REST И WS-*

Маниакальный веблог, 02.11.2008

Этот пост лежит у меня в черновиках уже очень давно. Все боюсь флейм нездоровый породить :-). Но все же тема мне представляется важной, и в рамках борьбы со старинными хвостами я его таки вот написал. Заранее извиняюсь, если это "и так все знают", тема действительно уже не нова.

Также должен оговориться, что я не буду подробно описывать, что такое REST сам по себе, потому что это отдельный глубокий и труднопередаваемый вопрос. Вместо этого сошлюсь например на аналогичную этой статью классика темы Джо Грегорио "REST and WS-*", там он про REST пишет более подробно. А эта статья -- о том, почему REST и WS-* нельзя сравнивать, почему их все-таки сравнивают, и почему первое лучше второго. Да, никакой объективности от меня тут ждать не надо :-).

ПРЕДМЕТНАЯ ОБЛАСТЬ

Речь пойдет о построении веб-сервисов: сайтов, открытых в публичный веб и явно предполагающих машинную обработку. Определение слегка общее, поэтому стоит подчеркнуть, что к ним не относится:

- сайты на вебе, предлагающие только человекочитаемый HTML-интерфейс, несмотря на то, что теоретически работу с ними можно автоматизировать
- интегрированные клиент-серверные системы, использующие для общения внутри себя протокол HTTP, но не предлагающие публичного интерфейса на вебе

REST (КРАТКО)

Representational State Transfer -- это набор общих принципов построения веб-сервисов с определенными приоритетами: масштабируемость, независимость от платформы, расширяемость. Эти принципы были сформулированы Роем Филдингом, одним из разработчиков HTTP 1.1, в

своей диссертации, посвященной как раз этим вопросам.

Поскольку REST -- это архитектура (или точнее "архитектурный стиль"), то в природе, по идее, не может существовать таких вещей, как "протокол REST" или "библиотека REST". Хотя библиотеки и есть, но они, на самом деле, реализуют не шарообразный REST а обычно что-то более конкретное с использованием REST-принципов. Также и с протоколом: обычно все имеют в виду самый распространенный протокол общего назначения, реализованный на REST-принципах -- HTTP 1.1. Однако эти принципы можно применять и с другими протоколами, например с той же CORBA. А также эти принципы можно применять не целиком: ваша системы не обязана быть абсолютно чисто REST'овой, а может быть таковой местами.

Основной смысл REST'а, таким образом, в том, что он описывает то, в каких терминах надо думать, чтобы делать клиент-серверные системы, работающие в вебе, и не нарушающие общей экосистемы.

WS-* (КРАТКО)

WS-* -- это обобщенное название набора спецификаций, большая часть которых начинается с букв "WS" (хотя самая базовая -- SOAP -- с них не начинается). "WS" означает буквально "Web Services" и, надо сказать, является источником заблуждения, что веб-сервисы -- это то, что построено именно на этом стеке.

Одним из приоритетов подхода, заложенного в WS-спецификации, является автоматизация публикации и доступа к объектам программных систем в виде интерфейса, доступного через HTTP. В идеале программист должен работать с чем-то на другом веб-сервисе так же прозрачно, как и с объектами в памяти его собственной программы. Пересылку данных, проверку типов и прочий сервис обеспечивает WS-*. Для достижения этого спецификации не абстрактны, а наоборот, очень конкретны и детальны, и, что важно, подразумевают не прямое программирование по описанным протоколам, а использование инструментов среды программирования, генерирующих необходимую обвязку.

Стоит добавить, что главными продвигателями WS-* являются Microsoft и IBM.

СРАВНЕНИЕ

Из кратких описаний должно быть понятно, что впрямую сравнивать эти вещи нельзя. Во-первых, они просто разного уровня: одно -- архитектурные принципы, другое -- набор протоколов с реализациями. Во-вторых, они не взаимоисключающи: вполне можно делать системы на WS-* протоколах с использованием REST-принципов. У меня в голове как-то родилась аналогия, что сравнение REST и WS-* -- это примерно то же, что сравнение функционального программирования и Visual Studio.

Но тем не менее... Тем не менее, сравнение все таки напрашивается, пусть даже и на эмоциональном уровне. И оно таки имеет смысл!

REST и WS-* являются *ключевыми понятиями* для совершенно разных подходов к созданию веб-сервисов. И неважно, что они сами находятся на разных уровнях, важно, что они подразумевают очень конкретный инструментальный стек, в рамках которого работают:

	REST	WS-*
Архитектура	REST	RPC-стиль
Транспортный протокол	HTTP	HTTP
Протокол описания сообщений	HTTP	SOAP
Сервисные протоколы (авторизация, кеширование, проксирование)	HTTP	WS-*
Описание форматов	MIME-типы	XML Schema
Глобальная система идентификации	URI/URL	Отсутствует
Описание интерфейсов сервисов	Отсутствует	WSDL
Инструментарий	Низкоуровневый (парсинг XML, JSON и др.)	Высокоуровневый (абстракция доступа к удаленным объектам)

Все это нуждается в разъяснениях.

АРХИТЕКТУРА

Архитектура REST диктует, что

- Система представляется отдельными ресурсами со своим состоянием и не хранит состояние клиентского процесса для *масштабируемости*.
- Ресурсы представляются стандартными mime-типами для *независимости от платформы* и *самоописываемости*.
- Ресурсы имеют универсальный интерфейс, идентифицируются собственными URL'ами и связываются только через явное указание ссылок и форм в их представлениях для обеспечения *интеграции* между сервисами.

WS-сервис никакой конкретной архитектурой не обладает. Снаружи -- это "черный ящик", у которого есть набор специфичных методов, которые можно вызывать. Что у него внутри, известно только его разработчику. Также исключительно на совести разработчика находится масштабируемость сервиса, поскольку WS-стек тут никаких решений не подсказывает. Чаще всего поэтому WS-системы предоставляют RPC-подобный интерфейс, потому что это то, что получается само собой, если над ним особенно не задумываться.

ПРОТОКОЛЫ

Оба стека используют HTTP, но сильно по-разному.

HTTP создавался как каноническая реализация REST-принципов, и цели этой достиг. Теперь для большинства REST-ориентированных система HTTP -- это все: хлеб, масло и колбаса сверху. Или более формально -- и транспорт, и метаданные и сервис. А именно:

- для доступа к ресурсу используется небольшое количество методов HTTP (GET, PUT, DELETE, POST и др.) в соответствии с их изначальным смыслом
- для передачи метаданных используются HTTP-заголовки
- кеширование во всех видах обеспечивает HTTP
- проксирование -- HTTP
- авторизация -- тоже через HTTP, причем не только базовая, но и какая

угодно другая

WS-* же использует HTTP фактически только как транспорт для передачи SOAP-вызовов. И, собственно говоря, заявляется, что HTTP для WS-* -- только один из транспортов, и можно использовать другие. Для всего остального в стеке придуманы свои аналоги. В частности, SOAP-сообщение состоит из заголовков и тела, и именно эти заголовки, а не HTTPшные, имеют значение. Куча остальных фич реализована различными WS-протоколами, про которые я подробно распространяться не буду за незнанием, а лучше еще раз сошлюсь на источник.

ФОРМАТ ДАННЫХ

Одно из основных положений REST'а -- использование для описания форматов данных MIME-типов. Лучше всего, если сервис использует уже известный и описанный MIME-тип. Например, если вы выдаете какие-то календарные события, то желательно их выдавать в формате iCal с заголовком `Content-type: text/calendar`. Еще одна возможность -- использование XHTML с микроформатами, потому что это тоже описанные данные, и под них есть парсеры. И если только совсем не получается найти подходящий MIME-тип, тогда только можно придумать свой собственный XML и очень тщательно его описать. Еще желательно его в IANA зарегистрировать, но это уж как получится :-).

WS-стек не имеет описания формата передаваемых данных *в целом*. Точнее, форматом является некий абстрактный SOAP-конверт, но он не означает ничего конкретного -- это просто транспорт. Вместо этого описываются типы параметров, которые передаются внутри этого конверта. Строки, числа, даты. При передаче они сами по себе не связаны никакой семантикой, и что с ними делать, определяется тем, какой метод какого объекта вызывается с этими параметрами.

Это различие, в сочетании с универсальным интерфейсом, делает REST-сообщения самоописывающимися, а SOAP-сообщения -- нет. Даже если оно содержит только известные типы данных, смысл его разный для каждого конкретного вызова каждого конкретного объекта.

ГЛОБАЛЬНЫЕ ИДЕНТИФИКАТОРЫ

В REST есть понятие глобального идентификатора, которым по сути является хорошо всем известный URL (точнее, URI, но разница

несущественна). Прелесть его наличия в том, что он создает единый интерфейс ссылок на ресурсы между разными сервисами, которые друг про друга ничего не знают. Например, REST-сервис, работающий с картинками, может принимать к себе как непосредственно представления картинок с MIME-типом `image/something`, так и URL'ы картинок, которые могут быть расположены где угодно.

Еще один важный плюс -- это то, что одними только URL'ам сервис может описывать свой собственный протокол. URL'ы, встречающиеся в выдаваемых представлениях ресурсов, как в телах, так и в заголовках -- это новые ресурсы, в которые клиент может стучаться для того, чтобы делать с сервисом что-то еще. И поскольку формирование URL'ов является обязанностью сервиса, это позволяет ему свободно менять схему своих URL'ов, не боясь сломать клиентов, которые всегда работают только с готовыми URL'ами, а не составляют их по каким-то правилам.

В WS-стеке глобальных идентификаторов нет, так как нет самого понятия разных ресурсов. Все операции с сервисом делаются только через одну точку по уникальным для этого сервиса правилам.

ОПИСАНИЕ ИНТЕРФЕЙСА

Для описание интерфейсов WS-сервисов есть развитый язык WSDL, который подробно описывает, какие у сервиса есть объекты, какие у них есть методы, и какие у них есть параметры. Формат сложный, с-трудом-человекочитаемый, и рассчитан на то, что он будет генерироваться средствами IDE автоматически по реально существующим объектам системы.

В REST-сервисах никакой прямой аналог WSDL не нужен, потому что все ресурсы обладают уже заранее известным интерфейсом.

Стоит заметить, что WSDL в WS-стеке играет примерно ту же роль, что наличие URL'ов в REST-системах -- описывает то, что можно делать с сервисом. Хотя очевидна разница в акценте. WSDL полезен клиенту по сути только для автоматической генерации стабов, и ничего не говорит о семантике вызовов. В то время как по наличию URL'ов в REST-ресурсах, описанных в подходящих местах, можно делать конкретные выводы о том, как их использовать.

Есть еще одна проблема с автоматической публикацией интерфейсов объектов наружу. Интерфейсы объектов внутри хорошо

спроектированной системы гарантируют только корректность обращений к самим объектам, но не всегда определяет, в каком, грубо говоря, порядке их нужно вызывать по бизнес-логике. Например в неком форуме могут быть внутренние объекты "Топик" и "Статья", и бизнес-правило, что в топике должна быть хотя бы одна статья. Простая публикация интерфейсов объектов наружу не даст гарантии, что пользователи будут создавать статью к каждому созданному топику. Поэтому для внешнего мира обычно все равно нужно создавать отдельный слой кода, который такую логику поддерживает, и без этого одна только публикация объектов бесполезна.

ИНСТРУМЕНТАРИЙ

Инструментарий -- по сути главная часть WS-стека. Его задача -- создать в IDE программиста клиентского приложения удобную абстракцию для доступа к серверным объектам. Соответственно, без автоматической генерации и парсинга WSDL-описаний и SOAP-сообщений тут делать нечего. Я не возьмусь судить, насколько это все выходит удобным, но то, что выглядит это все довольно Солидным и Сложным -- это точно. Есть даже точка зрения, что вся эта великая концепция была как раз и рождена для создания рынка инструментария. Большой минус у этого подхода в том, что она работает только на тех платформах, которые интересно обслуживать производителям инструментов.

Для работы с REST-системами никакого одного набора инструментов нет и быть не может. Вместо этого вам предлагается пользоваться вещами, подходящими для конкретных задач. Нужно общаться с HTTP -- возьмите хорошую HTTP-библотеку, коих много. Нужна более сложная авторизация -- есть библиотеки для digest-авторизации, OAuth'а и т.д. Парсинг XML, JSON и прочих форматов и подформатов -- тоже в библиотеках. Каждую из таких небольших библиотек вполне может написать один человек, а не целая компания.

КУЛЬТУРНОЕ РАЗЛИЧИЕ

Все технические различия в подходах, акцентах и назначениях между REST и WS-* складываются в хорошо ощутимое различие культурное.

WS-стек -- это инструмент для разработчиков, которым нравится работать в среде, которую для них создает большой солидный вендор, которому

они сильно доверяют. Их не беспокоят мелочные вопросы типа "как работают вещи внутри", им неважно, будет ли их сервис удобно программировать кому-то на странном маргинальном языке программирования. Им интересно, чтобы у каждой решаемой задачи уже был кем-то одобренный верный способ решения. А если он не решает задачу, то разработчик вроде как все равно не виноват -- надо просто купить новую "более enterprise" версию решения, в которой это все точно уже сделано.

REST -- явный любимец open source культуры. Для людей, которые не будут спать ночами, придумывая, как покрасивее описать интерфейс системы в виде stateless-ресурсов. Которым важно знать, что если им захочется написать сервис на Erlang'е, они смогут опубликовать ресурс в XML-виде, не дожидаясь, пока кто-то напишет SOAP и WSDL генераторы для Erlang'а. И которым в принципе не все равно, насколько *незлым* будет их сервис по отношению к другим жителям веба.

И вот поэтому я утверждаю, что:

> WS-* — ПЛОХОЙ СПОСОБ ДЕЛАТЬ ВЕБ-СЕРВИСЫ. ОН РЕШАЕТ НЕНУЖНЫЕ ПРОБЛЕМЫ, ЗАТО НЕ РЕШАЕТ НУЖНЫХ.

ССЫЛКИ

При написании этой статьи у меня скопилось некоторое количество ссылок по теме. Привожу их здесь в произвольном порядке:

- MochiAds - Flash Game Ad Network -- распределенная система показывания рекламы в играх, использующая HTTP для связи серверных компонентов на Erlang и клиента на Python.
- REST, I just want it и Explaining REST to Damien Katz -- реакции на статью Дамиена Каца о том, как он не понимает REST.
- Learning REST -- подборка ссылок Тима Брея с различными объяснениями концепций REST.
- REST APIs must be hypertext-driven -- недавняя нашумевшая статья Самого Филдинга о том, как многие создатели REST API не схватывают принципа ссылочности.

КОММЕНТАРИИ: 57 (ОСОБО ЦЕННЫХ: 1)

Александр Кошелев

Свершилось! Поздравляю;-)

hodzanassredin

Да в этом году был на Microsoft Platform 2008 так там был Девид Чаппел. У него тема была про Рест вс ВС сервисы. Если кратко то резюме было ВС для ентерпрайз а рест для всех остальных кому не требуется возможности типа транзакций и т.п. Хотя думаю в дальнейшем ВС стандарты будут брать на вооружение многое из рест. Примером этому могут служить Wsdl 2 и Soap 1.2

Stump

Заметны хорошее знание автором REST и просто удручающие пробелы в противопоставляемой области.
Довольно убедительно звучат размышления о культурных различиях, но когда дело доходит до технических деталей все становится достаточно неубедительно.
Но зато какой вывод в конце :)

maxp

Не имею практического опыта с WS-*, но чисто интуитивно очень согласен с

""Есть даже точка зрения, что вся эта великая концепция была как раз и рождена для создания рынка инструментария.""

и с

""Он решает ненужные проблемы, зато не решает нужных"".

xaratt

Немножко занудства. В табличке сравнения вместо

> Архитектура REST

может лучше написать

> Архитектура CRUD

?
Всяко логичнее смотрится возле "RPC-стиль".

культурное различие | λ в жизни

> [...] http://softwaremaniacs.org/blog/2008/11/02/rest-vs-ws/ [...]

dobrych

Обожаю эмоциональные посты :-)

Иван, достаточно четко описаны все важное про REST vs. WS-* Теперь будет куда отправить интересующихся!

PS: enterprise в сад!

Иван Сагалаев

> Если кратко то резюме было ВС для ентерпрайз а рест для всех остальных кому не требуется возможности типа транзакций и т.п.

О-о-о! Этот микрософтовский FUD про "энтерпрайз" и "транзакции" уже оскомину набил :-). Они всегда эти слова вспоминают, когда надо что-нибудь про REST поопровергать.

На самом же деле, как раз из-за того, что REST -- это не коробочное решение, а архитектурные принципы, вся эта чушь не выдерживает никакой критики. Нет совершенно ничего в REST-принципах, что мешало бы реализовать транзакции. Один из

примеров, как это делается, приводят в своей книге RESTful Web Services Сэм Руби и Леонард Ричардсон. КНижку, кстати, я очень-очень рекомендую всем прочитать.

> Заметны хорошее знание автором REST и просто удручающие пробелы в противопоставляемой области.

Сергей, так вот в этом и проблема. Про WS-стек невозможно найти и прочитать ничего вменяемого. Большинство убеждений как от Микрософт, так и от ее сторонников не выглядят убедительно как раз из-за того, что перемешаны с религией и идеологией по самую крышу. Я технарь, я очень плохо воспринимаю вещи, которые мне пытаются впарить, а не объяснить.

> может лучше написать
>
> Архитектура CRUD

Может быть... Хотя CRUD мне представляется менее конкретной концепцией, чем REST. В общем, не так важно :-)

Макс Лапшин

Ваня, тут смотри какое дело. Главное различие между REST и WS- в том, что WS пользуется HTTP как транспортом (точно так же, как и CORBA может бегать по HTTP), а уровень приложения завертывается в следующий протокол, шныряющий поверх HTTP пакетов,

а REST подразумевает, что у тебя HTTP — и есть **протокол уровня приложения**.

Таким образом для REST-а главная библиотека — просто HTTP библиотека.

Однако, чувствую я, что ты не иопся с C++ =) Для него WSDL нужен для того, что бы генерировать стабы, что бы генератор этих стабов сам сгенерил код маршалинга JSON/XML ответов.

Именно поэтому для «энтерпрайза», т.е. явы, C# и C++ отсутствие

протокола описания формата ответов (читай XSD схемы ответов) обуславливает то, что при обмене объектами в которых по 1000 полей, пользоваться REST-ом попросту невозможно.

Т.е. WSDL нужен прежде всего что бы импортировать в статически-типизированный язык объектную модель, экспортируемую по слабо-типизированному протоколу.

Artemy Tregubenko

У WDSL есть обратная сторона: очень интересно работать с вебсервисом, WDSL которого не соответствует действительности. Весь клёвый инструментарий внезапно оказывается способным только пискнуть «ошибка» ;)

Можно сказать, что это редкий случай, но я это встречал в тот единственный раз, когда мне пришлось с ним работать, интегрируя сайт с Google Adsense.

Artemy Tregubenko

Чёрт, нужно уточнить: было время, когда WSDL **Google** Adsense не соответствовал реальности. Казалось бы, куда уж крупнее энтерпрайз.

Voidus

Эм... А как же WADL? Вполне себе язык описания веб-приложений :)

Правда я ни разу не видел его применения в в реальной жизни :)

Stump

> Сергей, так вот в этом и проблема. Про WS-стек невозможно найти и прочитать ничего вменяемого.

Ага. Я как раз и расчитывал тут почитать хороший технический

анализ двух подходов. Но увидел больше эмоции.
Иван, вот ты сравниваешь REST и WS-*. Ну звездочку надо бы раскрыть.
Расказать, например, о WS-ReliableMessaging, WS-MetaDataExchange или WS-security, и о том как с этими аспектами обстоят дела у REST.
Говоря о идентификаторах почему не упомянуть WS-Addressing? Это очень важная спецификация на которой строятся многие другие протоколы WS.

О неточностях:
- Стек WS не привязан к транспорту HTTP - это важно.
- WS не ориентирован на RPC стиль, WS это в первую очередь message ориентированный стек. Это тоже важно (первоначальные реализации в .Net и Java действительно были зациклены на RPC стиле и это принесло много вреда.).
- Инструментарий. В стеке WS нет ничего об инструментарии. Ничто не мешает сформировать зпрос с SOAP пакетом ручками, хотя конечно утомительно перечислять все эти xnlns-ы. Ты говоришь о том что WS ориентирован на использование готовых фрэймворков, и это верно. Но этот факт лишь следствие того, что стек WS ориентирован на обеспечение автоматической обработки сообщений. Имея только URL конечной точки можно получить всю информацию о контракте сервиса и автоматически сгенерировать код стаба для конкретной плаформы. Имея само сообщение, можно узнать правила его обработки, маршрутизации и т.д. Именно этим целям служит пугающее количество различных непонятных WS-* протоколов.

Ну и наконец выводы, которые меня больше всего развеселили. Пост который начинался вроде как "технический" завершился, как "эмоциональный". "Злые" и "добрые" сервисы, это конечно забавно, но хотелось услышать что-нибудь на счет того, для каких задач лучше REST и наоборот.

Я вовсе не противник REST и не апологет WS-*, хотя работал в основном с последними. И поэтому мне было интересно почитать профессиональное сравнение двух стилей от человека хорошо знакомого с REST. Но я разочарован.

Макс Лапшин

Артём, ну это, конечно, грустная ситуация. Наверное, WSDL всё таки *должен* быть динамически генерируемым без участия человека, что бы такого не происходило.

Вопрос именно в том, что по REST-овому API вообще ничего автоматом не сгенерить. Впрочем, оно обычно используется там, где вообще ничего генерить автоматом не надо.

Artemy Tregubenko

Честно говоря, я пристрастен к SOAP не только из-за собственных проблем. На меня ещё сильно повлиял небольшой юмористический диалог «S stands for Simple», который сейчас пропал из инета : (На него всё ещё много ссылок, в том числе из негативного отзыва о SOAP разработчика того самого апи, с которым мне пришлось помучиться. Там он, среди прочего, приводит аргумент против динамической генерации WSDL: по его словам, любое изменение в ней требует пересборки всех компилируемых клиентов.

Кстати, мифическое «не привязан к транспорту HTTP» в том тексте тоже упоминалось, со словами в духе «покажите мне хоть одну способную на это библиотеку» ;)

Eugene Lazutkin ★ Особо ценный комментарий

JSON упомянут, но больше вскользь, хотя он предоставляет многие "недостающие" средства и активно пользуется в имплементиции современных REST приложений для представления структурированных данных. Причём речь идёт как о создании специализированых систем, так и систем общего назначения (примеры: CouchDB, Persevere).

Например, весьма интересен JSON SMD, который сам по себе базирован на JSON Schema. JSON SMD успешно описывает REST сервисы и позволяет исключить "устные знания" необходимые для написания клиентов под опубликованный сервис — против чего активно выступает Филдинг называя это "out-of-band information".

Вообще JSON и основанные на нём инструменты активно развиваются. Желающие могут погуглить JSON Path, JSON Query, JSON Referencing чтобы увидеть что делается и какие проблемы пытаются решать разработчики.

Иван Сагалаев

> Ага. Я как раз и расчитывал тут почитать хороший технический анализ двух подходов. Но увидел больше эмоции.

Ну с этим я расквитался еще во вступлении :-). Но эмоциональность поста не умаляет сути, мне кажется. Я постарался объяснить, за что именно я не люблю WS-стек.

> Иван, вот ты сравниваешь REST и WS-*. Ну звездочку надо бы раскрыть.
> Расказать, например, о WS-ReliableMessaging, WS-MetaDataExchange или WS-security

Я не могу это раскрыть по той причине, что не знаю этих вещей. Тем не менее, да, я берусь судить об этих вещах, имея о них даже поверхностное представление, и мне не кажется это странным. Например, человеку не обязательно реально прыгать с пятого этажа, чтобы наверняка предполагать, что он разобьется. Так же и с WS-технологиями. Одно наличие большого числа сложно написанных спецификаций уже показывает ущербность этого подхода, потому что REST-принципы ясно показывают, что собственно главной цели -- создания полезных, надежных, масштабируемых веб-сервисов -- можно достигать и без этой сложности. Значит она просто лишняя.

> Стек WS не привязан к транспорту HTTP - это важно.

Я об этом, кстати, упомянул. Опять таки, едва ли это можно записать в плюсы к WS. HTTP -- хороший протокол. Зачем нужно было делать дополнительный уровень абстракции, заново реализующий все, что в HTTP уже есть? Только ради того, чтобы

заявить, что эта штука "transport agnostic"? Но это же фича только ради галочки...

> WS не ориентирован на RPC стиль, WS это в первую очередь message ориентированный стек.

Я не вижу особенной разницы между термином "вызов процедуры" и "передача сообщения" до тех пор, пока это сообщение не self-descriptive. Именно в этом принципиальная разница между открытой архитектурой REST-системы с универсальным интерфейсом и *стандартными* форматами сообщений и WS-сервисами, каждый из которых не похож на другие ровно ничем.

> Инструментарий. В стеке WS нет ничего об инструментарии. Ничто не мешает сформировать зпрос с SOAP пакетом ручками, хотя конечно утомительно перечислять все эти xnlns-ы

Хм... Ну формально можно говорить, что и бинарные файлы ничто не мешает писать вручную, правда? Нет, правда в том, что делать SOAP вручную просто непрактично, и так никто не делает.

> Имея только URL конечной точки можно получить всю информацию о контракте сервиса и автоматически сгенерировать код стаба для конкретной плаформы.

Вот это red herring. Автоматическое получение стаба -- это *не сервис*. Потому что если бы на той стороне не было WS-сервиса, не было бы нужды в самом стабе :-). Про это я и говорю, когда пишу, что WS-стек решает ненужные проблемы.

А то, что он не решает нужных -- это про то, что WSDL-описание не дает никакой информации о семантике системы. Оно мне расскажет, что у сервиса есть объект "Worker", у которого есть метод "getWorkerState". Но *что* это такое? Когда и зачем его вызывать? Про это узнать невозможно без подробного знакомства с этой системой. Теперь посмотрите на REST-сервис. Я могу не знать про него ничего, но если с какого-то URL'а, я получил, скажем, документ с типом application/atom+xml, то я знаю, что это такое, и знаю, например, могу ли я это редактировать, где

именно это делать и как именно. И для этого самому сервису не надо документировать реально ничего. То есть даже в обещании discovery из одной точки REST-ориентированные системы выигрывают у WS очень сильно.

> но хотелось услышать что-нибудь на счет того, для каких задач лучше REST и наоборот

Я описал предметную область в самом начале. Если вы делаете публичный веб-сервис, его надо делать REST'овым. Всем будет легче. Где применять WS-* мне сказать определенно трудно, но думаю, что оно подойдет для соединения нескольких компонентов одной большой системы, написанной в одном организационном стиле и на Windows-технологиях. То есть это внутрисистемная технология, а не внешняя.

> Но я разочарован.

Что ж... Нельзя понравиться всем :-).

Станис Шрамко

Арти, ну из экзотики был, например, SOAP, туннелированный через SMTP.

bialix

Эта статья сильно похоже из разряда "все открытое и опенсурсное" лучше "закрытого и коммерческого". Тобишь опять же все на эмоциях. Против коих сам же Иван и выступает.

Однако, эта мысль Ивана:

> Я могу не знать про него ничего, но если с какого-то URL'а, я получил, скажем, документ с типом application/atom+xml, то я знаю, что это такое, и знаю, например, могу ли я это редактировать, где именно это делать и как именно.

свидетельствует о том, что речь идет всего лишь про стандартизацию интерфейсов. Однако это вступает в противоречие с фразой про форматы в статье (выделение мое):

> Еще одна возможность -- использование XHTML с микроформатами, потому что это тоже описанные данные, и под них есть парсеры. **И если только совсем не получается найти подходящий MIME-тип, тогда только можно придумать свой собственный XML и очень тщательно его описать.**

Т.е. вопрос стоит просто в стандартизации форматов данных, ибо формат данных -- это все. И все равно возможна ситуация, когда стандартных средств окажется недостаточно. Поэтому когда вы сравниваете:

> у сервиса есть объект "Worker", у которого есть метод "getWorkerState".

Каким будет аналог для REST? Я имею в виду как описать данные работника в REST? А то получилось снова сравнение яблок и апельсинов (объект Worker vs. application/atom+xml).

И почему для REST вы допускаете возможность придумать новый нестандартный тип и тщательно его описать, а для WS-системы подразумевается, что описание данных иметь невозможно?

Это выглядит странно, а мой коментарий -- всего лишь ИМХО как человека далекого от построения веб-сервисов.

Artemy Tregubenko

> И почему для REST вы допускаете возможность придумать новый нестандартный тип и тщательно его описать, а для WS-системы подразумевается, что описание данных иметь невозможно?

я думаю, тут ответ прост: иногда придумать и стандартизовать новый тип данных можно, но не стоит так делать каждый раз для

каждого интерфейса. А чем меньше нестандартных типов используется в веб-сервисе, тем проще его будет исопльзовать

ещё в свежей с утра голове появился вот какой аргумент. WSDL — это жёсткая фиксация интерфейса. Сами объекты, которые бегают туда-сюда, ещё могут быть гибкими, но вот интерфейс не позволяет отклонений. А весь мир двигается в сторону более расслабленной нечёткости, начиная с XML — на то он и extensible. И **чтобы комплекс был хоть насколько-то future-prone, желательно, чтобы и данные в нём, и сам интерфейс, были хоть немного гибкими**. Вообще с этой точки зрения SOAP начинает выглядеть, как попытка забыть, что мир меняется, и расслабленно программировать в фиксированном java-подобном окружении

мне, конечно, могут возразить, что WSDL не настолько строг, но я опять-таки опираюсь на свой неудачный опыт, где дело было именно в этом. Не все сервисы и не все производители библиотек поддерживают эту вероятную нестрогость, если она и существует

norguhtar.livejournal.com/

Сравнивать REST и WS-* это тоже самое, что сравнивать сборку ПО при помощи configure и при помощи пакетной системы. Первое быстро и везде работает, но очень неудобно использовать если требуется одинаковая конфигурация сборки на более чем пяти системах. Вторая более муторно, надо читать документацию делать кучу не нужных телодвижений, но потом более просто в сопровождении. Тут тоже самое. REST удобно использовать для проектов где типы данных простые их немного и нет сложной логики и собственно сам проект сам в себе. Но как только усложняются типы данных, логика или требуется взаимодействие с вашим сервисом других сервисов, вот тут-то REST начнет больше мешаться, чем помогать. Ярким примером где такое может быть, это SaaS. Попробуйте заметить там WS-* на REST и увидите, что это не так хорошо как бы хотелось.

Иван Сагалаев

> И почему для REST вы допускаете возможность придумать новый нестандартный тип и тщательно его описать, а для WS-системы подразумевается, что описание данных иметь невозможно?

Потому что WS-* вообще не подразумевает описания типов сообщений. Вместо этого описывается только синтаксис интерфейсов. Это, во-первых, сильно менее полезною, а во-вторых, даже здесь нет никаких попыток делать это стандартно. То есть, это не невозможно чисто теоретически, но по факту никто так не стремится делать: ваш интерфейс -- это только ваш интерфейс.

Иван Сагалаев

> И чтобы комплекс был хоть насколько-то future-prone, желательно, чтобы и данные в нём, и сам интерфейс, были хоть немного гибкими.

+1

Яркие примеры проблем с негибкостью наблюдаю в Яндексе периодически. Если меняется интерфейс CORBA-метода, то его нельзя просто поменять в обратно-совместимом виде и выложить, потому что сломаются все клиенты. Приходится либо менять всех синхронно, либо оставлять старый метод, а рядом делать какой-нибудь myMethodEx или myMethod2. И это не умозрительная проблема.

Stump

> Я не могу это раскрыть по той причине, что не знаю этих вещей. Тем не менее, да, я берусь судить об этих вещах

Эх. Снова: "Я Пастернака не читал, но осуждаю...". Коенчно, твое право иметь собственное мнение основанное не на знаниях а на эмоциях. Но это, как бы помягче сказать, не профессионально

как-то. Дети тоже любят спорить, кто сильнее тепловоз или самолет.

> WSDL-описание не дает никакой информации о семантике системы...
> ...Теперь посмотрите на REST-сервис. Я могу не знать про него ничего, но если с какого-то URL'а, я получил, скажем, документ с типом application/atom+xml, то я знаю, что это такое, и знаю, например, могу ли я это редактировать, где именно это делать и как именно.

Да WSDL не описывает семантику, но он хотя бы описывает контракт. У REST нет и этого. А что есть? URL и набор глаголов (get put post delete), Но в wsdl это тоже есть - операции и порты. А к примеру по URL http://example.com/customers я получил документ типа application/json и по URL http://example.com/orders я тоже получил документ типа application/json а альше что? Кто мне объяснит их семантику, что там внутри и для чего? WSDL есть хотябы xsd описание формата сообщений.

> Зачем нужно было делать дополнительный уровень абстракции, заново реализующий все, что в HTTP уже есть? Только ради того, чтобы заявить, что эта штука "transport agnostic"?

Нет, не ради этого. На сегодняшний день реализации WS стека не только "transport agnostic", но "message formating agnostic", "language agnostic" и много от чего не зависим.
Я просто описываю контракт сервиса (операции) и форматы данных. Все остальное, как то: привязку к URL, транспорт, спостоб форматирования (xml, бинарный, MIME), способ аутентификации и многое другое я определяю при развертывании сервиса в его конфигурации. Для публикации в интернет я выберу HTTP, xml форматирование и дайджест аутентификацию. Для того же сервиса в локальной сети я сделаю публикацию по TCP? бинарное форматирование и Kerberos. Вот ради чего этот уровень абстракции. Вероятно, ты об этом не знал.

> Автоматическое получение стаба -- это не сервис. Потому что если бы на той стороне не было WS-сервиса, не было бы нужды в самом стабе :-). Про это я и говорю, когда пишу, что WS-стек решает ненужные проблемы.

А в чем ты увидел проблемы? В том что я могу автоматически создасть стаб для обращения к срвису на нужном мне языке (java, c#, c, vbasic, python). так это не проблема вовсе.

> Где применять WS-* мне сказать определенно трудно, но думаю, что оно подойдет для соединения нескольких компонентов одной большой системы, написанной в одном организационном стиле и на Windows-технологиях. То есть это внутрисистемная технология, а не внешняя.

Весь стек WS-* реализован в большинстве Java серверов, в том числе и в open source (Apache Axis2 к примеру, имеет реализацию на java и на C). WS-* это в первая ряду подобных (CORBA, DCOM) технология, которая реально платформонезависима. Мне казалось, это известно всем.

Но суть не в этом. Мне понятно, что REST удобен для создания публичных сервисов. Но как, например, мне решить такие задачи:

- у меня есть ресурс Order который я опубликовал через REST сервис. Get http://example.com/orders должен выдать мне список ордеров, а если их там 10 000 000 штук. Что выдаст http://example.com/orders? Первые 25? А остальные?
- Мне надо опубликовать для ресурса Orders операцию, которая создает для объекта Order объект Bill (создает счет для заказа). Как это сделать при помощи REST?
- Я воспользовался REST сервисом ресурса Order для создания нового заказа. Вот так:

POST http://example.com/orders/{order} // добавляю новый заказ

POST http://example.com/orders/xxx/{order line} // добавляю строку заказа

POST http://example.com/orders/xxx/{order line} // добавляю строку заказа

POST http://example.com/orders/xxx/{order line} // добавляю строку заказа

Что делать, если на третьем POST я получаю код 500?

- как при помощи REST организовать гарантированную доставку сообщений?
- как при помощи REST организовать publish / subscribe схему?

Я вот нигде не видел внятных ответов на эти вопросы. И я даже предполагаю почему. Потому что REST не сильно подходит для таких вещей. Это конечно не вина REST, молоток, к примеру, тоже не сильно подходит для заворачивания шурупов.
Есть круг задач, которые хорошо решаются при помощи REST. А есть круг задач с которыми REST не справляется, но зато WS-* решает из влет.

Stump

> Потому что WS-* вообще не подразумевает описания типов сообщений. Вместо этого описывается только синтаксис интерфейсов.

Иван, тут ты заблуждаешься. Возьми людой WSDL и посмотри, там есть описания типов сообщений, или ссылки на схему которая, содержит такие описания.

Иван Сагалаев

Сергей, забегая вперед, я порекомендую вам прочитать RESTful Web Services, потому что практические вопросы, которые вы задаете, хрестоматийны. Я отвечу только кратко, и постараюсь не повторяться.

> Да WSDL не описывает семантику, но он хотя бы описывает контракт. У REST нет и этого.

Вы опять впрямую сравниваете две вещи. REST -- архитектурный

стиль, он не имеет дела с конкретными реализациями. Ничто не мешает вам придумать контракты для REST-сервиса.

> Для публикации в интернет я выберу HTTP, xml форматирование и дайджест аутентификацию. Для того же сервиса в локальной сети я сделаю публикацию по TCP, бинарное форматирование и Kerberos. Вот ради чего этот уровень абстракции. Вероятно, ты об этом не знал.

Как ни странно, знал. Тут мне все же придется повторить мысль из статьи. Идея о публикации *одного и того же* сервиса и внутрь и наружу представляется мне бесполезной. Это возможно удобно для создателя сервиса (писать все один раз), но неудобно для пользователей сервиса, потому что изнутри и снаружи к интерфейсам предъявляются обычно разные требования по *семантике*, а не по форматам данных. В форматах-то как раз нет проблемы никакой: внутренние сервисы так же успешно могут пользоваться HTTP/XML. Я снова возвращаюсь к тому, что WS-* решает не те проблемы.

> Весь стек WS-* реализован в большинстве Java серверов, в том числе и в open source (Apache Axis2 к примеру, имеет реализацию на java и на C). WS-* это в первая ряду подобных (CORBA, DCOM) технология, которая реально платформонезависима. Мне казалось, это известно всем.

Извините, но .NET и Java -- это не кроссплатформенно. Факт в том, что подавляющее большинство веба работает на Линуксе и PHP. Потихоньку развиваются и занимают свое место Python и Ruby. До тех пор, пока технология делает вид, что этих языков нет -- она нишевая. Хотя сидя внутри нее вам может казаться, что все наоборот :-).

(Только пожалуйста, не надо говорить про IronPython).

> у меня есть ресурс Order который я опубликовал через REST сервис. Get http://example.com/orders должен выдать мне список ордеров, а если их там 10 000 000 штук. Что выдаст http://example.com/orders? Первые 25? А остальные?

```
<link href="http://example.com/orders?page=2" rel="next">
```

Это стандартный способ.

> Мне надо опубликовать для ресурса Orders операцию, которая создает для объекта Order объект Bill (создает счет для заказа). Как это сделать при помощи REST?

Сергей, не *при помощи*. Это не вещественная штука. Правильный вопрос "как это проектируется в терминах REST". В терминах REST у вас появляется ресурс Bill (или Bills), который вы создаете (PUT'ом или POST'ом, в зависимости от специфики), передавая представление нового счета с URL-ссылкой на существующий ресурс заказа.

> Что делать, если на третьем POST я получаю код 500?

Для маленьких заказов вы можете передать весь контент одним запросом. Если это не практично, вы создаете ресурс Транзакция, в который набираете заказ сколько угодно времени, а потом изменяете ее состояние на "committed", и только тогда заказ появляется в системе.

> А есть круг задач с которыми REST не справляется

С задачами справляется не REST, а программист, реализующий сервис на REST-принципах. Это не жонглирование словами, это принципиально. Вы ждете, что за вас стек протоколов будет решать конкретные задачи. Мы же утверждаем, что задачи бывают разные, решать их нужно по-разному, а REST -- это набор общих базовых принципов, которые стоит использовать при решении.

> Возьми людой WSDL и посмотри, там есть описания типов сообщений, или ссылки на схему которая, содержит такие описания.

Схема не является описанием сообщения! Она описывает только

синтаксис. А MIME-тип описывает семантику, потому что у него есть RFC, в котором человеческим языком описано, что с чем и как связано. И есть библиотеки, которые с этим работают. А по произвольной схеме родить автоматически консумера нельзя.

hodzanassredin

Иван провокатор. Интересно читать комментарии. Хочу добавить еще несколько копеек. На западе WS-* называют не иначе как WS-DeathStar :-). Кстати Девид Чаппел на конференции советовал прочитать туже книжку, что и ты Иван. МС не является противником Рест сервисов, сейчас делают много в этом направлении. Например в WCF(самая новая имплементация стека от МС в ява насколько я знаю, правда могу ошибаться, есть Metro с подобными функциями) я могу спокойно использовать контракт как в Рест так и в ВС стиле. В WCF можно спокойно использовать любой протокол и т.п. Штука в общем хорошая. Также МС двигает АСТОРИА это тоже на прямую завязано на Рест сервисы. Хотя с другой стороны Мигель Де Иказа(лидер проекта моно свободной имплементации дот нет) объявил что с WCF даже связываться не станут а рекомендуют всем испльзовать ICE(www.zeroc.com) кстати ICE можно юзать и на питоне. А про себя могу сказать что обеими руками за Рест но ВС мне тоже нравицо только не в том виде в котором она сейчас существует :-)

hodzanassredin

На тему изменяющихся контрактов есть интересная статья у Фаулера
http://martinfowler.com/articles/consumerDrivenContracts.html

Stump

> Сергей, забегая вперед, я порекомендую вам прочитать RESTful Web Services, потому что практические вопросы, которые вы задаете, хрестоматийны.

Мне простительно, поскольку я из вражеского enterprise лагеря :)
Книжку, я вряд ли стану читать, а вот пост в блоге на эту тему прочитал бы с удовольствием.
За ответы спасибо. Я, вобщем-то, так и предполагал, что контракты операций в терминах REST проектируются (надеюсь в этот раз я кошерно выразился) в треминах ресурсов. Даже когда это выглядит странно.

А вот, о существовании стандартов регламентирущих семантику RESTful сервисов не знал. Это интересно.
Кстати, и Microsoft заинтересовалась REST сервисами. Видимо пройдет совсем немного времени, все это обрастет стандартами и станет уже не отличить добрые RESTful серфисы от злых WS-DeathStar.
И придется вольным ковбоям интернета искать себе новые пастбища :)

meowth

> если вы делаете публичный веб-сервис, его надо делать REST'овым.

Вот с этого надо было начинать. Публичный. Эта самая публика пользуется такими инструментами/языками, для которых работа с wsdl превращается в сложную, муторную, почти невыполнимую задачу (привет, php!). При этом им не нужнен и 1% того, что позволяет реализовать WS. Поэтому REST для них - самое то, потому что это - минимальная надстройка над HTTP, который является совсем "родным" :). SOAP/WSDL/WS-* - это сильнее и мощнее, гораздо мощнее (например, можно сменить транспорт на не-HTTP, сохранить запросы в очереди, затем обработать и т.д.).

Так что не надо ругать wsdl, а вместе с ним веб-службы и enterprise в его базисе - он свое дело знает и делает хорошо :)

> WS-* — плохой способ делать веб-сервисы. Он решает ненужные проблемы, зато не решает нужных.
> ...

> Я не могу это (WS-*) раскрыть по той причине, что не знаю этих вещей.

На мой вгляд, несколько необоснованный вывод, нет?

Иван Сагалаев

> Вот с этого надо было начинать. Публичный.

А я с чего начал? Перечитайте в самом начале статьи раздел "Предметная область".

> Поэтому REST для них - самое то, потому что это - минимальная надстройка над HTTP

Кажется, вы не читали таки статью :-). Позвольте мне не повторять здесь заново раздел "REST (кратко)".

> WS-* — плохой способ делать веб-сервисы. Он решает ненужные проблемы, зато не решает нужных.
> ...
> Я не могу это (WS-*) раскрыть по той причине, что не знаю этих вещей.
>
> На мой вгляд, несколько необоснованный вывод, нет?

Нет. Аналогию с прыганием с пятого этажа я привел в том же абзаце. Если нужно без аналогий, то не обязательно изучать стек протоколов детально, чтобы понять, что он не подходит для задачи в целом. А то, что WS-* покрывает свою нишу я тоже уже выше в комментариях писал.

Кроме того, мне кажется, вы путаете мощность со сложностью.

meowth

Боюсь, мне не достаточно Ваших аргументов "вы не читали мою статью, перечитайте ещё раз" и странных аналогий, например, падение человека с некоторого этажа.

> WS-* покрывает свою нишу

Вы не про нишу, вы про девелоперов, по вашему мнению с фанатичным послушанием заглядывающих в глаза мегакорпорациям", выпускающим неудобные стандарты. Масонская ложа прямо, а не корпорации - организовали заговор против свободных разработчиков

> сравнение все таки напрашивается, пусть даже и на эмоциональном уровне

Очень жаль, что это вы сочли достаточным аргументом, чтобы сделать парадоксальный вывод в конце статьи.

Вы уверены, что доступно объяснили, но читатели, такие-сякие, не понимают глубины и гениальности изложения? Простите, не смогу поддержать обсуждение в такой же манере; присоединяюсь к Сергею Розовику - он своевременно указал на большую часть неточностей и передергиваний фактов, допущенных в статье.

P.S. Прокомментировал в блоге Вашу статью, не буду повторяться здесь.

bsdemon

*Сравнение REST и WS-**

> REST -- это архитектура
> ...
> WS-* -- это обобщенное название набора спецификаций

Этим всё сказано :). Сравнивать Нельзя!

Сравнение SOA и REST

стиль SOA:
http://somedaomain.com/action?with_data=data_id
стиль REST:
http://somedaomain.com/some_data/data_id/action
(ну конечно action в некоторых случаях можно заменить на DELETE запрос если мы удаляем ресурс, ну и всё в этом духе)

Разница то какая? Никакой, кому как нравится. (мне нравится REST)

Иван Сагалаев

> Разница то какая?

Дело в том, что разница принципиальная. Весь REST этому и посвящен, что *есть разница*, универсальная у вас семантика интерфейса или кастомная. Я специально вывел за пределы статьи подробное объяснение, почему это все важно, иначе у меня бы сил не хватило дописать. Но я всеми силами советую почитать приведенные ссылки, или почитать книжку, на которую я уже пару раз тут ссылался.

jekyll

Может я чего-то не понимаю, но в начале автор пишет, что это статья о том, что REST и WS - разные вещи и сравнивать их нельзя, а чуть ниже идёт подзаголовок сравнение...

Получается, что их сравнивать нельзя, но мне так захотелось и я сравнил.

Тогда уж честно и скажите, что Вы адепт REST и против WS.

Иван Сагалаев

> Получается, что их сравнивать нельзя, но мне так захотелось и я сравнил.

Ну что же в этом ужасного? Я обосновал, почему технически впрямую их сравнивать нельзя. Но пошел дальше и предположил, почему их таки все сравнивают на интуитивном уровне, и почему мне кажется, что это не бессмысленно.

> Тогда уж честно и скажите, что Вы адепт REST и против WS

Я и сказал честно: "А эта статья -- о том, почему REST и WS-* нельзя сравнивать, почему их все-таки сравнивают, и почему первое лучше второго. Да, никакой объективности от меня тут ждать не надо :-)."

Я понимаю, что адепты WS-протоколов не нашли эту статью для себя приятной. Но я не считаю это проблемой. Я крайне не согласен с тем, что эта плохая (на мой взгляд) технология активно предлагается как технология для публичного веба. Об этом эта статья и есть.

sin

Как уже не раз было отмечено, сравнивать семейство спецификаций и архитектурный стиль некорректно.

Если же говорить о SOA vs REST, то напрашивается аналогия — взять фреймворк или склеить всё самому :)

В первом случае за нас подумали лучшие умы и уже решили все возможные проблемы по-своему, написали тонны отлаженного кода: берем какую-нибудь 1С, клац-клац в IDE — веб-сервис готов!

Во втором случае нас не устраивает избыточность и сложность инструментария для нашей задачи, его стоимость или привязанность к платформе, какие-нибудь врождённые особенности — и мы собираем всё из кирпичиков: формулируем интерфейс в терминах ресурсов, любуясь урл-конфом, придумываем соглашения для заголовков и формат сущности, штудируя rfc и составляя ручками xsd.

Иван, любопытно, как вы сочетаете пылкую любовь к фреймворкам с неприятием промышленных стандартов веб-сервисов? Ваши аргументы в пользу фреймворков сейчас бы в уста

сторонников SOA...

Ну это я так, чтобы затухающую дискуссию поддержать, а то что-то WS-* заругали совсем :) Сам люблю больше и чаще использую REST. SOA использую только когда ситуация способствует, что обычно выражается в необходимости интеграции с существующей системой, где уже запрограммирована бизнес-логика на уже купленном коммерческом софте, который позволяет легко опубликовать бизнес-объекты через SOAP.

За пост спасибо — давно его ждал. Комментарии порадовали. Пишите ещё про REST, ближе к практике из собственного опыта, если можно.

Иван Сагалаев

> В первом случае за нас подумали лучшие умы и уже решили все возможные проблемы по-своему, написали тонны отлаженного кода: берем какую-нибудь 1С, клац-клац в IDE — веб-сервис готов!

Некоторое время работы разработчиком и менеджером убедили меня, что "клац-клац -- и готово" -- не бывает. Таким образом обычно можно получить максимум рабочий макет сервиса.

> и мы собираем всё из кирпичиков ... и составляя ручками xsd.

Нет, вот слово XSD тут лишнее :-)

> Иван, любопытно, как вы сочетаете пылкую любовь к фреймворкам с неприятием промышленных стандартов веб-сервисов?

Ну я же не абстрактную "идею фреймворков" пропагандирую. Я пропагандирую хорошо сделанные фреймворки для веб-разработки на Питоне. Точнее, один такой фреймворк. В нем нет и намека на "клац-клац -- и готово", в нем программировать надо.

Кроме того, уж простите, но нет никаких промышленных стандартов веб-сервисов. Про WS-* так говорят только те, кто его продает, и стандартом де-факто он возможно является только во внутрикорпоративной среде, сидящей на .NET или Java. Если мы говорим о *вебе* (о публичном конечно, хотя оговорка и излишняя), то WS-* тут практически не присутствует.

У REST-подхода тоже свои проблемы. В частности -- слишком высокий порог вхождения. Поэтому на деле, то, что мы имеем в качестве стандарта де-факто на вебе -- это самопальные протоколы на базе HTTP в RPC-стиле. Это плачевное положение дел стоит исправлять просвещением, и я думаю, что пропагандировть хороший архитектурный стиль куда важнее, чем какой-то конкретный продукт.

> Пишите ещё про REST, ближе к практике из собственного опыта, если можно.

Я буду работать в этом направлении :-)

sin

> Некоторое время работы разработчиком и менеджером убедили меня, что "клац-клац -- и готово" -- не бывает. Таким образом обычно можно получить максимум рабочий макет сервиса.

Я имел виду использование того, что предлагают многие платформы и IDE для быстрого создания веб-сервиса на WS-* (по аналогии с чужим кодом фреймворков), а не отсутствие необходимости писать код.

> Нет, вот слово XSD тут лишнее :-)

Это удобно в проектах интеграции готовых систем, когда необходимо договориться с другой стороной о том какой XML будем передавать в представлении ресурсов.

> Ну я же не абстрактную "идею фреймворков" пропагандирую.

Это хорошо. Мне так показалось после критики постов про WSGI.

> Кроме того, уж простите, но нет никаких промышленных стандартов веб-сервисов. Про WS-* так говорят только те, кто его продает, и стандартом де-факто он возможно является только во внутрикорпоративной среде, сидящей на .NET или Java. Если мы говорим о вебе (о публичном конечно, хотя оговорка и излишняя), то WS-* тут практически не присутствует.

И правильно, что не присутствует. Между публичным вебом и внутрикорпоративной средой есть веб-сервисы эм... междукорпоративные. Тут WS-* работает, но и REST есть место. Моё мнение — организациям сейчас проще договориться на WS-* и, учитывая готовый набор спецификаций и инструментов, я говорю о промышленном стандарте.

> У REST-подхода тоже свои проблемы. В частности -- слишком высокий порог вхождения. Поэтому на деле, то, что мы имеем в качестве стандарта де-факто на вебе -- это самопальные протоколы на базе HTTP в RPC-стиле.

Именно так. А может все в порядке вещей? Да, привыкнув к RPC тяжело переключиться на ресурсы/представления, особенно если для того чтобы это хорошо сделать нужно прочитать диссертацию Роя, а потом ещё и думать как это применить на системах чуть меньше чем вся www :) Самопальные протоколы, возможно, имеют место т.к. легко решают простые задачи, CRUD по HTTP ещё не REST независимо от структуры URL'ов, а суть REST такова, что основные преимущества проявляются для веб-проектов при масштабировании до уровня мировой популярности. Архитектурный стиль, на основе которого из HTTP 1.0 сделали 1.1, благодаря которому мы имеем расширяемый веб сегодня, не всегда уместно применять при построении веб-сервисов просто чтобы было. Есть мнения, что REST якобы проще и дешевле что ли, чем SOA. Мне так не кажется.

> Это плачевное положение дел стоит исправлять просвещением, и я думаю, что пропагандировть хороший архитектурный стиль куда важнее, чем какой-то конкретный продукт. ... Я буду работать в этом направлении :-)

Конкретный продукт — это упомянутый мной 1С? О, я его не пропагандирую, подвернулось, просто в последний раз по SOAP приходилось к 1С ходить, сейчас множество платформ автоматизации и тяжёлых инструментов разработки умеют SOAP. И REST пропагандировать лучше не надо, сейчас, мне кажется, не хватает примеров удачного проектирования конкретных систем с использованием некоторых принципов REST, на которых бы иллюстрировались расширяемость и масштабируемость по построению. Ну это моё пожелание такое, как читателя :)

Иван Сагалаев

> Нет, вот слово XSD тут лишнее :-)
> Это удобно в проектах интеграции готовых систем, когда необходимо договориться с другой стороной о том какой XML будем передавать в представлении ресурсов.

Я считаю, что XSD -- плохой способ описания "какой XML". Слишком невыразительный и жесткий. Лучше него, как ни странно, четкое описание словами на естественном языке. В идеале -- снабженное валидирующими тестами на свободном императивном языке, который может выразить любые ограничения. Но это уже другой холивор :-)

> Между публичным вебом и внутрикорпоративной средой есть веб-сервисы эм... междукорпоративные.

Вот с этой терминологией я буду воевать до конца :-). "Междукорпоративный веб" -- это оксюморон. Веб -- это World Wide Web. Публичная открытая гетерогенная и *очень большая* сеть. Очень мало похожая на корпоративные интранеты. Одно только наличие в них HTTP не делает их вебом ни под каким условием.

Для меня, кстати, очевидно, что первое слово в "Web Services" -- исключительно маркетинговое. Это сейчас модно. Так же, как в свое время разработчики XMLHttpRequest'а признавались, что слово "XML" засунули в название только чтобы проще протолкнуть эту технологию внутри микрософтовского менеджмента. Тогда это было модно :-)

> Моё мнение — организациям сейчас проще договориться на WS-* и, учитывая готовый набор спецификаций и инструментов

Тоже не соглашусь. По факту в WS-стеке все очень хреново с интероперабельностью (на яркий отзыв ссылался Arty выше). Поэтому готовым набором инструментов я бы это не назвал. Их все равно надо сильно дорабатывать, и здесь сложность стека играет в обратную сторону: его, очевидно, дорабатывать сложно. Сложнее, чем писать свое. Особенно, если знать, как. А REST как раз о том "как".

Например лет пять назад мы связывали сайт мобильных телефонов Самсунга и магазин Телефон.Ру как раз ad-hoc XML'ом через HTTP с уклоном в REST. С одной стороны реализация была на Perl, с другой -- PHP. Технически работа была сделана буквально за дни и хорошо потом работала долгое время. Сколько бы мы сваривали это через WS-* предсказать не берусь, кажется под Perl и PHP тогда не существовало ничего такого. И это, замечу, не публичный веб, а как раз то поле, где WS-* типа стандарт :-)

> Архитектурный стиль, на основе которого из HTTP 1.0 сделали 1.1, благодаря которому мы имеем расширяемый веб сегодня, не всегда уместно применять при построении веб-сервисов просто чтобы было.

Мне нравится наша беседа :-).

Да, в целом так. Но прелесть в том, что REST -- не эксклюзивное предложение. Можно делать сервисы, используя только часть принципов REST. Ту, которая имеет смысл в данном месте. И не убиваться, что "а вот тут можно использовать PUT, а мы к сожалению используем некошерный POST".

> Конкретный продукт — это упомянутый мной 1С?

Нет. Я имел в виду WS-* как таковой. Насколько я понимаю, реализаций стека не так много, и хот это вроде как стандартизованные протоколы, обычно речь идет о конкретной реализации или паре конкретных реализаций, совмещаемых друг с другом.

sin

Почитал яркий отзыв. Я сразу, интуитивно, чтобы избежать проблем интероперабельности, отказался от типов в SOAP, стал использовать строки исключительно, структуры в XML передавать строками. Ещё отпишу про XSD и междукорпоративные (B2B) веб-сервисы. Пойду спать :)

> И не убиваться, что "а вот тут можно использовать PUT, а мы к сожалению используем некошерный POST".

)))

ivanko

Хотел бы задать уточняющий вопрос.
Если у нас есть допустим функционал подписи. Т.е. в стиле RPC мы передаем идентификатор сертификата и данные для подписи (как вариант данные для проверки подписи), ожидая в ответ подписанные данные (как вариант ответ OK/FAIL). Как это могло бы быть сформулировано в терминах архитектуры REST? Что должно быть ресурсом к которому происходит обращение? Насколько я понял из написанного выше и ссылок предъявленных ранее, мы сначала создаем ресурс по какому-то ID (значение ресурса - данные), далее выполняем операцию GET этого ресурса с параметром - id сертификата.

Maverick Crank GRey

> Я считаю, что XSD -- плохой способ описания "какой XML". Слишком невыразительный и **жесткий**. Лучше него, как ни странно, четкое описание словами на **естественном языке**.

Я понимаю, что по "примеры ничего не доказывают" (с), но мой опыт говорит, что *жесткость* есть благо. Благо оно тогда, когда есть задача создать B2B взаимодействие, но:

- *естественный* английский язык не родной ни для одной из сторон разрабатывающих это взаимодействие (русские, индусы и израильтяне)
- другого способа общения, акромя WSDL/XSD не предусмотрено "межкорпоративным этикетом"
- по ту и эту сторону сидят люди, которым свойственно ошибаться и письмо с невалидным сообщением (конвертом) единственный способ донести до другой стороны ваши проблемы.

> В идеале -- снабженное валидирующими тестами на свободном императивном языке, который может выразить любые ограничения. Но это уже другой холивор :-)

Не могли бы вы дать ссылку на такое описание/пример для достаточно сложного REST ресурса (HelloWorld не предлагать =), что бы я примерил свои вышеописанные проблемы к REST-way способу выражения ограничений?

Иван Сагалаев

> Если у нас есть допустим функционал подписи.

Я, к сожалению, совсем не представляю себе этого процесса :-(. Поэтому возможно ответу очень невпопад.

> Т.е. в стиле RPC мы передаем идентификатор сертификата и данные для подписи (как вариант данные для проверки подписи), ожидая в ответ подписанные данные (как вариант ответ OK/FAIL).

А куда это передается? Где существуют данные, которые надо пописывать?

Если я правильно догадываюсь, то вариант может быть таким:

- У нас есть какой-то ресурс --Документ -- неважно как созданный ранее, важно, что у него есть URL.
- В этом документе есть поле Подпись, которое может быть пустым, а может быть заполненным.
- Берется представление этого документа, или уже существующее на клиенте или получаемое GET'ом.
- В этом представлении меняется подпись и документ PUT'ается обратно на свой URL.
- Сервис либо отвечает "200 OK", либо "400 Bad Request".

То есть, если "подписанность" представляется как свойство ресурса, то прямой способ его редактировать -- пытаться записывать отредактированный ресурс на сервер.

Иван Сагалаев

> другого способа общения, акромя WSDL/XSD не предусмотрено "межкорпоративным этикетом"

А это что за зверь? :-). Что мешает людям с разных сторон договориться так, как это им удобно?

> по ту и эту сторону сидят люди, которым свойственно ошибаться и письмо с невалидным сообщением (конвертом) единственный способ донести до другой стороны ваши проблемы

Регулярно доношу до партнеров (и они до меня) несогласованность форматов на просто русском языке в духе "к нам от вас приходят URL'ы вида http://hostnamehttp://, которые не открываются, почините пожалуйста".

Вот мне, кстати, интересно, есть в XSD средства выражения понятия "URL должен открываться и возвращать картинку"?

> > В идеале -- снабженное валидирующими тестами на свободном императивном языке, который может выразить любые ограничения.
>
> Не могли бы вы дать ссылку на такое описание/пример для достаточно сложного REST ресурса

Наверное нет... Из памяти достать не могу, а свой мы еще не написали :-). Но в качестве иллюстрации того, о чем я говорю, могу предложить conformache checker для HTML5. Они используют проверки на базе RelaxNG, а также недекларативные чекеры -- см. раздел Non-Schema Checkers в "About Service". Первый же пример там очень яркий: проверка целостности HTML-таблиц. Попробуйте выразить это на XSD...

Maverick Crank GRey

> > другого способа общения, акромя WSDL/XSD не предусмотрено "межкорпоративным этикетом"
>
> А это что за зверь? :-). Что мешает людям с разных сторон договориться так, как это им удобно?

Все просто. Ты аутсорсер. Работаешь на западную контору. Твоя "точка взаимодействия" с этой конторой жестка ограничена неким кругом лиц на той стороне, которые знают о твоём существовании. И это не обязательно инженеры, вполне могут быть обычные MBA-манагеры. Для другой части этой конторы ты просто не существуешь - есть только код тобой выдаваемый.

Так же у этой конторы есть другой аутсорсер или контора-контрактор (например, израильский Amdocs) "точка взаимодействия" которой также ограничена. Возможно даже более жёсче, чем у тебя (предположим, им платят за реализацию каждой конкретной фичи, а за время работы).

А теперь хочу обратить внимание, что люди из твоей "точки взаимодействия" могут не знать имен людей из "точки"

контрактора... А так удививший тебя "этикет" запрещает тебе даже узнать email "того парня из Amdocs", что написал сервис, который тебе приходиться вызывать. Даже если ты ему и напишешь, то его манагер "ляжет грудью на амбразуру" и не даст ему и пальцем шевельнуть - за это "не заплОчено".

Конечно, способ обмена информацией сущестует, но это очень напоминает голубиную почту. Это заставляет организовывать такое "общение" как можно более кратким, однозначным ("Пиши вопросы, на которые можно ответить только ДА или НЕТ.") и ёмким. Жесткий контракт WSDL/XSD этому помогает.

Maverick Crank GRey

> Регулярно доношу до партнеров (и они до меня) несогласованность форматов на просто русском языке в духе "к нам от вас приходят URL'ы вида http://hostnamehttp://, которые не открываются, почините пожалуйста".

В наших реалиях, бывает, что сервис реально не существует - есть только его контракт. То бишь тебе дали лишь WSDL/XSD и сказали, что нужно начать взаимодействовать с этим сервисом. А перед тем как выходить "в жизни", "те парни" нам тестовую реализацию подгонят... может быть, если успеют.

В таком случае неоткого принимать "кривые запросы". Ты по этому WSDL создаешь эмулятор, которые будет паинькой и всегда будет отвечать правильно. А что там реально будет написано на той стороне, бабушка вилами надвое написала.

P.S. Есть у меня подозрение, что в случае с REST я тоже могу написать "эмулятор будущего". Но я пока не грокнул идеи как можно

> allow servers to instruct clients on how to construct appropriate URIs, such as is done in HTML forms and URI templates, by defining those instructions within media types and link relations. [http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven]

Ведь какой-то общие понятия должны быть все равно. Может это из-за того, что я не могу понять упомянутой аналогии с формами в HTML? То есть то, что сервер может сегодня мне вернуть мне HTML содержащий форму с другим набором полей, я понимаю. И я надеюсь, что моего естественного интеллекта хватит, что бы её заполнить (если это, конечно, не форма налоговой отчётности =). Но как это сможет "понять" REST клиент? Как он поймёт, что тут теперь ещё и номер свидетельства о рождении нуна вписать? Если это не так, что дает сравнение с HTML формами? Возможность изменять значения в атрибутах **action** и **method**?

Иван Сагалаев

> Может это из-за того, что я не могу понять упомянутой аналогии с формами в HTML? То есть то, что сервер может сегодня мне вернуть мне HTML содержащий форму с другим набором полей, я понимаю. И я надеюсь, что моего естественного интеллекта хватит, что бы её заполнить (если это, конечно, не форма налоговой отчётности =). Но как это сможет “понять” REST клиент?

Этот подход подробно расписан в "RESTful Web Services", кстати.

REST не обещает того, что ПО клиента сможет *автоматически* понять по одной только гипермедии, что ему делать дальше. Искусственного интеллекта мы пока не придумали. Поэтому конечно какого-то полностью обобщенного REST-клиента, которому выдается URL на морду, а дальше он автомагически знает, что делать, написать невозможно. Весь смысл подхода с включением гипермедии в представление в том, что человек, посмотрев на это и поняв формат, в котором все происходит, может написать клиента для данного конкретного REST-ориентированного протокола или множества протоколов. И дальше этот клиент сможет работать с другими сервисами.

Maverick Crank GRey

> > Не могли бы вы дать ссылку на такое описание/пример для

> достаточно сложного REST ресурса
>
> в качестве иллюстрации того, о чем я говорю, могу предложить conformache checker для HTML5.

Ага. Кажется я начинаю понимать. В архитектуре REST ограничения накладываются только на документы ("существительные"), на их структуру. Накладывать ограничения на инструкции взаимодействия, видимо, нет смысла, так как методы ("глаголы") и так ограничены спецификацией HTTP. И если кто и применит неверный метод к некому документу/ресурсу (употребит не тот глагол с каким-нибудь существительмым), то получит "400 Bad Request".

Уж не помню где, кажется на последнем питерском SunDevDay, один из докладчиков показывал разницу между REST и WS. Он сказал, что WS -это когда у вас мало "сущностей" (обычно, это именно бизнес-сущности), но много "глаголов" для работы с ними. А REST стиль - это наличие множества "сущностей" (ресурсов) и ограниченного количества "глаголов" для операции с ними.

> Они используют проверки на базе RelaxNG, а также недекларативные чекеры -- см. раздел Non-Schema Checkers в "About Service". Первый же пример там очень яркий: проверка целостности HTML-таблиц. Попробуйте выразить это на XSD...

Согласен. Проверки написанные на "a Turing-complete programming language" бывают иногда единственным решением. Но признайтесь, часто ли бывают так сложны документы в системах межмашинного общения?)

Да и не стоит думать, что WSDL ограничен только XSD - http://www.w3.org/TR/2003/WD-wsdl12-20030611/#relax

Да и сама XSD может быть "облагорожена" - http://www.xfront.com/ExtendingSchemas.html

P.S. Прощу прощения, если увожу дискуссию от темы.

Maverick Crank GRey

> REST не обещает того, что ПО клиента сможет автоматически понять по одной только гипермедии, что ему делать дальше. Искусственного интеллекта мы пока не придумали.

Ага. А то уж я испугался, что пропустил что-то важное в IT =)

> Весь смысл подхода с включением гипермедии в представление в том, что человек, посмотрев на это и поняв формат, в котором все происходит, может написать клиента для данного конкретного REST-ориентированного протокола или множества протоколов.

То бишь такой подход настаивает на создании инструкций к сервису в человеко-читаемом представлении, нежеле машино-читаемом. Согласен, что сеё удобно для глобальной сети, но в "ынтырпрайз" мне тяжело это представить (см. описание стиля работы выше =).

> Вот мне, кстати, интересно, есть в XSD средства выражения понятия "URL должен открываться и возвращать картинку"?

Это делает не XSD. Это описывает WSDL, XSD ему в этом помогает.

P.S. Дак и как, кстати, в мире REST обстоят дела с написанием **стаба/имитатора/эмулятора** сервиса?

P.P.S. Пойду "RESTful Web Services", что-ли почитаю... =)

Иван Сагалаев

> Он сказал, что WS -это когда у вас мало "сущностей" (обычно, это именно бизнес-сущности), но много "глаголов" для работы с ними. А REST стиль - это наличие множества "сущностей" (ресурсов) и ограниченного количества "глаголов" для операции с ними.

Так и есть. А в посте я старался показать, что это отличие отнюдь не косметическое.

> Согласен. Проверки написанные на "a Turing-complete programming language" бывают иногда единственным решением. Но признайтесь, часто ли бывают так сложны документы в системах межмашинного общения?)

По-моему, сплошь и рядом... Пример с "должно открываться и быть картинкой" разве не оно? Потом, тут дело не в том, часто или редко. Тьюринг-полный язык имеет по факту больше возможностей, чем декларативная схема. Поэтому, если у него нет серьезных минусов для задачи, то его использовать логичнее. А в том, о чем мы говорим, минус скорее у XSD -- он сложнее и хуже читается, чем код на том же Питоне.

> То бишь такой подход настаивает на создании инструкций к сервису в человеко-читаемом представлении, нежеле машино-читаемом.

М-м... Не совсем. Нет вопроса в том, наставивать на человекочитамом описании или нет: разумеется, без людей мы никуда еще не денемся. Но REST рекомендует делать так, чтобы представление ресурса, будучи изначально машиночитаемом, в любом случае что-то говорило человеку, посмотревшем в браузере view source. Это никакая не высокая наука, но это дико важно.

> P.S. Дак и как, кстати, в мире REST обстоят дела с написанием стаба/имитатора/эмулятора сервиса?

Да в общем-то, никаких особенностей у REST-стиля тут нет. Нужна заглушка -- она берется и пишется. Интерфейс с реализацией разделен, поэтому проблем я никаких не вижу.

Maverick Crank GRey

> Но признайтесь, часто ли бывают так сложны документы в системах межмашинного общения?)

По-моему, сплошь и рядом... Пример с "должно открываться и быть картинкой" разве не оно?

Не уверен, что такой случай применим к WS.

Во-первых, на сколько я знаю, они всегда отвечают "сообщением в конверте". Даже если это multipart сообщение. Да и тип ответа у них будет что-то типа application/soap-xml - http://www.ietf.org/rfc/rfc3902.txt То бишь я не знаю как по SOAP "выплюнуть" в ответ только "image/png" поток. Это всегда будет XML. А раз так, он должен соответствовать WSDL типизированной XSD или чем-то другим (RelaxNG).

Во-вторых, WSDL декоративно, явно и заблаговременно указывает на какие URI имеет смысл посылать запросы. Так что, если ты и выслал запрос на "деревню к дедушке", то ССЗБ ибо тебя заранее инструктировали.

Голубой Гигант

Полностью согласен и подтверждаю замечания Maverick Crank GRey.
Такое чувство что REST предполагает неравенство взаимодействующих сторон. Т.е. с одной стороны гигант вроде Google, которому удобен REST т.к. можно довольно свободно изменять свой сервис, а с другой, куча мелких потребителей, знания которых ограничиваются PHP и которые готовы подстраиваться и меняться постоянно.
Когда описывается протокол для взаимодействия равных сторон, такой сценарий не проходит, потому что все спецификации утверждаются и подписываются задолго до начала собственно разработки. Текст WSDL прилагается к или даже включается в саму спецификацию. И это в миллион раз лучше любого совестного описания, т.к. исключает всякую двусмысленность в понимании. Не нужно долгих судебных разборок и филологических экспертиз: тест на соответствие WSDL дает однозначный ответ кто прав, а кто виноват, если что-то не

работает.

Иван Сагалаев

Нет, кажется вы напутали тут :-). REST сам по себе "рассчитан" только на то, чтобы система была простой, понятной и масштабируемой. Если вы очень хотите использовать для взаимодействия схему любого рода, никто вам не помешает, и никакому духу это не противоречит. То, что мы с Maverick'ом стали рассуждать про XSD и императивную валидацию -- это просто другая тема, к REST уже не имеющая отношения.

Сергей Шепелев

Пишем якобы нереально масштабируемый сервис, якобы по принципам REST, но спасибо, узнал много нового.

Не используем HTTP методы, коды, заголовки (только один - Content-Type: application/json). Урлы не содержат ничего, кроме имени сервиса, которое могло быть частью домена, то есть урлы вообще ничего не содержат. action/method и data/arguments идут в одном GET аргументе. То есть по транспорту есть полная независимость от HTTP, хотя она нафиг не нужна.

Господа, HTTP-agnostic это не фича, это галочка. Не потому что "у вас нет реализаций кроме хттп". Потому что другие не нужны.

"Сообщения" оборачиваются в JSON, сервис смотрит наружу, поэтому для JSON был написан "валидатор на простом императивном языке", извольте http://github.com/kmerenkov/validol

Ивану спасибо за статью, коментаторам спасибо за интересные мысли.

Alexander P.

Работаю разработчиком в компании, которая занимается автоматизацией складов. У нас есть сервер приложений на котором крутится бизнес-логика, - это большое количество

взаимосвязанных Java-классов, с множеством полей. Клиентские приложения подключаются к серверу через WS-*. К своему стыду должен признать, что несмотря на то, что Web-сервисы у нас являются важной частью всего приложения, я в них мало разбираюсь. Дело в том, что для разработки нашего приложения не требуется вникать в детали работы Web-сервисов, так же, как, например, не требуется вникать в детали работы Ethernet, UDP и или TCP/IP. Есть Java-классы, которые объявлены с аннотацией @Webservice и это, практически все что нужно, сделать на сервере, чтобы выставить наружу нужные методы, - они просто объявляются как public-методы Java-класса. На клиенте указывается http-путь к WSDL и автоматически генерятся (или обновляются) классы-stub'ы, которые соответствуют методам Web-сервиса и используемым типам (разумеется, это тоже могут быть классы). Разработчик занимается программированием предметной области (бизнес-логики на сервере и UI на клиентах), работает с классами и не думает о Web-сервисах. Экземпляры классов легко и непринужденно передаются от клиента к серверу и наоборот. Кстати, клиенты могут быть не только на Java, - это и .Net и C++ (с использованием gSOAP), - разработка проходит почти одинаково. Вывод будет такой, - может где-то и удобнее и лучше использовать RESTful сервисы, но не в нашем случае. Почитав статью и комментарии не увидел, для чего это может понадобится.

Ivan Sagalaev

Сразу после вступления статьи есть параграф "Предметная область". Там во втором пункте списка я конкретно ваш тип систем вывел из того, где нужен REST-подход. И собственно это вы и наблюдаете: те проблемы, которые решил бы REST, у вас не возникают.

ДОБАВИТЬ КОММЕНТАРИЙ

Имя или OpenID

Текст

Format with

markdown

Отправить