Первый протокол обмена данными, который мы рассмотрим, это великий и ужасный – JSON. Соответственно, JavaScript Object Notation. Наверняка, вы про него уже слышали. Наверняка, вы уже с ним работали. Он встречается практически в веб-приложениях повсеместно. В Ops-овых задачах он достаточно часто фигурирует, фигурирует также в формате описание логов. Это текстовый протокол — это значит, что ваши данные, ваши внутренние представления структур данных каким-то образом конвертируется в текст, и этот текст передается по сети.

Соответственно, в чём большое преимущество? Это в том, что JSON можно читать без дополнительного декодирования. Возможно, с дополнительным форматированием, потому что по умолчанию JSON — это супер-длинная строка. Нужно расставить отступы для того, чтобы вы могли понимать. Но опять же, формат позволяет автоматически это сделать. И вы прекрасно его можете прочитать и понять.

JSON стал очень популярным в последнее время. И встроен почти во все языки программирования. Таким образом, применить JSON без написания своего собственного конвертера, форматера, маршалера, анмаршалера очень просто. Но и где, соответственно, применяться JSON? Правильный ответ – вообще, где угодно. То есть практически везде в вебе он существует.

Естественно, самая большая область его применения это REST API. API для обмена данными между backend-ом и frontend-ом. То есть сайты, когда проектируются, как правило, в них закладывается REST API. Нативным форматом обмена для REST API, является именно JSON. Потому что он достаточно компактный, он хорошо видится в браузере. И самое интересное, он является нативным форматом для JavaScript-а. То есть JavaScript — это основной язык

браузеров, он же единственный язык браузеров, не считая всяких там транскомпилируемых, типа TypeScript и Dart, и прочих.

Соответственно, очень удобно поддерживать frontend. Ну и иногда JSON используют для хранения конфигурации. Например, для хранения конфигурации каких-то логеров, для хранения, когда у вас в конфигурации очень большое количество полей. Тоже удобно, потому что достаточно компактно получается. И соответственно, в принципе, вы этот формат можете встретить у себя довольно часто. И неплохо бы уметь с ним работать. И с Go.

Что мы сейчас и попробуем понять. Давайте посмотрим на JSON в натуральной среде его обитания. В первую очередь — это известная биржа криптовалюты, это не важно, в принципе, просто хочется показать на чём-нибудь динамическом, так сказать, современном, что такое эти JSON-ы. Соответственно, когда мы загружаем биржу, можем воспользоваться отладчиком браузера, который забирает все сетевые запросы, которые он делает. Ну то есть которые индуцируются JavaScript-ом этой самой биржи. То есть, так сказать, фронтовым приложением, frontend-приложением. И что мы здесь видим? Мы здесь видим кучу запросов. Если мы вот кликнем «all», то мы увидим вообще все запросы с картинками и прочим всем. Если мы кликнем «Fetch/XHR» — это будут запросы удалённые, так сказать. То есть запросы, которые к какому-то backend-у обращаются, которые содержат в себе какие-то данные.

Соответственно, допустим, вот запрос у нас. Мы здесь даже уже по адресу видим api/accounts/v1/public/authcenter/auth, то есть это какой-то, соответственно, путь к запросу. Путь, вернее, к какому-то контроллеру, который что-то возвращает. В данном случае авторизацию. И мы видим, что, если мы обращаемся к методу по этому URL-у, с какими-то заголовками, с методом post, об этом… Мы разберемся с этим немножко подробнее позже, в цикле лекций.

Соответственно, передаём туда что-то, а в ответ получаем, это важно, сейчас я покажу. В ответе мы получаем content-type: application/json, то есть тот самый JSON мы получаем. И здесь, если в сыром виде, он вот так выглядит. Тут есть, естественно, в браузере встроенное форматирование. И вот мы видим поля: code, message, success. Соответственно, тоже самое, identification лист у нас там будет, какие-то asset-ы. По кодам у нас получаются коды валют, то есть здесь какие-то массивы у нас в сыром виде.

Это, естественно, какая-то длинная строка, разделённая определённым образом. Кстати, как она разделена? Есть фигурные скобки, которые обозначают начало и конец объекта. Есть, соответственно, название полей. Есть их значение. Соответственно, значение: они бывают строковые, бывают они никакие, есть пустые, бывают массивы. Массивы отделяются квадратными скобочками. Соответственно, бывают цифры. То есть целочисленные и не целочисленные, и так далее. Собственно говоря, вот такой JSON в натуральной среде обитания.

Давайте посмотрим теперь, как мы с ним будем работать. Соответственно, вот здесь у меня есть пример - большого файла JSON. Вот так они обычно выглядят, если они прямо пришли откуда-то с машины обработки. Машинам не нужны отступы, не нужны даже знаки переноса строк. Грубо говоря — это всё может выглядеть так, без всяких пробелов.

Повторюсь, что формату JSON нужны служебные символы, нужны вот фигурные скобочки, нужны двоеточия, нужны запятые. Нужно разделять название полей, соответственно, кавычками. И в элементах нужно чётко соблюдать структуру какую-то. То есть для булевских переменных это True, false, слово без кавычек. Для int-овых переменных — это цифры без кавычек. Для строковых — это, соответственно, кавычки. В чём смысл? Смысл в том, что для человеческого восприятия, то, о чём я как раз говорю, вам нужно этот JSON отформатировать.

То есть в моей IDE Golang, также, как в любом другом продукте компании JetBrains, можно выделить текст, нажать CTRL ALT L, и он к нему применит форматирование. Если знает, какое оно должно быть.

Соответственно, для JSON-а форматированием является один пробел, для от отступов, то есть, простите, два пробела для отступов. И соответственно, 1 пробел для значения. И вот таким образом вы можете видеть JSON уже разобранный. Его можно сворачивать, потому что у меня файл большой, то есть, соответственно, 272 строки. Читать это всё не всегда удобно. И как-то так, то есть давайте теперь уже посмотрим, как JSON-ы уже, непосредственно, разбираются.

Соответственно, давайте посмотрим на простой пример, где мы разбираем JSON. Для этого примера я создал две структуры, тип птичка. С species, description, dimensions полями. Соответственно, поле species является строкой, как и description. И dimensions является другой структурой, в которой есть поля height и width.

Для разбора примера JSON-а, я не стал брать этот большой файл, который мы видели. Потому что в этом нет смысла. Это огромная структура, но принцип ровно такой же.

Соответственно, в чём здесь смысл? Мы объявляем строку здесь, строку в формате JSON. То есть вот она у нас. Соответственно, как можно видеть, она такая целиковая, в одну строку, без всяких там больших переносов и пробелов. И мы создаём тип Bird. То есть вот эта структура Bird, она пустая. Это просто ссылка, которая будет заполнена анмаршалером. И дальше переведем поток байтов в анмаршалер. Соответственно, все маршалеры, анмаршалеры в go стандартные и стандартного комплекта поставки. Вот отсюда они работают со строкой байтов. Для того, чтобы мы могли туда, например, подконектить какой-то стример,

работающий, например, с сетью. То есть это какой-то общий интерфейс. И, помимо этого, мы передаем, соответственно, ссылку на нашу структуру, которую мы создали, которая будет заполнена анмаршалером. Соответственно, анмаршалер возвращает ошибку. Ошибка, соответственно, может быть пустая в случае, если всё прошло хорошо. В силу отсутствия дженериков, мы не можем здесь объявить какой-то дженерик, который он…, его сам ещё и сконструирует, и вернёт. Соответственно, мы должны передавать туда переменную.

В чём дальше смысл? Вот мы видим этот файлик – тут, значит, есть species с маленькой буквы. Но это не важно для маршалера. Description, который смапится в поле. И dimensions, который смапится в другой класс height и width. Здесь, кстати, могла быть и (НРЗ 09:20), потому что это, по сути, говоря, строка и число, то есть они спокойно (НРЗ 09:29). То есть JSON, здесь видно, что здесь отдельный объект. Поэтому мы создали отдельный объект здесь. Отдельный объект выделяется фигурными скобочками. И после этого всего, мы напечатаем, что у нас получилось в поле. Соответственно, в поле у нас смапятся вот эти вот значения полей по именам. И должна вывести, всё в порядке.

Давайте посмотрим. Естественно, всё нормально. То есть это стандартное форматирование Go для вывода строкового, для вывода структур. То есть у нас есть здесь, значит, птичка, у нас есть птичка, которая likes to perch on rocks. И у нас есть, соответственно, здесь измерения. 24 и 10, высота, вернее, высота и ширина. Но, соответственно, в принципе, здесь всё, если мы сломаем JSON, то сработает вот эта часть кода, и выдастся ошибка.

Ошибки анмаршалеров в go, они не самые очевидные, они достаточно абстрактные. Поэтому для того, чтобы…, если у вас есть какая-то ошибка, лучше воспользоваться каким-то форматором.

То есть он либо построен в IDE, можно файлик закинуть, посмотреть, что не так. Либо вот есть всякие интернет-решение, то есть в данном случае он сейчас скажет, что он валидный JSON. Вот этот JSON formatter. Если убрать, то вот он покажет, что, значит, multiple JSON root elements, extra closing (HP3 11:10). То есть здесь прям куча будет ошибок, но хотя бы они будут подсвечены, там будет обычно сразу видно, в чём у вас проблема. То есть, если я здесь уберу, например, запятую, то вот сразу понятно, что вот здесь где-то есть ошибка, ожидается запятая. Ну и так далее.

То есть в базе очень, как мы видим, простой процесс. Объявляем…, значит, откуда получаем строку JSON-овскую, объявляем класс, структуру, куда она смапится, вызываем метод unmarshal у JSON-а, проверяем на ошибки. И у нас есть готовая структура, которую можно вернуть, и как-то с ней дальше работать.

Давайте теперь посмотрим, как собрать JSON. Можно заметить, что я не делаю обзоры, как делается маршалинг и анмаршалинг? Потому что это достаточно бесполезные знания — это рекурсивный алгоритм. Разбирается строка, проверяются типы полей. И соответственно, если поле — это другой объект, то всё вызывается по новой, если тип полей какой-то тип данных, то соответственно, вызывается обработчик для этого типа данных. Потом смапится все, соответственно, названия полей в нужный вам класс.

Там довольная большой код, он расположен в методе decode для анмаршалинга. И расположен в методе, соответственно, encode для анмаршалинга JSON-ов. То есть не в методе, а файле.

Давайте посмотрим на маршалинг. С маршалингом всё ещё проще. Мы создаём, соответственно, структуру данных, мы засовываем её в маршал, то есть практически любая структура данных может быть замаршалена, если её поля

умеют кастоваться в строки, либо являются строками. И соответственно, маршалер всё сделает за вас, вернув вам, соответственно, поток байт и ошибку. Ошибка, если у вас по какой-то причине что-то не маршалится. И соответственно, вы можете это напечатать.

Давайте посмотрим, как это выглядит. Соответственно, вот так это выглядит? Можно видеть, что у меня здесь какая-то строка, можно сразу напечатать хорошо форматированную строчку. В таком случае, можем вызвать метод marshalindent, то есть отмаршалить форматированную строчку, в которой будет префикс, и в которой будет какой-то отступ. Отступ может быть 1 пробел, 2, может быть что-то ещё, как уже вам угодно. Соответственно, с двумя пробелами — это форматируется, выглядит вот так. Можем здесь поиграться и посмотреть. Соответственно, видите, вот здесь мы видим какие-то элементы. Ну и можем поставить ему префикс. Это будет строка, которая добавится перед, соответственно, каждой линией. На мой взгляд, это довольно-таки ненужная тема, но я опять же не буду говорить за всех разработчиков.

В данном случае, мы можем как-то красиво отформатировать JSON. Например, сохранить его в файл. Если ваше приложение, например, берёт JSON откуда-то пришедший, как-то их обрабатывает и сохраняет их красивенько в файл, чтобы вы уже могли их посмотреть. В данном случае будет удобная штука.

Поехали дальше. Давайте посмотрим еще, как парсить, например. То есть стандартная спецификация, особенно на всяких веб-сайтах. Как бы объектно-оборачивание JSON-а — это стандартное представление. Бывает, что вам приходит массив. Как его спарсить? Да, точно, собственно говоря, также. Вот у вас здесь пришёл массив, можно видеть, квадратное скобочки. У вас есть…, вы делаете для него, какой-то slice, то есть в данном случае, slice типа String. И точно также парсите. У вас на выходе получится массив. То есть здесь можно видеть

One, Two, Three. Ввожу нулевой элемент. Соответственно, у вас One. То есть можно посмотреть на этот slice, всё здесь в порядке. Таким же образом, можно, кстати, парсить и slice-ы внутри каких-то полей класса. Всё абсолютно спокойно поддерживается.

Давайте теперь поговорим о двух моментах, которые мы ещё не разобрали. Как, соответственно, читать переменные, как читать JSON-а из файла. Вернее, мы, в принципе, уже знаем, как вообще читать, что угодно из файла. Но вдруг что-то отличается и, соответственно, как мапить поля. То есть, совершенно, не обязательно вам в JSON-ах называть поля — вот таким образом, вернее, не обязательно вам в структурах называть поля таким образом, как они названы в JSON-ах.

Давайте посмотрим, соответственно, на первый пример. Здесь мы загружаем файл из example.JSON структуру request, которая в себе содержит поле request и поле author. Можно увидеть, что поля author нет, вообще нет авторов. У нас есть только user-ы. И соответственно, в поле request Content у нас будет User и message. И поля massage здесь тоже нет, есть поле msg. И соответственно, для этого применяется… Для того, чтобы смапить что-то, применяются теги. Теги — это…, теги полей – это это фича go, которая позволяет вам вызывать метод специальный для просмотра, соответственно, структур, для просмотра полей структур. И у этих полей структур, если вы объявили какой-то тег, соответственно, будет тег. И в данном случае тег состоит структурно из 2 единиц. Для чего тег, какого типа тэг, и значение этого тега. В нашем случае, это просто тег типа JSON, со значением msg. То есть это, собственно говоря, указание анмаршалеру на то, что в JSON-е это поле будет называться иначе. То есть всё остальное очень просто. Тут user у нас смапится в author. Msg у нас смапится в message. Всё просто, если мы запустим этот метод для чтения из файла, мы увидим, да, нас

есть admin trying to send a massage, соответственно, это поле request, и у нас есть здесь author – это admin.

В принципе, давайте теперь посмотрим, как ещё можно спарсить вот это вот сообщение. Ещё момент, если вы не уверены в том, какие вам будут типы данных приходить, либо вам нужно как-то динамически их обрабатывать. Вы знаете, что какое-то поле будет такого типа, какое-то будет такого типа. Вы можете это сделать сами путём маппинга JSON-а в map interface от string-а. Таким образом, после анмаршалинга, у вас будут какие-то строковые поля. Можно заметить, что, собственно, это просто страховые поля, а внутри них будут лежать какие-то совершенно случайные типы. Если мы вот так замапим, заанмаршалим наш example в просто map строку от interface-а, то мы увидим, что у нас в objmap-е лежит, соответственно, ключ request, со значением Map string, в данном случае. Мы можем его скастить обратно. И получить Map string, если мы знаем, что там. И в нём уже лежат, соответственно, просто строки. И здесь, соответственно, тоже самое, то есть это вполне возможно.

Также вы можете, если вы знаете, что какие-то поля будут перемаплены, то есть, смотрите, у вас есть какой-то большой JSON, у вас есть какая-то структура. И вы знаете, что пара полей вам нужны, все остальное вам не нужно. То есть, допустим, у вас есть message, description, не знаю, IP-адрес, user (НРЗ 19:13), ну и прочее. То есть какая-то большая интернет-портянка. А вам нужно, из всего этого, только message, который вы знаете, что, скорее всего, будет строкой. Вы можете смапить это не полностью в структуру или писать большую структуру, и не писать Map-у. А вы можете это смапить, в так называемый, JSON raw message. То есть, что это такое? На самом деле — это просто массив байтов, который позволяет над собой выполнять операции анмаршалинга точно также. То есть здесь как это происходит? Мы маршалим наш example, вот эту нашу штуку, в, соответственно, вот в этот objmap. То есть это строка в массив байтов. А потом, когда мы уже

знаем, какого типа у нас это конкретное поле, мы можем ещё раз его размамапить. И соответсвенно, сэкономить на этом достаточно много ресурсов. Особенно, если JSON-ы у вас большие.

То есть выглядит это как-то вот так. Здесь у меня, соответственно, есть objmap. Вот можно видеть, что здесь какие-то у нас байты. На самом деле, видно, что это, по сути, говоря, вот эти строки JSON-овские, а потом мы берём просто, соответственно, request, вот эту вот часть, и мапим его уже в string Map-у. Получаем, соответственно, наш message.

Если мы включим отладку, то мы увидим, что у нас здесь такой тип есть, JSON raw message. И соответсвенно, после ремапинга, у нас уже появляется тип Internal Map, уже с нормальными ключами и значениями. Как-то так.

Давайте теперь посмотрим, как кастомно мапить и размапливать поля. И пойдём дальше. Итак, последнее, о чём хотелось бы рассказать применительно к JSON-у — это кастомный анмапинг. То есть в чём смысл? Вы, например можете иногда встретить вот такую историю, что у вас есть request, в request существует поле, например, tags. И в тегах у вас через запятую теги.

Бывает такое, что у вас там не массив, а физическая строка. Она в базе так хранится. Либо какой-то формат был старый, который на JSON перевезли, но как бы не переделали все поля.

В общем, периодически приходится с таким работать. Вам там нужно, например…, вам там нужен массив, а у вас там строка. Соответственно, здесь два пути есть. Здесь можно разобрать в tags, положить строку, а потом эту строку спикануть по запятой. А можно, но уже на этапе анмаршалинга, соответственно, решить эту проблему. Для этого существуют специальные механизмы внутри анмаршалера,

который вызывает кастомные методы у полей. Если у структуры, представляющей эти поля, такие методы есть.

То есть, как у это происходит? Вот наш старый знакомый – request. Он переименован, потому что я нахожусь в том же package-е, что и старый, нельзя называть одновременно две структуры одним и тем же именем. Соответственно, здесь отличие в том, что в структуре request Content добавилось поле tags, представляющее собой тип strslice. То есть это маппинг slice-а над строкой, у которого есть метод анмаршал JSON, принимающий в себя байт, массив байтов. И соответственно, производящими на них какими-то действия. То есть в данном случае, после анмаршалинга. То есть получается у меня строка, я эту строку сплитую по запятой. И присваиваю, то есть (НРЗ 22:53), короче, мой strslice. То есть то, что там было, присваиваю ему результат, возвращаемый с функцией Split.

Таким образом, при запуске нашего анмаршалера, в поле request у меня образуется уже slice — это не строка. Это удобно, когда у вас нужно переделать какие-то форматы прям при парсинге, и вы не хотите над этим долго думать. После, в коде избежать многих весёлых моментов, сделав это отдельно. Можно таким образом как бы отделить вот этот весь анмаршалинг в отдельный тип. И разделить его легко. Ну и, соответственно, это иногда просто реально удобно.