Client makes it easy to make simple HTTP requests, and the class org.restlet.data.Request hides the HttpURLConnection programming necessary to make more complex requests. Table 2 lists the features available in each library.

Table 2 – HTTP feature matrix for Java HTTP client libraries

	HttpURLConnection	HttpClient	Restlet
HTTPS	Yes	"	"
HTTP verbs	All	"	"
Custom data	Yes	"	"
Custom headers	Yes	"	"
Proxies	Yes	"	"
Compression	No	No	Yes
Caching	Yes	No	Yes
Auth methods	Basic, Digest, NTLM	"	Basic, Amazon
Cookies	Yes	"	"
Redirects	Yes	"	"

Now, let's consider an example of connection a client on Java with REST API. Apache HTTP Client makes the request processing simpler, and in case of using Maven (a build automation tool used primarily for Java projects) it is possible to write following XML:

```
<dependency>
<groupId>org.apache.httpcomponents</groupId>
<artifactId>httpclient</artifactId>
<version>4.3.6</version>
</dependency>
```

A REST API client uses the GET method in a request message to retrieve the state of a resource, in some representational form. A client's GET request message may contain headers but no body [3, p. 24].

Here is an example for GET requests:

```
public class Test {
    public static void main(String[] args) throws ClientProtocolException, IOException {
        HttpClient client = new DefaultHttpClient();
        HttpGet request = new HttpGet("http://restUrl");
        HttpResponse response = client.execute(request);
        BufferedReader rd = new BufferedReader (new
        InputStreamReader(response.getEntity().getContent()));
        String line = "";
        while ((line = rd.readLine()) != null) {
        System.out.println(line);
        }
    }
}
```

Clients use POST when attempting to create a new resource within a collection. The POST request's body contains the *suggested* state representation of the new resource to be added to the server-owned collection [3, p. 26].

An example for POST requests:

```
public class Test {
    public static void main(String[] args) throws ClientProtocolException, IOException {
        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost("http://restUrl");
        List nameValuePairs = new ArrayList(1);
        nameValuePairs.add(new BasicNameValuePair("name", "value"));
        post.setEntity(new UrlEncodedFormEntity(nameValuePairs));
        HttpResponse response = client.execute(post);
        BufferedReader rd = new BufferedReader(new
        InputStreamReader(response.getEntity().getContent()));
        String line = "";
        while ((line = rd.readLine()) != null) {
        System.out.println(line);
        }
    }
}
```

As the REST design philosophy becomes more popular, new frameworks are springing up to make RESTful design easy. Existing frameworks are acquiring RESTful modes and features. This, in turn, drives additional interest in REST [2, p. 339].

The Restlet Framework. Several web frameworks have embraced REST, perhaps none more decisively than Rails with its ActiveResource type, which implements a resource in the RESTful sense. Rails also emphasizes a RESTful style in routing, with CRUD request operations specified as standard HTTP verbs. Grails is a Rails knockoff implemented in Groovy, which in turn is a Ruby knockoff with access to the standard Java packages. Apache Sling (*http://incubator.apache.org/sling/site/index.html*) is a Java-based web framework with a RESTful orientation [1, p. 186].

The Restlet framework (*http://www.restlet.org*) adheres to the REST architectural style and draws inspiration from other lightweight but powerful frameworks such as Net-Kernel (*http://www.1060.org*) and Rails. As the name indicates, a restlet is a RESTful alternative to the traditional Java servlet. The restlet framework has a client and a service API. The framework is well designed, relatively straightforward, professionally implemented, and well documented. It plays well with existing technologies. For example, a restlet can be deployed in a servlet container such as Tomcat or Jetty. The restlet distribution includes integration support for the Spring framework and also comes with the Simple HTTP engine (*http://www.simpleframework.org*), which can be embedded in Java applications.

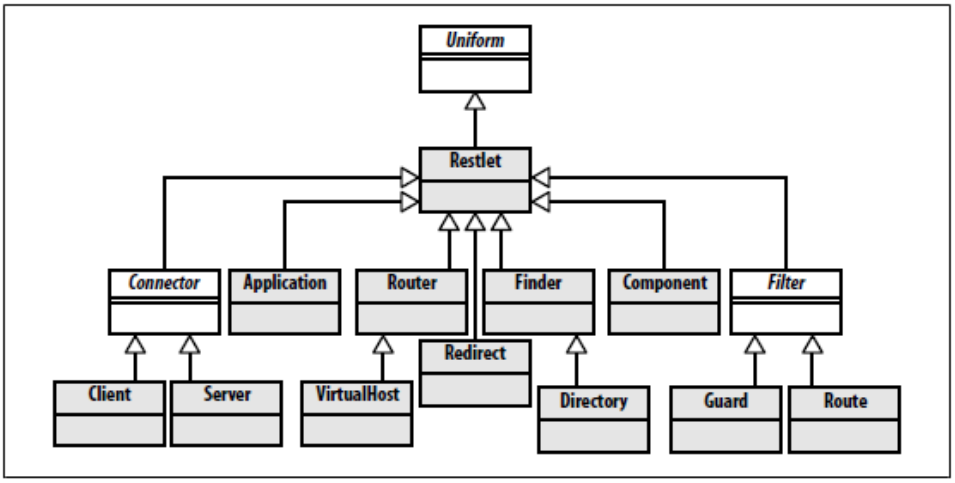
The Restlet framework provides easy-to-use Java wrappers such as Method, Request, Response, Form, Status, and MediaType for HTTP and MIME constructs. The framework supports virtual hosts for commercial-grade applications.

The Restlet download includes a subdirectory *RESTLET_HOME/lib* that houses the various JAR files for the restlet framework itself and for interoperability with Tomcat, Jetty, Spring, Simple, and so forth. For the sample restlet service in this section, the JAR files *com.noelios.restlet.jar*, *org.restlet.jar*, and *org.simpleframework.jar* must be on the classpath. A restlet client could be written using a standard class such as HttpURLConnection.

The Restlet terminology matches the terminology of REST as described in the Fielding thesis: resource, representation, connector, component, media type, language, and so on.

Restlet adds some specialized classes like Application, Filter, Finder, Router, and Route, to make it easier to combine restlets with each other, and to map incoming requests to the resources that ought to handle them.

The central concept of Restlet is the abstract Uniform class, and its concrete subclass Restlet. As the name implies, Uniform exposes a uniform interface as defined by REST. This interface is inspired by HTTP's uniform interface but can be used with other protocols like FTP and SMTP. The main method is handle, which takes two arguments: Request and Response. As it can be seen from Picture 2, every call handler that is exposed over the network (whether as client or server) is a subclass of Restlet – is a restlet – and respects this uniform interface. Because of this uniform interface, restlets can be combined in very sophisticated ways.



Picture 2 – The Restlet class hierarchy

Every protocol that Restlet supports is exposed through the handle method. This means HTTP (server and client), HTTPS, and SMTP, as well as JDBC, the file system, and even the class loaders all go through handle. This reduces the number of APIs the developer must learn. Filtering, security, data transformation, and routing are handled by chaining together subclasses of Restlet. Filters can provide processing before or after the handling of a call by the next restlet. Filter instances work like Rails filters, but they respond to the same handle method as the other Restlet classes, not to a filter-specific API.

A Router restlet has a number of Restlet objects attached to it, and routes each incoming protocol call to the appropriate Restlet handler. Routing is typically done on some aspect of the target URI, as in Rails. Unlike Rails, Restlet imposes no URI conventions on your resource hierarchies. You can set up your URIs however you want, so long as you program your Routers appropriately.

Routers can stretch beyond this common usage. You can use a Router to proxy calls with dynamic load balancing between several remote machines! Even a setup as complex as this still responds to Restlet's uniform interface, and can be used as a component in a larger routing system. The VirtualHost class (a subclass of Router) makes it possible to host several applications under several domain names on the same physical machine. Traditionally, to get this kind of feature you've had to bring in a front-end web server like Apache's httpd. With Restlet, it's just another Router that responds to the uniform interface.

An Application object can manage a portable set of restlets and provide common services. A "service" might be the transparent decoding of compressed requests, or tunneling methods like PUT and DELETE over overloaded POST using the method query parameter. Finally, Component objects can contain and orchestrate a set of Connectors, VirtualHosts, and Applications that can be run as a standalone Java application, or embedded in a larger system such as a J2EE environment.

5 Conclusions

In conclusion, it should be noted that REST API is one of the best ways to develop client-server applications, because of its easiness to understand and use in the process of creating web applications, mobile applications and other kinds of client-server based software. The restlet framework is a quick study. Its chief appeal is its RESTful orientation, which results in a lightweight but powerful software environment for developing and consuming RESTful services. The chief issue is whether the restlet framework can gain the market and mind share to become the standard environment for RESTful services in Java. The Jersey framework has the JSR seal of approval, which gives this framework a clear advantage.

References

1 Kalin M. Java Web Services: Up and Running. – O'Reilly Media, Inc., 2009. – 300 p.
2 Richardson L., Ruby, S. RESTful Web Services. – O'Reilly Media, Inc., 2007. – 422 p.
3 Massé M. REST API Design Rulebook. – O'Reilly Media, Inc., 2012. – 96 p.

Article was received by the editorial office: 20.12.2018

САЛЫКОВА, О.С., ИВАНОВА, И.В., ЖИЕНБАЕВА, А.А., ТУЛЕТАЕВ, Е.С.

КЛИЕНТ-СЕРВЕРЛІК ҚОСЫМШАЛАРДЫ ЖАСАУ БАРЫСЫНДА REST-АРХИТЕКТУРАСЫН ҚОЛДАНУ ЕРЕКШЕЛІКТЕРІ

Берілген мақалада клиент-серверлік қосымшаларды жасау үшін REST-архитектурасын қолдану ерекшеліктері қарастырылған. Java бағдарламалау тіліне арналған HTTP клиент кітапханалары мен Java-да GET және POST сұрауларын жасау мысалдары келтірілген. Java бағдарламалау тілінде қосымшалар жасауға арналған Restlet фреймворкының жұмыс принциптері көрсетілген.

Кілт сөздер: *REST-архитектурасы, клиент-серверлік қосымша, HTTP, Restlet, Java бағдарламалау тілі.*

САЛЫКОВА, О.С., ИВАНОВА, И.В., ЖИЕНБАЕВА, А.А., ТУЛЕТАЕВ, Е.С.
ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ REST-АРХИТЕКТУРЫ ДЛЯ РАЗРАБОТКИ КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ

В данной статье рассматриваются особенности использования REST-архитектуры для разработки с клиент-серверными приложениями. Приведены клиент библиотеки HTTP для языка программирования Java, а также примеры создания GET и POST запросов на Java. Представлены основные принципы работы фреймворка Restlet для разработки приложений на языке программирования Java.

Ключевые слова: *REST-архитектура, клиент-серверное приложение, HTTP, Restlet, язык программирования Java.*

УДК 796.012

Сивохин, И.П.
доктор педагогических наук,
главный специалист по научной деятельности,
КГПУ, Костанай, Казахстан
Огиенко, Н.А.
кандидат педагогических наук,
зав. кафедрой ТиПФКиС, КГПУ, Костанай
Бекмухамбетова, Л.С.
магистр экономических наук,
преподаватель кафедры ТиПФКиС,
КГПУ, Костанай, Казахстан
Маткаримов, Р.М.
кандидат педагогических наук, профессор,
заведующий кафедрой ТиМТАВиКС, УГУФКиС,
Ташкент, Узбекистан.

БИОМЕХАНИЧЕСКИЕ АСПЕКТЫ СОВЕРШЕНСТВОВАНИЯ ДВИГАТЕЛЬНЫХ ДЕЙСТВИЙ В СПОРТЕ

Аннотация

Для регистрации траектории движения штанги и расчета кинематических и динамических показателей был использован специализированный аппаратно-программный комплекс и соответствующее программное обеспечение. В исследовании приняли участие спортсмены высокой квалификации (n=13). Анализ полученных биомеханических показателей техники толчка штанги от груди позволил выявить факторы, влияющие на эффективность двигательного действия. На основе полученных данных была разработана инновационная программа совершенствования двигательного действия и проверена в ходе педагогического эксперимента (n=5) с использованием биомеханического контроля.

Ключевые слова: *тяжелая атлетика, биомеханический контроль, техника толчка штанги от груди, статистический анализ, факторный анализ, совершенствование технического мастерства.*

1 Введение

Актуальность исследования. Совершенствование двигательных действий и технического мастерства тяжелоатлетов связано с применением различных современных инструментальных методик объективного контроля, которые необходимы для получения точных количественных показателей биомеханической структуры двигательных действий спортсменов,