Zápočtová úloha z předmětu KIV/ZSWI

OBJEKTOVÝ NÁVRH APLIKACE

pro JAVA aplikace pro monitoring běhu komponent

6. května 2015

Název týmu: **CrossCafé team**

Členové týmu:

Luboš Hubáček (vedoucí týmu)	lubos.hubacek@diginex.cz
Ondřej Pittl	ondrej.pittl@gmail.com
Jiří Homolka	jiri.homolka22@gmail.com
Jan Kohlíček	kohlicejan@gmail.com
Petr Kozler	pkozler@students.zcu.cz

Obsah dokumentu

1. Úvod

1.1 Účel systému

Tato aplikace bude používána ke zjišťování aktuálního stavu daných komponent, instancí PF, mezi nimiž bude uživateli poskytnuto přepínat pomocí grafického rozhraní. O stavu instancí bude uživatel informován přímo na obrazovce, tedy výpis bude umožněn grafickým rozhraním.

Výstup bude rovněž ukládán do rotujících souborů, z nichž bude možné sledovat i historii jejich stavu — v závislosti na konstantní velikosti souborů.

Aplikace bude poskytovat již zmíněné jednoduché grafické uživatelské rozhraní pro pohodlné ovládání programu. Bude sestávat z menu ovládající chod aplikace, stromu instancí PF v levé části a v pravé části komponentou, jež bude zobrazovat aktuální stav komponent.

1.2 Slovníček definic, pojmů a zkratek

PeerFile (PF) – server, k němuž se pomocí aplikace přistupuje a jehož komponenty monitoruje

REST API – architektura rozhraní, navržená pro distribuované prostředí; použito pro jednotný a snadný přístup ke zdrojům

JSON – způsob přenosu dat nezávislý na počítačové platformě. Vstupem je libovolná datová struktura, výstupem je vždy řetězec. Složitost hierarchie vstupní proměnné není teoreticky nijak omezena – JSON může mít různě složitou strukturu

Rolling File – soubor s konstantní velikostí dat, v němž dochází k přepisování starších dat v případě zápisu dat překračujících velikostní limit

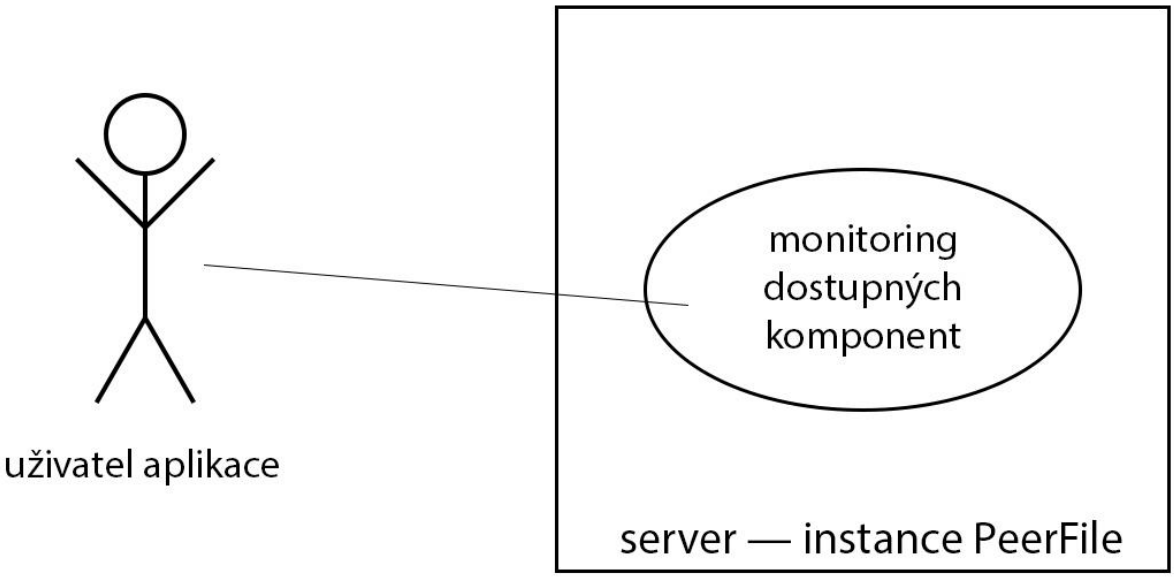
1.3 Odkazy

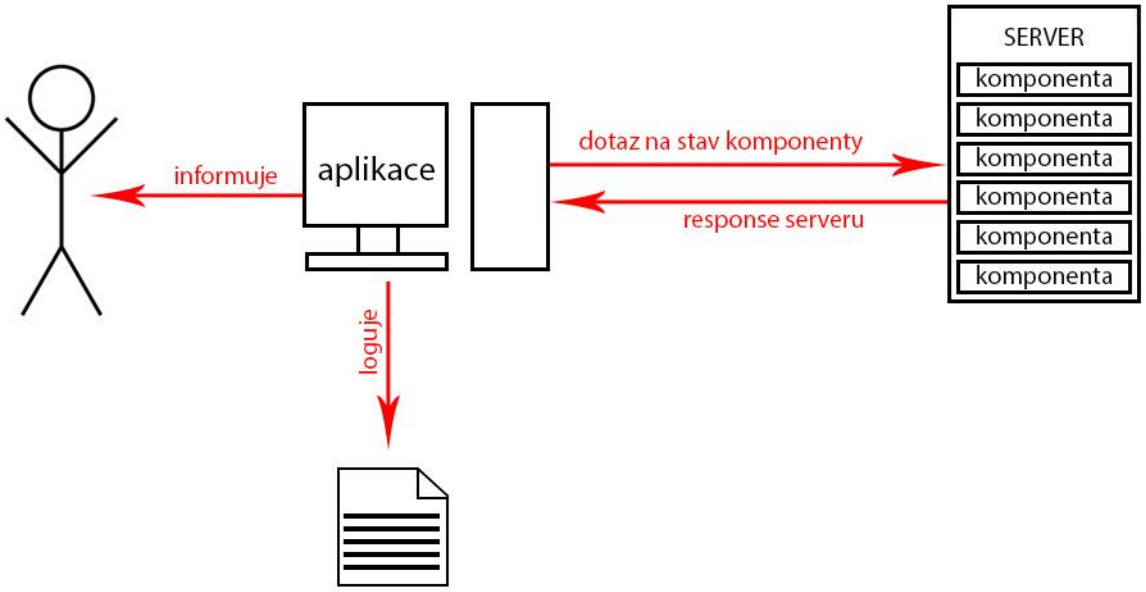
• **Unicorn**. Dokumentace Monitoring REST API. *Google dokumenty.* [online]. 2014 [cit. 2015-03-09]. Dostupné z: *http://bit.ly/1DeFsnZ*

• **CrossCafe team**. ZSWI monitoring. *GitHub.* [online]. 2015 [cit. 2015-03-09]. Dostupné z: *http://github.com/Sekiphp/ZSWI_monitoring*

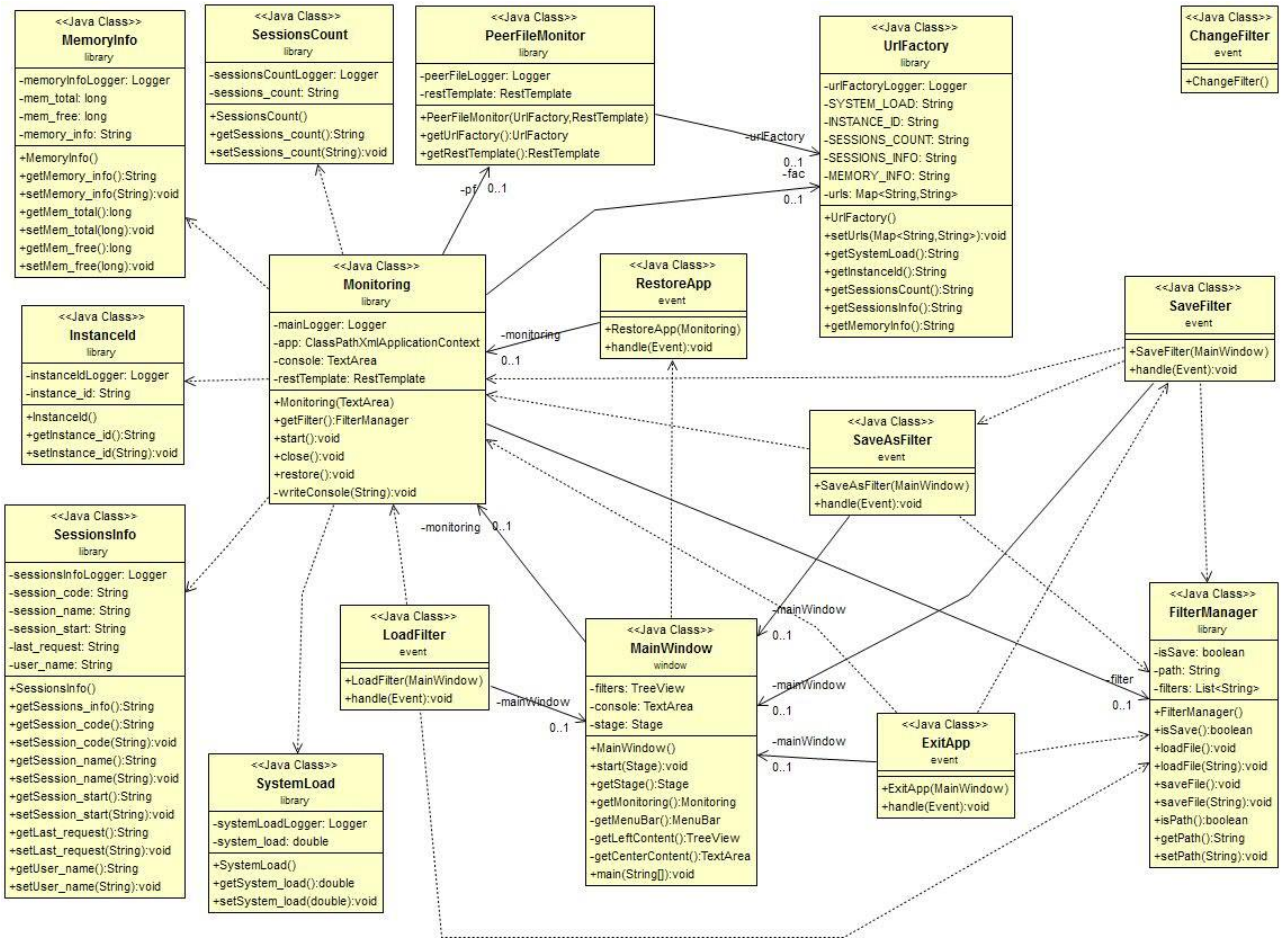
2. Kontext a architektura systému

2.1 Kontext systému





2.2 Architektura systému, přehled podsystémů



2.3 Zvolená technologie, programovací jazyk ad., důvody

Celý systém bude implementován ve vysokoúrovňovém programovacím jazyce *Java*. Důvodem je nejen požadavek zadavatele, ale i vhodnost použití právě tohoto jazyka. Pro implementaci nosné části aplikace je použit framework *Spring WS*, s naimplementovanými metodami pro snadnou práci s webovými službami — tato práce spočívá především v získávání dat z webové služby přes služby *REST* a přehledného logování pomocí knihovny *Log4J*, která umožňuje kvalitní a pohodlnou práci s logováním do rotujících souborů. Pro realizaci grafického uživatelského rozhraní jsme se po dohodě se zadavatelem rozhodli použít *JavaFX*, jelikož byl ukončen vývoj Java Swing, je zastaralý a nebude do budoucna podporován.

3. Typy informací zpracovávané systémem

V našem případě nejde ani tak o vstupní soubory jako o response instancí PF, které jsou dostupné na vzdáleném serveru, a přistupuje se k nim pomocí URL. Předběžný seznam aktuálně známých URL je obsažen v DSP. Odkazy na nové instance se přidávají prostřednictvím souboru application-context.xml, kde se do Bean UrlFactory připisují URL na instance služby PF. Po vytvoření nového záznamu (bean) reprezentujícího novou instanci PF je třeba vytvořit ještě příslušnou třídu, jež bude pomocí application contextu instancována prostřednictvím beans. Pro korektní chod aplikace je třeba změnu zanést i do třídy UrlFactory.

Co se týče výstupních souborů, bude jich hned několik. Pro každou instanci bude k dispozici několik archivních tzv. rolling file souborů o konstantní velikosti. Pro snadnější implementaci rotujících souborů využijeme knihovnu Log4J 2.

4. Návrh systému

4.1 Přehled tříd

4.1.1 library.Monitoring

Hlavní třída aplikace, spouští sekvenci příkazů, řídí aplikaci.

4.1.1.1 Konstruktory

- `public Monitoring(TextArea console)`

4.1.1.2 Metody

- `public FilterManager getFilter()`
  Getr aktuálního filtru logovaneho vystupu.

- `public void start()`
  Spouští monitorování.

- `public void close()`
  Ukončuje aplikaci.

4.1.2 window.MainWindow

Třída poskytující grafické uživatelské rozhraní, které je zde sestavováno z jednotlivých komponent.

4.1.2.1 Metody

- `public void start(Stage stage)`
  Inicializační metoda hlavního okna grafického uživatelského rozhraní.

- `public Stage getStage()`
  Getr top-level kontejneru — `javafx.stage.Stage`.

- `public Monitoring getMonitoring()`
  Getr monitorujícího objektu.

4.1.3 event.ExitApp

EventHandler, zajištuje obsluhuje událost ukončení aplikace.

4.1.3.1 Metody

- `public void handle(Event event)`
  Metoda odchytávající událost, na níž je třeba zareagovat a ukončit aplikaci.

4.1.4 event.ChangeFilter

EventHandler zajišťující změnu aktivní sledované instance.

4.1.4.1 Metody

- `public void handle(Event event)`
  Metoda odchytávající událost změny aktivní instance ve stromě dostupných instancí.

4.1.5 event.LoadFilter

EventHandler zajišťující načtení filtru, který omezuje výstup na obrazovku — jinými slovy filtruje všechny logovací hlášky na ty, které nás zajímají.

4.1.5.1 Konstruktory

- `public LoadFilter(MainWindow mainWindow)`

4.1.5.2 Metody

- `public void handle(Event event)`
  Metoda odchytávající událost, obsluhuje načtení filtru.

4.1.6 event.RestoreApp

EventHandler obsluhující událost, zprostředkovává začátek monitorování.

4.1.6.1 Konstruktory

- `public RestoreApp(Monitoring monitorig)`

4.1.6.2 Metody

- `public void handle(Event event)`
  Metoda odchytávající událost, spouští monitorování.

4.1.7 event.SaveAsFilter

EventHandler obsluhující událost uložení nastavení filtru do nového souboru.

4.1.7.1 Konstruktory

- `public SaveAsFilter(MainWindow mainWindow)`

4.1.7.2 Metody

- `public void handle(Event event)`
  Metoda odchytávající událost, zajišťuje uložení nastavení filtru do nového souboru.

4.1.8 event.saveFilter

EventHandler obsluhující událost, přepisuje již existující filtr.

4.1.8.1 Konstruktory

- `public LoadFilter(MainWindow mainWindow)`

4.1.8.2 Metody

- `public void handle(Event event)`
  Metoda odchytávající událost, ukládá filtr do již existujícího souboru.

4.1.9 library.FilterManager

Souborový manažer, zajišťuje práci s filtry — načítání, ukládání.

4.1.9.1 Konstruktory

- `public FilterManager()`

4.1.9.2 Metody

- `public void loadFile()`
  Načítá filter z defaultního adresáře.

- `public void loadFile(String filePath)`
  Načítá filter z explicitně definovaného adresáře, jehož cesta je určena argumentem metody.

- `public void saveFile()`
  Ukládá filter do defaultního adresáře.

- `public void saveFile(String filePath)`
  Ukládá filter do explicitně definovaného adresáře, jehož cesta je určena argumentem metody.

- `public String getPath()`
  Getr uchovávané cesty adresáře.

- `public String setPath(String path)`
  Setr uchovávané cesty adresáře.

4.1.10 library.InstanceId

Třída reprezentující instanci PF, tato třída je instancována pomocí beans definovaných v `application-context.xml`. Uchovává se v ní response ze strany serveru.

4.1.10.1 Metody

- `public String getInstance_id()`
  Getr ID instance, získání response.

- `public void setInstance_id(String instance_id)`
  Setr ID instance, využívá Spring WS.

4.1.11 library.MemoryInfo

Třída reprezentující instanci PF, tato třída je instancována pomocí beans definovaných v `application-context.xml`. Uchovává se v ní response ze strany serveru.

4.1.11.1 Metody

- `public String getMemory_info()`
  Getr response.

- `public void setMemory_info(String memory_info)`
  Setr response, využívá Spring WS.

- `public long getMem_total()`
  Getr response.

- `public void setMem_total(long mem_total)`
  Setr response, využívá Spring WS.

- `public long getMem_free()`
  Getr response.

- `public void setMem_free(long mem_free)`
  Setr response, využívá Spring WS.

4.1.12 library.SessionsCount

Třída reprezentující instanci PF, tato třída je instancována pomocí beans definovaných v `application-context.xml`. Uchovává se v ní response ze strany serveru.

4.1.12.1 Metody

- `public String getSessions_count()`
  Getr response.

- `public void setInstance_id(String instance_id)`
  Setr response, využívá Spring WS.

4.1.13 library.SessionInfo

Třída reprezentující instanci PF, tato třída je instancována pomocí beans definovaných v `application-context.xml`. Uchovává se v ní response ze strany serveru.

4.1.13.1 Metody

- `public public String getSessions_info()`
  Getr response.
- `public String getSession_code()`
  Getr response.
- `public void setSession_code(String session_code)`
  Setr response, využívá Spring WS.
- `public String getSession_name()`
  Getr response.
- `public void setSession_name(String session_name)`
  Setr response, využívá Spring WS.
- `String getSession_start()`
  Getr response.
- `public void setSession_start(String session_start)`
  Setr response, využívá Spring WS.
- `public String getLast_request()`
  Getr response.
- `public void setLast_request(String last_request)`
  Setr response, využívá Spring WS.
- `public String getUser_name()`
  Getr response.
- `public void setUser_name(String user_name)`
  Setr response, využívá Spring WS.

4.1.14 library.SystemLoad

Třída reprezentující instanci PF, tato třída je instancována pomocí beans definovaných v `application-context.xml`. Uchovává se v ní response ze strany serveru.

4.1.14.1 Metody

- `public double getSystem_load()`
  Getr response.
- `public void setSystem_load(double systemLoad)`
  Setr response, využívá Spring WS.

4.1.15 library.UrlFactory

Třída uchovávající množinu url instancí PF. Tato třída je instancována pomocí beans definovaných v `application-context.xml`.

4.1.15.1 Metody

- `public void setUrls(Map<String, String> urls)`
  Setr url instancí.

- `public String getSystemLoad()`
  Getr url instance SystemLoad.

- `public String getInstanceId()`
  Getr url instance InstanceId.

- `public String getSessionsCount()`
  Getr url instance SessionsCount.

- `public String getSessionsInfo()`
  Getr url instance SessionsInfo.

- `public String getMemoryInfo()`
  Getr url instance MemoryInfo.

4.1.16 library.PeerFileMonitor

Třída uchovávající odkaz na UrlFactory a REST službu, RestTemplate, která zprostředkovává komunikaci se serverem a získání response ze strany serveru.

4.1.16.1 Konstruktory

- `public PeerFileMonitor(UrlFactory urlFactory,`
  `RestTemplate restTemplate)`

4.1.16.1 Metody

- `public UrlFactory getUrlFactory()`
  Getr UrlFactory.

- `public RestTemplate getRestTemplate()`
  Getr RestTemplate (REST služba).

5. Přiřazení tříd/modulů programátorům

V této kapitole jsou ke každé existující třídě přiřazeni zodpovědní lidé, kteří se starají o implementaci daných tříd. Není ovšem vyloučeno, že do dané třídy bude zasahovat i jiný člen týmu, protože jednotlivé třídy jsou mezi sebou těsně provázány a potřebu úprav ostatních tříd, byť jen minimální, nelze zcela vyloučit.

library.Monitoring	—	celý tým dle potřeb	(implementace nových metod)
library.UrlFactory	—	celý tým	(správa URL instancí PF)
library.PeerFileMonitor	—	Ondřej Pittl	(správa UrlFactory & REST service)
library.SystemLoad	—	Ondřej Pittl	(reprezentace instance PF)
library.SessionsCount	—	Luboš Hubáček	(reprezentace instance PF)
library.InstanceId	—	Jan Kohlíček	(reprezentace instance PF)
library.SessionInfo	—	Jiří Homolka	(reprezentace instance PF)
library.MemoryInfo	—	Jiří Homolka	(reprezentace instance PF)
window.*	—	Jan Kohlíček	(GUI)
event.*	—	Jan Kohlíček	(GUI)