Итак, второй формат данных, наверное, самый популярный, после формата JSON, либо наравне с ним по популярности — это XML, eXtensible Markup Language.

Соответственно, из слова Markup можно сделать вывод, что это скорее язык разметки, чем язык какой-то передачи данных. Но, на самом деле, достаточно универсальный язык получился, формат. И, соответственно, в этом формате можно передавать данные, оборачивать.

Он тоже текстовый, он тоже читается, соответственно, людьми, в отличие от JSON. JSON сосредоточен как бы на том, чтобы представлять данные. XML - это всё-таки какие-то теги. Но такой же, в принципе, того же назначения, как HTML, но вот так вот получилось, что в нём ещё можно удобно хранить данные, если представить этот tag based формат. Мы на это посмотрим.

Соответственно, читается людьми без дополнительного декодирования. Единственное, что удобнее читать с дополнительным инструментарием, потому что довольно многословный язык, довольно длинные конструкции. И удобно иногда что-то мочь свернуть и так далее. В текстовом просто формате у вас таких вариантов, естественно, нет.

Именно для задач хранения и передачи данных он более многословный, по сравнению с JSON. И менее удобный, потому что есть, значит, тег, в нём есть значение, есть в теге ещё атрибуты. И это не очень удобно для представления каких-то данных, потому что есть много способов их представить. Поэтому в JSON у вас один способ. Он более строгий, да. В XML у вас способов несколько.

Для чего его применяют? В некоторых случаях, довольно олдскульных случаях, ещё применяется для API, если вы работаете в банковском секторе, в финтехе, в каких-то бранчах, каких-то ветках, индустриях, где очень старый код, где очень старые протоколы, то скорее всего вы этот XML точно встретите.

Что в XML удобно делать, это хранить конфигурацию. Многие фреймворки хранят конфигурацию в XML, потому что как раз за счёт вот этой гибкости, что можно положить, положить как бы значение, можно положить атрибуты.

Ну, и очень часто, вернее не то, что очень часто, это, в принципе, стандарт индустрии, он используется для описания визуальных форматов. Например, в Андроиде, в iOS, как ни странно, используется XML для описания экранов мобильного приложения. То есть там есть отдельный дизайнер. Вы накидываете туда элементы, и можете всегда посмотреть, какой у вас XML сгенерировался.

Соответственно, в Андроиде это очевидно прямо. XML в iOS - это какой-то свой формат. Но в итоге, это всё равно XML-based, так сказать, формат. То есть, довольно часто его можно встретить в наших задачах, задачах DevOps и веб-разработке. Ну, всё-таки реже вам придётся работать с XML именно в формате визуальной презентации.

Ну, и давайте посмотрим, что значит работать с XML, и как с ним работать. Давайте посмотрим на XML в его реальной среде обитания. Соответственно, это вот ответ на какой-то там API от портала, насколько я помню, это сохранение листинга с eBay.

Как мы можем видеть, здесь у нас массив, здесь уже всё не так очевидно. Просто у нас есть куча одинаковых элементов, то есть, вот здесь, соответственно, у нас есть входные теги, есть открывающий тег, есть закрывающий тег, внутри этого, этих тегов, у нас уже лежат отдельные подтеги, да.

Называется вот это всё вложенность, по-английски, nesting. Соответственно, в root есть nested элементы, то здесь, на самом деле, массив у нас. Вот они все одинаковые эти листинги. А внутри уже листингов какие-то у нас объекты. Seller

info, seller name. Уже внутри, соответственно, seller name у нас есть какое-то строковой значение. Как можно видеть, здесь нет никаких уже кавычек.

Ну, как бы вот, как-то так оборачивается, все ответы от API.

Удобно это или нет, решать вам. Интернет, в принципе, решил, что JSON удобнее. Я склонен с ними согласиться, по той причине, что, ну, достаточно это тяжело читается, даже с учётом того, что у меня в браузере есть и форматирование, и подсветка этих полей, и их свернуть можно. Мне кажется, что это довольно, ну, довольно плохо подходящая для обмена именно данными формат. Но тем не менее, существует API, например, у Google Doc, у Google Cloud. Существует API с XML-ем.

Вот здесь ещё можно увидеть, так называемые, заголовки XML, которые говорят, какой версии этот XML и какая у него кодировка. И можно увидеть, что ещё используются атрибуты. Вот, то есть, это элемент. В нем лежат какие-то другие элементы. У них есть, значит, свои там, соответственно, значения. А есть атрибуты у этих элементов. И можно ещё заценить, где применяется XML, помимо формата обмена данными.

Соответственно, я нашёл приложение доблестной компании Nextcloud. Их, соответственно, клиент под Android. Это Open Source приложение. Здесь можно увидеть стандартные Android приложения. Поэтому, если мы зайдём в исходники этого приложения, соответственно, вот уже в src/main мы видим XML файл с какими-то там конфигурациями, с какими-то комментариями. Это комментарии, кстати. Кто работает с HTML, видит, что они абсолютно идентичные. И здесь какие-то, значит, у нас есть элементы. Это корневой элемент. Здесь у него подэлементы.

Как можно видеть, в Андроиде это широко используется для, например, хранения настроек. Ну, что, в принципе, достаточно просто читается, достаточно нормально понимается. Например, используют фичу Android Camera, но она не требуется для запуска приложения. Ну, и так далее.

Вот, и также используется оно для lay outing, для описания всяких там экранов. То есть, давайте посмотрим там, вот, например, там какой-то список есть. У этого списка существует Layout. Вот тут, значит, у нас вложенные элементы. В принципе, ну, если вы с HTML работаете, вы понимаете, да, что это наверно самая сейчас удобная структура.

Здесь у нас то же самое, здесь у нас, значит, строки. Да, только это HTML, а это XML. То есть, разница в том, что в HTML формат несколько другой и, соответственно, теги другие. В XML, ну, примерно, примерно это всё похоже.

Ну, вот как-то так XML живёт в реальном мире, в Android приложениях, в iOS приложениях. Там lay outing точно также построен. Для хранения настроек во многих фреймворках, и иногда в структурах обмена данных.

В своей DevOps-овской практике я с XML много не работал, много их там не видел, скажем так. Как разработчик видел несколько раз протокол обмена, в основном банковские всякие, которые использовали XML.

Давайте посмотрим, как в Go работать с ним. Итак, при маппинге XML первое, что нам нужно понимать, это то, что у каждого XML ест свой тег. Тег мы определяем сами. Название тега - это, соответственно, поле xml.Name в любой структуре. Без поля xml.Name он не поймет, какой тег, и, соответственно, не сможете его разобрать. Мы здесь видим, что здесь сразу у нас есть теги, соответственно, есть тег Plant. Это значит, что XML.Name для нашей структуры будет plant. То есть, открывающий plant, что-то там внутри, что-то там закрывающее.

У Plant есть теги Name, которые являются, содержат в себе строки и теги Origin, которые являются массивом строк. Соответственно, давайте посмотрим, создадим XML и посмотрим, как он выглядит. Так, здесь я пока уберу метод декодирования. Запускаем.

И вот видим, да, что у нас есть plant, у него есть id. Здесь обратите внимание на, значит, после запятой мы указываем, что для xml, что id — это атрибут этого plant. И потом, что у нас по имени name, соответственно, содержится, содержится кофе, и его родина, происхождение — это Бразилия.

Дайте посмотрим теперь, как XML декодируется. Тут тоже, в принципе, с декодированием ничего сложного нет. Создаём переменную Plant, анмаршалим её, обрабатываем ошибки и, соответственно, печатаем этот plant.

То есть, здесь абсолютно всё то же самое, что у нас было бы в JSON. Точно такой же byte, точно такая же структура. Но структура немножко другая и, соответственно, вместо JSON, анмаршал у нас xml.Unmarshal**.** А в целом всё работает точно также. Раскомментируем её и посмотрим, как что у нас происходит. Значит, видим здесь, что здесь есть кастомные переопределения метода String. То есть, выведется нам более симпатичная, симпатичная репрезентация типов Plant.

Соответственно, вот мы видим, что, значит, id 27, Name у него кофе и страна происхождения Эфиопия, и Бразилия. В принципе, ну, так-то то же самое, что у нас с JSON. Но, поскольку XML несколько отличается от JSON, и там может быть много вложенных тегов, есть возможность их пропускать. Как? Мы сейчас узнаем. Бывают у вас XML такого рода, у вас есть какая-то XML, вернее, у вас есть какой-то тег, в нём лежит ещё какой-то тег, в нём лежит ещё какой-то тег, и только в нём лежит что-то полезное.

На самом деле, такое бывает достаточно часто. И эти данные, они обычно являются мусорными, то есть, они вам не нужны по большому счёту. И чтобы из XML можно было пропустить мусорные данные, либо пропустить части, которые вам не нужны, у вас есть возможность определить его, так называемый, nesting. Вот здесь, в данном случае, мы прямо внутри определяем структуру. Здесь я на ходу собираю XML. Внутри структуры мы определяем XML тег nesting и просто растения, то есть, просто Plants.

И говорим, что, значит, есть parent, в нём существует child, и в нём существует plant. И вот этот plant надо записать в Plants. То есть, мы не делаем, что, значит, у нас есть parent, у нас есть потом child, у нас есть потом plant, и вот у нас там вложенные структуры друг друга. Мы этого ничего не делаем. Мы просто берем и пишем, соответственно, в Plants.

И тут дальше всё, всё то же самое. То есть, здесь я собираю структуру. У нас, у меня получается массив байт, и потом просто анмаршелю её в, соответственно, result, который является типом nesting, и получаю, получаю, собственно говоря, его и получаю. Вот эти вот растения, и без определения нескольких вложенных надтипов структур.