Клиент-серверное взаимодействие

HTTP. REST. JSON. SOAP

Файзрахманова Карина Эмильевна,
доцент кафедры АСУ, 6-315
malikovakarina@gmail.com

Клиент-серверное взаимодействие

В настоящее время глобальное информационное пространство представлено множеством веб-приложений. Как правило, в их основу положена трехуровневая архитектура клиент-серверного взаимодействия, предполагающая разделение обработки на три логически различимых сегмента. Рассмотрим три ключевых компонента архитектурной модели с целью отражения их функциональности и организации взаимодействия.

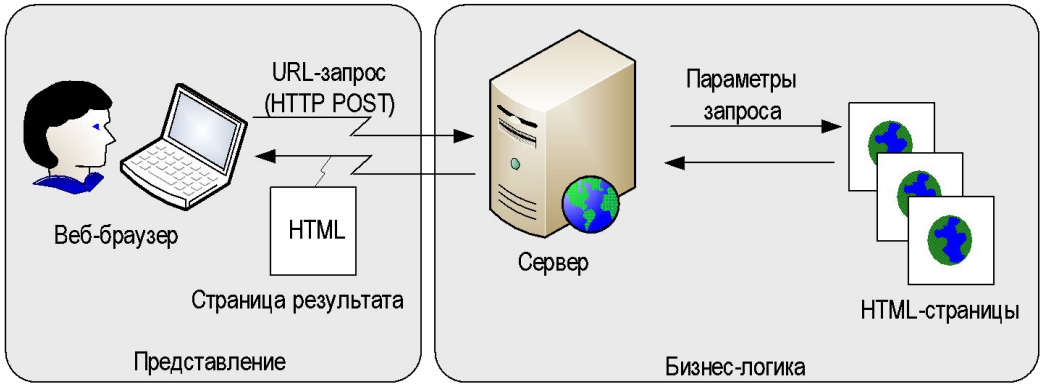
1. **Представление.** Это интерфейсный (обычно графический) компонент комплекса, предоставляемый конечному пользователю. Этот уровень не должен иметь прямых связей с базой данных (по требованиям безопасности и масштабируемости), быть нагруженным основной бизнес-логикой (по требованиям масштабируемости) и хранить состояние приложения (по требованиям надёжности). На этот уровень обычно выносится только простейшая бизнес-логика: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции с данными (сортировка, группировка, подсчёт значений), уже загруженными на терминал.

2. Веб-сервер, обеспечивающий обработку запросов клиентов и формирование ответов на основе исполнения сценариев и запросов к серверным базам данных или иным структурированным или неструктурированным источникам.

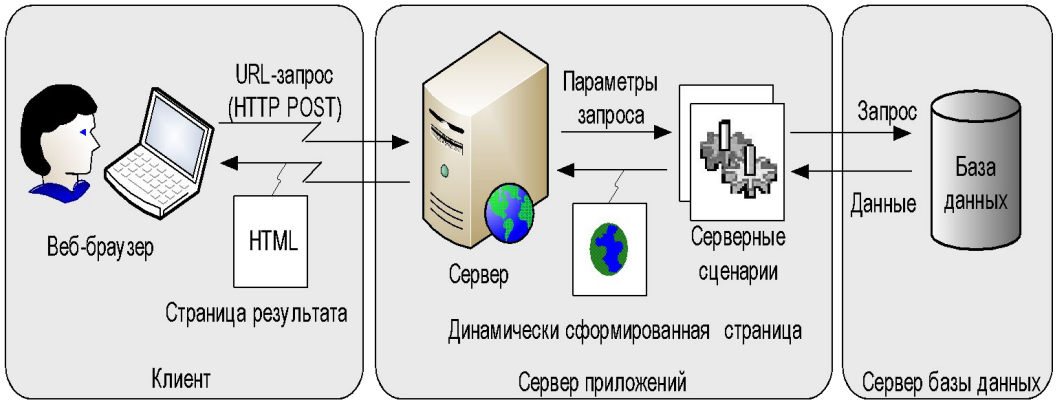
3. Хранение данных. Службы хранения данных обеспечиваются различными структурированными и неструктурированными информационными хранилищами, вспомогательными инструментами, которые обеспечивают доступ к необходимым данным из различных областей приложения

Существует широкий выбор компонентов и технологий, которые могут быть частью архитектуры веб-приложений. В результате построение архитектурной модели сводится к выбору основных функциональных компонентов системы, архитектурных решений на каждом из уровней проектируемого веб-приложения, где центральное место занимают процессы интерпретации.

Клиент-серверное взаимодействие



Статическое веб-приложение предполагает применение следующей схемы клиент-серверного взаимодействия. Взаимодействие клиент-серверных веб-систем начинается с инициализации соединения и отправки URL-запроса. В случае если клиентский запрос принимается на определенную статическую страницу, то серверное приложение, самостоятельно обслужив запрос, отправляет HTML-содержимое запрашиваемой страницы.



Динамическое веб-приложение реализует следующую модель поведения. Клиентская сторона обращается к серверному сценарию, который, получив соответствующий запрос, выполняет предусмотренные действия для динамического формирования страницы. Серверные сценарии по мере необходимости выполняют выборку данных из соответствующего источника и представляют динамически сгенерированное содержание клиенту.

Общая схема взаимодействия клиента и интернет-приложения

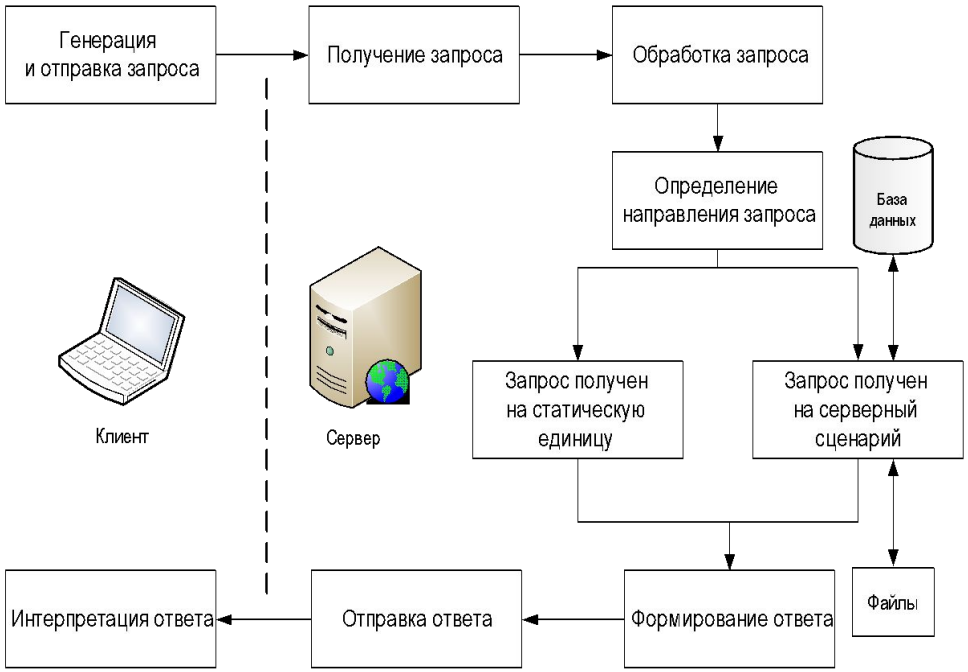
Взаимодействие клиент-серверных веб-систем начинается с инициализации соединения и отправки URL-запроса.

В случае если клиентский запрос принимается на определенную статическую страницу, то серверное приложение, самостоятельно обслужив запрос, отправляет HTML-содержимое запрашиваемой страницы.

Если клиентская сторона обращается к серверному сценарию, выполняются предусмотренные действия для динамического формирования страницы.

Серверные сценарии по мере необходимости выполняют выборку данных из соответствующего источника и представляют динамически сгенерированное содержание клиенту.

В простейшем случае это HTML-код, в соответствии с которым браузер формирует изображение. В более сложном случае ответ включает информационные объекты, предназначенные для дальнейшей интерпретации на стороне клиента.



Протокол прикладного уровня HTTP

Протокол – набор соглашений, которые задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения. Как правило, в качестве фундамента связи между пользователем и сервером применяется протокол HTTP *(HyperText Transfer Protocol)*, поддерживаемый всеми браузерами.

HTTP – протокол прикладного уровня передачи данных, изначально – в виде гипертекстовых документов в формате HTML. **Ресурс HTTP-запроса** – различные объекты: файлы, документы, фото и т. п. **URI –** унифицированный) идентификатор ресурса (/department/). URL – это URI, который, помимо идентификации ресурса, содержит и информацию о местонахождении ресурса (http:://localhost:8080/department/).

Характеристики HTTP протокола.

Основан на клиент-серверной архитектуре (клиент – программа (например, браузер), который устанавливает соединение с сервером для отправки одного или нескольких HTTP-запросов, сервер – программа, которая обрабатывает запрос и отправляет HTTP-ответ).

Каждый ресурс идентифицируется с помощью унифицированного идентификатора ресурса.

HTTP не привязан к конкретному типу данных. Можно передавать любой тип данных при условии, что клиент и сервер способны обрабатывать выбранный тип данных. Сервер и клиент должны определяют тип контента с помощью MIME(Multipurpose Internet Mail Extensions - многоцелевые расширения интернет-почты). MIME-тип – стандарт, который описывает формат документа, файла или набора байтов. Структура MIME-типа: тип/подтип (например, application/json).

Протокол HTTP синхронный, но позволяет клиенту отправлять несколько запросов подряд, не дожидаясь ответа сервера, при условии, что сервер отправит ответы на запросы в том порядке, как они приходили.

Взаимодействует только через соединение. Клиент и сервер могут взаимодействовать друг с другом только с помощью запроса, ни клиент, ни сервер не могут получить информацию за пределами запроса.

Не зависит от соединения. Для отправки запроса клиент устанавливает соединение с сервером и отсоединяется после отправки запроса. Сервер, в свою очередь, обрабатывает запрос и устанавливает соединение с клиентом для отправки ответа и отсоединяется после нее. HTTP/1.0 использует соединение для каждого цикла “запрос/ответ”. HTTP/1.1 может использовать один или несколько циклов “запрос-ответ” внутри одного соединения.

Структура HTTP-запроса

Каждое HTTP-сообщение состоит из элементов, которые передаются в указанном порядке:

– стартовая строка (обязательный элемент);

– заголовки (опциональный элемент);

– пустая строка (обязательный элемент, которая определят конец полей элемента header);

– тело сообщения (опциональный элемент).

Стартовая строка (starting line) или строка состояния – определяет метод передачи, версию протокола HTTP, а также URL, к которому должен выполняется запрос. Стартовая строка запроса имеет следующий формат: метод (пробел) URI запроса (пробел) версия-HTTP (следующая строка).

```
GET /students.html HTTP/1.1 (строка-запрос, сделанный клиентом)
```

Заголовки (headers) – характеризуют тело сообщения, параметры передачи и различную служебную информацию. Порядок HTTP заголовков не имеет значения. Однако желателен следующий порядок заголовков: общие заголовки, заголовки запроса (или заголовки ответа) и заголовки объекта.

```
User-Agent:  Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
             (KHTML, like Gecko) Chrome/84.0.4147.105 Safari/537.36
Last-Modified: Sat, 23 Jan 2020 21:16:42 GMT
Authorization: sso_1.0_b390cde1-532e-4387-bcbd-f9e0a4aaac13
Content-Type: application/json
Content-Length: 20
```

Тело сообщения (message body) содержит объект, связанный с запросом, либо с ответом. Тело сообщения содержит данные HTTP запроса (тип данных и т.д.), а HTTP ответ содержит данные, полученные от сервера (файлы, изображения и т.д.).

```
{caption_like: "Test"}
```

HTTP-методы GET, POST, HEAD, PUT, DELETE

HTTP метод – регистрозависимая последовательность символов, указывающая на основную операцию над ресурсом. Сторона сервера может использовать любые методы, обязательных методов для серверных и клиентских сторон нет. Если сервер не идентифицировал указанный клиентом метод, то он возвращает статус 501 (Not Implemented). Если серверу метод известен, но данный метод неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed).

GET – метод генерирования данных, используемый клиентским приложением для получения информации от сервера по заданному URL. Запросы клиентов, использующие метод GET должны получать только данные и не должны оказывать влияния на данные. Например, получение HTML-файла, который содержит информацию о студентах кафедры.

POST – необходим для создания нового ресурса. Например, отправить данные на сторону сервера из HTML-формы, а также добавить данные в базу данных.

HEAD – используется для получения метаинформации об объекте без отправки тела HTTP-сообщения. В качестве ответа сервер отправляет только заголовки и статусную строку без тела HTTP-сообщения. Например, тестирование HTTP-соединения, а также достижимости узлов/ресурсов (тестирование HTTP методом HEAD осуществляется быстрее, т.к. не предполагает отправку содержимого). Например, для получения информации о дате обновления ресурса (затем использовать GET для получения его содержания).

PUT – обеспечивает обновление существующего ресурса (или создание нового). Например, обновление информации о конкретном студенте.

DELETE – запрашивает сервер об удалении ресурса. Действие метода DELETE может быть отменено вмешательством администратора сервера или программным кодом. Использование данного метода гарантирует завершение операции удаления, даже если код состояния, возвращенный первоначальным сервером указывает на то, что действие было завершено успешно.

Структура HTTP-ответа

После получения и обработки запроса от клиента, сервер отправляет ответ в виде HTTP-сообщения, которое имеет вид:

– строка статуса (обязательный элемент);
– заголовок (опциональный элемент);
– пустая строка (указывает на окончание заголовка – обязательный элемент);
– тело сообщения (опциональный элемент).

Стартовая строка ответа сервера имеет следующий формат:

```
Версия-HTTP (пробел) Код-статуса (пробел) Текстовое-сообщение
HTTP/1.1 200 OK (строка-Статуса, отправленная сервером)
```

Код статуса (состояния) – это число, определяющие тип ответа, а также конкретную ошибку.

```
403 Forbidden //доступ запрещен
```

Заголовок ответа – позволяет серверу передать дополнительную информацию об ответе, которая не может быть помещена в строке состояния:

```
Content-Type: application/json; charset=UTF-8
Date: Tue, 26 Jan 2021 16:47:30 GMT
```

Коды состояния в HTTP протоколе

Код состояния – элемент ответа HTTP-сервера, представлен целым числом из трех десятичных цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделенная пробелом поясняющая фраза, которая содержит причину данного ответа. (Например, 201 Created, 401 Unauthorized, 507 Insufficient Storage). По коду ответа клиент определяет результаты запроса, а также необходимые дальнейшие действия. Набор кодов состояния является стандартом, который описан в соответствующих документах RFC.
Описание HTTP кодов состояния.

1xx: информационные коды состояния. В этот класс выделены коды, информирующие о процессе передачи. Например,
`100 Continue` – сервер удовлетворен начальными сведениями о запросе, клиент может продолжать отправлять заголовки;
`102 Processing` – запрос принят, но на его обработку необходимо длительное время.

2xx: успешные коды состояния. Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может также отправить заголовки и тело сообщения. Например:

`200 OK` – успешный запрос. Если клиентом были запрошены данные, то они находятся в заголовке и/или теле сообщения;
`201 Created` – в результате успешного выполнения запроса был создан новый ресурс.

3xx: коды перенаправления. Коды данного класса сообщают клиенту, что для успешного выполнения операции необходимо сделать другой запрос, как правило, по другому URI. Например:

`301 Moved Permanently` – запрошенный документ был перенесен на новый URI, указанный в поле Location заголовка;

`300 Multiple Choices` – по указанному URI существует несколько вариантов предоставления ресурса по типу MIME (по языку или по другим характеристикам). Сервер передает с сообщением список альтернатив, предоставляя возможность сделать клиенту выбор.

4xx: коды ошибок клиента. Предназначены для указания ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя. Например:

`400 Bad Request` – в запросе клиента обнаружена синтаксическая ошибка;

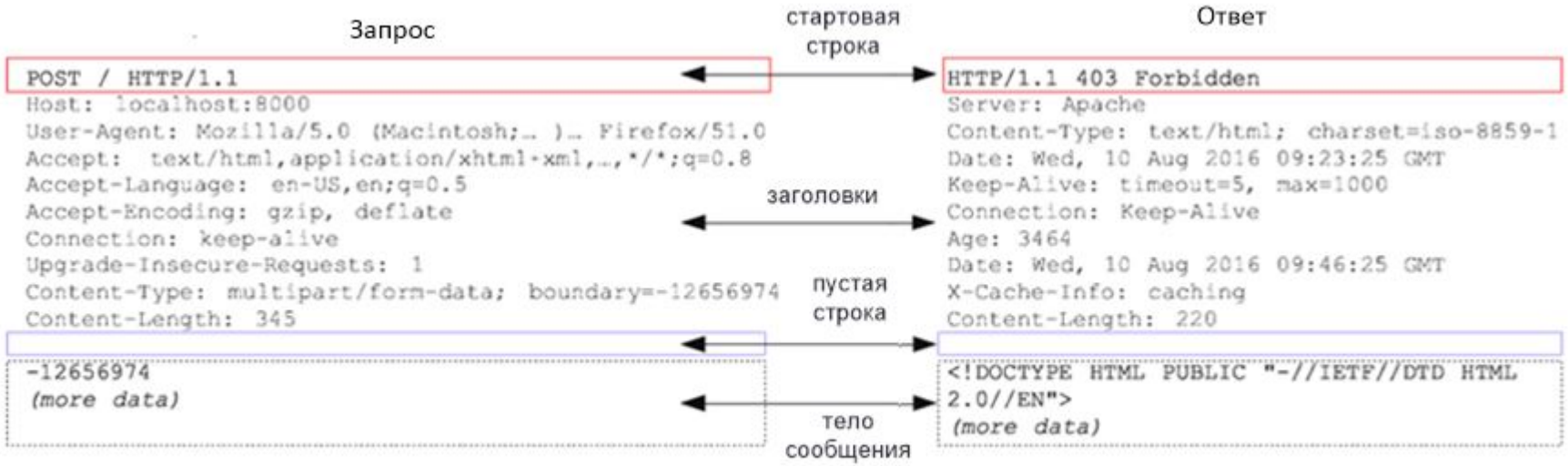
`401 Unauthorized` – для доступа к запрашиваемому ресурсу требуется аутентификация;

`404 Not Found` – ошибка в написании адреса страницы.

5xx: коды ошибок сервера. Выделены для случаев необработанных исключений при выполнении на стороне сервера. Для всех запросов (кроме HEAD-метода) сервер должен включать в тело сообщения отображение для пользователя. Например:
`502 Bad Gateway` – сервер, выступая в роли шлюза или прокси-сервера, получил недействительное ответное сообщение от вышестоящего сервера;

`500 Internal Server Error` – любая внутренняя ошибка сервера, которая не входит в рамки остальных ошибок класса.

Пример HTTP-ответа и запроса



Архитектура REST

Архитектурный стиль взаимодействия компонентов распределенного приложения REST впервые представил один из авторов HTTP-протокола Рой Филдинг в 2000 г. Отличительной особенностью REST-приложений является то, что они позволяют использовать протокол HTTP наилучшим образом. Архитектура REST не привязана к конкретным технологиям и протоколам, но подразумевает использование HTTP и распространенных форматов представления ресурсов, например, JSON, XML.

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения. Представляет собой согласованный набор ограничений, учитываемых при проектировании распределенной системы. Накладываемые ограничения определяют работу сервера в том, как он может обрабатывать и отвечать на запросы клиентов. Действуя в рамках этих ограничений, система приобретает свойства производительности, масштабируемости, простоты, способности к изменениям, переносимости, отслеживаемости и надежности.

REST-ресурс – абстракция некоторой сущности, которой можно дать имя (как классы в Java): пользователь, документ, задача, список задач, отчет, заказ, видео, анимация, PDF-файл.

URL-ресурса – последовательность символов, которая не подлежит изменению при изменении состояния ресурса, идентифицирующая абстрактный или физический ресурс. Идентификатором в REST является URI. Пример: POST/users (создать пользователя), DELETE /users/1 (удалить пользователя), GET /users (вернуть (получить) всех пользователей), GET /users/1 (Получить одного пользователя).

RESTful система – система, построенная с учетом REST (то есть не нарушающая накладываемых архитектурой ограничений).

Принципы REST

1. Модель клиент-сервер. Первым принципом, применимым к архитектуре REST, является приведение архитектуры к модели клиент-сервер. Отделение потребности интерфейса клиента от потребностей сервера, хранящего данные, повышает переносимость кода клиентского интерфейса на другие платформы, повышает масштабируемость, а также предоставляет возможность отдельным частям разрабатываемого приложения развиваться независимо друг от друга.

2. Независимость от состояния (stateless protocol или «протокол без сохранения состояния). Протокол взаимодействия между клиентом и сервером требует соблюдения следующего условия: в период между запросами клиента никакая информация о состоянии клиента не хранится на стороне сервера. Все запросы от клиента должны быть составлены так, чтобы сторона сервера получила всю необходимую информацию для выполнения запроса. Состояние сессии при этом сохраняется на стороне клиента. Информация о состоянии сессии может быть передана сервером какому-либо другому сервису (например, в службу базы данных) для поддержания устойчивого состояния, например, на период установления аутентификации. Клиент инициирует отправку запросов, когда он готов (возникает необходимость) перейти в новое состояние.

3. Кэшируемая архитектура (Cacheable). Клиенты, а также промежуточные узлы могут выполнять кэширование ответов сервера. Ответы сервера, в свою очередь, должны иметь явное или неявное обозначение как кэшируемые (на определенный период) или некэшируемые с целью предотвращения получения клиентами устаревших или неверных данных в ответ на последующие запросы. Правильное использование кэширования способно полностью или частично устранить некоторые клиент-серверные взаимодействия, ещё больше повышая производительность и расширяемость системы.

4. Единый унифицированный программный интерфейс. Наличие унифицированного интерфейса является фундаментальным требованием дизайна REST-приложений. Унифицированные интерфейсы позволяют каждому из сервисов развиваться независимо от функциональности системы.

5. Многоуровневая архитектура (Layered System). Предполагается выделение в системе иерархии слоев, согласно следующему условию. Каждому компоненту REST-приложения доступны только компоненты непосредственно следующего слоя. Например, при вызове службы PayPal, данной платежной системой будет выполнен вызов службы поддержки кредитных Visa.

6. Представление данных. В качестве представления данных объекта передаются данные в формате JSON, XML или в обоих форматах.

RESTful приложения. Формат обмена данными.

Обмен данными осуществляется в XML (eXtensible Markup Language), JSON (JavaScript Object Notation) или в обоих форматах. За счет своей лаконичности по сравнению с XML формат JSON может быть более подходящим для сериализации сложных структур.

Пример JSON-представления данных об объекте student	**Пример XML-представления** данных об объекте student

{"person" :
{
"firstName": "Иван",
"lastName": "Иванов",
"address": {
"streetAddress": "К.Маркса 12",
"city": "Уфа",
"postalCode": 450000
"phoneNumbers": [
"9271234567",
"9371234567"
]
}
}
}
	<person>
	<firstName>Иван</firstName>
	<lastName>Иванов</lastName>
	<address>
	<streetAddress>К.Маркса 12
	</streetAddress>
	<city>Уфа</city>
	<postalCode>45000</postalCode>
	</address>
	<phoneNumbers>
	<Number>9271234567</Number>
	<Number>9271234567</Number>
	</phoneNumbers>
	</person>
	
	<person name="Иван" lastName="Иванов">
	<address streetAddress="К.Маркса 12"
	code="450000" />
	<phoneNumbers number_1="9271234567"
	number_2="9371234567" />
	</person>

RESTful приложения. Язык описания веб-приложений

Определения сервиса. При интеграции одной системы с другими системами (обслуживаемыми несколькими различными компаниями) данным системам необходимо понимать форматы запросов и ответов. Таким образом необходимо строгое описание взаимодействия, даже если интеграция систем, предполагает возможности общения между командами разработчиков (телефон, электронная почта).

WADL (Web Application Definition Language) – машиночитаемое XML-описание приложений на основе HTTP. Описывается с помощью набора элементов ресурса.

Элемент запроса указывает как представлять ввод, какие типы требуются и какие требуются конкретные заголовки HTTP.

Ответ описывает представление ответа сервиса, а также любую информацию о неисправности.

WADL является REST-эквивалентом SOAP «Web Services Description Language (WSDL), который также может быть использован для описания REST-приложений.

```
<resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
  <resource path="newsSearch">
    <method name="GET" id="search">
      <request>
        <param name="appid" type="xsd:string"
          style="query" required="true"/>
        <param name="query" type="xsd:string"
          style="query" required="true"/>
        <param name="type" style="query" default="all">
          <option value="all"/>
          <option value="any"/>
          <option value="phrase"/>
        </param>
        <param name="results" style="query" type="xsd:int" default="10"/>
        <param name="start" style="query" type="xsd:int" default="1"/>
        <param name="sort" style="query" default="rank">
          <option value="rank"/>
          <option value="date"/>
        </param>
        <param name="language" style="query" type="xsd:string"/>
      </request>
      <response status="200">
        <representation mediaType="application/xml"
          element="yn:ResultSet"/>
      </response>
      <response status="400">
        <representation mediaType="application/xml"
          element="ya:Error"/>
      </response>
    </method>
  </resource>
</resources>
```

RESTful приложения. Использование HTTP-методов

Архитектура REST полностью построена на основе протокола HTTP, а также устанавливает однозначное соответствие между CRUD-операциями (Create, Read, Update и Delete) и HTTP-методами. Согласно данному соответствию, для создания ресурса на стороне сервера используется метод POST, для извлечения ресурса – метод GET, для изменения состояния ресурса или его обновления – PUT, для удаления ресурса – DELETE.

Например, для получения информации о конкретном студенте (по идентификатору, по значению поля name) в формате json необходимо выполнить следующие запросы:

```
GET /students/5
Accept : application/json
GET /firt/acs/student?name=Иван
```

Следующий запрос обеспечивает создание нового студента (с указанием наименования идентификатора факультета и наименования идентификатора кафедры) со следующими полями: имя – Иван, фамилия – Иванов, наименование кафедры АСУ:

```
POST /firt/acs/student {
 "name" : "Иван",
      "surname" : "Иванов",
     }
```

Редактирование фамилии студента осуществляется следующим образом:

```
PUT /firt/acs/student/5 {
      "surname" : "Петров",
}
```

Удаление информации о студенте осуществляется HTTP-методом DELETE :

```
DELETE /firt/acs/student/5
```

RESTful приложения. Предоставление URI

Согласно архитектуре REST, URI рассматривается как самодокументирующийся интерфейс, не требующий пояснения или обращения к команде разработчиков для его понимания, а также получения необходимых ресурсов. Таким образом, структура URI должна быть простой, предсказуемой и интуитивно понятной. Построение URI согласно аналогии структуры директорий позволяет построить иерархию URI – исходящие из одного корневого пути ветвления, отображают основные функции приложения.

Рекомендации на представление структуры URI для RESTful-приложений:

– URI должны быть статичными и не подлежать изменению при модификации ресурсов или реализации сервиса;

– скрывать расширения файлов серверных сценариев (.jsp, .php, .asp) для выполнения портирования приложений на другую технологию без изменения URI;

– содержать только строчные буквы;

– заменять пробелы дефисами или знаками подчеркивания;

– максимально избегать использования строк запросов.

Например, для получения списка студентов необходим URL: `GET localhost:8080/students`

Для получения информации о конкретном студенте URI – `GET /students/1`

Для получения информации о факультетах – GET `localhost:8080/faculty`

Для удаления информации о конкретном студенте – DELETE `localhost:8080/students/1`

Для добавления информации факультете – POST `localhost:8080/faculty`

```
{"code": "ENF",
 "dean": "Декан ЕНФ",
 "name": "ЕНФ"}
```

RESTful приложения. Формат обмена данными JSON

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript. Структуры данных, используемые JSON, поддерживаются любым современным языком программирования, что позволяет применять JSON для обмена данными между различными программными системами.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

1. Набор пар ключ: значение:

– значением может быть любая допустимая форма представления;

– ключом – только строка, регистрозависимость строки определяется документацией программного обеспечения и не регулируется стандартом;

– повторяющиеся имена ключей допустимы (не рекомендуются стандартом), а обработка также происходит согласно документации программного обеспечения программного обеспечения. Например, учет только первого ключа, учет только последнего ключа, генерирование ошибки.

2. Упорядоченный набор значений, который во многих языках программирования реализован как массив, вектор, список или последовательность.

RESTful приложения. Представление значений в нотации JSON

Строка – заданная последовательность, заключенная в двойные кавычки. "Иван" обозначает строку, поскольку является набором символов внутри двойных кавычек, firstName – ключ: {"firstName": "Иван“}.

Объект – неупорядоченный набор пар ключ/значение. Объект начинается с открывающей фигурной скобки и заканчивается закрывающейся. Пример объекта students, где students – ключ, а всё, что находится внутри фигурных скобок – объект:

```
{"students" : {"firstName":"Иван", "lastName":"Иванов"}}
```

Число – в JSON должно быть целым или с плавающей запятой.

```
{"content": {"dateBegin": "2020-10-26",
             "name": "Тестовая услуга",
             "repeatPeriod": 30, "amount": 450.40 … }}
```

Булевый тип также может быть использован в качестве значения:

```
{"content": {"name": "Тестовая услуга",
             "isClientSupport": true, "moment": false}}
```

null – показывает отсутствие информации:

```
{"content": {"name": "Тестовая услуга",
             "statusComment": null, "subprocess": null}}
```

Массив – упорядоченная коллекция значений. Пример массива, состоящего из трех объектов:

```
{"content": {"name": "Тестовая услуга для формирования",
             "subprocess": [
              {"id":  1350, "dateBegin": "2020-10-26", "finish": true},
              {"id":  22, "dateBegin": "2020-12-26", "finish": true},
              {"id":  2, "dateBegin": "2021-04-01", "finish": false} ]}
}
```

RESTful приложения. Представление вложенных объектов JSON

Использование вложенности в JSON-формате позволяет обрабатывать сложные данные. Например, в файле service.json для каждой услуги предусматривается вложенный JSON-объект, который описывает статус услуги конкретной услуги.

```
{"content": [
    {"name": "Тестовая услуга 1",
      "status": {
        "id": 4, "caption": "Действующая"
      }},
    { "name": "Тестовая услуга 2",
      "status": {
        "id": 2, "caption": "Черновик"
      }},
    {"name": "Тестовая услуга 3",
      "status": {
        "id": 3, "caption": "На согласовании"
      }},
    {"name": "Тестовая услуга 4",
      "status": {
        "id": 5, "caption": "Завершено"
      }}]
}
```

SOAP

SOAP (Simple Object Access Protocol) – протокол обмена структурированными сообщениями в распределенной среде. Протокол используется для обмена произвольными сообщениями в формате XML. SOAP может использоваться с любым протоколом прикладного уровня: SMTP, FTP, HTTP, HTTPS и др. Взаимодействие с каждым из этих протоколов имеет свои особенности. Чаще всего SOAP используется поверх HTTP.

Характеристики SOAP.

– протокол SOAP позволяет приложениям взаимодействовать между собой, используя SOAP-сообщения (XML-документы);

– SOAP совместим с любой объектной моделью, поскольку включает функции и методы, которые необходимы для формирования коммуникационной инфраструктуры;

– является независимым от платформы и конкретных приложений, а для его реализации может быть использован любой язык программирования;

– поддерживает практически любой транспортный протокол;

– поддерживает любые методы кодирования данных, которые позволяют приложениям, основанным на SOAP, отправлять в сообщениях SOAP информацию практически любого типа (например, изображения, объекты, документы и т.д.);

– вызов привязывается к HTTP-запросу, так что сообщение отправляется через HTTP-запрос POST;

– сообщение-ответ SOAP представляет собой документ HTTP-ответа, который содержит результаты вызова метода (например, возвращаемые значения, сообщения об ошибках и т.д.);

– для предоставления используется WSDL (Web Services Description Language);

– SOAP для передачи сообщений увеличивает их объем и снижает скорость обработки.

Структура-сообщения SOAP

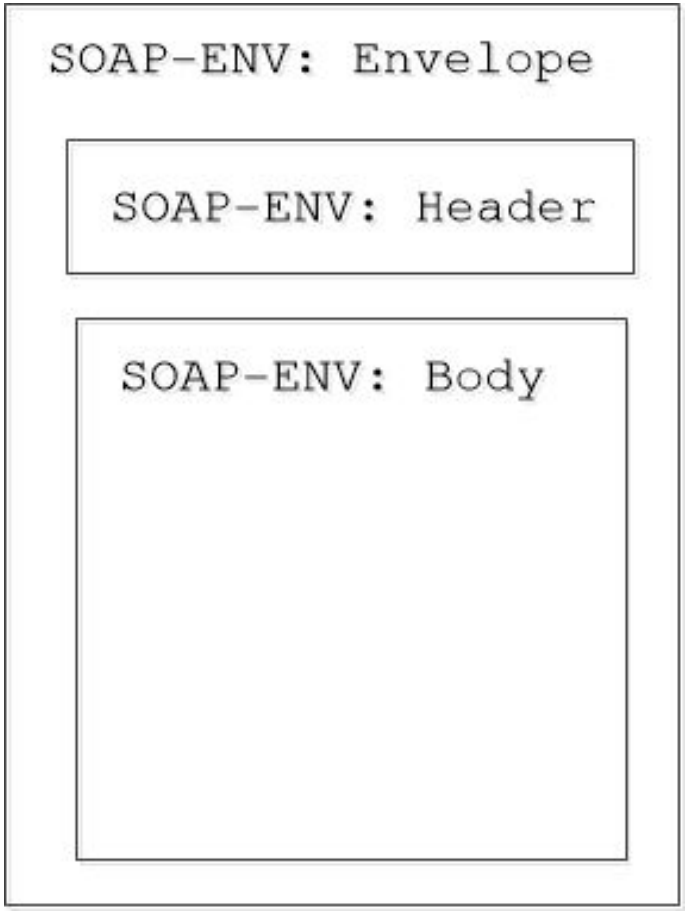
SOAP-сообщение представляет собой XML-документ. Сообщение состоит из трех основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP Body)

`Envelope` – корневой элемент (обязательный), который определяет сообщение (включает дочерний элемент Body) и пространство имен (определяется с помощью ENV и элемента Enveloper) документе. Необходимость данного элемента заключается в следующем. Клиентская сторона определяет, что сообщение получено полностью и готово к обработке.

`Header` – первый для обработки элемент (необязательный), который находится внутри элемента envelope, содержащий атрибуты сообщения. Элементов header в файле может быть несколько.

`Body` – обязательный элемент, содержащий XML-данные для отправки конечному пользователю SOAP-сообщения.

`Fault` — необязательный элемент, возвращающий информацию об ошибках, которые произошли при обработке сообщений. Если при обработке SOAP-сообщения SOAP-сервер сгенерирует ошибку, то дальнейшая обработка будет остановлена, а на сторону клиента будет отправлено SOAP-сообщение, содержащее один элемент Fault с сообщением об ошибке



Пример SOAP-запроса и SOAP-ответа

Данный пример демонстрирует использование SOAP-сообщения, отправляемое на сторону сервера с помощью метода POST. В представленном XML-файле передается наименование обработчика `(getProduct)` и наименование параметра `(productID)` с необходимым значением `(001)`.

```
<?xml version="1.0" encoding="utf-8"?>
    <soap:Envelope>
      <soap:Body>
          <getProduct>
             <productID>001</productID>
          </getProduct>
      </soap:Body>
</soap:Envelope>
```

Пример ответа представляет также XML-файл, который содержит ответ обработчика. Обработчик `getProduct` возвращает данные на основании параметра.

```
<?xml version="1.0" encoding="utf-8"?>
   <soap:Envelope>
      <getProductResult>
         <productID>001</productID>
         <productName>Наименование</productName>
         <description>Описание</description>
         <price>Цена</price>
     </getProductResult>
</soap:Body>
 </soap:Envelope>
```

Сравнение подходов SOAP и REST. «Простота против стандарта»

1. Производительность. SOAP представлен семейством протоколов, стандартов и обработчиков, что предполагает более сложный вариант обработки. REST – является архитектурным стилем, поэтому веб-сервисы RESTful имеют более высокую производительность. При создании системы с высокими требованиями к скорости, REST является отличным выбором.

2. Стандартизированность. SOAP спецификации являются официальными веб-стандартами, которые поддерживаются и разрабатываются Консорциумом World Wide Web (W3C). Безопасность, авторизация и обработка ошибок встроены в протокол. Архитектура REST устанавливает набор рекомендаций, которым необходимо следовать при разработке приложений (например, существование без сохранения состояния и использование кодов состояния HTTP).

3. HTTP. SOAP используют HTTP как транспортный протокол. REST базируется на HTTP, при помощи других протоколов возможности создать коммуникацию различных сервисов нет. Это означает, что все существующие возможности протокола HTTP (кэширование, масштабирование) используются в REST-архитектуре.

4. Используемые форматы сообщений. SOAP возвращает данные только в XML-формате. Использование SOAP для передачи сообщений позволяет размещать метаданные в дескрипторах, что позволяет лучше обрабатывать смешанный контент. Сложность и ресурсоемкость обработки XML-данных увеличивает их объем и снижает скорость обработки, что может привести к снижению времени загрузки страницы. REST допускает многие форматы – HTML, JSON, XML, а также простой текст.

5. Спецификации для описания WSDL/WADL. SOAP предполагает наличие WSDL–строгой и исчерпывающей спецификации для описания. Со стороны REST используется протокол WADL, который не получил широкого применения.

6. Поддержка инструментов. SOAP имеет очень широкую поддержку инструментов. Например, инструменты для определения интерфейса и создания файла спецификации. Кроме того, используемый в качестве представления XML-формат имеет схемы и валидаторы. REST-JSON не имеет такой широкой поддержки инструментов.

7. Обработка ошибок. В SOAP обработка ошибок полностью стандартизована. REST использует коды ошибок HTTP.

8. **SOAP работает с операциями, REST – с ресурсами**, которые представляет URL.