Итак, давайте посмотрим на последний протокол в рамках данного блока, протокол Protocol Buffers. Много слов – «протокол». Соответственно, разработанный Google-ом. И это второй бинарный протокол в этом блоке. Значит, что, опять же, данные представлены у нас в виде какой-то серии данных, в виде какой-то серии байтов. И должны уже разбираться уже, непосредственно, на машине из бинарного протокола. И не могут прочитаться людьми без специального софта.

Соответственно, работает тоже куда быстрее текстовых протоколов. Это вообще, в принципе, атрибут бинарных протоколов. Поскольку вы не завязаны со строками — это реально быстро.

И соответственно, в чём отличие от Gob, кроме того, что на всех языках и платформах доступен? Он имеет схему для валидации сообщений. То есть в чём смысл? Смысл в том, что в протоколе недостаточно данных для того, чтобы понять структуру этого сообщения. И поэтому у вас на стороне отправителя, на стороне приемника должен быть специальный файлик, который определяет, какое сообщение вам приходит и, собственно говоря, какой протокол, какой состав полей ему соответствует.

Схема для валидации сообщений обратно совместима — это значит, что, если у приёмника более новая схема, у отправителя более старая, или наоборот, соответственно, всё это будет работать.

И вследствие чего, Protobuf применяется для разработки микросервисов, обмена данных между ними. Особенно это важно, когда у вас, соответственно, нужно быстродействие, быстро конвертировать данные туда-сюда. А также потому, что json, например, не представляет никаких схем для валидации — то есть это ваша ответственность, вы должны сверять, что формат сообщения всегда

соответствует. Потому что валидный json может содержать сообщение любого формата. Protobuf не может. Вам нужно описание какого-то протокола.

Соответственно, в некоторых компаниях, например, в нашей компании, Protobuf используется для обмена телеметрией между некоторыми сервисами. То есть мы прямо собираем данные с каких-то запущенных сервисов в этом формате. И на их основе проверяем какие-то метрики, и строим какие-то там отчёты. Потому что очень удобно задать один раз формат. И знать, что всегда всё, что придёт обратно, это вот именно данные в том формате, который нужно. И используем тут тоже…, это в принципе, одно и тоже, что используется он для внутренних проб. Для того, чтобы как раз получить какую-то метрику или понять жива ли node-а. Вообще, поскольку требует меньше накладных расходов на конвертацию и деконвертацию.

Из минусов, соответственно, проба тоже должна знать, как работать с этим protobuf-ом. Поэтому тут надо использовать это аккуратно. Ну и развивая тему, углубимся в protobuf, посмотрим, как с ним работать.

Итак, протокол protobuf. Соответственно, поскольку вы уже познакомились с Gob, с бинарным протоколом, я буду фокусироваться на различиях. То есть protobuf точно также представит все, закодирует все в строку байтов. И будет кодировать из строки байтов на каком-то другом языке, на какой-то другой принимающей стороне, либо в вашем приложении, если вы каким-то таким образом упакуете данные, передадите их между различными приложениями.

И в чём его разница между Go Binary? Go Binary хранит в себе информацию в структурах Go-чных, protobuf хранит в себе только данные. Поэтому Protobuf для того, чтобы функционировать. Для того, чтобы клиент принимающих сообщений сервера, покупающих эти сообщения, могли друг друга понимать, упаковывать и

распаковывать сообщение, существует некий формат называемый – точка Proto, созданный, соответственно, протоколом protobuf для описание своих типов данных.

Соответственно, здесь можно увидеть описание как раз этого типа данных, у него указан синтаксис, какого протокола. Есть Proto2 и Proto3. Соответственно, Proto3 сейчас текущий синтаксис.

Есть, соответственно, сообщение. То есть в данном случае сообщение — это то, что мы пакуем, это тип данных, есть имя этих типов данных, структуры, вернее, данных. Есть уже как раз типы в нём. И есть самое интересное — это имя поля. Когда protobuf пакует сообщение, он кладёт данные, соответствующие этому типу сообщения, в какой-то определённое поле. Зачем это нужно? То есть в битовое поле с каким-то номером. Смысл в том, что таким образом вы не храните имена в этих полях, вы выбрасываете ненужную информацию, сообщение становится меньше. Во-вторых, это позволяет сохранять совместимость. То есть, если у вас разные версии. Если, например, у клиента нету H, но ему придёт H под номером два. Protobuf точно знает, что это у вас не имя изменилось, ничего не переехало. Это значит просто, что у него нет такого поля, он его спокойно пропустит и раскодирует только имя. Соответственно, именно поэтому, когда вы меняете версию протоколов, не рекомендуется, вернее апдейтите протокол, не рекомендуется менять как раз порядковые имена этих самых полей.

Итак, соответственно, после написания вашего файла Proto, вы не можете им сразу пользоваться, нет библиотечки, которая читает этот файл и что-то там распарсивает, согласно ему. Надо сгенерировать package на вашем языке, который уже будет делать всю непосредственную работу: создаст вам необходимую структуру данных, создаст какие-то методы для этих структур.

Существует приложенька Protoc, которое можно скачать с официального GitHub-а Google-а. Там, если написать «Protoc download». Это будет, соответственно, вторая ссылка, 1 ссылка ведёт, на самом деле, туда же, поэтому... Здесь скачаются просто сборки под вашу систему, ставится система, прописываться в (НРЗ 06:24).

И всё. У вас есть вот такая вот штука, но в ней нет по умолчанию Go. Для того, чтобы добавить go, плагин генерации go, вам нужно найти в как-раз Gotutorial. Там, помимо того, что есть офигенный tutorial. Вот на примере, как раз person, что есть massage, в massage-ах есть типы типа enum, которые тоже размапливается в перечисляемые типы. В massage-и могут вкладываться message-и. В massage-е могут быть, соответственно, повторяющиеся элементы, то есть массивы какого-то определённого типа. В нашем случае, несколько телефонных номеров. Ну и вообще, в принципе, достаточно всё удобно и хорошо.

И есть вот такая команда, Go Install, которая установит вам как раз-таки нужный плагин. Соответственно, для Go конкретно можно указать go package. То есть куда вы генерируете ваши файлы. И какой package у них будет. Соответсвенно, есть смысл указать одинаковые папки. И командой Protoc, указывая файлик, и указывая, соответственно, Go Out, для Java-ы будет Java Out, для Objective-C все будет (НРЗ 07:38). Ну и так далее. Вы указываете, соответственно, папочку, куда это генерируется. И собирается вот в такой вот файлик.

Здесь, соответственно, у нас есть ссылки на библиотеки, есть какие-то константы. Здесь всякие различные наши Mapper-ы, remapper-ы, определение String-ов, generic методов. В общем, этот файл генерируется самостоятельно, как и многое в go. И его, соответственно, трогать не надо, им надо просто пользоваться. Соответственно, как им пользоваться? Здесь супер все просто. Мы берём это самый, person из protobuf-а. Мы можем его замаршалить. Здесь почему-то у нас

маршал, здесь не encode/decode, но, в принципе, это хорошо. Это позволяет нам следовать, так сказать, общему формату. В результате маршала мы получаем, соответственно, поток байтов, а потом мы, соответственно, можем этот поток байтов точно также анмаршалить. То есть здесь всё стандартно, всё точно также, как и во всех наших остальных форматах, кроме Go Binary.

Если мы сейчас его запустим, то мы увидим, что у нас есть здесь какие-то данные, а потом всё, значит, замапилось в персону.

Что хотелось бы сказать, подытоживая? Что, на самом деле, все эти маршалы, анмаршалы работают достаточно одинаково. То есть в нашем случае, у вас стандартные операции делаются приблизительно одним и тем же методом, за исключением Go Binary. И переезжая с протокола на протокол, вы особой разницы не заметите, за исключением того, что вам нужно писать новые мапперы для структур данных.