Распределенные вычислительные системы

SOAP. REST

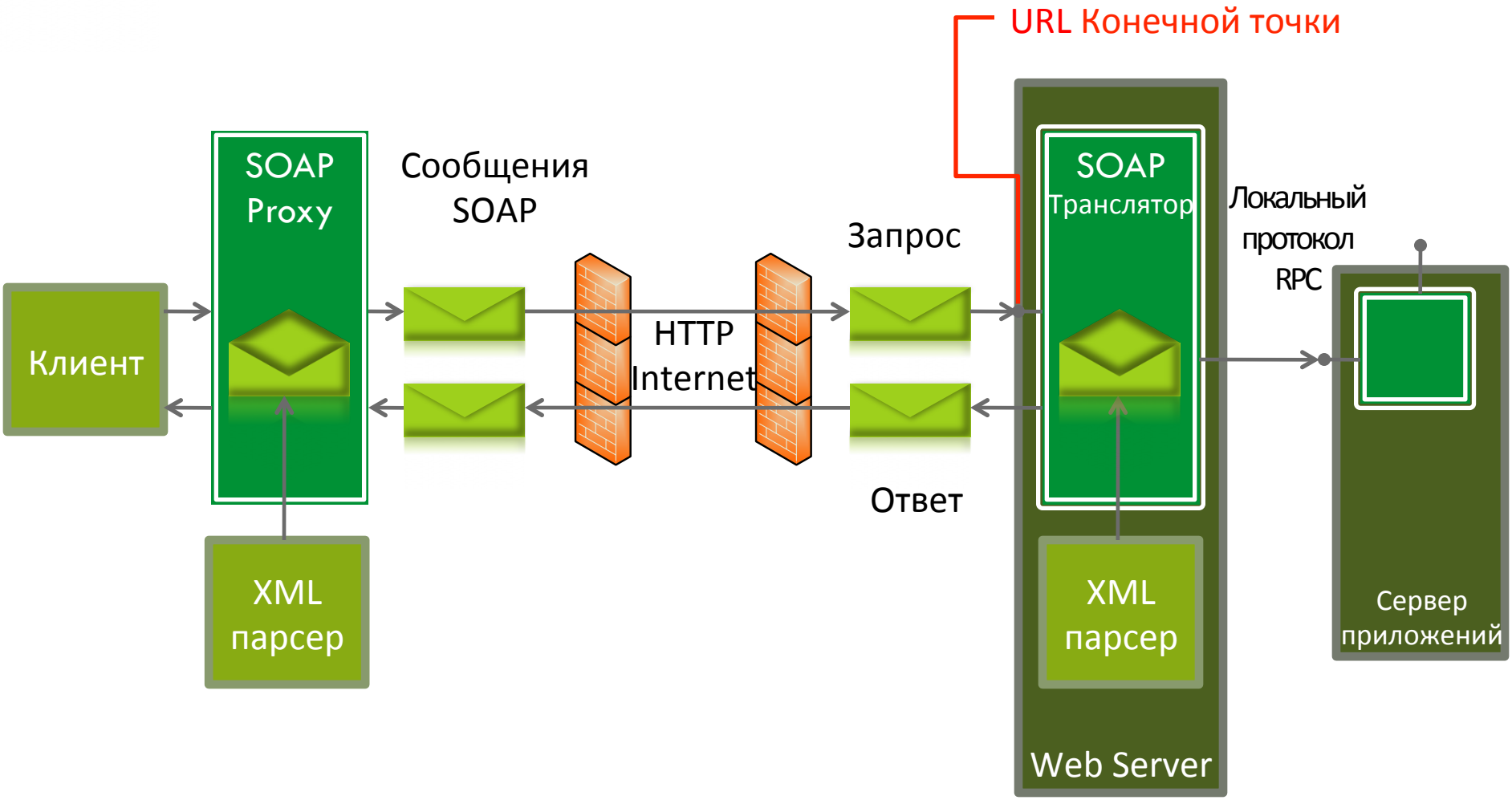
SOAP (уже не только Simple Object Access Protocol)

- SOAP – это протокол, основанный на обмене XML-документами.
- SOAP определяется следующим образом: «SOAP это основанный на XML протокол обмена информацией в децентрализованной распределенной среде.

Сообщение SOAP

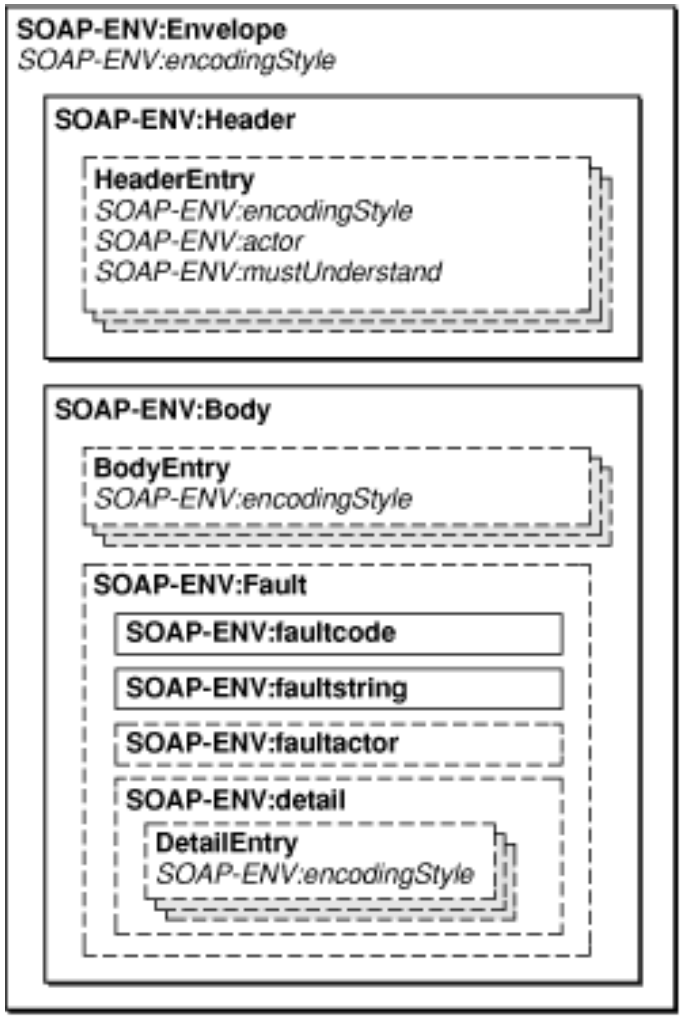
- Сообщением SOAP называют одностороннюю передачу информации между узлами SOAP: от источника к приемнику.
- Сообщения SOAP являются фундаментальным строительным блоком, обеспечивающим возможности более сложных шаблонов взаимодействия:
  - Запрос/ответ
  - «диалоговый» режим
  - и т.п.

Архитектура SOAP



Элементы сообщения SOAP

- Envelope (конверт) – корневой элемент сообщения.
- Header (заголовок) – не обязательный элемент сообщения. Может содержать дополнительную информацию для приложения, обрабатывающего запрос.
- Body (Тело) – обязательный элемент сообщения. Содержит вызовы необходимых методов и передаваемые параметры.



Шаблон сообщения SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap=http://www.w3.org/2001/12/soap-envelope
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

W3C schools website (http://www.w3schools.com)

Пример заголовка SOAP

- В заголовке мы можем ввести новый элемент, не предусмотренный стандартом SOAP. Например, номер транзакции.
- Атрибуты:
  - mustUnderstand – получатель обязан обрабатывать этот элемент;
  - actor – указывает конкретное приложение-получатель при обработке сообщения в цепочке

```
<soap:Header>

   <trans:Transaction
   xmlns:trans="http: //www.host.com/
   namespaces/space/"
   soap:mustUnderstand="1">

   12

   </trans:Transaction>

</soap:Header>
```

Тело сообщения SOAP

Запрос

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productID>12345</productID>
     </getProductDetails>
   </soap:Body>
 </soap:Envelope>
```

Ответ

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productID>12345</productID>
         <productName>Стакан граненый</productName>
         <description>Стакан граненый. 200 мл.</description>
         <price>9.95</price>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
 </soap:Envelope>
```

Связывание SOAP и HTTP

- Передача SOAP сообщений происходит поверх HTTP – протокола посредством запроса POST (начиная со стандарта SOAP 1.2 возможно применение GET)
- Стандартный протокол связи:
  - Клиент посылает запрос
  - Сервер отвечает OK

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap
+xml;
charset=utf-8
Content-Length: nnn
…

HTTP/1.1 200 OK
Content-Type: application/
soap; charset=utf-8
Content-Length: nnn
…
```

Советы по проектированию RPC сервисов

RPC: «Плоский API»

- «Плоский API»: создание интерфейсов, которые выглядят как методы классов

```
@WebMethod(operationName = "ReserveRentalCar")
public RentalOptions ReserveRentalCar (
  @WebParam( name = "RentalCity") String RentalCity,
  @WebParam( name = "PickupMonth") int PickupMonth,
  @WebParam( name = "PickupDay") int PickupDay,
  @WebParam( name = "PickupYear") int PickupYear,
  @WebParam( name = "ReturnMonth") int ReturnMonth,
  @WebParam( name = "ReturnDay") int ReturnDay,
  @WebParam( name = "ReturnYear") int ReturnYear,
  @WebParam (name = "RentalType") String RentalType )
{ // implementation would appear here }
```

- Такой API очень не гибкий, и очень хрупкий, т.к. в нем невозможно даже изменить последовательность параметров без необходимости замены кода во всех клиенах.
- Возможное решение: использование одного аргумента-сообщения.

© Gleb Radchenko

Daigneau, Robert (2011-10-25). Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

RPC: Посредники

- Не смотря на то, что можно формировать RPC-сообщения вручную, на клиентской стороне чаще всего используются посредники (proxy), которые инкапсулируют процесс сетевого взаимодействия. В этом случае, с клиентской точки зрения, процесс вызова удаленного метода не отличается от вызова локального метода.
- Для генерации посредников используются специальные программные генераторы кода, которые на основе описания *клиентского интерфейса* (например, WSDL-документа) создают код вызова удаленной процедуры.
- При изменении интерфейса сервиса, все клиенты должны перегенерировать свои proxy-методы

© Gleb Radchenko

Daigneau, Robert (2011-10-25). Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

RPC: Адресация

- Для обеспечения прозрачной адресации, необходимо чтобы клиенты не имели информации о конкретном расположении сервиса.
- В этом случае его можно перенести или реплицировать (при необходимости)
- Для реализации такой возможности необходимо использовать промежуточный «Соединитель сервисов» (Service Connector), который обеспечивает управление соединениями клиентов с сервисом, перенаправляя запросы от клиентов на реальный адрес используемого сервиса.

© Gleb Radchenko

Daigneau, Robert (2011-10-25). Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

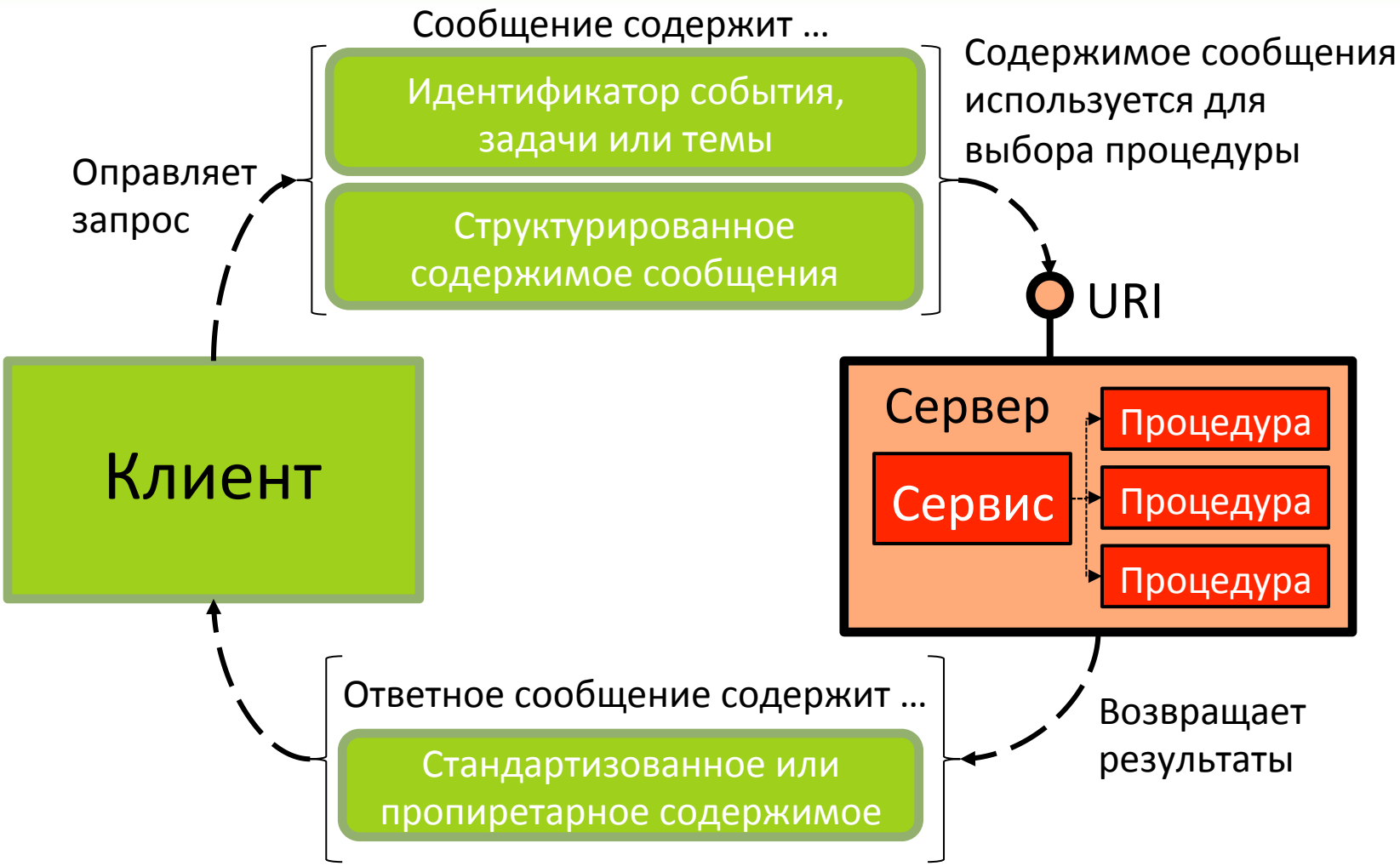
API сообщений

Стили API Веб-сервисов: API сообщений

- Если API сервиса основывается на реализуемых процедурах, сложно обрабатывать изменение такого API, т.к. необходимо менять код на всех клиентах. Если клиенты и сервисы управляются разными организациями, такой подход может стать не приемлемым.
- **Решение -** необходимо определить сообщения для обмена, которые бы не раскрывали интерфейс удаленных процедур.
- Данные сообщения должны содержать информацию об определенных событиях, задачах, которые необходимо выполнить.
- Клиент должен отправить сообщение с этой информацией по указанному URI, а сервер, на основе анализа сообщения должен выполнить определенные процедуры.

© Радченко Г.И., кафедра СП ЮУрГУ

Стили API Веб-сервисов: API для сообщений



© Радченко Г.И., кафедра СП ЮУрГУ

Стили API Веб-сервисов: API сообщений

- Фактически, такой подход строится на использовании тех же технологий, что и RPC API. Отличается метод проектирования системы:
  - Проектирование начинается с определения формата сообщений, которыми обмениваются удаленные компоненты.
  - Они могут быть определены на некотором промежуточном языке, например посредством XML Schema Language или Google Protocol Buffers IDL.
- Сервисы создаются после того, как сообщения были определены.
- Отличием сервисов для API сообщений от сервисов для RPC является количество аргументов в интерфейсе сервиса. Чаще всего, при использовании API сообщений, каждый метод имеет только 1 параметр.

© Радченко Г.И., кафедра СП ЮУрГУ

Сообщения

- Выделяют следующие типы сообщений:
  - *Командные сообщения:* используются для инициализации определенных действий на сервере (например, обработать запись).
  - *Сообщения о событиях:* оповещают получателя о произошедших у клиента событиях (например, закончилось место на складе).
  - *Документы-сообщения:* содержат информацию о тех или иных бизнес-сущностях (например, бланк заказа).
- Ответы сервера также определяются сообщениями: либо полноценным сообщением с ответом, либо простейшим подтверждением о получении сообщения.

© Радченко Г.И., кафедра СП ЮУрГУ

API Сообщений – пример

```
[ServiceContract( Name = "Dropbox" )]
public interface IDropbox {
   [OperationContract(
     Action = "http://acmeCorp.org/Dropbox/Invoice",
     IsOneWay = true)]
   void Invoice( InvoiceMessage invoice);

  [OperationContract(
     Action = "http://acmeCorp.org/Dropbox/Contractor",
     IsOneWay = true)]
  void Contractor( ContractorMessage contractor);
}
```

API Сообщений – пример

```
[MessageContract( IsWrapped = false)]
public class InvoiceMessage {
   [MessageBodyMember]
   public Invoice Invoice { get; set; }
}

[DataContract(Name ="Invoice")]
public class Invoice {
    [DataMember( IsRequired = true, Order = 1)]
    public int ContractorId { get; set; }

    [DataMember( IsRequired = true, Order = 2)]
    public string PurchaseOrder { get; set; }

    [DataMember( IsRequired = true, Order = 3)]
    public DateTime StartDate { get; set; }

    [DataMember( IsRequired = true, Order = 4)]
    public DateTime EndDate { get; set; }

    [DataMember( IsRequired = true, Order = 5)]
    public decimal Hours { get; set; }
}

public class MessageService : IDropbox
{
   void IDropbox.Invoice (InvoiceMessage invoice) {
   // select procedure to process message here
   }
   void IDropbox.Contractor( ContractorMessage contractor) {
   // select procedure to process message here
   }
}
```

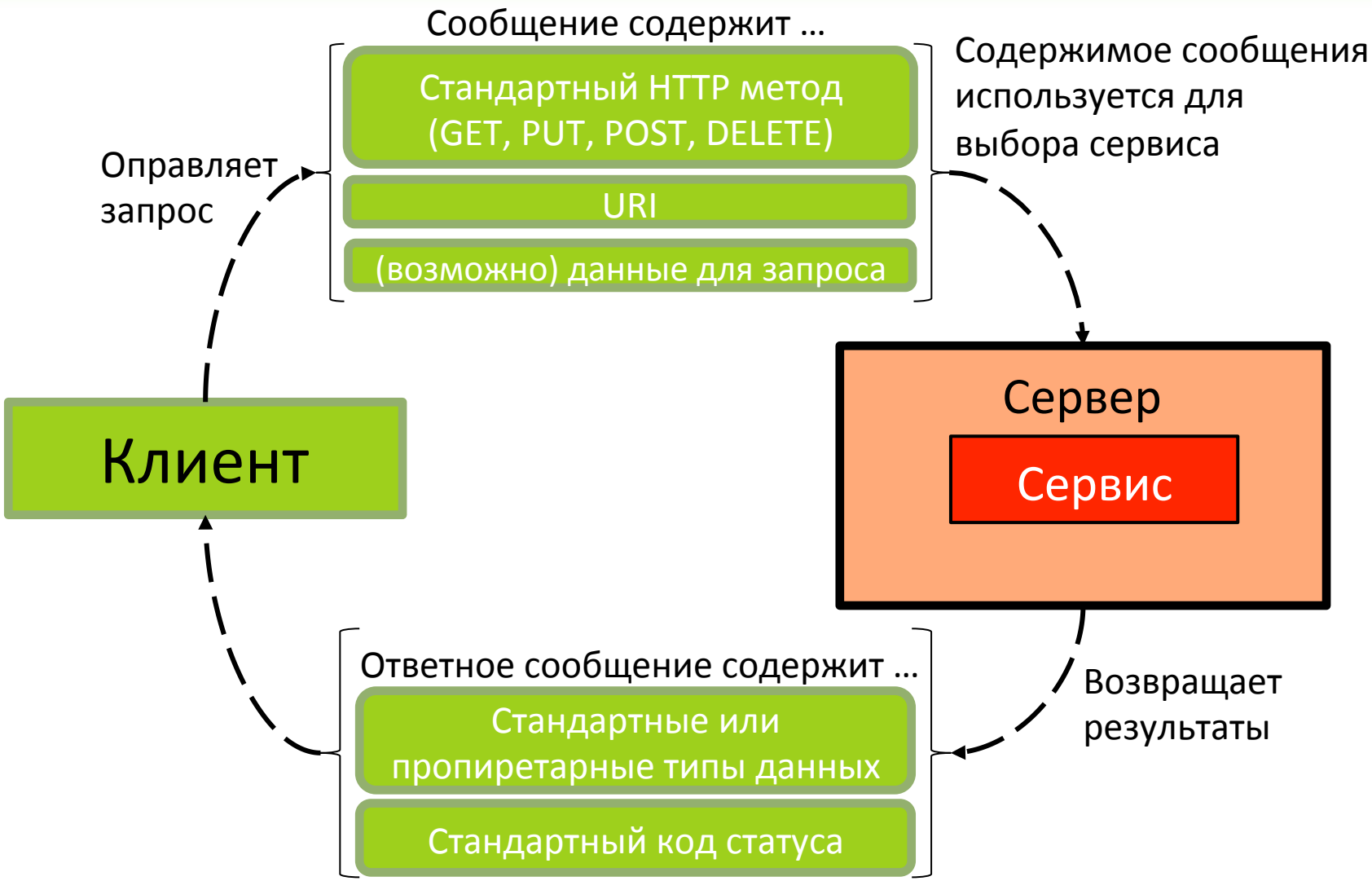
API ресурсов

Стили API Веб-сервисов: API ресурсов

- Если основа сервиса – это использование и модификация ресурсов, расположенных на удаленной системе, использование RPC может привести к разбуханию интерфейса (NewPerson, EditPerson, DeletePerson, GetPerson и др.)
- **Решение -** необходимо использовать стандартные глаголы HTTP (GET, PUT, POST, DELETE) для работы с ресурсами удаленных систем.
- Каждой процедуре, сущности предметной области, файлу назначается URI.
- Клиент должен использовать стандартные команды HTTP с указанием соответствующих URI, а сервер выполнять данные команды и использовать стандартные ответы HTTP там, где это возможно.

© Радченко Г.И., кафедра СП ЮУрГУ

Стили API Веб-сервисов: API для сообщений



© Радченко Г.И., кафедра СП ЮУрГУ

REST

- **Representational State Transfer (REST)** это архитектурный стиль, определяющий процесс работы с ресурсами в сети Интернет посредством стандартных HTTP запросов.
- Был представлен в 2000-м году Роем Филдингом в его докторской диссертации

REST

- Многие веб-сервисы используют сообщения, чтобы сформировать свои собственные API. Они часто реализуют общие логические команды CRUD:
  - Create Read Update Delete
- Тем не менее, может привести к разрастанию интерфейсов, даже в относительно небольших проблемных областях.
- REST предоставляют возможность манипулировать удаленными ресурсами, но избежать прямой зависимости от удаленных процедур, и свести к минимуму необходимость предметно-ориентированных API.
- Протокол HTTP позволяет клиентам относительно легко использовать логику удаленных процедур, при этом изолируя их от базовых технологий. Вместо того чтобы создавать предметно-ориентированный API, можно использовать стандарты, определенные в спецификации HTTP.

Daigneau, Robert (2011-10-25). Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

Все - это ресурс

- Ресурс может быть текстовым файлом, медиа-файлом (например, изображения, видео, аудио), конкретная строка в таблице базы данных, набор связанных данных (например, продукты), логическая транзакция, очередь, загружаемое программа, бизнес-процесс (т.е. процедура) – практически все может быть ресурсом
  - `http://music.site/users/ivan99`
  - `http://music.site/albums/8`
- Коллекция ресурсов - также ресурс
  - `http://music.site/users`

Daigneau, Robert (2011-10-25). Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Pearson Education. Kindle Edition.

ОСНОВЫ REST

- **Не зависимость от состояния**
  - Состояние клиента хранится только на клиенте
  - Вся информация, необходимая серверу для обработки запроса должна быть в запросе (самодокументированные сообщения)
- **Кэширования архитектура**
  - Ответ сервера может быть закеширован у клиента и использован повторно без новых обращений
- **Разделение клиент-сервер (слабосвязанность)**
  - Клиент знает все о интерфейс сервера, но ничего не знает о реализации сервера.

Советы по именованию

- Имена ресурсов должны быть существительными (желательно во множественном числе) а не глаголами.
- Если вы ведете глагол в названии, скорее всего, это ошибка проектирования REST-интерфейса:

```
/getAllDogs
/getAllLeashedDogs
/getDog
/newDog
/saveDog
```

- Вместо глаголов в названии ресурсов, необходимо использовать стандартные методы HTTP

API ресурсов в REST

- PUT используется для **создания** или **обновления** ресурсов.
- GET используется для **получения** представления ресурса.
- DELETE **удаляет** ресурс.
- POST используется для **создания зависимого** ресурса.

GET	PUT	DELETE	POST
=	=	=	=
READ	UPDATE	DELETE	CREATE

Стандартизованный интерфейс элемента

http://example.com/resources/item17

GET	**Получить** состояние элемента
PUT	**Заменить** этот элемент с другим элементом. Если такой элемент не существует, создать такой элемент
POST	Обычно не используется
DELETE	**Удалить** элемент

Стандартизованный интерфейс коллекции

http://example.com/resources

GET	**Перечислить URI** и другую информацию о элементах в данной коллекции
PUT	**Заменить** эту коллекцию другой коллекцией
POST	**Создать** новый элемент в коллекции
DELETE	**Удалить** коллекцию

Стандартные действия REST

Правильный интерфейс REST

`POST /albums` **– добавить новый альбом**

`GET /albums/2` **– получить альбом 2**

`PUT /albums/2` **– обновить альбом 2**

`DELETE /albums/2` **– удалить альбом 2**

Не правильный REST интерфейс

POST /albums/**create**

GET /albums/**show**/2

POST /albums/**update**/**2**

GET /**delete**/albums/2

DELETE /albums/3/**remove**

HTTP: Ответы сервера

- REST позволяют использовать стандартные типы данных и коды состояния.
- Ответы сервера - это HTTP-коды, определяющие статус операции
  - 200 – OK (“Вот ваша одежда”)
  - 201 – Created (“Не потеряйте бирку”)

  - 400 – Bad request (“Бирка сломана”)
  - 403 – Forbidden (“Вам запрещено это делать”)
  - 404 – Not found (“На вешалке с такой биркой ничего нет”)
  - 500 – Server error (“У нас обед”)