Programovanie v jazyku C#

Chyby a výnimky

prednáška 8

Ing. Ján Magyar, PhD.

ak. rok. 2021/2022 ZS

Kvalita kódu

- spoľahlivosť
- robustnosť
- použiteľnosť
- portabilita
- udržateľnosť
- výkon
- čitateľnosť

Motivácia

- žiaden softvér nemá 100% spoľahlivosť
- programátor musí očakávať chyby počas behu programu
- chyby môžu byť spôsobené
  - nesprávne napísaným kódom
  - nesprávnymi predpokladmi
  - nesprávnym vstupom od používateľa

Výhody spracovania chýb

- spoľahlivejší a robustnejší program
- špecifikácia spôsobu spracovania chyby
- oddelenie kódu pre identifikáciu chyby od kódu pre jej spracovanie

Spracovanie chýb

- kód rozdelíme na tri časti
- try
  - kód definujúci správny chod programu
- catch
  - kód definujúci spôsob spracovania chyby
  - môže obsahovať aj logovanie
- finally
  - kód pre “vyčistenie” zdrojov
  - vykoná sa vždy
  - nie je povinný

Spracovanie chýb - chod

1. spustí sa vykonávanie kódu v bloku `try`
2. ak sa chyba nevyskytne, vykoná sa celý `try` blok
   v prípade chyby sa vykonávanie presunie do `catch` bloku
3. chyba sa spracuje v `catch` bloku
4. na konci `catch` bloku sa vykonávanie presunie do `finally`
5. vykoná sa kód v bloku `finally` (ak bol definovaný)

Syntax v C#

```
try {
   // normal execution
}
catch
{
   // error handling
}
finally
{
   // clean up
}
```

Možnosti spracovania

- `finally` blok nie je povinný
- môžeme definovať viacero `catch` blokov
- pre `catch` bloky môžeme definovať filtre pre odchytenie konkrétnych výnimiek
- `catch` blok nie je povinný - zabezpečíme, že kód vo `finally` sa vykoná

Generovanie a odchytenie výnimiek

- chod programu sa posunie do bloku `catch` ak sa vygeneruje inštancia výnimky
- inštancia sa vytvorí na základe príkazu `throw`
  ```
  throw new StackOverflowException();
  ```
- správny `catch` blok sa vyberie na základe toho, ktorý blok definuje presnejšie typ výnimky
- ak sa výnimka vygeneruje vo vnorenom volaní, chod sa vráti k pôvodnému `try-catch` bloku

Výnimky a výkon

- spracovanie výnimiek vyžaduje operácie navyše
- mali by sme ich využívať iba ak naozaj reprezentujú výnimočné situácie
- ak výnimočný stav môže nastať častejšie, je lepšie napísať defenzívne riešenie

Spracovanie výnimiek z cudzieho kódu

- štandardné funkcie môžu vyprodukovať chyby
- knižnice nedefinujú spôsob spracovania výnimiek - je to úloha programátora
- potreba je zvyčajne identifikovaná počas debuggingu
- cieľom je minimalizovať počet výnimiek a zadefinovať vhodný spôsob ich spracovania

Chyby v C#

- objekt vytvorený v momente keď sa stane nepredvídaná situácia
- podtrieda triedy `Exception`
  - príliš všeobecná na to, aby nám poskytovala použiteľné informácie
  - podtriedy nerozširujú funkcionalitu pri výnimkách
- skoro všetky v mennom priestore `System`
- dve hlavné kategórie
  - `SystemException`
  - `ApplicationException`

`SystemException`

- väčšinou vygenerované .NET runtimeom
- všeobecné javy, ktoré môžu nastať v akejkoľvek aplikácii
- môžu byť fatálne a nefatálne
- príklady
  - `StackOverflowException`
  - `ArgumentException`

`ApplicationException`

- pôvodne base trieda pre vlastné výnimky
- nadtrieda výnimiek vygenerovaných CLR
- už nie je vhodná nadtrieda pre vlastné výnimky (`Exception`)
- príklad
  - `TargetInvocationException`

Properties výnimiek

- `Data` - pre ďalšie informácie formou kľúč/hodnota
- `HelpLink` - odkaz na súbor s ďalšími informáciami
- `InnerException` - pre výnimky generované v `catch` bloku
- `Message` - popis výnimky
- *`Source`* - komponent ktorý výnimku vygeneroval
- *`StackTrace`* - zoznam volaní funkcií pred výnimkou
- `HResult` - číselná hodnota výnimky
- `TargetSite` - objekt popisujúci metódu, ktorú chybu vygenerovala

Nastavenie properties

- pred vyhodením výnimky

```
var myException = new ExceptionClass(“Help!”);
myException.Source = “My application”;
myException.HelpLink = “MyHelpFile.txt”;
myException.Data[“ErrorDate”] = DateTime.Now;
myException.Data.Add(“Contact”, “address”);
throw myException;
```

Exception filter

- `catch` blok sa vykoná iba ak sa filter vyhodnotí ako `true`
- rozdielne spracovanie výnimiek rovnakého typu ale s inými vlastnosťami
- pomocou kľúčového slova `when`

```
catch (ExceptionType ex) when (condition)
```

Propagácia výnimiek

- ak počas spracovania jednej výnimky účelne vytvoríme ďalšiu výnimku
- zvyčajne zmeníme typ výnimky alebo rozšírime jej popis a dáta
- môžeme prísť o stack trace - nevieme určiť dodatočne kde sa chyba stala
- napr. v aplikácii potrebujeme iný zápis chyby do logu a inú správu pre používateľa

Typy propagácie chýb

- opätovné použitie chyby - zlý prístup
- zmena chyby - neprídeme o informácie
- opätovné vygenerovanie chyby - máme celý stack trace
- pridane filtrov - filter použijeme pre logy
  - jediný prípad použitia filtra na iné účely

Nespracované výnimky

- v .NET frameworku máme definovaný defaultný `try-catch` blok
- ak chybu nespracujeme my, “spracuje” to samotný .NET
- vypíše sa informácia o chybe a vykonávanie sa končí
- ak píšeme kód, mali by sme odchytiť čo najviac možných chýb
  - nie je potrebné ak píšeme knižnicu

Definícia vlastných výnimiek

- pre reprezentáciu javov špecifických pre aplikáciu
- podtrieda triedy `Exception`
- zvyčajne sa definujú iba konštruktory

otázky?