Tema 3. Server de servire modele cu rutare adaptivă și „circuit breaker”

Scop. Să rulați două versiuni de model (A/B) și să ruturați în funcție de latență/erori, cu izolare pe pool-uri.

- **Concurrency.** Bulkheads (pool separat per model), cozi cu prioritate, timeouts per request.
- **Pattern-uri.** Strategy (rutare: p95 latență), State (circuit: closed/half-open/open), Builder (config endpointuri).
- **Metaprogramare.** @Serve(model="A", path="/predictA") → codegen endpoints (router).
- **Reflecție.** Validare semnături handler (return type, anularea pe timeout).
- **AI.** Handlerele „simulează” modele cu distribuții de latență diferite; colectați p50/p95/p99.

Concept practic — „server de servire modele cu rutare adaptivă și circuit breaker”

1. „Server de servire modele” — ce înseamnă

„Servire” (model serving) înseamnă **faza de inferență în producție**:
un server primește cereri (de exemplu, imagini sau texte) și returnează predicții.

Exemplu concret:

Client → /predict → Server AI → Model (ML/DL) → Răspuns JSON

În practică:

- Serverul primește o cerere HTTP (/predict),
- alege un model AI potrivit,
- rulează predicția și
- trimite rezultatul înapoi.

2. „Rutare adaptivă” — alegerea dinamică a modelului

Uneori avem **mai multe versiuni de modele**:

- model A → rapid, dar mai puțin precis,
- model B → lent, dar mai exact.

Rutare adaptivă înseamnă că serverul **alege dinamic** care model să folosească, în funcție de:

- latență (timp de răspuns),
- rata de erori,
- încărcarea CPU,
- contextul cererii.

Exemplu:

```
if (modelA.latency() < 100ms && modelA.errors() < 2%)
   routeTo(modelA);
else
   routeTo(modelB);
```

Se poate aplica pattern-ul **Strategy**:

- fiecare strategie de rutare e o implementare diferită (latență, scor, round-robin),
- serverul alege strategia potrivită automat.

3. „Circuit breaker" — protecție împotriva supraîncărcării

Un **circuit breaker** (în sens software) este un **mecanism de protecție** împotriva apelurilor repetate către un serviciu care e deja în eroare.

E un *pattern* inspirat din sistemele electrice:

- **Closed** – totul funcționează normal; cererile trec.
- **Open** – sistemul e în eroare; cererile sunt blocate temporar.
- **Half-Open** – se testează dacă sistemul s-a refăcut.

Exemplu:

```
if (failures > threshold) openCircuit();
if (circuit == OPEN && timeSinceOpen > retryInterval) halfOpen();
```

Dacă modelul B dă erori sau depășește timpii, circuitul se deschide:

- cererile nu se mai trimit la modelul B,

- sistemul rotează automat cererile spre modelul A (fallback).

4. De ce e util în AI

În sistemele AI reale:

- un model poate fi *down*,
- GPU-ul poate fi saturat,
- latențele pot varia.

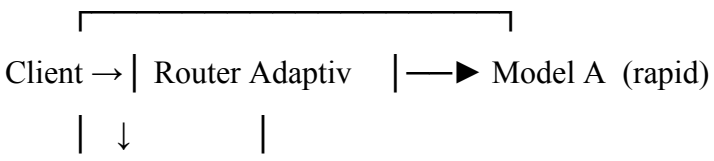
Un server inteligent trebuie să:

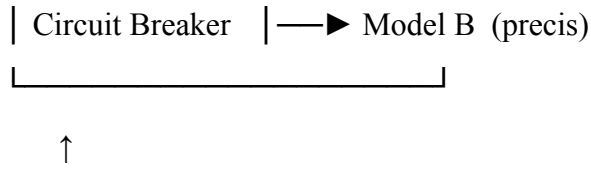
1. monitorizeze performanța fiecărui model (metrice),
2. ruteze adaptiv cererile,
3. evite apelurile spre modele „stricate” (circuit breaker),
4. reîncerce periodic reconectarea (half-open).

5. Legătura cu multithreading, pattern-uri și reflecție

Concept	Cum se folosește
Multithreading	Modelele rulează în *pool-uri* separate („bulkheads”), fiecare având propriul ExecutorService. Astfel, un model lent nu blochează restul.
Pattern-uri	- Strategy → rutare adaptivă - State → circuit breaker (open/closed/half-open) - Builder → configurarea serverului și modelelor
Reflecție	Încarci dinamic clasele de modele definite prin adnotări: @Serve(model="A", path="/predictA"). Poți adăuga noi modele fără recompilare.
Metaprogramare	Annotation Processor generează automat codul de routing (Router.java) în funcție de adnotările găsite.

6. Cum arată schematic





7. Exemplu scurt (schematic în Java)

```
@Serve(model="A", path="/predictA")
class ModelA implements Predictor {
   public Result predict(Data d){ simulate(50); return new Result("A"); }
}
@Serve(model="B", path="/predictB")
class ModelB implements Predictor {
   public Result predict(Data d){ simulate(200); return new Result("B"); }
}
public class AdaptiveServer {
   private final Map<String, Predictor> models = loadAnnotatedModels();
   private final Map<String, CircuitBreaker> circuits = initCircuits(models);
   public Result serve(Data d) {
      Predictor chosen = Router.choose(models, metrics());
      if (circuits.get(chosen.name()).isOpen()) return fallback();
      return chosen.predict(d);
   }
}
```

8. Ce obții

- **Scalabilitate** – modele rulează în paralel; un model lent nu blochează altele.
- **Rezistență la erori** – sistemul ocolește componente defecte.
- **Extensibilitate** – poți adăuga modele noi prin adnotări.
- **Automatizare** – rutare bazată pe metrice și stări, fără decizii hardcodate.

Pe scurt:

Termen	Înseamnă
Server de servire modele	Primește cereri și trimite date spre modele AI
Rutare adaptivă	Alege modelul potrivit în funcție de performanță
Circuit breaker	Protejează sistemul de apeluri repetate spre modele defecte
Multithreading	Rulează modele și cereri în paralel
Reflecție + adnotări	Încarcă automat modelele disponibile