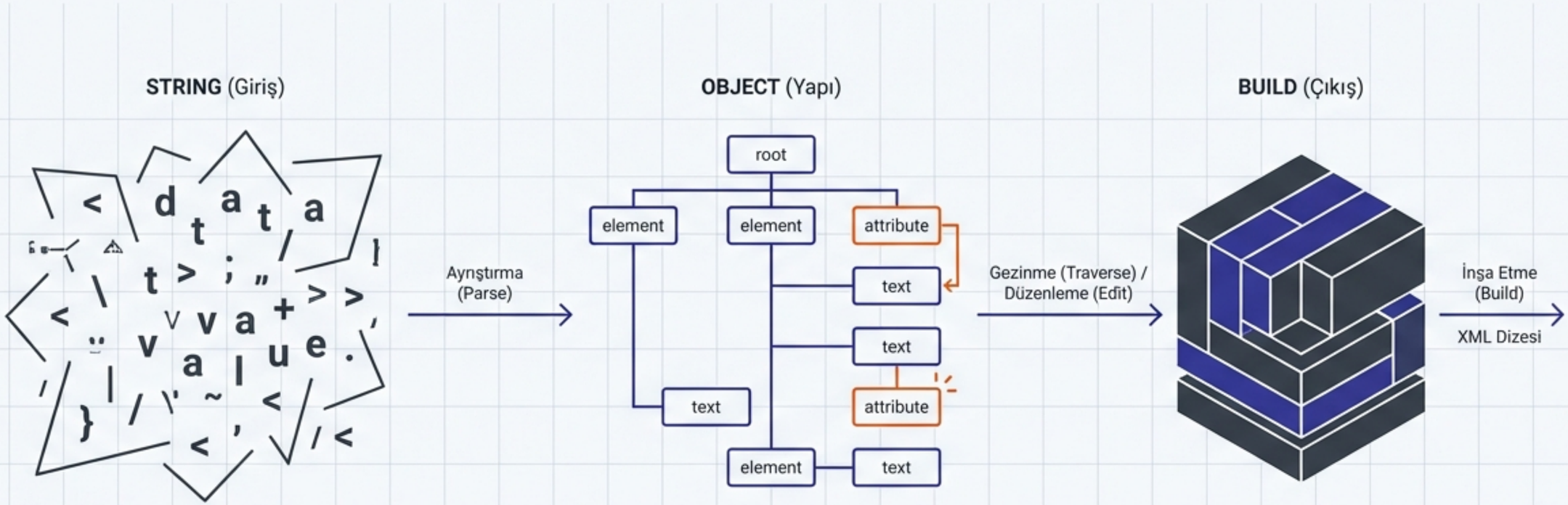
Dart ile XML İşlemleri: Ayrıştırma, Gezinme ve Oluşturma

`xml` paketi ile veri manipülasyonunda ustalaşın.

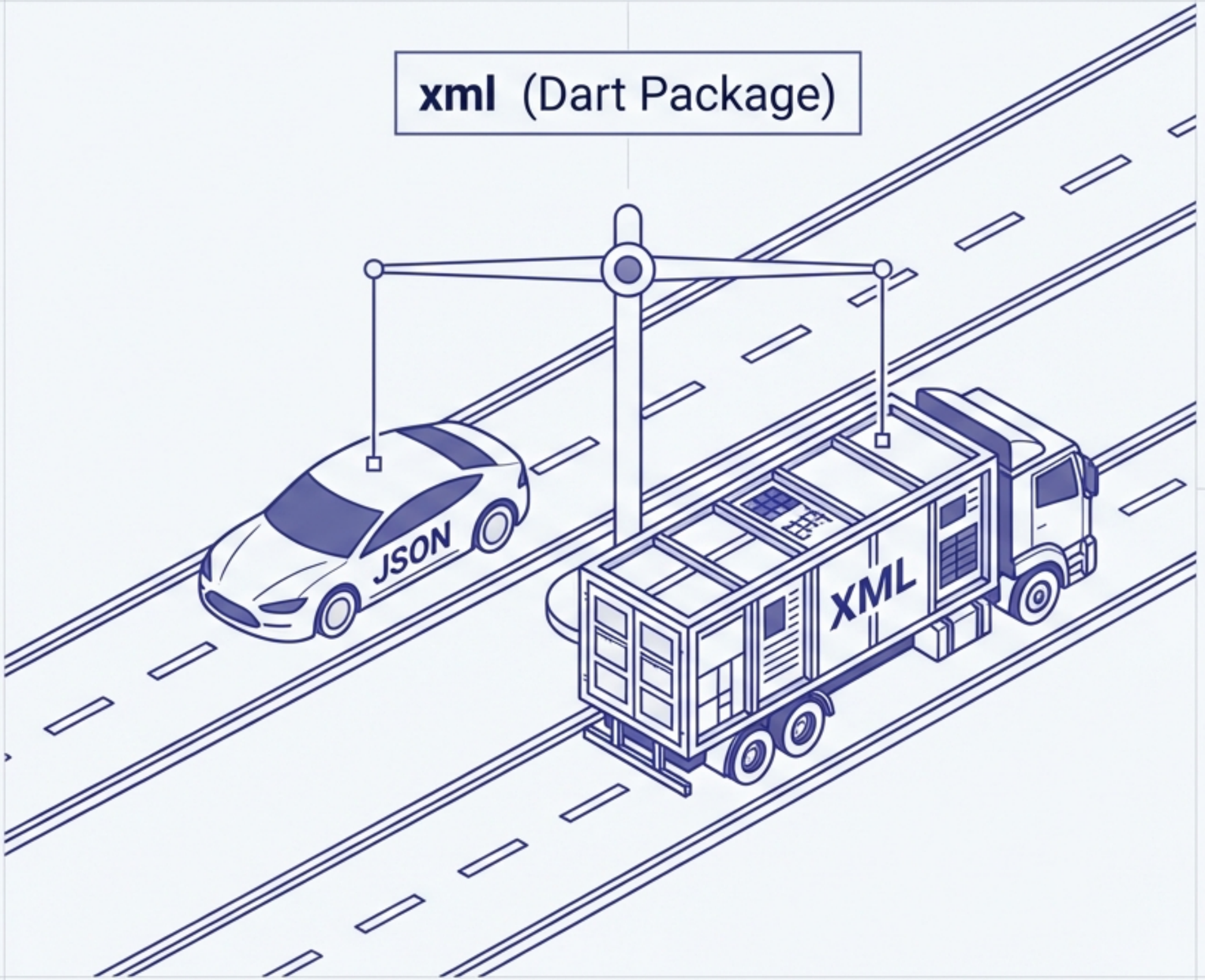


Ayrıştırma (Parsing) • Gezinme (Traversal) • İnşa Etme (Building)

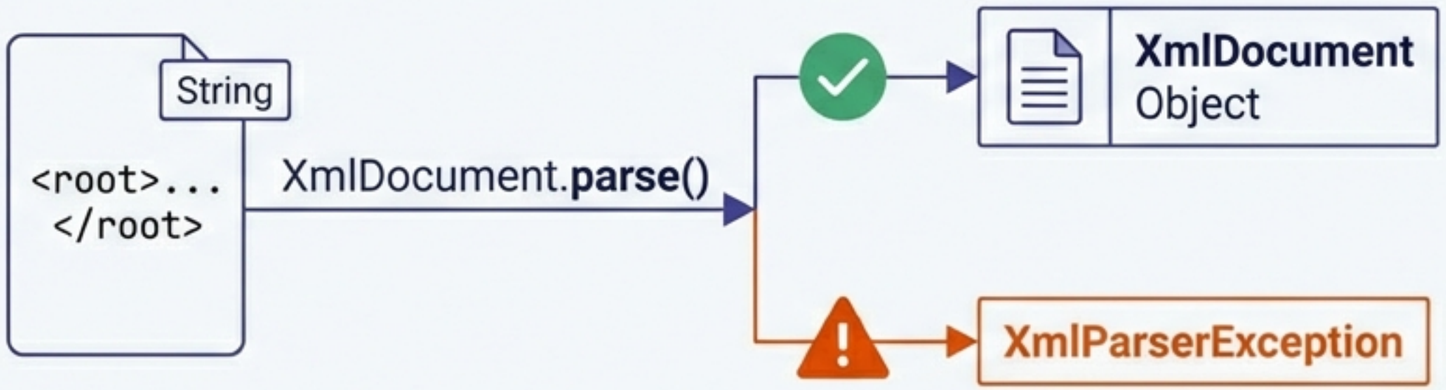
JSON çağında XML hala güçlü bir standarttır.

Veri değişimi için hala yaygın olarak kullanılan XML formatını yönetmek, her Dart geliştiricisinin alet çantasında bulunması gereken bir yetenektir. Bu rehberde, Lukas Renggli tarafından geliştirilen ve endüstri standardı olan `xml` paketini kullanacağız.

Sürüm 4.1.0 ve sonrası için güncel yöntemler içerir.



Ham Veriyi İşlenebilir Nesneye Nesneye Dönüştürmek



> `**XmlDocument.parse()**` metodu, eski üst düzey `parse()` fonksiyonunun yerini almıştır.
>
> **XmlParserException** ile bozuk (malformed) XML yapılarını güvenle yönetin.

Sürüm 4.1.0 ve sonrası için güncel yöntemler içerir.

```dart
import 'package:xml/xml.dart';

var xmlString = '<root>...</root>';
try {
  // String'i nesneye çevir
  final xmlDoc = XmlDocument.parse(xmlString);
} on XmlParserException {
  // Hatalı sözdizimi yönetimi
}
```

Çıktıyı Okunabilir Kılmak (Pretty Printing)

Raw String

Pretty Print

```
- root
  ├── child: data
  └── child: data
```

```dart
final output = xmlDoc.toXmlString(
  pretty: true,
  indent: '-' // Varsayılan boşluk yerine tire kullanımı
);
print(output);
```

`pretty: true` ayarı gereksiz boşlukları temizler ve hiyerarşiyi düzenler. `indent` parametresi ile (örneğin `\t` veya `--`) girinti stilini özelleştirebilirsiniz.

Kritik Tuzak: Karakter Kodlaması (Encoding)

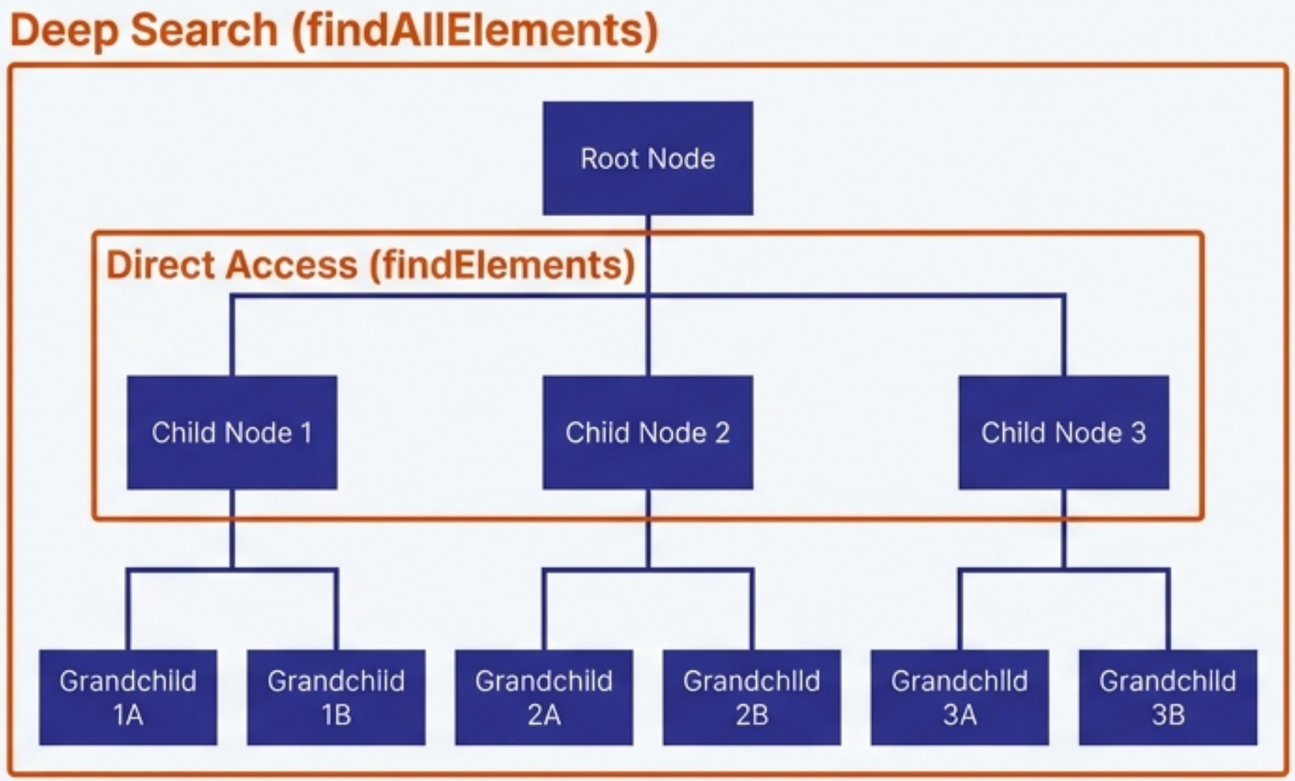
Sorun: XmlDocument.parse() sadece UTF formatlı stringler ile çalışır. HTTP GET isteklerinden gelen veri (örneğin Latin1) doğrudan işlenirse hata alabilirsiniz.

```
final response = await http.get('https://.../file.xml');

// Response body'yi güvenli bir Dart String'e (UTF-16) çevirin
final String xmlString = Latin1Decoder().convert(response.bodyBytes);

final xmlDoc = XmlDocument.parse(xmlString);
```

response.body her zaman doğru kodlamayı garanti etmez. Latin1Decoder veya Utf8Decoder kullanarak bodyBytes üzerinden dönüşüm yapın.

Veri Ağacında Gezinme Stratejileri

1. **Deep Search: findAllElements('name')**
   - Ağacın neresinde olursa olsun, ismi eşleşen *tüm* düğümleri bulur.
2. **Direct Access: findElements('name')**
   - Sadece *mevcut* düğümün doğrudan çocuklarını (direct children) tarar.

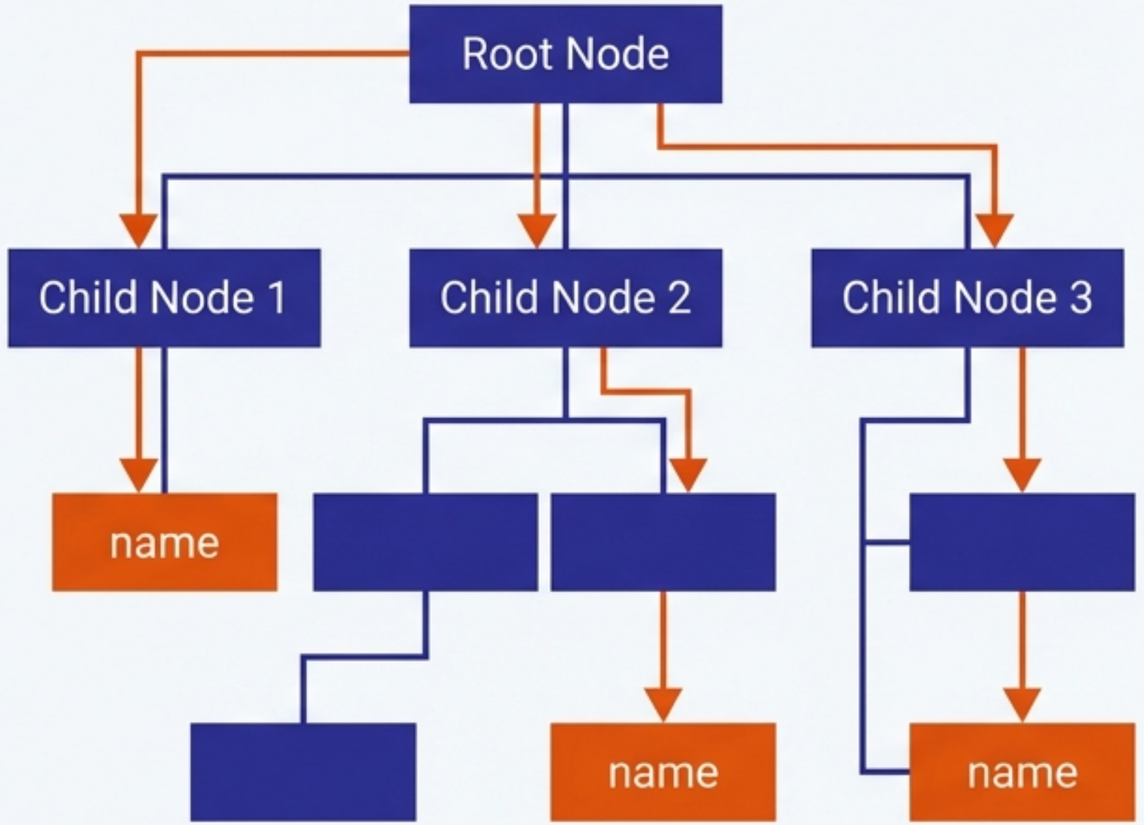



```
xmlDoc.findAllElements('book').map((node) => node.text)
```

Derinlemesine Arama: findAllElements

Senaryo: XML içindeki tüm kitap isimlerini listelemek.

```
xmlDoc.findAllElements('name')
   .map((item) => item.text)
   .forEach(print);
// Çıktı: Book, Tablet
```



Bu metod yinelemeli (recursive) çalışır. Root'tan en uç yapraklara kadar tarama yaparak eşleşen etiketlerin tamamını Iterable olarak döndürür.

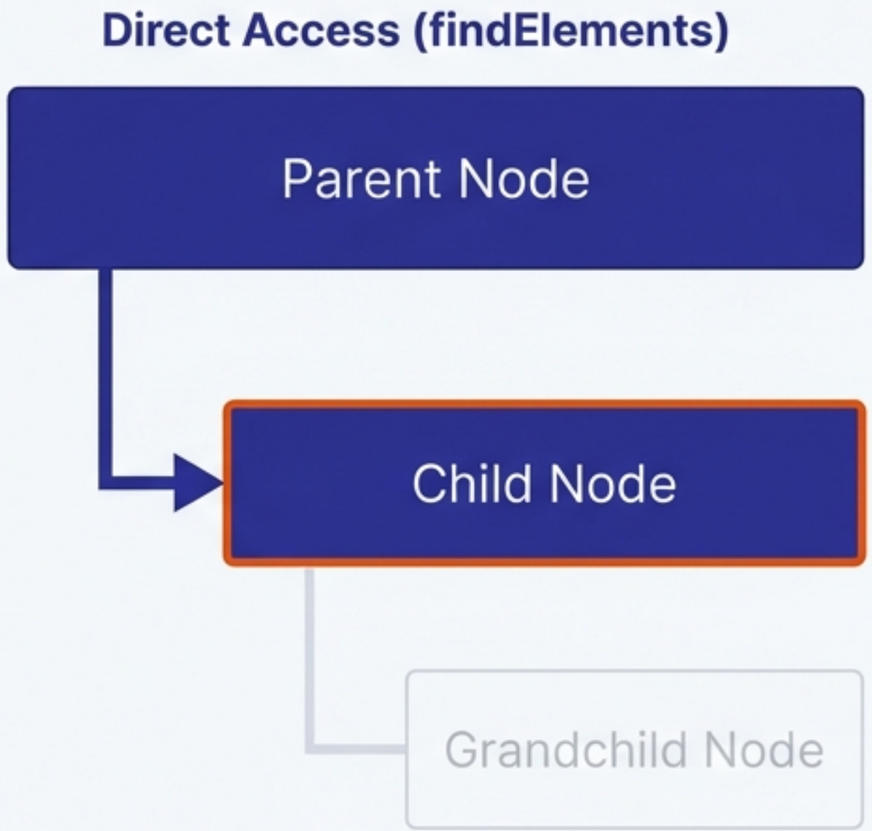
Doğrudan Erişim: findElements

Sadece bir alt seviyedeki (child) veriye odaklanmak için kullanılır.

Yardımcı Getters

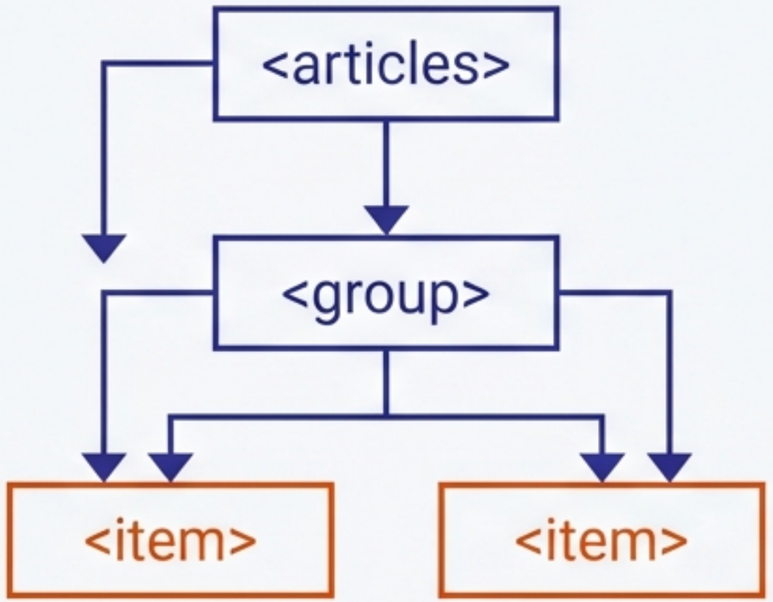
- **single**: Tek bir eşleşme bekler, yoksa veya birden fazlaysa hata fırlatır.
- **first**: İlk eşleşmeyi getirir.
- **last**: Son eşleşmeyi getirir.

```
final name = item.findFlements('name').single.text;
JetBrains Mono Medium
```



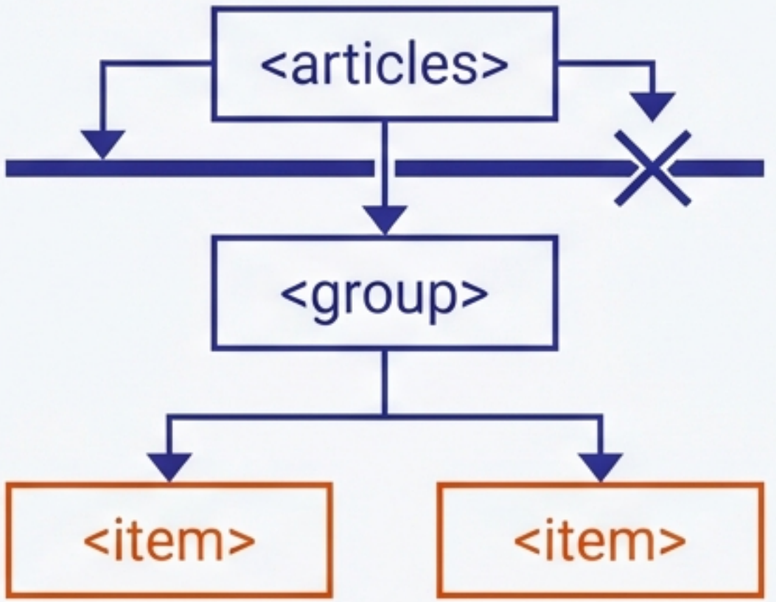
Karşılaştırma: findAll vs find

findAllElements('item')



SONUÇ: 2 Öğe. (Derinlemesine tarama başarılı)

findElements('item')



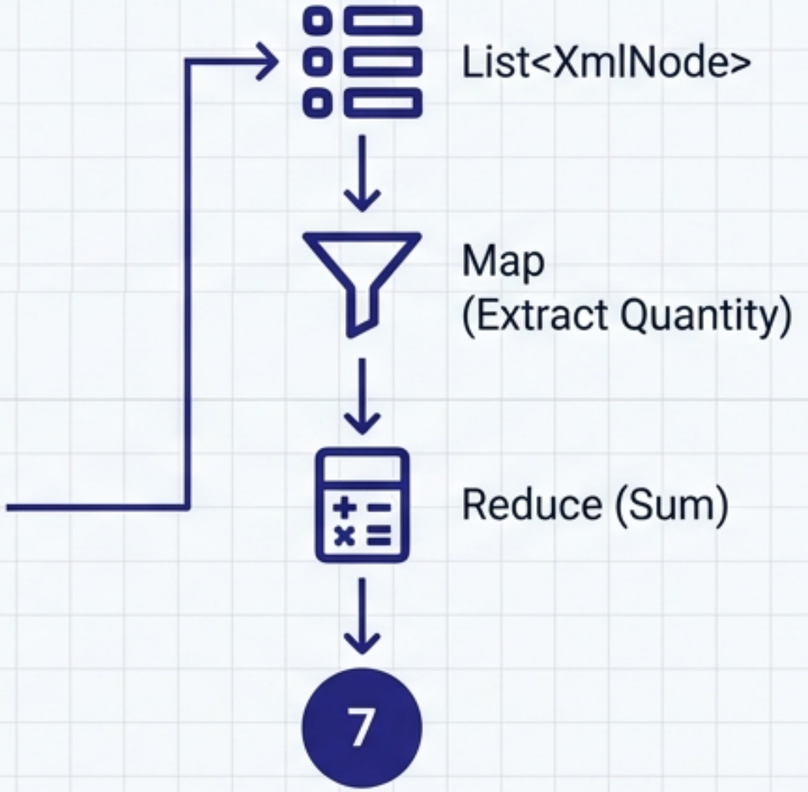
SONUÇ: Boş. (Kök altında doğrudan 'item' yok)

parse() işlemi sizi kök (root) düğüme koyar. Doğrudan çocuk olmayan veriler için findAllElements kullanın.

Gerçek Hayat Senaryosu: Veri Agregasyonu

Hedef: Tüm ürünlerin miktarlarını (quantity) toplamak.

```
final total = xmlDoc.findAllElements('item')
  .map((node) =>
    // Detayları çek
    int.parse(node.findElements('quantity').single.text)
  )
  .reduce((a, b) => a + b); // Topla

print(total); // Çıktı: 7
```

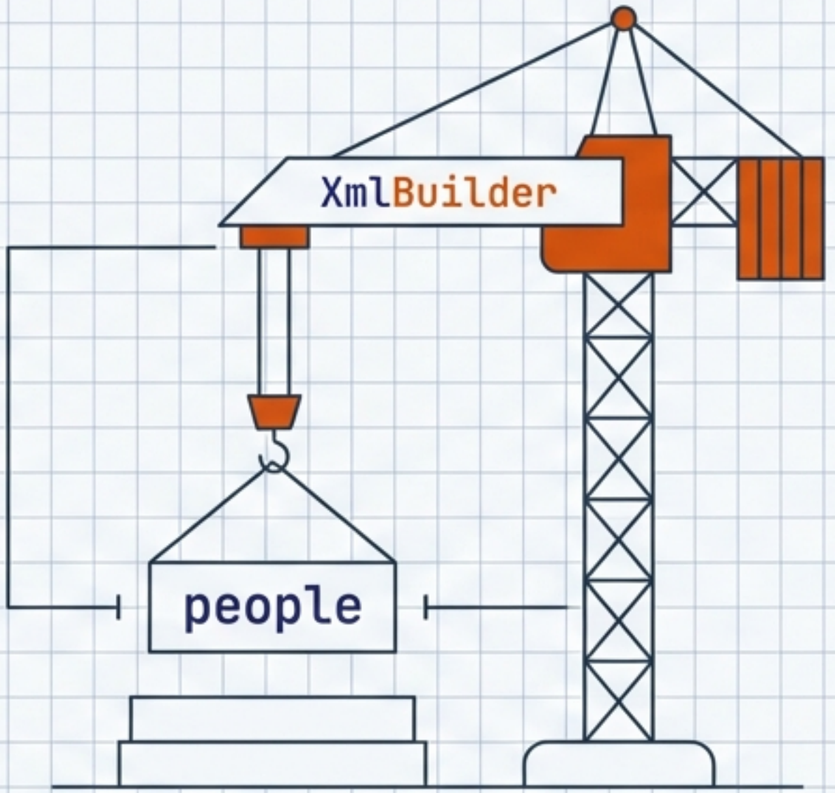


> Geniş arama için findAll, spesifik veri çekmek için find kombinasyonu en etkili yöntemdir.

Programatik XML Oluşturma

XML stringlerini elle birleştirmek karmaşıktır ve hataya açıktır. XmlBuilder sınıfı akıcı ve güvenli bir yapı sunar.

```
final builder = XmlBuilder();

// XML başlığını ekle
builder.processing('xml', 'version="1.0\"');

// Kök elementi ve içeriği başlat
builder.element('people', nest: () {
  // İçerik buraya
});
```

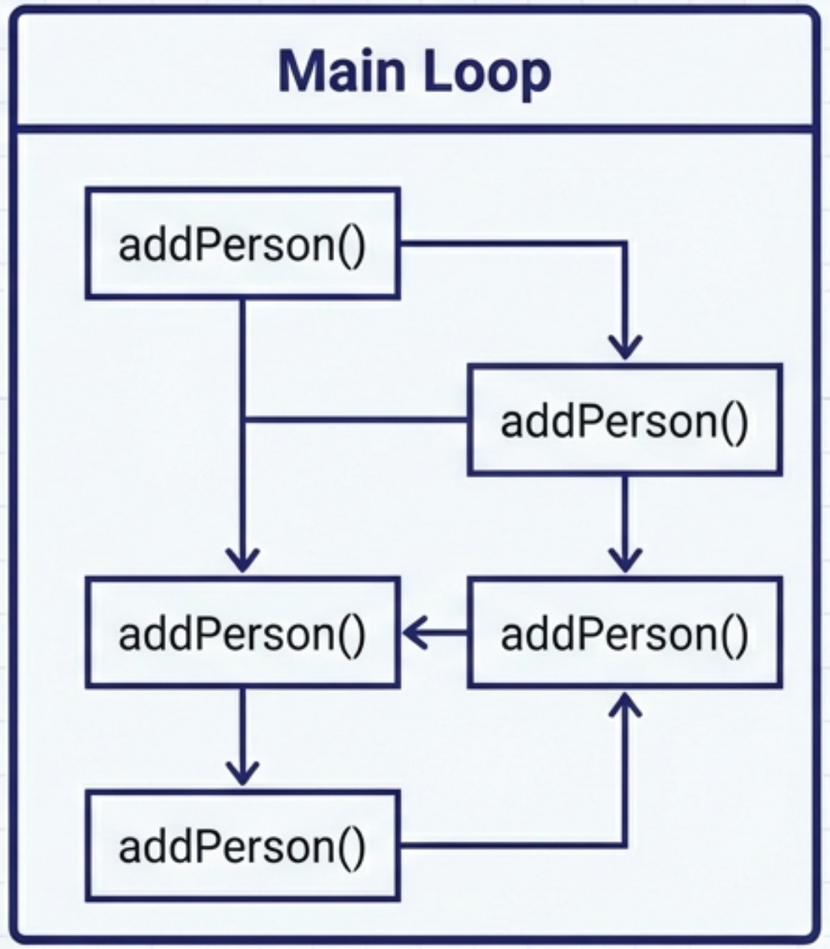


nest parametresi, bir ilkel değer (string) veya yeni bir yapı fonksiyonu alabilir.

Temiz Kod İçin Yuvalama (Nesting) Yönetimi

Sorun: Çok fazla iç içe fonksiyon okunabilirliği bozar.
Çözüm: Tekrarlayan yapıları yardımcı fonksiyonlara bölün.

```
void addPerson(XmlBuilder builder,
       {required String name, ...}) {

  builder.element('person', nest: () {
    builder.element('name', nest: name);
    builder.element('surname', nest: surname);
    // ...
  });
}
```



Dinamik Öznitelik (Attribute) Ekleme

Veriyi zenginleştirmek için elementlere mantıksal özellikler ekleyin.

```
builder.element('person', nest: () {
  // Yaş 18'den küçükse 'young' özelliği true
  olsun
  builder.attribute('young', age <= 18);

  builder.element('name', nest: name);
  // ...
  });
}
```

Final İnşa ve Dışa Aktarım

```dart
final peopleXML = builder.build();

// Okunabilir formatta çıktı al
print(peopleXML.toXmlString(pretty: true));
```

toString() ham (boşluksuz) çıktı verirken, toXmlString insan okuması için idealdir.

```xml
<people>
  <person young='false'>
    <name>Alberto</name>
    <surname>Miola</surname>
    <age>23</age>
  </person>
  ...
</people>
```

Geliştiricinin Araç Seti: Özet

Ayrıştır (Parse)

- **XmlDocument.parse()** kullanın.
- Web verisinde **Latin1Decoder** ihtiyacını unutmayın.

Gezin (Traverse)

- Geniş aramalar için **findAllElements**.
- Doğrudan çocuklar için **findElements**.
- Tekil veriler için **single**, **first**, **text**.

İnşa Et (Build)

- **XmlBuilder** ve **nest** yapısını kullanın.
- Tekrar eden blokları yardımcı fonksiyonlara bölün.
- Okunabilirlik için **pretty: true**.